Projeto de algoritmos

Profa Mariana Kolberg

(material adaptado dos slides do prof. Edson)

Programação Dinâmica

Multiplicação de cadeias de matrizes

- Dado o problema $M := M_1 \times M_2 \times \ldots \times M_n$

  Considere o subproblema (ou subseqüência)

$$
{}_iM_j = \begin{array}{ccccccc} M_i & \text{x} & M_{i+1} & \text{x} & \ldots & \text{x} & M_j \\ b_{i-1}\text{x}b_i & & b_i\text{x}b_{i+1} & & \ldots & & b_{j-1}\text{x}b_j \end{array}
$$

  Com $1 \leq i < j \leq n$ e custo mínimo dado por ${}_im_j$.

  Considere ${}_im_i = 0$, para i=1,…, n

- O objetivo é saber a ordem da multiplicação de custo mínimo
  - O tempo de encontrar esta ordem é compensado quando for se fazer as multiplicações.

Programação Dinâmica
Multiplicação de cadeias de matrizes

$$ {}_2M_3 = \underset{b_1 \text{ x } b_2}{M_2} \text{ x } \underset{b_2 \text{ x } b_3}{M_3} $$

$$ \underset{3 \text{ x } 5}{M_2} \text{ x } \underset{5 \text{ x } 40}{M_3} $$

$$ \Downarrow $$

$$ {}_2M_3 $$

$$ \underset{b_1}{3} \text{ x } \underset{b_3}{40} $$

$$ \underset{10 \text{ x } 3}{M_1} \text{ x } \underset{3 \text{ x } 5}{M_2} \text{ x } \underset{5 \text{ x } 40}{M_3} $$

vetor B

	10	3	5	40
posição	0	1	2	3

A matriz ${}_2M_3$ é uma matriz 3 x 40, ou seja, b_1 x b_3

Portanto, uma matriz ${}_iM_j$ é uma matriz b_{i-1} x b_j

$$ {}_iM_j = \underset{b_{i-1} \text{x} b_i}{M_i} \quad \text{x} \quad \underset{b_i \text{x} b_{i+1}}{M_{i+1}} \quad \text{x} \quad \underset{\ldots}{\ldots} \quad \text{x} \quad \underset{b_{j-1} \text{x} b_j}{M_j} $$

Programação Dinâmica
Multiplicação de cadeias de matrizes

O cálculo de ${}_iM_j$ com custo mínimo ${}_im_j$ pode ser decomposto em dois subproblemas. Considere **$i \leq k<j$,** logo

$$ {}_iM_k = M_i \text{ x } M_{i+1} \text{ x } \ldots \text{ x } M_k \qquad {}_{(k+1)}M_j = M_{k+1} \text{ x } M_{k+2} \text{ x } \ldots \text{ x } M_j $$

Onde

${}_iM_k$ tem custo mínimo ${}_im_k$ e dimensões b_{i-1} x b_k

${}_{(k+1)}M_j$ tem custo mínimo ${}_{(k+1)}m_j$ e dimensões b_k x b_j

O custo associado ao cálculo de ${}_iM_k$ x ${}_{(k+1)}M_j$, é dado por

$$ ({}_im_k + {}_{(k+1)}m_j) + (b_{i-1} \text{ x } b_k \text{x } b_j). $$

O custo mínimo é dado por

$$ {}_im_j = \min_{i \leq k < j} \{ ({}_im_k + {}_{(k+1)}m_j) + (b_{i-1} \text{ x } b_k \text{ x } b_j) \} $$

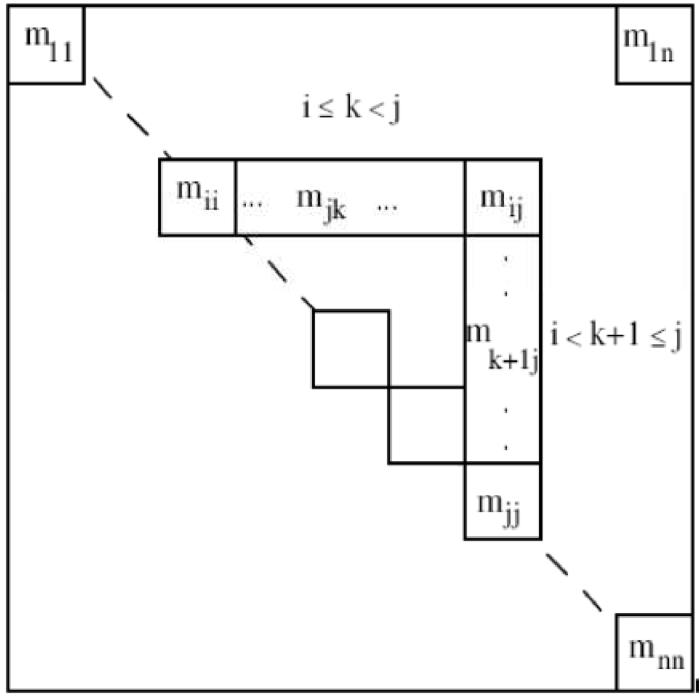
Programação Dinâmica
Multiplicação de cadeias de matrizes

