Mochila

Dados dois vetores $x[1\,..\,n]$ e $w[1\,..\,n]$, denotamos por $x \cdot w$ o produto escalar

$$w[1]x[1] + w[2]x[2] + \cdots + w[n]x[n].$$

Suponha dado um número inteiro não-negativo W e vetores positivos $w[1\,..\,n]$ e $v[1\,..\,n]$.

Uma mochila é qualquer vetor $x[1\,..\,n]$ tal que

$$x \cdot w \leq W \quad \text{e} \quad 0 \leq x[i] \leq 1 \text{ para todo } i$$

O valor de uma mochila é o número $x \cdot v$.

Uma mochila é ótima se tem valor máximo.

Problema booleano da mochila

Uma mochila $x[1\,..\,n]$ tal que $x[i] = 0$ ou $x[i] = 1$ para todo i é dita booleana.

Problema (Knapsack Problem): Dados (w, v, n, W), encontrar uma mochila boolena ótima.

Exemplo: $W = 50$, $n = 4$

	1	2	3	4	
w	40	30	20	10	
v	840	600	400	100	
x	1	0	0	0	valor $= 840$
x	1	0	0	1	valor $= 940$
x	0	1	1	0	valor $= 1000$

Algoritmo de programação dinâmica

Devolve o valor de uma mochila booleana ótima para (w, v, n, W).

```
MOCHILA-BOOLEANA (w, v, n, W)
 1  para Y ← 0 até W faça
 2        t[0, Y] ← 0
 3        para i ← 1 até n faça
 4              a ← t[i−1, Y]
 5              se w[i] > Y
 6                    então b ← 0
 7                    senão b ← t[i−1, Y −w[i]] + v[i]
 8              t[i, Y] ← max{a, b}
 9  devolva t[n, W]
```

Consumo de tempo é $\Theta(nW)$.

Conclusão

O consumo de tempo do algoritmo
MOCHILA-BOOLEANA é $\Theta(nW)$.

NOTA:
O consumo $\Theta(n2^{\lg W})$ é exponencial!

Explicação: o "tamanho" de W é $\lg W$ e não W
(tente multiplicar $w[1], \ldots, w[n]$ e W por 1000)

Se W é $\Omega(2^n)$ o consumo de tempo é $\Omega(n2^n)$,
mais lento que o algoritmo força bruta!

Obtenção da mochila

```
MOCHILA (w, n, W, t)
1    Y ← W
2    para i ← n decrescendo até 1 faça
3         se t[i, Y] = t[i−1, Y]
4              então x[i] ← 0
5              senão x[i] ← 1
6                    Y ← Y − w[i]
7    devolva x
```

Consumo de tempo é $\Theta(n)$.

Versão recursiva

MEMOIZED-MOCHILA-BOOLEANA (w, v, n, W)

1 **para** $i \leftarrow 0$ **até** n **faça**

2 **para** $Y \leftarrow 0$ **até** W **faça**

3 $t[i, Y] \leftarrow \infty$

3 **devolva** LOOKUP-MOC (w, v, n, W)

Versão recursiva

```
LOOKUP-MOC (w, v, i, Y)
1   se t[i, Y] < ∞
2       então devolva t[i, Y]
3   se n = 0 ou Y = 0 então t[i, Y] ← 0
    senão
4       se w[n] > Y
            então
5               t[i, Y] ← LOOKUP-MOC (w, v, n−1, Y)
            senão
6               a ← LOOKUP-MOC (w, v, i−1, Y)
7               b ← LOOKUP-MOC (w, v, i−1, Y−w[i]) + v[i]
8               t[i, Y] ← max {a, b}
9   devolva t[i, Y]
```

Algoritmos gulosos (*greedy*)

CLRS 16.1–16.3

Algoritmos gulosos

"A *greedy algorithm* starts with a solution to a very small subproblem and augments it successively to a solution for the big problem. The augmentation is done in a "greedy" fashion, that is, paying atention to short-term or local gain, without regard to whether it will lead to a good long-term or global solution. As in real life, greedy algorithms sometimes lead to the best solution, sometimes lead to pretty good solutions, and sometimes lead to lousy solutions. The trick is to determine when to be greedy."

"One thing you will notice about greedy algorithms is that they are usually easy to design, easy to implement, easy to analyse, and they are very fast, but they are *almost always difficult to prove correct*."

I. Parberry, *Problems on Algorithms*, Prentice Hall, 1995.

Problema fracionário da mochila

Problema: Dados (w, v, n, W), encontrar uma mochila ótima.

