Melhores momentos

AULA PASSADA

Ingredientes de programação dinâmica

- Subestrutura ótima: soluções ótimas contém 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.

AULA 14

Mais programação dinâmica

CLRS 15.4

= “recursão–com–tabela”

= transformação inteligente de recursão em iteração

Subseqüências

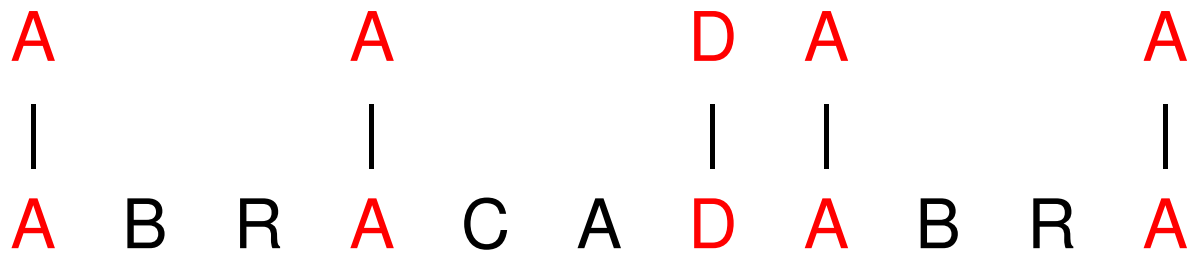
$\langle z_1, \ldots, z_k \rangle$ é subseqüência de $\langle x_1, \ldots, x_m \rangle$
se existem índices $i_1 < \cdots < i_k$ tais que

$$z_1 = x_{i_1} \quad \ldots \quad z_k = x_{i_k}$$

EXEMPLOS:

$\langle 5, 9, 2, 7 \rangle$ é subseqüência de $\langle 9, 5, 6, 9, 6, 2, 7, 3 \rangle$

⟨A,A, D, A, A⟩ é subseqüência de
⟨A, B, R, A, C, A, D, A, B, R, A⟩



Exercício

Problema: Decidir se $Z[1\,..\,m]$ é subseqüência de $X[1\,..\,n]$

Exercício

Problema: Decidir se $Z[1\,..\,m]$ é subseqüência de $X[1\,..\,n]$

```
SUB-SEQ- (Z, m, X, n)
1    i ← m
2    j ← n
3    enquanto i ≥ 1 e j ≥ 1 faça
4         se Z[i] = X[j]
5              então i ← i − 1
6         j ← j − 1
7    se i ≥ 1
8         então devolva "não é subseqüência"
9         senão devolva "é subseqüência"
```

Exercício

Problema: Decidir se $Z[1\,..\,m]$ é subseqüência de $X[1\,..\,n]$

```
SUB-SEQ- (Z, m, X, n)
1    i ← m
2    j ← n
3    enquanto i ≥ 1 e j ≥ 1 faça
4         se Z[i] = X[j]
5              então i ← i − 1
6         j ← j − 1
7    se i ≥ 1
8         então devolva "não é subseqüência"
9         senão devolva "é subseqüência"
```

Consumo de tempo é $\mathrm{O}(m+n)$ e $\Omega(\min\{m,n\})$.

Exercício

Problema: Decidir se $Z[1\,..\,m]$ é subseqüência de $X[1\,..\,n]$

```
SUB-SEQ- (Z, m, X, n)
1   i ← m
2   j ← n
3   enquanto i ≥ 1 e j ≥ 1 faça
4         se Z[i] = X[j]
5               então i ← i − 1
6         j ← j − 1
7   se i ≥ 1
8         então devolva "não é subseqüência"
9         senão devolva "é subseqüência"
```

Invariantes:

(i0) $Z[i{+}1\,..\,m]$ é subseqüência de $X[j{+}1\,..\,n]$

(i1) $Z[i\,..\,m]$ **não** é subseqüência de $X[j{+}1\,..\,n]$

Subseqüência comum máxima

Z é subseq comum de X e Y

se Z é subseqüência comum de X e de Y

ssco $=$ subseqüência comum

Exemplos: X = A B C B D A B

Y = B D C A B A

ssco = B C A

Outra ssco = B D A B

Problema

Problema: Encontrar uma ssco máxima de X e Y.

Exemplos: X = A B C B D A B

Y = B D C A B A

ssco = B C A

ssco maximal = A B A

ssco máxima = B C A B

Outra ssco máxima = B D A B

LCS = Longest Common Subsequence

diff

```
> more abracadabra
A
B
R
A
C
A
D
A
B
R
A
```

```
> more yabbadabbadoo
Y
A
B
B
A
D
A
B
B
A
D
O
O
```

diff -u abracadabra yabbadabbadoo

```
+Y
 A
 B
-R
-A
-C
+B
 A
 D
 A
 B
-R
+B
 A
+D
+O
+O
```

Subestrutura ótima

Suponha que $Z[1\,..\,k]$ é ssco máxima de $X[1\,..\,m]$ e $Y[1\,..\,n]$.

- Se $X[m] = Y[n]$, então $Z[k] = X[m] = Y[n]$ e $Z[1\,..\,k{-}1]$ é ssco máxima de $X[1\,..\,m{-}1]$ e $Y[1\,..\,n{-}1]$.
- Se $X[m] \neq Y[n]$, então $Z[k] \neq X[m]$ implica que $Z[1\,..\,k]$ é ssco máxima de $X[1\,..\,m{-}1]$ e $Y[1\,..\,n]$.
- Se $X[m] \neq Y[n]$, então $Z[k] \neq Y[n]$ implica que $Z[1\,..\,k]$ é ssco máxima de $X[1\,..\,m]$ e $Y[1\,..\,n{-}1]$.

Algoritmo recursivo

Devolve o comprimento de uma ssco máxima de $X[1\,..\,i]$ e $Y[1\,..\,j]$.

```
REC-LCS-LENGTH (X, i, Y, j)
 1  se i = 0 ou j = 0
 2       então devolva 0
 3  se X[i] = Y[j]
 4       então c ←REC-LCS-LENGTH (X, i−1, Y, j−1)
                                  +1
 5       senão q1 ← REC-LCS-LENGTH (X, i−1, Y, j)
 6             q2 ← REC-LCS-LENGTH (X, i, Y, j−1)
 7             se q1 ≥ q2
 8                  então c ← q1
 9                  senão c ← q2
10  devolva c
```

Consumo de tempo

$T(m, n) :=$ número máximo de comparações feitas por REC-LCS-LENGTH (X, m, Y, n)

Recorrência

$$T(0, n) = 0$$

$$T(m, 0) = 0$$

$$T(m, n) = T(m-1, n) + T(m, n-1) + 1 \text{ para } n \geq 0 \text{ e } m \geq 0$$

A que classe Ω pertence $T(m, n)$?

Recorrência

Note que $T(m,n) = T(n,m)$ para $n = 0, 1, \ldots$ e $m = 0, 1, \ldots$.

Seja $k := \min\{m, n\}$. Temos que

$$T(m,n) \geq T(k,k) \geq S(k),$$

onde

$$S(0) = 0$$
$$S(k) = 2S(k-1) + 1 \text{ para } k = 1, 2, \ldots$$

$S(k)$ é $\Theta(2^k) \Rightarrow T(m,n)$ é $\Omega(2^{\min\{m,n\}})$

$T(m,n)$ é exponencial

Conclusão

O consumo de tempo do algoritmo REC-LCS-LENGTH é $\Omega(2^{\min\{m,n\}})$.

Fórmula fechada

Prove que

$$T(m,n) = \binom{m+n}{m} - 1 \ .$$

Logo,

$$T(m,m) = \binom{2m}{m} - 1$$

$$> \frac{4^m}{2m+1} - 1.$$

Portanto, $T(m,m)$ é $\Omega(4^m/m)$.

Programação dinâmica

Problema: encontrar o comprimento de uma ssco máxima.

