MC458 — Projeto e Análise de Algoritmos I

C.C. de Souza C.N. da Silva O. Lee

Antes de mais nada...

- Uma versão anterior deste conjunto de slides foi preparada por Cid Carvalho de Souza e Cândida Nunes da Silva para uma instância anterior desta disciplina.
- O que vocês tem em mãos é uma versão modificada preparada para atender a meus gostos.
- Nunca é demais enfatizar que o material é apenas um guia e não deve ser usado como única fonte de estudo. Para isso consultem a bibliografia (em especial o CLR ou CLRS).

Orlando Lee

Agradecimentos (Cid e Cândida)

- Várias pessoas contribuíram direta ou indiretamente com a preparação deste material.
- Algumas destas pessoas cederam gentilmente seus arquivos digitais enquanto outras cederam gentilmente o seu tempo fazendo correções e dando sugestões.
- Uma lista destes "colaboradores" (em ordem alfabética) é dada abaixo:
  - Célia Picinin de Mello
  - José Coelho de Pina
  - Orlando Lee
  - Paulo Feofiloff
  - Pedro Rezende
  - Ricardo Dahab
  - Zanoni Dias

O Problema da Mochila

Problema da mochila

Seja $I = \{1, 2, \ldots, n\}$ um conjunto de itens. Cada item i tem um valor c_i e um peso w_i. Suponha que temos uma mochila com capacidade W (peso máximo que suporta).

O objetivo é escolher um subconjunto $S \subseteq I$ de itens tais que

- a soma dos pesos dos itens em S não ultrapassa W e
- a soma dos valores dos itens em S é máximo.

O Problema da Mochila

Problema da mochila

Seja $I = \{1, 2, \ldots, n\}$ um conjunto de itens. Cada item i tem um valor c_i e um peso w_i. Suponha que temos uma mochila com capacidade W (peso máximo que suporta).

O objetivo é escolher um subconjunto $S \subseteq I$ de itens tais que

- a soma dos pesos dos itens em S não ultrapassa W e
- a soma dos valores dos itens em S é máximo.

Exemplo: $n = 3, v = (10, 8, 9), w = (1, 2, 4)$ e $W = 6$.
Uma solução ótima é $S = \{1, 3\}$.
O peso de S é $w_1 + w_3 = 1 + 4 \leq 6$ e
o valor de S é $c_1 + c_3 = 10 + 9 = 19$.

O Problema da Mochila

Problema da mochila

Seja $I = \{1, 2, \ldots, n\}$ um conjunto de itens. Cada item i tem um valor v_i e um peso w_i. Suponha que temos uma mochila com capacidade W (peso máximo que suporta).

O objetivo é escolher um subconjunto $S \subseteq I$ de itens tais que

- a soma dos pesos dos itens em S não ultrapassa W e
- a soma dos valores dos itens em S é máximo.

Exemplo: $n = 4, v = (10, 7, 25, 24), w = (2, 1, 6, 5)$ e $W = 7$.
Uma solução ótima é $S = \{1, 4\}$.
O peso de S é $w_1 + w_4 = 2 + 5 \leq 7$ e
o valor de S é $v_1 + v_4 = 10 + 24 = 34$.

O Problema da Mochila

Problema da mochila

Seja $I = \{1, 2, \ldots, n\}$ um conjunto de itens. Cada item i tem um valor v_i e um peso w_i. Suponha que temos uma mochila com capacidade W (peso máximo que suporta).

O objetivo é escolher um subconjunto $S \subseteq I$ de itens tais que

- a soma dos pesos dos itens em S não ultrapassa W e
- a soma dos valores dos itens em S é máximo.

O número de subsconjuntos de I é 2^n. Assim, é impraticável enumerar todas as soluções viáveis.

O Problema da Mochila

Problema da mochila

Seja $I = \{1, 2, \ldots, n\}$ um conjunto de itens. Cada item i tem um valor v_i e um peso w_i. Suponha que temos uma mochila com capacidade W (peso máximo que suporta).

O objetivo é escolher um subconjunto $S \subseteq I$ de itens tais que

- a soma dos pesos dos itens em S não ultrapassa W e
- a soma dos valores dos itens em S é máximo.

O número de subsconjuntos de I é 2^n. Assim, é impraticável enumerar todas as soluções viáveis.

Hipótese: os pesos w_i e a capacidade W são **inteiros**.

O Problema da Mochila

- Podemos formular o problema da mochila como um problema de Programação Linear Inteira:

O Problema da Mochila

- Podemos formular o problema da mochila como um problema de Programação Linear Inteira:
  - Para cada item i temos uma variável x_i:
    $x_i = 1$, se o item i estiver na solução ótima e
    $x_i = 0$, caso contrário.

O Problema da Mochila

- Podemos formular o problema da mochila como um problema de Programação Linear Inteira:
  - Para cada item i temos uma variável x_i:
    $x_i = 1$, se o item i estiver na solução ótima e
    $x_i = 0$, caso contrário.
  - A modelagem do problema é simples:

$$\max \sum_{i=1}^{n} v_i x_i \tag{1}$$

$$\sum_{i=1}^{n} w_i x_i \leq W \tag{2}$$

$$x_i \in \{0, 1\} \tag{3}$$

O Problema da Mochila

- Podemos formular o problema da mochila como um problema de Programação Linear Inteira:
  - Para cada item i temos uma variável x_i:
    $x_i = 1$, se o item i estiver na solução ótima e
    $x_i = 0$, caso contrário.
  - A modelagem do problema é simples:

$$\max \sum_{i=1}^{n} v_i x_i \tag{1}$$

$$\sum_{i=1}^{n} w_i x_i \leq W \tag{2}$$

$$x_i \in \{0, 1\} \tag{3}$$

- (1) é a função objetivo e (2-3) é o conjunto de restrições.

Subestrutura ótima (Passo 1)

Considere uma instância do Problema da Mochila com n itens e uma mochila de capacidade W. Seja S uma solução ótima do problema. Considere o item n.

Subestrutura ótima (Passo 1)

Considere uma instância do Problema da Mochila com n itens e uma mochila de capacidade W. Seja S uma solução ótima do problema. Considere o item n.

- Se $n \notin S$, então S é uma solução ótima do problema da mochila com capacidade W considerando-se apenas os $n - 1$ primeiros itens.

Subestrutura ótima (Passo 1)

Considere uma instância do Problema da Mochila com n itens e uma mochila de capacidade W. Seja S uma solução ótima do problema. Considere o item n.

- Se $n \notin S$, então S é uma solução ótima do problema da mochila com capacidade W considerando-se apenas os $n-1$ primeiros itens.
- Se $n \in S$, então $S - \{n\}$ é uma solução ótima do problema da mochila com capacidade $W - w_n$ considerando-se apenas os $n-1$ primeiros itens.

Subestrutura ótima (Passo 1)

Considere uma instância do Problema da Mochila com n itens e uma mochila de capacidade W. Seja S uma solução ótima do problema. Considere o item n.

