

## CLRS 15.2–15.3

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

Se  $A$  é  $p \times q$  e  $B$  é  $q \times r$  então  $AB$  é  $p \times r$ .

$$(AB)[i,j] = \sum_k A[i,k]B[k,j]$$

MULT-MAT ( $p, A, q, B, r$ )

```

1  para  $i \leftarrow 1$  até  $p$  faça
2      para  $j \leftarrow 1$  até  $r$  faça
3           $AB[i,j] \leftarrow 0$ 
4          para  $k \leftarrow 1$  até  $q$  faça
5               $AB[i,j] \leftarrow AB[i,j] + A[i,k] \cdot B[k,j]$ 
    
```

Número de multiplicações escalares =  $p \cdot q \cdot r$

## Multiplicação iterada

**Problema:** Encontrar número mínimo de multiplicações escalares necessário para calcular produto  $A_1 A_2 \cdots A_n$ .

$p[0]$	$p[1]$	$p[2]$	...	$p[n-1]$	$p[n]$
$A_1$	$A_2$	...	...	$A_n$	

cada  $A_i$  é  $p[i-1] \times p[i]$  ( $A_i[1..p[i-1], 1..p[i]]$ )

**Exemplo:**  $A_1 \cdot A_2 \cdot A_3$

	10	$A_1$	100	$A_2$	5	$A_3$	50
$((A_1 A_2) A_3)$			7500				multiplicações escalares
$(A_1 (A_2 A_3))$			75000				multiplicações escalares

## Soluções ótimas contêm soluções ótimas

Se

$$(A_1 A_2)(A_3((A_4 A_5) A_6))$$

é ordem ótima de multiplicação então

$$(A_1 A_2) \text{ e } (A_3((A_4 A_5) A_6))$$

também são ordens ótimas.

**Decomposição:**  $(A_1 \cdots A_k)(A_{k+1} \cdots A_j)$

$m[i,j]$  = número mínimo de multiplicações escalares para calcular  $A_i \cdots A_j$

## Recorrência

$m[i, j]$  = número mínimo de multiplicações escalares para calcular  $A_i \cdots A_j$

se  $i = j$  então  $m[i, j] = 0$

se  $i < j$  então

$$m[i, j] = \min_{i \leq k < j} \{m[i, k] + p[i-1]p[k]p[j] + m[k+1, j]\}$$

Exemplo:

$$m[3, 7] = \min_{3 \leq k < 7} \{m[3, k] + p[2]p[k]p[7] + m[k+1, 7]\}$$

## Algoritmo recursivo

Recebe  $p[i-1..j]$  e devolve  $m[i, j]$

REC-MAT-CHAIN ( $p, i, j$ )

```

1  se  $i = j$ 
2    então devolva 0
3   $m[i, j] \leftarrow \infty$ 
4  para  $k \leftarrow i$  até  $j - 1$  faça
5     $q_1 \leftarrow \text{REC-MAT-CHAIN}(p, i, k)$ 
6     $q_2 \leftarrow \text{REC-MAT-CHAIN}(p, k + 1, j)$ 
7     $q \leftarrow q_1 + p[i-1]p[k]p[j] + q_2$ 
8    se  $q < m[i, j]$ 
9      então  $m[i, j] \leftarrow q$ 
10 devolva  $m[i, j]$ 

```

Consumo de tempo?

## Consumo de tempo

$T(n)$  = número comparações entre  $q$  e  $m[\star, \star]$  na linha 8 quando  $n := j - i + 1$

$$T(1) = 0$$

$$\begin{aligned}
 T(n) &= \sum_{h=1}^{n-1} (T(h) + T(n-h) + 1) = 2 \sum_{h=2}^{n-1} T(h) + (n-1) \\
 &= 2(T(2) + \cdots + T(n-1)) + (n-1) \text{ para } n \geq 2
 \end{aligned}$$

## Consumo de tempo

$T(n)$  = número comparações entre  $q$  e  $m[\star, \star]$  na linha 8 quando  $n := j - i + 1$

$$T(n) = 2(T(2) + \cdots + T(n-1)) + (n-1)$$

Considere a mesma fórmula para  $n-1$ :

$$T(n-1) = 2(T(2) + \cdots + T(n-2)) + (n-2)$$

e subtraia a primeira da segunda:

$$T(n) - T(n-1) = 2T(n-1) + 1.$$

Logo  $T(n) = 3T(n-1) + 1$ .