$c[i, j] =$ comprimento de uma ssco máxima de $X[1\,..\,i]$ e $Y[1\,..\,j]$

Recorrência:

$c[0, j] = c[i, 0] = 0$

$c[i, j] = c[i-1, j-1] + 1$ se $X[i] = Y[j]$

$c[i, j] = \max(c[i, j-1], c[i-1, j])$ se $X[i] \neq Y[j]$

Programação dinâmica

Cada subproblema, comprimento de uma ssco máxima de

$$X[1\,..\,i] \quad \text{e} \quad Y[1\,..\,j],$$

é resolvido uma só vez.

Em que ordem calcular os componentes da tabela c?

Para calcular $c[3, 5]$ preciso de ...

Programação dinâmica

Cada subproblema, comprimento de uma ssco máxima de

$$X[1\,..\,i] \quad \text{e} \quad Y[1\,..\,j],$$

é resolvido uma só vez.

Em que ordem calcular os componentes da tabela c?

Para calcular $c[3,5]$ preciso de ...

$c[3,4]$, $c[2,5]$ e de $c[2,4]$.

Programação dinâmica

Cada subproblema, comprimento de uma ssco máxima de

$$X[1\,..\,i] \quad \text{e} \quad Y[1\,..\,j],$$

é resolvido uma só vez.

Em que ordem calcular os componentes da tabela c?

Para calcular $c[3,5]$ preciso de ...

$c[3,4]$, $c[2,5]$ e de $c[2,4]$.

Calcule todos os $c[i,j]$ com $i = 1, j = 0, 1, \ldots, n$,
depois todos com $i = 2, j = 0, 1, \ldots, n$,
depois todos com $i = 3, j = 0, 1, \ldots, n$,
etc.

Programação dinâmica

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0							
2	0				$\star$	$\star$		
3	0				$\star$	??		
4	0							
5	0							
6	0							
7	0							

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	??						
B	2	0							
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	??					
B	2	0							
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	??				
B	2	0							
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	??			
B	2	0							
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	??		
B	2	0							
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	??	
B	2	0							
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	??						
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	??					
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	??				
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	??			
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	??		
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	??	
C	3	0							
B	4	0							
D	5	0							
A	6	0							
B	7	0							

i

Simulação

			B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	??						
B	4	0							
D	5	0							
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	??					
B	4	0							
D	5	0							
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	??				
B	4	0							
D	5	0							
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	??			
B	4	0							
D	5	0							
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	??		
B	4	0							
D	5	0							
A	6	0							
B	7	0							

i

Simulação

		Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	??	
B	4	0							
D	5	0							
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	??						
D	5	0							
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	??					
D	5	0							
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	??				
D	5	0							
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	??			
D	5	0							
A	6	0							
B	7	0							

i

Simulação

		Y		B	D	C	A	B	A	
X			0	1	2	3	4	5	6	j
	0		0	0	0	0	0	0	0	
A	1		0	0	0	0	1	1	1	
B	2		0	1	1	1	1	2	2	
C	3		0	1	1	2	2	2	2	
B	4		0	1	1	2	2	??		
D	5		0							
A	6		0							
B	7		0							

i

Simulação

		Y		B	D	C	A	B	A	
X			0	1	2	3	4	5	6	j
	0		0	0	0	0	0	0	0	
A	1		0	0	0	0	1	1	1	
B	2		0	1	1	1	1	2	2	
C	3		0	1	1	2	2	2	2	
B	4		0	1	1	2	2	3	??	
D	5		0							
A	6		0							
B	7		0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	??						
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	??					
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	??				
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	??			
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	??		
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	??	
A	6	0							
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	??						
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	??					
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	??				
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	2	??			
B	7	0							

i

Simulação

			B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	2	3	??		
B	7	0							

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	2	3	3	??	
B	7	0							

i

Simulação

			B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	2	3	3	4	
B	7	0	??						

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	2	3	3	4	
B	7	0	1	??					

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	2	3	3	4	
B	7	0	1	2	??				

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	2	3	3	4	
B	7	0	1	2	2	??			

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	2	3	3	4	
B	7	0	1	2	2	3	??		

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	2	3	3	4	
B	7	0	1	2	2	3	4	??	

i

Simulação

	Y		B	D	C	A	B	A	
X		0	1	2	3	4	5	6	j
	0	0	0	0	0	0	0	0	
A	1	0	0	0	0	1	1	1	
B	2	0	1	1	1	1	2	2	
C	3	0	1	1	2	2	2	2	
B	4	0	1	1	2	2	3	3	
D	5	0	1	2	2	2	3	3	
A	6	0	1	2	2	3	3	4	
B	7	0	1	2	2	3	4	4	

i

Algoritmo de programação dinâmica

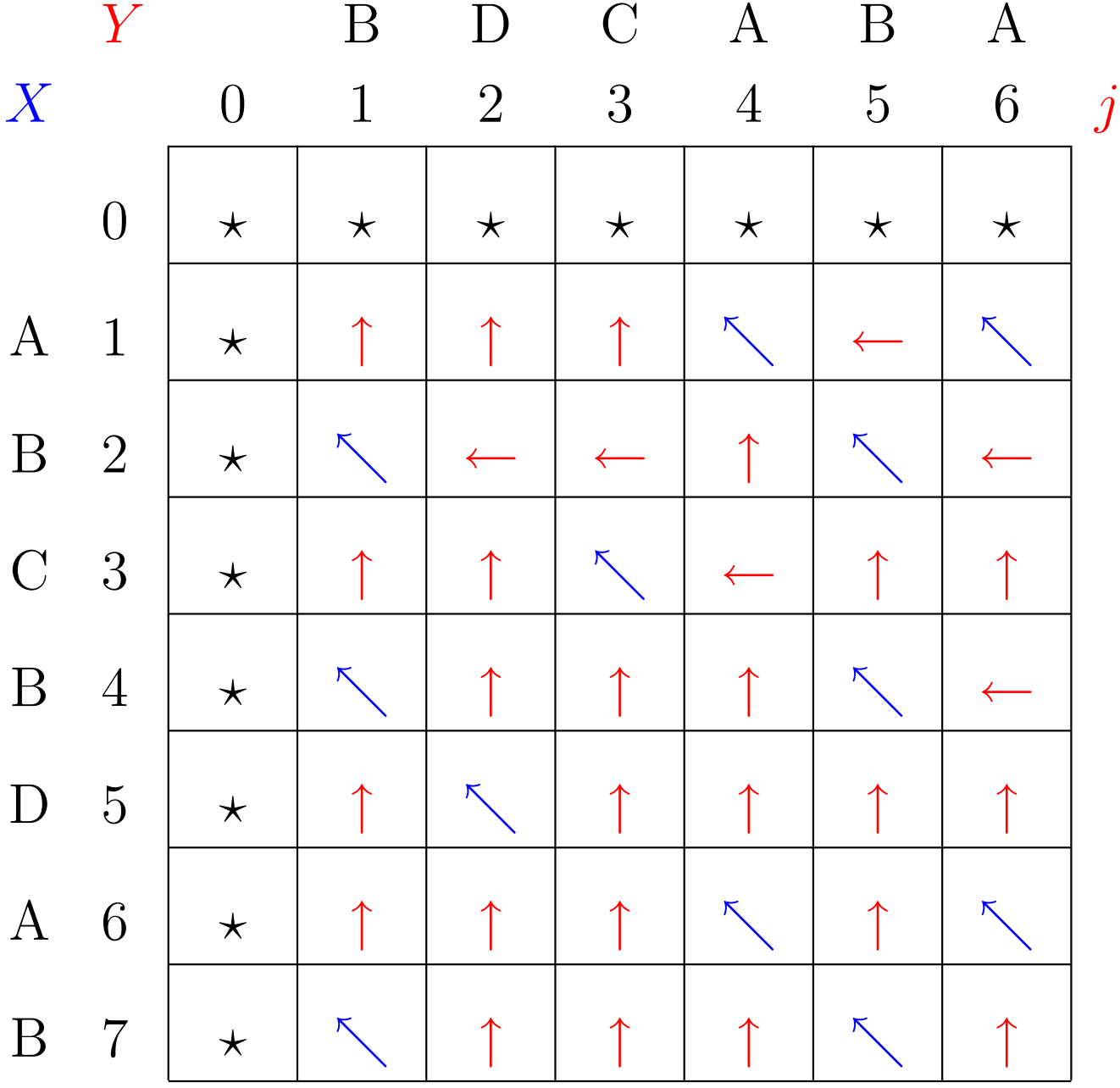
Devolve o comprimento de uma ssco máxima de $X[1\,..\,m]$ e $Y[1\,..\,n]$.

```
LCS-LENGTH (X, m, Y, n)
 1  para i ← 0 até m faça
 2        c[i, 0] ← 0
 3  para j ← 1 até n faça
 4        c[0, j] ← 0
 5  para i ← 1 até m faça
 6        para j ← 1 até n faça
 7              se X[i] = Y[j]
 8                    então c[i, j] ← c[i − 1, j − 1] + 1
 9                    senão se c[i − 1, j] ≥ c[i, j − 1]
10                                 então c[i, j] ← c[i − 1, j]
11                                 senão c[i, j] ← c[i, j − 1]
12  devolva c[m, n]
```