O produto de n matrizes

- A diagonal é inicializada, ${}_i m_i = 0$

- Os valores para as diagonais superiores são calculados;

- Após o processo, o resultado final encontra-se no canto superior direito da matriz



Programação Dinâmica

Multiplicação de cadeias de matrizes

O algoritmo Multi_Mat

✶Recebe como entrada um vetor B : vetor [0..n] de IN , com as dimensões das n matrizes

✶ Fornece como saída um natural (m[1, n]) , correspondente ao número mínimo de multiplicações para o produto.

✶ Usa como área auxiliar de armazenamento uma matriz m : M, sendo

M := matriz [1..n, 1..n] de IN.

É Assumido que o tamanho da entrada é o número n de matrizes a serem multiplicadas.

As operações aritméticas sobre os naturais são consideradas operações fundamentais.

Programação Dinâmica
Multiplicação de cadeias de matrizes

Função: Multi_Mat(b : D) $\rightarrow$ IN {custo mínimo de produto de matrizes}

1. para i de 1 até n faça m[i , i] $\leftarrow$ 0 ; {inicializa diagonal principal}
2. para u de 1 até n - 1 faça {deslocamento da diagonal: 7}
3. para i de 1 até n - u faça {posição na diagonal: 6}
4. j $\leftarrow$ i + u ; {u = j - i} ;
5. m[i , j] $\leftarrow \min_{i \leq k < j}$ {(m[i , k] + m[k + 1 , j]) + (b[i - 1] x b[k] x b[j])} ;
6. fim-para {3: i de 1 até n - u}
7. fim-para {2: u de 1 até n - 1}
8. retorne-saída(m[1 , n]) ; {dá saída extraída}
9. fim-Função {fim do algoritmo Multi_Mat: custo mínimo de produto}

Programação Dinâmica
Multiplicação de cadeias de matrizes

<u>*Função*</u>: Multi_Mat(b : D) $\rightarrow$ IN

1. <u>para</u> i <u>de</u> 1 <u>até</u> n <u>faça</u> m[i , i] $\leftarrow$ 0 ;
2. <u>para</u> u <u>de</u> 1 <u>até</u> n - 1 <u>faça</u>
3. <u>para</u> i <u>de</u> 1 <u>até</u> n - u <u>faça</u>
4. j $\leftarrow$ i + u ; {u = j - i} ;
5. $m[i,j] \leftarrow \min_{i \leq k < j} \{(m[i,k] + m[k+1,j]) + (b[i-1] \times b[k] \times b[j])\}$;
6. <u>fim-para</u>
7. <u>fim-para</u>
8. <u>retorne-saída</u>(m[1 , n]) ;
9. <u>fim-Função</u>

A inicialização

1. <u>para</u> i <u>de</u> 1 <u>até</u> n <u>faça</u> m[i , i] $\leftarrow$ 0 ;

Inicializa a diagonal principal

A operação m[i , i] $\leftarrow$ 0 é executada n vezes

Logo,

desemp[Inicialização] = n . 1 = n;

$$\Downarrow$$

$$c_p^{\leq}[\text{Inicialização}] = O(n)$$

Programação Dinâmica
Multiplicação de cadeias de matrizes

A iteração nas linhas de 2 a 7 executa n - 1 vezes as linhas de 3 a 6

2. <u>para</u> u <u>de</u> 1 <u>até</u> n - 1 <u>faça</u> {deslocamento da diagonal: 7}

Executa n-1 vezes

3. <u>para</u> i <u>de</u> 1 <u>até</u> n - u <u>faça</u> {posição na diagonal: 6}
4. $j \leftarrow i + u$; $\{u = j - i\}$;
5. $m[i,j] \leftarrow \min_{i \le k < j} \{(m[i,k] + m[k+1,j]) + (b[i-1] \times b[k] \times b[j])\}$;
6. <u>fim-para</u> {3: i de 1 até n - u}

