CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 21/10/2024 - 12:06:52.9
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
vivxhb2a V: 1.02
75565 ARIEL GUSTAVO RODRIGUES DA S
24JOG506 - 1 _____ / _____ / _____

UVA 108 - Maior subsoma em matriz

Observação: este texto está baseado em https://www.redgreencode.com/three-ways-solve-uva-108/ do autor Duncan Smith.

O problema original - 108

Como está lá, já traduzido:
Um problema que é simples de resolver numa dimensão é muitas vezes muito mais difícil de resolver em mais de uma dimensão. Considere satisfazer uma expressão booleana na forma normal conjuntiva na qual cada conjunto consiste em exatamente 3 disjuntos. Este problema (3-SAT) é NP-completo. O problema 2-SAT é resolvido de forma bastante eficiente, no entanto. Em contraste, alguns problemas pertencem à mesma classe de complexidade independentemente da dimensionalidade do problema. Dada uma matriz bidimensional de inteiros positivos e negativos, encontre o subretângulo com o maior soma. A soma de um retângulo é a soma de todos os elementos desse retângulo. Neste problema o subretângulo com a maior soma é chamado de subretângulo máximo. Um subretângulo é qualquer submatriz contígua de tamanho 1 x 1 ou maior localizado sobre a matriz. Como exemplo o sub-retângulo máximo do array

```
0   -2   -7   0
9    2   -6   2
-4   1   -4   1
-1   8    0  -2
```

está no canto esquerdo inferior

```
9   2
-4  1
-1  8
```

cuja soma é 15

Entrada:
A entrada consiste em uma matriz N × N de inteiros. A entrada começa com um único inteiro positivo N em uma linha isolada indicando o tamanho do quadrado da matriz bidimensional. Isto é seguido por números inteiros N^2 separados por espaços em branco (novas linhas e espaços). Esses inteiros N^2 constituem a matriz em ordem de linha maior (ou seja, todos os números na primeira linha, da esquerda para a direita, depois, todos os números da segunda linha, da esquerda para a direita, etc.). N pode ser tão grande quanto 100. Os números no array estarão no intervalo $[-127, 127]$.

Saída:
A saída é a soma do sub-retângulo máximo.

Exemplo de entrada

```
4
0 -2 -7 0 9 -2 -6 2
-4 1 -4 1 -1
8 0 -2
```

Exemplo de saída
15

Pesquisa completa

A abordagem de busca completa para este problema requer que enumeremos todos os subarrays do array de entrada e calculemos a soma de cada um separadamente. Para enumerar os subarrays de uma matriz, considere que você pode identificar cada submatriz por um conjunto de quatro inteiros 1 e N: linha inicial, coluna inicial, linha final e coluna final. Você pode pensar nesses quatro inteiros como as coordenadas dos cantos superior esquerdo e inferior direito do subarray. Tendo identificado um subarray usando essa abordagem de busca completa, a maneira ingênua de encontrar sua soma é adicionar todos os seus números juntos.

Para gerar todas as coordenadas do subarray, podemos usar quatro loops aninhados, um para cada coordenada. Outros dois loops aninhados são suficientes para iterar pelas linhas e colunas do subarray e calcular sua soma. Esses seis loops aninhados nos dão uma algoritmo. Embora o limite de tempo UVa, $O(N^6)$, seja bastante frouxo para esse problema, ele não é tão tolerante. Então, precisamos de uma abordagem mais eficiente. Veja uma possível implementação do algoritmo näive

```
maximo=-999999
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    para k de i a N
      para L de 1 a N
        soma=0
        para a de i a k
          para b de j a L
            soma=soma+MAT[a][b]
          fimpara
        fimpara
        se soma>maximo
          guarda=i,j,k,L
          maximo=soma
        fimse
      fimpara
    fimpara
  fimpara
fimpara
```

Estratégia de programação dinâmica

Com a reutilização de somas de subarrays. Para problemas básicos de programação dinâmica, como UVA 108, o segredo é descobrir quais informações você pode armazenar em etapas anteriores que economizarão tempo em etapas subsequentes.

A ideia-chave para UVa 108 é que os resultados de um cálculo anterior (uma soma, neste caso) podem ser usados em cálculos subsequentes. Especificamente, uma vez que calculamos a soma de parte de um subarray, não precisamos calculá-la repetidamente quando estamos processando outros subarrays que se sobrepõem à mesma região. **Se pudermos encontrar uma maneira eficiente de armazenar e reutilizar essas somas, podemos economizar muito tempo de cálculo**. A estrutura de dados usada em programação dinâmica para esse propósito é tradicionalmente chamada de tabela de memorando .

A tabela de memorandos tem o mesmo tamanho, formato e tipo que a matriz de entrada: a matriz de inteiros. O valor inteiro na posição (i, j) na tabela de memorandos armazena a soma da submatriz cujo canto superior esquerdo é o canto superior esquerdo da matriz de entrada e cujo canto inferior direito é (i, j) na matriz de entrada. Por exemplo, posição da tabela de memorandos $(2, 2)$ armazenaria a soma do subarray quadrado de $(1, 1)$ até $(2, 2)$ em nosso esquema de numeração baseado em 1.

Etapa 1: Gerar a tabela de memorandos
A chave para gerar a tabela de memorandos de forma eficiente é iterar pela matriz de entrada linha por linha e usar resultados anteriores para calcular os subsequentes. Você está basicamente usando a tabela de memorandos para gerar a si mesma. Para você acompanhar, eis como ficaria a tabela de memorandos do exemplo acima citado

```
0   -2    5    5
9    9   10   12
5    6    3    6
4   13   10   11
```

Para ver como isso funciona, considere a $4, 4$ matriz tabela de memorando mostrada abaixo. Os números de 01 para 16 representam a ordem na qual iremos iterar pelos dezesseis valores de entrada e calcular os dezesseis valores da tabela de memorando. Preencher a tabela dessa forma simplificará o código. Podemos ler a entrada e preencher a tabela de memorando simultaneamente. Portanto, não há necessidade de armazenar uma matriz separada de valores de entrada. Todas as informações de que precisamos na próxima etapa estarão na tabela de memorando.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Veja como o processo funciona: Escolha uma célula na matriz. Como exemplo, usarei a célula 10. Isso representa o canto inferior direito de um sub-array de 6 células que começa na célula 1 (três linhas e duas colunas). De acordo com nossa definição de tabela de memorando, a célula 10 deve conter a soma dos valores de entrada 01, 02, 05, 06, 09, e 10. Mas lembre-se de que estamos preenchendo a tabela de memorando em ordem. Então, quando chegamos à célula 10, já temos valores corretos para células 01 até 09. Como podemos usar esses valores?

A resposta é que podemos usar três informações para calcular cada soma de submatriz:

- A célula única que forma seu canto inferior direito. Neste caso, célula 10.
- O subarray acima dele. Neste caso, é o subarray $2, 2$ começando em 01 e terminando em 06. A soma deste subarray é armazenada na célula 06.
- O subarray à esquerda dele. Neste caso, é o subarray $3, 1$ começando em 01 e terminando em 09. A soma deste subarray é armazenada na célula 09.

Então, em vez de adicionar dez células para obter a soma que estamos procurando, podemos simplesmente adicionar três células: 06, 09, e 10. E para submatrizes maiores, a economia de tempo é maior.

Se você estiver acompanhando, notará que esse processo não está muito certo. O subarray acima e o subarray esquerdo 01 se sobrepõem nas células 01 e 05. Então, temos que subtrair a soma dessas células para evitar contagem dupla. Felizmente, a quantidade que precisamos subtrair é armazenada em uma única célula, a 05. Essa quantidade contada duas vezes sempre será uma célula diagonalmente para cima e para a esquerda da nossa célula-alvo.

Agora estamos no final do Passo 1, então nossa tabela de memorando contém as somas de todos os subarrays que começam no canto superior esquerdo do array de entrada. Mas o subarray de soma máxima pode começar em uma posição diferente. Para lidar com o caso geral, precisamos do Passo 2.

Etapa 2: Avalie todas as posições iniciais e finais
Esta etapa compartilha algumas ideias com a abordagem da Busca Completa descrita acima. A diferença é que as somas do subarray já foram calculadas eficientemente na Etapa 1. Então, em vez de um algoritmo de (Busca Completa) $O(N^6)$, temos um $O(N^2)$ passo (Passo 1) seguido por um passo $O(N^4)$ (Etapa 2). Como você deve saber pelas regras da análise de algoritmos assintóticos, isso nos dá uma visão geral de tempo de execução $O(N^4)$, o que é suficiente para que esta solução seja aceita na UVa OJ.

Assim como na Busca Completa, a Etapa 2 usa quatro loops aninhados, um para cada linha inicial, coluna inicial, linha final e coluna final. Essas quatro coordenadas definem o subarray em questão. Temos que verificar cada subarray para ver se ele tem a soma máxima vista até agora.

Veja o que acontece dentro do loop mais interno:

Primeiro, recuperamos a soma memorizada que calculamos na Etapa 1 para o subarray que termina na linha e coluna atuais. Lembre-se de que isso representa a soma do subarray que começa no canto superior esquerdo da entrada. Como isso pode não ser o mesmo que o canto superior esquerdo do subarray que estamos verificando no momento, temos que ajustar a soma usando um processo semelhante ao que usamos na Etapa 1. Para ilustrar, suponha que temos a seguinte tabela de memorando concluída. Como antes, os números identificam a ordem em que as células foram calculadas.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Como exemplo, suponha que nossos loops aninhados atualmente identifiquem uma posição inicial da célula 11 (linha 3, coluna 3) e uma posição final da célula 16 (linha 4, coluna 4). Precisamos encontrar a soma do subarray $2, 2$. O valor memorizado na célula 16 nos dá a soma de todo o array, então temos que nos livrar das partes que não precisamos. Como na Etapa 1, podemos fazer isso usando os valores que estão acima e à esquerda do alvo.

Aqui está outra maneira de analisar a tabela de memorandos:

```
ZZ ZZ XX XX
ZZ ZZ XX XX
```

```
YY YY 11 12
YY YY 15 16
```

As células XX identificam o subarray que está acima do alvo. As células YY identificam o subarray que está à esquerda do alvo. E as células ZZ identificam o subarray que está acima e à esquerda (ou seja, diagonalmente para cima e à esquerda) do alvo.

Esta visualização deve ser familiar do Passo 1. Mas neste passo, precisamos subtrair em vez de adicionar, já que estamos removendo somas que não precisamos. Então, começando com a soma na célula de memorando 16, precisamos:

Subtraia o valor na célula memo 08. Isso remove a soma do subarray que está acima do alvo (os XXs e ZZs). Subtraia o valor na célula memo 14. Isso remove a soma do subarray que está à esquerda do alvo (os YYs e ZZs). Observe que subtraímos o ZZs duas vezes. Nós os contamos duas vezes, assim como contamos duas vezes o valor diagonal no Passo 1. Portanto, precisamos que adicione novamente o valor na célula de memorando 06, os ZZs. Depois que adicionamos de volta uma das cópias contadas duas vezes do ZZs, temos uma soma precisa para nosso subarray de destino. Então, apenas comparamos o resultado com nossa soma máxima atual e atualizamos o máximo, se necessário.

Quando tivermos passado por todas as coordenadas iniciais e finais possíveis, nosso valor máximo será a resposta, e simplesmente imprimiremos esse inteiro.

Importante Se o seu ambiente de programação começa as numerações em 0 (Python, JS, Java, C/C++), para implementar os algoritmos, é conveniente deixar uma linha zerada (a linha 0) e igualmente uma coluna zerada (a coluna 0). Agora as verificações passam a ser em referência à coluna e linha zeros e não a 1 como nos algoritmos. Aliás, no site acima referido o autor faz isso.

O Algoritmo final

Veja como ficou uma possível solução

```
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    se i>1
      MAT[i][j]=MAT[i][j]+MAT[i-1][j]
    fimse
    se j>1
      MAT[i][j]=MAT[i][j]+MAT[i][j-1]
    fimse
    se (i>1) AND (j>1)
      MAT[i][j]=MAT[i][j]-MAT[i-1][j-1]
    fimse
  fimpara
fimpara
ans = 0
para i de 1 a N
  para j de 1 a N
    para k de i a N // note o i
      para L de j a N // idem j
        cur = MAT[k;L]
        se i > 1
          cur=cur-MAT[i-1][L]
        fimse
        se j>1
          cur=cur-MAT[i][j-1]
        fimse
        se (i>1) AND (j>1)
          cur=cur+MAT[i-1][j-1]
        fimse
        se and<cur
          ans=cur
          salva i,j,k,L
        fimse
      fimpara
    fimpara
  fimpara
fimpara
imprima (ans,salva)
```

Se você comparar os 2 algoritmos acima, verá que são muito parecidos. Só que o primeiro (força bruta) tem 6 *paras* um dentro do outro o que caracteriza o desempenho $O(N^6)$. Já o segundo tem os mesmos 6 *paras*, mas primeiro há um dentro do outro ($O(N^2)$) e depois, há 4 *paras*. Da teoria, ao final os 2 *paras* superiores podem ser desconsiderados e o resultado final fica só com $O(N^4)$.

Um exemplo completo

Para você testar seu programa. A matriz

```
-41 -98  91  71 -42   8   3 -79 -17  16
 76 -11  46  74  44  61  42  49 -96  78
  5  -7 -86  43  -2  34  37 -60  84  74
 79   9 -72  -9  98 -56 -10 -36   3  77
-12  -6  62 -26 -57 100 -69  27  24 -99
-99  55  46 -36 -16  37  37 -58  68  42
 66 -81 -83  53  26 -57 -57 -83 -22  91
 90 -22 -46  39 -43  56  57 -15 -43 -61
-97 -61  97 -51  64 -72 -20  21 -64  66
-68  98 -48 -53 -79 -56  27  40  59  40
```

Deu como resposta 588 2, 1, 6, 10.

☞ Para você fazer

Calcule a máxima subsoma da matriz 10×10 abaixo e determine também as dimensões e posição da submatriz. Para tanto identifique o canto superior esquerdo e inferior direito da submatriz (origem=1). Os números estão entre $-600 < n \leq 600$.

```
-498 -326  128 -405 -396   40 -168 -348 -240  211
  32  240 -294 -176  330  191 -299  105 -261  273
-304 -126 -236  396 -326 -159  286  102   37  182
 292 -394 -269 -526 -558 -392 -462 -449  266  272
-231 -190  397 -440  -14 -425  367   44  236 -551
  80 -350 -287    1 -330  364 -513  -36  275 -143
 347 -256 -436  325 -584   -7  -18  149 -357 -389
-349  290  -81  387 -425 -182  162   82 -577   86
 -52  311   93    1 -111 -381 -412  -96 -359  118
-524 -152 -568  192   51 -335 -320  189 -218  247
```

Responda aqui:

soma	localização

506-75565 -

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 21/10/2024 - 12:06:52.9
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
vivxhb2a V: 1.02
75572 CASSIANO MAGNO CHAGAS E SA
24JOG506 - 2 _____ / _____ / _____

UVA 108 - Maior subsoma em matriz

Observação: este texto está baseado em https://www.redgreencode.com/three-ways-solve-uva-108/ do autor Duncan Smith.

O problema original - 108

Como está lá, já traduzido:
Um problema que é simples de resolver numa dimensão é muitas vezes muito mais difícil de resolver em mais de uma dimensão. Considere satisfazer uma expressão booleana na forma normal conjuntiva na qual cada conjunto consiste em exatamente 3 disjuntos. Este problema (3-SAT) é NP-completo. O problema 2-SAT é resolvido de forma bastante eficiente, no entanto. Em contraste, alguns problemas pertencem à mesma classe de complexidade independentemente da dimensionalidade do problema. Dada uma matriz bidimensional de inteiros positivos e negativos, encontre o subretângulo com o maior soma. A soma de um retângulo é a soma de todos os elementos desse retângulo. Neste problema o subretângulo com a maior soma é chamado de subretângulo máximo. Um subretângulo é qualquer submatriz contígua de tamanho 1 x 1 ou maior localizado sobre a matriz. Como exemplo o sub-retângulo máximo do array

```
0   -2   -7   0
9    2   -6   2
-4   1   -4   1
-1   8    0  -2
```

está no canto esquerdo inferior

```
9   2
-4  1
-1  8
```

cuja soma é 15

Entrada:
A entrada consiste em uma matriz N × N de inteiros. A entrada começa com um único inteiro positivo N em uma linha isolada indicando o tamanho do quadrado da matriz bidimensional. Isto é seguido por números inteiros N^2 separados por espaços em branco (novas linhas e espaços). Esses inteiros N^2 constituem a matriz em ordem de linha maior (ou seja, todos os números na primeira linha, da esquerda para a direita, depois, todos os números da segunda linha, da esquerda para a direita, etc.). N pode ser tão grande quanto 100. Os números no array estarão no intervalo $[-127, 127]$.

Saída:
A saída é a soma do sub-retângulo máximo.

Exemplo de entrada

```
4
0 -2 -7 0 9 -2 -6 2
-4 1 -4 1 -1
8 0 -2
```

Exemplo de saída
15

Pesquisa completa

A abordagem de busca completa para este problema requer que enumeremos todos os subarrays do array de entrada e calculemos a soma de cada um separadamente. Para enumerar os subarrays de uma matriz, considere que você pode identificar cada submatriz por um conjunto de quatro inteiros 1 e N: linha inicial, coluna inicial, linha final e coluna final. Você pode pensar nesses quatro inteiros como as coordenadas dos cantos superior esquerdo e inferior direito do subarray. Tendo identificado um subarray usando essa abordagem de busca completa, a maneira ingênua de encontrar sua soma é adicionar todos os seus números juntos.

Para gerar todas as coordenadas do subarray, podemos usar quatro loops aninhados, um para cada coordenada. Outros dois loops aninhados são suficientes para iterar pelas linhas e colunas do subarray e calcular sua soma. Esses seis loops aninhados nos dão uma algoritmo. Embora o limite de tempo UVa, $O(N^6)$, seja bastante frouxo para esse problema, ele não é tão tolerante. Então, precisamos de uma abordagem mais eficiente. Veja uma possível implementação do algoritmo näive

```
maximo=-999999
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    para k de i a N
      para L de 1 a N
        soma=0
        para a de i a k
          para b de j a L
            soma=soma+MAT[a][b]
          fimpara
        fimpara
        se soma>maximo
          guarda=i,j,k,L
          maximo=soma
        fimse
      fimpara
    fimpara
  fimpara
fimpara
```

Estratégia de programação dinâmica

Com a reutilização de somas de subarrays. Para problemas básicos de programação dinâmica, como UVA 108, o segredo é descobrir quais informações você pode armazenar em etapas anteriores que economizarão tempo em etapas subsequentes.

A ideia-chave para UVa 108 é que os resultados de um cálculo anterior (uma soma, neste caso) podem ser usados em cálculos subsequentes. Especificamente, uma vez que calculamos a soma de parte de um subarray, não precisamos calculá-la repetidamente quando estamos processando outros subarrays que se sobrepõem à mesma região. **Se pudermos encontrar uma maneira eficiente de armazenar e reutilizar essas somas, podemos economizar muito tempo de cálculo**. A estrutura de dados usada em programação dinâmica para esse propósito é tradicionalmente chamada de tabela de memorando .

A tabela de memorandos tem o mesmo tamanho, formato e tipo que a matriz de entrada: a matriz de inteiros. O valor inteiro na posição (i, j) na tabela de memorandos armazena a soma da submatriz cujo canto superior esquerdo é o canto superior esquerdo da matriz de entrada e cujo canto inferior direito é (i, j) na matriz de entrada. Por exemplo, posição da tabela de memorandos $(2, 2)$ armazenaria a soma do subarray quadrado de $(1, 1)$ até $(2, 2)$ em nosso esquema de numeração baseado em 1.

Etapa 1: Gerar a tabela de memorandos
A chave para gerar a tabela de memorandos de forma eficiente é iterar pela matriz de entrada linha por linha e usar resultados anteriores para calcular os subsequentes. Você está basicamente usando a tabela de memorandos para gerar a si mesma. Para você acompanhar, eis como ficaria a tabela de memorandos do exemplo acima citado

```
0   -2    5    5
9    9   10   12
5    6    3    6
4   13   10   11
```

Para ver como isso funciona, considere a $4, 4$ matriz tabela de memorando mostrada abaixo. Os números de 01 para 16 representam a ordem na qual iremos iterar pelos dezesseis valores de entrada e calcular os dezesseis valores da tabela de memorando. Preencher a tabela dessa forma simplificará o código. Podemos ler a entrada e preencher a tabela de memorando simultaneamente. Portanto, não há necessidade de armazenar uma matriz separada de valores de entrada. Todas as informações de que precisamos na próxima etapa estarão na tabela de memorando.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Veja como o processo funciona: Escolha uma célula na matriz. Como exemplo, usarei a célula 10. Isso representa o canto inferior direito de um sub-array de 6 células que começa na célula 1 (três linhas e duas colunas). De acordo com nossa definição de tabela de memorando, a célula 10 deve conter a soma dos valores de entrada 01, 02, 05, 06, 09, e 10. Mas lembre-se de que estamos preenchendo a tabela de memorando em ordem. Então, quando chegamos à célula 10, já temos valores corretos para células 01 até 09. Como podemos usar esses valores?

A resposta é que podemos usar três informações para calcular cada soma de submatriz:

- A célula única que forma seu canto inferior direito. Neste caso, célula 10.
- O subarray acima dele. Neste caso, é o subarray $2, 2$ começando em 01 e terminando em 06. A soma deste subarray é armazenada na célula 06.
- O subarray à esquerda dele. Neste caso, é o subarray $3, 1$ começando em 01 e terminando em 09. A soma deste subarray é armazenada na célula 09.

Então, em vez de adicionar dez células para obter a soma que estamos procurando, podemos simplesmente adicionar três células: 06, 09, e 10. E para submatrizes maiores, a economia de tempo é maior.

Se você estiver acompanhando, notará que esse processo não está muito certo. O subarray acima e o subarray esquerdo 01 se sobrepõem nas células 01 e 05. Então, temos que subtrair a soma dessas células para evitar contagem dupla. Felizmente, a quantidade que precisamos subtrair é armazenada em uma única célula, a 05. Essa quantidade contada duas vezes sempre será uma célula diagonalmente para cima e para a esquerda da nossa célula-alvo.

Agora estamos no final do Passo 1, então nossa tabela de memorando contém as somas de todos os subarrays que começam no canto superior esquerdo do array de entrada. Mas o subarray de soma máxima pode começar em uma posição diferente. Para lidar com o caso geral, precisamos do Passo 2.

Etapa 2: Avalie todas as posições iniciais e finais
Esta etapa compartilha algumas ideias com a abordagem da Busca Completa descrita acima. A diferença é que as somas do subarray já foram calculadas eficientemente na Etapa 1. Então, em vez de um algoritmo de (Busca Completa) $O(N^6)$, temos um $O(N^2)$ passo (Passo 1) seguido por um passo $O(N^4)$ (Etapa 2). Como você deve saber pelas regras da análise de algoritmos assintóticos, isso nos dá uma visão geral de tempo de execução $O(N^4)$, o que é suficiente para que esta solução seja aceita na UVa OJ.

Assim como na Busca Completa, a Etapa 2 usa quatro loops aninhados, um para cada linha inicial, coluna inicial, linha final e coluna final. Essas quatro coordenadas definem o subarray em questão. Temos que verificar cada subarray para ver se ele tem a soma máxima vista até agora.

Veja o que acontece dentro do loop mais interno:

Primeiro, recuperamos a soma memorizada que calculamos na Etapa 1 para o subarray que termina na linha e coluna atuais. Lembre-se de que isso representa a soma do subarray que começa no canto superior esquerdo da entrada. Como isso pode não ser o mesmo que o canto superior esquerdo do subarray que estamos verificando no momento, temos que ajustar a soma usando um processo semelhante ao que usamos na Etapa 1. Para ilustrar, suponha que temos a seguinte tabela de memorando concluída. Como antes, os números identificam a ordem em que as células foram calculadas.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Como exemplo, suponha que nossos loops aninhados atualmente identifiquem uma posição inicial da célula 11 (linha 3, coluna 3) e uma posição final da célula 16 (linha 4, coluna 4). Precisamos encontrar a soma do subarray $2, 2$. O valor memorizado na célula 16 nos dá a soma de todo o array, então temos que nos livrar das partes que não precisamos. Como na Etapa 1, podemos fazer isso usando os valores que estão acima e à esquerda do alvo.

Aqui está outra maneira de analisar a tabela de memorandos:

```
ZZ ZZ XX XX
ZZ ZZ XX XX
```

```
YY YY 11 12
YY YY 15 16
```

As células XX identificam o subarray que está acima do alvo. As células YY identificam o subarray que está à esquerda do alvo. E as células ZZ identificam o subarray que está acima e à esquerda (ou seja, diagonalmente para cima e à esquerda) do alvo.

Esta visualização deve ser familiar do Passo 1. Mas neste passo, precisamos subtrair em vez de adicionar, já que estamos removendo somas que não precisamos. Então, começando com a soma na célula de memorando 16, precisamos:

Subtraia o valor na célula memo 08. Isso remove a soma do subarray que está acima do alvo (os XXs e ZZs). Subtraia o valor na célula memo 14. Isso remove a soma do subarray que está à esquerda do alvo (os YYs e ZZs). Observe que subtraímos o ZZs duas vezes. Nós os contamos duas vezes, assim como contamos duas vezes o valor diagonal no Passo 1. Portanto, precisamos que adicione novamente o valor na célula de memorando 06, os ZZs. Depois que adicionamos de volta uma das cópias contadas duas vezes do ZZs, temos uma soma precisa para nosso subarray de destino. Então, apenas comparamos o resultado com nossa soma máxima atual e atualizamos o máximo, se necessário.

Quando tivermos passado por todas as coordenadas iniciais e finais possíveis, nosso valor máximo será a resposta, e simplesmente imprimiremos esse inteiro.

Importante Se o seu ambiente de programação começa as numerações em 0 (Python, JS, Java, C/C++), para implementar os algoritmos, é conveniente deixar uma linha zerada (a linha 0) e igualmente uma coluna zerada (a coluna 0). Agora as verificações passam a ser em referência à coluna e linha zeros e não a 1 como nos algoritmos. Aliás, no site acima referido o autor faz isso.

O Algoritmo final

Veja como ficou uma possível solução

```
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    se i>1
      MAT[i][j]=MAT[i][j]+MAT[i-1][j]
    fimse
    se j>1
      MAT[i][j]=MAT[i][j]+MAT[i][j-1]
    fimse
    se (i>1) AND (j>1)
      MAT[i][j]=MAT[i][j]-MAT[i-1][j-1]
    fimse
  fimpara
fimpara
ans = 0
para i de 1 a N
  para j de 1 a N
    para k de i a N // note o i
      para L de j a N // idem j
        cur = MAT[k;L]
        se i > 1
          cur=cur-MAT[i-1][L]
        fimse
        se j>1
          cur=cur-MAT[i][j-1]
        fimse
        se (i>1) AND (j>1)
          cur=cur+MAT[i-1][j-1]
        fimse
        se and<cur
          ans=cur
          salva i,j,k,L
        fimse
      fimpara
    fimpara
  fimpara
fimpara
imprima (ans,salva)
```

Se você comparar os 2 algoritmos acima, verá que são muito parecidos. Só que o primeiro (força bruta) tem 6 *paras* um dentro do outro o que caracteriza o desempenho $O(N^6)$. Já o segundo tem os mesmos 6 *paras*, mas primeiro há um dentro do outro ($O(N^2)$) e depois, há 4 *paras*. Da teoria, ao final os 2 *paras* superiores podem ser desconsiderados e o resultado final fica só com $O(N^4)$.

Um exemplo completo

Para você testar seu programa. A matriz

```
-41 -98  91  71 -42   8   3 -79 -17  16
 76 -11  46  74  44  61  42  49 -96  78
  5  -7 -86  43  -2  34  37 -60  84  74
 79   9 -72  -9  98 -56 -10 -36   3  77
-12  -6  62 -26 -57 100 -69  27  24 -99
-99  55  46 -36 -16  37  37 -58  68  42
 66 -81 -83  53  26 -57 -57 -83 -22  91
 90 -22 -46  39 -43  56  57 -15 -43 -61
-97 -61  97 -51  64 -72 -20  21 -64  66
-68  98 -48 -53 -79 -56  27  40  59  40
```

Deu como resposta 588 2, 1, 6, 10.

☞ Para você fazer

Calcule a máxima subsoma da matriz 10×10 abaixo e determine também as dimensões e posição da submatriz. Para tanto identifique o canto superior esquerdo e inferior direito da submatriz (origem=1). Os números estão entre $-600 < n \leq 600$.

```
 123 -320 -582 -233  360 -514   18 -371 -299  231
-550 -341 -323  350  123  -96 -476  -77 -581  129
   2 -387  309   50 -376  316 -354 -104  370 -160
 169  266  381  353 -333 -179   85  274 -311  100
  -8 -571 -423  399 -141  176   74  275   31  120
-163 -227   38 -534  -12  181  175   24  -18  306
-477  274 -254  381  174 -423 -256 -498 -503  195
 148  -93  188 -491 -553  -36  -59   38  -52 -403
 268 -476  143 -249  213  320 -596 -562 -224  -68
-464   46 -326   65  364  101   52  -40  212 -466
```

Responda aqui:

soma	localização

506-75572 -

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 21/10/2024 - 12:06:52.9
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
vivxhb2a V: 1.02
75589 CHRISTOPHER BUCH FILIPAK
24JOG506 - 3 _____ / _____ / _____

UVA 108 - Maior subsoma em matriz

Observação: este texto está baseado em https://www.redgreencode.com/three-ways-solve-uva-108/ do autor Duncan Smith.

O problema original - 108

Como está lá, já traduzido:
Um problema que é simples de resolver numa dimensão é muitas vezes muito mais difícil de resolver em mais de uma dimensão. Considere satisfazer uma expressão booleana na forma normal conjuntiva na qual cada conjunto consiste em exatamente 3 disjuntos. Este problema (3-SAT) é NP-completo. O problema 2-SAT é resolvido de forma bastante eficiente, no entanto. Em contraste, alguns problemas pertencem à mesma classe de complexidade independentemente da dimensionalidade do problema. Dada uma matriz bidimensional de inteiros positivos e negativos, encontre o subretângulo com o maior soma. A soma de um retângulo é a soma de todos os elementos desse retângulo. Neste problema o subretângulo com a maior soma é chamado de subretângulo máximo. Um subretângulo é qualquer submatriz contígua de tamanho 1 x 1 ou maior localizado sobre a matriz. Como exemplo o sub-retângulo máximo do array

```
0   -2   -7   0
9    2   -6   2
-4   1   -4   1
-1   8    0  -2
```

está no canto esquerdo inferior

```
9   2
-4  1
-1  8
```

cuja soma é 15

Entrada:
A entrada consiste em uma matriz N × N de inteiros. A entrada começa com um único inteiro positivo N em uma linha isolada indicando o tamanho do quadrado da matriz bidimensional. Isto é seguido por números inteiros N^2 separados por espaços em branco (novas linhas e espaços). Esses inteiros N^2 constituem a matriz em ordem de linha maior (ou seja, todos os números na primeira linha, da esquerda para a direita, depois, todos os números da segunda linha, da esquerda para a direita, etc.). N pode ser tão grande quanto 100. Os números no array estarão no intervalo $[-127, 127]$.

Saída:
A saída é a soma do sub-retângulo máximo.

Exemplo de entrada

```
4
0 -2 -7 0 9 -2 -6 2
-4 1 -4 1 -1
8 0 -2
```

Exemplo de saída
15

Pesquisa completa

A abordagem de busca completa para este problema requer que enumeremos todos os subarrays do array de entrada e calculemos a soma de cada um separadamente. Para enumerar os subarrays de uma matriz, considere que você pode identificar cada submatriz por um conjunto de quatro inteiros 1 e N: linha inicial, coluna inicial, linha final e coluna final. Você pode pensar nesses quatro inteiros como as coordenadas dos cantos superior esquerdo e inferior direito do subarray. Tendo identificado um subarray usando essa abordagem de busca completa, a maneira ingênua de encontrar sua soma é adicionar todos os seus números juntos.

Para gerar todas as coordenadas do subarray, podemos usar quatro loops aninhados, um para cada coordenada. Outros dois loops aninhados são suficientes para iterar pelas linhas e colunas do subarray e calcular sua soma. Esses seis loops aninhados nos dão uma algoritmo. Embora o limite de tempo UVa, $O(N^6)$, seja bastante frouxo para esse problema, ele não é tão tolerante. Então, precisamos de uma abordagem mais eficiente. Veja uma possível implementação do algoritmo näive

```
maximo=-999999
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    para k de i a N
      para L de 1 a N
        soma=0
        para a de i a k
          para b de j a L
            soma=soma+MAT[a][b]
          fimpara
        fimpara
        se soma>maximo
          guarda=i,j,k,L
          maximo=soma
        fimse
      fimpara
    fimpara
  fimpara
fimpara
```

Estratégia de programação dinâmica

Com a reutilização de somas de subarrays. Para problemas básicos de programação dinâmica, como UVA 108, o segredo é descobrir quais informações você pode armazenar em etapas anteriores que economizarão tempo em etapas subsequentes.

A ideia-chave para UVa 108 é que os resultados de um cálculo anterior (uma soma, neste caso) podem ser usados em cálculos subsequentes. Especificamente, uma vez que calculamos a soma de parte de um subarray, não precisamos calculá-la repetidamente quando estamos processando outros subarrays que se sobrepõem à mesma região. **Se pudermos encontrar uma maneira eficiente de armazenar e reutilizar essas somas, podemos economizar muito tempo de cálculo**. A estrutura de dados usada em programação dinâmica para esse propósito é tradicionalmente chamada de tabela de memorando .

A tabela de memorandos tem o mesmo tamanho, formato e tipo que a matriz de entrada: a matriz de inteiros. O valor inteiro na posição (i, j) na tabela de memorandos armazena a soma da submatriz cujo canto superior esquerdo é o canto superior esquerdo da matriz de entrada e cujo canto inferior direito é (i, j) na matriz de entrada. Por exemplo, posição da tabela de memorandos $(2, 2)$ armazenaria a soma do subarray quadrado de $(1, 1)$ até $(2, 2)$ em nosso esquema de numeração baseado em 1.

Etapa 1: Gerar a tabela de memorandos
A chave para gerar a tabela de memorandos de forma eficiente é iterar pela matriz de entrada linha por linha e usar resultados anteriores para calcular os subsequentes. Você está basicamente usando a tabela de memorandos para gerar a si mesma. Para você acompanhar, eis como ficaria a tabela de memorandos do exemplo acima citado

```
0   -2    5    5
9    9   10   12
5    6    3    6
4   13   10   11
```

Para ver como isso funciona, considere a $4, 4$ matriz tabela de memorando mostrada abaixo. Os números de 01 para 16 representam a ordem na qual iremos iterar pelos dezesseis valores de entrada e calcular os dezesseis valores da tabela de memorando. Preencher a tabela dessa forma simplificará o código. Podemos ler a entrada e preencher a tabela de memorando simultaneamente. Portanto, não há necessidade de armazenar uma matriz separada de valores de entrada. Todas as informações de que precisamos na próxima etapa estarão na tabela de memorando.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Veja como o processo funciona: Escolha uma célula na matriz. Como exemplo, usarei a célula 10. Isso representa o canto inferior direito de um sub-array de 6 células que começa na célula 1 (três linhas e duas colunas). De acordo com nossa definição de tabela de memorando, a célula 10 deve conter a soma dos valores de entrada 01, 02, 05, 06, 09, e 10. Mas lembre-se de que estamos preenchendo a tabela de memorando em ordem. Então, quando chegamos à célula 10, já temos valores corretos para células 01 até 09. Como podemos usar esses valores?

A resposta é que podemos usar três informações para calcular cada soma de submatriz:

- A célula única que forma seu canto inferior direito. Neste caso, célula 10.
- O subarray acima dele. Neste caso, é o subarray $2, 2$ começando em 01 e terminando em 06. A soma deste subarray é armazenada na célula 06.
- O subarray à esquerda dele. Neste caso, é o subarray $3, 1$ começando em 01 e terminando em 09. A soma deste subarray é armazenada na célula 09.

Então, em vez de adicionar dez células para obter a soma que estamos procurando, podemos simplesmente adicionar três células: 06, 09, e 10. E para submatrizes maiores, a economia de tempo é maior.

Se você estiver acompanhando, notará que esse processo não está muito certo. O subarray acima e o subarray esquerdo 01 se sobrepõem nas células 01 e 05. Então, temos que subtrair a soma dessas células para evitar contagem dupla. Felizmente, a quantidade que precisamos subtrair é armazenada em uma única célula, a 05. Essa quantidade contada duas vezes sempre será uma célula diagonalmente para cima e para a esquerda da nossa célula-alvo.

Agora estamos no final do Passo 1, então nossa tabela de memorando contém as somas de todos os subarrays que começam no canto superior esquerdo do array de entrada. Mas o subarray de soma máxima pode começar em uma posição diferente. Para lidar com o caso geral, precisamos do Passo 2.

Etapa 2: Avalie todas as posições iniciais e finais
Esta etapa compartilha algumas ideias com a abordagem da Busca Completa descrita acima. A diferença é que as somas do subarray já foram calculadas eficientemente na Etapa 1. Então, em vez de um algoritmo de (Busca Completa) $O(N^6)$, temos um $O(N^2)$ passo (Passo 1) seguido por um passo $O(N^4)$ (Etapa 2). Como você deve saber pelas regras da análise de algoritmos assintóticos, isso nos dá uma visão geral de tempo de execução $O(N^4)$, o que é suficiente para que esta solução seja aceita na UVa OJ.

Assim como na Busca Completa, a Etapa 2 usa quatro loops aninhados, um para cada linha inicial, coluna inicial, linha final e coluna final. Essas quatro coordenadas definem o subarray em questão. Temos que verificar cada subarray para ver se ele tem a soma máxima vista até agora.

Veja o que acontece dentro do loop mais interno:

Primeiro, recuperamos a soma memorizada que calculamos na Etapa 1 para o subarray que termina na linha e coluna atuais. Lembre-se de que isso representa a soma do subarray que começa no canto superior esquerdo da entrada. Como isso pode não ser o mesmo que o canto superior esquerdo do subarray que estamos verificando no momento, temos que ajustar a soma usando um processo semelhante ao que usamos na Etapa 1. Para ilustrar, suponha que temos a seguinte tabela de memorando concluída. Como antes, os números identificam a ordem em que as células foram calculadas.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Como exemplo, suponha que nossos loops aninhados atualmente identifiquem uma posição inicial da célula 11 (linha 3, coluna 3) e uma posição final da célula 16 (linha 4, coluna 4). Precisamos encontrar a soma do subarray $2, 2$. O valor memorizado na célula 16 nos dá a soma de todo o array, então temos que nos livrar das partes que não precisamos. Como na Etapa 1, podemos fazer isso usando os valores que estão acima e à esquerda do alvo.

