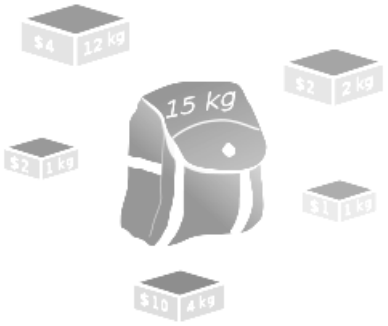
Universidade Positivo Sistemas de Informação
11/02/2019 - 12:34:01.3
Algoritmos Prof Dr P Kantek
(pkantek@up.edu.br)
Programação dinâmica: o problema da mochila 0-1 VIVO240a V: 1.04
1
- 1 _____ / _____ / _____

O problema da mochila 0-1

O problema da mochila (em inglês, Knapsack problem) é um problema de otimização combinatória. O nome é devido ao modelo de uma situação em que é necessário preencher uma mochila com objetos de diferentes pesos e valores. O objetivo é que se preencha a mochila com o maior valor possível, não ultrapassando o peso máximo.

O problema da mochila é um dos 21 problemas NP-completos de Richard Karp, exposto em 1972. A formulação do problema é extremamente simples, porém sua solução é mais complexa.



Normalmente este problema é resolvido com programação dinâmica, obtendo então a resolução exata do problema. Foi possivelmente reportado pela primeira vez na literatura por Dantzig (1957) e constitui um marco das técnicas de programação inteira, otimização combinatória e programação dinâmica.

Podemos definir o problema matematicamente da seguinte forma:

Dados vetores $x[1..n]$ e $w[1..n]$, vamos denotar por $x.w$ o produto escalar de x por w:

$$x.w = x[1]w[1] + ... + x[n]w[n]$$

Suponha dado um número positivo ou nulo W e vetores $w[1..n]$ e $v[1..n]$ com componentes positivos ou nulos. Uma mochila é qualquer vetor $x[1..n]$ tal que

$x.w \leq W$ e $0 \leq x[i] \leq 1$ para todo i

O valor de uma mochila é o número $x \times v$. Uma mochila é ótima se tem valor máximo.

Imagine que você possui um contêiner de certo tamanho e vários produtos com seus valores para serem colocados dentro dele. Porém os produtos devem ser colocados de um jeito que o contêiner tenha o maior valor.

Suponha que você tenha em sua máquina uma pasta cheia de arquivos e deseja gravá-los em CD, só que você já sabe de antemão que não vai caber tudo num CD só. Podem ser dois, três... dez CD's. Como proceder para encaixar o máximo de arquivos em cada CD desperdiçando o mínimo espaço em cada disco?

Desses exemplos podemos entender que a motivação do problema da mochila é colocar dentro de um compartimento uma quantidade de objetos com determinadas importâncias para que esse compartimento tenha a maior importância possível.

O modelo em si pode ser aplicado, além nos exemplos citados acima, em casos como: Investimento de capital, corte e empacotamento, carregamento de veículos, orçamento.

Foi também utilizado para a construção do algoritmo de Criptografia de chave pública, onde no contexto do problema da mochila a chave pública seria o peso total que o mochila pode carregar e a partir dai, por essa palavra a encriptação é gerada.

Por ser um problema combinatório é possivel resolve-lo por enumeração exaustiva, ou seja tentar todas as soluções possiveis e comparando-as para identificar a melhor solução. Porém, obter todos os subconjuntos de um conjunto dado tem complexidade de $O(2^n)$. Esta abordagem pode tornar completamente inviável a solução uma vez que, mesmo em pequenas dimensões como um problema com apenas 20 itens, haveria um numero enorme de respostas. Em um problema com mais de 80 itens, o algoritmo levaria bilhões de anos para ser concluido. Assim, o método de resolução por enumeração exaustiva é de utilidade muito reduzida, senão mesmo nula, para problemas reais.

Outras estratégias são o *backtracking* (enumeração inteligente através da construção de uma árvore), o *método guloso* (é um macete que sempre escolhe a melhor alternativa. É rápido, mas nem sempre dá certo). A estratégia a ser usada aqui é a da programação dinâmica.

É um método aprimorado de usar recursividade que ao invés de chamar uma função várias vezes ele, na primeira vez que é chamado, armazena o resultado para que cada vez que a função for chamada novamente volte o resultado e não uma requisição para ser resolvida.

Exemplo: Suponha $W = 50$ e $n = 4$. A figura abaixo dá os valores de $w[1..4]$ e $v[1..4]$. Mais abaixo, temos uma solução $x[1..4]$ do problema fracionário e uma solução $x'[1..4]$ do problema booleano. O valor da mochila fracionária é $x.v = 1040$. O valor da mochila booleana é $x'.v = 1000$.

w	40	30	20	10
v	840	600	400	100
x	1	0.333	0	0
x'	0	1	1	0

Algoritmo

O problema da mochila booleana pode ser resolvido por um algoritmo de programação dinâmica [CLR 17.2-1] que consome $O(nW)$ unidades de tempo. Mas isso só é possível se W e $w[1...n]$ são todos inteiros não-negativos.