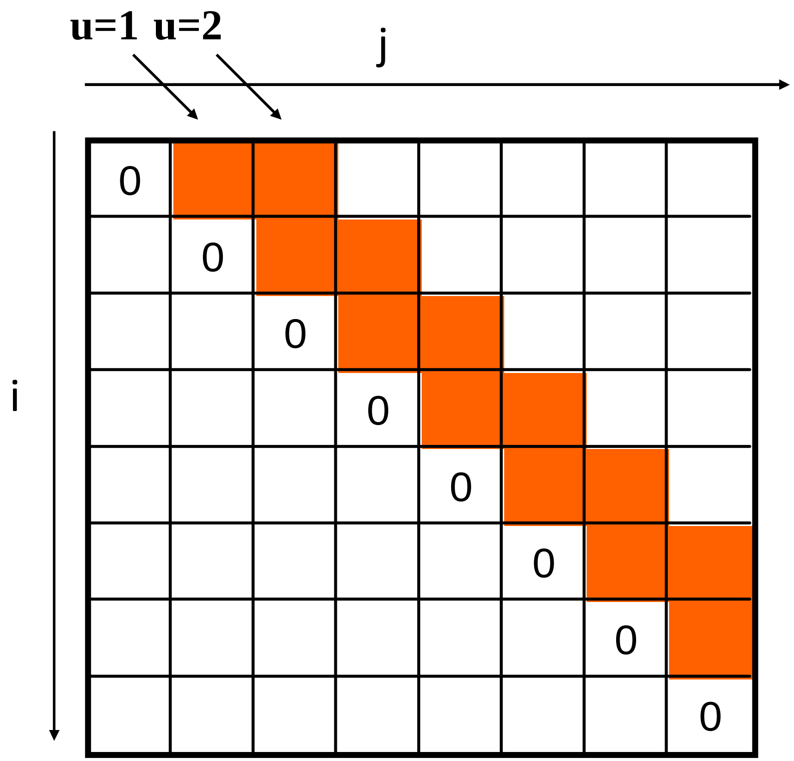
7. <u>fim-para</u> {2: u de 1 até n - 1}

Considere m_u os elementos da matriz acima da diagonal principal que são atualizados a cada iteração da linha 2.

u	0	1	...	u	...	n-1
$\dim(m_u)$	n	n-1	...	n-u	...	1

No início da u-ésima iteração $\dim(m_u) = n - u$.

Programação Dinâmica
Multiplicação de cadeias de matrizes



Programação Dinâmica
Multiplicação de cadeias de matrizes

Função: Multi_Mat(b : D) → IN

1. para i de 1 até n faça m[i, i] ← 0 ;
2. para u de 1 até n - 1 faça
3. para i de 1 até n - u faça
4. $j \leftarrow i + u$; $\{u = j - i\}$;
5. $m[i,j] \leftarrow \min_{i \le k < j} \{(m[i,k] + m[k+1,j]) + (b[i-1] \times b[k] \times b[j])\}$;
6. fim-para
7. fim-para
8. retorne-saída(m[1, n]) ;
9. fim-Função

A maior contribuição do corpo do laço é dada pela linha 5.

5. $m[i,j] \leftarrow \min_{i \le k < j} \{(m[i,k] + m[k+1,j]) + (b[i-1] \times b[k] \times b[j])\}$

Quantos valores intermediários têm que ser verificados para determinar o valor a ser armazenado em M[i,j] ?

$(j – 1) - i + 1 = j - i = u$

(de acordo com a linha 4)

4. $j \leftarrow i + u$; $\{u = j - i\}$;

A cada valor de k realizamos 4 operações aritméticas, logo

Desemp[Linha 5] $(m_u) = 4.u$

Na u-ésima iteração temos,

Comando	desempenho
4. $j \leftarrow i + u$;	1
5. $m[i,j] \leftarrow$ melhor valor;	4.u

Programação Dinâmica
Multiplicação de cadeias de matrizes

Função: Multi_Mat(b : D) → IN

1. para i de 1 até n faça m[i, i] ← 0 ;
2. para u de 1 até n - 1 faça
3. para i de 1 até n - u faça
4. $j \leftarrow i + u$; $\{u = j - i\}$;
5. $m[i, j] \leftarrow \min\limits_{i \le k < j} \{(m[i, k] + m[k+1, j]) + (b[i-1] \times b[k] \times b[j])\}$;
6. fim-para
7. fim-para
8. retorne-saída(m[1, n]) ;
9. fim-Função

Logo, o desempenho das linhas de 3 a 6

3. para i de 1 até n - u faça
4. $j \leftarrow i + u$; $\{u = j - i\}$;
5. $m[i, j] \leftarrow \min\limits_{i \le k < j} \{(m[i, k] + m[k+1, j]) + (b[i-1] \times b[k] \times b[j])\}$;
6. fim-para

na u-ésima iteração é igual a

$$desemp[\text{Crp-Multi-Mat}](m_u) = \sum_{i=1}^{n-u} (1 + 4u) = (1 + 4u).(n - u)$$