Aqui está outra maneira de analisar a tabela de memorandos:

```
ZZ ZZ XX XX
ZZ ZZ XX XX
```

```
YY YY 11 12
YY YY 15 16
```

As células XX identificam o subarray que está acima do alvo. As células YY identificam o subarray que está à esquerda do alvo. E as células ZZ identificam o subarray que está acima e à esquerda (ou seja, diagonalmente para cima e à esquerda) do alvo.

Esta visualização deve ser familiar do Passo 1. Mas neste passo, precisamos subtrair em vez de adicionar, já que estamos removendo somas que não precisamos. Então, começando com a soma na célula de memorando 16, precisamos:

Subtraia o valor na célula memo 08. Isso remove a soma do subarray que está acima do alvo (os XXs e ZZs). Subtraia o valor na célula memo 14. Isso remove a soma do subarray que está à esquerda do alvo (os YYs e ZZs). Observe que subtraímos o ZZs duas vezes. Nós os contamos duas vezes, assim como contamos duas vezes o valor diagonal no Passo 1. Portanto, precisamos que adicione novamente o valor na célula de memorando 06, os ZZs. Depois que adicionamos de volta uma das cópias contadas duas vezes do ZZs, temos uma soma precisa para nosso subarray de destino. Então, apenas comparamos o resultado com nossa soma máxima atual e atualizamos o máximo, se necessário.

Quando tivermos passado por todas as coordenadas iniciais e finais possíveis, nosso valor máximo será a resposta, e simplesmente imprimiremos esse inteiro.

Importante Se o seu ambiente de programação começa as numerações em 0 (Python, JS, Java, C/C++), para implementar os algoritmos, é conveniente deixar uma linha zerada (a linha 0) e igualmente uma coluna zerada (a coluna 0). Agora as verificações passam a ser em referência à coluna e linha zeros e não a 1 como nos algoritmos. Aliás, no site acima referido o autor faz isso.

O Algoritmo final

Veja como ficou uma possível solução

```
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    se i>1
      MAT[i][j]=MAT[i][j]+MAT[i-1][j]
    fimse
    se j>1
      MAT[i][j]=MAT[i][j]+MAT[i][j-1]
    fimse
    se (i>1) AND (j>1)
      MAT[i][j]=MAT[i][j]-MAT[i-1][j-1]
    fimse
  fimpara
fimpara
ans = 0
para i de 1 a N
  para j de 1 a N
    para k de i a N // note o i
      para L de j a N // idem j
        cur = MAT[k;L]
        se i > 1
          cur=cur-MAT[i-1][L]
        fimse
        se j>1
          cur=cur-MAT[i][j-1]
        fimse
        se (i>1) AND (j>1)
          cur=cur+MAT[i-1][j-1]
        fimse
        se and<cur
          ans=cur
          salva i,j,k,L
        fimse
      fimpara
    fimpara
  fimpara
fimpara
imprima (ans,salva)
```

Se você comparar os 2 algoritmos acima, verá que são muito parecidos. Só que o primeiro (força bruta) tem 6 *paras* um dentro do outro o que caracteriza o desempenho $O(N^6)$. Já o segundo tem os mesmos 6 *paras*, mas primeiro há um dentro do outro ($O(N^2)$) e depois, há 4 *paras*. Da teoria, ao final os 2 *paras* superiores podem ser desconsiderados e o resultado final fica só com $O(N^4)$.

Um exemplo completo

Para você testar seu programa. A matriz

```
-41 -98  91  71 -42   8   3 -79 -17  16
 76 -11  46  74  44  61  42  49 -96  78
  5  -7 -86  43  -2  34  37 -60  84  74
 79   9 -72  -9  98 -56 -10 -36   3  77
-12  -6  62 -26 -57 100 -69  27  24 -99
-99  55  46 -36 -16  37  37 -58  68  42
 66 -81 -83  53  26 -57 -57 -83 -22  91
 90 -22 -46  39 -43  56  57 -15 -43 -61
-97 -61  97 -51  64 -72 -20  21 -64  66
-68  98 -48 -53 -79 -56  27  40  59  40
```

Deu como resposta 588 2, 1, 6, 10.

☞ Para você fazer

Calcule a máxima subsoma da matriz 10×10 abaixo e determine também as dimensões e posição da submatriz. Para tanto identifique o canto superior esquerdo e inferior direito da submatriz (origem=1). Os números estão entre $-600 < n \leq 600$.

```
-163 -161  359   34  332 -307   82  231 -302  221
 278 -140  295  260  202 -379 -186    1  140  175
   9  -11  -92  173 -456  152 -234 -322 -417 -122
  22  173 -304 -256   39 -530  -67  176 -438  209
 -52 -598    1 -280   63 -576 -343  -49  133  110
 330 -307   21 -568  313 -104 -203  204 -365  225
  90 -100  293 -316 -546  237 -239 -159 -114  -86
-383  105 -385  350 -560  -98 -439 -418  204  -30
-328  264 -431 -278 -244 -447 -138 -248  350 -236
 140   63  -23 -116 -280 -442   11   41 -270 -114
```

Responda aqui:

soma	localização

506-75589 -

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 21/10/2024 - 12:06:52.9
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
vivxhb2a V: 1.02
75596 ERAN MARTINEZ RAMOS
24JOG506 - 4 _____ / _____ / _____

UVA 108 - Maior subsoma em matriz

Observação: este texto está baseado em https://www.redgreencode.com/three-ways-solve-uva-108/ do autor Duncan Smith.

O problema original - 108

Como está lá, já traduzido:
Um problema que é simples de resolver numa dimensão é muitas vezes muito mais difícil de resolver em mais de uma dimensão. Considere satisfazer uma expressão booleana na forma normal conjuntiva na qual cada conjunto consiste em exatamente 3 disjuntos. Este problema (3-SAT) é NP-completo. O problema 2-SAT é resolvido de forma bastante eficiente, no entanto. Em contraste, alguns problemas pertencem à mesma classe de complexidade independentemente da dimensionalidade do problema. Dada uma matriz bidimensional de inteiros positivos e negativos, encontre o subretângulo com o maior soma. A soma de um retângulo é a soma de todos os elementos desse retângulo. Neste problema o subretângulo com a maior soma é chamado de subretângulo máximo. Um subretângulo é qualquer submatriz contígua de tamanho 1 x 1 ou maior localizado sobre a matriz. Como exemplo o sub-retângulo máximo do array

```
0   -2   -7   0
9    2   -6   2
-4   1   -4   1
-1   8    0  -2
```

está no canto esquerdo inferior

```
9   2
-4  1
-1  8
```

cuja soma é 15

Entrada:
A entrada consiste em uma matriz N × N de inteiros. A entrada começa com um único inteiro positivo N em uma linha isolada indicando o tamanho do quadrado da matriz bidimensional. Isto é seguido por números inteiros N^2 separados por espaços em branco (novas linhas e espaços). Esses inteiros N^2 constituem a matriz em ordem de linha maior (ou seja, todos os números na primeira linha, da esquerda para a direita, depois, todos os números da segunda linha, da esquerda para a direita, etc.). N pode ser tão grande quanto 100. Os números no array estarão no intervalo $[-127, 127]$.

Saída:
A saída é a soma do sub-retângulo máximo.

Exemplo de entrada

```
4
0 -2 -7 0 9 -2 -6 2
-4 1 -4 1 -1
8 0 -2
```

Exemplo de saída
15

Pesquisa completa

A abordagem de busca completa para este problema requer que enumeremos todos os subarrays do array de entrada e calculemos a soma de cada um separadamente. Para enumerar os subarrays de uma matriz, considere que você pode identificar cada submatriz por um conjunto de quatro inteiros 1 e N: linha inicial, coluna inicial, linha final e coluna final. Você pode pensar nesses quatro inteiros como as coordenadas dos cantos superior esquerdo e inferior direito do subarray. Tendo identificado um subarray usando essa abordagem de busca completa, a maneira ingênua de encontrar sua soma é adicionar todos os seus números juntos.

Para gerar todas as coordenadas do subarray, podemos usar quatro loops aninhados, um para cada coordenada. Outros dois loops aninhados são suficientes para iterar pelas linhas e colunas do subarray e calcular sua soma. Esses seis loops aninhados nos dão uma algoritmo. Embora o limite de tempo UVa, $O(N^6)$, seja bastante frouxo para esse problema, ele não é tão tolerante. Então, precisamos de uma abordagem mais eficiente. Veja uma possível implementação do algoritmo näive

```
maximo=-999999
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    para k de i a N
      para L de 1 a N
        soma=0
        para a de i a k
          para b de j a L
            soma=soma+MAT[a][b]
          fimpara
        fimpara
        se soma>maximo
          guarda=i,j,k,L
          maximo=soma
        fimse
      fimpara
    fimpara
  fimpara
fimpara
```

Estratégia de programação dinâmica

Com a reutilização de somas de subarrays. Para problemas básicos de programação dinâmica, como UVA 108, o segredo é descobrir quais informações você pode armazenar em etapas anteriores que economizarão tempo em etapas subsequentes.

A ideia-chave para UVa 108 é que os resultados de um cálculo anterior (uma soma, neste caso) podem ser usados em cálculos subsequentes. Especificamente, uma vez que calculamos a soma de parte de um subarray, não precisamos calculá-la repetidamente quando estamos processando outros subarrays que se sobrepõem à mesma região. **Se pudermos encontrar uma maneira eficiente de armazenar e reutilizar essas somas, podemos economizar muito tempo de cálculo**. A estrutura de dados usada em programação dinâmica para esse propósito é tradicionalmente chamada de tabela de memorando .

A tabela de memorandos tem o mesmo tamanho, formato e tipo que a matriz de entrada: a matriz de inteiros. O valor inteiro na posição (i, j) na tabela de memorandos armazena a soma da submatriz cujo canto superior esquerdo é o canto superior esquerdo da matriz de entrada e cujo canto inferior direito é (i, j) na matriz de entrada. Por exemplo, posição da tabela de memorandos $(2, 2)$ armazenaria a soma do subarray quadrado de $(1, 1)$ até $(2, 2)$ em nosso esquema de numeração baseado em 1.

Etapa 1: Gerar a tabela de memorandos
A chave para gerar a tabela de memorandos de forma eficiente é iterar pela matriz de entrada linha por linha e usar resultados anteriores para calcular os subsequentes. Você está basicamente usando a tabela de memorandos para gerar a si mesma. Para você acompanhar, eis como ficaria a tabela de memorandos do exemplo acima citado

```
0   -2    5    5
9    9   10   12
5    6    3    6
4   13   10   11
```

Para ver como isso funciona, considere a $4, 4$ matriz tabela de memorando mostrada abaixo. Os números de 01 para 16 representam a ordem na qual iremos iterar pelos dezesseis valores de entrada e calcular os dezesseis valores da tabela de memorando. Preencher a tabela dessa forma simplificará o código. Podemos ler a entrada e preencher a tabela de memorando simultaneamente. Portanto, não há necessidade de armazenar uma matriz separada de valores de entrada. Todas as informações de que precisamos na próxima etapa estarão na tabela de memorando.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Veja como o processo funciona: Escolha uma célula na matriz. Como exemplo, usarei a célula 10. Isso representa o canto inferior direito de um sub-array de 6 células que começa na célula 1 (três linhas e duas colunas). De acordo com nossa definição de tabela de memorando, a célula 10 deve conter a soma dos valores de entrada 01, 02, 05, 06, 09, e 10. Mas lembre-se de que estamos preenchendo a tabela de memorando em ordem. Então, quando chegamos à célula 10, já temos valores corretos para células 01 até 09. Como podemos usar esses valores?

A resposta é que podemos usar três informações para calcular cada soma de submatriz:

- A célula única que forma seu canto inferior direito. Neste caso, célula 10.
- O subarray acima dele. Neste caso, é o subarray $2, 2$ começando em 01 e terminando em 06. A soma deste subarray é armazenada na célula 06.
- O subarray à esquerda dele. Neste caso, é o subarray $3, 1$ começando em 01 e terminando em 09. A soma deste subarray é armazenada na célula 09.

Então, em vez de adicionar dez células para obter a soma que estamos procurando, podemos simplesmente adicionar três células: 06, 09, e 10. E para submatrizes maiores, a economia de tempo é maior.

Se você estiver acompanhando, notará que esse processo não está muito certo. O subarray acima e o subarray esquerdo 01 se sobrepõem nas células 01 e 05. Então, temos que subtrair a soma dessas células para evitar contagem dupla. Felizmente, a quantidade que precisamos subtrair é armazenada em uma única célula, a 05. Essa quantidade contada duas vezes sempre será uma célula diagonalmente para cima e para a esquerda da nossa célula-alvo.

Agora estamos no final do Passo 1, então nossa tabela de memorando contém as somas de todos os subarrays que começam no canto superior esquerdo do array de entrada. Mas o subarray de soma máxima pode começar em uma posição diferente. Para lidar com o caso geral, precisamos do Passo 2.

Etapa 2: Avalie todas as posições iniciais e finais
Esta etapa compartilha algumas ideias com a abordagem da Busca Completa descrita acima. A diferença é que as somas do subarray já foram calculadas eficientemente na Etapa 1. Então, em vez de um algoritmo de (Busca Completa) $O(N^6)$, temos um $O(N^2)$ passo (Passo 1) seguido por um passo $O(N^4)$ (Etapa 2). Como você deve saber pelas regras da análise de algoritmos assintóticos, isso nos dá uma visão geral de tempo de execução $O(N^4)$, o que é suficiente para que esta solução seja aceita na UVa OJ.

Assim como na Busca Completa, a Etapa 2 usa quatro loops aninhados, um para cada linha inicial, coluna inicial, linha final e coluna final. Essas quatro coordenadas definem o subarray em questão. Temos que verificar cada subarray para ver se ele tem a soma máxima vista até agora.

Veja o que acontece dentro do loop mais interno:

Primeiro, recuperamos a soma memorizada que calculamos na Etapa 1 para o subarray que termina na linha e coluna atuais. Lembre-se de que isso representa a soma do subarray que começa no canto superior esquerdo da entrada. Como isso pode não ser o mesmo que o canto superior esquerdo do subarray que estamos verificando no momento, temos que ajustar a soma usando um processo semelhante ao que usamos na Etapa 1. Para ilustrar, suponha que temos a seguinte tabela de memorando concluída. Como antes, os números identificam a ordem em que as células foram calculadas.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Como exemplo, suponha que nossos loops aninhados atualmente identifiquem uma posição inicial da célula 11 (linha 3, coluna 3) e uma posição final da célula 16 (linha 4, coluna 4). Precisamos encontrar a soma do subarray $2, 2$. O valor memorizado na célula 16 nos dá a soma de todo o array, então temos que nos livrar das partes que não precisamos. Como na Etapa 1, podemos fazer isso usando os valores que estão acima e à esquerda do alvo.

Aqui está outra maneira de analisar a tabela de memorandos:

```
ZZ ZZ XX XX
ZZ ZZ XX XX
```

```
YY YY 11 12
YY YY 15 16
```

As células XX identificam o subarray que está acima do alvo. As células YY identificam o subarray que está à esquerda do alvo. E as células ZZ identificam o subarray que está acima e à esquerda (ou seja, diagonalmente para cima e à esquerda) do alvo.

Esta visualização deve ser familiar do Passo 1. Mas neste passo, precisamos subtrair em vez de adicionar, já que estamos removendo somas que não precisamos. Então, começando com a soma na célula de memorando 16, precisamos:

Subtraia o valor na célula memo 08. Isso remove a soma do subarray que está acima do alvo (os XXs e ZZs). Subtraia o valor na célula memo 14. Isso remove a soma do subarray que está à esquerda do alvo (os YYs e ZZs). Observe que subtraímos o ZZs duas vezes. Nós os contamos duas vezes, assim como contamos duas vezes o valor diagonal no Passo 1. Portanto, precisamos que adicione novamente o valor na célula de memorando 06, os ZZs. Depois que adicionamos de volta uma das cópias contadas duas vezes do ZZs, temos uma soma precisa para nosso subarray de destino. Então, apenas comparamos o resultado com nossa soma máxima atual e atualizamos o máximo, se necessário.

Quando tivermos passado por todas as coordenadas iniciais e finais possíveis, nosso valor máximo será a resposta, e simplesmente imprimiremos esse inteiro.

Importante Se o seu ambiente de programação começa as numerações em 0 (Python, JS, Java, C/C++), para implementar os algoritmos, é conveniente deixar uma linha zerada (a linha 0) e igualmente uma coluna zerada (a coluna 0). Agora as verificações passam a ser em referência à coluna e linha zeros e não a 1 como nos algoritmos. Aliás, no site acima referido o autor faz isso.

O Algoritmo final

Veja como ficou uma possível solução

```
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    se i>1
      MAT[i][j]=MAT[i][j]+MAT[i-1][j]
    fimse
    se j>1
      MAT[i][j]=MAT[i][j]+MAT[i][j-1]
    fimse
    se (i>1) AND (j>1)
      MAT[i][j]=MAT[i][j]-MAT[i-1][j-1]
    fimse
  fimpara
fimpara
ans = 0
para i de 1 a N
  para j de 1 a N
    para k de i a N // note o i
      para L de j a N // idem j
        cur = MAT[k;L]
        se i > 1
          cur=cur-MAT[i-1][L]
        fimse
        se j>1
          cur=cur-MAT[i][j-1]
        fimse
        se (i>1) AND (j>1)
          cur=cur+MAT[i-1][j-1]
        fimse
        se and<cur
          ans=cur
          salva i,j,k,L
        fimse
      fimpara
    fimpara
  fimpara
fimpara
imprima (ans,salva)
```

Se você comparar os 2 algoritmos acima, verá que são muito parecidos. Só que o primeiro (força bruta) tem 6 *paras* um dentro do outro o que caracteriza o desempenho $O(N^6)$. Já o segundo tem os mesmos 6 *paras*, mas primeiro há um dentro do outro ($O(N^2)$) e depois, há 4 *paras*. Da teoria, ao final os 2 *paras* superiores podem ser desconsiderados e o resultado final fica só com $O(N^4)$.

Um exemplo completo

Para você testar seu programa. A matriz

```
-41 -98  91  71 -42   8   3 -79 -17  16
 76 -11  46  74  44  61  42  49 -96  78
  5  -7 -86  43  -2  34  37 -60  84  74
 79   9 -72  -9  98 -56 -10 -36   3  77
-12  -6  62 -26 -57 100 -69  27  24 -99
-99  55  46 -36 -16  37  37 -58  68  42
 66 -81 -83  53  26 -57 -57 -83 -22  91
 90 -22 -46  39 -43  56  57 -15 -43 -61
-97 -61  97 -51  64 -72 -20  21 -64  66
-68  98 -48 -53 -79 -56  27  40  59  40
```

Deu como resposta 588 2, 1, 6, 10.

☞ Para você fazer

Calcule a máxima subsoma da matriz 10×10 abaixo e determine também as dimensões e posição da submatriz. Para tanto identifique o canto superior esquerdo e inferior direito da submatriz (origem=1). Os números estão entre $-600 < n \leq 600$.

```
-357  109  176  -37 -361  397 -341  -78 -434 -381
  95  -70  -79 -300   39 -411  -29  271 -518   70
 358 -197  370 -145 -463  382  131 -192  337  371
  32  -71 -153  291 -586  -87  138 -392 -276  243
 258 -394 -269 -131 -291  -17  136  207 -592  308
-193 -386 -452  118 -520 -322 -472 -366  250 -528
 231 -150  -35   61 -563  155   25 -455 -448 -181
-404 -152  295 -388  202  135  190  134 -459 -120
-376  386 -340  335 -257   77  117  300  317 -259
-234 -379  392 -565 -397 -556 -335 -596  100 -189
```

Responda aqui:

soma	localização

506-75596 -

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 21/10/2024 - 12:06:52.9
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
vivxhb2a V: 1.02
75608 HENRIQUE IVANKIO GUSSE
24JOG506 - 5 _____ / _____ / _____

UVA 108 - Maior subsoma em matriz

Observação: este texto está baseado em https://www.redgreencode.com/three-ways-solve-uva-108/ do autor Duncan Smith.

O problema original - 108

Como está lá, já traduzido:
Um problema que é simples de resolver numa dimensão é muitas vezes muito mais difícil de resolver em mais de uma dimensão. Considere satisfazer uma expressão booleana na forma normal conjuntiva na qual cada conjunto consiste em exatamente 3 disjuntos. Este problema (3-SAT) é NP-completo. O problema 2-SAT é resolvido de forma bastante eficiente, no entanto. Em contraste, alguns problemas pertencem à mesma classe de complexidade independentemente da dimensionalidade do problema. Dada uma matriz bidimensional de inteiros positivos e negativos, encontre o subretângulo com o maior soma. A soma de um retângulo é a soma de todos os elementos desse retângulo. Neste problema o subretângulo com a maior soma é chamado de subretângulo máximo. Um subretângulo é qualquer submatriz contígua de tamanho 1 x 1 ou maior localizado sobre a matriz. Como exemplo o sub-retângulo máximo do array

```
0   -2   -7   0
9    2   -6   2
-4   1   -4   1
-1   8    0  -2
```

está no canto esquerdo inferior

```
9   2
-4  1
-1  8
```

cuja soma é 15

Entrada:
A entrada consiste em uma matriz N × N de inteiros. A entrada começa com um único inteiro positivo N em uma linha isolada indicando o tamanho do quadrado da matriz bidimensional. Isto é seguido por números inteiros N^2 separados por espaços em branco (novas linhas e espaços). Esses inteiros N^2 constituem a matriz em ordem de linha maior (ou seja, todos os números na primeira linha, da esquerda para a direita, depois, todos os números da segunda linha, da esquerda para a direita, etc.). N pode ser tão grande quanto 100. Os números no array estarão no intervalo $[-127, 127]$.

Saída:
A saída é a soma do sub-retângulo máximo.

Exemplo de entrada

```
4
0 -2 -7 0 9 -2 -6 2
-4 1 -4 1 -1
8 0 -2
```

Exemplo de saída
15

Pesquisa completa

A abordagem de busca completa para este problema requer que enumeremos todos os subarrays do array de entrada e calculemos a soma de cada um separadamente. Para enumerar os subarrays de uma matriz, considere que você pode identificar cada submatriz por um conjunto de quatro inteiros 1 e N: linha inicial, coluna inicial, linha final e coluna final. Você pode pensar nesses quatro inteiros como as coordenadas dos cantos superior esquerdo e inferior direito do subarray. Tendo identificado um subarray usando essa abordagem de busca completa, a maneira ingênua de encontrar sua soma é adicionar todos os seus números juntos.

Para gerar todas as coordenadas do subarray, podemos usar quatro loops aninhados, um para cada coordenada. Outros dois loops aninhados são suficientes para iterar pelas linhas e colunas do subarray e calcular sua soma. Esses seis loops aninhados nos dão uma algoritmo. Embora o limite de tempo UVa, $O(N^6)$, seja bastante frouxo para esse problema, ele não é tão tolerante. Então, precisamos de uma abordagem mais eficiente. Veja uma possível implementação do algoritmo näive

```
maximo=-999999
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    para k de i a N
      para L de 1 a N
        soma=0
        para a de i a k
          para b de j a L
            soma=soma+MAT[a][b]
          fimpara
        fimpara
        se soma>maximo
          guarda=i,j,k,L
          maximo=soma
        fimse
      fimpara
    fimpara
  fimpara
fimpara
```

Estratégia de programação dinâmica

Com a reutilização de somas de subarrays. Para problemas básicos de programação dinâmica, como UVA 108, o segredo é descobrir quais informações você pode armazenar em etapas anteriores que economizarão tempo em etapas subsequentes.

A ideia-chave para UVa 108 é que os resultados de um cálculo anterior (uma soma, neste caso) podem ser usados em cálculos subsequentes. Especificamente, uma vez que calculamos a soma de parte de um subarray, não precisamos calculá-la repetidamente quando estamos processando outros subarrays que se sobrepõem à mesma região. **Se pudermos encontrar uma maneira eficiente de armazenar e reutilizar essas somas, podemos economizar muito tempo de cálculo**. A estrutura de dados usada em programação dinâmica para esse propósito é tradicionalmente chamada de tabela de memorando .

A tabela de memorandos tem o mesmo tamanho, formato e tipo que a matriz de entrada: a matriz de inteiros. O valor inteiro na posição (i, j) na tabela de memorandos armazena a soma da submatriz cujo canto superior esquerdo é o canto superior esquerdo da matriz de entrada e cujo canto inferior direito é (i, j) na matriz de entrada. Por exemplo, posição da tabela de memorandos $(2, 2)$ armazenaria a soma do subarray quadrado de $(1, 1)$ até $(2, 2)$ em nosso esquema de numeração baseado em 1.

Etapa 1: Gerar a tabela de memorandos
A chave para gerar a tabela de memorandos de forma eficiente é iterar pela matriz de entrada linha por linha e usar resultados anteriores para calcular os subsequentes. Você está basicamente usando a tabela de memorandos para gerar a si mesma. Para você acompanhar, eis como ficaria a tabela de memorandos do exemplo acima citado

```
0   -2    5    5
9    9   10   12
5    6    3    6
4   13   10   11
```

Para ver como isso funciona, considere a $4, 4$ matriz tabela de memorando mostrada abaixo. Os números de 01 para 16 representam a ordem na qual iremos iterar pelos dezesseis valores de entrada e calcular os dezesseis valores da tabela de memorando. Preencher a tabela dessa forma simplificará o código. Podemos ler a entrada e preencher a tabela de memorando simultaneamente. Portanto, não há necessidade de armazenar uma matriz separada de valores de entrada. Todas as informações de que precisamos na próxima etapa estarão na tabela de memorando.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Veja como o processo funciona: Escolha uma célula na matriz. Como exemplo, usarei a célula 10. Isso representa o canto inferior direito de um sub-array de 6 células que começa na célula 1 (três linhas e duas colunas). De acordo com nossa definição de tabela de memorando, a célula 10 deve conter a soma dos valores de entrada 01, 02, 05, 06, 09, e 10. Mas lembre-se de que estamos preenchendo a tabela de memorando em ordem. Então, quando chegamos à célula 10, já temos valores corretos para células 01 até 09. Como podemos usar esses valores?

A resposta é que podemos usar três informações para calcular cada soma de submatriz:

- A célula única que forma seu canto inferior direito. Neste caso, célula 10.
- O subarray acima dele. Neste caso, é o subarray $2, 2$ começando em 01 e terminando em 06. A soma deste subarray é armazenada na célula 06.
- O subarray à esquerda dele. Neste caso, é o subarray $3, 1$ começando em 01 e terminando em 09. A soma deste subarray é armazenada na célula 09.

Então, em vez de adicionar dez células para obter a soma que estamos procurando, podemos simplesmente adicionar três células: 06, 09, e 10. E para submatrizes maiores, a economia de tempo é maior.

Se você estiver acompanhando, notará que esse processo não está muito certo. O subarray acima e o subarray esquerdo 01 se sobrepõem nas células 01 e 05. Então, temos que subtrair a soma dessas células para evitar contagem dupla. Felizmente, a quantidade que precisamos subtrair é armazenada em uma única célula, a 05. Essa quantidade contada duas vezes sempre será uma célula diagonalmente para cima e para a esquerda da nossa célula-alvo.

Agora estamos no final do Passo 1, então nossa tabela de memorando contém as somas de todos os subarrays que começam no canto superior esquerdo do array de entrada. Mas o subarray de soma máxima pode começar em uma posição diferente. Para lidar com o caso geral, precisamos do Passo 2.

Etapa 2: Avalie todas as posições iniciais e finais
Esta etapa compartilha algumas ideias com a abordagem da Busca Completa descrita acima. A diferença é que as somas do subarray já foram calculadas eficientemente na Etapa 1. Então, em vez de um algoritmo de (Busca Completa) $O(N^6)$, temos um $O(N^2)$ passo (Passo 1) seguido por um passo $O(N^4)$ (Etapa 2). Como você deve saber pelas regras da análise de algoritmos assintóticos, isso nos dá uma visão geral de tempo de execução $O(N^4)$, o que é suficiente para que esta solução seja aceita na UVa OJ.

Assim como na Busca Completa, a Etapa 2 usa quatro loops aninhados, um para cada linha inicial, coluna inicial, linha final e coluna final. Essas quatro coordenadas definem o subarray em questão. Temos que verificar cada subarray para ver se ele tem a soma máxima vista até agora.

Veja o que acontece dentro do loop mais interno:

Primeiro, recuperamos a soma memorizada que calculamos na Etapa 1 para o subarray que termina na linha e coluna atuais. Lembre-se de que isso representa a soma do subarray que começa no canto superior esquerdo da entrada. Como isso pode não ser o mesmo que o canto superior esquerdo do subarray que estamos verificando no momento, temos que ajustar a soma usando um processo semelhante ao que usamos na Etapa 1. Para ilustrar, suponha que temos a seguinte tabela de memorando concluída. Como antes, os números identificam a ordem em que as células foram calculadas.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Como exemplo, suponha que nossos loops aninhados atualmente identifiquem uma posição inicial da célula 11 (linha 3, coluna 3) e uma posição final da célula 16 (linha 4, coluna 4). Precisamos encontrar a soma do subarray $2, 2$. O valor memorizado na célula 16 nos dá a soma de todo o array, então temos que nos livrar das partes que não precisamos. Como na Etapa 1, podemos fazer isso usando os valores que estão acima e à esquerda do alvo.

Aqui está outra maneira de analisar a tabela de memorandos:

```
ZZ ZZ XX XX
ZZ ZZ XX XX
```

```
YY YY 11 12
YY YY 15 16
```

As células XX identificam o subarray que está acima do alvo. As células YY identificam o subarray que está à esquerda do alvo. E as células ZZ identificam o subarray que está acima e à esquerda (ou seja, diagonalmente para cima e à esquerda) do alvo.

Esta visualização deve ser familiar do Passo 1. Mas neste passo, precisamos subtrair em vez de adicionar, já que estamos removendo somas que não precisamos. Então, começando com a soma na célula de memorando 16, precisamos:

Subtraia o valor na célula memo 08. Isso remove a soma do subarray que está acima do alvo (os XXs e ZZs). Subtraia o valor na célula memo 14. Isso remove a soma do subarray que está à esquerda do alvo (os YYs e ZZs). Observe que subtraímos o ZZs duas vezes. Nós os contamos duas vezes, assim como contamos duas vezes o valor diagonal no Passo 1. Portanto, precisamos que adicione novamente o valor na célula de memorando 06, os ZZs. Depois que adicionamos de volta uma das cópias contadas duas vezes do ZZs, temos uma soma precisa para nosso subarray de destino. Então, apenas comparamos o resultado com nossa soma máxima atual e atualizamos o máximo, se necessário.

Quando tivermos passado por todas as coordenadas iniciais e finais possíveis, nosso valor máximo será a resposta, e simplesmente imprimiremos esse inteiro.

Importante Se o seu ambiente de programação começa as numerações em 0 (Python, JS, Java, C/C++), para implementar os algoritmos, é conveniente deixar uma linha zerada (a linha 0) e igualmente uma coluna zerada (a coluna 0). Agora as verificações passam a ser em referência à coluna e linha zeros e não a 1 como nos algoritmos. Aliás, no site acima referido o autor faz isso.

O Algoritmo final Veja como ficou uma possível solução

```
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    se i>1
      MAT[i][j]=MAT[i][j]+MAT[i-1][j]
    fimse
    se j>1
      MAT[i][j]=MAT[i][j]+MAT[i][j-1]
    fimse
    se (i>1) AND (j>1)
      MAT[i][j]=MAT[i][j]-MAT[i-1][j-1]
    fimse
  fimpara
fimpara
ans = 0
para i de 1 a N
  para j de 1 a N
    para k de i a N // note o i
      para L de j a N // idem j
        cur = MAT[k;L]
        se i > 1
          cur=cur-MAT[i-1][L]
        fimse
        se j>1
          cur=cur-MAT[i][j-1]
        fimse
        se (i>1) AND (j>1)
          cur=cur+MAT[i-1][j-1]
        fimse
        se and<cur
          ans=cur
          salva i,j,k,L
        fimse
      fimpara
    fimpara
  fimpara
fimpara
imprima (ans,salva)
```

Se você comparar os 2 algoritmos acima, verá que são muito parecidos. Só que o primeiro (força bruta) tem 6 *paras* um dentro do outro o que caracteriza o desempenho $O(N^6)$. Já o segundo tem os mesmos 6 *paras*, mas primeiro há um dentro do outro ($O(N^2)$) e depois, há 4 *paras*. Da teoria, ao final os 2 *paras* superiores podem ser desconsiderados e o resultado final fica só com $O(N^4)$.

Um exemplo completo Para você testar seu programa. A matriz

```
-41 -98  91  71 -42   8   3 -79 -17  16
 76 -11  46  74  44  61  42  49 -96  78
  5  -7 -86  43  -2  34  37 -60  84  74
 79   9 -72  -9  98 -56 -10 -36   3  77
-12  -6  62 -26 -57 100 -69  27  24 -99
-99  55  46 -36 -16  37  37 -58  68  42
 66 -81 -83  53  26 -57 -57 -83 -22  91
 90 -22 -46  39 -43  56  57 -15 -43 -61
-97 -61  97 -51  64 -72 -20  21 -64  66
-68  98 -48 -53 -79 -56  27  40  59  40
```

Deu como resposta 588 2, 1, 6, 10.

☞ Para você fazer

Calcule a máxima subsoma da matriz 10×10 abaixo e determine também as dimensões e posição da submatriz. Para tanto identifique o canto superior esquerdo e inferior direito da submatriz (origem=1). Os números estão entre $-600 < n \leq 600$.

```
-429   15   17 -513  255   11  144  319 -455  167
-213  351  291  292 -160  124  162   20  -26   55
-541  -99 -243 -520  -63 -376   66 -125   79   86
-546 -343 -533  213  123 -241 -593 -567  141 -293
-162 -289  331 -420   92 -484 -159 -260   81  -17
 369   27 -465  287  239  130  -61 -315 -225 -450
   5  382  123  170  267 -393   17  254  375  -36
-201 -239  -41 -522  -66  -18   93 -133 -454 -447
-425 -298  143  154 -514 -309 -309 -379  362  115
  69  -24 -589  137   67  219 -279  -88 -157   39
```

Responda aqui:

soma	localização

506-75608 -

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 21/10/2024 - 12:06:52.9
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
vivxhb2a V: 1.02
75615 IAN DOLABELLA OHATA
24JOG506 - 6 _____ / _____ / _____

UVA 108 - Maior subsoma em matriz

Observação: este texto está baseado em https://www.redgreencode.com/three-ways-solve-uva-108/ do autor Duncan Smith.

O problema original - 108

Como está lá, já traduzido:
Um problema que é simples de resolver numa dimensão é muitas vezes muito mais difícil de resolver em mais de uma dimensão. Considere satisfazer uma expressão booleana na forma normal conjuntiva na qual cada conjunto consiste em exatamente 3 disjuntos. Este problema (3-SAT) é NP-completo. O problema 2-SAT é resolvido de forma bastante eficiente, no entanto. Em contraste, alguns problemas pertencem à mesma classe de complexidade independentemente da dimensionalidade do problema. Dada uma matriz bidimensional de inteiros positivos e negativos, encontre o subretângulo com o maior soma. A soma de um retângulo é a soma de todos os elementos desse retângulo. Neste problema o subretângulo com a maior soma é chamado de subretângulo máximo. Um subretângulo é qualquer submatriz contígua de tamanho 1 x 1 ou maior localizado sobre a matriz. Como exemplo o sub-retângulo máximo do array

```
0   -2   -7   0
9    2   -6   2
-4   1   -4   1
-1   8    0  -2
```

está no canto esquerdo inferior

```
9   2
-4  1
-1  8
```

cuja soma é 15

Entrada:
A entrada consiste em uma matriz N × N de inteiros. A entrada começa com um único inteiro positivo N em uma linha isolada indicando o tamanho do quadrado da matriz bidimensional. Isto é seguido por números inteiros N^2 separados por espaços em branco (novas linhas e espaços). Esses inteiros N^2 constituem a matriz em ordem de linha maior (ou seja, todos os números na primeira linha, da esquerda para a direita, depois, todos os números da segunda linha, da esquerda para a direita, etc.). N pode ser tão grande quanto 100. Os números no array estarão no intervalo $[-127, 127]$.

Saída:
A saída é a soma do sub-retângulo máximo.

Exemplo de entrada

```
4
0 -2 -7 0 9 -2 -6 2
-4 1 -4 1 -1
8 0 -2
```

Exemplo de saída
15

Pesquisa completa

A abordagem de busca completa para este problema requer que enumeremos todos os subarrays do array de entrada e calculemos a soma de cada um separadamente. Para enumerar os subarrays de uma matriz, considere que você pode identificar cada submatriz por um conjunto de quatro inteiros 1 e N: linha inicial, coluna inicial, linha final e coluna final. Você pode pensar nesses quatro inteiros como as coordenadas dos cantos superior esquerdo e inferior direito do subarray. Tendo identificado um subarray usando essa abordagem de busca completa, a maneira ingênua de encontrar sua soma é adicionar todos os seus números juntos.

Para gerar todas as coordenadas do subarray, podemos usar quatro loops aninhados, um para cada coordenada. Outros dois loops aninhados são suficientes para iterar pelas linhas e colunas do subarray e calcular sua soma. Esses seis loops aninhados nos dão uma algoritmo. Embora o limite de tempo UVa, $O(N^6)$, seja bastante frouxo para esse problema, ele não é tão tolerante. Então, precisamos de uma abordagem mais eficiente. Veja uma possível implementação do algoritmo näive

```
maximo=-999999
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    para k de i a N
      para L de 1 a N
        soma=0
        para a de i a k
          para b de j a L
            soma=soma+MAT[a][b]
          fimpara
        fimpara
        se soma>maximo
          guarda=i,j,k,L
          maximo=soma
        fimse
      fimpara
    fimpara
  fimpara
fimpara
```

Estratégia de programação dinâmica

Com a reutilização de somas de subarrays. Para problemas básicos de programação dinâmica, como UVA 108, o segredo é descobrir quais informações você pode armazenar em etapas anteriores que economizarão tempo em etapas subsequentes.