- Se $n \notin S$, então S é uma solução ótima do problema da mochila com capacidade W considerando-se apenas os $n-1$ primeiros itens.
- Se $n \in S$, então $S - \{n\}$ é uma solução ótima do problema da mochila com capacidade $W - w_n$ considerando-se apenas os $n-1$ primeiros itens.
- Note também que se $w_n > W$ então necessariamente ocorre o primeiro caso.

Recorrência (Passo 2)

Seja $z[i, d]$ o valor ótimo do problema da mochila com capacidade d considerendo-se os i primeiros itens.

Recorrência (Passo 2)

Seja $z[i, d]$ o valor ótimo do problema da mochila com capacidade d considerendo-se os i primeiros itens.

- Se $i = 0$ ou $d = 0$ então $z[i, d] = 0$ (caso base).

Recorrência (Passo 2)

Seja $z[i, d]$ o valor ótimo do problema da mochila com capacidade d considerendo-se os i primeiros itens.

- Se $i = 0$ ou $d = 0$ então $z[i, d] = 0$ (caso base).
- Se $w_i > d$ então $z[i, d] = z[i - 1, d]$.

Recorrência (Passo 2)

Seja $z[i, d]$ o valor ótimo do problema da mochila com capacidade d considerendo-se os i primeiros itens.

- Se $i = 0$ ou $d = 0$ então $z[i, d] = 0$ (caso base).
- Se $w_i > d$ então $z[i, d] = z[i - 1, d]$.
- Se a solução ótima não usa o item i então:

$$z[i, d] = z[i - 1, d].$$

Recorrência (Passo 2)

Seja $z[i, d]$ o valor ótimo do problema da mochila com capacidade d considerendo-se os i primeiros itens.

- Se $i = 0$ ou $d = 0$ então $z[i, d] = 0$ (caso base).
- Se $w_i > d$ então $z[i, d] = z[i - 1, d]$.
- Se a solução ótima não usa o item i então:

$$z[i, d] = z[i - 1, d].$$

- Se a solução ótima usa o item i então:

$$z[i, d] = z[i - 1, d - w_i] + v_i.$$

Recorrência (Passo 2)

Seja $z[i, d]$ o valor ótimo do problema da mochila com capacidade d considerendo-se os i primeiros itens.

- Se $i = 0$ ou $d = 0$ então $z[i, d] = 0$ (caso base).
- Se $w_i > d$ então $z[i, d] = z[i - 1, d]$.
- Se a solução ótima não usa o item i então:

$$z[i, d] = z[i - 1, d].$$

- Se a solução ótima usa o item i então:

$$z[i, d] = z[i - 1, d - w_i] + v_i.$$

- Assim,

$$z[i, d] = \max\{z[i - 1, d], z[i - 1, d - w_i] + v_i\}.$$

Recorrência (Passo 2)

- Seja $z[i, d]$ o valor ótimo do problema da mochila com capacidade d considerando-se os i primeiros itens.
- Temos então a seguinte recorrência:

Recorrência (Passo 2)

- Seja $z[i, d]$ o valor ótimo do problema da mochila com capacidade d considerando-se os i primeiros itens.
- Temos então a seguinte recorrência:

$$z[i,d] = \begin{cases} 0 & \text{se } i = 0, \\ 0 & \text{se } d = 0, \\ z[i-1,d] & \text{se } w_i > d, \\ \max\{z[i-1,d], z[i-1,d-w_i] + v_i\} & \text{se } w_i \leq d. \end{cases}$$

Recorrência (Passo 2)

- Seja $z[i, d]$ o valor ótimo do problema da mochila com capacidade d considerando-se os i primeiros itens.
- Temos então a seguinte recorrência:

$$z[i,d] = \begin{cases} 0 & \text{se } i = 0, \\ 0 & \text{se } d = 0, \\ z[i-1,d] & \text{se } w_i > d, \\ \max\{z[i-1,d], z[i-1,d-w_i] + v_i\} & \text{se } w_i \leq d. \end{cases}$$

- Note que queremos calcular $z[n, W]$.

Solução recursiva

Recorrência:

$$z[i,d] = \begin{cases} 0 & \text{se } i = 0, \\ 0 & \text{se } d = 0, \\ z[i-1,d] & \text{se } w_i > d, \\ \max\{z[i-1,d], z[i-1,d-w_i] + v_i\} & \text{se } w_i \leq d. \end{cases}$$

MOCHILA-REC(c, w, i, d)

1. **se** $i = 0$ **então devolva** 0
2. **se** $d = 0$ **então devolva** 0
3. **se** $w_i > d$ **então devolva** MOCHILA-REC$(c, w, i-1, d)$
4. **devolva** $\max\{$MOCHILA-REC$(c, w, i-1, d)$,
   MOCHILA-REC$(c, w, i-1, d-w_i) + v_i\}$

Solução recursiva

MOCHILA-REC(c, w, i, d)

1. **se** $i = 0$ **então devolva** 0
2. **se** $d = 0$ **então devolva** 0
3. **se** $w_i > d$ **então devolva** MOCHILA-REC$(c, w, i-1, d)$
4. **devolva** $\max\{$MOCHILA-REC$(c, w, i-1, d)$,
   MOCHILA-REC$(c, w, i-1, d-w_i) + v_i\}$

Solução recursiva

MOCHILA-REC(c, w, i, d)

1. **se** $i = 0$ **então devolva** 0
2. **se** $d = 0$ **então devolva** 0
3. **se** $w_i > d$ **então devolva** MOCHILA-REC$(c, w, i-1, d)$
4. **devolva** $\max\{$MOCHILA-REC$(c, w, i-1, d)$,
   MOCHILA-REC$(c, w, i-1, d-w_i) + v_i\}$

Pode-se construir instâncias em que a complexidade de MOCHILA-REC é $\Omega(2^n)$.

