U Positivo UTFPR PUCPr 11/02/2019 – 12:33:59.5 – Prof Dr P Kantek (pkantek@gmail.com) O problema da partição VIVO231a V: 3.22 – 1
–
Exercício : 1 _____ / _____ / _____

O problema da partição

Seja o problema da partição. Dado um conjunto de n inteiros, $J = \{x_1, x_2, ...x_n\}$, achar uma maneira de dividi-los em 2 conjuntos, digamos o conjunto I e o conjunto F, de tal maneira que a soma de I seja igual à soma de F.

Por exemplo, se o conjunto de dados for $X = 10, 5, 6, 9, 7, 8, 2$ *e* 3 haverá dois subconjuntos disjuntos de X que tenham a mesma soma ?

Deve-se começar somando os valores de X e encontrando a metade. Esta é a semi-soma. No exemplo, a soma de X é 50, e a semi-soma é 25.

Neste caso, verifica-se que existem os conjuntos I, F. I=10,6,9 e F=5,7,8,2,3.

Quando a quantidade de itens é pequena, a solução pode ser encontrada com alguma facilidade. Entretanto, se existem por exemplo 1000 itens, a busca pode ser bem mais demorada. Na verdade, a primeira vista a solução cabal deste exercício passa pela construção de todos os subconjuntos do conjunto dado.

Relembrando, quando um dos conjuntos formados tiver peso igual à metade do total, o restante dos elementos que não fazem parte deste conjunto, formará a outra metade e o problema estará resolvido.

Nesse sentido, parece que a solução está em construir todos os subconjuntos de um conjunto dado. Isto é trivial, o problema está em outro lugar, vejamos qual.

Se o conjunto possui um elemento $C = \{A\}$, ele tem 2 subconjuntos: o próprio C e o subconjunto vazio (ϕ).

Se o conjunto possui dois elementos ($C = \{A, B\}$), eis os seus subconjuntos: ϕ, {A}, {B} e {A, B}, em total de 4.

Se o conjunto possui três elementos ($C = \{A, B, C\}$), seus subconjuntos são: ϕ, {A}, {B}, {C}, {A, B}, {A, C}, {B, C} e {A, B, C}, no total de 8.

Se o conjunto possui quatro elementos, seus subconjuntos são 16.

Finalmente, se o conjunto possui n elementos, seus subconjuntos são 2^n. Eis aqui um algoritmo exponencial. Veja-se como ele se comporta, supondo um computador capaz de testar 1000 casos por segundo (cada caso demora 1 milésimo de segundo):

objetos n	subconjuntos 2^n	tempo processamento a 1000 comparações/seg
5	32	32 milisseg
10	1.024	1 segundo
15	32.768	32 segundos
20	1.048.576	17 minutos
30	1.073.741.824	298 horas
40	1.099.511. 628.000	34,8 anos
50	$1,125899907 \times 10^{15}$	35.702 anos
100	$1,2676506 \times 10^{30}$	40.196.936. 850.000.000 milênios

Como vimos na aula, para este tipo de algoritmo, o computador – por mais rápido que seja – só consegue resolver instâncias pequenas. Suponham que precisamos resolver uma instância com 10.000 objetos.

Neste caso, há que se buscar um algoritmo que fuja da maldição exponencial. Para este problema este algoritmo existe e é o seguinte:

```
 1: Algoritmo PARTICAO X
 2: inteiro N ← tamanho(X)
 3: inteiro ME
 4: inteiro SEMI ← somatório dos elementos de X
 5: se SEMI não é par
 6:     ERRO "Este algoritmo não pode ser executado"
 7: fim{se}
 8: ME ← SEMI ÷ 2
 9: inteiro RES [N] [ME] ← 0
10: RES [1][X[1]] ← X[1]
11: para I de 2 até N
12:     para J de 1 até ME
13:         se RES [I - 1][J] ≠ 0
14:             RES [I][J] ← RES [I - 1][J]
15:         senão
16:             se (J = X[I])
17:                 RES [I][J] ← X[I]
18:             senão
19:                 se J > X[I] ∧ RES [I-1][J-X[I]] ≠ 0
20:                     RES[I][J] ← (RES[I-1][J-X[I]]×10)+X[I]
21:                     se J=ME
22:                         escreva ("A solucao e ",RES[I][J])
23:                         abandone
24:                     fim{se}
25:                 fim{se}
26:             fim{se}
27:         fim{se}
28:     fim{para}
29: fim{para}
```