Conclusão

O consumo de tempo do algoritmo LCS-LENGTH é $\Theta(mn)$.

Subseqüência comum máxima



Algoritmo de programação dinâmica

```
LCS-LENGTH (X, m, Y, n)
 1   para i ← 0 até m faça
 2       c[i, 0] ← 0
 3   para j ← 1 até n faça
 4       c[0, j] ← 0
 5   para i ← 1 até m faça
 6       para j ← 1 até n faça
 7           se X[i] = Y[j]
 8               então c[i, j] ← c[i − 1, j − 1] + 1
 8                     b[i, j] ← "↖"
 9               senão se c[i − 1, j] ≥ c[i, j − 1]
10                         então c[i, j] ← c[i − 1, j]
10                               b[i, j] ← "↑"
11                         senão c[i, j] ← c[i, j − 1]
11                               b[i, j] ← "←"
12   devolva c e b
```

Get-LCS

```
GET-LCS (X, m, n, b, máxcomp)
 1   k ← máxcomp
 2   i ← m
 2   j ← n
 3   enquanto i > 0 e j > 0 faça
 4         se b[i, j] = "↖"
 5               então Z[k] ← X[i]
 6                     k ← k − 1    i ← i − 1    j ← j − 1
 9               senão se b[i, j] = "←"
10                           então j ← j − 1
11                           senão i ← i − 1
12   devolva Z
```

Consumo de tempo é $\mathrm{O}(m+n)$ e $\Omega(\min\{m,n\})$.

Versão recursiva eficiente

MEMOIZED-LCS-LENGTH (X, m, Y, n)

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

2 **para** $j \leftarrow 1$ **até** n **faça**

3 $c[i, j] \leftarrow \infty$

4 **devolva** LOOKUP-LCS (c, m, n)

Versão recursiva eficiente

```
LOOKUP-LCS (c, i, j)
 1   se c[i, j] < ∞
 2         então devolva c[i, j]
 3   se i = 0 ou j = 0 então c[i, j] ← 0
 4   senão se X[i] = Y[j]
 5               então c[i, j] ←LOOKUP-LCS (c, i−1, j−1)
                                          +1
 6               senão q1 ← LOOKUP-LCS (c, i−1, j)
 7                     q2 ← LOOKUP-LCS (c, i, j−1)
 8                     se q1 ≥ q2
 9                           então c[i, j] ← q1
10                           senão c[i, j] ← q2
11   devolva c[i, j]
```

Exercícios

Exercício 20.A

Escreva um algoritmo para decidir se $\langle z_1, \ldots, z_k \rangle$ é subseqüência de $\langle x_1, \ldots, x_m \rangle$. Prove rigorosamente que o seu algoritmo está correto.

Exercício 20.B

Suponha que os elementos de uma seqüência $\langle a_1, \ldots, a_n \rangle$ são distintos dois a dois. Quantas subseqüências tem a seqüência?

Exercício 20.C

Uma subseqüência crescente Z de uma seqüência X e é *máxima* se não existe outra subseqüência crescente mais longa. A subseqüência $\langle 5, 6, 9 \rangle$ de $\langle 9, 5, 6, 9, 6, 2, 7 \rangle$ é máxima? Dê uma seqüência crescente máxima de $\langle 9, 5, 6, 9, 6, 2, 7 \rangle$. Mostre que o algoritmo "guloso" óbvio não é capaz, em geral, de encontrar uma subseqüência crescente máxima de uma seqüência dada. (Algoritmo guloso óbvio: escolha o menor elemento de X; a partir daí, escolha sempre o próximo elemento de X que seja maior ou igual ao último escolhido.)

Exercício 20.D

Escreva um algoritmo de programação dinâmica para resolver o problema da subseqüência crescente máxima.

Mais exercícios

Exercício 20.E [CLRS 15.4-5]
Mostre como o algoritmo da subseqüência comum máxima pode ser usado para resolver o problema da subseqüência crescente máxima de uma seqüência numérica. Dê uma delimitação justa, em notação Θ, do consumo de tempo de sua solução.

Exercício 20.F [Printing neatly. CLRS 15-2]
Considere a seqüência $P_1, P_2, \ldots, P_n$ de palavras que constitui um parágrafo de texto. A palavra P_i tem l_i caracteres. Queremos imprimir as palavras em linhas, na ordem dada, de modo que cada linha tenha no máximo M caracteres. Se uma determinada linha contém as palavras $P_i, P_{i+1}, \ldots, P_j$ (com $i \leq j$) e há exatamente um espaço entre cada par de palavras consecutivas, o número de espaços no fim da linha é

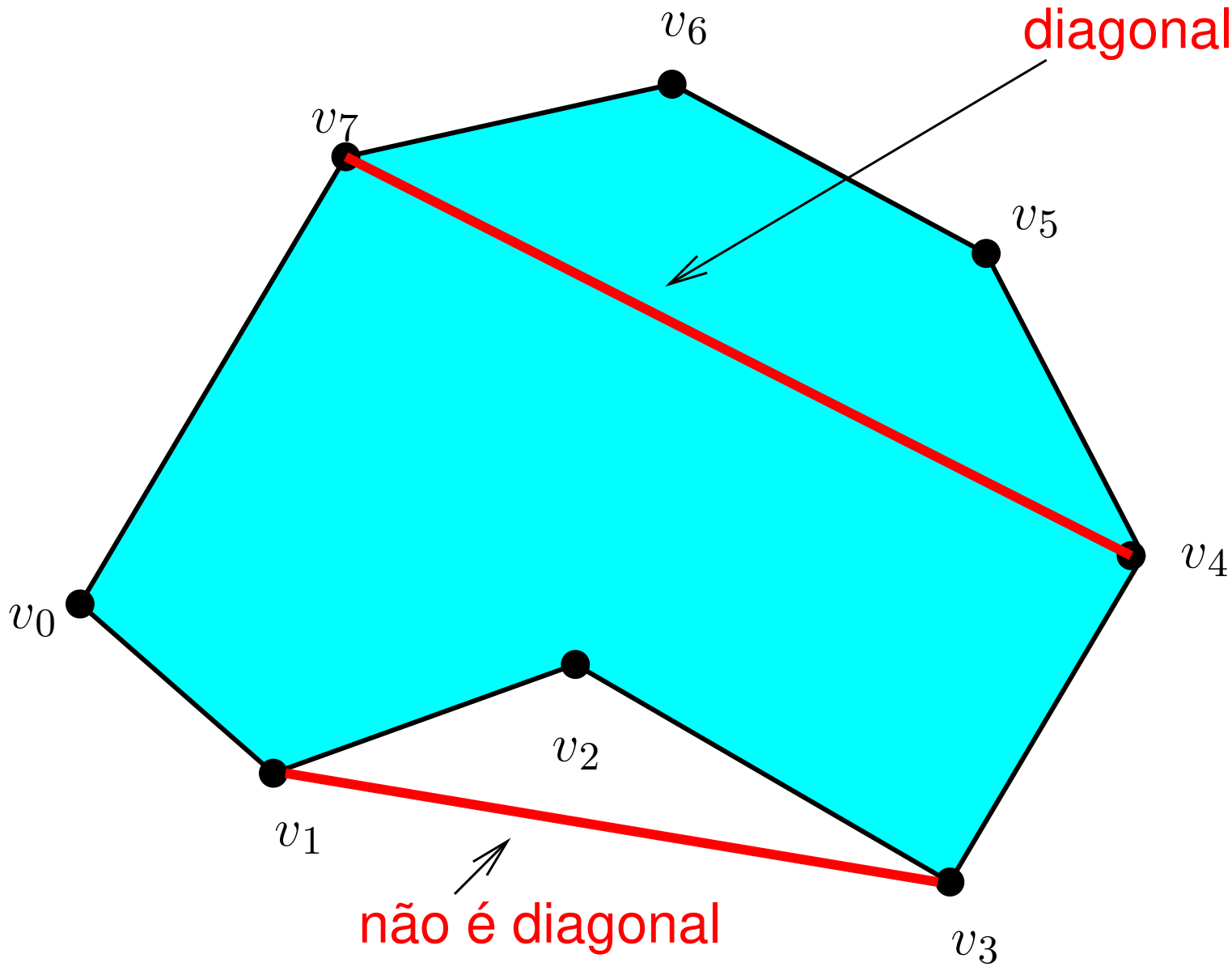
$$M - (l_i + 1 + l_{i+1} + 1 + \cdots + 1 + l_j)\,.$$