A ideia-chave para UVa 108 é que os resultados de um cálculo anterior (uma soma, neste caso) podem ser usados em cálculos subsequentes. Especificamente, uma vez que calculamos a soma de parte de um subarray, não precisamos calculá-la repetidamente quando estamos processando outros subarrays que se sobrepõem à mesma região. **Se pudermos encontrar uma maneira eficiente de armazenar e reutilizar essas somas, podemos economizar muito tempo de cálculo**. A estrutura de dados usada em programação dinâmica para esse propósito é tradicionalmente chamada de tabela de memorando .

A tabela de memorandos tem o mesmo tamanho, formato e tipo que a matriz de entrada: a matriz de inteiros. O valor inteiro na posição (i, j) na tabela de memorandos armazena a soma da submatriz cujo canto superior esquerdo é o canto superior esquerdo da matriz de entrada e cujo canto inferior direito é (i, j) na matriz de entrada. Por exemplo, posição da tabela de memorandos $(2, 2)$ armazenaria a soma do subarray quadrado de $(1, 1)$ até $(2, 2)$ em nosso esquema de numeração baseado em 1.

Etapa 1: Gerar a tabela de memorandos
A chave para gerar a tabela de memorandos de forma eficiente é iterar pela matriz de entrada linha por linha e usar resultados anteriores para calcular os subsequentes. Você está basicamente usando a tabela de memorandos para gerar a si mesma. Para você acompanhar, eis como ficaria a tabela de memorandos do exemplo acima citado

```
0   -2    5    5
9    9   10   12
5    6    3    6
4   13   10   11
```

Para ver como isso funciona, considere a $4, 4$ matriz tabela de memorando mostrada abaixo. Os números de 01 para 16 representam a ordem na qual iremos iterar pelos dezesseis valores de entrada e calcular os dezesseis valores da tabela de memorando. Preencher a tabela dessa forma simplificará o código. Podemos ler a entrada e preencher a tabela de memorando simultaneamente. Portanto, não há necessidade de armazenar uma matriz separada de valores de entrada. Todas as informações de que precisamos na próxima etapa estarão na tabela de memorando.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Veja como o processo funciona: Escolha uma célula na matriz. Como exemplo, usarei a célula 10. Isso representa o canto inferior direito de um sub-array de 6 células que começa na célula 1 (três linhas e duas colunas). De acordo com nossa definição de tabela de memorando, a célula 10 deve conter a soma dos valores de entrada 01, 02, 05, 06, 09, e 10. Mas lembre-se de que estamos preenchendo a tabela de memorando em ordem. Então, quando chegamos à célula 10, já temos valores corretos para células 01 até 09. Como podemos usar esses valores?

A resposta é que podemos usar três informações para calcular cada soma de submatriz:

- A célula única que forma seu canto inferior direito. Neste caso, célula 10.
- O subarray acima dele. Neste caso, é o subarray $2, 2$ começando em 01 e terminando em 06. A soma deste subarray é armazenada na célula 06.
- O subarray à esquerda dele. Neste caso, é o subarray $3, 1$ começando em 01 e terminando em 09. A soma deste subarray é armazenada na célula 09.

Então, em vez de adicionar dez células para obter a soma que estamos procurando, podemos simplesmente adicionar três células: 06, 09, e 10. E para submatrizes maiores, a economia de tempo é maior.

Se você estiver acompanhando, notará que esse processo não está muito certo. O subarray acima e o subarray esquerdo 01 se sobrepõem nas células 01 e 05. Então, temos que subtrair a soma dessas células para evitar contagem dupla. Felizmente, a quantidade que precisamos subtrair é armazenada em uma única célula, a 05. Essa quantidade contada duas vezes sempre será uma célula diagonalmente para cima e para a esquerda da nossa célula-alvo.

Agora estamos no final do Passo 1, então nossa tabela de memorando contém as somas de todos os subarrays que começam no canto superior esquerdo do array de entrada. Mas o subarray de soma máxima pode começar em uma posição diferente. Para lidar com o caso geral, precisamos do Passo 2.

Etapa 2: Avalie todas as posições iniciais e finais
Esta etapa compartilha algumas ideias com a abordagem da Busca Completa descrita acima. A diferença é que as somas do subarray já foram calculadas eficientemente na Etapa 1. Então, em vez de um algoritmo de (Busca Completa) $O(N^6)$, temos um $O(N^2)$ passo (Passo 1) seguido por um passo $O(N^4)$ (Etapa 2). Como você deve saber pelas regras da análise de algoritmos assintóticos, isso nos dá uma visão geral de tempo de execução $O(N^4)$, o que é suficiente para que esta solução seja aceita na UVa OJ.

Assim como na Busca Completa, a Etapa 2 usa quatro loops aninhados, um para cada linha inicial, coluna inicial, linha final e coluna final. Essas quatro coordenadas definem o subarray em questão. Temos que verificar cada subarray para ver se ele tem a soma máxima vista até agora.

Veja o que acontece dentro do loop mais interno:

Primeiro, recuperamos a soma memorizada que calculamos na Etapa 1 para o subarray que termina na linha e coluna atuais. Lembre-se de que isso representa a soma do subarray que começa no canto superior esquerdo da entrada. Como isso pode não ser o mesmo que o canto superior esquerdo do subarray que estamos verificando no momento, temos que ajustar a soma usando um processo semelhante ao que usamos na Etapa 1. Para ilustrar, suponha que temos a seguinte tabela de memorando concluída. Como antes, os números identificam a ordem em que as células foram calculadas.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Como exemplo, suponha que nossos loops aninhados atualmente identifiquem uma posição inicial da célula 11 (linha 3, coluna 3) e uma posição final da célula 16 (linha 4, coluna 4). Precisamos encontrar a soma do subarray $2, 2$. O valor memorizado na célula 16 nos dá a soma de todo o array, então temos que nos livrar das partes que não precisamos. Como na Etapa 1, podemos fazer isso usando os valores que estão acima e à esquerda do alvo.

Aqui está outra maneira de analisar a tabela de memorandos:

```
ZZ ZZ XX XX
ZZ ZZ XX XX
```

```
YY YY 11 12
YY YY 15 16
```

As células XX identificam o subarray que está acima do alvo. As células YY identificam o subarray que está à esquerda do alvo. E as células ZZ identificam o subarray que está acima e à esquerda (ou seja, diagonalmente para cima e à esquerda) do alvo.

Esta visualização deve ser familiar do Passo 1. Mas neste passo, precisamos subtrair em vez de adicionar, já que estamos removendo somas que não precisamos. Então, começando com a soma na célula de memorando 16, precisamos:

Subtraia o valor na célula memo 08. Isso remove a soma do subarray que está acima do alvo (os XXs e ZZs). Subtraia o valor na célula memo 14. Isso remove a soma do subarray que está à esquerda do alvo (os YYs e ZZs). Observe que subtraímos o ZZs duas vezes. Nós os contamos duas vezes, assim como contamos duas vezes o valor diagonal no Passo 1. Portanto, precisamos que adicione novamente o valor na célula de memorando 06, os ZZs. Depois que adicionamos de volta uma das cópias contadas duas vezes do ZZs, temos uma soma precisa para nosso subarray de destino. Então, apenas comparamos o resultado com nossa soma máxima atual e atualizamos o máximo, se necessário.

Quando tivermos passado por todas as coordenadas iniciais e finais possíveis, nosso valor máximo será a resposta, e simplesmente imprimiremos esse inteiro.

Importante Se o seu ambiente de programação começa as numerações em 0 (Python, JS, Java, C/C++), para implementar os algoritmos, é conveniente deixar uma linha zerada (a linha 0) e igualmente uma coluna zerada (a coluna 0). Agora as verificações passam a ser em referência à coluna e linha zeros e não a 1 como nos algoritmos. Aliás, no site acima referido o autor faz isso.

O Algoritmo final

Veja como ficou uma possível solução

```
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    se i>1
      MAT[i][j]=MAT[i][j]+MAT[i-1][j]
    fimse
    se j>1
      MAT[i][j]=MAT[i][j]+MAT[i][j-1]
    fimse
    se (i>1) AND (j>1)
      MAT[i][j]=MAT[i][j]-MAT[i-1][j-1]
    fimse
  fimpara
fimpara
ans = 0
para i de 1 a N
  para j de 1 a N
    para k de i a N // note o i
      para L de j a N // idem j
        cur = MAT[k;L]
        se i > 1
          cur=cur-MAT[i-1][L]
        fimse
        se j>1
          cur=cur-MAT[i][j-1]
        fimse
        se (i>1) AND (j>1)
          cur=cur+MAT[i-1][j-1]
        fimse
        se and<cur
          ans=cur
          salva i,j,k,L
        fimse
      fimpara
    fimpara
  fimpara
fimpara
imprima (ans,salva)
```

Se você comparar os 2 algoritmos acima, verá que são muito parecidos. Só que o primeiro (força bruta) tem 6 *paras* um dentro do outro o que caracteriza o desempenho $O(N^6)$. Já o segundo tem os mesmos 6 *paras*, mas primeiro há um dentro do outro ($O(N^2)$) e depois, há 4 *paras*. Da teoria, ao final os 2 *paras* superiores podem ser desconsiderados e o resultado final fica só com $O(N^4)$.

Um exemplo completo

Para você testar seu programa. A matriz

```
-41 -98  91  71 -42   8   3 -79 -17  16
 76 -11  46  74  44  61  42  49 -96  78
  5  -7 -86  43  -2  34  37 -60  84  74
 79   9 -72  -9  98 -56 -10 -36   3  77
-12  -6  62 -26 -57 100 -69  27  24 -99
-99  55  46 -36 -16  37  37 -58  68  42
 66 -81 -83  53  26 -57 -57 -83 -22  91
 90 -22 -46  39 -43  56  57 -15 -43 -61
-97 -61  97 -51  64 -72 -20  21 -64  66
-68  98 -48 -53 -79 -56  27  40  59  40
```

Deu como resposta 588 2, 1, 6, 10.

☞ Para você fazer

Calcule a máxima subsoma da matriz 10×10 abaixo e determine também as dimensões e posição da submatriz. Para tanto identifique o canto superior esquerdo e inferior direito da submatriz (origem=1). Os números estão entre $-600 < n \leq 600$.

```
-315 -279  -78  -89   23 -425  212   17 -579  162
-116 -119  -24 -456 -541  178  366 -523  -10  109
 -80 -499 -461 -573   58 -536  -17 -192  216   27
 339  340  175   20  126  186  341 -373  314  337
-568  367 -501  274 -476  -57 -103  -52  262 -485
-393  305   89  104  152 -317 -334 -148  302  -95
  11 -351  120 -216 -399 -525 -541  -32 -376  334
-480  385 -251  264 -107  -92  -22 -211  291 -500
-451 -130  168 -357  216   13 -260  -78 -368  133
-490 -501 -322 -516 -228 -418 -596  -62 -487 -401
```

Responda aqui:

soma	localização

506-75615 -

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
vivxhb2a V: 1.02
75622 JOAO BONFIN LINO
24JOG506 - 7 _____ / _____ / _____

21/10/2024 - 12:06:52.9

UVA 108 - Maior subsoma em matriz

Observação: este texto está baseado em https://www.redgreencode.com/three-ways-solve-uva-108/ do autor Duncan Smith.

O problema original - 108

Como está lá, já traduzido:
Um problema que é simples de resolver numa dimensão é muitas vezes muito mais difícil de resolver em mais de uma dimensão. Considere satisfazer uma expressão booleana na forma normal conjuntiva na qual cada conjunto consiste em exatamente 3 disjuntos. Este problema (3-SAT) é NP-completo. O problema 2-SAT é resolvido de forma bastante eficiente, no entanto. Em contraste, alguns problemas pertencem à mesma classe de complexidade independentemente da dimensionalidade do problema. Dada uma matriz bidimensional de inteiros positivos e negativos, encontre o subretângulo com o maior soma. A soma de um retângulo é a soma de todos os elementos desse retângulo. Neste problema o subretângulo com a maior soma é chamado de subretângulo máximo. Um subretângulo é qualquer submatriz contígua de tamanho 1 x 1 ou maior localizado sobre a matriz. Como exemplo o sub-retângulo máximo do array

```
0   -2   -7   0
9    2   -6   2
-4   1   -4   1
-1   8    0  -2
```

está no canto esquerdo inferior

```
9   2
-4  1
-1  8
```

cuja soma é 15

Entrada:
A entrada consiste em uma matriz N × N de inteiros. A entrada começa com um único inteiro positivo N em uma linha isolada indicando o tamanho do quadrado da matriz bidimensional. Isto é seguido por números inteiros N^2 separados por espaços em branco (novas linhas e espaços). Esses inteiros N^2 constituem a matriz em ordem de linha maior (ou seja, todos os números na primeira linha, da esquerda para a direita, depois, todos os números da segunda linha, da esquerda para a direita, etc.). N pode ser tão grande quanto 100. Os números no array estarão no intervalo $[-127, 127]$.

Saída:
A saída é a soma do sub-retângulo máximo.

Exemplo de entrada

```
4
0 -2 -7 0 9 -2 -6 2
-4 1 -4 1 -1
8 0 -2
```

Exemplo de saída
15

Pesquisa completa

A abordagem de busca completa para este problema requer que enumeremos todos os subarrays do array de entrada e calculemos a soma de cada um separadamente. Para enumerar os subarrays de uma matriz, considere que você pode identificar cada submatriz por um conjunto de quatro inteiros 1 e N: linha inicial, coluna inicial, linha final e coluna final. Você pode pensar nesses quatro inteiros como as coordenadas dos cantos superior esquerdo e inferior direito do subarray. Tendo identificado um subarray usando essa abordagem de busca completa, a maneira ingênua de encontrar sua soma é adicionar todos os seus números juntos.

Para gerar todas as coordenadas do subarray, podemos usar quatro loops aninhados, um para cada coordenada. Outros dois loops aninhados são suficientes para iterar pelas linhas e colunas do subarray e calcular sua soma. Esses seis loops aninhados nos dão uma algoritmo. Embora o limite de tempo UVa, $O(N^6)$, seja bastante frouxo para esse problema, ele não é tão tolerante. Então, precisamos de uma abordagem mais eficiente. Veja uma possível implementação do algoritmo näive

```
maximo=-999999
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    para k de i a N
      para L de 1 a N
        soma=0
        para a de i a k
          para b de j a L
            soma=soma+MAT[a][b]
          fimpara
        fimpara
        se soma>maximo
          guarda=i,j,k,L
          maximo=soma
        fimse
      fimpara
    fimpara
  fimpara
fimpara
```

Estratégia de programação dinâmica

Com a reutilização de somas de subarrays. Para problemas básicos de programação dinâmica, como UVA 108, o segredo é descobrir quais informações você pode armazenar em etapas anteriores que economizarão tempo em etapas subsequentes.

A ideia-chave para UVa 108 é que os resultados de um cálculo anterior (uma soma, neste caso) podem ser usados em cálculos subsequentes. Especificamente, uma vez que calculamos a soma de parte de um subarray, não precisamos calculá-la repetidamente quando estamos processando outros subarrays que se sobrepõem à mesma região. **Se pudermos encontrar uma maneira eficiente de armazenar e reutilizar essas somas, podemos economizar muito tempo de cálculo**. A estrutura de dados usada em programação dinâmica para esse propósito é tradicionalmente chamada de tabela de memorando .

A tabela de memorandos tem o mesmo tamanho, formato e tipo que a matriz de entrada: a matriz de inteiros. O valor inteiro na posição (i, j) na tabela de memorandos armazena a soma da submatriz cujo canto superior esquerdo é o canto superior esquerdo da matriz de entrada e cujo canto inferior direito é (i, j) na matriz de entrada. Por exemplo, posição da tabela de memorandos $(2, 2)$ armazenaria a soma do subarray quadrado de $(1, 1)$ até $(2, 2)$ em nosso esquema de numeração baseado em 1.

Etapa 1: Gerar a tabela de memorandos
A chave para gerar a tabela de memorandos de forma eficiente é iterar pela matriz de entrada linha por linha e usar resultados anteriores para calcular os subsequentes. Você está basicamente usando a tabela de memorandos para gerar a si mesma. Para você acompanhar, eis como ficaria a tabela de memorandos do exemplo acima citado

```
0   -2    5    5
9    9   10   12
5    6    3    6
4   13   10   11
```

Para ver como isso funciona, considere a $4, 4$ matriz tabela de memorando mostrada abaixo. Os números de 01 para 16 representam a ordem na qual iremos iterar pelos dezesseis valores de entrada e calcular os dezesseis valores da tabela de memorando. Preencher a tabela dessa forma simplificará o código. Podemos ler a entrada e preencher a tabela de memorando simultaneamente. Portanto, não há necessidade de armazenar uma matriz separada de valores de entrada. Todas as informações de que precisamos na próxima etapa estarão na tabela de memorando.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Veja como o processo funciona: Escolha uma célula na matriz. Como exemplo, usarei a célula 10. Isso representa o canto inferior direito de um sub-array de 6 células que começa na célula 1 (três linhas e duas colunas). De acordo com nossa definição de tabela de memorando, a célula 10 deve conter a soma dos valores de entrada 01, 02, 05, 06, 09, e 10. Mas lembre-se de que estamos preenchendo a tabela de memorando em ordem. Então, quando chegamos à célula 10, já temos valores corretos para células 01 até 09. Como podemos usar esses valores?

A resposta é que podemos usar três informações para calcular cada soma de submatriz:

- A célula única que forma seu canto inferior direito. Neste caso, célula 10.
- O subarray acima dele. Neste caso, é o subarray $2, 2$ começando em 01 e terminando em 06. A soma deste subarray é armazenada na célula 06.
- O subarray à esquerda dele. Neste caso, é o subarray $3, 1$ começando em 01 e terminando em 09. A soma deste subarray é armazenada na célula 09.

Então, em vez de adicionar dez células para obter a soma que estamos procurando, podemos simplesmente adicionar três células: 06, 09, e 10. E para submatrizes maiores, a economia de tempo é maior.

Se você estiver acompanhando, notará que esse processo não está muito certo. O subarray acima e o subarray esquerdo 01 se sobrepõem nas células 01 e 05. Então, temos que subtrair a soma dessas células para evitar contagem dupla. Felizmente, a quantidade que precisamos subtrair é armazenada em uma única célula, a 05. Essa quantidade contada duas vezes sempre será uma célula diagonalmente para cima e para a esquerda da nossa célula-alvo.

Agora estamos no final do Passo 1, então nossa tabela de memorando contém as somas de todos os subarrays que começam no canto superior esquerdo do array de entrada. Mas o subarray de soma máxima pode começar em uma posição diferente. Para lidar com o caso geral, precisamos do Passo 2.

Etapa 2: Avalie todas as posições iniciais e finais
Esta etapa compartilha algumas ideias com a abordagem da Busca Completa descrita acima. A diferença é que as somas do subarray já foram calculadas eficientemente na Etapa 1. Então, em vez de um algoritmo de (Busca Completa) $O(N^6)$, temos um $O(N^2)$ passo (Passo 1) seguido por um passo $O(N^4)$ (Etapa 2). Como você deve saber pelas regras da análise de algoritmos assintóticos, isso nos dá uma visão geral de tempo de execução $O(N^4)$, o que é suficiente para que esta solução seja aceita na UVa OJ.

Assim como na Busca Completa, a Etapa 2 usa quatro loops aninhados, um para cada linha inicial, coluna inicial, linha final e coluna final. Essas quatro coordenadas definem o subarray em questão. Temos que verificar cada subarray para ver se ele tem a soma máxima vista até agora.

Veja o que acontece dentro do loop mais interno:

Primeiro, recuperamos a soma memorizada que calculamos na Etapa 1 para o subarray que termina na linha e coluna atuais. Lembre-se de que isso representa a soma do subarray que começa no canto superior esquerdo da entrada. Como isso pode não ser o mesmo que o canto superior esquerdo do subarray que estamos verificando no momento, temos que ajustar a soma usando um processo semelhante ao que usamos na Etapa 1. Para ilustrar, suponha que temos a seguinte tabela de memorando concluída. Como antes, os números identificam a ordem em que as células foram calculadas.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Como exemplo, suponha que nossos loops aninhados atualmente identifiquem uma posição inicial da célula 11 (linha 3, coluna 3) e uma posição final da célula 16 (linha 4, coluna 4). Precisamos encontrar a soma do subarray $2, 2$. O valor memorizado na célula 16 nos dá a soma de todo o array, então temos que nos livrar das partes que não precisamos. Como na Etapa 1, podemos fazer isso usando os valores que estão acima e à esquerda do alvo.

Aqui está outra maneira de analisar a tabela de memorandos:

```
ZZ ZZ XX XX
ZZ ZZ XX XX
```

```
YY YY 11 12
YY YY 15 16
```

As células XX identificam o subarray que está acima do alvo. As células YY identificam o subarray que está à esquerda do alvo. E as células ZZ identificam o subarray que está acima e à esquerda (ou seja, diagonalmente para cima e à esquerda) do alvo.

Esta visualização deve ser familiar do Passo 1. Mas neste passo, precisamos subtrair em vez de adicionar, já que estamos removendo somas que não precisamos. Então, começando com a soma na célula de memorando 16, precisamos:

Subtraia o valor na célula memo 08. Isso remove a soma do subarray que está acima do alvo (os XXs e ZZs). Subtraia o valor na célula memo 14. Isso remove a soma do subarray que está à esquerda do alvo (os YYs e ZZs). Observe que subtraímos o ZZs duas vezes. Nós os contamos duas vezes, assim como contamos duas vezes o valor diagonal no Passo 1. Portanto, precisamos que adicione novamente o valor na célula de memorando 06, os ZZs. Depois que adicionamos de volta uma das cópias contadas duas vezes do ZZs, temos uma soma precisa para nosso subarray de destino. Então, apenas comparamos o resultado com nossa soma máxima atual e atualizamos o máximo, se necessário.

Quando tivermos passado por todas as coordenadas iniciais e finais possíveis, nosso valor máximo será a resposta, e simplesmente imprimiremos esse inteiro.

Importante Se o seu ambiente de programação começa as numerações em 0 (Python, JS, Java, C/C++), para implementar os algoritmos, é conveniente deixar uma linha zerada (a linha 0) e igualmente uma coluna zerada (a coluna 0). Agora as verificações passam a ser em referência à coluna e linha zeros e não a 1 como nos algoritmos. Aliás, no site acima referido o autor faz isso.

O Algoritmo final

Veja como ficou uma possível solução

```
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    se i>1
      MAT[i][j]=MAT[i][j]+MAT[i-1][j]
    fimse
    se j>1
      MAT[i][j]=MAT[i][j]+MAT[i][j-1]
    fimse
    se (i>1) AND (j>1)
      MAT[i][j]=MAT[i][j]-MAT[i-1][j-1]
    fimse
  fimpara
fimpara
ans = 0
para i de 1 a N
  para j de 1 a N
    para k de i a N // note o i
      para L de j a N // idem j
        cur = MAT[k;L]
        se i > 1
          cur=cur-MAT[i-1][L]
        fimse
        se j>1
          cur=cur-MAT[i][j-1]
        fimse
        se (i>1) AND (j>1)
          cur=cur+MAT[i-1][j-1]
        fimse
        se and<cur
          ans=cur
          salva i,j,k,L
        fimse
      fimpara
    fimpara
  fimpara
fimpara
imprima (ans,salva)
```

Se você comparar os 2 algoritmos acima, verá que são muito parecidos. Só que o primeiro (força bruta) tem 6 *paras* um dentro do outro o que caracteriza o desempenho $O(N^6)$. Já o segundo tem os mesmos 6 *paras*, mas primeiro há um dentro do outro ($O(N^2)$) e depois, há 4 *paras*. Da teoria, ao final os 2 *paras* superiores podem ser desconsiderados e o resultado final fica só com $O(N^4)$.

Um exemplo completo

Para você testar seu programa. A matriz

```
-41 -98  91  71 -42   8   3 -79 -17  16
 76 -11  46  74  44  61  42  49 -96  78
  5  -7 -86  43  -2  34  37 -60  84  74
 79   9 -72  -9  98 -56 -10 -36   3  77
-12  -6  62 -26 -57 100 -69  27  24 -99
-99  55  46 -36 -16  37  37 -58  68  42
 66 -81 -83  53  26 -57 -57 -83 -22  91
 90 -22 -46  39 -43  56  57 -15 -43 -61
-97 -61  97 -51  64 -72 -20  21 -64  66
-68  98 -48 -53 -79 -56  27  40  59  40
```

Deu como resposta 588 2, 1, 6, 10.

☞ Para você fazer

Calcule a máxima subsoma da matriz 10×10 abaixo e determine também as dimensões e posição da submatriz. Para tanto identifique o canto superior esquerdo e inferior direito da submatriz (origem=1). Os números estão entre $-600 < n \leq 600$.

```
 -77 -567  -23 -197  -27 -439 -355  -25 -479 -376
-199  -10  153   90  254  238  106   37 -376 -450
 123  241 -516   27 -299 -246  183 -248 -347  -39
-489  236 -258 -585 -124 -167  -11  309 -575  105
  76  -77   19  252   41  -82 -293 -304 -143 -287
 -13 -178 -232 -522 -185  350   56 -227 -538  151
-370   -9  206  -20 -450  192 -549 -589    5 -255
-330 -456 -301  342 -479 -488 -184 -459  -99 -552
 -63 -511  313  140    1 -569 -562  196 -333  388
  -6  306 -251 -423  225  385  -81   51  396   48
```

Responda aqui:

soma	localização

506-75622 -

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 21/10/2024 - 12:06:52.9
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
vivxhb2a V: 1.02
75639 JOAO LUCA FERNANDES DE OLIVE
24JOG506 - 8 _____ / _____ / _____

UVA 108 - Maior subsoma em matriz

Observação: este texto está baseado em https://www.redgreencode.com/three-ways-solve-uva-108/ do autor Duncan Smith.

O problema original - 108

Como está lá, já traduzido:
Um problema que é simples de resolver numa dimensão é muitas vezes muito mais difícil de resolver em mais de uma dimensão. Considere satisfazer uma expressão booleana na forma normal conjuntiva na qual cada conjunto consiste em exatamente 3 disjuntos. Este problema (3-SAT) é NP-completo. O problema 2-SAT é resolvido de forma bastante eficiente, no entanto. Em contraste, alguns problemas pertencem à mesma classe de complexidade independentemente da dimensionalidade do problema. Dada uma matriz bidimensional de inteiros positivos e negativos, encontre o subretângulo com o maior soma. A soma de um retângulo é a soma de todos os elementos desse retângulo. Neste problema o subretângulo com a maior soma é chamado de subretângulo máximo. Um subretângulo é qualquer submatriz contígua de tamanho 1 x 1 ou maior localizado sobre a matriz. Como exemplo o sub-retângulo máximo do array

```
0   -2   -7   0
9    2   -6   2
-4   1   -4   1
-1   8    0  -2
```

está no canto esquerdo inferior

```
9   2
-4  1
-1  8
```

cuja soma é 15

Entrada:
A entrada consiste em uma matriz N × N de inteiros. A entrada começa com um único inteiro positivo N em uma linha isolada indicando o tamanho do quadrado da matriz bidimensional. Isto é seguido por números inteiros N^2 separados por espaços em branco (novas linhas e espaços). Esses inteiros N^2 constituem a matriz em ordem de linha maior (ou seja, todos os números na primeira linha, da esquerda para a direita, depois, todos os números da segunda linha, da esquerda para a direita, etc.). N pode ser tão grande quanto 100. Os números no array estarão no intervalo $[-127, 127]$.

Saída:
A saída é a soma do sub-retângulo máximo.

Exemplo de entrada

```
4
0 -2 -7 0 9 -2 -6 2
-4 1 -4 1 -1
8 0 -2
```

Exemplo de saída
15

Pesquisa completa

A abordagem de busca completa para este problema requer que enumeremos todos os subarrays do array de entrada e calculemos a soma de cada um separadamente. Para enumerar os subarrays de uma matriz, considere que você pode identificar cada submatriz por um conjunto de quatro inteiros 1 e N: linha inicial, coluna inicial, linha final e coluna final. Você pode pensar nesses quatro inteiros como as coordenadas dos cantos superior esquerdo e inferior direito do subarray. Tendo identificado um subarray usando essa abordagem de busca completa, a maneira ingênua de encontrar sua soma é adicionar todos os seus números juntos.

Para gerar todas as coordenadas do subarray, podemos usar quatro loops aninhados, um para cada coordenada. Outros dois loops aninhados são suficientes para iterar pelas linhas e colunas do subarray e calcular sua soma. Esses seis loops aninhados nos dão uma algoritmo. Embora o limite de tempo UVa, $O(N^6)$, seja bastante frouxo para esse problema, ele não é tão tolerante. Então, precisamos de uma abordagem mais eficiente. Veja uma possível implementação do algoritmo näive

```
maximo=-999999
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    para k de i a N
      para L de 1 a N
        soma=0
        para a de i a k
          para b de j a L
            soma=soma+MAT[a][b]
          fimpara
        fimpara
        se soma>maximo
          guarda=i,j,k,L
          maximo=soma
        fimse
      fimpara
    fimpara
  fimpara
fimpara
```

Estratégia de programação dinâmica

Com a reutilização de somas de subarrays. Para problemas básicos de programação dinâmica, como UVA 108, o segredo é descobrir quais informações você pode armazenar em etapas anteriores que economizarão tempo em etapas subsequentes.

A ideia-chave para UVa 108 é que os resultados de um cálculo anterior (uma soma, neste caso) podem ser usados em cálculos subsequentes. Especificamente, uma vez que calculamos a soma de parte de um subarray, não precisamos calculá-la repetidamente quando estamos processando outros subarrays que se sobrepõem à mesma região. **Se pudermos encontrar uma maneira eficiente de armazenar e reutilizar essas somas, podemos economizar muito tempo de cálculo**. A estrutura de dados usada em programação dinâmica para esse propósito é tradicionalmente chamada de tabela de memorando .

A tabela de memorandos tem o mesmo tamanho, formato e tipo que a matriz de entrada: a matriz de inteiros. O valor inteiro na posição (i, j) na tabela de memorandos armazena a soma da submatriz cujo canto superior esquerdo é o canto superior esquerdo da matriz de entrada e cujo canto inferior direito é (i, j) na matriz de entrada. Por exemplo, posição da tabela de memorandos $(2, 2)$ armazenaria a soma do subarray quadrado de $(1, 1)$ até $(2, 2)$ em nosso esquema de numeração baseado em 1.

Etapa 1: Gerar a tabela de memorandos
A chave para gerar a tabela de memorandos de forma eficiente é iterar pela matriz de entrada linha por linha e usar resultados anteriores para calcular os subsequentes. Você está basicamente usando a tabela de memorandos para gerar a si mesma. Para você acompanhar, eis como ficaria a tabela de memorandos do exemplo acima citado

```
0   -2    5    5
9    9   10   12
5    6    3    6
4   13   10   11
```

Para ver como isso funciona, considere a $4, 4$ matriz tabela de memorando mostrada abaixo. Os números de 01 para 16 representam a ordem na qual iremos iterar pelos dezesseis valores de entrada e calcular os dezesseis valores da tabela de memorando. Preencher a tabela dessa forma simplificará o código. Podemos ler a entrada e preencher a tabela de memorando simultaneamente. Portanto, não há necessidade de armazenar uma matriz separada de valores de entrada. Todas as informações de que precisamos na próxima etapa estarão na tabela de memorando.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Veja como o processo funciona: Escolha uma célula na matriz. Como exemplo, usarei a célula 10. Isso representa o canto inferior direito de um sub-array de 6 células que começa na célula 1 (três linhas e duas colunas). De acordo com nossa definição de tabela de memorando, a célula 10 deve conter a soma dos valores de entrada 01, 02, 05, 06, 09, e 10. Mas lembre-se de que estamos preenchendo a tabela de memorando em ordem. Então, quando chegamos à célula 10, já temos valores corretos para células 01 até 09. Como podemos usar esses valores?

A resposta é que podemos usar três informações para calcular cada soma de submatriz:

- A célula única que forma seu canto inferior direito. Neste caso, célula 10.
- O subarray acima dele. Neste caso, é o subarray $2, 2$ começando em 01 e terminando em 06. A soma deste subarray é armazenada na célula 06.
- O subarray à esquerda dele. Neste caso, é o subarray $3, 1$ começando em 01 e terminando em 09. A soma deste subarray é armazenada na célula 09.

Então, em vez de adicionar dez células para obter a soma que estamos procurando, podemos simplesmente adicionar três células: 06, 09, e 10. E para submatrizes maiores, a economia de tempo é maior.

Se você estiver acompanhando, notará que esse processo não está muito certo. O subarray acima e o subarray esquerdo 01 se sobrepõem nas células 01 e 05. Então, temos que subtrair a soma dessas células para evitar contagem dupla. Felizmente, a quantidade que precisamos subtrair é armazenada em uma única célula, a 05. Essa quantidade contada duas vezes sempre será uma célula diagonalmente para cima e para a esquerda da nossa célula-alvo.

Agora estamos no final do Passo 1, então nossa tabela de memorando contém as somas de todos os subarrays que começam no canto superior esquerdo do array de entrada. Mas o subarray de soma máxima pode começar em uma posição diferente. Para lidar com o caso geral, precisamos do Passo 2.

Etapa 2: Avalie todas as posições iniciais e finais
Esta etapa compartilha algumas ideias com a abordagem da Busca Completa descrita acima. A diferença é que as somas do subarray já foram calculadas eficientemente na Etapa 1. Então, em vez de um algoritmo de (Busca Completa) $O(N^6)$, temos um $O(N^2)$ passo (Passo 1) seguido por um passo $O(N^4)$ (Etapa 2). Como você deve saber pelas regras da análise de algoritmos assintóticos, isso nos dá uma visão geral de tempo de execução $O(N^4)$, o que é suficiente para que esta solução seja aceita na UVa OJ.

Assim como na Busca Completa, a Etapa 2 usa quatro loops aninhados, um para cada linha inicial, coluna inicial, linha final e coluna final. Essas quatro coordenadas definem o subarray em questão. Temos que verificar cada subarray para ver se ele tem a soma máxima vista até agora.

Veja o que acontece dentro do loop mais interno:

Primeiro, recuperamos a soma memorizada que calculamos na Etapa 1 para o subarray que termina na linha e coluna atuais. Lembre-se de que isso representa a soma do subarray que começa no canto superior esquerdo da entrada. Como isso pode não ser o mesmo que o canto superior esquerdo do subarray que estamos verificando no momento, temos que ajustar a soma usando um processo semelhante ao que usamos na Etapa 1. Para ilustrar, suponha que temos a seguinte tabela de memorando concluída. Como antes, os números identificam a ordem em que as células foram calculadas.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Como exemplo, suponha que nossos loops aninhados atualmente identifiquem uma posição inicial da célula 11 (linha 3, coluna 3) e uma posição final da célula 16 (linha 4, coluna 4). Precisamos encontrar a soma do subarray $2, 2$. O valor memorizado na célula 16 nos dá a soma de todo o array, então temos que nos livrar das partes que não precisamos. Como na Etapa 1, podemos fazer isso usando os valores que estão acima e à esquerda do alvo.

Aqui está outra maneira de analisar a tabela de memorandos:

```
ZZ ZZ XX XX
ZZ ZZ XX XX
```

```
YY YY 11 12
YY YY 15 16
```

As células XX identificam o subarray que está acima do alvo. As células YY identificam o subarray que está à esquerda do alvo. E as células ZZ identificam o subarray que está acima e à esquerda (ou seja, diagonalmente para cima e à esquerda) do alvo.

Esta visualização deve ser familiar do Passo 1. Mas neste passo, precisamos subtrair em vez de adicionar, já que estamos removendo somas que não precisamos. Então, começando com a soma na célula de memorando 16, precisamos:

Subtraia o valor na célula memo 08. Isso remove a soma do subarray que está acima do alvo (os XXs e ZZs). Subtraia o valor na célula memo 14. Isso remove a soma do subarray que está à esquerda do alvo (os YYs e ZZs). Observe que subtraímos o ZZs duas vezes. Nós os contamos duas vezes, assim como contamos duas vezes o valor diagonal no Passo 1. Portanto, precisamos que adicione novamente o valor na célula de memorando 06, os ZZs. Depois que adicionamos de volta uma das cópias contadas duas vezes do ZZs, temos uma soma precisa para nosso subarray de destino. Então, apenas comparamos o resultado com nossa soma máxima atual e atualizamos o máximo, se necessário.

Quando tivermos passado por todas as coordenadas iniciais e finais possíveis, nosso valor máximo será a resposta, e simplesmente imprimiremos esse inteiro.

Importante Se o seu ambiente de programação começa as numerações em 0 (Python, JS, Java, C/C++), para implementar os algoritmos, é conveniente deixar uma linha zerada (a linha 0) e igualmente uma coluna zerada (a coluna 0). Agora as verificações passam a ser em referência à coluna e linha zeros e não a 1 como nos algoritmos. Aliás, no site acima referido o autor faz isso.

O Algoritmo final

Veja como ficou uma possível solução

```
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    se i>1
      MAT[i][j]=MAT[i][j]+MAT[i-1][j]
    fimse
    se j>1
      MAT[i][j]=MAT[i][j]+MAT[i][j-1]
    fimse
    se (i>1) AND (j>1)
      MAT[i][j]=MAT[i][j]-MAT[i-1][j-1]
    fimse
  fimpara
fimpara
ans = 0
para i de 1 a N
  para j de 1 a N
    para k de i a N // note o i
      para L de j a N // idem j
        cur = MAT[k;L]
        se i > 1
          cur=cur-MAT[i-1][L]
        fimse
        se j>1
          cur=cur-MAT[i][j-1]
        fimse
        se (i>1) AND (j>1)
          cur=cur+MAT[i-1][j-1]
        fimse
        se and<cur
          ans=cur
          salva i,j,k,L
        fimse
      fimpara
    fimpara
  fimpara
fimpara
imprima (ans,salva)
```

Se você comparar os 2 algoritmos acima, verá que são muito parecidos. Só que o primeiro (força bruta) tem 6 *paras* um dentro do outro o que caracteriza o desempenho $O(N^6)$. Já o segundo tem os mesmos 6 *paras*, mas primeiro há um dentro do outro ($O(N^2)$) e depois, há 4 *paras*. Da teoria, ao final os 2 *paras* superiores podem ser desconsiderados e o resultado final fica só com $O(N^4)$.