Obs: Se implementado em C ou Java (origem=0), além de mudar todos os índices, não esquecer de alterar as linhas 16 e 19 do algoritmo acima, substituindo J por $J + 1$.

Acompanhe um chinês. Seja $X = 3, 6, 7, 1, 2, 4, 5$. Veja como fica RES

```
   1 2 3  4  5  6  7  8  9 10  11  12  13  14
 +-----------------------------------------------
1| 0 0 3  0  0  0  0  0  0  0   0   0   0   0
2| 0 0 3  0  0  6  0  0 36  0   0   0   0   0
3| 0 0 3  0  0  6  7  0 36 37   0   0  67   0
4| 1 0 3 31  0  6  7 71 36 37 371   0  67 671
5| 1 2 3 31 32  6  7 71 36 37 371 372  67 671
6| 1 2 3 31 32  6  7 71 36 37 371 372  67 671
7| 1 2 3 31 32  6  7 71 36 37 371 372  67 671
```

Note que a resposta do problema está na primeira linha que tem a última coluna preenchida. Neste caso, em RES [4;14] e é 6+7+1 = 14 quilos.

Este algoritmo SEMPRE encontra a solução, se ela existir. A questão é: quanto tempo ele demora ?

Lembrando que o numero de subconjuntos de conjunto de n elementos é 2^n. Neste caso, em particular a complexidade é O(N×SEMI) e já que SEMI >> N, percebe-se que a complexidade está limitada a $O(SEMI^2)$. Esta expressão tem a "cara"de um polinômio.

Só que não devemos esquecer que a expressão da complexidade O, deve ser feita em termos do tamanho da entrada e não do seu conteúdo.

Chamando "S"ao maior elemento da série X, pode-se demonstrar que para este problema $N.SEMI < N \times S^2 < S^3$.

Quando um algoritmo tem sua complexidade expressa através de um polinômio em função de N e de S (o maior valor da entrada), diz-se que o algoritmo tem complexidade PSEUDO-POLINOMIAL.

A generalização deste problema (dividir ou partir um conjunto em N partes cada uma com uma certa característica) é importante para os sistemas de criptografia de chave pública.

Mais um exemplo completo:

X = { 4 3 5 3 5 5 2 1 }

Terá como tabela:

```
--  -  -  -  -  -  --  --  --  --  ---  ---  ---  ---  ---
1-  0  0  0  4  0   0   0   0   0    0    0    0    0    0
2-  0  0  3  4  0   0  43   0   0    0    0    0    0    0
3-  0  0  3  4  5   0  43  35  45    0    0  435    0    0
4-  0  0  3  4  5  33  43  35  45  433  353  435    0    0
5-  0  0  3  4  5  33  43  35  45  433  353  435  355  455
6-  0  0  3  4  5  33  43  35  45  433  353  435  355  455
7-  0  2  3  4  5  33  43  35  45  433  353  435  355  455
8-  1  2  3  4  5  33  43  35  45  433  353  435  355  455
```

A resposta correta é 455

☞ Para você fazer

Seja o seguinte conjunto de X = { 4 1 6 2 3 3 1 4 1 1 1 1 }

seguindo o algoritmo acima, determine:

Se a partição pode ou não ser atingida, e em caso positivo, determine quais números fazem parte do primeiro conjunto disjunto.

Obs: nem todos os espaços acima serão usados.

Bibliografia: The Turing Omnibus, A. Dewdney, pag 181. Algorithmics, The Spirit of Computation, D Harel. .