

Mais programação dinâmica

CLRS 15.5

- = “recursão-com-tabela”
- = transformação inteligente de recursão em iteração

Buscas em conjunto conhecido

Dadas **estimativas** do número de acessos a cada elemento de $v[1..n]$, qual é a melhor estrutura de dados para v ?

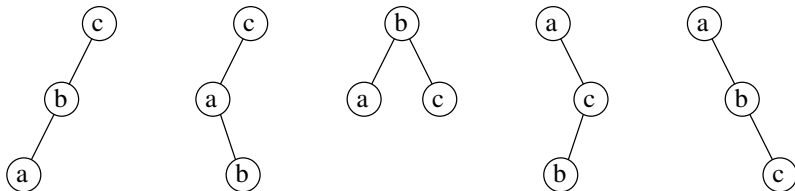
Buscas em conjunto conhecido

Dadas **estimativas** do número de acessos a cada elemento de $v[1..n]$, qual é a melhor estrutura de dados para v ?

Árvore de busca binária (ABB)?

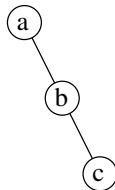
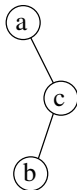
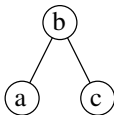
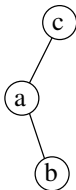
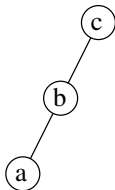
Exemplo: $n = 3$ e $e_1 = 10$, $e_2 = 20$, $e_3 = 40$.

Qual a melhor das ABBs?



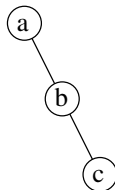
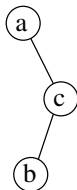
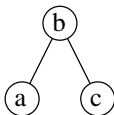
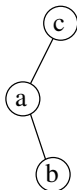
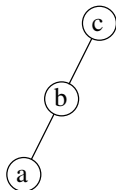
Exemplo

Exemplo: $n = 3$ e $e_1 = 10$, $e_2 = 20$, $e_3 = 40$.



Exemplo

Exemplo: $n = 3$ e $e_1 = 10$, $e_2 = 20$, $e_3 = 40$.

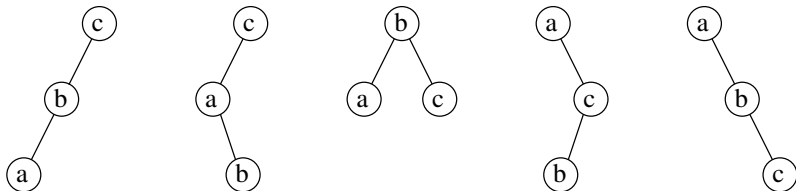


Número esperado de comparações:

- ▶ $10 \cdot 3 + 20 \cdot 2 + 40 \cdot 1 = 110$
- ▶ $10 \cdot 2 + 20 \cdot 3 + 40 \cdot 1 = 120$
- ▶ $10 \cdot 2 + 20 \cdot 1 + 40 \cdot 2 = 120$
- ▶ $10 \cdot 1 + 20 \cdot 3 + 40 \cdot 2 = 150$
- ▶ $10 \cdot 1 + 20 \cdot 2 + 40 \cdot 3 = 170$

Exemplo

Exemplo: $n = 3$ e $e_1 = 10$, $e_2 = 20$, $e_3 = 40$.



Número esperado de comparações:

- ▶ $10 \cdot 3 + 20 \cdot 2 + 40 \cdot 1 = 110$ ← ABB ótima
- ▶ $10 \cdot 2 + 20 \cdot 3 + 40 \cdot 1 = 120$
- ▶ $10 \cdot 2 + 20 \cdot 1 + 40 \cdot 2 = 120$
- ▶ $10 \cdot 1 + 20 \cdot 3 + 40 \cdot 2 = 150$
- ▶ $10 \cdot 1 + 20 \cdot 2 + 40 \cdot 3 = 170$

Árvore de busca ótima

Considere um vetor $e[1..n]$ de tipo `Ord` com uma estimativa do número de acessos a cada elemento de $\{1, \dots, n\}$.

Árvore de busca ótima

Considere um vetor $e[1..n]$ de tipo `Ord` com uma estimativa do número de acessos a cada elemento de $\{1, \dots, n\}$.