Um exemplo completo

Para você testar seu programa. A matriz

```
-41 -98  91  71 -42   8   3 -79 -17  16
 76 -11  46  74  44  61  42  49 -96  78
  5  -7 -86  43  -2  34  37 -60  84  74
 79   9 -72  -9  98 -56 -10 -36   3  77
-12  -6  62 -26 -57 100 -69  27  24 -99
-99  55  46 -36 -16  37  37 -58  68  42
 66 -81 -83  53  26 -57 -57 -83 -22  91
 90 -22 -46  39 -43  56  57 -15 -43 -61
-97 -61  97 -51  64 -72 -20  21 -64  66
-68  98 -48 -53 -79 -56  27  40  59  40
```

Deu como resposta 588 2, 1, 6, 10.

☞ Para você fazer

Calcule a máxima subsoma da matriz 10×10 abaixo e determine também as dimensões e posição da submatriz. Para tanto identifique o canto superior esquerdo e inferior direito da submatriz (origem=1). Os números estão entre $-600 < n \leq 600$.

```
-363  206   94  291 -148  -29   80 -449  395 -261
-158   -3  158  216 -129  285 -346 -580 -288  -27
-396 -561  -90 -342   45 -155 -588  284  391  -76
-227 -449  163   62  -56  214 -203   10 -130  234
-303   36  165 -400 -402 -232  205 -573 -583  266
-271  113 -280 -581 -394  333   33 -217  -41 -487
-153 -469 -286  276 -354   25   75 -191  235 -149
 190  -60  -89 -168 -119 -126  116  373 -140 -251
 -66 -104 -439  175  242  390 -323 -201 -121 -345
 146 -203 -596  -71  -46 -320  387  230 -346  325
```

Responda aqui:

soma	localização

506-75639 -

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 21/10/2024 - 12:06:52.9
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
vivxhb2a V: 1.02
75646 JOAO PEDRO DE BRITO DUARTE
24JOG506 - 9 _____ / _____ / _____

UVA 108 - Maior subsoma em matriz

Observação: este texto está baseado em https://www.redgreencode.com/three-ways-solve-uva-108/ do autor Duncan Smith.

O problema original - 108

Como está lá, já traduzido:
Um problema que é simples de resolver numa dimensão é muitas vezes muito mais difícil de resolver em mais de uma dimensão. Considere satisfazer uma expressão booleana na forma normal conjuntiva na qual cada conjunto consiste em exatamente 3 disjuntos. Este problema (3-SAT) é NP-completo. O problema 2-SAT é resolvido de forma bastante eficiente, no entanto. Em contraste, alguns problemas pertencem à mesma classe de complexidade independentemente da dimensionalidade do problema. Dada uma matriz bidimensional de inteiros positivos e negativos, encontre o subretângulo com o maior soma. A soma de um retângulo é a soma de todos os elementos desse retângulo. Neste problema o subretângulo com a maior soma é chamado de subretângulo máximo. Um subretângulo é qualquer submatriz contígua de tamanho 1 x 1 ou maior localizado sobre a matriz. Como exemplo o sub-retângulo máximo do array

```
0   -2   -7   0
9    2   -6   2
-4   1   -4   1
-1   8    0  -2
```

está no canto esquerdo inferior

```
9   2
-4  1
-1  8
```

cuja soma é 15

Entrada:
A entrada consiste em uma matriz N × N de inteiros. A entrada começa com um único inteiro positivo N em uma linha isolada indicando o tamanho do quadrado da matriz bidimensional. Isto é seguido por números inteiros N^2 separados por espaços em branco (novas linhas e espaços). Esses inteiros N^2 constituem a matriz em ordem de linha maior (ou seja, todos os números na primeira linha, da esquerda para a direita, depois, todos os números da segunda linha, da esquerda para a direita, etc.). N pode ser tão grande quanto 100. Os números no array estarão no intervalo $[-127, 127]$.

Saída:
A saída é a soma do sub-retângulo máximo.

Exemplo de entrada

```
4
0 -2 -7 0 9 -2 -6 2
-4 1 -4 1 -1
8 0 -2
```

Exemplo de saída
15

Pesquisa completa

A abordagem de busca completa para este problema requer que enumeremos todos os subarrays do array de entrada e calculemos a soma de cada um separadamente. Para enumerar os subarrays de uma matriz, considere que você pode identificar cada submatriz por um conjunto de quatro inteiros 1 e N: linha inicial, coluna inicial, linha final e coluna final. Você pode pensar nesses quatro inteiros como as coordenadas dos cantos superior esquerdo e inferior direito do subarray. Tendo identificado um subarray usando essa abordagem de busca completa, a maneira ingênua de encontrar sua soma é adicionar todos os seus números juntos.

Para gerar todas as coordenadas do subarray, podemos usar quatro loops aninhados, um para cada coordenada. Outros dois loops aninhados são suficientes para iterar pelas linhas e colunas do subarray e calcular sua soma. Esses seis loops aninhados nos dão uma algoritmo. Embora o limite de tempo UVa, $O(N^6)$, seja bastante frouxo para esse problema, ele não é tão tolerante. Então, precisamos de uma abordagem mais eficiente. Veja uma possível implementação do algoritmo näive

```
maximo=-999999
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    para k de i a N
      para L de 1 a N
        soma=0
        para a de i a k
          para b de j a L
            soma=soma+MAT[a][b]
          fimpara
        fimpara
        se soma>maximo
          guarda=i,j,k,L
          maximo=soma
        fimse
      fimpara
    fimpara
  fimpara
fimpara
```

Estratégia de programação dinâmica

Com a reutilização de somas de subarrays. Para problemas básicos de programação dinâmica, como UVA 108, o segredo é descobrir quais informações você pode armazenar em etapas anteriores que economizarão tempo em etapas subsequentes.

A ideia-chave para UVa 108 é que os resultados de um cálculo anterior (uma soma, neste caso) podem ser usados em cálculos subsequentes. Especificamente, uma vez que calculamos a soma de parte de um subarray, não precisamos calculá-la repetidamente quando estamos processando outros subarrays que se sobrepõem à mesma região. **Se pudermos encontrar uma maneira eficiente de armazenar e reutilizar essas somas, podemos economizar muito tempo de cálculo**. A estrutura de dados usada em programação dinâmica para esse propósito é tradicionalmente chamada de tabela de memorando .

A tabela de memorandos tem o mesmo tamanho, formato e tipo que a matriz de entrada: a matriz de inteiros. O valor inteiro na posição (i, j) na tabela de memorandos armazena a soma da submatriz cujo canto superior esquerdo é o canto superior esquerdo da matriz de entrada e cujo canto inferior direito é (i, j) na matriz de entrada. Por exemplo, posição da tabela de memorandos $(2, 2)$ armazenaria a soma do subarray quadrado de $(1, 1)$ até $(2, 2)$ em nosso esquema de numeração baseado em 1.

Etapa 1: Gerar a tabela de memorandos
A chave para gerar a tabela de memorandos de forma eficiente é iterar pela matriz de entrada linha por linha e usar resultados anteriores para calcular os subsequentes. Você está basicamente usando a tabela de memorandos para gerar a si mesma. Para você acompanhar, eis como ficaria a tabela de memorandos do exemplo acima citado

```
0   -2    5    5
9    9   10   12
5    6    3    6
4   13   10   11
```

Para ver como isso funciona, considere a $4, 4$ matriz tabela de memorando mostrada abaixo. Os números de 01 para 16 representam a ordem na qual iremos iterar pelos dezesseis valores de entrada e calcular os dezesseis valores da tabela de memorando. Preencher a tabela dessa forma simplificará o código. Podemos ler a entrada e preencher a tabela de memorando simultaneamente. Portanto, não há necessidade de armazenar uma matriz separada de valores de entrada. Todas as informações de que precisamos na próxima etapa estarão na tabela de memorando.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Veja como o processo funciona: Escolha uma célula na matriz. Como exemplo, usarei a célula 10. Isso representa o canto inferior direito de um sub-array de 6 células que começa na célula 1 (três linhas e duas colunas). De acordo com nossa definição de tabela de memorando, a célula 10 deve conter a soma dos valores de entrada 01, 02, 05, 06, 09, e 10. Mas lembre-se de que estamos preenchendo a tabela de memorando em ordem. Então, quando chegamos à célula 10, já temos valores corretos para células 01 até 09. Como podemos usar esses valores?

A resposta é que podemos usar três informações para calcular cada soma de submatriz:

- A célula única que forma seu canto inferior direito. Neste caso, célula 10.
- O subarray acima dele. Neste caso, é o subarray $2, 2$ começando em 01 e terminando em 06. A soma deste subarray é armazenada na célula 06.
- O subarray à esquerda dele. Neste caso, é o subarray $3, 1$ começando em 01 e terminando em 09. A soma deste subarray é armazenada na célula 09.

Então, em vez de adicionar dez células para obter a soma que estamos procurando, podemos simplesmente adicionar três células: 06, 09, e 10. E para submatrizes maiores, a economia de tempo é maior.

Se você estiver acompanhando, notará que esse processo não está muito certo. O subarray acima e o subarray esquerdo 01 se sobrepõem nas células 01 e 05. Então, temos que subtrair a soma dessas células para evitar contagem dupla. Felizmente, a quantidade que precisamos subtrair é armazenada em uma única célula, a 05. Essa quantidade contada duas vezes sempre será uma célula diagonalmente para cima e para a esquerda da nossa célula-alvo.

Agora estamos no final do Passo 1, então nossa tabela de memorando contém as somas de todos os subarrays que começam no canto superior esquerdo do array de entrada. Mas o subarray de soma máxima pode começar em uma posição diferente. Para lidar com o caso geral, precisamos do Passo 2.

Etapa 2: Avalie todas as posições iniciais e finais
Esta etapa compartilha algumas ideias com a abordagem da Busca Completa descrita acima. A diferença é que as somas do subarray já foram calculadas eficientemente na Etapa 1. Então, em vez de um algoritmo de (Busca Completa) $O(N^6)$, temos um $O(N^2)$ passo (Passo 1) seguido por um passo $O(N^4)$ (Etapa 2). Como você deve saber pelas regras da análise de algoritmos assintóticos, isso nos dá uma visão geral de tempo de execução $O(N^4)$, o que é suficiente para que esta solução seja aceita na UVa OJ.

Assim como na Busca Completa, a Etapa 2 usa quatro loops aninhados, um para cada linha inicial, coluna inicial, linha final e coluna final. Essas quatro coordenadas definem o subarray em questão. Temos que verificar cada subarray para ver se ele tem a soma máxima vista até agora.

Veja o que acontece dentro do loop mais interno:

Primeiro, recuperamos a soma memorizada que calculamos na Etapa 1 para o subarray que termina na linha e coluna atuais. Lembre-se de que isso representa a soma do subarray que começa no canto superior esquerdo da entrada. Como isso pode não ser o mesmo que o canto superior esquerdo do subarray que estamos verificando no momento, temos que ajustar a soma usando um processo semelhante ao que usamos na Etapa 1. Para ilustrar, suponha que temos a seguinte tabela de memorando concluída. Como antes, os números identificam a ordem em que as células foram calculadas.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Como exemplo, suponha que nossos loops aninhados atualmente identifiquem uma posição inicial da célula 11 (linha 3, coluna 3) e uma posição final da célula 16 (linha 4, coluna 4). Precisamos encontrar a soma do subarray $2, 2$. O valor memorizado na célula 16 nos dá a soma de todo o array, então temos que nos livrar das partes que não precisamos. Como na Etapa 1, podemos fazer isso usando os valores que estão acima e à esquerda do alvo.

Aqui está outra maneira de analisar a tabela de memorandos:

```
ZZ ZZ XX XX
ZZ ZZ XX XX
```

```
YY YY 11 12
YY YY 15 16
```

As células XX identificam o subarray que está acima do alvo. As células YY identificam o subarray que está à esquerda do alvo. E as células ZZ identificam o subarray que está acima e à esquerda (ou seja, diagonalmente para cima e à esquerda) do alvo.

Esta visualização deve ser familiar do Passo 1. Mas neste passo, precisamos subtrair em vez de adicionar, já que estamos removendo somas que não precisamos. Então, começando com a soma na célula de memorando 16, precisamos:

Subtraia o valor na célula memo 08. Isso remove a soma do subarray que está acima do alvo (os XXs e ZZs). Subtraia o valor na célula memo 14. Isso remove a soma do subarray que está à esquerda do alvo (os YYs e ZZs). Observe que subtraímos o ZZs duas vezes. Nós os contamos duas vezes, assim como contamos duas vezes o valor diagonal no Passo 1. Portanto, precisamos que adicione novamente o valor na célula de memorando 06, os ZZs. Depois que adicionamos de volta uma das cópias contadas duas vezes do ZZs, temos uma soma precisa para nosso subarray de destino. Então, apenas comparamos o resultado com nossa soma máxima atual e atualizamos o máximo, se necessário.

Quando tivermos passado por todas as coordenadas iniciais e finais possíveis, nosso valor máximo será a resposta, e simplesmente imprimiremos esse inteiro.

Importante Se o seu ambiente de programação começa as numerações em 0 (Python, JS, Java, C/C++), para implementar os algoritmos, é conveniente deixar uma linha zerada (a linha 0) e igualmente uma coluna zerada (a coluna 0). Agora as verificações passam a ser em referência à coluna e linha zeros e não a 1 como nos algoritmos. Aliás, no site acima referido o autor faz isso.

O Algoritmo final

Veja como ficou uma possível solução

```
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    se i>1
      MAT[i][j]=MAT[i][j]+MAT[i-1][j]
    fimse
    se j>1
      MAT[i][j]=MAT[i][j]+MAT[i][j-1]
    fimse
    se (i>1) AND (j>1)
      MAT[i][j]=MAT[i][j]-MAT[i-1][j-1]
    fimse
  fimpara
fimpara
ans = 0
para i de 1 a N
  para j de 1 a N
    para k de i a N // note o i
      para L de j a N // idem j
        cur = MAT[k;L]
        se i > 1
          cur=cur-MAT[i-1][L]
        fimse
        se j>1
          cur=cur-MAT[i][j-1]
        fimse
        se (i>1) AND (j>1)
          cur=cur+MAT[i-1][j-1]
        fimse
        se and<cur
          ans=cur
          salva i,j,k,L
        fimse
      fimpara
    fimpara
  fimpara
fimpara
imprima (ans,salva)
```

Se você comparar os 2 algoritmos acima, verá que são muito parecidos. Só que o primeiro (força bruta) tem 6 *paras* um dentro do outro o que caracteriza o desempenho $O(N^6)$. Já o segundo tem os mesmos 6 *paras*, mas primeiro há um dentro do outro ($O(N^2)$) e depois, há 4 *paras*. Da teoria, ao final os 2 *paras* superiores podem ser desconsiderados e o resultado final fica só com $O(N^4)$.

Um exemplo completo

Para você testar seu programa. A matriz

```
-41 -98  91  71 -42   8   3 -79 -17  16
 76 -11  46  74  44  61  42  49 -96  78
  5  -7 -86  43  -2  34  37 -60  84  74
 79   9 -72  -9  98 -56 -10 -36   3  77
-12  -6  62 -26 -57 100 -69  27  24 -99
-99  55  46 -36 -16  37  37 -58  68  42
 66 -81 -83  53  26 -57 -57 -83 -22  91
 90 -22 -46  39 -43  56  57 -15 -43 -61
-97 -61  97 -51  64 -72 -20  21 -64  66
-68  98 -48 -53 -79 -56  27  40  59  40
```

Deu como resposta 588 2, 1, 6, 10.

☞ Para você fazer

Calcule a máxima subsoma da matriz 10×10 abaixo e determine também as dimensões e posição da submatriz. Para tanto identifique o canto superior esquerdo e inferior direito da submatriz (origem=1). Os números estão entre $-600 < n \leq 600$.

```
 143   93  216 -401 -384 -259  -42  380 -382 -465
-144  279  394  395  -49  297   31 -388 -484  163
 176 -473 -192 -452  -56 -107  133 -200 -556 -383
 113 -404 -306   -7   20 -431 -281  130  365 -317
 131  174  -19  336 -334 -354  291 -282 -463 -202
-130 -500  101  104  340 -217    7 -460   77  220
-153   98 -561 -135  196  146  247 -457 -266 -396
-234 -396  135 -408 -573   51  295 -174  144 -297
-231 -507   58 -412  192  207  301 -177  183 -227
-435 -138  335  362 -327 -379   83 -155  225 -391
```

Responda aqui:

soma	localização

506-75646 -

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 21/10/2024 - 12:06:52.9
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
vivxhb2a V: 1.02
75653 JOAO PEDRO MARIANO SUEKE
24JOG506 - 10 _____ / _____ / _____

UVA 108 - Maior subsoma em matriz

Observação: este texto está baseado em https://www.redgreencode.com/three-ways-solve-uva-108/ do autor Duncan Smith.

O problema original - 108

Como está lá, já traduzido:
Um problema que é simples de resolver numa dimensão é muitas vezes muito mais difícil de resolver em mais de uma dimensão. Considere satisfazer uma expressão booleana na forma normal conjuntiva na qual cada conjunto consiste em exatamente 3 disjuntos. Este problema (3-SAT) é NP-completo. O problema 2-SAT é resolvido de forma bastante eficiente, no entanto. Em contraste, alguns problemas pertencem à mesma classe de complexidade independentemente da dimensionalidade do problema. Dada uma matriz bidimensional de inteiros positivos e negativos, encontre o subretângulo com o maior soma. A soma de um retângulo é a soma de todos os elementos desse retângulo. Neste problema o subretângulo com a maior soma é chamado de subretângulo máximo. Um subretângulo é qualquer submatriz contígua de tamanho 1 x 1 ou maior localizado sobre a matriz. Como exemplo o sub-retângulo máximo do array

```
0   -2   -7   0
9    2   -6   2
-4   1   -4   1
-1   8    0  -2
```

está no canto esquerdo inferior

```
9   2
-4  1
-1  8
```

cuja soma é 15

Entrada:
A entrada consiste em uma matriz N × N de inteiros. A entrada começa com um único inteiro positivo N em uma linha isolada indicando o tamanho do quadrado da matriz bidimensional. Isto é seguido por números inteiros N^2 separados por espaços em branco (novas linhas e espaços). Esses inteiros N^2 constituem a matriz em ordem de linha maior (ou seja, todos os números na primeira linha, da esquerda para a direita, depois, todos os números da segunda linha, da esquerda para a direita, etc.). N pode ser tão grande quanto 100. Os números no array estarão no intervalo $[-127, 127]$.

Saída:
A saída é a soma do sub-retângulo máximo.

Exemplo de entrada

```
4
0 -2 -7 0 9 -2 -6 2
-4 1 -4 1 -1
8 0 -2
```

Exemplo de saída
15

Pesquisa completa

A abordagem de busca completa para este problema requer que enumeremos todos os subarrays do array de entrada e calculemos a soma de cada um separadamente. Para enumerar os subarrays de uma matriz, considere que você pode identificar cada submatriz por um conjunto de quatro inteiros 1 e N: linha inicial, coluna inicial, linha final e coluna final. Você pode pensar nesses quatro inteiros como as coordenadas dos cantos superior esquerdo e inferior direito do subarray. Tendo identificado um subarray usando essa abordagem de busca completa, a maneira ingênua de encontrar sua soma é adicionar todos os seus números juntos.

Para gerar todas as coordenadas do subarray, podemos usar quatro loops aninhados, um para cada coordenada. Outros dois loops aninhados são suficientes para iterar pelas linhas e colunas do subarray e calcular sua soma. Esses seis loops aninhados nos dão uma algoritmo. Embora o limite de tempo UVa, $O(N^6)$, seja bastante frouxo para esse problema, ele não é tão tolerante. Então, precisamos de uma abordagem mais eficiente. Veja uma possível implementação do algoritmo näive

```
maximo=-999999
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    para k de i a N
      para L de 1 a N
        soma=0
        para a de i a k
          para b de j a L
            soma=soma+MAT[a][b]
          fimpara
        fimpara
        se soma>maximo
          guarda=i,j,k,L
          maximo=soma
        fimse
      fimpara
    fimpara
  fimpara
fimpara
```

Estratégia de programação dinâmica

Com a reutilização de somas de subarrays. Para problemas básicos de programação dinâmica, como UVA 108, o segredo é descobrir quais informações você pode armazenar em etapas anteriores que economizarão tempo em etapas subsequentes.

A ideia-chave para UVa 108 é que os resultados de um cálculo anterior (uma soma, neste caso) podem ser usados em cálculos subsequentes. Especificamente, uma vez que calculamos a soma de parte de um subarray, não precisamos calculá-la repetidamente quando estamos processando outros subarrays que se sobrepõem à mesma região. **Se pudermos encontrar uma maneira eficiente de armazenar e reutilizar essas somas, podemos economizar muito tempo de cálculo**. A estrutura de dados usada em programação dinâmica para esse propósito é tradicionalmente chamada de tabela de memorando .

A tabela de memorandos tem o mesmo tamanho, formato e tipo que a matriz de entrada: a matriz de inteiros. O valor inteiro na posição (i, j) na tabela de memorandos armazena a soma da submatriz cujo canto superior esquerdo é o canto superior esquerdo da matriz de entrada e cujo canto inferior direito é (i, j) na matriz de entrada. Por exemplo, posição da tabela de memorandos $(2, 2)$ armazenaria a soma do subarray quadrado de $(1, 1)$ até $(2, 2)$ em nosso esquema de numeração baseado em 1.

Etapa 1: Gerar a tabela de memorandos
A chave para gerar a tabela de memorandos de forma eficiente é iterar pela matriz de entrada linha por linha e usar resultados anteriores para calcular os subsequentes. Você está basicamente usando a tabela de memorandos para gerar a si mesma. Para você acompanhar, eis como ficaria a tabela de memorandos do exemplo acima citado

```
0   -2    5    5
9    9   10   12
5    6    3    6
4   13   10   11
```

Para ver como isso funciona, considere a $4, 4$ matriz tabela de memorando mostrada abaixo. Os números de 01 para 16 representam a ordem na qual iremos iterar pelos dezesseis valores de entrada e calcular os dezesseis valores da tabela de memorando. Preencher a tabela dessa forma simplificará o código. Podemos ler a entrada e preencher a tabela de memorando simultaneamente. Portanto, não há necessidade de armazenar uma matriz separada de valores de entrada. Todas as informações de que precisamos na próxima etapa estarão na tabela de memorando.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Veja como o processo funciona: Escolha uma célula na matriz. Como exemplo, usarei a célula 10. Isso representa o canto inferior direito de um sub-array de 6 células que começa na célula 1 (três linhas e duas colunas). De acordo com nossa definição de tabela de memorando, a célula 10 deve conter a soma dos valores de entrada 01, 02, 05, 06, 09, e 10. Mas lembre-se de que estamos preenchendo a tabela de memorando em ordem. Então, quando chegamos à célula 10, já temos valores corretos para células 01 até 09. Como podemos usar esses valores?

A resposta é que podemos usar três informações para calcular cada soma de submatriz:

- A célula única que forma seu canto inferior direito. Neste caso, célula 10.
- O subarray acima dele. Neste caso, é o subarray $2, 2$ começando em 01 e terminando em 06. A soma deste subarray é armazenada na célula 06.
- O subarray à esquerda dele. Neste caso, é o subarray $3, 1$ começando em 01 e terminando em 09. A soma deste subarray é armazenada na célula 09.

Então, em vez de adicionar dez células para obter a soma que estamos procurando, podemos simplesmente adicionar três células: 06, 09, e 10. E para submatrizes maiores, a economia de tempo é maior.

Se você estiver acompanhando, notará que esse processo não está muito certo. O subarray acima e o subarray esquerdo 01 se sobrepõem nas células 01 e 05. Então, temos que subtrair a soma dessas células para evitar contagem dupla. Felizmente, a quantidade que precisamos subtrair é armazenada em uma única célula, a 05. Essa quantidade contada duas vezes sempre será uma célula diagonalmente para cima e para a esquerda da nossa célula-alvo.

Agora estamos no final do Passo 1, então nossa tabela de memorando contém as somas de todos os subarrays que começam no canto superior esquerdo do array de entrada. Mas o subarray de soma máxima pode começar em uma posição diferente. Para lidar com o caso geral, precisamos do Passo 2.

Etapa 2: Avalie todas as posições iniciais e finais
Esta etapa compartilha algumas ideias com a abordagem da Busca Completa descrita acima. A diferença é que as somas do subarray já foram calculadas eficientemente na Etapa 1. Então, em vez de um algoritmo de (Busca Completa) $O(N^6)$, temos um $O(N^2)$ passo (Passo 1) seguido por um passo $O(N^4)$ (Etapa 2). Como você deve saber pelas regras da análise de algoritmos assintóticos, isso nos dá uma visão geral de tempo de execução $O(N^4)$, o que é suficiente para que esta solução seja aceita na UVa OJ.

Assim como na Busca Completa, a Etapa 2 usa quatro loops aninhados, um para cada linha inicial, coluna inicial, linha final e coluna final. Essas quatro coordenadas definem o subarray em questão. Temos que verificar cada subarray para ver se ele tem a soma máxima vista até agora.

Veja o que acontece dentro do loop mais interno:

Primeiro, recuperamos a soma memorizada que calculamos na Etapa 1 para o subarray que termina na linha e coluna atuais. Lembre-se de que isso representa a soma do subarray que começa no canto superior esquerdo da entrada. Como isso pode não ser o mesmo que o canto superior esquerdo do subarray que estamos verificando no momento, temos que ajustar a soma usando um processo semelhante ao que usamos na Etapa 1. Para ilustrar, suponha que temos a seguinte tabela de memorando concluída. Como antes, os números identificam a ordem em que as células foram calculadas.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Como exemplo, suponha que nossos loops aninhados atualmente identifiquem uma posição inicial da célula 11 (linha 3, coluna 3) e uma posição final da célula 16 (linha 4, coluna 4). Precisamos encontrar a soma do subarray $2, 2$. O valor memorizado na célula 16 nos dá a soma de todo o array, então temos que nos livrar das partes que não precisamos. Como na Etapa 1, podemos fazer isso usando os valores que estão acima e à esquerda do alvo.

Aqui está outra maneira de analisar a tabela de memorandos:

```
ZZ ZZ XX XX
ZZ ZZ XX XX
```

```
YY YY 11 12
YY YY 15 16
```

As células XX identificam o subarray que está acima do alvo. As células YY identificam o subarray que está à esquerda do alvo. E as células ZZ identificam o subarray que está acima e à esquerda (ou seja, diagonalmente para cima e à esquerda) do alvo.

Esta visualização deve ser familiar do Passo 1. Mas neste passo, precisamos subtrair em vez de adicionar, já que estamos removendo somas que não precisamos. Então, começando com a soma na célula de memorando 16, precisamos:

Subtraia o valor na célula memo 08. Isso remove a soma do subarray que está acima do alvo (os XXs e ZZs). Subtraia o valor na célula memo 14. Isso remove a soma do subarray que está à esquerda do alvo (os YYs e ZZs). Observe que subtraímos o ZZs duas vezes. Nós os contamos duas vezes, assim como contamos duas vezes o valor diagonal no Passo 1. Portanto, precisamos que adicione novamente o valor na célula de memorando 06, os ZZs. Depois que adicionamos de volta uma das cópias contadas duas vezes do ZZs, temos uma soma precisa para nosso subarray de destino. Então, apenas comparamos o resultado com nossa soma máxima atual e atualizamos o máximo, se necessário.

Quando tivermos passado por todas as coordenadas iniciais e finais possíveis, nosso valor máximo será a resposta, e simplesmente imprimiremos esse inteiro.

Importante Se o seu ambiente de programação começa as numerações em 0 (Python, JS, Java, C/C++), para implementar os algoritmos, é conveniente deixar uma linha zerada (a linha 0) e igualmente uma coluna zerada (a coluna 0). Agora as verificações passam a ser em referência à coluna e linha zeros e não a 1 como nos algoritmos. Aliás, no site acima referido o autor faz isso.

O Algoritmo final

Veja como ficou uma possível solução

```
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    se i>1
      MAT[i][j]=MAT[i][j]+MAT[i-1][j]
    fimse
    se j>1
      MAT[i][j]=MAT[i][j]+MAT[i][j-1]
    fimse
    se (i>1) AND (j>1)
      MAT[i][j]=MAT[i][j]-MAT[i-1][j-1]
    fimse
  fimpara
fimpara
ans = 0
para i de 1 a N
  para j de 1 a N
    para k de i a N // note o i
      para L de j a N // idem j
        cur = MAT[k;L]
        se i > 1
          cur=cur-MAT[i-1][L]
        fimse
        se j>1
          cur=cur-MAT[i][j-1]
        fimse
        se (i>1) AND (j>1)
          cur=cur+MAT[i-1][j-1]
        fimse
        se and<cur
          ans=cur
          salva i,j,k,L
        fimse
      fimpara
    fimpara
  fimpara
fimpara
imprima (ans,salva)
```

Se você comparar os 2 algoritmos acima, verá que são muito parecidos. Só que o primeiro (força bruta) tem 6 *paras* um dentro do outro o que caracteriza o desempenho $O(N^6)$. Já o segundo tem os mesmos 6 *paras*, mas primeiro há um dentro do outro ($O(N^2)$) e depois, há 4 *paras*. Da teoria, ao final os 2 *paras* superiores podem ser desconsiderados e o resultado final fica só com $O(N^4)$.

Um exemplo completo

Para você testar seu programa. A matriz

```
-41 -98  91  71 -42   8   3 -79 -17  16
 76 -11  46  74  44  61  42  49 -96  78
  5  -7 -86  43  -2  34  37 -60  84  74
 79   9 -72  -9  98 -56 -10 -36   3  77
-12  -6  62 -26 -57 100 -69  27  24 -99
-99  55  46 -36 -16  37  37 -58  68  42
 66 -81 -83  53  26 -57 -57 -83 -22  91
 90 -22 -46  39 -43  56  57 -15 -43 -61
-97 -61  97 -51  64 -72 -20  21 -64  66
-68  98 -48 -53 -79 -56  27  40  59  40
```

Deu como resposta 588 2, 1, 6, 10.

☞ Para você fazer

Calcule a máxima subsoma da matriz 10×10 abaixo e determine também as dimensões e posição da submatriz. Para tanto identifique o canto superior esquerdo e inferior direito da submatriz (origem=1). Os números estão entre $-600 < n \leq 600$.

```
-183   27 -464  -31 -127  -89  114 -115 -430  144
-172  122 -387  262  312 -306 -208   38 -114 -146
 230 -577  279 -195 -572  166   -8   81  349 -363
-299 -375 -230  247  196  205 -464  251 -125 -240
-589 -324   72 -344 -206   57 -295  198  -75 -300
-535 -488   37  -21  361 -324    6 -200    3 -487
 275  268 -221 -536  -91 -441 -396  231 -158  273
-341 -569  124  -81 -515 -410 -417 -516  173 -252
-249  -22 -426  343   84 -544  248  347  -82  210
  82   38  346  -68 -126  -68 -189  100 -187 -237
```

Responda aqui:

soma	localização

506-75653 -

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 21/10/2024 - 12:06:52.9
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
vivxhb2a V: 1.02
75660 JOAO VICTOR GOMES VILELA GER 24JOG506 - 11 _____ / _____ / _____

UVA 108 - Maior subsoma em matriz

Observação: este texto está baseado em https://www.redgreencode.com/three-ways-solve-uva-108/ do autor Duncan Smith.

O problema original - 108

Como está lá, já traduzido:
Um problema que é simples de resolver numa dimensão é muitas vezes muito mais difícil de resolver em mais de uma dimensão. Considere satisfazer uma expressão booleana na forma normal conjuntiva na qual cada conjunto consiste em exatamente 3 disjuntos. Este problema (3-SAT) é NP-completo. O problema 2-SAT é resolvido de forma bastante eficiente, no entanto. Em contraste, alguns problemas pertencem à mesma classe de complexidade independentemente da dimensionalidade do problema. Dada uma matriz bidimensional de inteiros positivos e negativos, encontre o subretângulo com o maior soma. A soma de um retângulo é a soma de todos os elementos desse retângulo. Neste problema o subretângulo com a maior soma é chamado de subretângulo máximo. Um subretângulo é qualquer submatriz contígua de tamanho 1 x 1 ou maior localizado sobre a matriz. Como exemplo o sub-retângulo máximo do array

```
0   -2   -7   0
9    2   -6   2
-4   1   -4   1
-1   8    0  -2
```

está no canto esquerdo inferior

```
9   2
-4  1
-1  8
```

cuja soma é 15

Entrada:
A entrada consiste em uma matriz N × N de inteiros. A entrada começa com um único inteiro positivo N em uma linha isolada indicando o tamanho do quadrado da matriz bidimensional. Isto é seguido por números inteiros N^2 separados por espaços em branco (novas linhas e espaços). Esses inteiros N^2 constituem a matriz em ordem de linha maior (ou seja, todos os números na primeira linha, da esquerda para a direita, depois, todos os números da segunda linha, da esquerda para a direita, etc.). N pode ser tão grande quanto 100. Os números no array estarão no intervalo $[-127, 127]$.

Saída:
A saída é a soma do sub-retângulo máximo.

Exemplo de entrada

```
4
0 -2 -7 0 9 -2 -6 2
-4 1 -4 1 -1
8 0 -2
```

Exemplo de saída
15

Pesquisa completa

A abordagem de busca completa para este problema requer que enumeremos todos os subarrays do array de entrada e calculemos a soma de cada um separadamente. Para enumerar os subarrays de uma matriz, considere que você pode identificar cada submatriz por um conjunto de quatro inteiros 1 e N: linha inicial, coluna inicial, linha final e coluna final. Você pode pensar nesses quatro inteiros como as coordenadas dos cantos superior esquerdo e inferior direito do subarray. Tendo identificado um subarray usando essa abordagem de busca completa, a maneira ingênua de encontrar sua soma é adicionar todos os seus números juntos.

Para gerar todas as coordenadas do subarray, podemos usar quatro loops aninhados, um para cada coordenada. Outros dois loops aninhados são suficientes para iterar pelas linhas e colunas do subarray e calcular sua soma. Esses seis loops aninhados nos dão uma algoritmo. Embora o limite de tempo UVa, $O(N^6)$, seja bastante frouxo para esse problema, ele não é tão tolerante. Então, precisamos de uma abordagem mais eficiente. Veja uma possível implementação do algoritmo näive

```
maximo=-999999
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    para k de i a N
      para L de 1 a N
        soma=0
        para a de i a k
          para b de j a L
            soma=soma+MAT[a][b]
          fimpara
        fimpara
        se soma>maximo
          guarda=i,j,k,L
          maximo=soma
        fimse
      fimpara
    fimpara
  fimpara
fimpara
```

Estratégia de programação dinâmica

Com a reutilização de somas de subarrays. Para problemas básicos de programação dinâmica, como UVA 108, o segredo é descobrir quais informações você pode armazenar em etapas anteriores que economizarão tempo em etapas subsequentes.

A ideia-chave para UVa 108 é que os resultados de um cálculo anterior (uma soma, neste caso) podem ser usados em cálculos subsequentes. Especificamente, uma vez que calculamos a soma de parte de um subarray, não precisamos calculá-la repetidamente quando estamos processando outros subarrays que se sobrepõem à mesma região. **Se pudermos encontrar uma maneira eficiente de armazenar e reutilizar essas somas, podemos economizar muito tempo de cálculo**. A estrutura de dados usada em programação dinâmica para esse propósito é tradicionalmente chamada de tabela de memorando .

A tabela de memorandos tem o mesmo tamanho, formato e tipo que a matriz de entrada: a matriz de inteiros. O valor inteiro na posição (i, j) na tabela de memorandos armazena a soma da submatriz cujo canto superior esquerdo é o canto superior esquerdo da matriz de entrada e cujo canto inferior direito é (i, j) na matriz de entrada. Por exemplo, posição da tabela de memorandos $(2, 2)$ armazenaria a soma do subarray quadrado de $(1, 1)$ até $(2, 2)$ em nosso esquema de numeração baseado em 1.

Etapa 1: Gerar a tabela de memorandos
A chave para gerar a tabela de memorandos de forma eficiente é iterar pela matriz de entrada linha por linha e usar resultados anteriores para calcular os subsequentes. Você está basicamente usando a tabela de memorandos para gerar a si mesma. Para você acompanhar, eis como ficaria a tabela de memorandos do exemplo acima citado

```
0   -2    5    5
9    9   10   12
5    6    3    6
4   13   10   11
```

Para ver como isso funciona, considere a $4, 4$ matriz tabela de memorando mostrada abaixo. Os números de 01 para 16 representam a ordem na qual iremos iterar pelos dezesseis valores de entrada e calcular os dezesseis valores da tabela de memorando. Preencher a tabela dessa forma simplificará o código. Podemos ler a entrada e preencher a tabela de memorando simultaneamente. Portanto, não há necessidade de armazenar uma matriz separada de valores de entrada. Todas as informações de que precisamos na próxima etapa estarão na tabela de memorando.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Veja como o processo funciona: Escolha uma célula na matriz. Como exemplo, usarei a célula 10. Isso representa o canto inferior direito de um sub-array de 6 células que começa na célula 1 (três linhas e duas colunas). De acordo com nossa definição de tabela de memorando, a célula 10 deve conter a soma dos valores de entrada 01, 02, 05, 06, 09, e 10. Mas lembre-se de que estamos preenchendo a tabela de memorando em ordem. Então, quando chegamos à célula 10, já temos valores corretos para células 01 até 09. Como podemos usar esses valores?

A resposta é que podemos usar três informações para calcular cada soma de submatriz:

- A célula única que forma seu canto inferior direito. Neste caso, célula 10.
- O subarray acima dele. Neste caso, é o subarray $2, 2$ começando em 01 e terminando em 06. A soma deste subarray é armazenada na célula 06.
- O subarray à esquerda dele. Neste caso, é o subarray $3, 1$ começando em 01 e terminando em 09. A soma deste subarray é armazenada na célula 09.

Então, em vez de adicionar dez células para obter a soma que estamos procurando, podemos simplesmente adicionar três células: 06, 09, e 10. E para submatrizes maiores, a economia de tempo é maior.

Se você estiver acompanhando, notará que esse processo não está muito certo. O subarray acima e o subarray esquerdo 01 se sobrepõem nas células 01 e 05. Então, temos que subtrair a soma dessas células para evitar contagem dupla. Felizmente, a quantidade que precisamos subtrair é armazenada em uma única célula, a 05. Essa quantidade contada duas vezes sempre será uma célula diagonalmente para cima e para a esquerda da nossa célula-alvo.

