Projeto e Análise de Algoritmos

Hamilton José Brumatto

Bacharelado em Ciência da Computação - UESC

9 de abril de 2019

Programação Dinâmica

Material desenvolvido com base no material de aula desenvolvido por: Cid Carvalho de Souza, Cândida Nunes da Silva e Orlando Lee

Conceitos Básicos

- O paradigma de Programação Dinâmica aplica-se basicamente a problemas de **otimização**.
- Pode-se aplicar este paradigmas em problemas onde há:
  - **Subestrutura Ótima:** As soluções ótimas do problema incluem soluções ótimas de subproblemas.
  - **Sobreposição de Subproblemas:** O cálculo da solução através de recursão implica no recálculo de subproblemas.

Técnica de Programação Dinâmica

- A técnica de Programação Dinâmica visa evitar o recálculo desnecessário das soluções dos subproblemas.
- As soluções dos subproblemas são armazenadas em tabelas.
- Para que o algoritmo de programação dinâmica seja eficiente, é preciso que o número total de subproblemas que devem ser resolvidos seja pequeno (polinomial no tamanho da entrada).

Multiplicação de Matrizes

Considere o seguinte problema computacional:

Problema: Multiplicação de Matrizes

Calcular o número mínimo de operações de multiplicação (escalar) necessários para computar a matriz M dada por:

$$M = M_1 \times M_2 \times ...M_i... \times M_n$$

onde M_i é uma matriz de b_{i-1} linhas e b_i colunas, para todo $i \in 1, ..., n$.

- Como as matrizes são multiplicadas aos pares, é necessário encontrar uma parentização ótima para a cadeia de matrizes.

Cálculo do número de operações

- Para calcular a matriz M' dada por $M_i \times M_{i+1}$ teremos:
  - M_i são b_{i-1} linhas e b_i colunas.
  - M_{i+1} são b_i linhas e $b_i + 1$ colunas.
- Cada elemento M'_{pq} é obtido multiplicando-se b_i elementos da linha p de M_i pelos b_i elementos da coluna q de M_{i+1}, como p varia de 1 a b_{i-1} e q varia de 1 a b_{i+1}, então são $b_{i-1} \cdot b_i \cdot b_{i+1}$ multiplicações.

Exemplo do problema

- Qual é o mínimo de multiplicações escalares necessárias para computar $M = M_1 \times M_2 \times M_3 \times M_4$ com $b = \{200, 2, 30, 20, 5\}$?
- As possibilidades de parentização são:

$$
\begin{aligned}
M &= (M_1 \times (M_2 \times (M_3 \times M_4))) \rightarrow 5.300 \text{ multiplicações} \\
M &= (M_1 \times ((M_2 \times M_3) \times M_4)) \rightarrow 3.400 \text{ multiplicações} \\
M &= ((M_1 \times M_2) \times (M_3 \times M_4)) \rightarrow 4.500 \text{ multiplicações} \\
M &= ((M_1 \times (M_2 \times M_3)) \times M_4) \rightarrow 29.200 \text{ multiplicações} \\
M &= (((M_1 \times M_2) \times M_3) \times M_4) \rightarrow 152.000 \text{ multiplicações}
\end{aligned}
$$

- Então a **ordem** das multiplicações faz grande diferença!

Ataque pela Força Bruta

- Uma opção é calcular o número de multiplicações para todas as possíveis parentizações e escolher a melhor.
- O número de possíveis parentizações é dada pela seguinte fórmula de recorrência:

$$P(n) = \begin{cases} 1, & n = 1 \\ \sum_{k=1}^{n-1} P(k) \cdot P(n-k) & n > 1 \end{cases}$$

- Mas $P(n) \in \Omega(4^n / n^{\frac{3}{2}})$, a estratégia da força bruta é impraticável.

Buscando uma subestrutura ótima

- Inicialmente, para todo (i, j) tal que $1 \leqslant i \leqslant j \leqslant n$, vamos definir as seguintes matrizes:

$$M_{ij} = M_i \times M_{i+1} \times ... \times M_j$$

- Agora, dada uma parentização ótima, existem dois pares de parêntesis que identificam o último par de matrizes que serão multiplicadas. Ou seja, existe k tal que $M = M_{1,k} \times M_{k+1,n}$.
- Como a parentização de M é ótima, as parentizações no cálculo de $M_{1,k}$ e $M_{k+1,n}$ devem ser ótimas também, caso contrário, seria possível obter uma parentização de M ainda melhor!
- Encontramos a SUBESTRUTURA ÓTIMA do problema: a parentização ótima de M inclui a parentização ótima de $M_{1,k}$ e $M_{k+1,n}$.