Solução recursiva

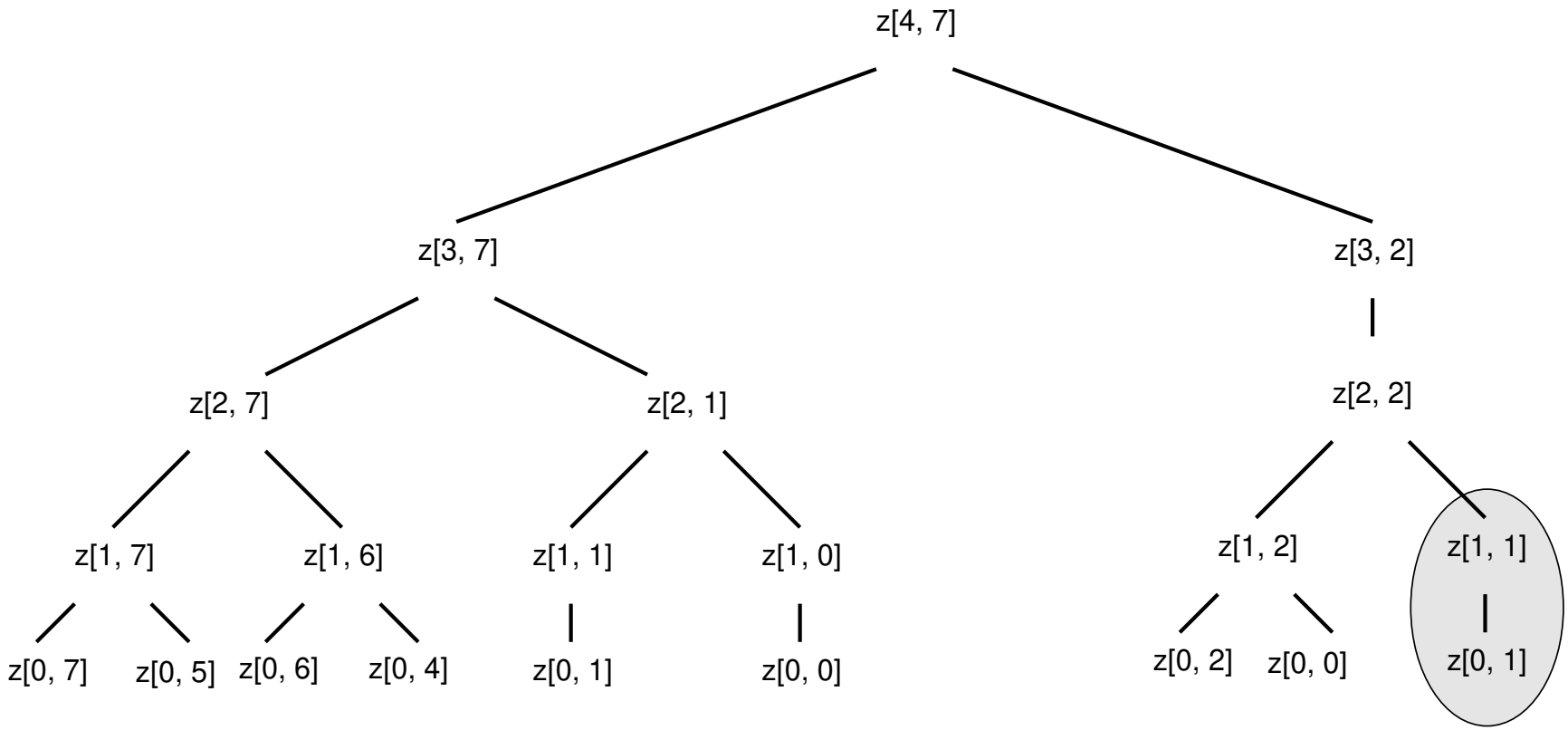
MOCHILA-REC(c, w, i, d)

1. **se** $i = 0$ **então devolva** 0
2. **se** $d = 0$ **então devolva** 0
3. **se** $w_i > d$ **então devolva** MOCHILA-REC$(c, w, i-1, d)$
4. **devolva** $\max\{$MOCHILA-REC$(c, w, i-1, d)$,
   MOCHILA-REC$(c, w, i-1, d-w_i) + v_i\}$

Pode-se construir instâncias em que a complexidade de MOCHILA-REC é $\Omega(2^n)$.

Como usual, a complexidade é decorrente da sobreposição de subproblemas: vários subproblemas são resolvidos várias vezes.

Árvore de recursão



Árvore de recursão de MOCHILA-REC(c, w, n, W) com $n = 4$, $w = (2, 1, 6, 5)$ e $W = 7$.

Programação Dinâmica

- O número total de subproblemas a serem computados é nW.

Programação Dinâmica

- O número total de subproblemas a serem computados é nW.
- Isso porque tanto os pesos dos itens quanto a capacidade da mochila são inteiros.

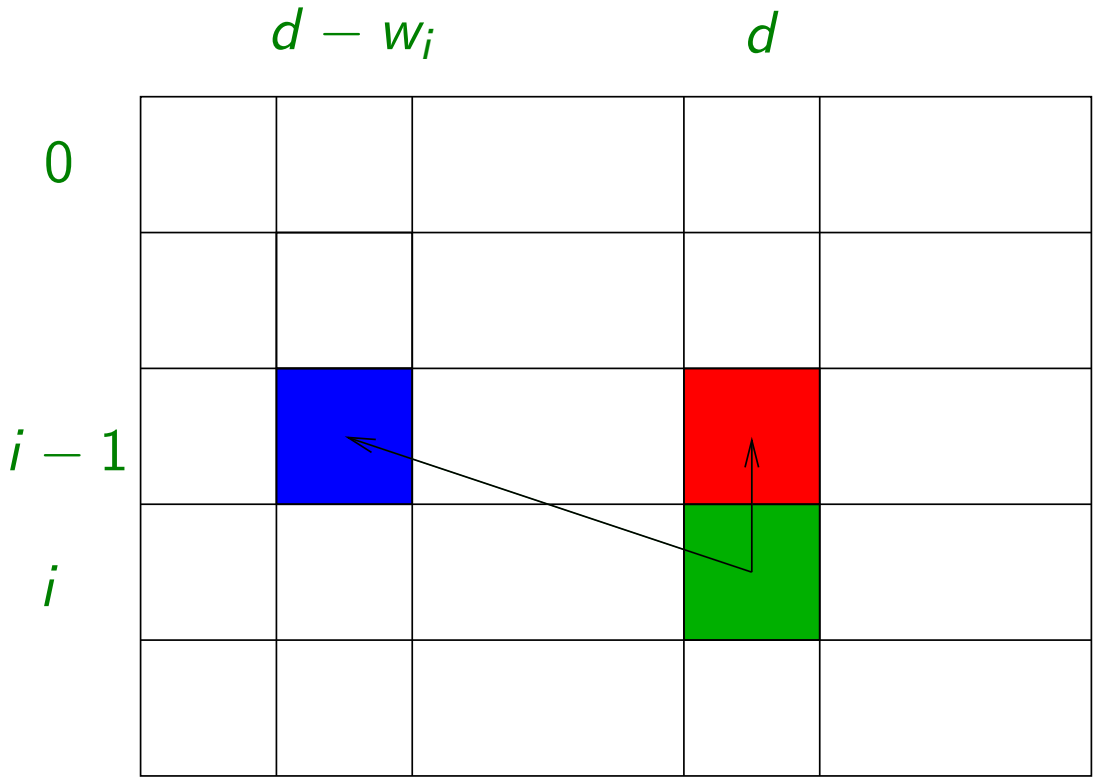
Programação Dinâmica

- O número total de subproblemas a serem computados é nW.
- Isso porque tanto os pesos dos itens quanto a capacidade da mochila são inteiros.
- Podemos então usar programação dinâmica para evitar o recálculo de subproblemas.

Programação Dinâmica

- O número total de subproblemas a serem computados é nW.
- Isso porque tanto os pesos dos itens quanto a capacidade da mochila são inteiros.
- Podemos então usar programação dinâmica para evitar o recálculo de subproblemas.
- Como o cálculo de $z[i, d]$ depende de $z[i-1, d]$ e $z[i-1, d-w_i]$, podemos preencher a tabela $z[\ ,\ ]$ linha a linha.