Fácil verificar que  $T(n) \geq \frac{3^{n-1}-1}{2}$  para  $n \geq 1$ .

# Recorrência

$n$	1	2	3	4	5	6	7	8
$T(n)$	0	1	4	13	40	121	364	1093
$3^{n-1} - 1$	0	2	8	26	80	242	728	2186

Prova: Para  $n = 1$ ,  $T(1) = 0 = (1 - 1)/2$ .

Para  $n \geq 2$ ,

$$\begin{aligned}
 T(n) &= 3T(n-1) + 1 \\
 &\stackrel{hi}{=} 3\left(\frac{3^{n-1} - 1}{2}\right) + 1 \\
 &= \frac{3^n - 3}{2} + 1 = \frac{3^n - 3 + 2}{2} \\
 &= \frac{3^n - 1}{2}.
 \end{aligned}$$

# Conclusão

$$T(n) \geq \frac{3^{n-1} - 1}{2} \text{ para } n \geq 1.$$

O consumo de tempo do algoritmo **REC-MAT-CHAIN** é  $\Omega(3^n)$ .

## Resolve subproblemas muitas vezes

$$p[0] = 10 \quad p[1] = 100 \quad p[2] = 5 \quad p[3] = 50$$

- REC-MAT-CHAIN(p, 1, 3)
  - REC-MAT-CHAIN(p, 1, 1)
  - REC-MAT-CHAIN(p, 2, 3)
    - REC-MAT-CHAIN(p, 2, 2)
    - REC-MAT-CHAIN(p, 3, 3)
  - REC-MAT-CHAIN(p, 1, 2)
    - REC-MAT-CHAIN(p, 1, 1)
    - REC-MAT-CHAIN(p, 2, 2)
  - REC-MAT-CHAIN(p, 3, 3)

Número mínimo de mults = 7500

## Resolve subproblemas muitas vezes

- |                        |                        |                        |
|------------------------|------------------------|------------------------|
| REC-MAT-CHAIN(p, 1, 5) | REC-MAT-CHAIN(p, 4, 4) | REC-MAT-CHAIN(p, 1, 1) |
| REC-MAT-CHAIN(p, 1, 1) | REC-MAT-CHAIN(p, 5, 5) | REC-MAT-CHAIN(p, 2, 4) |
| REC-MAT-CHAIN(p, 2, 5) | REC-MAT-CHAIN(p, 1, 2) | REC-MAT-CHAIN(p, 2, 2) |
| REC-MAT-CHAIN(p, 2, 2) | REC-MAT-CHAIN(p, 1, 1) | REC-MAT-CHAIN(p, 3, 4) |
| REC-MAT-CHAIN(p, 3, 5) | REC-MAT-CHAIN(p, 2, 2) | REC-MAT-CHAIN(p, 3, 3) |
| REC-MAT-CHAIN(p, 3, 3) | REC-MAT-CHAIN(p, 3, 5) | REC-MAT-CHAIN(p, 4, 4) |
| REC-MAT-CHAIN(p, 4, 5) | REC-MAT-CHAIN(p, 3, 3) | REC-MAT-CHAIN(p, 2, 3) |
| REC-MAT-CHAIN(p, 4, 4) | REC-MAT-CHAIN(p, 4, 5) | REC-MAT-CHAIN(p, 2, 2) |
| REC-MAT-CHAIN(p, 5, 5) | REC-MAT-CHAIN(p, 4, 4) | REC-MAT-CHAIN(p, 3, 3) |
| REC-MAT-CHAIN(p, 3, 4) | REC-MAT-CHAIN(p, 5, 5) | REC-MAT-CHAIN(p, 4, 4) |
| REC-MAT-CHAIN(p, 3, 3) | REC-MAT-CHAIN(p, 3, 4) | REC-MAT-CHAIN(p, 1, 2) |
| REC-MAT-CHAIN(p, 4, 4) | REC-MAT-CHAIN(p, 3, 3) | REC-MAT-CHAIN(p, 1, 1) |
| REC-MAT-CHAIN(p, 5, 5) | REC-MAT-CHAIN(p, 4, 4) | REC-MAT-CHAIN(p, 2, 2) |
| REC-MAT-CHAIN(p, 2, 3) | REC-MAT-CHAIN(p, 5, 5) | REC-MAT-CHAIN(p, 3, 4) |
| REC-MAT-CHAIN(p, 2, 2) | REC-MAT-CHAIN(p, 1, 3) | REC-MAT-CHAIN(p, 3, 3) |
| REC-MAT-CHAIN(p, 3, 3) | REC-MAT-CHAIN(p, 1, 1) | REC-MAT-CHAIN(p, 4, 4) |
| REC-MAT-CHAIN(p, 4, 5) | REC-MAT-CHAIN(p, 2, 3) | REC-MAT-CHAIN(p, 1, 3) |
| REC-MAT-CHAIN(p, 4, 4) | REC-MAT-CHAIN(p, 2, 2) | REC-MAT-CHAIN(p, 1, 1) |
| REC-MAT-CHAIN(p, 5, 5) | REC-MAT-CHAIN(p, 3, 3) | REC-MAT-CHAIN(p, 2, 3) |
| REC-MAT-CHAIN(p, 2, 4) | REC-MAT-CHAIN(p, 1, 2) | REC-MAT-CHAIN(p, 2, 2) |
| REC-MAT-CHAIN(p, 2, 2) | REC-MAT-CHAIN(p, 1, 1) | REC-MAT-CHAIN(p, 3, 3) |
| REC-MAT-CHAIN(p, 3, 4) | REC-MAT-CHAIN(p, 2, 2) | REC-MAT-CHAIN(p, 1, 2) |
| REC-MAT-CHAIN(p, 3, 3) | REC-MAT-CHAIN(p, 3, 3) | REC-MAT-CHAIN(p, 1, 1) |
| REC-MAT-CHAIN(p, 4, 4) | REC-MAT-CHAIN(p, 4, 5) | REC-MAT-CHAIN(p, 2, 2) |
| REC-MAT-CHAIN(p, 2, 3) | REC-MAT-CHAIN(p, 4, 4) | REC-MAT-CHAIN(p, 3, 3) |
| REC-MAT-CHAIN(p, 2, 2) | REC-MAT-CHAIN(p, 5, 5) | REC-MAT-CHAIN(p, 4, 4) |
| REC-MAT-CHAIN(p, 3, 3) | REC-MAT-CHAIN(p, 1, 4) | REC-MAT-CHAIN(p, 5, 5) |