Como u varia de 1 a n-1, o desempenho da iteração é igual a

$$desemp[\text{Iteração}] = \sum_{u=1}^{n-1} ((1 + 4u).(n - u))$$

Programação Dinâmica

Multiplicação de cadeias de matrizes

$$S = \sum_{u=1}^{n-1} [(1+4u)(n-u)]$$

$$= \sum_{u=1}^{n-1} (n + 4un - u - 4u^2)]$$

$$= \sum_{u=1}^{n-1} (n + 4un) - \sum_{u=1}^{n-1} (u + 4u^2)$$

$$= n(n-1) + \sum_{u=1}^{n-1} (4un) - \sum_{u=1}^{n-1} u - \sum_{u=1}^{n-1} 4u^2$$

$$= n(n-1) + 4n \sum_{u=1}^{n-1} (u) - \frac{(n-1)n}{2} - 4 \sum_{u=1}^{n-1} u^2$$

$$= n(n-1) + 4n \frac{(n-1)n}{2} - \frac{(n-1)n}{2} - 4 \sum_{u=1}^{n-1} u^2$$

Programação Dinâmica

Multiplicação de cadeias de matrizes

$$S = 4n\frac{(n-1)n}{2} + \frac{(n-1)n}{2} - 4\sum_{u=1}^{n-1} u^2$$

$$S = 4n\frac{(n-1)n}{2} + \frac{(n-1)n}{2} - 4\frac{n(n-1)(2n-1)}{6}$$

Programação Dinâmica

Multiplicação de cadeias de matrizes

$$S = \frac{12n(n-1)n + 3n(n-1) - 4n(n-1)(2n-1)}{6}$$

$$S = \frac{(n-1)n}{6}(12n + 3 - 4(2n-1))$$

$$S = \frac{(n-1)n}{6}(12n + 3 - 8n + 4)$$

$$S = \frac{(n-1)n}{6}(4n+7) = \frac{(n^2-n)(4n+7)}{6}$$

$$S = \frac{4n^3 + 7n^2 - 4n^2 - 7n}{6} = \frac{4n^3 + 3n^2 - 7n}{6}$$

$$desemp[\text{Iteração}] = \frac{4n^3 + 3n^2 - 7n}{6}$$

$$c_p^{\leq}[\text{Iteração}] = O(n^3)$$

Programação Dinâmica
Multiplicação de cadeias de matrizes

A ordem da complexidade pessimista do algoritmo de Multicação de Matrizes é dada pela soma das três contribuições : inicialização, iteração e finalização.

$$c_p^{\leq}[\text{Inicialização}] = O(n)$$

$$+$$

$$c_p^{\leq}[\text{Iteração}] = O(n^3)$$

$$+$$

$$c_p^{\leq}[\text{Finalização}] = O(1)$$

$$c_p^{\leq}[\text{Algoritmo}] = O(n^3)$$

Programação Dinâmica

Caminhos de custo mínimo em grafo orientado

O problema de caminhos de custo mínimo em grafo orientado.

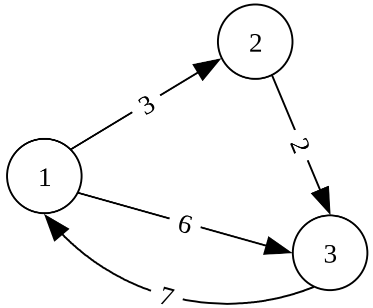
O problema consiste em determinar caminhos de custo mínimo entre cada par de vértices de um grafo.

A programação dinâmica é aplicada sobre um conjunto I de vértices intermediários, o qual é incrementado a cada passo.

A idéia é determinar os caminhos de custo mínimo com a restrição de usar como vértices intermediários apenas os vértices do conjunto I.

Programação Dinâmica
Caminhos de custo mínimo em grafo orientado

Considere um grafo orientado G e a sua matriz de custos:

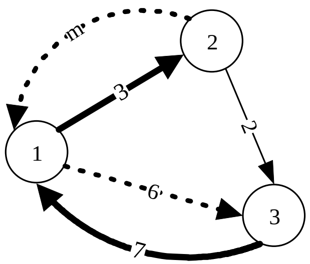


custo	v_1	v_2	v_3
v_1	0	3	6
v_2	m	0	2
v_3	7	m	0

Considere como vértice intermediário o vértice v_1 e m um valor inteiro grande.

O caminho de v_3 a v_2 passando por v_1 tem comprimento 7+3=10 < m.

O caminho de v_2 a v_3 passando por v_1 tem comprimento m+6 >2.



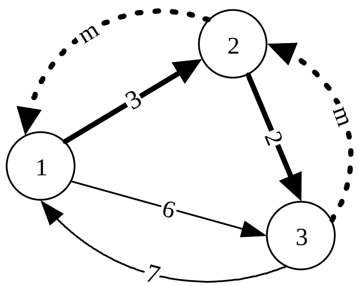
dist	v_1	v_2	v_3
v_1	0	3	6
v_2	m	0	2
v_3	7	**10**	0

Programação Dinâmica
Caminhos de custo mínimo em grafo orientado

Considere como vértice intermediário o vértice v_2.