Preenchimento da tabela



$$z[i, d] = \max\{z[i-1, d], z[i-1, d-w_i] + v_i\}$$

Programação Dinâmica (Passo 3)

$$z[i,d] = \begin{cases} 0 & \text{se } i = 0, \\ 0 & \text{se } d = 0, \\ z[i-1,d] & \text{se } w_i > d, \\ \max\{z[i-1,d], z[i-1,d-w_i] + v_i\} & \text{se } w_i \leq d. \end{cases}$$

PD-MOCHILA(c, w, n, W)

1. **para** $d \leftarrow 0$ **até** W **faça** $z[0,d] \leftarrow 0$
2. **para** $i \leftarrow 1$ **até** n **faça** $z[i,0] \leftarrow 0$
3. **para** $i \leftarrow 1$ **até** n **faça**
4. **para** $d \leftarrow 1$ **até** W **faça**
5. $z[i,d] \leftarrow z[i-1,d]$
6. **se** $w_i \leq d$ **e** $z[i-1,d-w_i] + v_i > z[i,d]$
7. **então** $z[i,d] \leftarrow z[i-1,d-w_i] + v_i$
8. **devolva** $z[n,W]$

Programação Dinâmica: Exemplo

- Exemplo: $n = 4$, $c = (10, 7, 25, 24)$, $w = (2, 1, 6, 5)$ e $W = 7$.

i \ d	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0							
2	0							
3	0							
4	0							

$$z[i, d] = \max\{z[i-1, d], z[i-1, d-w_i] + v_i\} \text{ se } w_i \leq d.$$

Programação Dinâmica: Exemplo

- Exemplo: $n = 4$, $c = (10, 7, 25, 24)$, $w = (2, 1, 6, 5)$ e $W = 7$.

i \ d	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	10	10	10	10	10	10
2	0							
3	0							
4	0							

$$z[i, d] = \max\{z[i-1, d], z[i-1, d-w_i] + v_i\} \text{ se } w_i \leq d.$$

Programação Dinâmica: Exemplo

- Exemplo: $n = 4$, $c = (10, 7, 25, 24)$, $w = (2, 1, 6, 5)$ e $W = 7$.

i \ d	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	10	10	10	10	10	10
2	0	7	10	17	17	17	17	17
3	0							
4	0							

$$z[i, d] = \max\{z[i-1, d], z[i-1, d-w_i] + v_i\} \text{ se } w_i \leq d.$$

Programação Dinâmica: Exemplo

- Exemplo: $n = 4$, $c = (10, 7, 25, 24)$, $w = (2, 1, 6, 5)$ e $W = 7$.

i \ d	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	10	10	10	10	10	10
2	0	7	10	17	17	17	17	17
3	0	7	10	17	17	17	25	32
4	0							

$$z[i, d] = \max\{z[i-1, d], z[i-1, d-w_i] + v_i\} \text{ se } w_i \leq d.$$

Programação Dinâmica: Exemplo

- Exemplo: $n = 4$, $c = (10, 7, 25, 24)$, $w = (2, 1, 6, 5)$ e $W = 7$.

i \ d	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	10	10	10	10	10	10
2	0	7	10	17	17	17	17	17
3	0	7	10	17	17	17	25	32
4	0	7	10	17	17	24	31	34

$$z[i, d] = \max\{z[i-1, d], z[i-1, d-w_i] + v_i\} \text{ se } w_i \leq d.$$

Programação Dinâmica: Exemplo

- Exemplo: $n = 4$, $c = (10, 7, 25, 24)$, $w = (2, 1, 6, 5)$ e $W = 7$.

i \ d	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	10	10	10	10	10	10
2	0	7	10	17	17	17	17	17
3	0	7	10	17	17	17	25	32
4	0	7	10	17	17	24	31	34

$$z[i, d] = \max\{z[i-1, d], z[i-1, d-w_i] + v_i\} \text{ se } w_i \leq d.$$

Programação Dinâmica (Passo 3): Complexidade

- Claramente, a complexidade de PD-Mochila é $O(nW)$.

Programação Dinâmica (Passo 3): Complexidade

- Claramente, a complexidade de PD-Mochila é $O(nW)$.
- O tamanho da entrada é $n \cdot \lg C_{\max} + n \cdot \lg W_{\max} + \lg W$, onde $C_{\max}$ é o maior valor e $W_{\max}$ é o maior peso de um item.
  Informalmente, este é o número de bits necessários para descrever a entrada.

Programação Dinâmica (Passo 3): Complexidade

- Claramente, a complexidade de PD-Mochila é $O(nW)$.
- O tamanho da entrada é $n \cdot \lg C_{\max} + n \cdot \lg W_{\max} + \lg W$, onde $C_{\max}$ é o maior valor e $W_{\max}$ é o maior peso de um item.
  Informalmente, este é o número de bits necessários para descrever a entrada.
- Assim, PD-Mochila não é um algoritmo polinomial.

Programação Dinâmica (Passo 3): Complexidade

- Claramente, a complexidade de PD-Mochila é $O(nW)$.
- O tamanho da entrada é $n \cdot \lg C_{\max} + n \cdot \lg W_{\max} + \lg W$, onde $C_{\max}$ é o maior valor e $W_{\max}$ é o maior peso de um item.
  Informalmente, este é o número de bits necessários para descrever a entrada.
- Assim, PD-Mochila não é um algoritmo polinomial.
- PD-Mochila é um algoritmo pseudo-polinomial: sua complexidade depende de W.

Programação Dinâmica (Passo 3): Complexidade

- Claramente, a complexidade de PD-Mochila é $O(nW)$.
- O tamanho da entrada é $n \cdot \lg C_{\max} + n \cdot \lg W_{\max} + \lg W$, onde $C_{\max}$ é o maior valor e $W_{\max}$ é o maior peso de um item.
  Informalmente, este é o número de bits necessários para descrever a entrada.
- Assim, PD-Mochila não é um algoritmo polinomial.
- PD-Mochila é um algoritmo pseudo-polinomial: sua complexidade depende de W.

  Outro exemplo: o algoritmo ingênuo para testar se um número n é primo tem complexidade $O(n)$. Este é pseudo-polinomial pois o tamanho da entrada é $\lg n$.

Recuperação da Solução (Passo 4)

Recuperação da Solução (Passo 4)

- O algoritmo PD-Mochila não devolve o subconjunto de valor total máximo, apenas o valor máximo.

Recuperação da Solução (Passo 4)

- O algoritmo PD-Mochila não devolve o subconjunto de valor total máximo, apenas o valor máximo.
- É fácil recuperar o subconjunto a partir da tabela z preenchida, lembrando das escolhas que levam à solução ótima.

Recuperação da Solução (Passo 4)

- O algoritmo PD-Mochila não devolve o subconjunto de valor total máximo, apenas o valor máximo.
- É fácil recuperar o subconjunto a partir da tabela z preenchida, lembrando das escolhas que levam à solução ótima.
- Considere o subproblema de calcular $z[i, d]$. O item i faz parte de uma solução ótima deste subproblema se $w_i \leq d$ e $z[i, d] = z[i-1, d-w_i] + v_i$.