Uma ABB ótima com respeito ao vetor e é uma ABB para o conjunto $\{1, \dots, n\}$ que minimiza o número

$$\sum_{i=1}^n h_i e_i,$$

onde h_i é o número de nós no caminho de i até a raiz da árvore.

Árvore de busca ótima

Considere um vetor $e[1..n]$ de tipo `Ord` com uma estimativa do número de acessos a cada elemento de $\{1, \dots, n\}$.

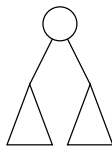
Uma ABB ótima com respeito ao vetor e é uma ABB para o conjunto $\{1, \dots, n\}$ que minimiza o número

$$\sum_{i=1}^n h_i e_i,$$

onde h_i é o número de nós no caminho de i até a raiz da árvore.

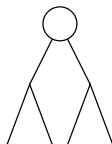
Problema (ABB Ótima): Dado $e[1..n]$, encontrar uma árvore de busca binária ótima com respeito a e .

Subestrutura ótima



Subárvores esquerda e direita de uma ABB ótima
são ABBs ótimas.

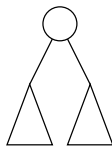
Subestrutura ótima



Subárvores esquerda e direita de uma ABB ótima são ABBs ótimas.

Resta determinar a raiz da ABB ótima.

Subestrutura ótima



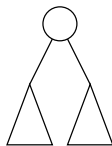
Subárvores esquerda e direita de uma ABB ótima são ABBs ótimas.

Resta determinar a **raiz** da ABB ótima.

$c[i, j]$: custo min de uma ABB para $e[i..j]$

$s[i, j]$: soma dos acessos em $e[i..j]$

Subestrutura ótima



Subárvores esquerda e direita de uma ABB ótima são ABBs ótimas.

Resta determinar a **raiz** da ABB ótima.

$c[i, j]$: custo min de uma ABB para $e[i..j]$

$s[i, j]$: soma dos acessos em $e[i..j]$

$$c[i, j] = \begin{cases} 0 & \text{se } i > j \\ \min_{i \leq k \leq j} \{c[i, k-1] + c[k+1, j] + s[i, j]\} & \text{se } i \leq j \end{cases}$$

Custo de uma ABB ótima

$c[i, j]$: custo min de uma ABB para $e[i..j]$

$s[j]$: soma dos acessos em $e[1..j]$

$s[j] - s[i-1]$: soma dos acessos em $e[i..j]$

$$c[i, j] = \begin{cases} 0 & \text{se } i > j \\ \min_{i \leq k \leq j} \{c[i, k-1] + c[k+1, j]\} + s[j] - s[i-1] & \text{se } i \leq j \end{cases}$$

Custo de uma ABB ótima

$c[i, j]$: custo min de uma ABB para $e[i..j]$

$s[j]$: soma dos acessos em $e[1..j]$

$s[j] - s[i-1]$: soma dos acessos em $e[i..j]$

$$c[i, j] = \begin{cases} 0 & \text{se } i > j \\ \min_{i \leq k \leq j} \{c[i, k-1] + c[k+1, j]\} + s[j] - s[i-1] & \text{se } i \leq j \end{cases}$$

Para calcular s :

- 1 $s[0] = 0$
- 2 **para** $i \leftarrow 1$ **até** n **faça**
- 3 $s[i] \leftarrow s[i-1] + e[i]$

Custo de uma ABB ótima

$c[i, j]$: custo min de uma ABB para $e[i..j]$

$s[j]$: soma dos acessos em $e[1..j]$

$s[j] - s[i-1]$: soma dos acessos em $e[i..j]$

$$c[i, j] = \begin{cases} 0 & \text{se } i > j \\ \min_{i \leq k \leq j} \{c[i, k-1] + c[k+1, j]\} + s[j] - s[i-1] & \text{se } i \leq j \end{cases}$$

Como preencher a matriz c ?

Em que ordem?

Custo de uma ABB ótima

$c[i, j]$: custo min de uma ABB para $e[i..j]$

$s[j]$: soma dos acessos em $e[1..j]$

$s[j] - s[i-1]$: soma dos acessos em $e[i..j]$

$$c[i, j] = \begin{cases} 0 & \text{se } i > j \\ \min_{i \leq k \leq j} \{c[i, k-1] + c[k+1, j]\} + s[j] - s[i-1] & \text{se } i \leq j \end{cases}$$

Como preencher a matriz c ?