É claro que não devemos permitir que esse número seja negativo. Queremos minimizar, com relação a todas as linhas exceto a última, a soma dos cubos dos números de espaços no fim de cada linha. (Assim, se temos linhas $1, 2, \ldots, L$ e b_p espaços no fim da linha p, queremos minimizar $b_1^3 + b_2^3 + \cdots + b_{L-1}^3$).
Dê um exemplo para mostrar que algoritmos inocentes não resolvem o problema. Dê um algoritmo de programação dinâmica que resolva o problema. Qual a "optimal substructure property" para esse problema? Faça uma análise do consumo de tempo do algoritmo.

Mais programação dinâmica

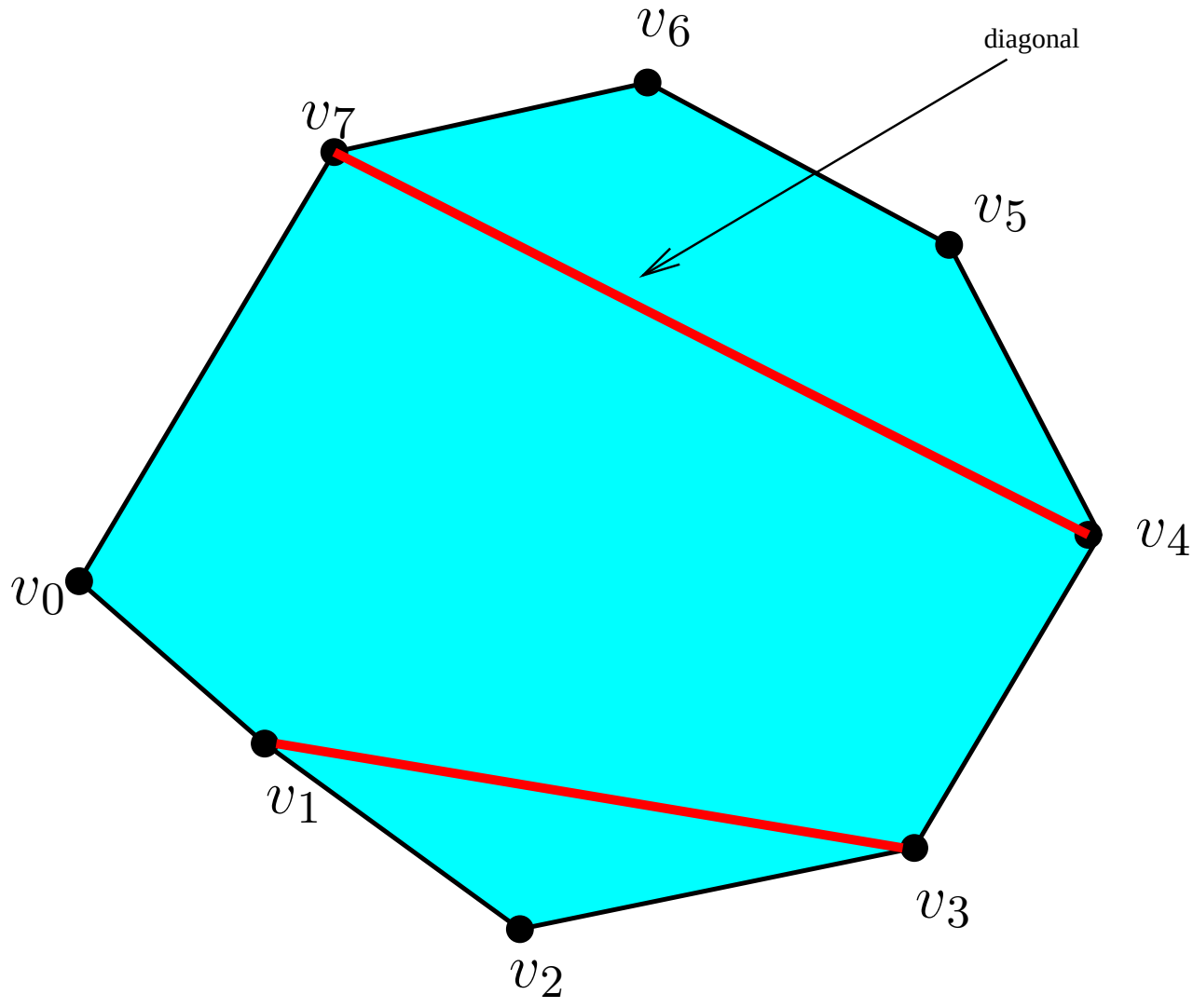
CLR 16.4

Polígono e diagonais



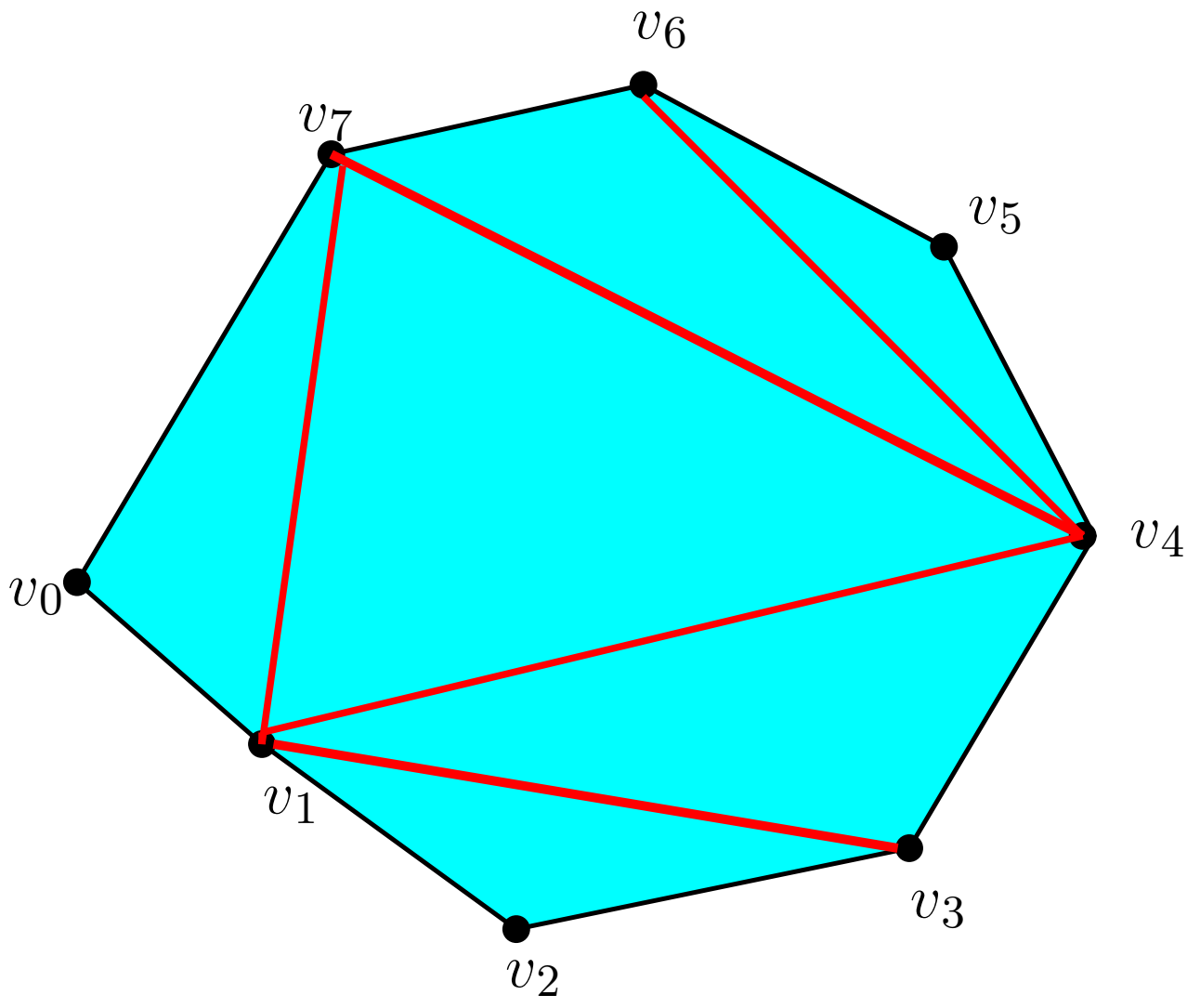
Polígono convexo

$P = \langle v_0, v_1, \ldots, v_{n-1}, v_n \rangle$ é convexo se todo segmento $v_i v_j$ é diagonal para todo $i \neq j \pm 1$.



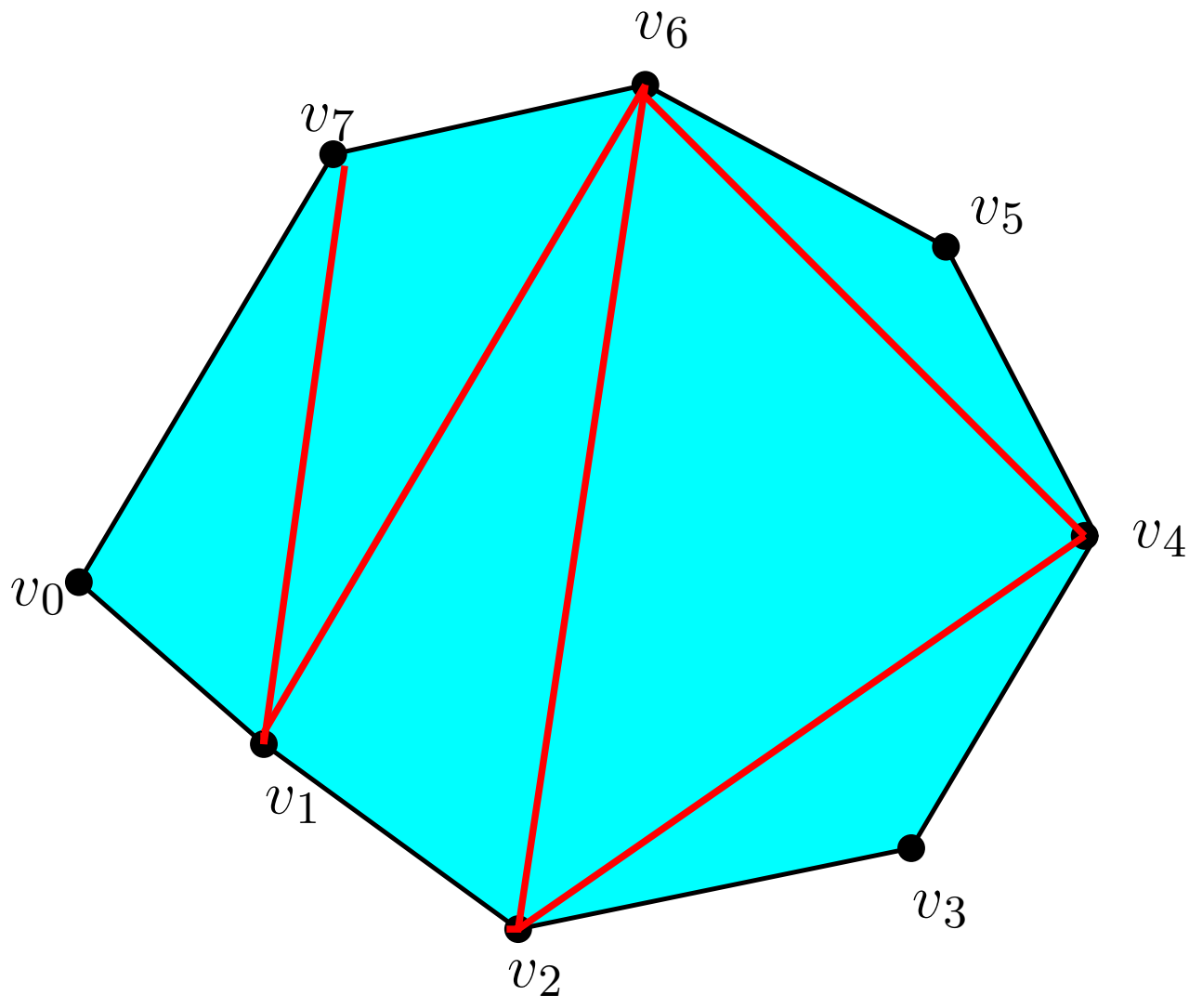
Triangulação de um polígono

Se colocarmos em um polígono P o maior número possível de diagonais que duas-a-duas não se cruzam obteremos uma triangulação de P.