Agora estamos no final do Passo 1, então nossa tabela de memorando contém as somas de todos os subarrays que começam no canto superior esquerdo do array de entrada. Mas o subarray de soma máxima pode começar em uma posição diferente. Para lidar com o caso geral, precisamos do Passo 2.

Etapa 2: Avalie todas as posições iniciais e finais
Esta etapa compartilha algumas ideias com a abordagem da Busca Completa descrita acima. A diferença é que as somas do subarray já foram calculadas eficientemente na Etapa 1. Então, em vez de um algoritmo de (Busca Completa) $O(N^6)$, temos um $O(N^2)$ passo (Passo 1) seguido por um passo $O(N^4)$ (Etapa 2). Como você deve saber pelas regras da análise de algoritmos assintóticos, isso nos dá uma visão geral de tempo de execução $O(N^4)$, o que é suficiente para que esta solução seja aceita na UVa OJ.

Assim como na Busca Completa, a Etapa 2 usa quatro loops aninhados, um para cada linha inicial, coluna inicial, linha final e coluna final. Essas quatro coordenadas definem o subarray em questão. Temos que verificar cada subarray para ver se ele tem a soma máxima vista até agora.

Veja o que acontece dentro do loop mais interno:

Primeiro, recuperamos a soma memorizada que calculamos na Etapa 1 para o subarray que termina na linha e coluna atuais. Lembre-se de que isso representa a soma do subarray que começa no canto superior esquerdo da entrada. Como isso pode não ser o mesmo que o canto superior esquerdo do subarray que estamos verificando no momento, temos que ajustar a soma usando um processo semelhante ao que usamos na Etapa 1. Para ilustrar, suponha que temos a seguinte tabela de memorando concluída. Como antes, os números identificam a ordem em que as células foram calculadas.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Como exemplo, suponha que nossos loops aninhados atualmente identifiquem uma posição inicial da célula 11 (linha 3, coluna 3) e uma posição final da célula 16 (linha 4, coluna 4). Precisamos encontrar a soma do subarray $2, 2$. O valor memorizado na célula 16 nos dá a soma de todo o array, então temos que nos livrar das partes que não precisamos. Como na Etapa 1, podemos fazer isso usando os valores que estão acima e à esquerda do alvo.

Aqui está outra maneira de analisar a tabela de memorandos:

```
ZZ ZZ XX XX
ZZ ZZ XX XX
```

```
YY YY 11 12
YY YY 15 16
```

As células XX identificam o subarray que está acima do alvo. As células YY identificam o subarray que está à esquerda do alvo. E as células ZZ identificam o subarray que está acima e à esquerda (ou seja, diagonalmente para cima e à esquerda) do alvo.

Esta visualização deve ser familiar do Passo 1. Mas neste passo, precisamos subtrair em vez de adicionar, já que estamos removendo somas que não precisamos. Então, começando com a soma na célula de memorando 16, precisamos:

Subtraia o valor na célula memo 08. Isso remove a soma do subarray que está acima do alvo (os XXs e ZZs). Subtraia o valor na célula memo 14. Isso remove a soma do subarray que está à esquerda do alvo (os YYs e ZZs). Observe que subtraímos o ZZs duas vezes. Nós os contamos duas vezes, assim como contamos duas vezes o valor diagonal no Passo 1. Portanto, precisamos que adicione novamente o valor na célula de memorando 06, os ZZs. Depois que adicionamos de volta uma das cópias contadas duas vezes do ZZs, temos uma soma precisa para nosso subarray de destino. Então, apenas comparamos o resultado com nossa soma máxima atual e atualizamos o máximo, se necessário.

Quando tivermos passado por todas as coordenadas iniciais e finais possíveis, nosso valor máximo será a resposta, e simplesmente imprimiremos esse inteiro.

Importante Se o seu ambiente de programação começa as numerações em 0 (Python, JS, Java, C/C++), para implementar os algoritmos, é conveniente deixar uma linha zerada (a linha 0) e igualmente uma coluna zerada (a coluna 0). Agora as verificações passam a ser em referência à coluna e linha zeros e não a 1 como nos algoritmos. Aliás, no site acima referido o autor faz isso.

O Algoritmo final

Veja como ficou uma possível solução

```
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    se i>1
      MAT[i][j]=MAT[i][j]+MAT[i-1][j]
    fimse
    se j>1
      MAT[i][j]=MAT[i][j]+MAT[i][j-1]
    fimse
    se (i>1) AND (j>1)
      MAT[i][j]=MAT[i][j]-MAT[i-1][j-1]
    fimse
  fimpara
fimpara
ans = 0
para i de 1 a N
  para j de 1 a N
    para k de i a N // note o i
      para L de j a N // idem j
        cur = MAT[k;L]
        se i > 1
          cur=cur-MAT[i-1][L]
        fimse
        se j>1
          cur=cur-MAT[i][j-1]
        fimse
        se (i>1) AND (j>1)
          cur=cur+MAT[i-1][j-1]
        fimse
        se and<cur
          ans=cur
          salva i,j,k,L
        fimse
      fimpara
    fimpara
  fimpara
fimpara
imprima (ans,salva)
```

Se você comparar os 2 algoritmos acima, verá que são muito parecidos. Só que o primeiro (força bruta) tem 6 *paras* um dentro do outro o que caracteriza o desempenho $O(N^6)$. Já o segundo tem os mesmos 6 *paras*, mas primeiro há um dentro do outro ($O(N^2)$) e depois, há 4 *paras*. Da teoria, ao final os 2 *paras* superiores podem ser desconsiderados e o resultado final fica só com $O(N^4)$.

Um exemplo completo

Para você testar seu programa. A matriz

```
-41 -98  91  71 -42   8   3 -79 -17  16
 76 -11  46  74  44  61  42  49 -96  78
  5  -7 -86  43  -2  34  37 -60  84  74
 79   9 -72  -9  98 -56 -10 -36   3  77
-12  -6  62 -26 -57 100 -69  27  24 -99
-99  55  46 -36 -16  37  37 -58  68  42
 66 -81 -83  53  26 -57 -57 -83 -22  91
 90 -22 -46  39 -43  56  57 -15 -43 -61
-97 -61  97 -51  64 -72 -20  21 -64  66
-68  98 -48 -53 -79 -56  27  40  59  40
```

Deu como resposta 588 2, 1, 6, 10.

☞ Para você fazer

Calcule a máxima subsoma da matriz 10×10 abaixo e determine também as dimensões e posição da submatriz. Para tanto identifique o canto superior esquerdo e inferior direito da submatriz (origem=1). Os números estão entre $-600 < n \leq 600$.

```
-169 -479  177 -102  290 -310  213 -574 -559 -504
 140  185  285  220  -99 -424   72 -478    3   66
-404 -173   13 -431  119 -205 -458 -582  290 -419
  58  228 -101  387 -131 -364 -210   25   51 -487
 262   58  -29  -68 -540  229  -28 -376 -128 -418
-303   65   52  -37  320 -135  259   -8 -471  251
-542  293  -47  256  -67 -137  308 -525  -64  294
 -54 -397  175  -16  332  329  141 -581 -320 -331
  31 -327 -229  262  262 -242 -312   26 -119   96
  85 -224 -515 -105 -346 -222  150  304 -153  170
```

Responda aqui:

soma	localização

506-75660 -

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 21/10/2024 - 12:06:52.9
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
vivxhb2a V: 1.02
75765 LUCAS ANTOSZCZYSZEN
24JOG506 - 12 _____ / _____ / _____

UVA 108 - Maior subsoma em matriz

Observação: este texto está baseado em https://www.redgreencode.com/three-ways-solve-uva-108/ do autor Duncan Smith.

O problema original - 108

Como está lá, já traduzido:
Um problema que é simples de resolver numa dimensão é muitas vezes muito mais difícil de resolver em mais de uma dimensão. Considere satisfazer uma expressão booleana na forma normal conjuntiva na qual cada conjunto consiste em exatamente 3 disjuntos. Este problema (3-SAT) é NP-completo. O problema 2-SAT é resolvido de forma bastante eficiente, no entanto. Em contraste, alguns problemas pertencem à mesma classe de complexidade independentemente da dimensionalidade do problema. Dada uma matriz bidimensional de inteiros positivos e negativos, encontre o subretângulo com o maior soma. A soma de um retângulo é a soma de todos os elementos desse retângulo. Neste problema o subretângulo com a maior soma é chamado de subretângulo máximo. Um subretângulo é qualquer submatriz contígua de tamanho 1 x 1 ou maior localizado sobre a matriz. Como exemplo o sub-retângulo máximo do array

```
0   -2   -7   0
9    2   -6   2
-4   1   -4   1
-1   8    0  -2
```

está no canto esquerdo inferior

```
9   2
-4  1
-1  8
```

cuja soma é 15

Entrada:
A entrada consiste em uma matriz N × N de inteiros. A entrada começa com um único inteiro positivo N em uma linha isolada indicando o tamanho do quadrado da matriz bidimensional. Isto é seguido por números inteiros N^2 separados por espaços em branco (novas linhas e espaços). Esses inteiros N^2 constituem a matriz em ordem de linha maior (ou seja, todos os números na primeira linha, da esquerda para a direita, depois, todos os números da segunda linha, da esquerda para a direita, etc.). N pode ser tão grande quanto 100. Os números no array estarão no intervalo $[-127, 127]$.

Saída:
A saída é a soma do sub-retângulo máximo.

Exemplo de entrada

```
4
0 -2 -7 0 9 -2 -6 2
-4 1 -4 1 -1
8 0 -2
```

Exemplo de saída
15

Pesquisa completa

A abordagem de busca completa para este problema requer que enumeremos todos os subarrays do array de entrada e calculemos a soma de cada um separadamente. Para enumerar os subarrays de uma matriz, considere que você pode identificar cada submatriz por um conjunto de quatro inteiros 1 e N: linha inicial, coluna inicial, linha final e coluna final. Você pode pensar nesses quatro inteiros como as coordenadas dos cantos superior esquerdo e inferior direito do subarray. Tendo identificado um subarray usando essa abordagem de busca completa, a maneira ingênua de encontrar sua soma é adicionar todos os seus números juntos.

Para gerar todas as coordenadas do subarray, podemos usar quatro loops aninhados, um para cada coordenada. Outros dois loops aninhados são suficientes para iterar pelas linhas e colunas do subarray e calcular sua soma. Esses seis loops aninhados nos dão uma algoritmo. Embora o limite de tempo UVa, $O(N^6)$, seja bastante frouxo para esse problema, ele não é tão tolerante. Então, precisamos de uma abordagem mais eficiente. Veja uma possível implementação do algoritmo näive

```
maximo=-999999
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    para k de i a N
      para L de 1 a N
        soma=0
        para a de i a k
          para b de j a L
            soma=soma+MAT[a][b]
          fimpara
        fimpara
        se soma>maximo
          guarda=i,j,k,L
          maximo=soma
        fimse
      fimpara
    fimpara
  fimpara
fimpara
```

Estratégia de programação dinâmica

Com a reutilização de somas de subarrays. Para problemas básicos de programação dinâmica, como UVA 108, o segredo é descobrir quais informações você pode armazenar em etapas anteriores que economizarão tempo em etapas subsequentes.

A ideia-chave para UVa 108 é que os resultados de um cálculo anterior (uma soma, neste caso) podem ser usados em cálculos subsequentes. Especificamente, uma vez que calculamos a soma de parte de um subarray, não precisamos calculá-la repetidamente quando estamos processando outros subarrays que se sobrepõem à mesma região. **Se pudermos encontrar uma maneira eficiente de armazenar e reutilizar essas somas, podemos economizar muito tempo de cálculo**. A estrutura de dados usada em programação dinâmica para esse propósito é tradicionalmente chamada de tabela de memorando .

A tabela de memorandos tem o mesmo tamanho, formato e tipo que a matriz de entrada: a matriz de inteiros. O valor inteiro na posição (i, j) na tabela de memorandos armazena a soma da submatriz cujo canto superior esquerdo é o canto superior esquerdo da matriz de entrada e cujo canto inferior direito é (i, j) na matriz de entrada. Por exemplo, posição da tabela de memorandos $(2, 2)$ armazenaria a soma do subarray quadrado de $(1, 1)$ até $(2, 2)$ em nosso esquema de numeração baseado em 1.

Etapa 1: Gerar a tabela de memorandos
A chave para gerar a tabela de memorandos de forma eficiente é iterar pela matriz de entrada linha por linha e usar resultados anteriores para calcular os subsequentes. Você está basicamente usando a tabela de memorandos para gerar a si mesma. Para você acompanhar, eis como ficaria a tabela de memorandos do exemplo acima citado

```
0   -2    5    5
9    9   10   12
5    6    3    6
4   13   10   11
```

Para ver como isso funciona, considere a $4, 4$ matriz tabela de memorando mostrada abaixo. Os números de 01 para 16 representam a ordem na qual iremos iterar pelos dezesseis valores de entrada e calcular os dezesseis valores da tabela de memorando. Preencher a tabela dessa forma simplificará o código. Podemos ler a entrada e preencher a tabela de memorando simultaneamente. Portanto, não há necessidade de armazenar uma matriz separada de valores de entrada. Todas as informações de que precisamos na próxima etapa estarão na tabela de memorando.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Veja como o processo funciona: Escolha uma célula na matriz. Como exemplo, usarei a célula 10. Isso representa o canto inferior direito de um sub-array de 6 células que começa na célula 1 (três linhas e duas colunas). De acordo com nossa definição de tabela de memorando, a célula 10 deve conter a soma dos valores de entrada 01, 02, 05, 06, 09, e 10. Mas lembre-se de que estamos preenchendo a tabela de memorando em ordem. Então, quando chegamos à célula 10, já temos valores corretos para células 01 até 09. Como podemos usar esses valores?

A resposta é que podemos usar três informações para calcular cada soma de submatriz:

- A célula única que forma seu canto inferior direito. Neste caso, célula 10.
- O subarray acima dele. Neste caso, é o subarray $2, 2$ começando em 01 e terminando em 06. A soma deste subarray é armazenada na célula 06.
- O subarray à esquerda dele. Neste caso, é o subarray $3, 1$ começando em 01 e terminando em 09. A soma deste subarray é armazenada na célula 09.

Então, em vez de adicionar dez células para obter a soma que estamos procurando, podemos simplesmente adicionar três células: 06, 09, e 10. E para submatrizes maiores, a economia de tempo é maior.

Se você estiver acompanhando, notará que esse processo não está muito certo. O subarray acima e o subarray esquerdo 01 se sobrepõem nas células 01 e 05. Então, temos que subtrair a soma dessas células para evitar contagem dupla. Felizmente, a quantidade que precisamos subtrair é armazenada em uma única célula, a 05. Essa quantidade contada duas vezes sempre será uma célula diagonalmente para cima e para a esquerda da nossa célula-alvo.

Agora estamos no final do Passo 1, então nossa tabela de memorando contém as somas de todos os subarrays que começam no canto superior esquerdo do array de entrada. Mas o subarray de soma máxima pode começar em uma posição diferente. Para lidar com o caso geral, precisamos do Passo 2.

Etapa 2: Avalie todas as posições iniciais e finais
Esta etapa compartilha algumas ideias com a abordagem da Busca Completa descrita acima. A diferença é que as somas do subarray já foram calculadas eficientemente na Etapa 1. Então, em vez de um algoritmo de (Busca Completa) $O(N^6)$, temos um $O(N^2)$ passo (Passo 1) seguido por um passo $O(N^4)$ (Etapa 2). Como você deve saber pelas regras da análise de algoritmos assintóticos, isso nos dá uma visão geral de tempo de execução $O(N^4)$, o que é suficiente para que esta solução seja aceita na UVa OJ.

Assim como na Busca Completa, a Etapa 2 usa quatro loops aninhados, um para cada linha inicial, coluna inicial, linha final e coluna final. Essas quatro coordenadas definem o subarray em questão. Temos que verificar cada subarray para ver se ele tem a soma máxima vista até agora.

Veja o que acontece dentro do loop mais interno:

Primeiro, recuperamos a soma memorizada que calculamos na Etapa 1 para o subarray que termina na linha e coluna atuais. Lembre-se de que isso representa a soma do subarray que começa no canto superior esquerdo da entrada. Como isso pode não ser o mesmo que o canto superior esquerdo do subarray que estamos verificando no momento, temos que ajustar a soma usando um processo semelhante ao que usamos na Etapa 1. Para ilustrar, suponha que temos a seguinte tabela de memorando concluída. Como antes, os números identificam a ordem em que as células foram calculadas.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Como exemplo, suponha que nossos loops aninhados atualmente identifiquem uma posição inicial da célula 11 (linha 3, coluna 3) e uma posição final da célula 16 (linha 4, coluna 4). Precisamos encontrar a soma do subarray $2, 2$. O valor memorizado na célula 16 nos dá a soma de todo o array, então temos que nos livrar das partes que não precisamos. Como na Etapa 1, podemos fazer isso usando os valores que estão acima e à esquerda do alvo.

Aqui está outra maneira de analisar a tabela de memorandos:

```
ZZ ZZ XX XX
ZZ ZZ XX XX
```

```
YY YY 11 12
YY YY 15 16
```

As células XX identificam o subarray que está acima do alvo. As células YY identificam o subarray que está à esquerda do alvo. E as células ZZ identificam o subarray que está acima e à esquerda (ou seja, diagonalmente para cima e à esquerda) do alvo.

Esta visualização deve ser familiar do Passo 1. Mas neste passo, precisamos subtrair em vez de adicionar, já que estamos removendo somas que não precisamos. Então, começando com a soma na célula de memorando 16, precisamos:

Subtraia o valor na célula memo 08. Isso remove a soma do subarray que está acima do alvo (os XXs e ZZs). Subtraia o valor na célula memo 14. Isso remove a soma do subarray que está à esquerda do alvo (os YYs e ZZs). Observe que subtraímos o ZZs duas vezes. Nós os contamos duas vezes, assim como contamos duas vezes o valor diagonal no Passo 1. Portanto, precisamos que adicione novamente o valor na célula de memorando 06, os ZZs. Depois que adicionamos de volta uma das cópias contadas duas vezes do ZZs, temos uma soma precisa para nosso subarray de destino. Então, apenas comparamos o resultado com nossa soma máxima atual e atualizamos o máximo, se necessário.

Quando tivermos passado por todas as coordenadas iniciais e finais possíveis, nosso valor máximo será a resposta, e simplesmente imprimiremos esse inteiro.

Importante Se o seu ambiente de programação começa as numerações em 0 (Python, JS, Java, C/C++), para implementar os algoritmos, é conveniente deixar uma linha zerada (a linha 0) e igualmente uma coluna zerada (a coluna 0). Agora as verificações passam a ser em referência à coluna e linha zeros e não a 1 como nos algoritmos. Aliás, no site acima referido o autor faz isso.

O Algoritmo final

Veja como ficou uma possível solução

```
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    se i>1
      MAT[i][j]=MAT[i][j]+MAT[i-1][j]
    fimse
    se j>1
      MAT[i][j]=MAT[i][j]+MAT[i][j-1]
    fimse
    se (i>1) AND (j>1)
      MAT[i][j]=MAT[i][j]-MAT[i-1][j-1]
    fimse
  fimpara
fimpara
ans = 0
para i de 1 a N
  para j de 1 a N
    para k de i a N // note o i
      para L de j a N // idem j
        cur = MAT[k;L]
        se i > 1
          cur=cur-MAT[i-1][L]
        fimse
        se j>1
          cur=cur-MAT[i][j-1]
        fimse
        se (i>1) AND (j>1)
          cur=cur+MAT[i-1][j-1]
        fimse
        se and<cur
          ans=cur
          salva i,j,k,L
        fimse
      fimpara
    fimpara
  fimpara
fimpara
imprima (ans,salva)
```

Se você comparar os 2 algoritmos acima, verá que são muito parecidos. Só que o primeiro (força bruta) tem 6 *paras* um dentro do outro o que caracteriza o desempenho $O(N^6)$. Já o segundo tem os mesmos 6 *paras*, mas primeiro há um dentro do outro ($O(N^2)$) e depois, há 4 *paras*. Da teoria, ao final os 2 *paras* superiores podem ser desconsiderados e o resultado final fica só com $O(N^4)$.

Um exemplo completo

Para você testar seu programa. A matriz

```
-41 -98  91  71 -42   8   3 -79 -17  16
 76 -11  46  74  44  61  42  49 -96  78
  5  -7 -86  43  -2  34  37 -60  84  74
 79   9 -72  -9  98 -56 -10 -36   3  77
-12  -6  62 -26 -57 100 -69  27  24 -99
-99  55  46 -36 -16  37  37 -58  68  42
 66 -81 -83  53  26 -57 -57 -83 -22  91
 90 -22 -46  39 -43  56  57 -15 -43 -61
-97 -61  97 -51  64 -72 -20  21 -64  66
-68  98 -48 -53 -79 -56  27  40  59  40
```

Deu como resposta 588 2, 1, 6, 10.

☞ Para você fazer

Calcule a máxima subsoma da matriz 10×10 abaixo e determine também as dimensões e posição da submatriz. Para tanto identifique o canto superior esquerdo e inferior direito da submatriz (origem=1). Os números estão entre $-600 < n \leq 600$.

```
-241 -470 -200  198 -139  162 -260 -584 -324 -566
-410 -528  -11  240 -537 -117 -417  166  -90  384
-416  348  117  -11  293 -329  126 -333 -240 -493
-139 -509 -154 -305 -410 -113  186 -456 -200 -203
 303   90  326  381   50  395 -194  186 -392  398
-539 -293 -538 -212 -406  -48  348 -234  381 -444
-251 -267 -425 -566   96 -381 -429 -516  -98 -401
 -32 -502  224  243 -195  397 -539  -81 -376  307
 358 -344  378  154 -115 -588 -277 -342  281 -116
 258  -84 -538  270  389    5  267  -16  142  209
```

Responda aqui:

soma	localização

506-75765 -

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 21/10/2024 - 12:06:52.9
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
vivxhb2a V: 1.02
75677 LUIZ FELIPE GUEDES BUCCHERI
24JOG506 - 13 _____ / _____ / _____

UVA 108 - Maior subsoma em matriz

Observação: este texto está baseado em https://www.redgreencode.com/three-ways-solve-uva-108/ do autor Duncan Smith.

O problema original - 108

Como está lá, já traduzido:
Um problema que é simples de resolver numa dimensão é muitas vezes muito mais difícil de resolver em mais de uma dimensão. Considere satisfazer uma expressão booleana na forma normal conjuntiva na qual cada conjunto consiste em exatamente 3 disjuntos. Este problema (3-SAT) é NP-completo. O problema 2-SAT é resolvido de forma bastante eficiente, no entanto. Em contraste, alguns problemas pertencem à mesma classe de complexidade independentemente da dimensionalidade do problema. Dada uma matriz bidimensional de inteiros positivos e negativos, encontre o subretângulo com o maior soma. A soma de um retângulo é a soma de todos os elementos desse retângulo. Neste problema o subretângulo com a maior soma é chamado de subretângulo máximo. Um subretângulo é qualquer submatriz contígua de tamanho 1 x 1 ou maior localizado sobre a matriz. Como exemplo o sub-retângulo máximo do array

```
0   -2   -7   0
9    2   -6   2
-4   1   -4   1
-1   8    0  -2
```

está no canto esquerdo inferior

```
9   2
-4  1
-1  8
```

cuja soma é 15

Entrada:
A entrada consiste em uma matriz N × N de inteiros. A entrada começa com um único inteiro positivo N em uma linha isolada indicando o tamanho do quadrado da matriz bidimensional. Isto é seguido por números inteiros N^2 separados por espaços em branco (novas linhas e espaços). Esses inteiros N^2 constituem a matriz em ordem de linha maior (ou seja, todos os números na primeira linha, da esquerda para a direita, depois, todos os números da segunda linha, da esquerda para a direita, etc.). N pode ser tão grande quanto 100. Os números no array estarão no intervalo $[-127, 127]$.

Saída:
A saída é a soma do sub-retângulo máximo.

Exemplo de entrada

```
4
0 -2 -7 0 9 -2 -6 2
-4 1 -4 1 -1
8 0 -2
```

Exemplo de saída
15

Pesquisa completa

A abordagem de busca completa para este problema requer que enumeremos todos os subarrays do array de entrada e calculemos a soma de cada um separadamente. Para enumerar os subarrays de uma matriz, considere que você pode identificar cada submatriz por um conjunto de quatro inteiros 1 e N: linha inicial, coluna inicial, linha final e coluna final. Você pode pensar nesses quatro inteiros como as coordenadas dos cantos superior esquerdo e inferior direito do subarray. Tendo identificado um subarray usando essa abordagem de busca completa, a maneira ingênua de encontrar sua soma é adicionar todos os seus números juntos.

Para gerar todas as coordenadas do subarray, podemos usar quatro loops aninhados, um para cada coordenada. Outros dois loops aninhados são suficientes para iterar pelas linhas e colunas do subarray e calcular sua soma. Esses seis loops aninhados nos dão uma algoritmo. Embora o limite de tempo UVa, $O(N^6)$, seja bastante frouxo para esse problema, ele não é tão tolerante. Então, precisamos de uma abordagem mais eficiente. Veja uma possível implementação do algoritmo näive

```
maximo=-999999
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    para k de i a N
      para L de 1 a N
        soma=0
        para a de i a k
          para b de j a L
            soma=soma+MAT[a][b]
          fimpara
        fimpara
        se soma>maximo
          guarda=i,j,k,L
          maximo=soma
        fimse
      fimpara
    fimpara
  fimpara
fimpara
```

Estratégia de programação dinâmica

Com a reutilização de somas de subarrays. Para problemas básicos de programação dinâmica, como UVA 108, o segredo é descobrir quais informações você pode armazenar em etapas anteriores que economizarão tempo em etapas subsequentes.

A ideia-chave para UVa 108 é que os resultados de um cálculo anterior (uma soma, neste caso) podem ser usados em cálculos subsequentes. Especificamente, uma vez que calculamos a soma de parte de um subarray, não precisamos calculá-la repetidamente quando estamos processando outros subarrays que se sobrepõem à mesma região. **Se pudermos encontrar uma maneira eficiente de armazenar e reutilizar essas somas, podemos economizar muito tempo de cálculo**. A estrutura de dados usada em programação dinâmica para esse propósito é tradicionalmente chamada de tabela de memorando .

A tabela de memorandos tem o mesmo tamanho, formato e tipo que a matriz de entrada: a matriz de inteiros. O valor inteiro na posição (i, j) na tabela de memorandos armazena a soma da submatriz cujo canto superior esquerdo é o canto superior esquerdo da matriz de entrada e cujo canto inferior direito é (i, j) na matriz de entrada. Por exemplo, posição da tabela de memorandos $(2, 2)$ armazenaria a soma do subarray quadrado de $(1, 1)$ até $(2, 2)$ em nosso esquema de numeração baseado em 1.

Etapa 1: Gerar a tabela de memorandos
A chave para gerar a tabela de memorandos de forma eficiente é iterar pela matriz de entrada linha por linha e usar resultados anteriores para calcular os subsequentes. Você está basicamente usando a tabela de memorandos para gerar a si mesma. Para você acompanhar, eis como ficaria a tabela de memorandos do exemplo acima citado

```
0   -2    5    5
9    9   10   12
5    6    3    6
4   13   10   11
```

Para ver como isso funciona, considere a $4, 4$ matriz tabela de memorando mostrada abaixo. Os números de 01 para 16 representam a ordem na qual iremos iterar pelos dezesseis valores de entrada e calcular os dezesseis valores da tabela de memorando. Preencher a tabela dessa forma simplificará o código. Podemos ler a entrada e preencher a tabela de memorando simultaneamente. Portanto, não há necessidade de armazenar uma matriz separada de valores de entrada. Todas as informações de que precisamos na próxima etapa estarão na tabela de memorando.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Veja como o processo funciona: Escolha uma célula na matriz. Como exemplo, usarei a célula 10. Isso representa o canto inferior direito de um sub-array de 6 células que começa na célula 1 (três linhas e duas colunas). De acordo com nossa definição de tabela de memorando, a célula 10 deve conter a soma dos valores de entrada 01, 02, 05, 06, 09, e 10. Mas lembre-se de que estamos preenchendo a tabela de memorando em ordem. Então, quando chegamos à célula 10, já temos valores corretos para células 01 até 09. Como podemos usar esses valores?

A resposta é que podemos usar três informações para calcular cada soma de submatriz:

- A célula única que forma seu canto inferior direito. Neste caso, célula 10.
- O subarray acima dele. Neste caso, é o subarray $2, 2$ começando em 01 e terminando em 06. A soma deste subarray é armazenada na célula 06.
- O subarray à esquerda dele. Neste caso, é o subarray $3, 1$ começando em 01 e terminando em 09. A soma deste subarray é armazenada na célula 09.

Então, em vez de adicionar dez células para obter a soma que estamos procurando, podemos simplesmente adicionar três células: 06, 09, e 10. E para submatrizes maiores, a economia de tempo é maior.

Se você estiver acompanhando, notará que esse processo não está muito certo. O subarray acima e o subarray esquerdo 01 se sobrepõem nas células 01 e 05. Então, temos que subtrair a soma dessas células para evitar contagem dupla. Felizmente, a quantidade que precisamos subtrair é armazenada em uma única célula, a 05. Essa quantidade contada duas vezes sempre será uma célula diagonalmente para cima e para a esquerda da nossa célula-alvo.

Agora estamos no final do Passo 1, então nossa tabela de memorando contém as somas de todos os subarrays que começam no canto superior esquerdo do array de entrada. Mas o subarray de soma máxima pode começar em uma posição diferente. Para lidar com o caso geral, precisamos do Passo 2.

Etapa 2: Avalie todas as posições iniciais e finais
Esta etapa compartilha algumas ideias com a abordagem da Busca Completa descrita acima. A diferença é que as somas do subarray já foram calculadas eficientemente na Etapa 1. Então, em vez de um algoritmo de (Busca Completa) $O(N^6)$, temos um $O(N^2)$ passo (Passo 1) seguido por um passo $O(N^4)$ (Etapa 2). Como você deve saber pelas regras da análise de algoritmos assintóticos, isso nos dá uma visão geral de tempo de execução $O(N^4)$, o que é suficiente para que esta solução seja aceita na UVa OJ.

Assim como na Busca Completa, a Etapa 2 usa quatro loops aninhados, um para cada linha inicial, coluna inicial, linha final e coluna final. Essas quatro coordenadas definem o subarray em questão. Temos que verificar cada subarray para ver se ele tem a soma máxima vista até agora.

Veja o que acontece dentro do loop mais interno:

Primeiro, recuperamos a soma memorizada que calculamos na Etapa 1 para o subarray que termina na linha e coluna atuais. Lembre-se de que isso representa a soma do subarray que começa no canto superior esquerdo da entrada. Como isso pode não ser o mesmo que o canto superior esquerdo do subarray que estamos verificando no momento, temos que ajustar a soma usando um processo semelhante ao que usamos na Etapa 1. Para ilustrar, suponha que temos a seguinte tabela de memorando concluída. Como antes, os números identificam a ordem em que as células foram calculadas.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Como exemplo, suponha que nossos loops aninhados atualmente identifiquem uma posição inicial da célula 11 (linha 3, coluna 3) e uma posição final da célula 16 (linha 4, coluna 4). Precisamos encontrar a soma do subarray $2, 2$. O valor memorizado na célula 16 nos dá a soma de todo o array, então temos que nos livrar das partes que não precisamos. Como na Etapa 1, podemos fazer isso usando os valores que estão acima e à esquerda do alvo.

Aqui está outra maneira de analisar a tabela de memorandos:

```
ZZ ZZ XX XX
ZZ ZZ XX XX
```

```
YY YY 11 12
YY YY 15 16
```

As células XX identificam o subarray que está acima do alvo. As células YY identificam o subarray que está à esquerda do alvo. E as células ZZ identificam o subarray que está acima e à esquerda (ou seja, diagonalmente para cima e à esquerda) do alvo.

Esta visualização deve ser familiar do Passo 1. Mas neste passo, precisamos subtrair em vez de adicionar, já que estamos removendo somas que não precisamos. Então, começando com a soma na célula de memorando 16, precisamos:

Subtraia o valor na célula memo 08. Isso remove a soma do subarray que está acima do alvo (os XXs e ZZs). Subtraia o valor na célula memo 14. Isso remove a soma do subarray que está à esquerda do alvo (os YYs e ZZs). Observe que subtraímos o ZZs duas vezes. Nós os contamos duas vezes, assim como contamos duas vezes o valor diagonal no Passo 1. Portanto, precisamos que adicione novamente o valor na célula de memorando 06, os ZZs. Depois que adicionamos de volta uma das cópias contadas duas vezes do ZZs, temos uma soma precisa para nosso subarray de destino. Então, apenas comparamos o resultado com nossa soma máxima atual e atualizamos o máximo, se necessário.

Quando tivermos passado por todas as coordenadas iniciais e finais possíveis, nosso valor máximo será a resposta, e simplesmente imprimiremos esse inteiro.

Importante Se o seu ambiente de programação começa as numerações em 0 (Python, JS, Java, C/C++), para implementar os algoritmos, é conveniente deixar uma linha zerada (a linha 0) e igualmente uma coluna zerada (a coluna 0). Agora as verificações passam a ser em referência à coluna e linha zeros e não a 1 como nos algoritmos. Aliás, no site acima referido o autor faz isso.

O Algoritmo final

Veja como ficou uma possível solução

```
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    se i>1
      MAT[i][j]=MAT[i][j]+MAT[i-1][j]
    fimse
    se j>1
      MAT[i][j]=MAT[i][j]+MAT[i][j-1]
    fimse
    se (i>1) AND (j>1)
      MAT[i][j]=MAT[i][j]-MAT[i-1][j-1]
    fimse
  fimpara
fimpara
ans = 0
para i de 1 a N
  para j de 1 a N
    para k de i a N // note o i
      para L de j a N // idem j
        cur = MAT[k;L]
        se i > 1
          cur=cur-MAT[i-1][L]
        fimse
        se j>1
          cur=cur-MAT[i][j-1]
        fimse
        se (i>1) AND (j>1)
          cur=cur+MAT[i-1][j-1]
        fimse
        se and<cur
          ans=cur
          salva i,j,k,L
        fimse
      fimpara
    fimpara
  fimpara
fimpara
imprima (ans,salva)
```

Se você comparar os 2 algoritmos acima, verá que são muito parecidos. Só que o primeiro (força bruta) tem 6 *paras* um dentro do outro o que caracteriza o desempenho $O(N^6)$. Já o segundo tem os mesmos 6 *paras*, mas primeiro há um dentro do outro ($O(N^2)$) e depois, há 4 *paras*. Da teoria, ao final os 2 *paras* superiores podem ser desconsiderados e o resultado final fica só com $O(N^4)$.

Um exemplo completo

Para você testar seu programa. A matriz

```
-41 -98  91  71 -42   8   3 -79 -17  16
 76 -11  46  74  44  61  42  49 -96  78
  5  -7 -86  43  -2  34  37 -60  84  74
 79   9 -72  -9  98 -56 -10 -36   3  77
-12  -6  62 -26 -57 100 -69  27  24 -99
-99  55  46 -36 -16  37  37 -58  68  42
 66 -81 -83  53  26 -57 -57 -83 -22  91
 90 -22 -46  39 -43  56  57 -15 -43 -61
-97 -61  97 -51  64 -72 -20  21 -64  66
-68  98 -48 -53 -79 -56  27  40  59  40
```

Deu como resposta 588 2, 1, 6, 10.

☞ Para você fazer

Calcule a máxima subsoma da matriz 10×10 abaixo e determine também as dimensões e posição da submatriz. Para tanto identifique o canto superior esquerdo e inferior direito da submatriz (origem=1). Os números estão entre $-600 < n \leq 600$.

```
-452  -25  -10  254  300  380 -396 -535    2 -428
 263 -567  117  174 -335  312  249 -562  252 -158
-214 -467  -12  329  149  -79   12  131 -202 -567
-330  247 -567   30 -217  333 -276  -20 -254 -363
-416 -336  185 -276 -214  161  109 -335   73  -80
 -50 -156 -557  -48 -328  -57 -295  196  150 -251
-146  109 -161 -392  117  280 -267  341  246  336
 244 -336 -556  -96  230  130  343  -54 -155  309
-133  -35 -247  -84  -79 -178 -431 -308  -72 -586
 233 -178  348  185 -147 -425  111  378   -9 -276
```

Responda aqui:

soma	localização

506-75677 -

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 21/10/2024 - 12:06:52.9
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
vivxhb2a V: 1.02
75684 MANUELLA LEAL DE MEIRELLES F
24JOG506 - 14 _____ / _____ / _____

UVA 108 - Maior subsoma em matriz

Observação: este texto está baseado em https://www.redgreencode.com/three-ways-solve-uva-108/ do autor Duncan Smith.

O problema original - 108

Como está lá, já traduzido:
Um problema que é simples de resolver numa dimensão é muitas vezes muito mais difícil de resolver em mais de uma dimensão. Considere satisfazer uma expressão booleana na forma normal conjuntiva na qual cada conjunto consiste em exatamente 3 disjuntos. Este problema (3-SAT) é NP-completo. O problema 2-SAT é resolvido de forma bastante eficiente, no entanto. Em contraste, alguns problemas pertencem à mesma classe de complexidade independentemente da dimensionalidade do problema. Dada uma matriz bidimensional de inteiros positivos e negativos, encontre o subretângulo com o maior soma. A soma de um retângulo é a soma de todos os elementos desse retângulo. Neste problema o subretângulo com a maior soma é chamado de subretângulo máximo. Um subretângulo é qualquer submatriz contígua de tamanho 1 x 1 ou maior localizado sobre a matriz. Como exemplo o sub-retângulo máximo do array

```
0   -2   -7   0
9    2   -6   2
-4   1   -4   1
-1   8    0  -2
```

está no canto esquerdo inferior

```
9   2
-4  1
-1  8
```

cuja soma é 15

Entrada:
A entrada consiste em uma matriz N × N de inteiros. A entrada começa com um único inteiro positivo N em uma linha isolada indicando o tamanho do quadrado da matriz bidimensional. Isto é seguido por números inteiros N^2 separados por espaços em branco (novas linhas e espaços). Esses inteiros N^2 constituem a matriz em ordem de linha maior (ou seja, todos os números na primeira linha, da esquerda para a direita, depois, todos os números da segunda linha, da esquerda para a direita, etc.). N pode ser tão grande quanto 100. Os números no array estarão no intervalo $[-127, 127]$.