Em que ordem?

Como no problema da parentização! Pelas diagonais!

Programação dinâmica

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

Simulação

$e[1]=10$ $e[2]=20$ $e[3]=30$ $e[4]=15$ $e[5]=30$

Simulação

$e[1]=10$ $e[2]=20$ $e[3]=30$ $e[4]=15$ $e[5]=30$

	0	1	2	3	4	5	<i>j</i>
1	0	??					
2		0					
3			0				
4				0			
5					0		
6						0	
<i>i</i>							

Simulação

$e[1]=10$ $e[2]=20$ $e[3]=30$ $e[4]=15$ $e[5]=30$

	0	1	2	3	4	5	<i>j</i>
1	0	10					
2		0					
3			0				
4				0			
5					0		
6						0	
<i>i</i>							

$$c[1, 1-1] + e[1] + c[1+1, 1] = 0 + 10 + 0 = 10$$

Simulação

$e[1]=10$ $e[2]=20$ $e[3]=30$ $e[4]=15$ $e[5]=30$

	0	1	2	3	4	5	<i>j</i>
1	0	10					
2		0	??				
3			0				
4				0			
5					0		
6						0	
<i>i</i>							

Simulação

$e[1]=10$ $e[2]=20$ $e[3]=30$ $e[4]=15$ $e[5]=30$

	0	1	2	3	4	5	<i>j</i>
1	0	10					
2		0	20				
3			0				
4				0			
5					0		
6						0	
<i>i</i>							

$$c[2, 2-1] + e[2] + c[2+1, 2] = 0 + 20 + 0 = 20$$

Simulação

$e[1]=10$ $e[2]=20$ $e[3]=30$ $e[4]=15$ $e[5]=30$

	0	1	2	3	4	5	<i>j</i>
1	0	10					
2		0	20				
3			0	??			
4				0			
5					0		
6						0	
<i>i</i>							

Simulação

$e[1]=10$ $e[2]=20$ $e[3]=30$ $e[4]=15$ $e[5]=30$

	0	1	2	3	4	5	<i>j</i>
1	0	10					
2		0	20				
3			0	30			
4				0			
5					0		
6						0	
<i>i</i>							

$$c[3, 3-1] + e[3] + c[3+1, 3] = 0 + 30 + 0 = 30$$

Simulação

$e[1]=10$ $e[2]=20$ $e[3]=30$ $e[4]=15$ $e[5]=30$

	0	1	2	3	4	5	<i>j</i>
1	0	10					
2		0	20				
3			0	30			
4				0	??		
5					0		
6						0	
<i>i</i>							

Simulação

$e[1]=10$ $e[2]=20$ $e[3]=30$ $e[4]=15$ $e[5]=30$

	0	1	2	3	4	5	<i>j</i>
1	0	10					
2		0	20				
3			0	30			
4				0	15		
5					0		
6						0	
<i>i</i>							

$$c[4, 4+1] + e[4] + c[4+1, 4] = 0 + 15 + 0 = 15$$

Simulação

$e[1]=10$ $e[2]=20$ $e[3]=30$ $e[4]=15$ $e[5]=30$

	0	1	2	3	4	5	<i>j</i>
1	0	10					
2		0	20				
3			0	30			
4				0	15		
5					0	??	
6						0	
<i>i</i>							

Simulação

$e[1]=10$ $e[2]=20$ $e[3]=30$ $e[4]=15$ $e[5]=30$

	0	1	2	3	4	5	<i>j</i>
1	0	10					
2		0	20				
3			0	30			
4				0	15		
5					0	30	
6						0	
<i>i</i>							

$$c[5, 5+1] + e[5] + c[5+1, 5] = 0 + 30 + 0 = 30$$

Simulação

$e[1]=10$ $e[2]=20$ $e[3]=30$ $e[4]=15$ $e[5]=30$

	0	1	2	3	4	5	<i>j</i>
1	0	10	??				
2		0	20				
3			0	30			
4				0	15		
5					0	30	
6						0	
<i>i</i>							