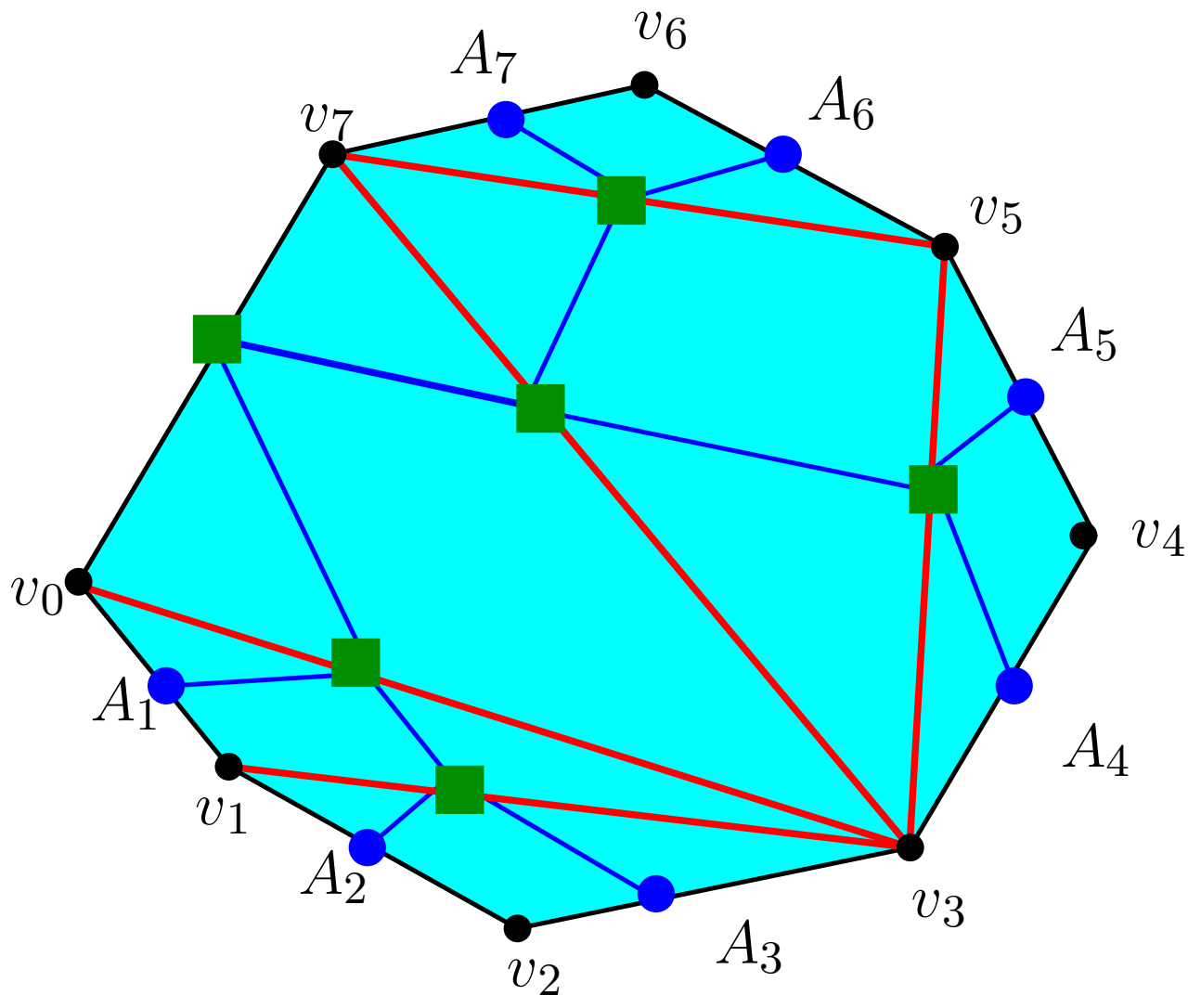


Triangulação de um polígono

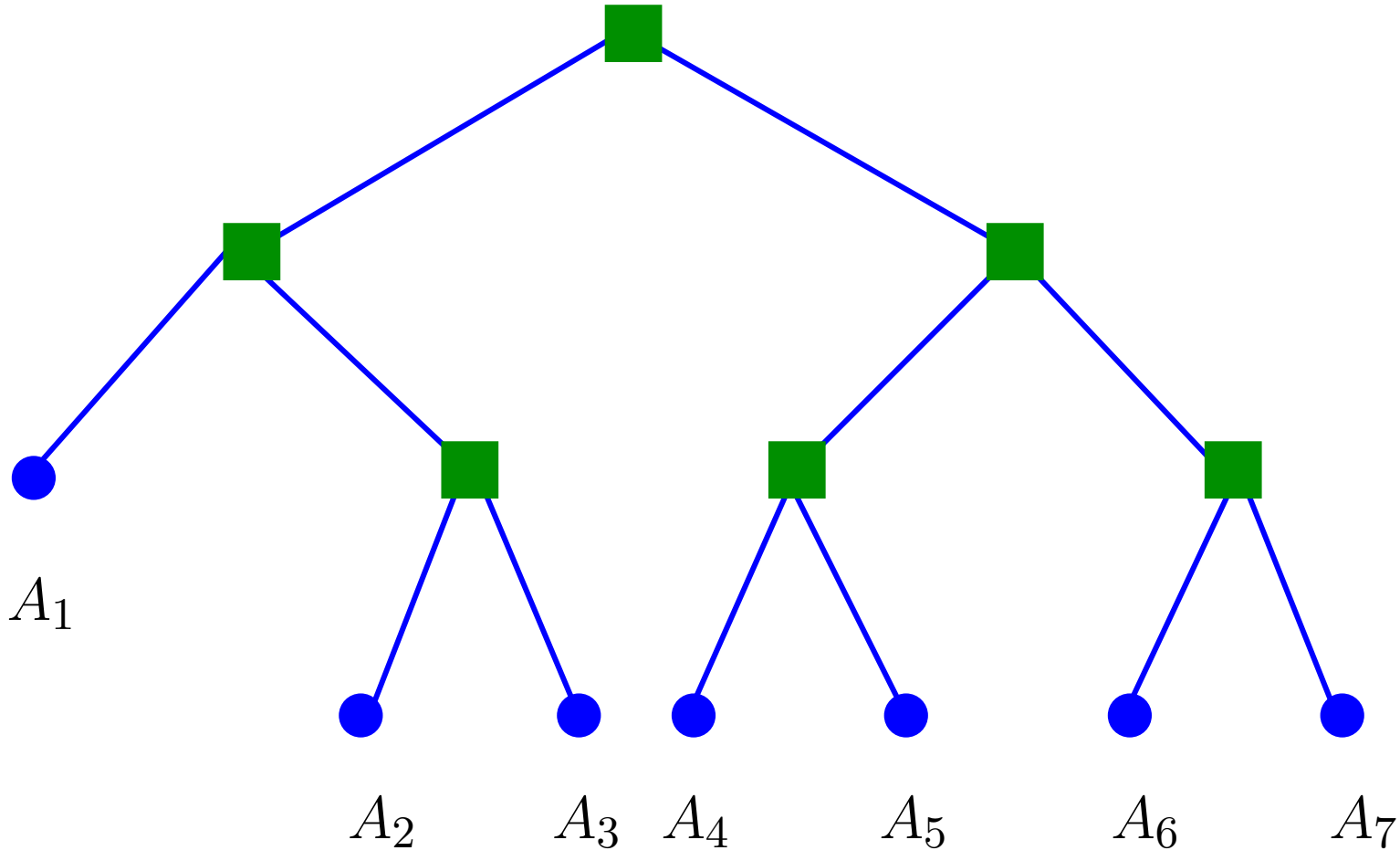
Se colocarmos em um polígono P o maior número possível de diagonais que duas-a-duas não se cruzam obteremos uma triangulação de P.