Exemplo: $W = 50$, $n = 4$

	1	2	3	4	
w	40	30	20	10	
v	840	600	400	100	
x	1	0	0	0	valor $= 840$
x	1	0	0	1	valor $= 940$
x	0	1	1	0	valor $= 1000$
x	1	$1/3$	0	0	valor $= 1040$

A propósito . . .

O problema fracionário da mochila é um problema de programação linear (PL): encontrar um vetor x que

$$\begin{array}{rlll} \text{maximize} & x \cdot v & & \\ \text{sob as restrições} & x \cdot w & \leq W & \\ & x[i] & \geq 0 & \text{para } i = 1, \ldots, n \\ & x[i] & \leq 1 & \text{para } i = 1, \ldots, n \end{array}$$

PL's podem ser resolvidos por

SIMPLEX: no pior caso consome tempo exponencial
na prática é muito rápido

ELIPSÓIDES: consome tempo polinomial
na prática é lento

PONTOS-INTERIORES: consome tempo polinomial
na prática é rápido

Subestrutura ótima

Suponha que $x[1\,..\,n]$ é mochila ótima para o problema (w, v, n, W).

Se $x[n] = \delta$

então $x[1\,..\,n{-}1]$ é mochila ótima para

$$(w, v, n-1, W - \delta\, w[n])$$

NOTA. Não há nada de especial acerca do índice n. Uma afirmação semelhante vale para qualquer índice i.

Escolha gulosa

Suponha $w[i] \neq 0$ para todo i.

Se $v[n]/w[n] \geq v[i]/w[i]$ para todo i

então EXISTE uma mochila ótima $x[1\,..\,n]$ tal que

$$x[n] = \min\left\{1, \frac{W}{w[n]}\right\}$$

Algoritmo guloso

Esta propriedade da escolha gulosa sugere um algoritmo que atribui os valores de $x[1\,..\,n]$ supondo que os dados estejam em ordem decrescente de "valor específico" :

$$\frac{v[1]}{w[1]} \leq \frac{v[2]}{w[2]} \leq \cdots \leq \frac{v[n]}{w[n]}$$

É nessa ordem "mágica" que está o segredo do funcionamento do algoritmo.

Algoritmo guloso

Devolve uma mochila ótima para (w, v, n, W).

MOCHILA-FRACIONÁRIA (w, v, n, W)

```
0   ordene w e v de tal forma que
                v[1]/w[1] ≤ v[2]/w[2] ≤ ··· ≤ v[n]/w[n]
1   para i ← n decrescendo até 1 faça
2       se w[i] ≤ W
3           então x[i] ← 1
4                 W ← W − w[i]
5           senão x[i] ← W/w[i]
6                 W ← 0
7   devolva x
```

Consumo de tempo da linha 0 é $\Theta(n \lg n)$.
Consumo de tempo das linhas 1–7 é $\Theta(n)$.

Invariante

Seja W_0 o valor original de W.
No início de cada execução da linha 1 vale que

(i0) $x' = x[i{+}1\,..\,n]$ é mochila ótima para

$$(w', v', n', W_0)$$

onde

$$w' = w[i{+}1\,..\,n]$$
$$v' = v[i{+}1\,..\,n]$$
$$n' = n - i$$

Na última iteração $i = 0$ e portanto $x[1\,..\,n]$ é mochila ótima para (w, v, n, W_0).

Conclusão

O consumo de tempo do algoritmo MOCHILA-FRACIONÁRIA é $\Theta(n \lg n)$.

Escolha gulosa

Precisamos mostrar que se $x[1\,..\,n]$ é uma mochila ótima, então podemos supor que

$$x[n] = \alpha := \min\left\{1, \frac{W}{w[n]}\right\}$$

Depois de mostrar isto, indução faz o resto do serviço.

Técnica: transformar uma solução ótima em uma solução ótima 'gulosa'.

Esta transformação é semelhante ao processo de pivotação feito pelo algoritmo SIMPLEX para programação linear.

Algoritmos gulosos

Algoritmo guloso

- procura ótimo local e acaba obtendo ótimo global

costuma ser

- muito simples e intuitivo
- muito eficiente
- difícil provar que está correto

Problema precisa ter

- subestrutura ótima (como na programação dinâmica)
- propriedade da escolha gulosa (*greedy-choice property*)

Exercício: O problema da mochila booleana pode ser resolvido por um algoritmo guloso?