Simulação

$e[1]=10$ $e[2]=20$ $e[3]=30$ $e[4]=15$ $e[5]=30$

	0	1	2	3	4	5	<i>j</i>
1	0	10	50				
2		0	20				
3			0	30			
4				0	15		
5					0	30	
6						0	
<i>i</i>							

$$c[1, 1-1] + (e[1] + e[2]) + c[1+1, 2] = 0 + 30 + 20 = 50$$

Simulação

$e[1]=10$ $e[2]=20$ $e[3]=30$ $e[4]=15$ $e[5]=30$

	0	1	2	3	4	5	<i>j</i>
1	0	10	40				
2		0	20				
3			0	30			
4				0	15		
5					0	30	
6						0	
<i>i</i>							

$$c[1,2-1] + (e[1] + e[2]) + c[2+1,2] = 10 + 30 + 0 = 40$$

Simulação

$e[1]=10$ $e[2]=20$ $e[3]=30$ $e[4]=15$ $e[5]=30$

	0	1	2	3	4	5	<i>j</i>
1	0	10	40				
2		0	20	??			
3			0	30			
4				0	15		
5					0	30	
6						0	
<i>i</i>							

Simulação

$e[1]=10$ $e[2]=20$ $e[3]=30$ $e[4]=15$ $e[5]=30$

	0	1	2	3	4	5	<i>j</i>
1	0	10	40				
2		0	20	80			
3			0	30			
4				0	15		
5					0	30	
6						0	
<i>i</i>							

$$c[2, 2-1] + (e[2] + e[3]) + c[2+1, 3] = 0 + 50 + 30 = 80$$

Simulação

$e[1]=10$ $e[2]=20$ $e[3]=30$ $e[4]=15$ $e[5]=30$

	0	1	2	3	4	5	<i>j</i>
1	0	10	40				
2		0	20	70			
3			0	30			
4				0	15		
5					0	30	
6						0	
<i>i</i>							

$$c[2, 3-1] + (e[2] + e[3]) + c[3+1, 3] = 20 + 50 + 0 = 70$$

Simulação

$e[1]=10$ $e[2]=20$ $e[3]=30$ $e[4]=15$ $e[5]=30$

	0	1	2	3	4	5	<i>j</i>
1	0	10	40				
2		0	20	70			
3			0	30	??		
4				0	15		
5					0	30	
6						0	
<i>i</i>							

Simulação

$e[1]=10$ $e[2]=20$ $e[3]=30$ $e[4]=15$ $e[5]=30$

	0	1	2	3	4	5	<i>j</i>
1	0	10	40				
2		0	20	70			
3			0	30	60		
4				0	15		
5					0	30	
6						0	
<i>i</i>							

$$c[3, 3-1] + (e[3] + e[4]) + c[3+1, 4] = 0 + 45 + 15 = 60$$

Simulação

$e[1]=10$ $e[2]=20$ $e[3]=30$ $e[4]=15$ $e[5]=30$

	0	1	2	3	4	5	<i>j</i>
1	0	10	40				
2		0	20	70			
3			0	30	60		
4				0	15		
5					0	30	
6						0	
<i>i</i>							

$$c[3, 4-1] + (e[3] + e[4]) + c[4+1, 4] = 30 + 45 + 0 = 75$$

Simulação

$e[1]=10$ $e[2]=20$ $e[3]=30$ $e[4]=15$ $e[5]=30$

	0	1	2	3	4	5	<i>j</i>
1	0	10	40				
2		0	20	70			
3			0	30	60		
4				0	15	??	
5					0	30	
6						0	
<i>i</i>							

Simulação

$e[1]=10$ $e[2]=20$ $e[3]=30$ $e[4]=15$ $e[5]=30$

	0	1	2	3	4	5	<i>j</i>
1	0	10	40				
2		0	20	70			
3			0	30	60		
4				0	15	75	
5					0	30	
6						0	
<i>i</i>							

$$c[4, 4-1] + (e[4] + e[5]) + c[4+1, 5] = 0 + 45 + 30 = 75$$

Simulação

$e[1]=10$ $e[2]=20$ $e[3]=30$ $e[4]=15$ $e[5]=30$

	0	1	2	3	4	5	<i>j</i>
1	0	10	40				
2		0	20	70			
3			0	30	60		
4				0	15	60	
5					0	30	
6						0	
<i>i</i>							

$$c[4,5-1] + (e[4] + e[5]) + c[5+1,5] = 15 + 45 + 0 = 60$$

Simulação

$e[1]=10$ $e[2]=20$ $e[3]=30$ $e[4]=15$ $e[5]=30$

	0	1	2	3	4	5	<i>j</i>
1	0	10	40	??			
2		0	20	70			
3			0	30	60		
4				0	15	60	
5					0	30	
6						0	
<i>i</i>							

