

AULA 15

Mochila

Dados dois vetores $w[1..n]$ e $x[1..n]$ e , denotamos por $w \cdot x$ 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 de inteiros não-negativos $w[1..n]$ e $v[1..n]$.

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

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

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

Uma mochila é **ótima** se tem valor máximo.

Problema booleano da mochila

Um 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 .

Solução recursiva

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

```
REC-MOCHILA ( $w, v, i, Y$ )
1  se  $i = 0$  ou  $Y = 0$ 
2      então devolva 0
3  se  $w[i] > Y$ 
4      então devolva REC-MOCHILA ( $w, v, i-1, Y$ )
5   $a \leftarrow$  REC-MOCHILA ( $w, v, i-1, Y$ )
6   $b \leftarrow$  REC-MOCHILA ( $w, v, i-1, Y - w[n]$ ) +  $v[n]$ 
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.

Exemplo

Suponha $w[i] = 1$ para todo i e $W \geq n$.

$T(i, Y)$ = número de execuções da linha 7 na chamada para (w, v, i, Y)

$T(0, Y) = 0$ para todo Y

$T(i, 0) = 0$ para todo i

$T(i, Y) = T(i-1, Y) + T(i-1, Y-1) + 1$ se $i > 0$ e $Y > 0$

Verifique que $T(n, W) = 2^n - 1$ para $W \geq n$.

$$T(i, Y)$$

	0	1	2	3	4	5	6	7	Y
0	0	0	0	0	0	0	0	0	
1	0	1	1	1	1	1	1	1	
2	0	2	3	3	3	3	3	3	
3	0	3	6	7	7	7	7	7	
4	0	4	10	14	15	15	15	15	
5	0	5	15	25	30	31	31	31	
6	0	6	21	40	46	62	63	63	
7	0	7	28	62	87	109	126	127	

i

Programação dinâmica

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 $v \cdot x$ sujeito à restrição

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

onde x é uma **mochila booleana ótima**

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

Recorrência

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

$$w \cdot x \leq Y$$

onde x é um vetor **booleano**

$t[0, Y] = 0$ 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** $w[i] \leq Y$

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 ?

Para calcular $t[4, 6]$ preciso de ...

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 ?

Para calcular $t[4, 6]$ preciso de ...

$$t[4-1, 6], t[4-1, 5], t[4-1, 4] \\ t[4-1, 3], t[4-1, 2], t[4-1, 1] \text{ e de } t[4-1, 0] .$$

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 ?

Para calcular $t[4, 6]$ preciso de ...

$t[4-1, 6], t[4-1, 5], t[4-1, 4]$
 $t[4-1, 3], t[4-1, 2], t[4-1, 1]$ e de $t[4-1, 0]$.

Calcule todos os $t[i, Y]$ com $Y = 1, i = 0, 1, \dots, n$,
depois todos com $Y = 2, i = 0, 1, \dots, n$,
depois todos com $Y = 3, i = 0, 1, \dots, n$,
etc.

Programação dinâmica

	1	2	3	4	5	6	7	8	<i>Y</i>
1	0	0	0	0	0	0	0	0	
2	0								
3	0	★	★	★	★	★			
4	0					??			
5	0								
6	0								
7	0								
8	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	710	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 $\Omega(n2^n)$,
tão lento quanto o algoritmo **força bruta**!

Obtenção da mochila

MOCHILA (w, n, W, t)

```
1   $Y \leftarrow W$ 
2  para  $i \leftarrow n$  decrescendo 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 $t[i, Y] \leftarrow \infty$
- 4 **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  $i = 0$  ou  $Y = 0$  então  $t[i, Y] \leftarrow 0$ 
   senão
4      se  $w[i] > Y$ 
       então
5           $t[i, Y] \leftarrow \text{LOOKUP-MOC}(w, v, i-1, Y)$ 
       senão
6           $a \leftarrow \text{LOOKUP-MOC}(w, v, i-1, Y)$ 
7           $b \leftarrow \text{LOOKUP-MOC}(w, v, i-1, Y - w[i]) + v[i]$ 
8           $t[i, Y] \leftarrow \max\{a, b\}$ 
9  devolva  $t[i, Y]$ 
```

Algoritmos gulosos (*greedy*)

CLRS 16.1–16.3

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 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

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

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

Se $x[n] = \delta$

então $x[1 \dots n-1]$ 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 $\Theta(n)$.

Correção

No início de cada execução da linha 1 vale que

(i0) $x' = x[i+1 .. n]$ é **mochila ótima** para

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

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) .

Conclusão

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

Escolha gulosa

Precisamos mostrar que se $x[1 \dots n]$ 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 um **solução ótima** em uma **solução ótima 'gulosa'**.

Esta transformação é semelhante ao processo de pivotação feita 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 $x[n] < \alpha$.

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

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

Mais escolha gulosa ainda

Temos ainda que

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

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

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

Máximo e maximal

\mathcal{S} = coleção de subconjuntos de $\{1, \dots, n\}$

Elemento X de \mathcal{S} é **máximo** se
não existe Y em \mathcal{S} tal que $|Y| > |X|$.

Elemento X de \mathcal{S} é **maximal** se
não existe Y em \mathcal{S} tal que $Y \supset X$.

Exemplo:

$\{ \{1, 2\}, \{2, 3\}, \{4, 5\}, \{1, 2, 3\}, \{2, 3, 4, 5\}, \{1, 3, 4, 5\} \}$

$\{1, 2\}$ **não é maximal**

$\{1, 2, 3\}$ **é maximal**

$\{2, 3, 4, 5\}$ **é maximal e máximo**

Problemas

Problema 1: Encontrar elemento **máximo** de \mathcal{S} .
Usualmente difícil.

Problema 2: Encontrar elemento **maximal** de \mathcal{S} .
Muito fácil: aplique algoritmo “guloso”.

\mathcal{S} “tem estrutura gulosa” se **todo** maximal é máximo.
Se \mathcal{S} tem estrutura gulosa, Problema 1 é fácil.

Algoritmos gulosos

Algoritmo guloso

- procura maximal e acaba obtendo máximo
- 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*)

Exercícios

Exercício 21.A

O problema da soma de subconjunto do exercício 19.F pode ser resolvido por um algoritmo guloso? O problema tem a propriedade da escolha gulosa?

Exercício 21.B

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

Exercício 21.C [Mochila fracionária. CLRS 16.2-1]

O problema da mochila fracionária consiste no seguinte: dados números inteiros não-negativos v_1, \dots, v_n e w_1, \dots, w_n , W , encontrar racionais x_1, \dots, x_n no intervalo $[0, 1]$ tais que

$$x_1 w_1 + \dots + x_n w_n \leq W \text{ e } x_1 v_1 + \dots + x_n v_n \text{ é máxima.}$$

(Imagine que w_i é o *peso* e v_i é o *valor* do objeto i .) Escreva um algoritmo guloso para resolver o problema.

Para provar que o seu algoritmo está correto, verifique as seguintes propriedades.

Propriedades da subestrutura ótima: se x_1, \dots, x_n é solução ótima do problema

(w, v, n, W) então x_1, \dots, x_{n-1} é solução ótima do problema $(w, v, n-1, W - x_n w_n)$.

Propriedade da escolha gulosa: se $v_n/w_n \geq v_i/w_i$ para todo i então existe uma solução ótima x tal que $x_n = \min(1, W/w_n)$.