Calculando o mínimo de multiplicações

- De forma geral, se $m[i,j]$ é número mínimo de multiplicações que devem ser efetuadas para computar $M_i \times M_{i+1} \times ... \times M_j$, então $m[i,j]$ é dado por:

$$m[i,j] = \min \{m[i,k] + m[k+1,j] + b_{i-1} \cdot b_k \cdot b_j\}$$

- Podemos então projetar um algoritmo recursivo (indutivo) para resolver o problema.

Projetando o Algoritmo Recursivo para calcular o mínimo de multiplicações

- Entrada: Vetor b com as dimensões das matrizes e os índices i e j que delimitam o ínicio e o término da subcadeia.
- Saída: O número mínimo de multiplicações escalares necessárias para computar a multiplicação da subcadeia. Esse valor é registrado em uma tabela ($m[i,j]$), bem como o índice da divisão em subcadeias ótimas ($s[i,j]$).
- Este algoritmo será chamado inicialmente para a cadeia $1..n$ de matrizes.

O algoritmo

Algoritmo MinimoMultiplicacoesRecursivo(b,i,j)

se $i = j$ **então**

Retorna 0

$m[i,j] \leftarrow \infty$

para $k \leftarrow i$ **até** $j-1$ **faça**

$q \leftarrow MinimoMultiplicacoesRecursivo(b,i,k)+$
$MinimoMultiplicacoesRecursivo(b,k+1,j)+$
$b[i-1]*b[k]*b[j]$

se $m[i,j] > q$ **então**

$m[i,j] \leftarrow q$

$s[i,j] \leftarrow k$

Retorna $m[i,j]$

Analisando a complexidade do algoritmo encontrado:

- O número mínimo de operações feitas pelo algoritmo recursivo é dado pela recorrência:

$$T(n) \geqslant \begin{cases} 1, & n = 1 \\ 1 + \sum_{k=1}^{n-1} [T(k) + T(n-k) + 1] & n > 1, \end{cases}$$

- Portanto, $T(n) \geqslant 2 \sum_{k=1}^{n-1} T(k) + n$, para $n > 1$.
- É possível provar (por substituição) que $T(n) \geqslant 2^{n-1}$, ou seja, o algoritmo recursivo tem a complexidade $\Omega(2^n)$, também impraticável.

O recálculo

- A ineficiência do algoritmo recursivo deve-se a *sobreposição de subproblemas*: o cálculo do mesmo $m[i, j]$ pode ser requerido em vários subproblemas.
- Por exemplo, para $n = 4$, $m[1, 2]$, $m[2, 3]$, e $m[3, 4]$ são computados duas vezes.
- O número total de $m[i, j]$'s calculados é $O(n^2)$ apenas!
- Portanto, podemos obter um algoritmo mais eficiente se evitarmos recálculos de subproblemas.

Técnica da Memorização[1]

- Cria-se a tabela com os valores ótimos calculados.
- Antes de cada chamada recursiva, faz-se uma consulta à tabela, se o valor já foi calculado, então devolve o valor já calculado.
- Esta técnica não evita chamadas recursivas desnecessárias. Portanto se o tempo (desprezado) da chamada recursiva for da ordem do tempo de operação, então ainda temos problemas.

[1] No CLRS é referido como Memoização

Algoritmo de Memorização

Algoritmo MinimoMultiplicacoesMemorizado(b,n)
  para $i \leftarrow 1$ **até** n **faça**
    para $j \leftarrow 1$ **até** n **faça**
      $m[i,j] \leftarrow \infty$
  Retorna Memorizacao(b,1,n)

- Primeiro é necessário zerar a "Memória" pois alguma informação indica que aquele subproblema já foi calculado.

Algoritmo de Memorização

Algoritmo MEMORIZACAO(b,i,j)

se $m[i,j] < \infty$ **então**

Retorna $m[i,j]$

se $i = j$ **então**

$m[i,j] = 0$

senão

para $k \leftarrow 1$ **até** $j-1$ **faça**

$q \leftarrow Memorizacao(b,i,k) +$
$Memorizacao(b,k+1,j) +$
$b[i-1] * b[k] * b[j]$

se $m[i,j] > q$ **então**

$m[i,j] \leftarrow q; s[i,j] \leftarrow k$

Retorna $m[i,j]$

Técnica da Programação Dinâmica

- Preenche uma tabela que registra o valor ótimo para cada subproblema de forma apropriada.
- A computação do valor ótimo de cada subproblema depende somente de subproblemas já previamente computados.
- Elimina completamente o uso da recursão.
- A abordagem é "bottom-up" calcula-se os valores ótimos para os menores problemas, e com estes valores computa-se os valores para problemas maiores.
- No problema das subcadeias, computamos a partir dos valores crescentes dos tamanhos das subcadeias u, primeiro todas subcadeias de tamanho 2, subcadeias de tamanho 3, ...