Saída:
A saída é a soma do sub-retângulo máximo.

Exemplo de entrada

```
4
0 -2 -7 0 9 -2 -6 2
-4 1 -4 1 -1
8 0 -2
```

Exemplo de saída
15

Pesquisa completa

A abordagem de busca completa para este problema requer que enumeremos todos os subarrays do array de entrada e calculemos a soma de cada um separadamente. Para enumerar os subarrays de uma matriz, considere que você pode identificar cada submatriz por um conjunto de quatro inteiros 1 e N: linha inicial, coluna inicial, linha final e coluna final. Você pode pensar nesses quatro inteiros como as coordenadas dos cantos superior esquerdo e inferior direito do subarray. Tendo identificado um subarray usando essa abordagem de busca completa, a maneira ingênua de encontrar sua soma é adicionar todos os seus números juntos.

Para gerar todas as coordenadas do subarray, podemos usar quatro loops aninhados, um para cada coordenada. Outros dois loops aninhados são suficientes para iterar pelas linhas e colunas do subarray e calcular sua soma. Esses seis loops aninhados nos dão uma algoritmo. Embora o limite de tempo UVa, $O(N^6)$, seja bastante frouxo para esse problema, ele não é tão tolerante. Então, precisamos de uma abordagem mais eficiente. Veja uma possível implementação do algoritmo näive

```
maximo=-999999
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    para k de i a N
      para L de 1 a N
        soma=0
        para a de i a k
          para b de j a L
            soma=soma+MAT[a][b]
          fimpara
        fimpara
        se soma>maximo
          guarda=i,j,k,L
          maximo=soma
        fimse
      fimpara
    fimpara
  fimpara
fimpara
```

Estratégia de programação dinâmica

Com a reutilização de somas de subarrays. Para problemas básicos de programação dinâmica, como UVA 108, o segredo é descobrir quais informações você pode armazenar em etapas anteriores que economizarão tempo em etapas subsequentes.

A ideia-chave para UVa 108 é que os resultados de um cálculo anterior (uma soma, neste caso) podem ser usados em cálculos subsequentes. Especificamente, uma vez que calculamos a soma de parte de um subarray, não precisamos calculá-la repetidamente quando estamos processando outros subarrays que se sobrepõem à mesma região. **Se pudermos encontrar uma maneira eficiente de armazenar e reutilizar essas somas, podemos economizar muito tempo de cálculo**. A estrutura de dados usada em programação dinâmica para esse propósito é tradicionalmente chamada de tabela de memorando .

A tabela de memorandos tem o mesmo tamanho, formato e tipo que a matriz de entrada: a matriz de inteiros. O valor inteiro na posição (i, j) na tabela de memorandos armazena a soma da submatriz cujo canto superior esquerdo é o canto superior esquerdo da matriz de entrada e cujo canto inferior direito é (i, j) na matriz de entrada. Por exemplo, posição da tabela de memorandos $(2, 2)$ armazenaria a soma do subarray quadrado de $(1, 1)$ até $(2, 2)$ em nosso esquema de numeração baseado em 1.

Etapa 1: Gerar a tabela de memorandos
A chave para gerar a tabela de memorandos de forma eficiente é iterar pela matriz de entrada linha por linha e usar resultados anteriores para calcular os subsequentes. Você está basicamente usando a tabela de memorandos para gerar a si mesma. Para você acompanhar, eis como ficaria a tabela de memorandos do exemplo acima citado

```
0   -2    5    5
9    9   10   12
5    6    3    6
4   13   10   11
```

Para ver como isso funciona, considere a $4, 4$ matriz tabela de memorando mostrada abaixo. Os números de 01 para 16 representam a ordem na qual iremos iterar pelos dezesseis valores de entrada e calcular os dezesseis valores da tabela de memorando. Preencher a tabela dessa forma simplificará o código. Podemos ler a entrada e preencher a tabela de memorando simultaneamente. Portanto, não há necessidade de armazenar uma matriz separada de valores de entrada. Todas as informações de que precisamos na próxima etapa estarão na tabela de memorando.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Veja como o processo funciona: Escolha uma célula na matriz. Como exemplo, usarei a célula 10. Isso representa o canto inferior direito de um sub-array de 6 células que começa na célula 1 (três linhas e duas colunas). De acordo com nossa definição de tabela de memorando, a célula 10 deve conter a soma dos valores de entrada 01, 02, 05, 06, 09, e 10. Mas lembre-se de que estamos preenchendo a tabela de memorando em ordem. Então, quando chegamos à célula 10, já temos valores corretos para células 01 até 09. Como podemos usar esses valores?

A resposta é que podemos usar três informações para calcular cada soma de submatriz:

- A célula única que forma seu canto inferior direito. Neste caso, célula 10.
- O subarray acima dele. Neste caso, é o subarray $2, 2$ começando em 01 e terminando em 06. A soma deste subarray é armazenada na célula 06.
- O subarray à esquerda dele. Neste caso, é o subarray $3, 1$ começando em 01 e terminando em 09. A soma deste subarray é armazenada na célula 09.

Então, em vez de adicionar dez células para obter a soma que estamos procurando, podemos simplesmente adicionar três células: 06, 09, e 10. E para submatrizes maiores, a economia de tempo é maior.

Se você estiver acompanhando, notará que esse processo não está muito certo. O subarray acima e o subarray esquerdo 01 se sobrepõem nas células 01 e 05. Então, temos que subtrair a soma dessas células para evitar contagem dupla. Felizmente, a quantidade que precisamos subtrair é armazenada em uma única célula, a 05. Essa quantidade contada duas vezes sempre será uma célula diagonalmente para cima e para a esquerda da nossa célula-alvo.

Agora estamos no final do Passo 1, então nossa tabela de memorando contém as somas de todos os subarrays que começam no canto superior esquerdo do array de entrada. Mas o subarray de soma máxima pode começar em uma posição diferente. Para lidar com o caso geral, precisamos do Passo 2.

Etapa 2: Avalie todas as posições iniciais e finais
Esta etapa compartilha algumas ideias com a abordagem da Busca Completa descrita acima. A diferença é que as somas do subarray já foram calculadas eficientemente na Etapa 1. Então, em vez de um algoritmo de (Busca Completa) $O(N^6)$, temos um $O(N^2)$ passo (Passo 1) seguido por um passo $O(N^4)$ (Etapa 2). Como você deve saber pelas regras da análise de algoritmos assintóticos, isso nos dá uma visão geral de tempo de execução $O(N^4)$, o que é suficiente para que esta solução seja aceita na UVa OJ.

Assim como na Busca Completa, a Etapa 2 usa quatro loops aninhados, um para cada linha inicial, coluna inicial, linha final e coluna final. Essas quatro coordenadas definem o subarray em questão. Temos que verificar cada subarray para ver se ele tem a soma máxima vista até agora.

Veja o que acontece dentro do loop mais interno:

Primeiro, recuperamos a soma memorizada que calculamos na Etapa 1 para o subarray que termina na linha e coluna atuais. Lembre-se de que isso representa a soma do subarray que começa no canto superior esquerdo da entrada. Como isso pode não ser o mesmo que o canto superior esquerdo do subarray que estamos verificando no momento, temos que ajustar a soma usando um processo semelhante ao que usamos na Etapa 1. Para ilustrar, suponha que temos a seguinte tabela de memorando concluída. Como antes, os números identificam a ordem em que as células foram calculadas.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Como exemplo, suponha que nossos loops aninhados atualmente identifiquem uma posição inicial da célula 11 (linha 3, coluna 3) e uma posição final da célula 16 (linha 4, coluna 4). Precisamos encontrar a soma do subarray $2, 2$. O valor memorizado na célula 16 nos dá a soma de todo o array, então temos que nos livrar das partes que não precisamos. Como na Etapa 1, podemos fazer isso usando os valores que estão acima e à esquerda do alvo.

Aqui está outra maneira de analisar a tabela de memorandos:

```
ZZ ZZ XX XX
ZZ ZZ XX XX
```

```
YY YY 11 12
YY YY 15 16
```

As células XX identificam o subarray que está acima do alvo. As células YY identificam o subarray que está à esquerda do alvo. E as células ZZ identificam o subarray que está acima e à esquerda (ou seja, diagonalmente para cima e à esquerda) do alvo.

Esta visualização deve ser familiar do Passo 1. Mas neste passo, precisamos subtrair em vez de adicionar, já que estamos removendo somas que não precisamos. Então, começando com a soma na célula de memorando 16, precisamos:

Subtraia o valor na célula memo 08. Isso remove a soma do subarray que está acima do alvo (os XXs e ZZs). Subtraia o valor na célula memo 14. Isso remove a soma do subarray que está à esquerda do alvo (os YYs e ZZs). Observe que subtraímos o ZZs duas vezes. Nós os contamos duas vezes, assim como contamos duas vezes o valor diagonal no Passo 1. Portanto, precisamos que adicione novamente o valor na célula de memorando 06, os ZZs. Depois que adicionamos de volta uma das cópias contadas duas vezes do ZZs, temos uma soma precisa para nosso subarray de destino. Então, apenas comparamos o resultado com nossa soma máxima atual e atualizamos o máximo, se necessário.

Quando tivermos passado por todas as coordenadas iniciais e finais possíveis, nosso valor máximo será a resposta, e simplesmente imprimiremos esse inteiro.

Importante Se o seu ambiente de programação começa as numerações em 0 (Python, JS, Java, C/C++), para implementar os algoritmos, é conveniente deixar uma linha zerada (a linha 0) e igualmente uma coluna zerada (a coluna 0). Agora as verificações passam a ser em referência à coluna e linha zeros e não a 1 como nos algoritmos. Aliás, no site acima referido o autor faz isso.

O Algoritmo final

Veja como ficou uma possível solução

```
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    se i>1
      MAT[i][j]=MAT[i][j]+MAT[i-1][j]
    fimse
    se j>1
      MAT[i][j]=MAT[i][j]+MAT[i][j-1]
    fimse
    se (i>1) AND (j>1)
      MAT[i][j]=MAT[i][j]-MAT[i-1][j-1]
    fimse
  fimpara
fimpara
ans = 0
para i de 1 a N
  para j de 1 a N
    para k de i a N // note o i
      para L de j a N // idem j
        cur = MAT[k;L]
        se i > 1
          cur=cur-MAT[i-1][L]
        fimse
        se j>1
          cur=cur-MAT[i][j-1]
        fimse
        se (i>1) AND (j>1)
          cur=cur+MAT[i-1][j-1]
        fimse
        se and<cur
          ans=cur
          salva i,j,k,L
        fimse
      fimpara
    fimpara
  fimpara
fimpara
imprima (ans,salva)
```

Se você comparar os 2 algoritmos acima, verá que são muito parecidos. Só que o primeiro (força bruta) tem 6 *paras* um dentro do outro o que caracteriza o desempenho $O(N^6)$. Já o segundo tem os mesmos 6 *paras*, mas primeiro há um dentro do outro ($O(N^2)$) e depois, há 4 *paras*. Da teoria, ao final os 2 *paras* superiores podem ser desconsiderados e o resultado final fica só com $O(N^4)$.

Um exemplo completo

Para você testar seu programa. A matriz

```
-41 -98  91  71 -42   8   3 -79 -17  16
 76 -11  46  74  44  61  42  49 -96  78
  5  -7 -86  43  -2  34  37 -60  84  74
 79   9 -72  -9  98 -56 -10 -36   3  77
-12  -6  62 -26 -57 100 -69  27  24 -99
-99  55  46 -36 -16  37  37 -58  68  42
 66 -81 -83  53  26 -57 -57 -83 -22  91
 90 -22 -46  39 -43  56  57 -15 -43 -61
-97 -61  97 -51  64 -72 -20  21 -64  66
-68  98 -48 -53 -79 -56  27  40  59  40
```

Deu como resposta 588 2, 1, 6, 10.

☞ Para você fazer

Calcule a máxima subsoma da matriz 10×10 abaixo e determine também as dimensões e posição da submatriz. Para tanto identifique o canto superior esquerdo e inferior direito da submatriz (origem=1). Os números estão entre $-600 < n \leq 600$.

```
-445 -371 -362  -73 -208 -133 -240 -501  176  205
 -15 -202 -380  -84 -545 -260  155  -69 -495  258
-362  197 -186 -245 -504 -419   96   73  -27 -533
-441  108 -375  -96 -437 -249  367   65  273  192
-430  276  395 -285  -47  318 -319 -238  123 -397
 -79 -282 -290 -426  -23 -309 -584 -566 -378  228
 112  324   49 -519 -141    6 -256 -107 -456  155
-178  -35 -516    8  135  381 -423 -400  153 -126
-477 -122  386  330  127  252 -304 -311 -457 -395
  13  136  323 -371 -138  245 -534  253  -26 -401
```

Responda aqui:

soma	localização

506-75684 -

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 21/10/2024 - 12:06:52.9
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
vivxhb2a V: 1.02
75691 MARIA EDUARDA DA SILVA
24JOG506 - 15 _____ / _____ / _____

UVA 108 - Maior subsoma em matriz

Observação: este texto está baseado em https://www.redgreencode.com/three-ways-solve-uva-108/ do autor Duncan Smith.

O problema original - 108

Como está lá, já traduzido:
Um problema que é simples de resolver numa dimensão é muitas vezes muito mais difícil de resolver em mais de uma dimensão. Considere satisfazer uma expressão booleana na forma normal conjuntiva na qual cada conjunto consiste em exatamente 3 disjuntos. Este problema (3-SAT) é NP-completo. O problema 2-SAT é resolvido de forma bastante eficiente, no entanto. Em contraste, alguns problemas pertencem à mesma classe de complexidade independentemente da dimensionalidade do problema. Dada uma matriz bidimensional de inteiros positivos e negativos, encontre o subretângulo com o maior soma. A soma de um retângulo é a soma de todos os elementos desse retângulo. Neste problema o subretângulo com a maior soma é chamado de subretângulo máximo. Um subretângulo é qualquer submatriz contígua de tamanho 1 x 1 ou maior localizado sobre a matriz. Como exemplo o sub-retângulo máximo do array

```
0   -2   -7   0
9    2   -6   2
-4   1   -4   1
-1   8    0  -2
```

está no canto esquerdo inferior

```
9   2
-4  1
-1  8
```

cuja soma é 15

Entrada:
A entrada consiste em uma matriz N × N de inteiros. A entrada começa com um único inteiro positivo N em uma linha isolada indicando o tamanho do quadrado da matriz bidimensional. Isto é seguido por números inteiros N^2 separados por espaços em branco (novas linhas e espaços). Esses inteiros N^2 constituem a matriz em ordem de linha maior (ou seja, todos os números na primeira linha, da esquerda para a direita, depois, todos os números da segunda linha, da esquerda para a direita, etc.). N pode ser tão grande quanto 100. Os números no array estarão no intervalo $[-127, 127]$.

Saída:
A saída é a soma do sub-retângulo máximo.

Exemplo de entrada

```
4
0 -2 -7 0 9 -2 -6 2
-4 1 -4 1 -1
8 0 -2
```

Exemplo de saída
15

Pesquisa completa

A abordagem de busca completa para este problema requer que enumeremos todos os subarrays do array de entrada e calculemos a soma de cada um separadamente. Para enumerar os subarrays de uma matriz, considere que você pode identificar cada submatriz por um conjunto de quatro inteiros 1 e N: linha inicial, coluna inicial, linha final e coluna final. Você pode pensar nesses quatro inteiros como as coordenadas dos cantos superior esquerdo e inferior direito do subarray. Tendo identificado um subarray usando essa abordagem de busca completa, a maneira ingênua de encontrar sua soma é adicionar todos os seus números juntos.

Para gerar todas as coordenadas do subarray, podemos usar quatro loops aninhados, um para cada coordenada. Outros dois loops aninhados são suficientes para iterar pelas linhas e colunas do subarray e calcular sua soma. Esses seis loops aninhados nos dão uma algoritmo. Embora o limite de tempo UVa, $O(N^6)$, seja bastante frouxo para esse problema, ele não é tão tolerante. Então, precisamos de uma abordagem mais eficiente. Veja uma possível implementação do algoritmo näive

```
maximo=-999999
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    para k de i a N
      para L de 1 a N
        soma=0
        para a de i a k
          para b de j a L
            soma=soma+MAT[a][b]
          fimpara
        fimpara
        se soma>maximo
          guarda=i,j,k,L
          maximo=soma
        fimse
      fimpara
    fimpara
  fimpara
fimpara
```

Estratégia de programação dinâmica

Com a reutilização de somas de subarrays. Para problemas básicos de programação dinâmica, como UVA 108, o segredo é descobrir quais informações você pode armazenar em etapas anteriores que economizarão tempo em etapas subsequentes.

A ideia-chave para UVa 108 é que os resultados de um cálculo anterior (uma soma, neste caso) podem ser usados em cálculos subsequentes. Especificamente, uma vez que calculamos a soma de parte de um subarray, não precisamos calculá-la repetidamente quando estamos processando outros subarrays que se sobrepõem à mesma região. **Se pudermos encontrar uma maneira eficiente de armazenar e reutilizar essas somas, podemos economizar muito tempo de cálculo**. A estrutura de dados usada em programação dinâmica para esse propósito é tradicionalmente chamada de tabela de memorando .

A tabela de memorandos tem o mesmo tamanho, formato e tipo que a matriz de entrada: a matriz de inteiros. O valor inteiro na posição (i, j) na tabela de memorandos armazena a soma da submatriz cujo canto superior esquerdo é o canto superior esquerdo da matriz de entrada e cujo canto inferior direito é (i, j) na matriz de entrada. Por exemplo, posição da tabela de memorandos $(2, 2)$ armazenaria a soma do subarray quadrado de $(1, 1)$ até $(2, 2)$ em nosso esquema de numeração baseado em 1.

Etapa 1: Gerar a tabela de memorandos
A chave para gerar a tabela de memorandos de forma eficiente é iterar pela matriz de entrada linha por linha e usar resultados anteriores para calcular os subsequentes. Você está basicamente usando a tabela de memorandos para gerar a si mesma. Para você acompanhar, eis como ficaria a tabela de memorandos do exemplo acima citado

```
0   -2    5    5
9    9   10   12
5    6    3    6
4   13   10   11
```

Para ver como isso funciona, considere a $4, 4$ matriz tabela de memorando mostrada abaixo. Os números de 01 para 16 representam a ordem na qual iremos iterar pelos dezesseis valores de entrada e calcular os dezesseis valores da tabela de memorando. Preencher a tabela dessa forma simplificará o código. Podemos ler a entrada e preencher a tabela de memorando simultaneamente. Portanto, não há necessidade de armazenar uma matriz separada de valores de entrada. Todas as informações de que precisamos na próxima etapa estarão na tabela de memorando.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Veja como o processo funciona: Escolha uma célula na matriz. Como exemplo, usarei a célula 10. Isso representa o canto inferior direito de um sub-array de 6 células que começa na célula 1 (três linhas e duas colunas). De acordo com nossa definição de tabela de memorando, a célula 10 deve conter a soma dos valores de entrada 01, 02, 05, 06, 09, e 10. Mas lembre-se de que estamos preenchendo a tabela de memorando em ordem. Então, quando chegamos à célula 10, já temos valores corretos para células 01 até 09. Como podemos usar esses valores?

A resposta é que podemos usar três informações para calcular cada soma de submatriz:

- A célula única que forma seu canto inferior direito. Neste caso, célula 10.
- O subarray acima dele. Neste caso, é o subarray $2, 2$ começando em 01 e terminando em 06. A soma deste subarray é armazenada na célula 06.
- O subarray à esquerda dele. Neste caso, é o subarray $3, 1$ começando em 01 e terminando em 09. A soma deste subarray é armazenada na célula 09.

Então, em vez de adicionar dez células para obter a soma que estamos procurando, podemos simplesmente adicionar três células: 06, 09, e 10. E para submatrizes maiores, a economia de tempo é maior.

Se você estiver acompanhando, notará que esse processo não está muito certo. O subarray acima e o subarray esquerdo 01 se sobrepõem nas células 01 e 05. Então, temos que subtrair a soma dessas células para evitar contagem dupla. Felizmente, a quantidade que precisamos subtrair é armazenada em uma única célula, a 05. Essa quantidade contada duas vezes sempre será uma célula diagonalmente para cima e para a esquerda da nossa célula-alvo.

Agora estamos no final do Passo 1, então nossa tabela de memorando contém as somas de todos os subarrays que começam no canto superior esquerdo do array de entrada. Mas o subarray de soma máxima pode começar em uma posição diferente. Para lidar com o caso geral, precisamos do Passo 2.

Etapa 2: Avalie todas as posições iniciais e finais
Esta etapa compartilha algumas ideias com a abordagem da Busca Completa descrita acima. A diferença é que as somas do subarray já foram calculadas eficientemente na Etapa 1. Então, em vez de um algoritmo de (Busca Completa) $O(N^6)$, temos um $O(N^2)$ passo (Passo 1) seguido por um passo $O(N^4)$ (Etapa 2). Como você deve saber pelas regras da análise de algoritmos assintóticos, isso nos dá uma visão geral de tempo de execução $O(N^4)$, o que é suficiente para que esta solução seja aceita na UVa OJ.

Assim como na Busca Completa, a Etapa 2 usa quatro loops aninhados, um para cada linha inicial, coluna inicial, linha final e coluna final. Essas quatro coordenadas definem o subarray em questão. Temos que verificar cada subarray para ver se ele tem a soma máxima vista até agora.

Veja o que acontece dentro do loop mais interno:

Primeiro, recuperamos a soma memorizada que calculamos na Etapa 1 para o subarray que termina na linha e coluna atuais. Lembre-se de que isso representa a soma do subarray que começa no canto superior esquerdo da entrada. Como isso pode não ser o mesmo que o canto superior esquerdo do subarray que estamos verificando no momento, temos que ajustar a soma usando um processo semelhante ao que usamos na Etapa 1. Para ilustrar, suponha que temos a seguinte tabela de memorando concluída. Como antes, os números identificam a ordem em que as células foram calculadas.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Como exemplo, suponha que nossos loops aninhados atualmente identifiquem uma posição inicial da célula 11 (linha 3, coluna 3) e uma posição final da célula 16 (linha 4, coluna 4). Precisamos encontrar a soma do subarray $2, 2$. O valor memorizado na célula 16 nos dá a soma de todo o array, então temos que nos livrar das partes que não precisamos. Como na Etapa 1, podemos fazer isso usando os valores que estão acima e à esquerda do alvo.

Aqui está outra maneira de analisar a tabela de memorandos:

```
ZZ ZZ XX XX
ZZ ZZ XX XX
```

```
YY YY 11 12
YY YY 15 16
```

As células XX identificam o subarray que está acima do alvo. As células YY identificam o subarray que está à esquerda do alvo. E as células ZZ identificam o subarray que está acima e à esquerda (ou seja, diagonalmente para cima e à esquerda) do alvo.

Esta visualização deve ser familiar do Passo 1. Mas neste passo, precisamos subtrair em vez de adicionar, já que estamos removendo somas que não precisamos. Então, começando com a soma na célula de memorando 16, precisamos:

Subtraia o valor na célula memo 08. Isso remove a soma do subarray que está acima do alvo (os XXs e ZZs). Subtraia o valor na célula memo 14. Isso remove a soma do subarray que está à esquerda do alvo (os YYs e ZZs). Observe que subtraímos o ZZs duas vezes. Nós os contamos duas vezes, assim como contamos duas vezes o valor diagonal no Passo 1. Portanto, precisamos que adicione novamente o valor na célula de memorando 06, os ZZs. Depois que adicionamos de volta uma das cópias contadas duas vezes do ZZs, temos uma soma precisa para nosso subarray de destino. Então, apenas comparamos o resultado com nossa soma máxima atual e atualizamos o máximo, se necessário.

Quando tivermos passado por todas as coordenadas iniciais e finais possíveis, nosso valor máximo será a resposta, e simplesmente imprimiremos esse inteiro.

Importante Se o seu ambiente de programação começa as numerações em 0 (Python, JS, Java, C/C++), para implementar os algoritmos, é conveniente deixar uma linha zerada (a linha 0) e igualmente uma coluna zerada (a coluna 0). Agora as verificações passam a ser em referência à coluna e linha zeros e não a 1 como nos algoritmos. Aliás, no site acima referido o autor faz isso.

O Algoritmo final

Veja como ficou uma possível solução

```
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    se i>1
      MAT[i][j]=MAT[i][j]+MAT[i-1][j]
    fimse
    se j>1
      MAT[i][j]=MAT[i][j]+MAT[i][j-1]
    fimse
    se (i>1) AND (j>1)
      MAT[i][j]=MAT[i][j]-MAT[i-1][j-1]
    fimse
  fimpara
fimpara
ans = 0
para i de 1 a N
  para j de 1 a N
    para k de i a N // note o i
      para L de j a N // idem j
        cur = MAT[k;L]
        se i > 1
          cur=cur-MAT[i-1][L]
        fimse
        se j>1
          cur=cur-MAT[i][j-1]
        fimse
        se (i>1) AND (j>1)
          cur=cur+MAT[i-1][j-1]
        fimse
        se and<cur
          ans=cur
          salva i,j,k,L
        fimse
      fimpara
    fimpara
  fimpara
fimpara
imprima (ans,salva)
```

Se você comparar os 2 algoritmos acima, verá que são muito parecidos. Só que o primeiro (força bruta) tem 6 *paras* um dentro do outro o que caracteriza o desempenho $O(N^6)$. Já o segundo tem os mesmos 6 *paras*, mas primeiro há um dentro do outro ($O(N^2)$) e depois, há 4 *paras*. Da teoria, ao final os 2 *paras* superiores podem ser desconsiderados e o resultado final fica só com $O(N^4)$.

Um exemplo completo

Para você testar seu programa. A matriz

```
-41 -98  91  71 -42   8   3 -79 -17  16
 76 -11  46  74  44  61  42  49 -96  78
  5  -7 -86  43  -2  34  37 -60  84  74
 79   9 -72  -9  98 -56 -10 -36   3  77
-12  -6  62 -26 -57 100 -69  27  24 -99
-99  55  46 -36 -16  37  37 -58  68  42
 66 -81 -83  53  26 -57 -57 -83 -22  91
 90 -22 -46  39 -43  56  57 -15 -43 -61
-97 -61  97 -51  64 -72 -20  21 -64  66
-68  98 -48 -53 -79 -56  27  40  59  40
```

Deu como resposta 588 2, 1, 6, 10.

☞ Para você fazer

Calcule a máxima subsoma da matriz 10×10 abaixo e determine também as dimensões e posição da submatriz. Para tanto identifique o canto superior esquerdo e inferior direito da submatriz (origem=1). Os números estão entre $-600 < n \leq 600$.

```
   9 -225  243   -9 -446 -311   49   42 -432   66
-266 -171   85  123  252 -530  286    6  185  226
 283  344 -123  166   95  375 -253 -265 -164 -320
 193 -555 -327  116  203  250 -188 -542 -550  155
 115 -178  159 -180 -255 -462 -598 -472 -390  296
 365  335  160 -549 -214  -91  251   34   53  131
-430  -95  -33 -504  224 -403 -533 -445 -198  262
-274 -260  140 -207  -83 -379 -364  -59   34  224
-123 -263 -396 -480  165 -382  193   26  265  291
-421 -515 -591   58  -50  274   57  365  346 -193
```

Responda aqui:

soma	localização

506-75691 -

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 21/10/2024 - 12:06:52.9
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
vivxhb2a V: 1.02
75703 MATHEUS AITA FABRICIO DE CAR
24JOG506 - 16 _____ / _____ / _____

UVA 108 - Maior subsoma em matriz

Observação: este texto está baseado em https://www.redgreencode.com/three-ways-solve-uva-108/ do autor Duncan Smith.

O problema original - 108

Como está lá, já traduzido:
Um problema que é simples de resolver numa dimensão é muitas vezes muito mais difícil de resolver em mais de uma dimensão. Considere satisfazer uma expressão booleana na forma normal conjuntiva na qual cada conjunto consiste em exatamente 3 disjuntos. Este problema (3-SAT) é NP-completo. O problema 2-SAT é resolvido de forma bastante eficiente, no entanto. Em contraste, alguns problemas pertencem à mesma classe de complexidade independentemente da dimensionalidade do problema. Dada uma matriz bidimensional de inteiros positivos e negativos, encontre o subretângulo com o maior soma. A soma de um retângulo é a soma de todos os elementos desse retângulo. Neste problema o subretângulo com a maior soma é chamado de subretângulo máximo. Um subretângulo é qualquer submatriz contígua de tamanho 1 x 1 ou maior localizado sobre a matriz. Como exemplo o sub-retângulo máximo do array

```
0   -2   -7   0
9    2   -6   2
-4   1   -4   1
-1   8    0  -2
```

está no canto esquerdo inferior

```
9   2
-4  1
-1  8
```

cuja soma é 15

Entrada:
A entrada consiste em uma matriz N × N de inteiros. A entrada começa com um único inteiro positivo N em uma linha isolada indicando o tamanho do quadrado da matriz bidimensional. Isto é seguido por números inteiros N^2 separados por espaços em branco (novas linhas e espaços). Esses inteiros N^2 constituem a matriz em ordem de linha maior (ou seja, todos os números na primeira linha, da esquerda para a direita, depois, todos os números da segunda linha, da esquerda para a direita, etc.). N pode ser tão grande quanto 100. Os números no array estarão no intervalo $[-127, 127]$.

Saída:
A saída é a soma do sub-retângulo máximo.

Exemplo de entrada

```
4
0 -2 -7 0 9 -2 -6 2
-4 1 -4 1 -1
8 0 -2
```

Exemplo de saída
15

Pesquisa completa

A abordagem de busca completa para este problema requer que enumeremos todos os subarrays do array de entrada e calculemos a soma de cada um separadamente. Para enumerar os subarrays de uma matriz, considere que você pode identificar cada submatriz por um conjunto de quatro inteiros 1 e N: linha inicial, coluna inicial, linha final e coluna final. Você pode pensar nesses quatro inteiros como as coordenadas dos cantos superior esquerdo e inferior direito do subarray. Tendo identificado um subarray usando essa abordagem de busca completa, a maneira ingênua de encontrar sua soma é adicionar todos os seus números juntos.

Para gerar todas as coordenadas do subarray, podemos usar quatro loops aninhados, um para cada coordenada. Outros dois loops aninhados são suficientes para iterar pelas linhas e colunas do subarray e calcular sua soma. Esses seis loops aninhados nos dão uma algoritmo. Embora o limite de tempo UVa, $O(N^6)$, seja bastante frouxo para esse problema, ele não é tão tolerante. Então, precisamos de uma abordagem mais eficiente. Veja uma possível implementação do algoritmo näive

```
maximo=-999999
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    para k de i a N
      para L de 1 a N
        soma=0
        para a de i a k
          para b de j a L
            soma=soma+MAT[a][b]
          fimpara
        fimpara
        se soma>maximo
          guarda=i,j,k,L
          maximo=soma
        fimse
      fimpara
    fimpara
  fimpara
fimpara
```

Estratégia de programação dinâmica

Com a reutilização de somas de subarrays. Para problemas básicos de programação dinâmica, como UVA 108, o segredo é descobrir quais informações você pode armazenar em etapas anteriores que economizarão tempo em etapas subsequentes.

A ideia-chave para UVa 108 é que os resultados de um cálculo anterior (uma soma, neste caso) podem ser usados em cálculos subsequentes. Especificamente, uma vez que calculamos a soma de parte de um subarray, não precisamos calculá-la repetidamente quando estamos processando outros subarrays que se sobrepõem à mesma região. **Se pudermos encontrar uma maneira eficiente de armazenar e reutilizar essas somas, podemos economizar muito tempo de cálculo**. A estrutura de dados usada em programação dinâmica para esse propósito é tradicionalmente chamada de tabela de memorando .

A tabela de memorandos tem o mesmo tamanho, formato e tipo que a matriz de entrada: a matriz de inteiros. O valor inteiro na posição (i, j) na tabela de memorandos armazena a soma da submatriz cujo canto superior esquerdo é o canto superior esquerdo da matriz de entrada e cujo canto inferior direito é (i, j) na matriz de entrada. Por exemplo, posição da tabela de memorandos $(2, 2)$ armazenaria a soma do subarray quadrado de $(1, 1)$ até $(2, 2)$ em nosso esquema de numeração baseado em 1.

Etapa 1: Gerar a tabela de memorandos
A chave para gerar a tabela de memorandos de forma eficiente é iterar pela matriz de entrada linha por linha e usar resultados anteriores para calcular os subsequentes. Você está basicamente usando a tabela de memorandos para gerar a si mesma. Para você acompanhar, eis como ficaria a tabela de memorandos do exemplo acima citado

```
0   -2    5    5
9    9   10   12
5    6    3    6
4   13   10   11
```

Para ver como isso funciona, considere a $4, 4$ matriz tabela de memorando mostrada abaixo. Os números de 01 para 16 representam a ordem na qual iremos iterar pelos dezesseis valores de entrada e calcular os dezesseis valores da tabela de memorando. Preencher a tabela dessa forma simplificará o código. Podemos ler a entrada e preencher a tabela de memorando simultaneamente. Portanto, não há necessidade de armazenar uma matriz separada de valores de entrada. Todas as informações de que precisamos na próxima etapa estarão na tabela de memorando.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Veja como o processo funciona: Escolha uma célula na matriz. Como exemplo, usarei a célula 10. Isso representa o canto inferior direito de um sub-array de 6 células que começa na célula 1 (três linhas e duas colunas). De acordo com nossa definição de tabela de memorando, a célula 10 deve conter a soma dos valores de entrada 01, 02, 05, 06, 09, e 10. Mas lembre-se de que estamos preenchendo a tabela de memorando em ordem. Então, quando chegamos à célula 10, já temos valores corretos para células 01 até 09. Como podemos usar esses valores?

A resposta é que podemos usar três informações para calcular cada soma de submatriz:

- A célula única que forma seu canto inferior direito. Neste caso, célula 10.
- O subarray acima dele. Neste caso, é o subarray $2, 2$ começando em 01 e terminando em 06. A soma deste subarray é armazenada na célula 06.
- O subarray à esquerda dele. Neste caso, é o subarray $3, 1$ começando em 01 e terminando em 09. A soma deste subarray é armazenada na célula 09.

Então, em vez de adicionar dez células para obter a soma que estamos procurando, podemos simplesmente adicionar três células: 06, 09, e 10. E para submatrizes maiores, a economia de tempo é maior.

Se você estiver acompanhando, notará que esse processo não está muito certo. O subarray acima e o subarray esquerdo 01 se sobrepõem nas células 01 e 05. Então, temos que subtrair a soma dessas células para evitar contagem dupla. Felizmente, a quantidade que precisamos subtrair é armazenada em uma única célula, a 05. Essa quantidade contada duas vezes sempre será uma célula diagonalmente para cima e para a esquerda da nossa célula-alvo.

Agora estamos no final do Passo 1, então nossa tabela de memorando contém as somas de todos os subarrays que começam no canto superior esquerdo do array de entrada. Mas o subarray de soma máxima pode começar em uma posição diferente. Para lidar com o caso geral, precisamos do Passo 2.

Etapa 2: Avalie todas as posições iniciais e finais
Esta etapa compartilha algumas ideias com a abordagem da Busca Completa descrita acima. A diferença é que as somas do subarray já foram calculadas eficientemente na Etapa 1. Então, em vez de um algoritmo de (Busca Completa) $O(N^6)$, temos um $O(N^2)$ passo (Passo 1) seguido por um passo $O(N^4)$ (Etapa 2). Como você deve saber pelas regras da análise de algoritmos assintóticos, isso nos dá uma visão geral de tempo de execução $O(N^4)$, o que é suficiente para que esta solução seja aceita na UVa OJ.

Assim como na Busca Completa, a Etapa 2 usa quatro loops aninhados, um para cada linha inicial, coluna inicial, linha final e coluna final. Essas quatro coordenadas definem o subarray em questão. Temos que verificar cada subarray para ver se ele tem a soma máxima vista até agora.

Veja o que acontece dentro do loop mais interno:

Primeiro, recuperamos a soma memorizada que calculamos na Etapa 1 para o subarray que termina na linha e coluna atuais. Lembre-se de que isso representa a soma do subarray que começa no canto superior esquerdo da entrada. Como isso pode não ser o mesmo que o canto superior esquerdo do subarray que estamos verificando no momento, temos que ajustar a soma usando um processo semelhante ao que usamos na Etapa 1. Para ilustrar, suponha que temos a seguinte tabela de memorando concluída. Como antes, os números identificam a ordem em que as células foram calculadas.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Como exemplo, suponha que nossos loops aninhados atualmente identifiquem uma posição inicial da célula 11 (linha 3, coluna 3) e uma posição final da célula 16 (linha 4, coluna 4). Precisamos encontrar a soma do subarray $2, 2$. O valor memorizado na célula 16 nos dá a soma de todo o array, então temos que nos livrar das partes que não precisamos. Como na Etapa 1, podemos fazer isso usando os valores que estão acima e à esquerda do alvo.

Aqui está outra maneira de analisar a tabela de memorandos:

```
ZZ ZZ XX XX
ZZ ZZ XX XX
```

```
YY YY 11 12
YY YY 15 16
```

As células XX identificam o subarray que está acima do alvo. As células YY identificam o subarray que está à esquerda do alvo. E as células ZZ identificam o subarray que está acima e à esquerda (ou seja, diagonalmente para cima e à esquerda) do alvo.

Esta visualização deve ser familiar do Passo 1. Mas neste passo, precisamos subtrair em vez de adicionar, já que estamos removendo somas que não precisamos. Então, começando com a soma na célula de memorando 16, precisamos:

Subtraia o valor na célula memo 08. Isso remove a soma do subarray que está acima do alvo (os XXs e ZZs). Subtraia o valor na célula memo 14. Isso remove a soma do subarray que está à esquerda do alvo (os YYs e ZZs). Observe que subtraímos o ZZs duas vezes. Nós os contamos duas vezes, assim como contamos duas vezes o valor diagonal no Passo 1. Portanto, precisamos que adicione novamente o valor na célula de memorando 06, os ZZs. Depois que adicionamos de volta uma das cópias contadas duas vezes do ZZs, temos uma soma precisa para nosso subarray de destino. Então, apenas comparamos o resultado com nossa soma máxima atual e atualizamos o máximo, se necessário.

Quando tivermos passado por todas as coordenadas iniciais e finais possíveis, nosso valor máximo será a resposta, e simplesmente imprimiremos esse inteiro.