Recuperação da Solução (Passo 4)

- O algoritmo PD-Mochila não devolve o subconjunto de valor total máximo, apenas o valor máximo.
- É fácil recuperar o subconjunto a partir da tabela z preenchida, lembrando das escolhas que levam à solução ótima.
- Considere o subproblema de calcular $z[i, d]$. O item i faz parte de uma solução ótima deste subproblema se $w_i \leq d$ e $z[i, d] = z[i-1, d-w_i] + v_i$.

  Ou equivalentemente, $w_i \leq d$ e $z[i, d] \neq z[i-1, d]$.

Recuperação da Solução (Passo 4)

```
MOCHILA-SOLUÇÃO(x, z, n, W)
  para i := 1 até n faça x[i] ← 0
  MOCHILA-SOLUÇÃO-AUX(x, z, n, W)
  devolva (x)

MOCHILA-SOLUÇÃO-AUX(x, z, i, d)
se i ≠ 0 então
  se z[i, d] = z[i − 1, d] então
    x[i] ← 0;
    MOCHILA-SOLUÇÃO-AUX(x, z, i − 1, d)
  senão
    x[i] ← 1;
    MOCHILA-SOLUÇÃO-AUX(x, z, i − 1, d − w_i)
```

Recuperação da Solução - Exemplo

- Exemplo: $c = \{10, 7, 25, 24\}$, $w = \{2, 1, 6, 5\}$ e $W = 7$.

i \ d	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	10	10	10	10	10	10
2	0	7	10	17	17	17	17	17
3	0	7	10	17	17	17	25	32
4	0	7	10	17	17	24	31	34

$$x = (\ ?,\ ?,\ ?,\ ?)$$

$$z[i, d] = \max\{z[i-1, d], z[i-1, d-w_i] + v_i\} \text{ se } w_i \leq d.$$

Recuperação da Solução - Exemplo

- Exemplo: $c = \{10, 7, 25, 24\}$, $w = \{2, 1, 6, 5\}$ e $W = 7$.

i \ d	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	10	10	10	10	10	10
2	0	7	10	17	17	17	17	17
3	0	7	10	17	17	17	25	32
4	0	7	10	17	17	24	31	34

$$x = (\ ?,\ ?,\ ?,\ 1)$$

$$z[i, d] = \max\{z[i-1, d], z[i-1, d-w_i] + v_i\} \text{ se } w_i \leq d.$$

Recuperação da Solução - Exemplo

- Exemplo: $c = \{10, 7, 25, 24\}$, $w = \{2, 1, 6, 5\}$ e $W = 7$.

i \ d	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	10	10	10	10	10	10
2	0	7	10	17	17	17	17	17
3	0	7	10	17	17	17	25	32
4	0	7	10	17	17	24	31	34

$$x = (\ ?,\ ?,\ 0,\ 1)$$

$$z[i, d] = \max\{z[i-1, d], z[i-1, d-w_i] + v_i\} \text{ se } w_i \leq d.$$

Recuperação da Solução - Exemplo

- Exemplo: $c = \{10, 7, 25, 24\}$, $w = \{2, 1, 6, 5\}$ e $W = 7$.

i \ d	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	10	10	10	10	10	10
2	0	7	10	17	17	17	17	17
3	0	7	10	17	17	17	25	32
4	0	7	10	17	17	24	31	34

$$x = (\ ?,\ 0,\ 0,\ 1)$$

$$z[i, d] = \max\{z[i-1, d], z[i-1, d-w_i] + v_i\} \text{ se } w_i \leq d.$$

Recuperação da Solução - Exemplo

- Exemplo: $c = \{10, 7, 25, 24\}$, $w = \{2, 1, 6, 5\}$ e $W = 7$.

i \ d	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	10	10	10	10	10	10
2	0	7	10	17	17	17	17	17
3	0	7	10	17	17	17	25	32
4	0	7	10	17	17	24	31	34

$$x = (\ 1,\ 0,\ 0,\ 1)$$

$$z[i, d] = \max\{z[i-1, d], z[i-1, d-w_i] + v_i\} \text{ se } w_i \leq d.$$

Complexidade

- MOCHILA-SOLUÇÃO tem complexidade $O(n)$.

Complexidade

- MOCHILA-SOLUÇÃO tem complexidade $O(n)$.
- Portanto, a complexidade de tempo e de espaço do algoritmo de programação dinâmica para o problema da mochila é $O(nW)$.

Complexidade

- MOCHILA-SOLUÇÃO tem complexidade $O(n)$.
- Portanto, a complexidade de tempo e de espaço do algoritmo de programação dinâmica para o problema da mochila é $O(nW)$.
- Se quisermos apenas o valor ótimo, é possível economizar memória da seguinte forma. Como cada linha depende apenas da anterior, podemos armazenas apenas duas linhas: a que está sendo preenchida e a anterior.

Subsequência comum mais longa

Definição: Subsequência

Dada uma cadeia $S = a_1 \dots a_n$, dizemos que $S' = b_1 \dots b_p$ é uma subsequência de S se existem p índices $i(j)$ satisfazendo:

(a) $i(j) \in \{1, \dots, n\}$ para todo $j \in \{1, \dots, p\}$;

(b) $i(j) < i(j+1)$ para todo $j \in \{1, \dots, p-1\}$;

(c) $b_j = a_{i(j)}$ para todo $j \in \{1, \dots, p\}$.

Subsequência comum mais longa

Definição: Subsequência

Dada uma cadeia $S = a_1 \dots a_n$, dizemos que $S' = b_1 \dots b_p$ é uma subsequência de S se existem p índices $i(j)$ satisfazendo:

(a) $i(j) \in \{1, \dots, n\}$ para todo $j \in \{1, \dots, p\}$;

(b) $i(j) < i(j+1)$ para todo $j \in \{1, \dots, p-1\}$;

(c) $b_j = a_{i(j)}$ para todo $j \in \{1, \dots, p\}$.

- **Exemplo:** $S = ABCDEFG$ e $S' = ADFG$.

Subsequência comum mais longa

Definição: Subsequência

Dada uma cadeia $S = a_1 \dots a_n$, dizemos que $S' = b_1 \dots b_p$ é uma subsequência de S se existem p índices $i(j)$ satisfazendo:

(a) $i(j) \in \{1, \dots, n\}$ para todo $j \in \{1, \dots, p\}$;

(b) $i(j) < i(j+1)$ para todo $j \in \{1, \dots, p-1\}$;

(c) $b_j = a_{i(j)}$ para todo $j \in \{1, \dots, p\}$.

- **Exemplo:** $S = ABCDEFG$ e $S' = ADFG$.
- Ou seja, $i(j)$ é a posição da cadeia S na qual b_j aparece e os b_j's tem que aparecer na mesma ordem em S.

Subsequência comum mais longa

Definição: Subsequência

Dada uma cadeia $S = a_1 \dots a_n$, dizemos que $S' = b_1 \dots b_p$ é uma subsequência de S se existem p índices $i(j)$ satisfazendo:

(a) $i(j) \in \{1, \dots, n\}$ para todo $j \in \{1, \dots, p\}$;

(b) $i(j) < i(j+1)$ para todo $j \in \{1, \dots, p-1\}$;

(c) $b_j = a_{i(j)}$ para todo $j \in \{1, \dots, p\}$.