Simulação

$e[1]=10$ $e[2]=20$ $e[3]=30$ $e[4]=15$ $e[5]=30$

	0	1	2	3	4	5	<i>j</i>
1	0	10	40	130			
2		0	20	70			
3			0	30	60		
4				0	15	60	
5					0	30	
6						0	
<i>i</i>							

$$c[1, 1-1] + (e[1] + e[2] + e[3]) + c[1+1, 3] = 0 + 60 + 70 = 130$$

Simulação

$e[1]=10$ $e[2]=20$ $e[3]=30$ $e[4]=15$ $e[5]=30$

	0	1	2	3	4	5	<i>j</i>
1	0	10	40	100			
2		0	20	70			
3			0	30	60		
4				0	15	60	
5					0	30	
6						0	
<i>i</i>							

$$c[1, 2-1] + (e[1]) + e[2] + e[3] + c[2+1, 3] = 10 + 60 + 30 = 100$$

Simulação

$e[1]=10$ $e[2]=20$ $e[3]=30$ $e[4]=15$ $e[5]=30$

	0	1	2	3	4	5	<i>j</i>
1	0	10	40	100			
2		0	20	70			
3			0	30	60		
4				0	15	60	
5					0	30	
6						0	
<i>i</i>							

$$c[1, 3-1] + (e[1] + e[2] + e[3]) + c[3+1, 3] = 40 + 60 + 0 = 100$$

Simulação

$e[1]=10$ $e[2]=20$ $e[3]=30$ $e[4]=15$ $e[5]=30$

	0	1	2	3	4	5	<i>j</i>
1	0	10	40	100			
2		0	20	70	??		
3			0	30	60		
4				0	15	60	
5					0	30	
6						0	
<i>i</i>							

Simulação

$e[1]=10$ $e[2]=20$ $e[3]=30$ $e[4]=15$ $e[5]=30$

	0	1	2	3	4	5	<i>j</i>
1	0	10	40	100			
2		0	20	70	125		
3			0	30	60		
4				0	15	60	
5					0	30	
6						0	
<i>i</i>							

$$c[2, 2-1] + (e[2] + e[3] + e[4]) + c[2+1, 4] = 0 + 65 + 60 = 125$$

Simulação

$e[1]=10$ $e[2]=20$ $e[3]=30$ $e[4]=15$ $e[5]=30$

	0	1	2	3	4	5	<i>j</i>
1	0	10	40	100			
2		0	20	70	100		
3			0	30	60		
4				0	15	60	
5					0	30	
6						0	
<i>i</i>							

$$c[2, 3-1] + (e[2] + e[3] + e[4]) + c[3+1, 4] = 20 + 65 + 15 = 100$$

Navigation icons: back, forward, search, etc.

Simulação

$e[1]=10$ $e[2]=20$ $e[3]=30$ $e[4]=15$ $e[5]=30$

	0	1	2	3	4	5	<i>j</i>
1	0	10	40	100			
2		0	20	70	100		
3			0	30	60		
4				0	15	60	
5					0	30	
6						0	
<i>i</i>							

$$c[2, 4-1] + (e[2] + e[3] + e[4]) + c[4+1, 4] = 70 + 65 + 0 = 135$$

Simulação

$e[1]=10$ $e[2]=20$ $e[3]=30$ $e[4]=15$ $e[5]=30$

	0	1	2	3	4	5	<i>j</i>
1	0	10	40	100			
2		0	20	70	100		
3			0	30	60	??	
4				0	15	60	
5					0	30	
6						0	
<i>i</i>							