Importante Se o seu ambiente de programação começa as numerações em 0 (Python, JS, Java, C/C++), para implementar os algoritmos, é conveniente deixar uma linha zerada (a linha 0) e igualmente uma coluna zerada (a coluna 0). Agora as verificações passam a ser em referência à coluna e linha zeros e não a 1 como nos algoritmos. Aliás, no site acima referido o autor faz isso.

O Algoritmo final

Veja como ficou uma possível solução

```
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    se i>1
      MAT[i][j]=MAT[i][j]+MAT[i-1][j]
    fimse
    se j>1
      MAT[i][j]=MAT[i][j]+MAT[i][j-1]
    fimse
    se (i>1) AND (j>1)
      MAT[i][j]=MAT[i][j]-MAT[i-1][j-1]
    fimse
  fimpara
fimpara
ans = 0
para i de 1 a N
  para j de 1 a N
    para k de i a N // note o i
      para L de j a N // idem j
        cur = MAT[k;L]
        se i > 1
          cur=cur-MAT[i-1][L]
        fimse
        se j>1
          cur=cur-MAT[i][j-1]
        fimse
        se (i>1) AND (j>1)
          cur=cur+MAT[i-1][j-1]
        fimse
        se and<cur
          ans=cur
          salva i,j,k,L
        fimse
      fimpara
    fimpara
  fimpara
fimpara
imprima (ans,salva)
```

Se você comparar os 2 algoritmos acima, verá que são muito parecidos. Só que o primeiro (força bruta) tem 6 *paras* um dentro do outro o que caracteriza o desempenho $O(N^6)$. Já o segundo tem os mesmos 6 *paras*, mas primeiro há um dentro do outro ($O(N^2)$) e depois, há 4 *paras*. Da teoria, ao final os 2 *paras* superiores podem ser desconsiderados e o resultado final fica só com $O(N^4)$.

Um exemplo completo

Para você testar seu programa. A matriz

```
-41 -98  91  71 -42   8   3 -79 -17  16
 76 -11  46  74  44  61  42  49 -96  78
  5  -7 -86  43  -2  34  37 -60  84  74
 79   9 -72  -9  98 -56 -10 -36   3  77
-12  -6  62 -26 -57 100 -69  27  24 -99
-99  55  46 -36 -16  37  37 -58  68  42
 66 -81 -83  53  26 -57 -57 -83 -22  91
 90 -22 -46  39 -43  56  57 -15 -43 -61
-97 -61  97 -51  64 -72 -20  21 -64  66
-68  98 -48 -53 -79 -56  27  40  59  40
```

Deu como resposta 588 2, 1, 6, 10.

☞ Para você fazer

Calcule a máxima subsoma da matriz 10×10 abaixo e determine também as dimensões e posição da submatriz. Para tanto identifique o canto superior esquerdo e inferior direito da submatriz (origem=1). Os números estão entre $-600 < n \leq 600$.

```
-327   42 -406  204  -14  271    6 -218 -470  202
  67 -525  179 -498  -49 -101 -240 -412 -204   66
-167  168 -203 -452 -442   64  101   87 -590  387
 245 -411 -437 -433  210 -331 -109 -162 -209 -366
-567 -333 -342  160 -581 -276 -552 -112 -210  258
 275  156   64 -357 -280  116 -140 -336 -362  167
 373 -111  226 -435  272 -564 -453 -212 -470  364
 221  381  319  349 -598 -168  -92  -55  244  127
-415 -376  306  372 -438  189 -131 -170  150 -500
-377  225  157  348   -7  262  221 -423 -311 -163
```

Responda aqui:

soma	localização

506-75703 -

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 21/10/2024 - 12:06:52.9
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
vivxhb2a V: 1.02
75710 MATHEUS LEANDRO DE BITENCOUR
24JOG506 - 17 _____ / _____ / _____

UVA 108 - Maior subsoma em matriz

Observação: este texto está baseado em https://www.redgreencode.com/three-ways-solve-uva-108/ do autor Duncan Smith.

O problema original - 108

Como está lá, já traduzido:
Um problema que é simples de resolver numa dimensão é muitas vezes muito mais difícil de resolver em mais de uma dimensão. Considere satisfazer uma expressão booleana na forma normal conjuntiva na qual cada conjunto consiste em exatamente 3 disjuntos. Este problema (3-SAT) é NP-completo. O problema 2-SAT é resolvido de forma bastante eficiente, no entanto. Em contraste, alguns problemas pertencem à mesma classe de complexidade independentemente da dimensionalidade do problema. Dada uma matriz bidimensional de inteiros positivos e negativos, encontre o subretângulo com o maior soma. A soma de um retângulo é a soma de todos os elementos desse retângulo. Neste problema o subretângulo com a maior soma é chamado de subretângulo máximo. Um subretângulo é qualquer submatriz contígua de tamanho 1 x 1 ou maior localizado sobre a matriz. Como exemplo o sub-retângulo máximo do array

```
0   -2   -7   0
9    2   -6   2
-4   1   -4   1
-1   8    0  -2
```

está no canto esquerdo inferior

```
9   2
-4  1
-1  8
```

cuja soma é 15

Entrada:
A entrada consiste em uma matriz N × N de inteiros. A entrada começa com um único inteiro positivo N em uma linha isolada indicando o tamanho do quadrado da matriz bidimensional. Isto é seguido por números inteiros N^2 separados por espaços em branco (novas linhas e espaços). Esses inteiros N^2 constituem a matriz em ordem de linha maior (ou seja, todos os números na primeira linha, da esquerda para a direita, depois, todos os números da segunda linha, da esquerda para a direita, etc.). N pode ser tão grande quanto 100. Os números no array estarão no intervalo $[-127, 127]$.

Saída:
A saída é a soma do sub-retângulo máximo.

Exemplo de entrada

```
4
0 -2 -7 0 9 -2 -6 2
-4 1 -4 1 -1
8 0 -2
```

Exemplo de saída
15

Pesquisa completa

A abordagem de busca completa para este problema requer que enumeremos todos os subarrays do array de entrada e calculemos a soma de cada um separadamente. Para enumerar os subarrays de uma matriz, considere que você pode identificar cada submatriz por um conjunto de quatro inteiros 1 e N: linha inicial, coluna inicial, linha final e coluna final. Você pode pensar nesses quatro inteiros como as coordenadas dos cantos superior esquerdo e inferior direito do subarray. Tendo identificado um subarray usando essa abordagem de busca completa, a maneira ingênua de encontrar sua soma é adicionar todos os seus números juntos.

Para gerar todas as coordenadas do subarray, podemos usar quatro loops aninhados, um para cada coordenada. Outros dois loops aninhados são suficientes para iterar pelas linhas e colunas do subarray e calcular sua soma. Esses seis loops aninhados nos dão uma algoritmo. Embora o limite de tempo UVa, $O(N^6)$, seja bastante frouxo para esse problema, ele não é tão tolerante. Então, precisamos de uma abordagem mais eficiente. Veja uma possível implementação do algoritmo näive

```
maximo=-999999
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    para k de i a N
      para L de 1 a N
        soma=0
        para a de i a k
          para b de j a L
            soma=soma+MAT[a][b]
          fimpara
        fimpara
        se soma>maximo
          guarda=i,j,k,L
          maximo=soma
        fimse
      fimpara
    fimpara
  fimpara
fimpara
```

Estratégia de programação dinâmica

Com a reutilização de somas de subarrays. Para problemas básicos de programação dinâmica, como UVA 108, o segredo é descobrir quais informações você pode armazenar em etapas anteriores que economizarão tempo em etapas subsequentes.

A ideia-chave para UVa 108 é que os resultados de um cálculo anterior (uma soma, neste caso) podem ser usados em cálculos subsequentes. Especificamente, uma vez que calculamos a soma de parte de um subarray, não precisamos calculá-la repetidamente quando estamos processando outros subarrays que se sobrepõem à mesma região. **Se pudermos encontrar uma maneira eficiente de armazenar e reutilizar essas somas, podemos economizar muito tempo de cálculo**. A estrutura de dados usada em programação dinâmica para esse propósito é tradicionalmente chamada de tabela de memorando .

A tabela de memorandos tem o mesmo tamanho, formato e tipo que a matriz de entrada: a matriz de inteiros. O valor inteiro na posição (i, j) na tabela de memorandos armazena a soma da submatriz cujo canto superior esquerdo é o canto superior esquerdo da matriz de entrada e cujo canto inferior direito é (i, j) na matriz de entrada. Por exemplo, posição da tabela de memorandos $(2, 2)$ armazenaria a soma do subarray quadrado de $(1, 1)$ até $(2, 2)$ em nosso esquema de numeração baseado em 1.

Etapa 1: Gerar a tabela de memorandos
A chave para gerar a tabela de memorandos de forma eficiente é iterar pela matriz de entrada linha por linha e usar resultados anteriores para calcular os subsequentes. Você está basicamente usando a tabela de memorandos para gerar a si mesma. Para você acompanhar, eis como ficaria a tabela de memorandos do exemplo acima citado

```
0   -2    5    5
9    9   10   12
5    6    3    6
4   13   10   11
```

Para ver como isso funciona, considere a $4, 4$ matriz tabela de memorando mostrada abaixo. Os números de 01 para 16 representam a ordem na qual iremos iterar pelos dezesseis valores de entrada e calcular os dezesseis valores da tabela de memorando. Preencher a tabela dessa forma simplificará o código. Podemos ler a entrada e preencher a tabela de memorando simultaneamente. Portanto, não há necessidade de armazenar uma matriz separada de valores de entrada. Todas as informações de que precisamos na próxima etapa estarão na tabela de memorando.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Veja como o processo funciona: Escolha uma célula na matriz. Como exemplo, usarei a célula 10. Isso representa o canto inferior direito de um sub-array de 6 células que começa na célula 1 (três linhas e duas colunas). De acordo com nossa definição de tabela de memorando, a célula 10 deve conter a soma dos valores de entrada 01, 02, 05, 06, 09, e 10. Mas lembre-se de que estamos preenchendo a tabela de memorando em ordem. Então, quando chegamos à célula 10, já temos valores corretos para células 01 até 09. Como podemos usar esses valores?

A resposta é que podemos usar três informações para calcular cada soma de submatriz:

- A célula única que forma seu canto inferior direito. Neste caso, célula 10.
- O subarray acima dele. Neste caso, é o subarray $2, 2$ começando em 01 e terminando em 06. A soma deste subarray é armazenada na célula 06.
- O subarray à esquerda dele. Neste caso, é o subarray $3, 1$ começando em 01 e terminando em 09. A soma deste subarray é armazenada na célula 09.

Então, em vez de adicionar dez células para obter a soma que estamos procurando, podemos simplesmente adicionar três células: 06, 09, e 10. E para submatrizes maiores, a economia de tempo é maior.

Se você estiver acompanhando, notará que esse processo não está muito certo. O subarray acima e o subarray esquerdo 01 se sobrepõem nas células 01 e 05. Então, temos que subtrair a soma dessas células para evitar contagem dupla. Felizmente, a quantidade que precisamos subtrair é armazenada em uma única célula, a 05. Essa quantidade contada duas vezes sempre será uma célula diagonalmente para cima e para a esquerda da nossa célula-alvo.

Agora estamos no final do Passo 1, então nossa tabela de memorando contém as somas de todos os subarrays que começam no canto superior esquerdo do array de entrada. Mas o subarray de soma máxima pode começar em uma posição diferente. Para lidar com o caso geral, precisamos do Passo 2.

Etapa 2: Avalie todas as posições iniciais e finais
Esta etapa compartilha algumas ideias com a abordagem da Busca Completa descrita acima. A diferença é que as somas do subarray já foram calculadas eficientemente na Etapa 1. Então, em vez de um algoritmo de (Busca Completa) $O(N^6)$, temos um $O(N^2)$ passo (Passo 1) seguido por um passo $O(N^4)$ (Etapa 2). Como você deve saber pelas regras da análise de algoritmos assintóticos, isso nos dá uma visão geral de tempo de execução $O(N^4)$, o que é suficiente para que esta solução seja aceita na UVa OJ.

Assim como na Busca Completa, a Etapa 2 usa quatro loops aninhados, um para cada linha inicial, coluna inicial, linha final e coluna final. Essas quatro coordenadas definem o subarray em questão. Temos que verificar cada subarray para ver se ele tem a soma máxima vista até agora.

Veja o que acontece dentro do loop mais interno:

Primeiro, recuperamos a soma memorizada que calculamos na Etapa 1 para o subarray que termina na linha e coluna atuais. Lembre-se de que isso representa a soma do subarray que começa no canto superior esquerdo da entrada. Como isso pode não ser o mesmo que o canto superior esquerdo do subarray que estamos verificando no momento, temos que ajustar a soma usando um processo semelhante ao que usamos na Etapa 1. Para ilustrar, suponha que temos a seguinte tabela de memorando concluída. Como antes, os números identificam a ordem em que as células foram calculadas.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Como exemplo, suponha que nossos loops aninhados atualmente identifiquem uma posição inicial da célula 11 (linha 3, coluna 3) e uma posição final da célula 16 (linha 4, coluna 4). Precisamos encontrar a soma do subarray $2, 2$. O valor memorizado na célula 16 nos dá a soma de todo o array, então temos que nos livrar das partes que não precisamos. Como na Etapa 1, podemos fazer isso usando os valores que estão acima e à esquerda do alvo.

Aqui está outra maneira de analisar a tabela de memorandos:

```
ZZ ZZ XX XX
ZZ ZZ XX XX
```

```
YY YY 11 12
YY YY 15 16
```

As células XX identificam o subarray que está acima do alvo. As células YY identificam o subarray que está à esquerda do alvo. E as células ZZ identificam o subarray que está acima e à esquerda (ou seja, diagonalmente para cima e à esquerda) do alvo.

Esta visualização deve ser familiar do Passo 1. Mas neste passo, precisamos subtrair em vez de adicionar, já que estamos removendo somas que não precisamos. Então, começando com a soma na célula de memorando 16, precisamos:

Subtraia o valor na célula memo 08. Isso remove a soma do subarray que está acima do alvo (os XXs e ZZs). Subtraia o valor na célula memo 14. Isso remove a soma do subarray que está à esquerda do alvo (os YYs e ZZs). Observe que subtraímos o ZZs duas vezes. Nós os contamos duas vezes, assim como contamos duas vezes o valor diagonal no Passo 1. Portanto, precisamos que adicione novamente o valor na célula de memorando 06, os ZZs. Depois que adicionamos de volta uma das cópias contadas duas vezes do ZZs, temos uma soma precisa para nosso subarray de destino. Então, apenas comparamos o resultado com nossa soma máxima atual e atualizamos o máximo, se necessário.

Quando tivermos passado por todas as coordenadas iniciais e finais possíveis, nosso valor máximo será a resposta, e simplesmente imprimiremos esse inteiro.

Importante Se o seu ambiente de programação começa as numerações em 0 (Python, JS, Java, C/C++), para implementar os algoritmos, é conveniente deixar uma linha zerada (a linha 0) e igualmente uma coluna zerada (a coluna 0). Agora as verificações passam a ser em referência à coluna e linha zeros e não a 1 como nos algoritmos. Aliás, no site acima referido o autor faz isso.

O Algoritmo final

Veja como ficou uma possível solução

```
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    se i>1
      MAT[i][j]=MAT[i][j]+MAT[i-1][j]
    fimse
    se j>1
      MAT[i][j]=MAT[i][j]+MAT[i][j-1]
    fimse
    se (i>1) AND (j>1)
      MAT[i][j]=MAT[i][j]-MAT[i-1][j-1]
    fimse
  fimpara
fimpara
ans = 0
para i de 1 a N
  para j de 1 a N
    para k de i a N // note o i
      para L de j a N // idem j
        cur = MAT[k;L]
        se i > 1
          cur=cur-MAT[i-1][L]
        fimse
        se j>1
          cur=cur-MAT[i][j-1]
        fimse
        se (i>1) AND (j>1)
          cur=cur+MAT[i-1][j-1]
        fimse
        se and<cur
          ans=cur
          salva i,j,k,L
        fimse
      fimpara
    fimpara
  fimpara
fimpara
imprima (ans,salva)
```

Se você comparar os 2 algoritmos acima, verá que são muito parecidos. Só que o primeiro (força bruta) tem 6 *paras* um dentro do outro o que caracteriza o desempenho $O(N^6)$. Já o segundo tem os mesmos 6 *paras*, mas primeiro há um dentro do outro ($O(N^2)$) e depois, há 4 *paras*. Da teoria, ao final os 2 *paras* superiores podem ser desconsiderados e o resultado final fica só com $O(N^4)$.

Um exemplo completo

Para você testar seu programa. A matriz

```
-41 -98  91  71 -42   8   3 -79 -17  16
 76 -11  46  74  44  61  42  49 -96  78
  5  -7 -86  43  -2  34  37 -60  84  74
 79   9 -72  -9  98 -56 -10 -36   3  77
-12  -6  62 -26 -57 100 -69  27  24 -99
-99  55  46 -36 -16  37  37 -58  68  42
 66 -81 -83  53  26 -57 -57 -83 -22  91
 90 -22 -46  39 -43  56  57 -15 -43 -61
-97 -61  97 -51  64 -72 -20  21 -64  66
-68  98 -48 -53 -79 -56  27  40  59  40
```

Deu como resposta 588 2, 1, 6, 10.

☞ Para você fazer

Calcule a máxima subsoma da matriz 10×10 abaixo e determine também as dimensões e posição da submatriz. Para tanto identifique o canto superior esquerdo e inferior direito da submatriz (origem=1). Os números estão entre $-600 < n \leq 600$.

```
 -13 -503  -29  202 -540  281  161   76 -467   32
-105  169   -3  290 -232   98   34   22   76   39
 276 -364   15   47  195  226  280 -383 -213 -158
-232   61  286  362   31 -169   91  326 -392 -322
-115  194  -85 -484 -276  221   65  -37 -276 -556
  22 -549 -229  -40 -269  129 -316 -276  340 -384
-203  108  398 -392  -38 -598 -230 -524  398 -334
 223  121  121 -298 -501 -384  394   91 -100   30
 -32  186   69 -346  196 -101 -140  255 -585 -314
-446  385 -461  199   28 -374   24 -136 -405 -157
```

Responda aqui:

soma	localização

506-75710 -

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 21/10/2024 - 12:06:52.9
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
vivxhb2a V: 1.02
75727 MAYCON TAVARES WOLPE
24JOG506 - 18 _____ / _____ / _____

UVA 108 - Maior subsoma em matriz

Observação: este texto está baseado em https://www.redgreencode.com/three-ways-solve-uva-108/ do autor Duncan Smith.

O problema original - 108

Como está lá, já traduzido:
Um problema que é simples de resolver numa dimensão é muitas vezes muito mais difícil de resolver em mais de uma dimensão. Considere satisfazer uma expressão booleana na forma normal conjuntiva na qual cada conjunto consiste em exatamente 3 disjuntos. Este problema (3-SAT) é NP-completo. O problema 2-SAT é resolvido de forma bastante eficiente, no entanto. Em contraste, alguns problemas pertencem à mesma classe de complexidade independentemente da dimensionalidade do problema. Dada uma matriz bidimensional de inteiros positivos e negativos, encontre o subretângulo com o maior soma. A soma de um retângulo é a soma de todos os elementos desse retângulo. Neste problema o subretângulo com a maior soma é chamado de subretângulo máximo. Um subretângulo é qualquer submatriz contígua de tamanho 1 x 1 ou maior localizado sobre a matriz. Como exemplo o sub-retângulo máximo do array

```
0   -2   -7   0
9    2   -6   2
-4   1   -4   1
-1   8    0  -2
```

está no canto esquerdo inferior

```
9   2
-4  1
-1  8
```

cuja soma é 15

Entrada:
A entrada consiste em uma matriz N × N de inteiros. A entrada começa com um único inteiro positivo N em uma linha isolada indicando o tamanho do quadrado da matriz bidimensional. Isto é seguido por números inteiros N^2 separados por espaços em branco (novas linhas e espaços). Esses inteiros N^2 constituem a matriz em ordem de linha maior (ou seja, todos os números na primeira linha, da esquerda para a direita, depois, todos os números da segunda linha, da esquerda para a direita, etc.). N pode ser tão grande quanto 100. Os números no array estarão no intervalo $[-127, 127]$.

Saída:
A saída é a soma do sub-retângulo máximo.

Exemplo de entrada

```
4
0 -2 -7 0 9 -2 -6 2
-4 1 -4 1 -1
8 0 -2
```

Exemplo de saída
15

Pesquisa completa

A abordagem de busca completa para este problema requer que enumeremos todos os subarrays do array de entrada e calculemos a soma de cada um separadamente. Para enumerar os subarrays de uma matriz, considere que você pode identificar cada submatriz por um conjunto de quatro inteiros 1 e N: linha inicial, coluna inicial, linha final e coluna final. Você pode pensar nesses quatro inteiros como as coordenadas dos cantos superior esquerdo e inferior direito do subarray. Tendo identificado um subarray usando essa abordagem de busca completa, a maneira ingênua de encontrar sua soma é adicionar todos os seus números juntos.

Para gerar todas as coordenadas do subarray, podemos usar quatro loops aninhados, um para cada coordenada. Outros dois loops aninhados são suficientes para iterar pelas linhas e colunas do subarray e calcular sua soma. Esses seis loops aninhados nos dão uma algoritmo. Embora o limite de tempo UVa, $O(N^6)$, seja bastante frouxo para esse problema, ele não é tão tolerante. Então, precisamos de uma abordagem mais eficiente. Veja uma possível implementação do algoritmo näive

```
maximo=-999999
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    para k de i a N
      para L de 1 a N
        soma=0
        para a de i a k
          para b de j a L
            soma=soma+MAT[a][b]
          fimpara
        fimpara
        se soma>maximo
          guarda=i,j,k,L
          maximo=soma
        fimse
      fimpara
    fimpara
  fimpara
fimpara
```

Estratégia de programação dinâmica

Com a reutilização de somas de subarrays. Para problemas básicos de programação dinâmica, como UVA 108, o segredo é descobrir quais informações você pode armazenar em etapas anteriores que economizarão tempo em etapas subsequentes.

A ideia-chave para UVa 108 é que os resultados de um cálculo anterior (uma soma, neste caso) podem ser usados em cálculos subsequentes. Especificamente, uma vez que calculamos a soma de parte de um subarray, não precisamos calculá-la repetidamente quando estamos processando outros subarrays que se sobrepõem à mesma região. **Se pudermos encontrar uma maneira eficiente de armazenar e reutilizar essas somas, podemos economizar muito tempo de cálculo**. A estrutura de dados usada em programação dinâmica para esse propósito é tradicionalmente chamada de tabela de memorando .

A tabela de memorandos tem o mesmo tamanho, formato e tipo que a matriz de entrada: a matriz de inteiros. O valor inteiro na posição (i, j) na tabela de memorandos armazena a soma da submatriz cujo canto superior esquerdo é o canto superior esquerdo da matriz de entrada e cujo canto inferior direito é (i, j) na matriz de entrada. Por exemplo, posição da tabela de memorandos $(2, 2)$ armazenaria a soma do subarray quadrado de $(1, 1)$ até $(2, 2)$ em nosso esquema de numeração baseado em 1.

Etapa 1: Gerar a tabela de memorandos
A chave para gerar a tabela de memorandos de forma eficiente é iterar pela matriz de entrada linha por linha e usar resultados anteriores para calcular os subsequentes. Você está basicamente usando a tabela de memorandos para gerar a si mesma. Para você acompanhar, eis como ficaria a tabela de memorandos do exemplo acima citado

```
0   -2    5    5
9    9   10   12
5    6    3    6
4   13   10   11
```

Para ver como isso funciona, considere a $4, 4$ matriz tabela de memorando mostrada abaixo. Os números de 01 para 16 representam a ordem na qual iremos iterar pelos dezesseis valores de entrada e calcular os dezesseis valores da tabela de memorando. Preencher a tabela dessa forma simplificará o código. Podemos ler a entrada e preencher a tabela de memorando simultaneamente. Portanto, não há necessidade de armazenar uma matriz separada de valores de entrada. Todas as informações de que precisamos na próxima etapa estarão na tabela de memorando.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Veja como o processo funciona: Escolha uma célula na matriz. Como exemplo, usarei a célula 10. Isso representa o canto inferior direito de um sub-array de 6 células que começa na célula 1 (três linhas e duas colunas). De acordo com nossa definição de tabela de memorando, a célula 10 deve conter a soma dos valores de entrada 01, 02, 05, 06, 09, e 10. Mas lembre-se de que estamos preenchendo a tabela de memorando em ordem. Então, quando chegamos à célula 10, já temos valores corretos para células 01 até 09. Como podemos usar esses valores?

A resposta é que podemos usar três informações para calcular cada soma de submatriz:

- A célula única que forma seu canto inferior direito. Neste caso, célula 10.
- O subarray acima dele. Neste caso, é o subarray $2, 2$ começando em 01 e terminando em 06. A soma deste subarray é armazenada na célula 06.
- O subarray à esquerda dele. Neste caso, é o subarray $3, 1$ começando em 01 e terminando em 09. A soma deste subarray é armazenada na célula 09.

Então, em vez de adicionar dez células para obter a soma que estamos procurando, podemos simplesmente adicionar três células: 06, 09, e 10. E para submatrizes maiores, a economia de tempo é maior.

Se você estiver acompanhando, notará que esse processo não está muito certo. O subarray acima e o subarray esquerdo 01 se sobrepõem nas células 01 e 05. Então, temos que subtrair a soma dessas células para evitar contagem dupla. Felizmente, a quantidade que precisamos subtrair é armazenada em uma única célula, a 05. Essa quantidade contada duas vezes sempre será uma célula diagonalmente para cima e para a esquerda da nossa célula-alvo.

Agora estamos no final do Passo 1, então nossa tabela de memorando contém as somas de todos os subarrays que começam no canto superior esquerdo do array de entrada. Mas o subarray de soma máxima pode começar em uma posição diferente. Para lidar com o caso geral, precisamos do Passo 2.

Etapa 2: Avalie todas as posições iniciais e finais
Esta etapa compartilha algumas ideias com a abordagem da Busca Completa descrita acima. A diferença é que as somas do subarray já foram calculadas eficientemente na Etapa 1. Então, em vez de um algoritmo de (Busca Completa) $O(N^6)$, temos um $O(N^2)$ passo (Passo 1) seguido por um passo $O(N^4)$ (Etapa 2). Como você deve saber pelas regras da análise de algoritmos assintóticos, isso nos dá uma visão geral de tempo de execução $O(N^4)$, o que é suficiente para que esta solução seja aceita na UVa OJ.

Assim como na Busca Completa, a Etapa 2 usa quatro loops aninhados, um para cada linha inicial, coluna inicial, linha final e coluna final. Essas quatro coordenadas definem o subarray em questão. Temos que verificar cada subarray para ver se ele tem a soma máxima vista até agora.

Veja o que acontece dentro do loop mais interno:

Primeiro, recuperamos a soma memorizada que calculamos na Etapa 1 para o subarray que termina na linha e coluna atuais. Lembre-se de que isso representa a soma do subarray que começa no canto superior esquerdo da entrada. Como isso pode não ser o mesmo que o canto superior esquerdo do subarray que estamos verificando no momento, temos que ajustar a soma usando um processo semelhante ao que usamos na Etapa 1. Para ilustrar, suponha que temos a seguinte tabela de memorando concluída. Como antes, os números identificam a ordem em que as células foram calculadas.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Como exemplo, suponha que nossos loops aninhados atualmente identifiquem uma posição inicial da célula 11 (linha 3, coluna 3) e uma posição final da célula 16 (linha 4, coluna 4). Precisamos encontrar a soma do subarray $2, 2$. O valor memorizado na célula 16 nos dá a soma de todo o array, então temos que nos livrar das partes que não precisamos. Como na Etapa 1, podemos fazer isso usando os valores que estão acima e à esquerda do alvo.

Aqui está outra maneira de analisar a tabela de memorandos:

```
ZZ ZZ XX XX
ZZ ZZ XX XX
```

```
YY YY 11 12
YY YY 15 16
```

As células XX identificam o subarray que está acima do alvo. As células YY identificam o subarray que está à esquerda do alvo. E as células ZZ identificam o subarray que está acima e à esquerda (ou seja, diagonalmente para cima e à esquerda) do alvo.

Esta visualização deve ser familiar do Passo 1. Mas neste passo, precisamos subtrair em vez de adicionar, já que estamos removendo somas que não precisamos. Então, começando com a soma na célula de memorando 16, precisamos:

Subtraia o valor na célula memo 08. Isso remove a soma do subarray que está acima do alvo (os XXs e ZZs). Subtraia o valor na célula memo 14. Isso remove a soma do subarray que está à esquerda do alvo (os YYs e ZZs). Observe que subtraímos o ZZs duas vezes. Nós os contamos duas vezes, assim como contamos duas vezes o valor diagonal no Passo 1. Portanto, precisamos que adicione novamente o valor na célula de memorando 06, os ZZs. Depois que adicionamos de volta uma das cópias contadas duas vezes do ZZs, temos uma soma precisa para nosso subarray de destino. Então, apenas comparamos o resultado com nossa soma máxima atual e atualizamos o máximo, se necessário.

Quando tivermos passado por todas as coordenadas iniciais e finais possíveis, nosso valor máximo será a resposta, e simplesmente imprimiremos esse inteiro.

Importante Se o seu ambiente de programação começa as numerações em 0 (Python, JS, Java, C/C++), para implementar os algoritmos, é conveniente deixar uma linha zerada (a linha 0) e igualmente uma coluna zerada (a coluna 0). Agora as verificações passam a ser em referência à coluna e linha zeros e não a 1 como nos algoritmos. Aliás, no site acima referido o autor faz isso.

O Algoritmo final

Veja como ficou uma possível solução

```
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    se i>1
      MAT[i][j]=MAT[i][j]+MAT[i-1][j]
    fimse
    se j>1
      MAT[i][j]=MAT[i][j]+MAT[i][j-1]
    fimse
    se (i>1) AND (j>1)
      MAT[i][j]=MAT[i][j]-MAT[i-1][j-1]
    fimse
  fimpara
fimpara
ans = 0
para i de 1 a N
  para j de 1 a N
    para k de i a N // note o i
      para L de j a N // idem j
        cur = MAT[k;L]
        se i > 1
          cur=cur-MAT[i-1][L]
        fimse
        se j>1
          cur=cur-MAT[i][j-1]
        fimse
        se (i>1) AND (j>1)
          cur=cur+MAT[i-1][j-1]
        fimse
        se and<cur
          ans=cur
          salva i,j,k,L
        fimse
      fimpara
    fimpara
  fimpara
fimpara
imprima (ans,salva)
```

Se você comparar os 2 algoritmos acima, verá que são muito parecidos. Só que o primeiro (força bruta) tem 6 *paras* um dentro do outro o que caracteriza o desempenho $O(N^6)$. Já o segundo tem os mesmos 6 *paras*, mas primeiro há um dentro do outro ($O(N^2)$) e depois, há 4 *paras*. Da teoria, ao final os 2 *paras* superiores podem ser desconsiderados e o resultado final fica só com $O(N^4)$.

Um exemplo completo

Para você testar seu programa. A matriz

```
-41 -98  91  71 -42   8   3 -79 -17  16
 76 -11  46  74  44  61  42  49 -96  78
  5  -7 -86  43  -2  34  37 -60  84  74
 79   9 -72  -9  98 -56 -10 -36   3  77
-12  -6  62 -26 -57 100 -69  27  24 -99
-99  55  46 -36 -16  37  37 -58  68  42
 66 -81 -83  53  26 -57 -57 -83 -22  91
 90 -22 -46  39 -43  56  57 -15 -43 -61
-97 -61  97 -51  64 -72 -20  21 -64  66
-68  98 -48 -53 -79 -56  27  40  59  40
```

Deu como resposta 588 2, 1, 6, 10.

☞ Para você fazer

Calcule a máxima subsoma da matriz 10×10 abaixo e determine também as dimensões e posição da submatriz. Para tanto identifique o canto superior esquerdo e inferior direito da submatriz (origem=1). Os números estão entre $-600 < n \leq 600$.

```
-429 -183    0  -94 -235 -568 -243  111   14  372
  75 -449   77 -520  257  322  140 -211 -507 -468
 -61 -249 -192  229 -118 -425 -443  201   11 -461
-594  253  -60   68  -38 -531 -213  198  384  376
 282 -488   96 -507 -588 -264 -205 -505 -229 -428
 -85 -487 -140 -453 -291 -276 -265   60  -57 -279
   0   91 -394 -372  362  -56   15  133 -220 -596
  93  132  356  -17  303 -299 -454   70 -446  104
 -44  200    7 -209 -116  125  -91  265  328 -598
  26 -330  158  146  378 -315 -267  127 -228 -337
```

Responda aqui:

soma	localização

506-75727 -

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação
21/10/2024 - 12:06:52.9
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
vivxhb2a V: 1.02
75734 PEDRO TOLEDO LEAL
24JOG506 - 19 _____ / _____ / _____

UVA 108 - Maior subsoma em matriz

Observação: este texto está baseado em https://www.redgreencode.com/three-ways-solve-uva-108/ do autor Duncan Smith.

O problema original - 108

Como está lá, já traduzido:
Um problema que é simples de resolver numa dimensão é muitas vezes muito mais difícil de resolver em mais de uma dimensão. Considere satisfazer uma expressão booleana na forma normal conjuntiva na qual cada conjunto consiste em exatamente 3 disjuntos. Este problema (3-SAT) é NP-completo. O problema 2-SAT é resolvido de forma bastante eficiente, no entanto. Em contraste, alguns problemas pertencem à mesma classe de complexidade independentemente da dimensionalidade do problema. Dada uma matriz bidimensional de inteiros positivos e negativos, encontre o subretângulo com o maior soma. A soma de um retângulo é a soma de todos os elementos desse retângulo. Neste problema o subretângulo com a maior soma é chamado de subretângulo máximo. Um subretângulo é qualquer submatriz contígua de tamanho 1 x 1 ou maior localizado sobre a matriz. Como exemplo o sub-retângulo máximo do array

```
0   -2   -7   0
9    2   -6   2
-4   1   -4   1
-1   8    0  -2
```

está no canto esquerdo inferior

```
9   2
-4  1
-1  8
```

cuja soma é 15

Entrada:
A entrada consiste em uma matriz N × N de inteiros. A entrada começa com um único inteiro positivo N em uma linha isolada indicando o tamanho do quadrado da matriz bidimensional. Isto é seguido por números inteiros N^2 separados por espaços em branco (novas linhas e espaços). Esses inteiros N^2 constituem a matriz em ordem de linha maior (ou seja, todos os números na primeira linha, da esquerda para a direita, depois, todos os números da segunda linha, da esquerda para a direita, etc.). N pode ser tão grande quanto 100. Os números no array estarão no intervalo $[-127, 127]$.

Saída:
A saída é a soma do sub-retângulo máximo.

Exemplo de entrada

```
4
0 -2 -7 0 9 -2 -6 2
-4 1 -4 1 -1
8 0 -2
```

Exemplo de saída
15

Pesquisa completa

A abordagem de busca completa para este problema requer que enumeremos todos os subarrays do array de entrada e calculemos a soma de cada um separadamente. Para enumerar os subarrays de uma matriz, considere que você pode identificar cada submatriz por um conjunto de quatro inteiros 1 e N: linha inicial, coluna inicial, linha final e coluna final. Você pode pensar nesses quatro inteiros como as coordenadas dos cantos superior esquerdo e inferior direito do subarray. Tendo identificado um subarray usando essa abordagem de busca completa, a maneira ingênua de encontrar sua soma é adicionar todos os seus números juntos.

Para gerar todas as coordenadas do subarray, podemos usar quatro loops aninhados, um para cada coordenada. Outros dois loops aninhados são suficientes para iterar pelas linhas e colunas do subarray e calcular sua soma. Esses seis loops aninhados nos dão uma algoritmo. Embora o limite de tempo UVa, $O(N^6)$, seja bastante frouxo para esse problema, ele não é tão tolerante. Então, precisamos de uma abordagem mais eficiente. Veja uma possível implementação do algoritmo näive

```
maximo=-999999
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    para k de i a N
      para L de 1 a N
        soma=0
        para a de i a k
          para b de j a L
            soma=soma+MAT[a][b]
          fimpara
        fimpara
        se soma>maximo
          guarda=i,j,k,L
          maximo=soma
        fimse
      fimpara
    fimpara
  fimpara
fimpara
```

Estratégia de programação dinâmica

Com a reutilização de somas de subarrays. Para problemas básicos de programação dinâmica, como UVA 108, o segredo é descobrir quais informações você pode armazenar em etapas anteriores que economizarão tempo em etapas subsequentes.

A ideia-chave para UVa 108 é que os resultados de um cálculo anterior (uma soma, neste caso) podem ser usados em cálculos subsequentes. Especificamente, uma vez que calculamos a soma de parte de um subarray, não precisamos calculá-la repetidamente quando estamos processando outros subarrays que se sobrepõem à mesma região. **Se pudermos encontrar uma maneira eficiente de armazenar e reutilizar essas somas, podemos economizar muito tempo de cálculo**. A estrutura de dados usada em programação dinâmica para esse propósito é tradicionalmente chamada de tabela de memorando .

A tabela de memorandos tem o mesmo tamanho, formato e tipo que a matriz de entrada: a matriz de inteiros. O valor inteiro na posição (i, j) na tabela de memorandos armazena a soma da submatriz cujo canto superior esquerdo é o canto superior esquerdo da matriz de entrada e cujo canto inferior direito é (i, j) na matriz de entrada. Por exemplo, posição da tabela de memorandos $(2, 2)$ armazenaria a soma do subarray quadrado de $(1, 1)$ até $(2, 2)$ em nosso esquema de numeração baseado em 1.

Etapa 1: Gerar a tabela de memorandos
A chave para gerar a tabela de memorandos de forma eficiente é iterar pela matriz de entrada linha por linha e usar resultados anteriores para calcular os subsequentes. Você está basicamente usando a tabela de memorandos para gerar a si mesma. Para você acompanhar, eis como ficaria a tabela de memorandos do exemplo acima citado

```
0   -2    5    5
9    9   10   12
5    6    3    6
4   13   10   11
```

Para ver como isso funciona, considere a $4, 4$ matriz tabela de memorando mostrada abaixo. Os números de 01 para 16 representam a ordem na qual iremos iterar pelos dezesseis valores de entrada e calcular os dezesseis valores da tabela de memorando. Preencher a tabela dessa forma simplificará o código. Podemos ler a entrada e preencher a tabela de memorando simultaneamente. Portanto, não há necessidade de armazenar uma matriz separada de valores de entrada. Todas as informações de que precisamos na próxima etapa estarão na tabela de memorando.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Veja como o processo funciona: Escolha uma célula na matriz. Como exemplo, usarei a célula 10. Isso representa o canto inferior direito de um sub-array de 6 células que começa na célula 1 (três linhas e duas colunas). De acordo com nossa definição de tabela de memorando, a célula 10 deve conter a soma dos valores de entrada 01, 02, 05, 06, 09, e 10. Mas lembre-se de que estamos preenchendo a tabela de memorando em ordem. Então, quando chegamos à célula 10, já temos valores corretos para células 01 até 09. Como podemos usar esses valores?