- **Exemplo:** $S = ABCDEFG$ e $S' = ADFG$.
- Ou seja, $i(j)$ é a posição da cadeia S na qual b_j aparece e os b_j's tem que aparecer na mesma ordem em S.
- Os caracteres não precisam ser consecutivos em nenhuma das cadeias.

Subsubsequência comum mais longa

Problema da Subsubsequência Comum Mais Longa

Dadas duas cadeias de caracteres X e Y sobre um alfabeto Σ, determinar uma subsubsequência comum mais longa Z de X e Y

Subsubsequência comum mais longa

Problema da Subsubsequência Comum Mais Longa

Dadas duas cadeias de caracteres X e Y sobre um alfabeto Σ, determinar uma subsubsequência comum mais longa Z de X e Y

- $X = \epsilon$, $Y = ABC$
  $Z = \epsilon$

Subsubsequência comum mais longa

Problema da Subsubsequência Comum Mais Longa

Dadas duas cadeias de caracteres X e Y sobre um alfabeto Σ, determinar uma subsubsequência comum mais longa Z de X e Y

- $X = \epsilon$, $Y = ABC$
  $Z = \epsilon$
- $X = ABC$, $Y = ADB$
  $Z = AB$

Subsubsequência comum mais longa

Problema da Subsubsequência Comum Mais Longa

Dadas duas cadeias de caracteres X e Y sobre um alfabeto Σ, determinar uma subsubsequência comum mais longa Z de X e Y

- $X = \epsilon$, $Y = ABC$
  $Z = \epsilon$
- $X = ABC$, $Y = ADB$
  $Z = AB$
- $X = ABCDEF$, $Y = BAEFCD$
  $Z = AEF$ ou $Z = ACD$ ou $Z = BEF$

Subsubsequência comum mais longa

Problema da Subsubsequência Comum Mais Longa

Dadas duas cadeias de caracteres X e Y sobre um alfabeto Σ, determinar uma subsubsequência comum mais longa Z de X e Y

- $X = \epsilon$, $Y = ABC$
  $Z = \epsilon$
- $X = ABC$, $Y = ADB$
  $Z = AB$
- $X = ABCDEF$, $Y = BAEFCD$
  $Z = AEF$ ou $Z = ACD$ ou $Z = BEF$
- $X = ABCBDAB$, $Y = BDCABA$
  $Z = BCBA$ ou $Z = BDAB$

Subsequência comum mais longa

- É um problema de otimização combinatória. Será que tem subestrutura ótima?

Subsequência comum mais longa

- É um problema de otimização combinatória. Será que tem subestrutura ótima?
- **Notação:** Seja S uma cadeia de tamanho n. Para todo $i = 1, \ldots, n$, o prefixo de tamanho i de S é denotado por S_i.

Subsequência comum mais longa

- É um problema de otimização combinatória. Será que tem subestrutura ótima?
- **Notação:** Seja S uma cadeia de tamanho n. Para todo $i = 1, \ldots, n$, o prefixo de tamanho i de S é denotado por S_i.
- **Exemplo:** Para $S = ABCDEFG$, $S_2 = AB$ e $S_4 = ABCD$.

Subsequência comum mais longa

- É um problema de otimização combinatória. Será que tem subestrutura ótima?
- **Notação:** Seja S uma cadeia de tamanho n. Para todo $i = 1, \ldots, n$, o prefixo de tamanho i de S é denotado por S_i.
- **Exemplo:** Para $S = ABCDEFG$, $S_2 = AB$ e $S_4 = ABCD$.
- **Definição:** $c[i, j]$ é o tamanho de uma subsequência comum mais longa dos prefixos X_i e Y_j. Logo, se $|X| = m$ e $|Y| = n$, $c[m, n]$ é o valor ótimo.

Subestrutura ótima e recorrência (Passos 1 e 2)

Para cadeias X e Y, denote por $\texttt{SCML}(X, Y)$ uma subsequência comum mais longa de X e Y.

Teorema. Sejam $X = x_1 x_2 \ldots x_{m-1} x_m$ e $Y = y_1 y_2 \ldots y_{n-1} y_n$ cadeias e seja $Z = \texttt{SCML}(X, Y) = z_1 z_2 \ldots z_k$.

1. Se $x_m = y_n$ então $z_k = x_m = y_n$ e $Z_{k-1} = \texttt{SCML}(X_{m-1}, Y_{n-1})$.
2. Se $x_m \neq y_n$ então $z_k \neq x_m$ implica que $Z = \texttt{SCML}(X_{m-1}, Y)$.
3. Se $x_m \neq y_n$ então $z_k \neq y_n$ implica que $Z = \texttt{SCML}(X, Y_{n-1})$.

Subestrutura ótima e recorrência (Passos 1 e 2)

Para cadeias X e Y, denote por $\texttt{SCML}(X, Y)$ uma subsequência comum mais longa de X e Y.

Teorema. Sejam $X = x_1 x_2 \ldots x_{m-1} x_m$ e $Y = y_1 y_2 \ldots y_{n-1} y_n$ cadeias e seja $Z = \texttt{SCML}(X, Y) = z_1 z_2 \ldots z_k$.

1. Se $x_m = y_n$ então $z_k = x_m = y_n$ e $Z_{k-1} = \texttt{SCML}(X_{m-1}, Y_{n-1})$.
2. Se $x_m \neq y_n$ então $z_k \neq x_m$ implica que $Z = \texttt{SCML}(X_{m-1}, Y)$.
3. Se $x_m \neq y_n$ então $z_k \neq y_n$ implica que $Z = \texttt{SCML}(X, Y_{n-1})$.

Prova de (1): $X = x_1 x_2 \ldots x_{m-1} \alpha$ e $Y = y_1 y_2 \ldots y_{n-1} \alpha$. Se $z_k \neq \alpha$, então Z é uma subsequência comum de X_{m-1} e Y_{n-1}. Mas então $Z' = Z\alpha$ é uma SCML de X e Y de comprimento maior que o de Z, uma contradição.

Subestrutura ótima e recorrência (Passos 1 e 2)

Para cadeias X e Y, denote por $\texttt{SCML}(X, Y)$ uma subsequência comum mais longa de X e Y.

Teorema. Sejam $X = x_1 x_2 \ldots x_{m-1} x_m$ e $Y = y_1 y_2 \ldots y_{n-1} y_n$ cadeias e seja $Z = \texttt{SCML}(X, Y) = z_1 z_2 \ldots z_k$.

1. Se $x_m = y_n$ então $z_k = x_m = y_n$ e $Z_{k-1} = \texttt{SCML}(X_{m-1}, Y_{n-1})$.
2. Se $x_m \neq y_n$ então $z_k \neq x_m$ implica que $Z = \texttt{SCML}(X_{m-1}, Y)$.
3. Se $x_m \neq y_n$ então $z_k \neq y_n$ implica que $Z = \texttt{SCML}(X, Y_{n-1})$.