O caminho de v_1 a v_3 passando por v_2 com comprimento $3+2 = 5 < 6$

O caminho de v_3 a v_1 passando por v_2 tem comprimento $10+m>7$

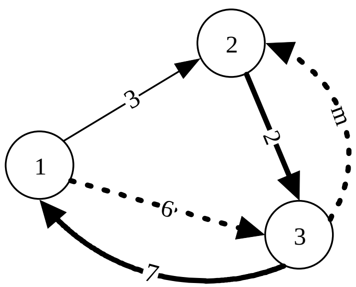


dist	v_1	v_2	v_3
v_1	0	3	**5**
v_2	m	0	2
v_3	7	10	0

Considere como vértice intermediário o vértice v_3.

O caminho de v_2 a v_1 passando por v_3 com comprimento $2+7 = 9 < m$

O caminho de v_1 a v_2 passando por v_3 tem comprimento $5+10>3$



dist	v_1	v_2	v_3
v_1	0	3	5
v_2	**9**	0	2
v_3	7	10	0

Programação Dinâmica
Caminhos de custo mínimo em grafo orientado

- Subestrutura ótima
  - Subcaminhos de caminhos mais curtos são caminhos mais curtos
- Uso de vertices intermediários:
  - Suponha o conjunto de vertices intermediários possíveis: {1,2,..,k}
  - Se k não é um vertice intermediário do caminho p, todos os possiveis vertices intermediários estão em {1,2,..,k-1}, logo o caminho mais curto vi-vj com vértices intermediátios em {1,2,..,k-1} é tb o caminho mais curto de {1,2,..,k}.
  - Se k é vertice intermediário, então temos 2 caminhos: p1 de i a k e p2 de k a j. cada um deles tem vertices intermediários possíveis em {1,2,..,k-1}.
- Solução recursiva

$$d_{ij}^{k} = \begin{cases} w_{ij} \; se \; k = 0 \\ \min\,(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1})se \; k \geq 1 \end{cases}$$

- Cálculo da solução bottom-up
  - Idéia é usar os caminhos de l=2 para achar caminhos de l=3 e assim por diante
  - Uso de uma matriz predecessora para saber o caminho mais curto entre um par de vértices

Programação Dinâmica

Caminhos de custo mínimo em grafo orientado

Algoritmo: Custo mínimo de caminhos entre vértices de grafo orientado
Função Dist_vrt(d:D)→R {distância entre vértices de grafo}

{ *Entrada*: grafo orientado G com conjunto V de vértices e matriz cst:M (custos)
Saída: matriz dst:R (custos dos melhores caminhos entre vértices de G) }

Inicialização

```
1. I←∅;                                        {inicializa conjunto I sem intermediários}
2. para cada u∈ V faça                         {inicializa distância sob I: 7}
3.   dst[u,u]←0;                               {inicializa diagonal com zero}
4.   para cada v∈ (V−{u}) faça                 {inicializa não diagonal: 7}
5.       dst[u,v]←cst[u,v];                    {distância inicial como custo}
6.   fim-para                                  {4: cada v∈ (V−{u})}
7. fim-para                                    {2: cada u∈ V}
```

Iteração

```
8. enqto I≠V faça                              {itera: 16}
9.   escolha w∈ (V−I);                         {seleciona novo vértice}
10.  I←I∪{w};                                  {atualiza I com novo intermediário}
11.  para cada u∈ V faça                       {atualiza distância sob I: 15}
12.      para cada v∈ (V−{u}) faça             {atualiza não diagonal: 14}
13.          dst[u,v]←mín(dst[u,v],dst[u,w]+dst[w,v]);
14.      fim-para                              {12: cada v∈ (V−{u})}
15.  fim-para                                  {11: cada u∈ V}
16. fim-enqto                                  {8: I≠V (todos os vértices)}
```

Finalização

```
17. retorne-saída(dst);                        {dá como saída resposta pronta}
18. fim-Função                                 {fim do algoritmo Dist_vrt: distância entre vértices}
```

Programação Dinâmica
Caminhos de custo mínimo em grafo orientado

O algoritmo

- recebe como entrada um grafo orientado G com conjunto V de vértices e matriz de custos
- fornece como saída uma matriz com os custos dos melhores caminhos entre os vértices de G)
- O tamanho da entrada corresponde ao número n de vértices de G.
- As operações fundamentais são as manipulações com os vértices e as matrizes.

