

# 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

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

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 booleana 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
$x$	1	0	0	1
$x$	0	1	1	0

valor = 840

valor = 940

valor = 1000

# Subestrutura ótima

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

Se  $x[n] = 1$

então  $x[1..n-1]$  é **mochila booleana ótima** para  $(w, v, n-1, W - w[n])$

senão  $x[1..n]$  é **mochila booleana ótima** para  $(w, v, n-1, W)$

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

# Simplificação

**Problema:** encontrar o **valor** de uma mochila booleana ótima.

$t[i, Y]$  = valor de uma mochila booleana ótima  
para  $(w, v, i, Y)$

= valor da expressão  $x \cdot v$  sujeito às restrições

$$x \cdot w \leq Y,$$

onde  $x$  é uma mochila booleana ótima

Possíveis valores de  $Y$ :  $0, 1, 2, \dots, W$

# Recorrência

$t[i, Y]$  = valor da expressão  $x \cdot v$  sujeito à restrição

$$x \cdot w \leq Y$$

$t[0, Y] = 0$  para todo  $Y$

$t[i, 0] = 0$  para todo  $i$

$t[i, Y] = t[i-1, Y]$  se  $w[i] > Y$

$t[i, Y] = \max \{t[i-1, Y], t[i-1, Y-w[i]] + v[i]\}$  se  $w[i] \leq Y$

# Solução recursiva

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

**REC-MOCHILA**  $(w, v, n, W)$

1 **se**  $n = 0$  **ou**  $W = 0$

2 **então devolva** 0

3 **se**  $w[n] > W$

4 **então devolva** **REC-MOCHILA**  $(w, v, n-1, W)$

5  $a \leftarrow$  **REC-MOCHILA**  $(w, v, n-1, W)$

6  $b \leftarrow$  **REC-MOCHILA**  $(w, v, n-1, W - w[n]) + v[n]$

7 **devolva**  $\max \{a, b\}$

Consumo de tempo no **pior caso** é  $\Omega(2^n)$

Por que demora tanto?

O mesmo subproblema é resolvido muitas vezes.

# Programação dinâmica

Cada subproblema, valor de uma mochila ótima para

$$(w, v, i, Y),$$

é resolvido **uma só** vez.

Em que ordem calcular os componentes da tabela  $t$ ?

# Programação dinâmica

Cada subproblema, valor de uma mochila ótima para

$$(w, v, i, Y),$$

é resolvido **uma só** vez.

Em que ordem calcular os componentes da tabela  $t$ ?

Olhe a recorrência e pense...

$$t[i, Y] = t[i-1, Y] \text{ se } w[i] > Y$$

$$t[i, Y] = \max \{t[i-1, Y], t[i-1, Y-w[i]] + v[i]\} \text{ se } w[i] \leq Y$$

# Programação dinâmica

	0	1	2	3	4	5	6	7	<i>Y</i>
0	0	0	0	0	0	0	0	0	
1	0								
2	0	*	*	*	*	*			
3	0					??			
4	0								
5	0								
6	0								
7	0								

*i*

# Exemplo

$$W = 5 \text{ e } n = 4$$

	1	2	3	4
$w$	4	2	1	3
$v$	500	400	300	450

	0	1	2	3	4	5	$Y$
0	0	0	0	0	0	0	
1	0	0	0	0	500	500	
2	0	0	400	400	500	500	
3	0	300	400	400	700	800	
4	0	300	400	450	750	850	
$i$							

# 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  $Y \leftarrow 0$  até  $W$  faça
2     $t[0, Y] \leftarrow 0$ 
3    para  $i \leftarrow 1$  até  $n$  faça
4       $a \leftarrow t[i-1, Y]$ 
5      se  $w[i] > Y$ 
6        então  $b \leftarrow 0$ 
7        senão  $b \leftarrow t[i-1, Y - w[i]] + v[i]$ 
8       $t[i, Y] \leftarrow \max\{a, b\}$ 
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  $w[1], \dots, w[n]$  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  $Y \leftarrow W$

2 **para**  $i \leftarrow n$  **decrecendo até 1 faça**

3     **se**  $t[i, Y] = t[i-1, Y]$

4         **então**  $x[i] \leftarrow 0$

5         **senão**  $x[i] \leftarrow 1$

6              $Y \leftarrow Y - w[i]$

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**  $t[i, Y] < \infty$

2     **então devolva**  $t[i, Y]$

3 **se**  $n = 0$  **ou**  $Y = 0$  **então**  $t[i, Y] \leftarrow 0$

**senão**

4     **se**  $w[n] > Y$

**então**

5              $t[i, Y] \leftarrow$  **LOOKUP-MOC** ( $w, v, n-1, Y$ )