# Programação dinâmica

Cada subproblema

$$A_i \dots A_j$$

é resolvido **uma só** vez.

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

Para calcular  $m[2,6]$  preciso de ...

$m[2,2], m[2,3], m[2,4], m[2,5]$  e de  $m[3,6], m[4,6], m[5,6], m[6,6]$ .

Calcule todos os  $m[i,j]$  com  $j - i + 1 = 2$ ,  
depois todos com  $j - i + 1 = 3$ ,  
depois todos com  $j - i + 1 = 4$ ,  
etc.

# Programação dinâmica

	1	2	3	4	5	6	7	8	$j$
1	0								
2		0	★	★	★	??			
3			0			★			
4				0		★			
5					0	★			
6						0			
7							0		
8								0	

$i$

## Simulação

$p[0]=10$   $p[1]=10$   $p[2]=20$   $p[3]=30$   $p[4]=10$   $p[5]=15$   $p[6]=30$

	1	2	3	4	5	6	$j$
1	0	??					
2		0					
3			0				
4				0			
5					0		
6						0	

$i$

	1	2	3	4	5	6	$j$
1	0	2000					
2		0					
3			0				
4				0			
5					0		
6						0	

## Algoritmo de programação dinâmica

Recebe  $p[0..n]$  e devolve  $m[1,n]$ .

**MATRIX-CHAIN-ORDER** ( $p, n$ )

```

1  para  $i \leftarrow 1$  até  $n$  faça
2       $m[i, i] \leftarrow 0$ 
3  para  $\ell \leftarrow 2$  até  $n$  faça
4      para  $i \leftarrow 1$  até  $n - \ell + 1$  faça
5           $j \leftarrow i + \ell - 1$ 
6           $m[i, j] \leftarrow \infty$ 
7          para  $k \leftarrow i$  até  $j - 1$  faça
8               $q \leftarrow m[i, k] + p[i - 1]p[k]p[j] + m[k + 1, j]$ 
9              se  $q < m[i, j]$ 
10                 então  $m[i, j] \leftarrow q$ 
11 devolva  $m[1, n]$ 
    
```

## Correção e consumo de tempo

Linhas 3–10: tratam subcadeias  $A_i \cdots A_j$  de comprimento  $\ell$

Consumo de tempo: ???