Programação Dinâmica

Caminhos de custo mínimo em grafo orientado

Algoritmo: Custo mínimo de caminhos entre vértices de grafo orientado
Função Dist_vrt$(d:D) \rightarrow R$ {distância entre vértices de grafo}

{*Entrada*: grafo orientado G com conjunto V de vértices e matriz cst:M (custos)
Saída: matriz dst:R (custos dos melhores caminhos entre vértices de G)}

Inicialização

```
1. I ← ∅;                                  {inicializa conjunto I sem intermediários}
2. para cada u ∈ V faça                    {inicializa distância sob I: 7}
3.   dst[u,u] ← 0;                         {inicializa diagonal com zero}
4.   para cada v ∈ (V – {u}) faça          {inicializa não diagonal: 7}
5.       dst[u,v] ← cst[u,v];              {distância inicial como custo}
6.   fim-para                              {4: cada v ∈ (V – {u})}
7. fim-para                                {2: cada u ∈ V}
```

Iteração

```
8. enqto I ≠ V faça                        {itera: 16}
9.   escolha w ∈ (V – I);                  {seleciona novo vértice}
10.  I ← I ∪ {w};                          {atualiza I com novo intermediário}
11.  para cada u ∈ V faça                  {atualiza distância sob I: 15}
12.      para cada v ∈ (V – {u}) faça      {atualiza não diagonal: 14}
13.          dst[u,v] ← mín(dst[u,v], dst[u,w] + dst[w,v]);
14.      fim-para                          {12: cada v ∈ (V – {u})}
15.  fim-para                              {11: cada u ∈ V}
16. fim-enqto                              {8: I ≠ V (todos os vértices)}
```

Finalização

```
17. retorne-saída(dst);                    {dá como saída resposta pronta}
18. fim-Função                             {fim do algoritmo Dist_vrt: distância entre vértices}
```

Programação Dinâmica

Caminhos de custo mínimo em grafo orientado

Algoritmo

```
I ← ∅ ;
para cada u ∈ V faça
   dst[u, u] ← 0 ;
   para cada v ∈ (V − {u}) faça
        dst[u, v] ← cst[u, v] ;
   fim-para
fim-para
enqto I ≠ V faça
   escolha w ∈ (V − I) ;
   I ← I ∪ {w} ;
   para cada u ∈ V faça
        para cada v ∈ (V − {u}) faça
             dst[u, v] ← mín(dst[u, v], dst[u, w] + dst[w, v]) ;
        fim-para
   fim-para
fim-enqto
retorne-saída(dst) ;
fim-Função
```

A inicialização

comando	nº de vezes	esforço
1. I ← ∅ ;	1	1
2. para cada u ∈ V faça	n	
3. dst[u, u] ← 0 ;	1	1
4. para cada v ∈ (V − {u}) faça	(n - 1)	
5. dst[u, v] ← cst[u, v] ;	1	1
6. fim-para		
7. fim-para		

$$desemp[Incz - Dist - Vrt](G) = 1 + \sum_{k=1}^{n}(1 + \sum_{j=1}^{n-1} 1)$$

$$desemp[Incz - Dist - Vrt](G) = 1 + \sum_{k=1}^{n}(1 + (n - 1))$$

$$desemp[Incz - Dist - Vrt](G) = 1 + \sum_{k=1}^{n}(n)$$

Logo,

$$desemp[\text{Incz\_Dist\_vrt}](G) = n^2 + 1$$

Programação Dinâmica

Caminhos de custo mínimo em grafo orientado

Algoritmo

```
I ← ∅;
para cada u ∈ V faça
  dst[u, u] ← 0;
  para cada v ∈ (V − {u}) faça
      dst[u, v] ← cst[u, v] ;
  fim-para
fim-para
enqto I ≠ V faça
  escolha w ∈ (V − I);
  I ← I ∪ {w};
  para cada u ∈ V faça
      para cada v ∈ (V − {u}) faça
          dst[u, v] ← mín(dst[u, v], dst[u, w] + dst[w, v]) ;
      fim-para
  fim-para
fim-enqto
retorne-saída(dst);
fim-Função
```

A **Iteração**

```
enqto I ≠ V faça
  escolha w ∈ (V − I);
  I ← I ∪ {w};
  para cada u ∈ V faça
      para cada v ∈ (V − {u}) faça
          dst[u, v] ← mín(dst[u, v], dst[u, w] + dst[w, v]) ;
      fim-para
  fim-para
fim-enqto
```

Quantos elementos tenho em I no início da primeira iteração? E no fim?

A quantidade de elementos do Conjunto I varia com a iteração. Portanto no final da i-ésima iteração temos $| I_i | = i$

Programação Dinâmica
Caminhos de custo mínimo em grafo orientado

Algoritmo

```
1. I ← ∅ ;
2. para cada u ∈ V faça
3.    dst[u, u] ← 0 ;
4.    para cada v ∈ (V − {u}) faça
5.          dst[u, v] ← cst[u, v] ;
6.    fim-para
7. fim-para
8. enqto I ≠ V faça
9.    escolha w ∈ (V − I) ;
10.   I ← I ∪ {w} ;
11.   para cada u ∈ V faça
12.         para cada v ∈ (V − {u}) faça
13.               dst[u, v] ← mín( dst[u, v], dst[u, w] + dst[w, v]) ;
14.         fim-para
15.   fim-para
16. fim-enqto
17. retorne-saída( dst ) ;
18. fim-Função
```