A resposta é que podemos usar três informações para calcular cada soma de submatriz:

- A célula única que forma seu canto inferior direito. Neste caso, célula 10.
- O subarray acima dele. Neste caso, é o subarray $2, 2$ começando em 01 e terminando em 06. A soma deste subarray é armazenada na célula 06.
- O subarray à esquerda dele. Neste caso, é o subarray $3, 1$ começando em 01 e terminando em 09. A soma deste subarray é armazenada na célula 09.

Então, em vez de adicionar dez células para obter a soma que estamos procurando, podemos simplesmente adicionar três células: 06, 09, e 10. E para submatrizes maiores, a economia de tempo é maior.

Se você estiver acompanhando, notará que esse processo não está muito certo. O subarray acima e o subarray esquerdo 01 se sobrepõem nas células 01 e 05. Então, temos que subtrair a soma dessas células para evitar contagem dupla. Felizmente, a quantidade que precisamos subtrair é armazenada em uma única célula, a 05. Essa quantidade contada duas vezes sempre será uma célula diagonalmente para cima e para a esquerda da nossa célula-alvo.

Agora estamos no final do Passo 1, então nossa tabela de memorando contém as somas de todos os subarrays que começam no canto superior esquerdo do array de entrada. Mas o subarray de soma máxima pode começar em uma posição diferente. Para lidar com o caso geral, precisamos do Passo 2.

Etapa 2: Avalie todas as posições iniciais e finais
Esta etapa compartilha algumas ideias com a abordagem da Busca Completa descrita acima. A diferença é que as somas do subarray já foram calculadas eficientemente na Etapa 1. Então, em vez de um algoritmo de (Busca Completa) $O(N^6)$, temos um $O(N^2)$ passo (Passo 1) seguido por um passo $O(N^4)$ (Etapa 2). Como você deve saber pelas regras da análise de algoritmos assintóticos, isso nos dá uma visão geral de tempo de execução $O(N^4)$, o que é suficiente para que esta solução seja aceita na UVa OJ.

Assim como na Busca Completa, a Etapa 2 usa quatro loops aninhados, um para cada linha inicial, coluna inicial, linha final e coluna final. Essas quatro coordenadas definem o subarray em questão. Temos que verificar cada subarray para ver se ele tem a soma máxima vista até agora.

Veja o que acontece dentro do loop mais interno:

Primeiro, recuperamos a soma memorizada que calculamos na Etapa 1 para o subarray que termina na linha e coluna atuais. Lembre-se de que isso representa a soma do subarray que começa no canto superior esquerdo da entrada. Como isso pode não ser o mesmo que o canto superior esquerdo do subarray que estamos verificando no momento, temos que ajustar a soma usando um processo semelhante ao que usamos na Etapa 1. Para ilustrar, suponha que temos a seguinte tabela de memorando concluída. Como antes, os números identificam a ordem em que as células foram calculadas.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Como exemplo, suponha que nossos loops aninhados atualmente identifiquem uma posição inicial da célula 11 (linha 3, coluna 3) e uma posição final da célula 16 (linha 4, coluna 4). Precisamos encontrar a soma do subarray $2, 2$. O valor memorizado na célula 16 nos dá a soma de todo o array, então temos que nos livrar das partes que não precisamos. Como na Etapa 1, podemos fazer isso usando os valores que estão acima e à esquerda do alvo.

Aqui está outra maneira de analisar a tabela de memorandos:

```
ZZ ZZ XX XX
ZZ ZZ XX XX
```

```
YY YY 11 12
YY YY 15 16
```

As células XX identificam o subarray que está acima do alvo. As células YY identificam o subarray que está à esquerda do alvo. E as células ZZ identificam o subarray que está acima e à esquerda (ou seja, diagonalmente para cima e à esquerda) do alvo.

Esta visualização deve ser familiar do Passo 1. Mas neste passo, precisamos subtrair em vez de adicionar, já que estamos removendo somas que não precisamos. Então, começando com a soma na célula de memorando 16, precisamos:

Subtraia o valor na célula memo 08. Isso remove a soma do subarray que está acima do alvo (os XXs e ZZs). Subtraia o valor na célula memo 14. Isso remove a soma do subarray que está à esquerda do alvo (os YYs e ZZs). Observe que subtraímos o ZZs duas vezes. Nós os contamos duas vezes, assim como contamos duas vezes o valor diagonal no Passo 1. Portanto, precisamos que adicione novamente o valor na célula de memorando 06, os ZZs. Depois que adicionamos de volta uma das cópias contadas duas vezes do ZZs, temos uma soma precisa para nosso subarray de destino. Então, apenas comparamos o resultado com nossa soma máxima atual e atualizamos o máximo, se necessário.

Quando tivermos passado por todas as coordenadas iniciais e finais possíveis, nosso valor máximo será a resposta, e simplesmente imprimiremos esse inteiro.

Importante Se o seu ambiente de programação começa as numerações em 0 (Python, JS, Java, C/C++), para implementar os algoritmos, é conveniente deixar uma linha zerada (a linha 0) e igualmente uma coluna zerada (a coluna 0). Agora as verificações passam a ser em referência à coluna e linha zeros e não a 1 como nos algoritmos. Aliás, no site acima referido o autor faz isso.

O Algoritmo final

Veja como ficou uma possível solução

```
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    se i>1
      MAT[i][j]=MAT[i][j]+MAT[i-1][j]
    fimse
    se j>1
      MAT[i][j]=MAT[i][j]+MAT[i][j-1]
    fimse
    se (i>1) AND (j>1)
      MAT[i][j]=MAT[i][j]-MAT[i-1][j-1]
    fimse
  fimpara
fimpara
ans = 0
para i de 1 a N
  para j de 1 a N
    para k de i a N // note o i
      para L de j a N // idem j
        cur = MAT[k;L]
        se i > 1
          cur=cur-MAT[i-1][L]
        fimse
        se j>1
          cur=cur-MAT[i][j-1]
        fimse
        se (i>1) AND (j>1)
          cur=cur+MAT[i-1][j-1]
        fimse
        se and<cur
          ans=cur
          salva i,j,k,L
        fimse
      fimpara
    fimpara
  fimpara
fimpara
imprima (ans,salva)
```

Se você comparar os 2 algoritmos acima, verá que são muito parecidos. Só que o primeiro (força bruta) tem 6 *paras* um dentro do outro o que caracteriza o desempenho $O(N^6)$. Já o segundo tem os mesmos 6 *paras*, mas primeiro há um dentro do outro ($O(N^2)$) e depois, há 4 *paras*. Da teoria, ao final os 2 *paras* superiores podem ser desconsiderados e o resultado final fica só com $O(N^4)$.

Um exemplo completo

Para você testar seu programa. A matriz

```
-41 -98  91  71 -42   8   3 -79 -17  16
 76 -11  46  74  44  61  42  49 -96  78
  5  -7 -86  43  -2  34  37 -60  84  74
 79   9 -72  -9  98 -56 -10 -36   3  77
-12  -6  62 -26 -57 100 -69  27  24 -99
-99  55  46 -36 -16  37  37 -58  68  42
 66 -81 -83  53  26 -57 -57 -83 -22  91
 90 -22 -46  39 -43  56  57 -15 -43 -61
-97 -61  97 -51  64 -72 -20  21 -64  66
-68  98 -48 -53 -79 -56  27  40  59  40
```

Deu como resposta 588 2, 1, 6, 10.

☞ Para você fazer

Calcule a máxima subsoma da matriz 10×10 abaixo e determine também as dimensões e posição da submatriz. Para tanto identifique o canto superior esquerdo e inferior direito da submatriz (origem=1). Os números estão entre $-600 < n \leq 600$.

```
-274 -547   87 -397 -535  249 -433  130  106  100
-511   93  235   85   73  207 -263 -256 -515 -355
 274 -314  239 -109  214 -153 -230 -100 -496 -283
-531   17 -480  182 -176  195 -117 -501 -421  331
-361 -222 -494  341   95 -337 -518 -410  209  110
  45 -189 -223  376 -210  281 -428 -399 -162 -163
 229  189 -115 -406 -212   56 -454 -349  301  -46
-441  247   92 -481 -573  -21 -231 -440  130 -562
  68 -168  123  252  162 -512  -55 -446  264 -297
-479  301 -163 -174  222 -179 -555  379   84  180
```

Responda aqui:

soma	localização

506-75734 -

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 21/10/2024 - 12:06:52.9
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
vivxhb2a V: 1.02
75741 PHELIPE GABRIEL LIMA DA SILV
24JOG506 - 20 _____ / _____ / _____

UVA 108 - Maior subsoma em matriz

Observação: este texto está baseado em https://www.redgreencode.com/three-ways-solve-uva-108/ do autor Duncan Smith.

O problema original - 108

Como está lá, já traduzido:
Um problema que é simples de resolver numa dimensão é muitas vezes muito mais difícil de resolver em mais de uma dimensão. Considere satisfazer uma expressão booleana na forma normal conjuntiva na qual cada conjunto consiste em exatamente 3 disjuntos. Este problema (3-SAT) é NP-completo. O problema 2-SAT é resolvido de forma bastante eficiente, no entanto. Em contraste, alguns problemas pertencem à mesma classe de complexidade independentemente da dimensionalidade do problema. Dada uma matriz bidimensional de inteiros positivos e negativos, encontre o subretângulo com o maior soma. A soma de um retângulo é a soma de todos os elementos desse retângulo. Neste problema o subretângulo com a maior soma é chamado de subretângulo máximo. Um subretângulo é qualquer submatriz contígua de tamanho 1 x 1 ou maior localizado sobre a matriz. Como exemplo o sub-retângulo máximo do array

```
0   -2   -7   0
9    2   -6   2
-4   1   -4   1
-1   8    0  -2
```

está no canto esquerdo inferior

```
9   2
-4  1
-1  8
```

cuja soma é 15

Entrada:
A entrada consiste em uma matriz N × N de inteiros. A entrada começa com um único inteiro positivo N em uma linha isolada indicando o tamanho do quadrado da matriz bidimensional. Isto é seguido por números inteiros N^2 separados por espaços em branco (novas linhas e espaços). Esses inteiros N^2 constituem a matriz em ordem de linha maior (ou seja, todos os números na primeira linha, da esquerda para a direita, depois, todos os números da segunda linha, da esquerda para a direita, etc.). N pode ser tão grande quanto 100. Os números no array estarão no intervalo $[-127, 127]$.

Saída:
A saída é a soma do sub-retângulo máximo.

Exemplo de entrada

```
4
0 -2 -7 0 9 -2 -6 2
-4 1 -4 1 -1
8 0 -2
```

Exemplo de saída
15

Pesquisa completa

A abordagem de busca completa para este problema requer que enumeremos todos os subarrays do array de entrada e calculemos a soma de cada um separadamente. Para enumerar os subarrays de uma matriz, considere que você pode identificar cada submatriz por um conjunto de quatro inteiros 1 e N: linha inicial, coluna inicial, linha final e coluna final. Você pode pensar nesses quatro inteiros como as coordenadas dos cantos superior esquerdo e inferior direito do subarray. Tendo identificado um subarray usando essa abordagem de busca completa, a maneira ingênua de encontrar sua soma é adicionar todos os seus números juntos.

Para gerar todas as coordenadas do subarray, podemos usar quatro loops aninhados, um para cada coordenada. Outros dois loops aninhados são suficientes para iterar pelas linhas e colunas do subarray e calcular sua soma. Esses seis loops aninhados nos dão uma algoritmo. Embora o limite de tempo UVa, $O(N^6)$, seja bastante frouxo para esse problema, ele não é tão tolerante. Então, precisamos de uma abordagem mais eficiente. Veja uma possível implementação do algoritmo näive

```
maximo=-999999
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    para k de i a N
      para L de 1 a N
        soma=0
        para a de i a k
          para b de j a L
            soma=soma+MAT[a][b]
          fimpara
        fimpara
        se soma>maximo
          guarda=i,j,k,L
          maximo=soma
        fimse
      fimpara
    fimpara
  fimpara
fimpara
```

Estratégia de programação dinâmica

Com a reutilização de somas de subarrays. Para problemas básicos de programação dinâmica, como UVA 108, o segredo é descobrir quais informações você pode armazenar em etapas anteriores que economizarão tempo em etapas subsequentes.

A ideia-chave para UVa 108 é que os resultados de um cálculo anterior (uma soma, neste caso) podem ser usados em cálculos subsequentes. Especificamente, uma vez que calculamos a soma de parte de um subarray, não precisamos calculá-la repetidamente quando estamos processando outros subarrays que se sobrepõem à mesma região. **Se pudermos encontrar uma maneira eficiente de armazenar e reutilizar essas somas, podemos economizar muito tempo de cálculo**. A estrutura de dados usada em programação dinâmica para esse propósito é tradicionalmente chamada de tabela de memorando .

A tabela de memorandos tem o mesmo tamanho, formato e tipo que a matriz de entrada: a matriz de inteiros. O valor inteiro na posição (i, j) na tabela de memorandos armazena a soma da submatriz cujo canto superior esquerdo é o canto superior esquerdo da matriz de entrada e cujo canto inferior direito é (i, j) na matriz de entrada. Por exemplo, posição da tabela de memorandos $(2, 2)$ armazenaria a soma do subarray quadrado de $(1, 1)$ até $(2, 2)$ em nosso esquema de numeração baseado em 1.

Etapa 1: Gerar a tabela de memorandos
A chave para gerar a tabela de memorandos de forma eficiente é iterar pela matriz de entrada linha por linha e usar resultados anteriores para calcular os subsequentes. Você está basicamente usando a tabela de memorandos para gerar a si mesma. Para você acompanhar, eis como ficaria a tabela de memorandos do exemplo acima citado

```
0   -2    5    5
9    9   10   12
5    6    3    6
4   13   10   11
```

Para ver como isso funciona, considere a $4, 4$ matriz tabela de memorando mostrada abaixo. Os números de 01 para 16 representam a ordem na qual iremos iterar pelos dezesseis valores de entrada e calcular os dezesseis valores da tabela de memorando. Preencher a tabela dessa forma simplificará o código. Podemos ler a entrada e preencher a tabela de memorando simultaneamente. Portanto, não há necessidade de armazenar uma matriz separada de valores de entrada. Todas as informações de que precisamos na próxima etapa estarão na tabela de memorando.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Veja como o processo funciona: Escolha uma célula na matriz. Como exemplo, usarei a célula 10. Isso representa o canto inferior direito de um sub-array de 6 células que começa na célula 1 (três linhas e duas colunas). De acordo com nossa definição de tabela de memorando, a célula 10 deve conter a soma dos valores de entrada 01, 02, 05, 06, 09, e 10. Mas lembre-se de que estamos preenchendo a tabela de memorando em ordem. Então, quando chegamos à célula 10, já temos valores corretos para células 01 até 09. Como podemos usar esses valores?

A resposta é que podemos usar três informações para calcular cada soma de submatriz:

- A célula única que forma seu canto inferior direito. Neste caso, célula 10.
- O subarray acima dele. Neste caso, é o subarray $2, 2$ começando em 01 e terminando em 06. A soma deste subarray é armazenada na célula 06.
- O subarray à esquerda dele. Neste caso, é o subarray $3, 1$ começando em 01 e terminando em 09. A soma deste subarray é armazenada na célula 09.

Então, em vez de adicionar dez células para obter a soma que estamos procurando, podemos simplesmente adicionar três células: 06, 09, e 10. E para submatrizes maiores, a economia de tempo é maior.

Se você estiver acompanhando, notará que esse processo não está muito certo. O subarray acima e o subarray esquerdo 01 se sobrepõem nas células 01 e 05. Então, temos que subtrair a soma dessas células para evitar contagem dupla. Felizmente, a quantidade que precisamos subtrair é armazenada em uma única célula, a 05. Essa quantidade contada duas vezes sempre será uma célula diagonalmente para cima e para a esquerda da nossa célula-alvo.

Agora estamos no final do Passo 1, então nossa tabela de memorando contém as somas de todos os subarrays que começam no canto superior esquerdo do array de entrada. Mas o subarray de soma máxima pode começar em uma posição diferente. Para lidar com o caso geral, precisamos do Passo 2.

Etapa 2: Avalie todas as posições iniciais e finais
Esta etapa compartilha algumas ideias com a abordagem da Busca Completa descrita acima. A diferença é que as somas do subarray já foram calculadas eficientemente na Etapa 1. Então, em vez de um algoritmo de (Busca Completa) $O(N^6)$, temos um $O(N^2)$ passo (Passo 1) seguido por um passo $O(N^4)$ (Etapa 2). Como você deve saber pelas regras da análise de algoritmos assintóticos, isso nos dá uma visão geral de tempo de execução $O(N^4)$, o que é suficiente para que esta solução seja aceita na UVa OJ.

Assim como na Busca Completa, a Etapa 2 usa quatro loops aninhados, um para cada linha inicial, coluna inicial, linha final e coluna final. Essas quatro coordenadas definem o subarray em questão. Temos que verificar cada subarray para ver se ele tem a soma máxima vista até agora.

Veja o que acontece dentro do loop mais interno:

Primeiro, recuperamos a soma memorizada que calculamos na Etapa 1 para o subarray que termina na linha e coluna atuais. Lembre-se de que isso representa a soma do subarray que começa no canto superior esquerdo da entrada. Como isso pode não ser o mesmo que o canto superior esquerdo do subarray que estamos verificando no momento, temos que ajustar a soma usando um processo semelhante ao que usamos na Etapa 1. Para ilustrar, suponha que temos a seguinte tabela de memorando concluída. Como antes, os números identificam a ordem em que as células foram calculadas.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Como exemplo, suponha que nossos loops aninhados atualmente identifiquem uma posição inicial da célula 11 (linha 3, coluna 3) e uma posição final da célula 16 (linha 4, coluna 4). Precisamos encontrar a soma do subarray $2, 2$. O valor memorizado na célula 16 nos dá a soma de todo o array, então temos que nos livrar das partes que não precisamos. Como na Etapa 1, podemos fazer isso usando os valores que estão acima e à esquerda do alvo.

Aqui está outra maneira de analisar a tabela de memorandos:

```
ZZ ZZ XX XX
ZZ ZZ XX XX
```

```
YY YY 11 12
YY YY 15 16
```

As células XX identificam o subarray que está acima do alvo. As células YY identificam o subarray que está à esquerda do alvo. E as células ZZ identificam o subarray que está acima e à esquerda (ou seja, diagonalmente para cima e à esquerda) do alvo.

Esta visualização deve ser familiar do Passo 1. Mas neste passo, precisamos subtrair em vez de adicionar, já que estamos removendo somas que não precisamos. Então, começando com a soma na célula de memorando 16, precisamos:

Subtraia o valor na célula memo 08. Isso remove a soma do subarray que está acima do alvo (os XXs e ZZs). Subtraia o valor na célula memo 14. Isso remove a soma do subarray que está à esquerda do alvo (os YYs e ZZs). Observe que subtraímos o ZZs duas vezes. Nós os contamos duas vezes, assim como contamos duas vezes o valor diagonal no Passo 1. Portanto, precisamos que adicione novamente o valor na célula de memorando 06, os ZZs. Depois que adicionamos de volta uma das cópias contadas duas vezes do ZZs, temos uma soma precisa para nosso subarray de destino. Então, apenas comparamos o resultado com nossa soma máxima atual e atualizamos o máximo, se necessário.

Quando tivermos passado por todas as coordenadas iniciais e finais possíveis, nosso valor máximo será a resposta, e simplesmente imprimiremos esse inteiro.

Importante Se o seu ambiente de programação começa as numerações em 0 (Python, JS, Java, C/C++), para implementar os algoritmos, é conveniente deixar uma linha zerada (a linha 0) e igualmente uma coluna zerada (a coluna 0). Agora as verificações passam a ser em referência à coluna e linha zeros e não a 1 como nos algoritmos. Aliás, no site acima referido o autor faz isso.

O Algoritmo final

Veja como ficou uma possível solução

```
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    se i>1
      MAT[i][j]=MAT[i][j]+MAT[i-1][j]
    fimse
    se j>1
      MAT[i][j]=MAT[i][j]+MAT[i][j-1]
    fimse
    se (i>1) AND (j>1)
      MAT[i][j]=MAT[i][j]-MAT[i-1][j-1]
    fimse
  fimpara
fimpara
ans = 0
para i de 1 a N
  para j de 1 a N
    para k de i a N // note o i
      para L de j a N // idem j
        cur = MAT[k;L]
        se i > 1
          cur=cur-MAT[i-1][L]
        fimse
        se j>1
          cur=cur-MAT[i][j-1]
        fimse
        se (i>1) AND (j>1)
          cur=cur+MAT[i-1][j-1]
        fimse
        se and<cur
          ans=cur
          salva i,j,k,L
        fimse
      fimpara
    fimpara
  fimpara
fimpara
imprima (ans,salva)
```

Se você comparar os 2 algoritmos acima, verá que são muito parecidos. Só que o primeiro (força bruta) tem 6 *paras* um dentro do outro o que caracteriza o desempenho $O(N^6)$. Já o segundo tem os mesmos 6 *paras*, mas primeiro há um dentro do outro ($O(N^2)$) e depois, há 4 *paras*. Da teoria, ao final os 2 *paras* superiores podem ser desconsiderados e o resultado final fica só com $O(N^4)$.

Um exemplo completo

Para você testar seu programa. A matriz

```
-41 -98  91  71 -42   8   3 -79 -17  16
 76 -11  46  74  44  61  42  49 -96  78
  5  -7 -86  43  -2  34  37 -60  84  74
 79   9 -72  -9  98 -56 -10 -36   3  77
-12  -6  62 -26 -57 100 -69  27  24 -99
-99  55  46 -36 -16  37  37 -58  68  42
 66 -81 -83  53  26 -57 -57 -83 -22  91
 90 -22 -46  39 -43  56  57 -15 -43 -61
-97 -61  97 -51  64 -72 -20  21 -64  66
-68  98 -48 -53 -79 -56  27  40  59  40
```

Deu como resposta 588 2, 1, 6, 10.

☞ Para você fazer

Calcule a máxima subsoma da matriz 10×10 abaixo e determine também as dimensões e posição da submatriz. Para tanto identifique o canto superior esquerdo e inferior direito da submatriz (origem=1). Os números estão entre $-600 < n \leq 600$.

```
 174 -115 -134  335   23  154  -88   93 -232 -581
-406  -30  -45  -85  269 -466 -422 -596 -215  264
-134  148 -491 -189 -536 -374  393 -174  303 -596
 139 -487 -538 -430  250 -226    0 -592  -12   72
-290   98 -572  251 -481  129 -511 -361  -36   61
 225  334  -32  -65 -546 -487  138  325 -164 -417
 288  189 -436 -277 -500  115 -587  -82  158 -257
-432 -177 -170 -444 -213 -196  -97  400  134 -367
-171  174 -120  340 -269  130  -50 -458 -230  318
-239 -444  369   15  386  301 -157 -479 -564 -540
```

Responda aqui:

soma	localização

506-75741 -

CEP-UFPR-UTFPR-PUC/Pr-UP Sistemas de Informação 21/10/2024 - 12:06:52.9
Matemática aplicada Prof Dr P Kantek (pkantek@gmail.com)
vivxhb2a V: 1.02
75758 REBECA CABRAL DOS SANTOS
24JOG506 - 21 _____ / _____ / _____

UVA 108 - Maior subsoma em matriz

Observação: este texto está baseado em https://www.redgreencode.com/three-ways-solve-uva-108/ do autor Duncan Smith.

O problema original - 108

Como está lá, já traduzido:
Um problema que é simples de resolver numa dimensão é muitas vezes muito mais difícil de resolver em mais de uma dimensão. Considere satisfazer uma expressão booleana na forma normal conjuntiva na qual cada conjunto consiste em exatamente 3 disjuntos. Este problema (3-SAT) é NP-completo. O problema 2-SAT é resolvido de forma bastante eficiente, no entanto. Em contraste, alguns problemas pertencem à mesma classe de complexidade independentemente da dimensionalidade do problema. Dada uma matriz bidimensional de inteiros positivos e negativos, encontre o subretângulo com o maior soma. A soma de um retângulo é a soma de todos os elementos desse retângulo. Neste problema o subretângulo com a maior soma é chamado de subretângulo máximo. Um subretângulo é qualquer submatriz contígua de tamanho 1 x 1 ou maior localizado sobre a matriz. Como exemplo o sub-retângulo máximo do array

```
0   -2   -7   0
9    2   -6   2
-4   1   -4   1
-1   8    0  -2
```

está no canto esquerdo inferior

```
9   2
-4  1
-1  8
```

cuja soma é 15

Entrada:
A entrada consiste em uma matriz N × N de inteiros. A entrada começa com um único inteiro positivo N em uma linha isolada indicando o tamanho do quadrado da matriz bidimensional. Isto é seguido por números inteiros N^2 separados por espaços em branco (novas linhas e espaços). Esses inteiros N^2 constituem a matriz em ordem de linha maior (ou seja, todos os números na primeira linha, da esquerda para a direita, depois, todos os números da segunda linha, da esquerda para a direita, etc.). N pode ser tão grande quanto 100. Os números no array estarão no intervalo $[-127, 127]$.

Saída:
A saída é a soma do sub-retângulo máximo.

Exemplo de entrada

```
4
0 -2 -7 0 9 -2 -6 2
-4 1 -4 1 -1
8 0 -2
```

Exemplo de saída
15

Pesquisa completa

A abordagem de busca completa para este problema requer que enumeremos todos os subarrays do array de entrada e calculemos a soma de cada um separadamente. Para enumerar os subarrays de uma matriz, considere que você pode identificar cada submatriz por um conjunto de quatro inteiros 1 e N: linha inicial, coluna inicial, linha final e coluna final. Você pode pensar nesses quatro inteiros como as coordenadas dos cantos superior esquerdo e inferior direito do subarray. Tendo identificado um subarray usando essa abordagem de busca completa, a maneira ingênua de encontrar sua soma é adicionar todos os seus números juntos.

Para gerar todas as coordenadas do subarray, podemos usar quatro loops aninhados, um para cada coordenada. Outros dois loops aninhados são suficientes para iterar pelas linhas e colunas do subarray e calcular sua soma. Esses seis loops aninhados nos dão uma algoritmo. Embora o limite de tempo UVa, $O(N^6)$, seja bastante frouxo para esse problema, ele não é tão tolerante. Então, precisamos de uma abordagem mais eficiente. Veja uma possível implementação do algoritmo näive

```
maximo=-999999
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    para k de i a N
      para L de 1 a N
        soma=0
        para a de i a k
          para b de j a L
            soma=soma+MAT[a][b]
          fimpara
        fimpara
        se soma>maximo
          guarda=i,j,k,L
          maximo=soma
        fimse
      fimpara
    fimpara
  fimpara
fimpara
```

Estratégia de programação dinâmica

Com a reutilização de somas de subarrays. Para problemas básicos de programação dinâmica, como UVA 108, o segredo é descobrir quais informações você pode armazenar em etapas anteriores que economizarão tempo em etapas subsequentes.

A ideia-chave para UVa 108 é que os resultados de um cálculo anterior (uma soma, neste caso) podem ser usados em cálculos subsequentes. Especificamente, uma vez que calculamos a soma de parte de um subarray, não precisamos calculá-la repetidamente quando estamos processando outros subarrays que se sobrepõem à mesma região. **Se pudermos encontrar uma maneira eficiente de armazenar e reutilizar essas somas, podemos economizar muito tempo de cálculo**. A estrutura de dados usada em programação dinâmica para esse propósito é tradicionalmente chamada de tabela de memorando .

A tabela de memorandos tem o mesmo tamanho, formato e tipo que a matriz de entrada: a matriz de inteiros. O valor inteiro na posição (i, j) na tabela de memorandos armazena a soma da submatriz cujo canto superior esquerdo é o canto superior esquerdo da matriz de entrada e cujo canto inferior direito é (i, j) na matriz de entrada. Por exemplo, posição da tabela de memorandos $(2, 2)$ armazenaria a soma do subarray quadrado de $(1, 1)$ até $(2, 2)$ em nosso esquema de numeração baseado em 1.

Etapa 1: Gerar a tabela de memorandos
A chave para gerar a tabela de memorandos de forma eficiente é iterar pela matriz de entrada linha por linha e usar resultados anteriores para calcular os subsequentes. Você está basicamente usando a tabela de memorandos para gerar a si mesma. Para você acompanhar, eis como ficaria a tabela de memorandos do exemplo acima citado

```
0   -2    5    5
9    9   10   12
5    6    3    6
4   13   10   11
```

Para ver como isso funciona, considere a $4, 4$ matriz tabela de memorando mostrada abaixo. Os números de 01 para 16 representam a ordem na qual iremos iterar pelos dezesseis valores de entrada e calcular os dezesseis valores da tabela de memorando. Preencher a tabela dessa forma simplificará o código. Podemos ler a entrada e preencher a tabela de memorando simultaneamente. Portanto, não há necessidade de armazenar uma matriz separada de valores de entrada. Todas as informações de que precisamos na próxima etapa estarão na tabela de memorando.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Veja como o processo funciona: Escolha uma célula na matriz. Como exemplo, usarei a célula 10. Isso representa o canto inferior direito de um sub-array de 6 células que começa na célula 1 (três linhas e duas colunas). De acordo com nossa definição de tabela de memorando, a célula 10 deve conter a soma dos valores de entrada 01, 02, 05, 06, 09, e 10. Mas lembre-se de que estamos preenchendo a tabela de memorando em ordem. Então, quando chegamos à célula 10, já temos valores corretos para células 01 até 09. Como podemos usar esses valores?

A resposta é que podemos usar três informações para calcular cada soma de submatriz:

- A célula única que forma seu canto inferior direito. Neste caso, célula 10.
- O subarray acima dele. Neste caso, é o subarray $2, 2$ começando em 01 e terminando em 06. A soma deste subarray é armazenada na célula 06.
- O subarray à esquerda dele. Neste caso, é o subarray $3, 1$ começando em 01 e terminando em 09. A soma deste subarray é armazenada na célula 09.

Então, em vez de adicionar dez células para obter a soma que estamos procurando, podemos simplesmente adicionar três células: 06, 09, e 10. E para submatrizes maiores, a economia de tempo é maior.

Se você estiver acompanhando, notará que esse processo não está muito certo. O subarray acima e o subarray esquerdo 01 se sobrepõem nas células 01 e 05. Então, temos que subtrair a soma dessas células para evitar contagem dupla. Felizmente, a quantidade que precisamos subtrair é armazenada em uma única célula, a 05. Essa quantidade contada duas vezes sempre será uma célula diagonalmente para cima e para a esquerda da nossa célula-alvo.

Agora estamos no final do Passo 1, então nossa tabela de memorando contém as somas de todos os subarrays que começam no canto superior esquerdo do array de entrada. Mas o subarray de soma máxima pode começar em uma posição diferente. Para lidar com o caso geral, precisamos do Passo 2.

Etapa 2: Avalie todas as posições iniciais e finais
Esta etapa compartilha algumas ideias com a abordagem da Busca Completa descrita acima. A diferença é que as somas do subarray já foram calculadas eficientemente na Etapa 1. Então, em vez de um algoritmo de (Busca Completa) $O(N^6)$, temos um $O(N^2)$ passo (Passo 1) seguido por um passo $O(N^4)$ (Etapa 2). Como você deve saber pelas regras da análise de algoritmos assintóticos, isso nos dá uma visão geral de tempo de execução $O(N^4)$, o que é suficiente para que esta solução seja aceita na UVa OJ.

Assim como na Busca Completa, a Etapa 2 usa quatro loops aninhados, um para cada linha inicial, coluna inicial, linha final e coluna final. Essas quatro coordenadas definem o subarray em questão. Temos que verificar cada subarray para ver se ele tem a soma máxima vista até agora.

Veja o que acontece dentro do loop mais interno:

Primeiro, recuperamos a soma memorizada que calculamos na Etapa 1 para o subarray que termina na linha e coluna atuais. Lembre-se de que isso representa a soma do subarray que começa no canto superior esquerdo da entrada. Como isso pode não ser o mesmo que o canto superior esquerdo do subarray que estamos verificando no momento, temos que ajustar a soma usando um processo semelhante ao que usamos na Etapa 1. Para ilustrar, suponha que temos a seguinte tabela de memorando concluída. Como antes, os números identificam a ordem em que as células foram calculadas.

```
01 02 03 04
05 06 07 08
09 10 11 12
13 14 15 16
```

Como exemplo, suponha que nossos loops aninhados atualmente identifiquem uma posição inicial da célula 11 (linha 3, coluna 3) e uma posição final da célula 16 (linha 4, coluna 4). Precisamos encontrar a soma do subarray $2, 2$. O valor memorizado na célula 16 nos dá a soma de todo o array, então temos que nos livrar das partes que não precisamos. Como na Etapa 1, podemos fazer isso usando os valores que estão acima e à esquerda do alvo.

Aqui está outra maneira de analisar a tabela de memorandos:

```
ZZ ZZ XX XX
ZZ ZZ XX XX
```

```
YY YY 11 12
YY YY 15 16
```

As células XX identificam o subarray que está acima do alvo. As células YY identificam o subarray que está à esquerda do alvo. E as células ZZ identificam o subarray que está acima e à esquerda (ou seja, diagonalmente para cima e à esquerda) do alvo.

Esta visualização deve ser familiar do Passo 1. Mas neste passo, precisamos subtrair em vez de adicionar, já que estamos removendo somas que não precisamos. Então, começando com a soma na célula de memorando 16, precisamos:

Subtraia o valor na célula memo 08. Isso remove a soma do subarray que está acima do alvo (os XXs e ZZs). Subtraia o valor na célula memo 14. Isso remove a soma do subarray que está à esquerda do alvo (os YYs e ZZs). Observe que subtraímos o ZZs duas vezes. Nós os contamos duas vezes, assim como contamos duas vezes o valor diagonal no Passo 1. Portanto, precisamos que adicione novamente o valor na célula de memorando 06, os ZZs. Depois que adicionamos de volta uma das cópias contadas duas vezes do ZZs, temos uma soma precisa para nosso subarray de destino. Então, apenas comparamos o resultado com nossa soma máxima atual e atualizamos o máximo, se necessário.

Quando tivermos passado por todas as coordenadas iniciais e finais possíveis, nosso valor máximo será a resposta, e simplesmente imprimiremos esse inteiro.

Importante Se o seu ambiente de programação começa as numerações em 0 (Python, JS, Java, C/C++), para implementar os algoritmos, é conveniente deixar uma linha zerada (a linha 0) e igualmente uma coluna zerada (a coluna 0). Agora as verificações passam a ser em referência à coluna e linha zeros e não a 1 como nos algoritmos. Aliás, no site acima referido o autor faz isso.

O Algoritmo final

Veja como ficou uma possível solução

```
obtem MAT // de dimensoes N x N
para i de 1 a N
  para j de 1 a N
    se i>1
      MAT[i][j]=MAT[i][j]+MAT[i-1][j]
    fimse
    se j>1
      MAT[i][j]=MAT[i][j]+MAT[i][j-1]
    fimse
    se (i>1) AND (j>1)
      MAT[i][j]=MAT[i][j]-MAT[i-1][j-1]
    fimse
  fimpara
fimpara
ans = 0
para i de 1 a N
  para j de 1 a N
    para k de i a N // note o i
      para L de j a N // idem j
        cur = MAT[k;L]
        se i > 1
          cur=cur-MAT[i-1][L]
        fimse
        se j>1
          cur=cur-MAT[i][j-1]
        fimse
        se (i>1) AND (j>1)
          cur=cur+MAT[i-1][j-1]
        fimse
        se and<cur
          ans=cur
          salva i,j,k,L
        fimse
      fimpara
    fimpara
  fimpara
fimpara
imprima (ans,salva)
```

Se você comparar os 2 algoritmos acima, verá que são muito parecidos. Só que o primeiro (força bruta) tem 6 *paras* um dentro do outro o que caracteriza o desempenho $O(N^6)$. Já o segundo tem os mesmos 6 *paras*, mas primeiro há um dentro do outro ($O(N^2)$) e depois, há 4 *paras*. Da teoria, ao final os 2 *paras* superiores podem ser desconsiderados e o resultado final fica só com $O(N^4)$.

Um exemplo completo

Para você testar seu programa. A matriz

```
-41 -98  91  71 -42   8   3 -79 -17  16
 76 -11  46  74  44  61  42  49 -96  78
  5  -7 -86  43  -2  34  37 -60  84  74
 79   9 -72  -9  98 -56 -10 -36   3  77
-12  -6  62 -26 -57 100 -69  27  24 -99
-99  55  46 -36 -16  37  37 -58  68  42
 66 -81 -83  53  26 -57 -57 -83 -22  91
 90 -22 -46  39 -43  56  57 -15 -43 -61
-97 -61  97 -51  64 -72 -20  21 -64  66
-68  98 -48 -53 -79 -56  27  40  59  40
```

Deu como resposta 588 2, 1, 6, 10.

☞ Para você fazer

Calcule a máxima subsoma da matriz 10×10 abaixo e determine também as dimensões e posição da submatriz. Para tanto identifique o canto superior esquerdo e inferior direito da submatriz (origem=1). Os números estão entre $-600 < n \leq 600$.

```
 -58  -58 -233 -283  271 -385 -109 -378  314  145
-576  147 -467  346 -295 -101 -523  173  -60   51
  68   39 -326 -423 -410 -430  252  170   49  233
-519  324 -319  334  318  178 -447  270  142 -170
  89  292 -411 -225  105   21  -24 -172  304 -460
  -7  322 -104  148 -509 -298 -584   21 -333   60
  74 -530  343 -520 -309 -461  -48  392 -483   49
-510 -514   99 -350 -491   43  -99  309 -413 -276
  97 -270 -127  244 -517 -333  159  -58 -305 -476
-227 -490  304 -418  216  389 -104  -37   54  309
```

Responda aqui:

soma	localização

506-75758 -