O consumo $O(nW)$ é bem melhor que o do algoritmo recursivo, mas não é nenhuma maravilha, porque é proporcional a W.

A tabela t contém as soluções das instâncias do problema. A tabela é definida assim:

$t[i, Y]$ é o valor máximo da expressão $x[1..i].v[1..i]$ sujeita à restrição $x[1..i].w[1..i] \leq Y$.

Em português de gente, t[i][j] contém o valor da mochila composta pelos melhores i primeiros produtos, cujo peso é menor ou igual a j.

Os possíveis valores de Y são $0, 1, ..., W$ (É claro que isso só funciona porque estamos supondo W inteiro) e os possíveis valores de i são $0, 1, ..., n$. É importante que 0 seja um possível valor de Y e de i. Por exemplo, se w[1]=0 então t[1,0]=v[1].

Pode-se ver que t[0,Y]=0 para todo Y. Se $i > 0$ então $t[i, Y] = A$ se $w[i] > Y$ e $t[i, Y] = max(A, B)$ se $w[i] \leq Y$,

onde $A = t[i-1, Y]$ e $B = t[i-1, Y-w[i]]+v[i]$. Traduzindo para o português:

Se a mochila ótima para 1..i não usa i então ela é também uma mochila ótima para 1..i-1. Se a mochila ótima para 1..i usa i então ela é o resultado de juntar i com uma mochila ótima para 1..i-1.

A partir dessa recorrência é possível escrever um algoritmo de programação dinâmica para determinar t e o vetor x:

```
1: Mochila-Booleana (w, v, W)
2: n ← dimensão de w
3: para Y ← 0 até W
4:    t[0][Y] ← 0
5:    para i ← 1 até n
6:       A ← t[i-1][Y]
7:       se w[i] > Y
8:          B ← 0
9:       senão
10:         B ← t[i-1][Y-w[i]]+v[i]
11:      fim{se}
12:      t[i][Y] ← max(A,B)
13:   fim{para}
14: fim{para}
15: Y ← W
16: para i ← n decrescendo até 1
17:    se t[i][Y] = t[i-1][Y]
18:       x[i] ← 0
19:    senão
20:       x[i] ← 1
21:       Y ← Y-w[i]
22:    fim{se}
23: fim{para}
24: devolva x, t[n][W]
```

Exemplos:

```
w=1 5 1 1 6 6 8 8;v=2 21 3 13 13 11 19 4;W=14
x=0 1 0 1 0 0 1 0;Y=53

w=1 3 4 2 4 2 3 3;v=2 14 13 17 21 22 22 9;W=13
x=1 0 0 1 1 1 1 0;Y=84

w=7 4 7 8 4 3 4 4;v=3 5 5 17 7 16 23 4;W=15
x=0 0 0 1 0 1 1 0;Y=56
```

☞ Para você fazer

Considere uma instância onde w= 9 10 8 10 2 5 10 6 2 1 7 1 , v= 8 20 21 20 16 19 18 14 14 13 17 10 , e W= 19 . E responda quais itens entram na mochila (0=não; 1=sim) e qual o valor da mesma:

x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
x[8]	x[9]	x[10]	x[11]	x[12]	Y	

```
1: #include<stdio.h>
2: #include<stdlib.h>
3: main(){
4:    int w[9]={0,1,5,1,1,6,6,8,8};
5:    int v[9]={0,2,21,3,13,13,11,19,4};
6:    int t[9][101];  //colunas igual a no minimo W.
                      //Pode colocar um valor bem gra
7:    int x[9]={-1,-1,-1,-1,-1,-1,-1,-1,-1};
8:    int n=8;
9:    int W=14;
10:   int Y,i,A,B;
11:   for(Y=0;Y<=W;Y++){
12:      t[0][Y]=0;
13:      for (i=1;i<=n;i++){
14:         A = t[i-1][Y];
15:         if (w[i]>Y) {
16:            B = 0;
17:         }
18:         else {
19:            B=t[i-1][Y-w[i]]+v[i];
20:         }
21:         if (A>=B) {
22:            t[i][Y]=A;
23:         }
24:         else {
25:            t[i][Y]=B;
26:         }
27:      }
28:   }
29:   Y = W;
30:   for (i=n;i>=1;i--){
31:      if (t[i][Y]==t[i-1][Y]) {
32:         x[i]=0;
33:      }
34:      else {
35:         x[i]=1;
36:         Y=Y-w[i];
37:      }
38:   }
39:   for (i=1;i<=n;i++){
40:      printf("%i ",x[i]);
41:   }
42:   printf("\n ====>%i", t[n][W]);
43: }
```


Obs: Veja as funções geracaso e mochila no WS 240, para mostrar em sala. Referência Goodrich e Tamassia, pág. 281.