Prova de (1): $X = x_1 x_2 \ldots x_{m-1} \alpha$ e $Y = y_1 y_2 \ldots y_{n-1} \alpha$. Se $z_k \neq \alpha$, então Z é uma subsequência comum de X_{m-1} e Y_{n-1}. Mas então $Z' = Z\alpha$ é uma SCML de X e Y de comprimento maior que o de Z, uma contradição.

Assim, $z_k = \alpha$.

Subestrutura ótima e recorrência (Passos 1 e 2)

Para cadeias X e Y, denote por $\texttt{SCML}(X, Y)$ uma subsequência comum mais longa de X e Y.

Teorema. Sejam $X = x_1 x_2 \ldots x_{m-1} x_m$ e $Y = y_1 y_2 \ldots y_{n-1} y_n$ cadeias e seja $Z = \texttt{SCML}(X, Y) = z_1 z_2 \ldots z_k$.

1. Se $x_m = y_n$ então $z_k = x_m = y_n$ e $Z_{k-1} = \texttt{SCML}(X_{m-1}, Y_{n-1})$.
2. Se $x_m \neq y_n$ então $z_k \neq x_m$ implica que $Z = \texttt{SCML}(X_{m-1}, Y)$.
3. Se $x_m \neq y_n$ então $z_k \neq y_n$ implica que $Z = \texttt{SCML}(X, Y_{n-1})$.

Prova de (1): $X = x_1 x_2 \ldots x_{m-1} \alpha$ e $Y = y_1 y_2 \ldots y_{n-1} \alpha$. Se $z_k \neq \alpha$, então Z é uma subsequência comum de X_{m-1} e Y_{n-1}. Mas então $Z' = Z\alpha$ é uma SCML de X e Y de comprimento maior que o de Z, uma contradição.

Assim, $z_k = \alpha$. Suponha por contradição que Z_{k-1} não seja uma SCML de X_{m-1} e Y_{n-1}. Assim, existe uma SCML Z' de X_{m-1} e Y_{n-1} de comprimento maior que o de Z_{k-1}. Mas então $Z'\alpha$ é uma subsequência comum de X e Y de comprimento maior que o de Z, uma contradição.

Subestrutura ótima e recorrência (Passos 1 e 2)

Para cadeias X e Y, denote por $\texttt{SCML}(X, Y)$ uma subsequência comum mais longa de X e Y.

Teorema. Sejam $X = x_1 x_2 \ldots x_{m-1} x_m$ e $Y = y_1 y_2 \ldots y_{n-1} y_n$ cadeias e seja $Z = \texttt{SCML}(X, Y) = z_1 z_2 \ldots z_k$.

1. Se $x_m = y_n$ então $z_k = x_m = y_n$ e $Z_{k-1} = \texttt{SCML}(X_{m-1}, Y_{n-1})$.
2. Se $x_m \neq y_n$ então $z_k \neq x_m$ implica que $Z = \texttt{SCML}(X_{m-1}, Y)$.
3. Se $x_m \neq y_n$ então $z_k \neq y_n$ implica que $Z = \texttt{SCML}(X, Y_{n-1})$.

Subestrutura ótima e recorrência (Passos 1 e 2)

Para cadeias X e Y, denote por $\texttt{SCML}(X, Y)$ uma subsequência comum mais longa de X e Y.

Teorema. Sejam $X = x_1 x_2 \ldots x_{m-1} x_m$ e $Y = y_1 y_2 \ldots y_{n-1} y_n$ cadeias e seja $Z = \texttt{SCML}(X, Y) = z_1 z_2 \ldots z_k$.

1. Se $x_m = y_n$ então $z_k = x_m = y_n$ e $Z_{k-1} = \texttt{SCML}(X_{m-1}, Y_{n-1})$.
2. Se $x_m \neq y_n$ então $z_k \neq x_m$ implica que $Z = \texttt{SCML}(X_{m-1}, Y)$.
3. Se $x_m \neq y_n$ então $z_k \neq y_n$ implica que $Z = \texttt{SCML}(X, Y_{n-1})$.

Prova de (2): $X = x_1 x_2 \ldots x_{m-1} \alpha$ e $Y = y_1 y_2 \ldots y_n$. Se $z_k \neq \alpha$, então Z é a SCML de X_{m-1} e Y. Caso contrário, existe uma SCML Z' de X_{m-1} e Y de comprimento maior que o de Z. Mas Z' também é uma subsequência comum de X e Y, uma contradição.

Subestrutura ótima e recorrência (Passos 1 e 2)

Para cadeias X e Y, denote por $\texttt{SCML}(X, Y)$ uma subsequência comum mais longa de X e Y.

Teorema. Sejam $X = x_1 x_2 \ldots x_{m-1} x_m$ e $Y = y_1 y_2 \ldots y_{n-1} y_n$ cadeias e seja $Z = \texttt{SCML}(X, Y) = z_1 z_2 \ldots z_k$.

1. Se $x_m = y_n$ então $z_k = x_m = y_n$ e $Z_{k-1} = \texttt{SCML}(X_{m-1}, Y_{n-1})$.
2. Se $x_m \neq y_n$ então $z_k \neq x_m$ implica que $Z = \texttt{SCML}(X_{m-1}, Y)$.
3. Se $x_m \neq y_n$ então $z_k \neq y_n$ implica que $Z = \texttt{SCML}(X, Y_{n-1})$.

Prova de (2): $X = x_1 x_2 \ldots x_{m-1} \alpha$ e $Y = y_1 y_2 \ldots y_n$. Se $z_k \neq \alpha$, então Z é a SCML de X_{m-1} e Y. Caso contrário, existe uma SCML Z' de X_{m-1} e Y de comprimento maior que o de Z. Mas Z' também é uma subsequência comum de X e Y, uma contradição.

Prova de (3): análogo.

Subestrutura ótima e recorrência (Passos 1 e 2)

Para cadeias X e Y, denote por $\texttt{SCML}(X, Y)$ uma subsequência comum mais longa de X e Y.

Teorema. Sejam $X = x_1 x_2 \ldots x_{m-1} x_m$ e $Y = y_1 y_2 \ldots y_{n-1} y_n$ cadeias e seja $Z = \texttt{SCML}(X, Y) = z_1 z_2 \ldots z_k$.

1. Se $x_m = y_n$ então $z_k = x_m = y_n$ e $Z_{k-1} = \texttt{SCML}(X_{m-1}, Y_{n-1})$.
2. Se $x_m \neq y_n$ então $z_k \neq x_m$ implica que $Z = \texttt{SCML}(X_{m-1}, Y)$.
3. Se $x_m \neq y_n$ então $z_k \neq y_n$ implica que $Z = \texttt{SCML}(X, Y_{n-1})$.

Subestrutura ótima e recorrência (Passos 1 e 2)

Para cadeias X e Y, denote por $\texttt{SCML}(X, Y)$ uma subsequência comum mais longa de X e Y.