Simulação

$e[1]=10$ $e[2]=20$ $e[3]=30$ $e[4]=15$ $e[5]=30$

	0	1	2	3	4	5	<i>j</i>
1	0	10	40	100			
2		0	20	70	100		
3			0	30	60	135	
4				0	15	60	
5					0	30	
6						0	
<i>i</i>							

$$c[3, 3-1] + (e[3] + e[4] + e[5]) + c[3+1, 5] = 0 + 75 + 60 = 135$$

Simulação

$e[1]=10$ $e[2]=20$ $e[3]=30$ $e[4]=15$ $e[5]=30$

	0	1	2	3	4	5	<i>j</i>
1	0	10	40	100			
2		0	20	70	100		
3			0	30	60	135	
4				0	15	60	
5					0	30	
6						0	
<i>i</i>							

$$c[3, 4-1] + (e[3] + e[4] + e[5]) + c[4+1, 5] = 30 + 75 + 30 = 135$$

Simulação

$e[1]=10$ $e[2]=20$ $e[3]=30$ $e[4]=15$ $e[5]=30$

	0	1	2	3	4	5	<i>j</i>
1	0	10	40	100			
2		0	20	70	100		
3			0	30	60	135	
4				0	15	60	
5					0	30	
6						0	
<i>i</i>							

$$c[3, 5-1] + (e[3] + e[4] + e[5]) + c[5+1, 5] = 60 + 75 + 0 = 135$$

Simulação

$e[1]=10$ $e[2]=20$ $e[3]=30$ $e[4]=15$ $e[5]=30$

	0	1	2	3	4	5	<i>j</i>
1	0	10	40	100			
2		0	20	70	100		
3			0	30	60	135	
4				0	15	60	
5					0	30	
6						0	

i

Exercício: Preencha o que falta!

Árvore de busca ótima

ABB-ÓTIMA (e, n)

```
1   $s[0] = 0$ 
2  para  $i \leftarrow 1$  até  $n$  faça
3       $s[i] \leftarrow s[i-1] + e[i]$ 
4  para  $i \leftarrow 1$  até  $n+1$  faça
5       $c[i, i-1] \leftarrow 0$ 
6  para  $\ell \leftarrow 1$  até  $n$  faça
7      para  $i \leftarrow 1$  até  $n-\ell+1$  faça
8           $j \leftarrow i+\ell-1$ 
9           $q \leftarrow c[i+1, j]$ 
10         para  $k \leftarrow i+1$  até  $j$  faça
11             se  $c[i, k-1] + c[k+1, j] < q$ 
12                 então  $q \leftarrow c[i, k-1] + c[k+1, j]$ 
13              $c[i, j] \leftarrow q + s[j] - s[i-1]$ 
14  devolva  $c[1, n]$ 
```

Árvore de busca ótima

Exercício: Como fazer para obter uma ABB ótima e não apenas o seu custo? Complete o serviço!

Mais programação dinâmica

KT 6.4

Aproveite para olhar todo o Cap 6 do KT,
que é sobre programação dinâmica.

= “recursão-com-tabela”

= transformação inteligente de recursão em iteração

Mochila

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

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

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

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

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.

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) .

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

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

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 $x[1..n-1]$ 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.

Simplificação

Problema:

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

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

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

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

$$x \cdot w \leq Y$$

Recorrência

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

$$x \cdot w \leq Y$$

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

Recorrência

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

$$x \cdot w \leq Y$$

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

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

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 $w[i] > Y$

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  $\max\{a, b\}$ 
```

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 $\Omega(2^n)$

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	Y
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

$W = 5$ 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						
2	0						
3	0						
4	0						

i

Exemplo

$W = 5$ 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					
2	0						
3	0						
4	0						
i							

Exemplo

$W = 5$ 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				
2	0						
3	0						
4	0						
i							

Exemplo

$W = 5$ 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			
2	0						
3	0						
4	0						

i

Exemplo

$W = 5$ 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		
2	0						
3	0						
4	0						

i

Exemplo

$W = 5$ 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						
3	0						
4	0						
i							

Exemplo

$W = 5$ 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					
3	0						
4	0						
i							

Exemplo

$W = 5$ 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				
3	0						
4	0						

i

Exemplo

$W = 5$ 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			
3	0						
4	0						

i

Exemplo

$W = 5$ 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		
3	0						
4	0						

i

Exemplo

$W = 5$ 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						
4	0						
i							

Exemplo

$W = 5$ 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	700	700	800	
4	0	300	400	700	700	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  $t[n, W]$ 
```

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 $\Theta(nW)$.

Conclusão

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

NOTA:

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

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)

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  $i \leftarrow n$  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  $x$ 
```

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 $t[i, Y] \leftarrow \infty$
- 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  $i = 0$  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  $t[i, Y]$ 
```