Consumo de tempo:  $O(n^3)$  (três loops encaixados)

Curioso verificar que consumo de tempo é  $\Omega(n^3)$ :

Número de execuções da linha 8:

$\ell$	$i$	execs linha 8
2	$1, \dots, n-1$	$(n-1) \cdot 1$
3	$1, \dots, n-2$	$(n-2) \cdot 2$
4	$1, \dots, n-3$	$(n-3) \cdot 3$
...	...	...
$n-1$	$1, 2$	$2 \cdot (n-2)$
$n$	$1$	$1 \cdot (n-1)$
total		$\sum_{h=1}^{n-1} h(n-h)$

## Consumo de tempo

Para  $n \geq 6$ ,  $\sum_{h=1}^{n-1} h(n-h) =$

$$= n \sum_{h=1}^{n-1} h - \sum_{h=1}^{n-1} h^2$$

$$= n \frac{1}{2} n(n-1) - \frac{1}{6} (n-1)n(2n-1) \quad (\text{CLRS p.1060})$$

$$\geq \frac{1}{2} n^2(n-1) - \frac{1}{6} 2n^3$$

$$\geq \frac{1}{2} n^2 \frac{5n}{6} - \frac{1}{3} n^3$$

$$= \frac{5}{12} n^3 - \frac{1}{3} n^3$$

$$= \frac{1}{12} n^3.$$

Consumo de tempo é  $\Omega(n^3)$

## Conclusão

O consumo de tempo do algoritmo **MATRIX-CHAIN-ORDER** é  $\Theta(n^3)$ .

## Versão recursiva eficiente

**MEMOIZED-MATRIX-CHAIN-ORDER** ( $p, n$ )

```
1 para  $i \leftarrow 1$  até  $n$  faça
2   para  $j \leftarrow 1$  até  $n$  faça
3      $m[i, j] \leftarrow \infty$ 
4 devolva LOOKUP-CHAIN ( $p, 1, n$ )
```

## Versão recursiva eficiente

LOOKUP-CHAIN ( $p, i, j$ )

```
1 se  $m[i, j] < \infty$ 
2   então devolva  $m[i, j]$ 
3 se  $i = j$ 
4   então  $m[i, j] \leftarrow 0$ 
5   senão para  $k \leftarrow i$  até  $j - 1$  faça
6      $q \leftarrow$  LOOKUP-CHAIN ( $p, i, k$ )
7       +  $p[i-1]p[k]p[j]$ 
8       + LOOKUP-CHAIN ( $p, k+1, j$ )
9     se  $q < m[i, j]$ 
10      então  $m[i, j] \leftarrow q$ 
11 devolva  $m[i, j]$ 
```

## Ingredientes de programação dinâmica

- ▶ **Subestrutura ótima:** soluções ótimas contêm soluções ótimas de subproblemas.
- ▶ **Subestrutura:** decomponha o problema em subproblemas menores e, com sorte, mais simples.
- ▶ **Bottom-up:** combine as soluções dos problemas menores para obter soluções dos maiores.
- ▶ **Tabela:** armazene as soluções dos subproblemas em uma tabela, pois soluções dos subproblemas são consultadas várias vezes.
- ▶ **Número de subproblemas:** para a eficiência do algoritmo é importante que o número de subproblemas resolvidos seja 'pequeno'.
- ▶ **Memoized:** versão *top-down*, recursão com tabela.

## Exercício

O algoritmo **MATRIX-CHAIN-ORDER** determina o **número mínimo** de multiplicações escalares necessário para calcular produto  $A_1 A_2 \cdots A_n$ .

Na aula, mencionamos uma maneira de obter uma parentização ótima a partir dos cálculos feitos, usando para isso um dado a mais que podemos guardar no decorrer do algoritmo.

Faça os ajustes sugeridos na aula, de modo a guardar esse dado extra, e devolvê-lo junto com o valor  $m[1, n]$ .

Faça uma rotina que recebe a informação extra armazenada pelo algoritmo acima e imprime uma parentização ótima das matrizes  $A_1 A_2 \cdots A_n$ .