Triângulações e árvores binárias



Mult. Matriz e árvores binárias

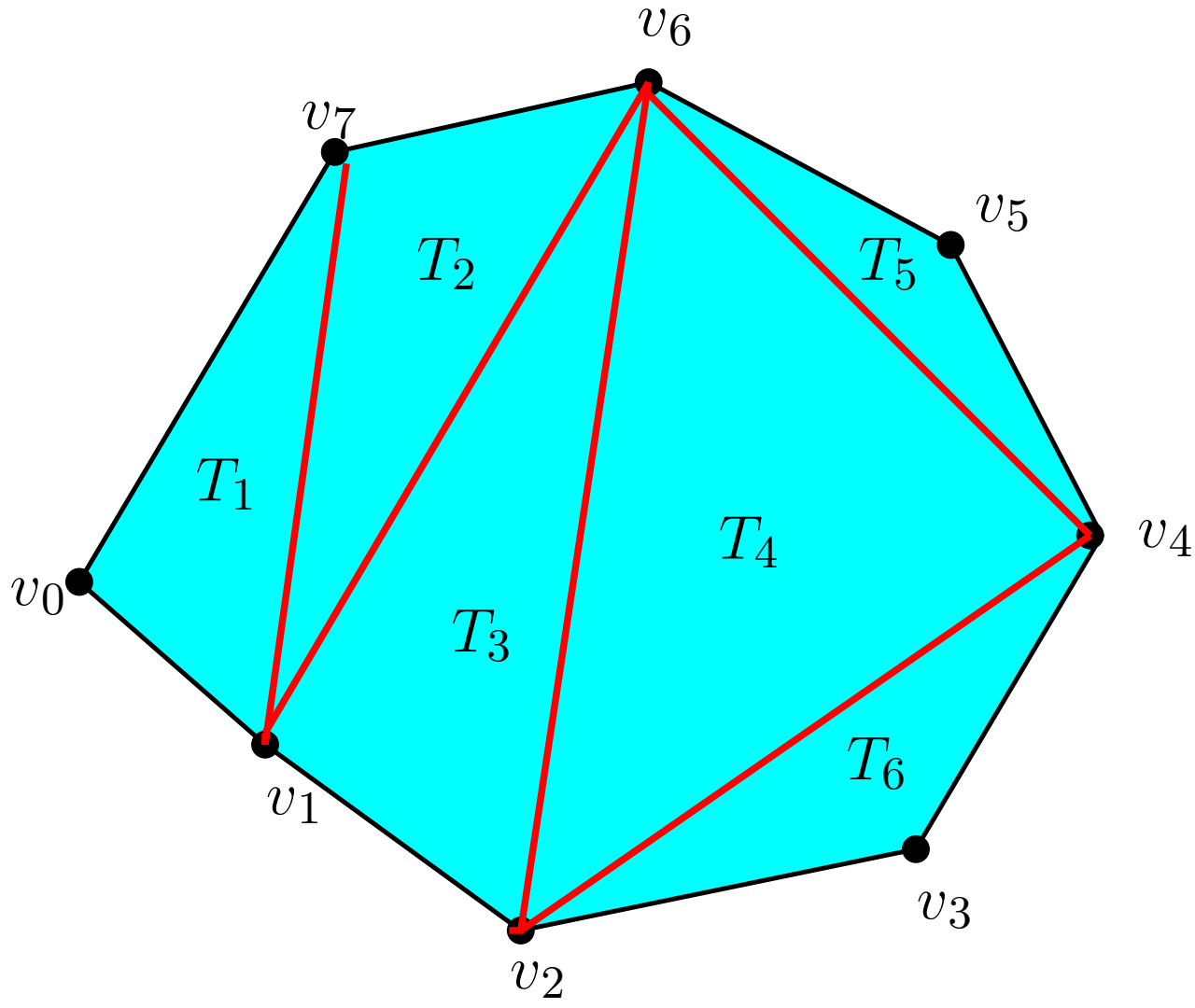


$$(A_1\,(A_2A_3))\,((A_4A_5)\,(A_6A_7))$$

Custo de uma triangulação

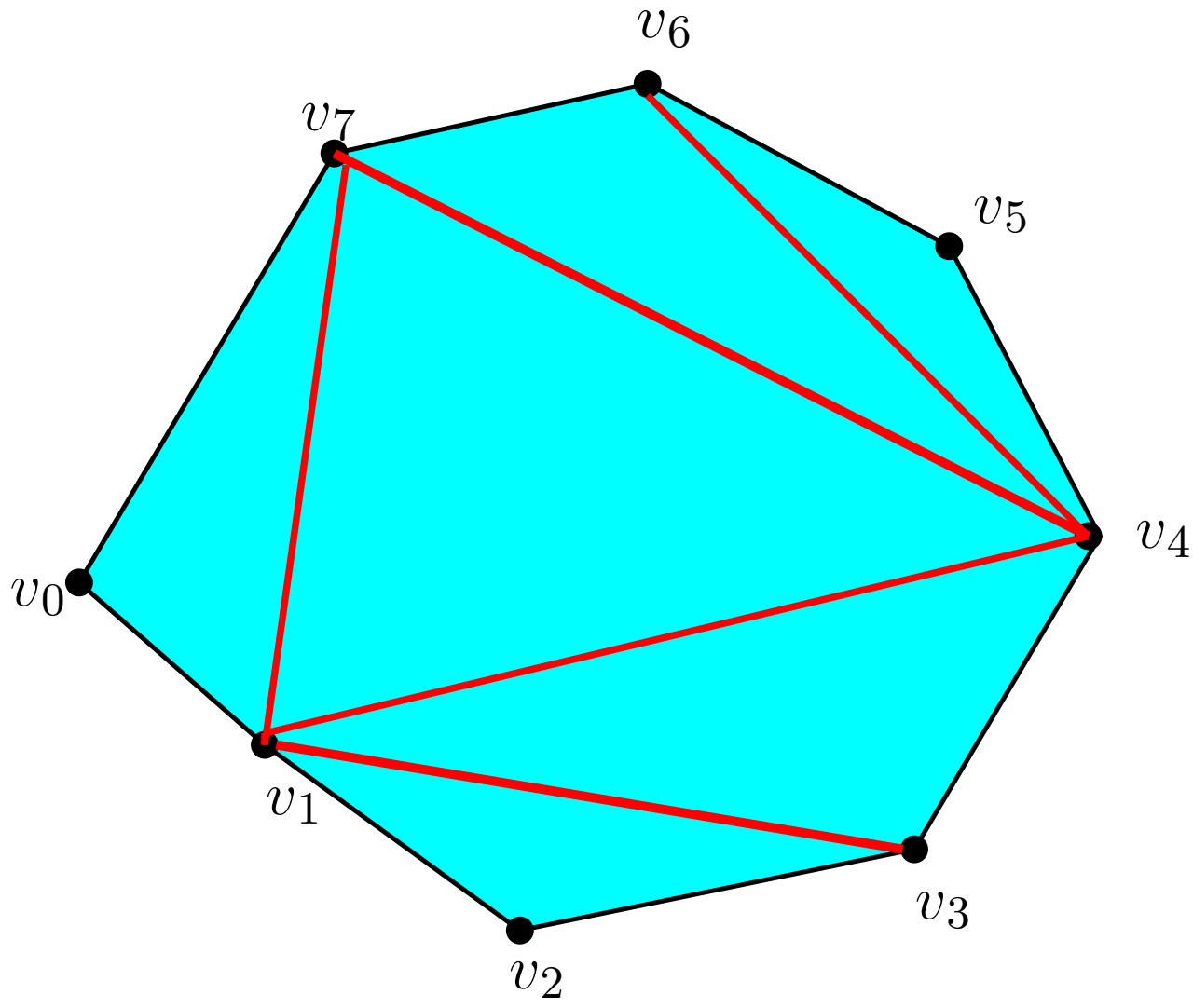
$c =$ função associa um custo $c(i, k, j)$ ao triângulo $\langle v_i, v_k, v_j \rangle$

custo de uma triangulação $=$ soma dos custos do triângulos



Triangulação ótima

Problema: Dado um polígono convexo $P = \langle v_0, v_1, \ldots, v_{n-1}, v_n \rangle$ e uma função custo c encontrar uma triangulação de P de custo mínimo.



Soluções ótimas contêm soluções ótimas

Se uma triangulação $\mathcal{T}$ que contém o triângulo

$$\langle v_0, v_k, v_n \rangle$$

é de custo mínimo então a restrição de $\mathcal{T}$ aos polígonos

$$\langle v_0, v_1, \ldots, v_k \rangle \quad \text{e} \quad \langle v_k, v_{k+1}, \ldots, v_n \rangle$$

também tem custo mínimo.

Decomposição: $\langle v_0, v_1, \ldots, v_k \rangle \langle v_k, v_{k+1}, \ldots, v_n \rangle$

Decomposição sugere um algoritmo recursivo.