Olhando as linhas 9 e 10.

```
9.   escolha w ∈ (V − I) ;
10.  I ← I ∪ {w} ;
```

Desemp[linha 9] = n-i+1

Desemp[linha 10] =1

Na i-ésima iteração,

Desemp[linha 9-linha 10] = (n-i+1)+1

Note que o enquanto das linhas 8-16 é executado **n** vezes, ou seja , o **i** varia de 1 até n

Olhando as linhas de 11 a 15.

```
11. para cada u ∈ V faça
12.       para cada v ∈ (V − {u}) faça
13.             dst[u, v] ← mín( dst[u, v], dst[u, w] + dst[w, v]) ;
14.       fim-para
15. fim-para
```

Temos

$$\text{Desemp[linha11...linha15]} = \sum_{i=1}^{n}\sum_{j=1}^{n-1} 1 = \sum_{i=1}^{n}(n-1) = n(n-1)$$

Programação Dinâmica

Caminhos de custo mínimo em grafo orientado

Algoritmo

```
I ← ∅ ;
para cada u ∈ V faça
  dst[u, u] ← 0 ;
  para cada v ∈ (V − {u}) faça
      dst[u, v] ← cst[u, v] ;
  fim-para
fim-para
enqto I ≠ V faça
  escolha w ∈ (V − I) ;
  I ← I ∪ {w} ;
  para cada u ∈ V faça
      para cada v ∈ (V − {u}) faça
          dst[u, v] ← mín(dst[u, v], dst[u, w] + dst[w, v]) ;
      fim-para
  fim-para
fim-enqto
retorne-saída(dst) ;
fim-Função
```

A iteração é executada n vezes


```
enqto I ≠ V faça
  escolha w ∈ (V − I) ;
  I ← I ∪ {w} ;
  para cada u ∈ V faça
      para cada v ∈ (V − {u}) faça
          dst[u, v] ← mín(dst[u, v], dst[u, w] + dst[w, v]) ;
      fim-para
  fim-para
fim-enqto
```


Combinando os desempenhos obtidos anteriormente, temos

$$\text{Desemp[Iter-Dist-Vrt]} = \sum_{i=1}^{n} [(n - i + 1) + 1 + n(n-1)]$$

$$\text{Desemp[Iter-Dist-Vrt]} = \sum_{i=1}^{n} (n - i + 1) + \sum_{i=1}^{n} 1 + \sum_{i=1}^{n} n(n-1)$$

$$\text{Desemp[Iter-Dist-Vrt]} = \sum_{i=1}^{n} i + n + \sum_{i=1}^{n} n(n-1)$$

Programação Dinâmica

Caminhos de custo mínimo em grafo orientado

$$\text{Desemp[Iter-Dist-Vrt]} = \frac{(n+1)n}{2} + n + n^2(n-1)$$

$$\text{Desemp[Iter-Dist-Vrt]} = \frac{n^2+n}{2} + n + n^3 - n^2 = \frac{2n^3 - n^2 + 3n}{2}$$

Sabemos

Desemp[Dist-Vrt] = Desemp[Incz-Dist-Vrt]+Desemp[Iter-Dist-Vrt]+Desemp[Fnlz-Dist-Vrt]

$$c_p^{\leq}[\text{Dist-Vrt}](n) = \mathrm{O}((1+n^2) + (n^3 - \frac{n^2}{2} + \frac{3n}{2}) + 1)$$

$$c_p^{\leq}[\text{Dist-Vrt}](n) = O(n^3)$$

Programação Dinâmica
Seqüência comum mais longa

- O que é uma subseqüência?
  - Caracteres aparecem na mesma ordem
  - Não necessáriamente consecutivos
- O que é uma subseqüência comum mais longa?
  - É a maior subsequencia que aparece em ambas sequencias originais
- importância
  - Forma de medir a semelhança entre duas sequencias
    - Cadeias genéticas
    - Comparação entre arquivos

Programação Dinâmica
Seqüência comum mais longa

- Exemplo 1:
  - X= {A,B,K,D,J}
  - Y= {A,R,G,K,J}
  - Subsequencias comuns {A,K}, {A,J},{A,K,J}
  - Subsequencia comum mais longa {A,K,J}
- Exemplo 2:
  - X= {A,B,C,B,D,A,B}
  - Y= {B,D,C,A,B,A}
  - Subsequencia comum mais longa {B,C,A,B}
  - Subsequencia comum mais longa {B,C,B,A}

Programação Dinâmica
Seqüência comum mais longa

- A solução por força bruta seria
  - Enumerar todas as subseqüências de X
  - Conferir quais destas subseqüências são também subseqüências de Y
  - A medida que for encontrando estas subseqüências comuns, ir atualizando a mais longa
- Cada subsequência corresponde a um subconjunto de índices {1, 2, 3, ..., m}
  - Existem 2^m subseqüências de X
- Complexidade exponencial!