## Exercícios

**Exercício 13.A** [CLRS 15.2-1]

Encontre a maneira ótima de fazer a multiplicação iterada das matrizes cujas dimensões são (5, 10, 3, 12, 5, 50, 6).

**Exercício 13.B** [CLRS 15.2-5]

Mostre que são necessários exatamente  $n - 1$  pares de parênteses para especificar exatamente a ordem de multiplicação de  $A_1 \cdot A_2 \cdots A_n$ .

**Exercício 13.C** [CLRS 15.3-2]

Desenhe a árvore de recursão para o algoritmo **MERGE-SORT** aplicado a um vetor de 16 elementos. Por que a técnica de programação dinâmica não é capaz de acelerar o algoritmo?

## Exercício 13.D [CLRS 15.3-5 expandido]

Considere o seguinte algoritmo para determinar a ordem de multiplicação de uma cadeia de matrizes  $A_1, A_2, \dots, A_n$  de dimensões  $p_0, p_1, \dots, p_n$ : primeiro, escolha  $k$  que minimize  $p_k$ ; depois, determine recursivamente as ordens de multiplicação de  $A_1, \dots, A_k$  e  $A_{k+1}, \dots, A_n$ . Esse algoritmo produz uma ordem que minimiza o número total de multiplicações escalares? E se  $k$  for escolhido de modo a maximizar  $p_k$ ? E se  $k$  for escolhido de modo a minimizar  $p_k$ ?

## Exercício 13.E

Prove que o número de execuções da linha 9 em `MATRIX-CHAIN-ORDER` é  $O(n^3)$ .

## CLRS 15.5

= “recursão-com-tabela”

= transformação inteligente de recursão em iteração

## Buscas em um conjunto conhecido

Considere um inteiro  $n$  e um vetor  $v[1..n]$  de tipo `Ord`.

**Problema:** Dado  $v[1..n]$  e uma sequência de  $k$  elementos do tipo `Ord`, decidir se cada elemento está ou não em  $v$ .

Se  $k$  é grande, como devemos armazenar o  $v$ ?

E se  $v$  armazena um conjunto bem conhecido, como por exemplo as palavras de uma língua? (A ser usado por um tradutor, ou um speller.)

Podemos ordenar  $v$  e aplicar busca binária.

Podemos fazer algo melhor?

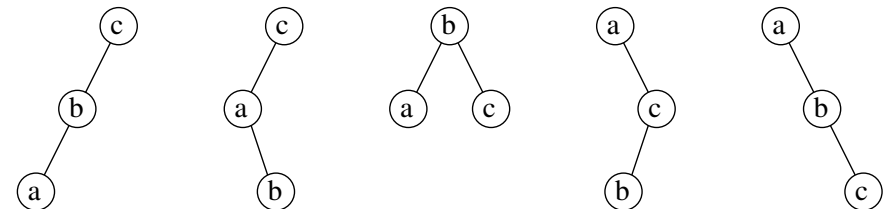
## Buscas em conjunto conhecido

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)?

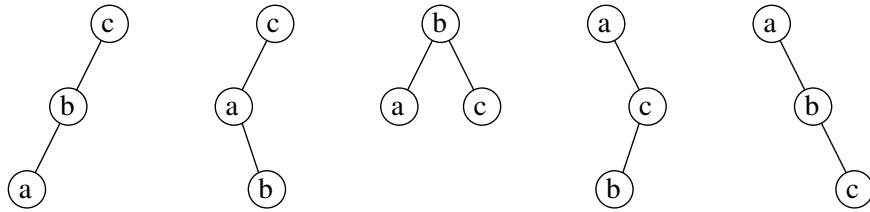
**Exemplo:**  $n = 3$  e  $e_1 = 10$ ,  $e_2 = 20$ ,  $e_3 = 40$ .

Qual a melhor das ABBs?



# Árvore de busca ótima

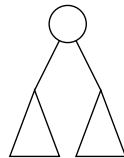
**Exemplo:**  $n = 3$  e  $e_1 = 10, e_2 = 20, e_3 = 40$ .



Qual a melhor das ABBs? Número esperado de comparações:

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

## 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  $e[i..j]$

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

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  $e$ .

## 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  $s[i] \leftarrow s[i-1] + e[i]$

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

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

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

## 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!