**senão**

6              $a \leftarrow$  **LOOKUP-MOC** ( $w, v, i-1, Y$ )

7              $b \leftarrow$  **LOOKUP-MOC** ( $w, v, i-1, Y - w[i]$ )  $+ v[i]$

8              $t[i, Y] \leftarrow \max \{a, b\}$

9 **devolva**  $t[i, Y]$

# Algoritmos gulosos (*greedy*)

CLRS 16.2

# 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 attention 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
$x$	1	0	0	1
$x$	0	1	1	0
$x$	1	1/3	0	0

valor = 840

valor = 940

valor = 1000

valor = 1040

# A propósito ...

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

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

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” :

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

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  
    $v[1]/w[1] \leq v[2]/w[2] \leq \dots \leq v[n]/w[n]$   
1  para  $i \leftarrow n$  decrecendo até 1 faça  
2    se  $w[i] \leq W$   
3      então  $x[i] \leftarrow 1$   
4           $W \leftarrow W - w[i]$   
5    senão  $x[i] \leftarrow W/w[i]$   
6           $W \leftarrow 0$   
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.

# Escolha gulosa

Seja  $x[1..n]$  uma mochila ótima para  $(w, v, n, W)$  tal que  $x[n]$  é máximo. Se  $x[n] = \alpha$ , não há o que mostrar.

Suponha  $x[n] < \alpha$ .

Podemos supor que  $x \cdot w = W$ . (Podemos mesmo?)

Seja  $i$  em  $[1..n-1]$  tal que  $x[i] > 0$ .

(Quem garante que existe um tal  $i$ ?).

Seja

$$\delta := \min \left\{ x[i], (\alpha - x[n]) \frac{w[n]}{w[i]} \right\}$$

e

$$\beta := \delta \frac{w[i]}{w[n]}.$$

Note que  $\delta > 0$  e  $\beta > 0$ .

# Mais escolha gulosa

Seja  $x'[1..n]$  tal que

$$x'[j] := \begin{cases} x[j] & \text{se } j \neq i \text{ e } j \neq n, \\ x[i] - \delta & \text{se } j = i, \\ x[n] + \beta & \text{se } j = n. \end{cases}$$

Verifique que  $0 \leq x[j] \leq 1$  para todo  $j$ .

Além disso, temos que

$$\begin{aligned} x' \cdot w &= x'[1]w[1] + \dots + x'[i]w[i] + \dots + x'[n]w[n] \\ &= x[1]w[1] + \dots + (x[i] - \delta)w[i] + \dots + (x[n] + \beta)w[n] \\ &= x[1]w[1] + \dots + (x[i] - \delta)w[i] + \dots + \left( x[n] + \delta \frac{w[i]}{w[n]} \right) w[n] \\ &= x[1]w[1] + \dots + x[i]w[i] - \delta w[i] + \dots + x[n]w[n] + \delta w[i] \\ &= W. \end{aligned}$$

# Mais escolha gulosa ainda

Temos ainda que

$$\begin{aligned}x' \cdot v &= x'[1]v[1] + \cdots + x'[i]v[i] + \cdots + x'[n]v[n] \\&= x[1]v[1] + \cdots + (x[i] - \delta)v[i] + \cdots + (x[n] + \beta)v[n] \\&= x[1]v[1] + \cdots + (x[i] - \delta)v[i] + \cdots + \left(x[n] + \delta \frac{w[i]}{w[n]}\right)v[n] \\&= x[1]v[1] + \cdots + x[i]v[i] - \delta v[i] + \cdots + x[n]v[n] + \delta w[i] \frac{v[n]}{w[n]} \\&= x \cdot v + \delta \left( w[i] \frac{v[n]}{w[n]} - v[i] \right) \\&\geq x \cdot v + \delta \left( w[i] \frac{v[i]}{w[i]} - v[i] \right) \quad (\text{devido a escolha gulosa!}) \\&= x \cdot v.\end{aligned}$$

# Escolha gulosa: epílogo

Assim,  $x'$  é uma mochila tal que  $x' \cdot v \geq x \cdot v$ .

Como  $x$  é **mochila ótima**, concluímos que  $x' \cdot v = x \cdot v$  e que  $x'$  é uma mochila ótima que contradiz a nossa escolha de  $x$ , já que

$$x'[n] = x[n] + \beta > x[n].$$

## Conclusão

Se  $v[n]/w[n] \geq v[i]/w[i]$  para todo  $i$   
e  $x$  é uma **mochila ótima** para  $(w, v, n, W)$  com  $x[n]$   
**máximo**, então

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

# 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?