Programação Dinâmica
Seqüência comum mais longa

- Definição do problema

SCML

Instância Duas seqüencias $X = \langle x_1, x_2, \ldots, x_m \rangle$ e $Y = \langle y_1, y_2, \ldots, y_n \rangle$.
Solução Uma subseqüência comum Z de X, Y.
Objetivo Maximizar o comprimento de Z.

- Ao comparar x_1 e y_1 podemos ter caracteres:
  - Iguais – temos mais um elemento igual na nossa subsequencia, e podemos então analisar x_2 e y_2
  - Diferentes – não temos mais um elemento e podemos
    - Comparar x_1 e a sequencia de y_2 a y_n
    - Comparar y_1 e a sequencia de x_2 a x_n

Programação Dinâmica
Seqüência comum mais longa

- Idéia: prefixo
  - O i-ésimo prefixo de X é $X_i = \langle x_1, x_2, x_3, \ldots, x_i \rangle$
    - Dado $X = \langle A, B, C, B, D, A, B \rangle$
    - $X_3 = \langle A, B, C \rangle$
    - $X_5 = \langle A, B, C, B, D \rangle$

Teorema: Subestrutura Ótima de uma SCML
Sejam as seqüências $X = \langle x_1, x_2, \ldots, x_m \rangle$ e $Y = \langle y_1, y_2, \ldots, y_n \rangle$, e seja $Z = \langle z_1, z_2, \ldots, z_k \rangle$ qualquer SCML de X e Y

- Se $x_m = y_n$, então $z_k = x_m = y_n$ e Z_{k-1} é uma SCML de X_{m-1} e Y_{n-1}
- Se $x_m \neq y_n$, então $z_k \neq x_m$ implica que Z é uma SCML de X_{m-1} e Y
- Se $x_m \neq y_n$, então $z_k \neq y_m$ implica que Z é uma SCML de X e Y_{n-1}

Programação Dinâmica

Seqüência comum mais longa

$$
c[i,j] = \begin{cases} 0 & \text{se } i=0 \text{ ou } j=0, \\ c[i-1, j-1] + 1 & \text{se } i, j > 0 \text{ e } x_i = y_j, \\ \max(c[i, j-1], c[i-1, j]) & \text{se } i, j > 0 \text{ e } x_i \neq y_j. \end{cases}
$$

- A sequencia comum mais longa é zero quando uma das duas strings é vazia
- Quando os caracteres são iguais, somamos 1 a solução ótima da sequencia comum mais longa até então encontrada
- Quando os caracteres são diferentes, mantemos o valor da maior sequencia ótima encontrada:
  - Considerando o y em questão e apenas o caracter anteior ao x em questão
  - Considerando o x em questão e apenas o caracter anteior ao y em questão

Programação Dinâmica
Seqüência comum mais longa

- Cada elemento de c[i,j] representará o tamanho da maior subsequencia comum entre a subsequencia de X de índice 1 a i e da subsequencia de Y de indice 1 a j.
- Solução vai calculando por linha da matriz, da esquerda para direita, começando na posição 1,1

	-	A	B	C	B	D	A	B
-								
B								
D								
C								
A								
B								
A								

Programação Dinâmica
Seqüência comum mais longa

$$c[i,j] = \begin{cases} 0 & \text{se } i=0 \text{ ou } j=0, \\ c[i-1, j-1] + 1 & \text{se } i, j > 0 \text{ e } x_i = y_j, \\ \max(c[i, j-1], c[i-1, j]) & \text{se } i, j > 0 \text{ e } x_i \neq y_j. \end{cases}$$

	-	A	B	C	B	D	A	B
-								
B								
D								
C								
A								
B								
A								

Programação Dinâmica
Seqüência comum mais longa

	-	A	B	C	B	D	A	B
-	0	0	0	0	0	0	0	0
B	0							
D	0							
C	0							
A	0							
B	0							
A	0							

Programação Dinâmica
Seqüência comum mais longa

	-	A	B	C	B	D	A	B
-	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

Programação Dinâmica
Seqüência comum mais longa

SCML

Entrada Dois strings X e Y e seus respectivos tamanhos m e n medidos em número de caracteres.

Saída O tamanho da maior subseqüência comum entre X e Y.

```
m := comprimento(X)
n := comprimento(Y)
for i := 0 to m do c[i,0] := 0;
for j := 1 to n do c[0,j] := 0;
for i := 1 to m do
    for j := 1 to n do
        if x_i = y_j then
            c[i,j] := c[i-1,j-1] + 1
        else
            c[i,j] := max(c[i,j-1], c[i-1,j])
        end if
    end for
return c[m,n]
```