Teorema. Sejam $X = x_1 x_2 \ldots x_{m-1} x_m$ e $Y = y_1 y_2 \ldots y_{n-1} y_n$ cadeias e seja $Z = \texttt{SCML}(X, Y) = z_1 z_2 \ldots z_k$.

1. Se $x_m = y_n$ então $z_k = x_m = y_n$ e $Z_{k-1} = \texttt{SCML}(X_{m-1}, Y_{n-1})$.
2. Se $x_m \neq y_n$ então $z_k \neq x_m$ implica que $Z = \texttt{SCML}(X_{m-1}, Y)$.
3. Se $x_m \neq y_n$ então $z_k \neq y_n$ implica que $Z = \texttt{SCML}(X, Y_{n-1})$.

Fórmula de Recorrência:

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

Programação Dinâmica (Passo 3)

Fórmula de Recorrência:

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

Programação Dinâmica (Passo 3)

Fórmula de Recorrência:

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

- A recorrência diz que o valor $c[i,j]$ depende apenas de $c[i-1, j-1]$, $c[i-1, j]$ e $c[i, j-1]$.

Programação Dinâmica (Passo 3)

Fórmula de Recorrência:

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

- A recorrência diz que o valor $c[i,j]$ depende apenas de $c[i-1, j-1]$, $c[i-1, j]$ e $c[i, j-1]$.
- Assim, podemos percorrer a matriz c linha a linha e da esquerda para a direita (ou seja, do "modo natural").

Programação Dinâmica (Passo 3)

Fórmula de Recorrência:

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

- A recorrência diz que o valor $c[i,j]$ depende apenas de $c[i-1, j-1]$, $c[i-1, j]$ e $c[i, j-1]$.
- Assim, podemos percorrer a matriz c linha a linha e da esquerda para a direita (ou seja, do "modo natural").
- Com isto, podemos sempre preencher corretamente a posição atual olhando os valores de posições anteriores já preenchidas.

Programação Dinâmica (Passo 3)

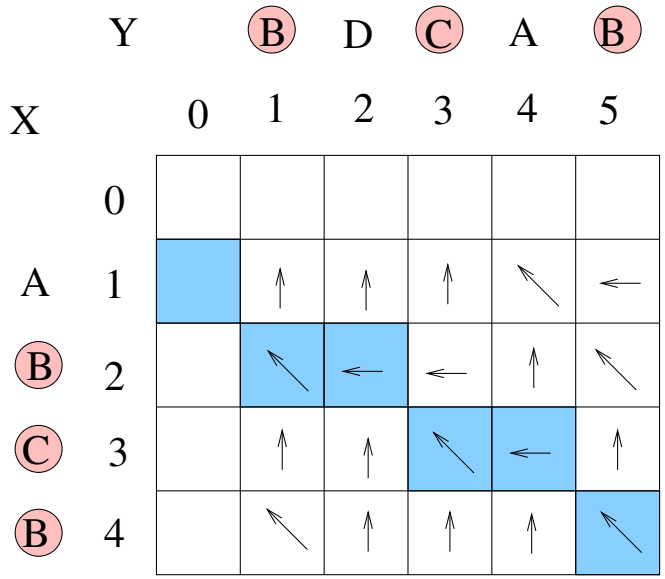
SUBCADEIA-COMUM-MAIS-LONGA(X, Y, m, n)

1. **para** $i \leftarrow 0$ **até** m **faça** $c[i, 0] \leftarrow 0$
2. **para** $j \leftarrow 1$ **até** n **faça** $c[0, j] \leftarrow 0$
3. **para** $i \leftarrow 1$ **até** m **faça**
4. **para** $j \leftarrow 1$ **até** n **faça**
5. **se** $x_i = y_j$ **então**
6. $c[i, j] \leftarrow c[i-1, j-1] + 1$; $b[i, j] \leftarrow$ "$\nwarrow$"
7. **senão**
8. **se** $c[i, j-1] > c[i-1, j]$ **então**
9. $c[i, j] \leftarrow c[i, j-1]$; $b[i, j] \leftarrow$ "$\leftarrow$"
10. **senão**
11. $c[i, j] \leftarrow c[i-1, j]$; $b[i, j] \leftarrow$ "$\uparrow$"
12. **devolva** $(c[m, n], b)$

Programação Dinâmica (Passo 3) - Exemplo

- **Exemplo:** $X = ABCB$ e $Y = BDCAB$, $m = 4$ e $n = 5$.

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



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

Programação Dinâmica (Passo 3) - Complexidade

- Claramente, a complexidade do algoritmo é $O(mn)$.

Programação Dinâmica (Passo 3) - Complexidade

- Claramente, a complexidade do algoritmo é $O(mn)$.
- O algoritmo não encontra a subsequência comum de tamanho máximo, apenas seu tamanho.

Programação Dinâmica (Passo 3) - Complexidade

- Claramente, a complexidade do algoritmo é $O(mn)$.
- O algoritmo não encontra a subsequência comum de tamanho máximo, apenas seu tamanho.
- Com a tabela b preenchida, é fácil encontrar a subsequência comum mais longa.

Recuperação da Solução (Passo 4)

IMPRIME-SCML(b, X, i, j)

1. **se** $i = 0$ e $j = 0$ **então retorne**
2. **se** $b[i, j] =$ "$\nwarrow$" **então**
3. IMPRIME-SCML$(b, X, i - 1, j - 1)$
4. **imprima** x_i
5. **senão**
6. **se** $b[i, j] =$ "$\uparrow$" **então**
7. IMPRIME-SCML$(b, X, i - 1, j)$
8. **senão**
9. IMPRIME-SCML$(b, X, i, j - 1)$

Para recuperar a solução, basta chamar IMPRIME-SCML(b, X, m, n).

Complexidade

- A determinação da subsequência comum mais longa é feita em tempo $O(mn)$ no pior caso.

Complexidade

- A determinação da subsequência comum mais longa é feita em tempo $O(mn)$ no pior caso.
- Portanto, a complexidade de tempo e de espaço do algoritmo de programação dinâmica para o problema da subsequência comum mais longa é $O(mn)$.

Complexidade

- A determinação da subsequência comum mais longa é feita em tempo $O(mn)$ no pior caso.
- Portanto, a complexidade de tempo e de espaço do algoritmo de programação dinâmica para o problema da subsequência comum mais longa é $O(mn)$.
- Note que a tabela b é dispensável. Podemos economizar memória recuperando a solução a partir da tabela c. Ainda assim, o gasto de memória seria $O(mn)$.

Complexidade

- A determinação da subsequência comum mais longa é feita em tempo $O(mn)$ no pior caso.
- Portanto, a complexidade de tempo e de espaço do algoritmo de programação dinâmica para o problema da subsequência comum mais longa é $O(mn)$.
- Note que a tabela b é dispensável. Podemos economizar memória recuperando a solução a partir da tabela c. Ainda assim, o gasto de memória seria $O(mn)$.
- Caso não haja interesse em determinar a subsequência comum mais longa, mas apenas seu tamanho, é possível reduzir o gasto de memória para $O(\min\{n, m\})$: basta registrar apenas a linha da tabela sendo preenchida e a anterior.