Algoritmo recursivo

Recebe $P = \langle v_{i-1}, v_i, \ldots, v_j \rangle$ e função custo c e devolve o custo mínimo de uma triangulação de P.

```
REC-TRIANG-ÓTIMA (c, i, j)
 1  se i = j
 2        então devolva 0
 3  t ← ∞
 4  para k ← i até j − 1 faça
 5        q1 ← REC-TRIANG-ÓTIMA (c, i, k)
 6        q2 ← REC-TRIANG-ÓTIMA (c, k + 1, j)
 7        q ← q1 + c(i − 1, k, j) + q2
 8        se q < t
 9              então t ← q
10  devolva t
```

Consumo de tempo?

Consumo de tempo

O consumo de tempo do algoritmo REC-TRIANG-ÓTIMA é $\Omega(3^n)$.

Veja a análise do algoritmo REC-MAT-CHAIN.

Programação dinâmica

$t[i, j] =$ custo mínimo de uma triangulação de $\langle v_{i-1}, v_i, \ldots, v_j \rangle$

se $i = j$ então $t[i, j] = 0$

se $i < j$ então

$$t[i, j] = \min_{i \leq k < j} \{ t[i, k] + c(i-1, k, j) + t[k+1, j] \}$$

Exemplo:

$$t[3, 7] = \min_{3 \leq k < 7} \{ t[3, k] + c(2, k, 7) + t[k+1, 7] \}$$

Programação dinâmica

Cada subproblema

$$\langle v_{i-1}, \ldots, v_j \rangle$$

é resolvido uma só vez.

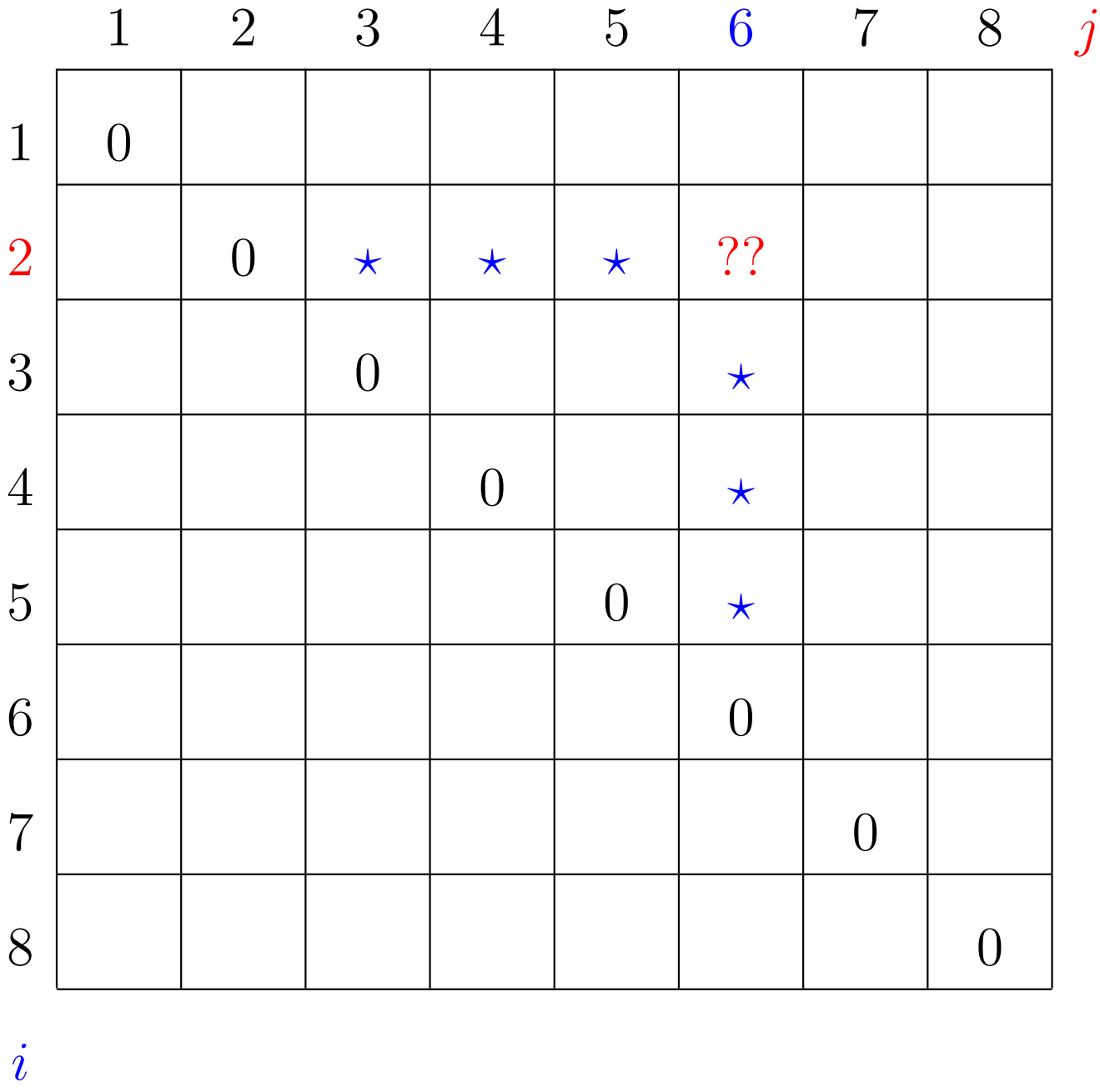
Em que ordem calcular os componentes da tabela m?

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

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

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

Programação dinâmica



Algoritmo de programação dinâmica

Recebe função custo c e devolve $t[1, n]$.

```
TRIANG-ÓTIMA (c, n)
 1  para i ← 1 até n faça
 2      t[i, i] ← 0
 3  para l ← 2 até n faça
 4      para i ← 1 até n − l + 1 faça
 5          j ← i + l − 1
 6          t[i, j] ← ∞
 7          para k ← i até j − 1 faça
 8              q ← t[i, k] + c(i − 1, k, j) + t[k+1, j]
 9              se q < t[i, j]
10                  então t[i, j] ← q
11  devolva t[1, n]
```

Correção e consumo de tempo

O consumo de tempo do algoritmo TRIANG-ÓTIMA é $\Theta(n^3)$.

Veja a análise do algoritmo MATRIX-CHAIN-ORDER.

Versão recursiva eficiente

```
MEMOIZED-TRIANG-ÓTIMA (p, n)
1   para i ← 1 até n faça
2       para j ← 1 até n faça
3           t[i, j] ← ∞
4   devolva LOOKUP-TRIANG (c, 1, n)
```

Versão recursiva eficiente

```
LOOKUP-TRIANG (c, i, j)
 1  se c[i, j] < ∞
 2        então devolva c[i, j]
 3  se i = j
 4        então t[i, j] ← 0
 5        senão para k ← i até j − 1 faça
 6                    q1 ← LOOKUP-TRIANG (c, i, k)
 7                    q2 ← LOOKUP-TRIANG (c, k+1, j)
 8                    q ← q1 + c(i−1, k, j) + q2
 9                    se q < t[i, j]
10                          então t[i, j] ← q
11  devolva t[1, n]
```

Exercícios

Exercício 20.G [CLR 16.4-1]
Prove que toda triangulação de um polígono convexo com n vértices possui $n-3$ diagonais e $n-2$ triângulos.

Exercício 20.H [CLR 16.4-2]
Professor Guinevere sugere que existe um algoritmo mais eficiente para resolver o problema da triângulação ótima quando custo de um triângulo é a sua área. O professor tem razão?

Exercício 20.I [CLR 16.4-3]
Suponha que um custo seja associado a cada diagonal. e que o custo de uma triangulação seja a soma dos custos de suas diagonais. Mostre que o problema da triangulação ótima com custos nas diagonais é um caso particular do problema da triangulação ótima com custos associados aos triângulos.

Exercício 20.J [CLR 16.4-4]
Encontre uma triângulação ótima de um poligono regular com 8 lados. Use para isto a função custo que associa a cada triângulo é o valor do seu perímetro.