O algoritmo baseado em programação dinâmica

Algoritmo MinimoMultiplicacoes(b)

para $i \leftarrow 1$ **até** n **faça**

$m[i,i] \leftarrow 0$

▷ Calcula o mínimo de todas sub-cadeias de tamanho $u+1$

para $u \leftarrow 1$ **até** $n-1$ **faça**

para $i \leftarrow 1$ **até** $n-u$ **faça**

$j \leftarrow i+u$

$m[i,j] \leftarrow \infty$

para $k \leftarrow i$ **até** $j-1$ **faça**

$q \leftarrow m[i,k] + m[k+1,j] +$
$b[i-1] * b[k] * b[j]$

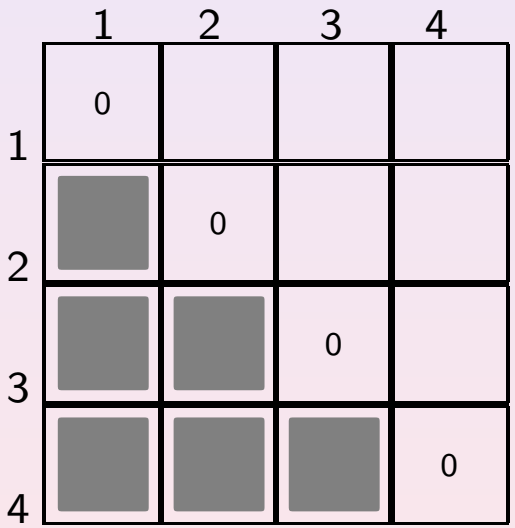
se $q < m[i,j]$ **então**

$m[i,j] \leftarrow q; s[i,j] \leftarrow k$

Retorna (m,s)

Aplicando o código no exemplo

- O vetor b é: $\{200, 2, 30, 20, 5\}$, indexado a partir do índice 0.



- Inicialmente os valores $m[i, i]$ é zerado.

Aplicando o código no exemplo

$$u = 1 \quad \left.\begin{array}{l} i = 1 \\ j = 2 \end{array}\right\} k = 1 \Rightarrow 200{\cdot}2{\cdot}30 + m[1,1] + m[2,2] = 12000 + 0 + 0$$

- $m[1,2] = 12000,\ s[1,2] = 1$

	1	2	3	4
1	0	$\frac{12000}{1}$		
2		0		
3			0	
4				0

Aplicando o código no exemplo

$$u = 1 \quad \left.\begin{array}{l} i = 2 \\ j = 3 \end{array}\right\} k = 2 \Rightarrow 2 \cdot 30 \cdot 20 + m[2,2] + m[3,3] = 1200 + 0 + 0$$

- $m[2,3] = 1200,\ s[2,3] = 2$

	1	2	3	4
1	0	$\frac{12000}{1}$		
2		0	$\frac{1200}{2}$	
3			0	
4				0

Aplicando o código no exemplo

$$u = 1 \quad \left.\begin{array}{l} i = 3 \\ j = 4 \end{array}\right\} k = 3 \Rightarrow 30 \cdot 20 \cdot 5 + m[3,3] + m[4,4] = 3000 + 0 + 0$$

- $m[3,4] = 3000,\ s[3,4] = 3$

	1	2	3	4
1	0	$\frac{12000}{1}$		
2		0	$\frac{1200}{2}$	
3			0	$\frac{3000}{3}$
4				0

Aplicando o código no exemplo

$$u = 2 \quad \left.\begin{array}{l} i = 1 \\ j = 3 \end{array}\right\} \quad \begin{array}{l} k = 1 \Rightarrow 200 \cdot 2 \cdot 20 + m[1,1] + m[2,3] = 8000 + 0 + 1200 = 9200 \\ k = 2 \Rightarrow 200 \cdot 30 \cdot 20 + m[1,2] + m[3,3] = 120000 + 12000 + 0 = 132000 \end{array}$$

- $m[1,3] = 9200,\ s[1,3] = 1$

	1	2	3	4
1	0	$\frac{12000}{1}$	$\frac{9200}{1}$	
2		0	$\frac{1200}{2}$	
3			0	$\frac{3000}{3}$
4				0

Aplicando o código no exemplo

$$u = 2 \quad \left.\begin{array}{l} i = 2 \\ j = 4 \end{array}\right\} \quad \begin{array}{l} k = 2 \Rightarrow 2 \cdot 30 \cdot 5 + m[2,2] + m[3,4] = 300 + 0 + 3000 = 3300 \\ k = 3 \Rightarrow 2 \cdot 20 \cdot 5 + m[2,3] + m[4,4] = 200 + 1200 + 0 = 1400 \end{array}$$

- $m[2,4] = 1400,\ s[2,4] = 3$

	1	2	3	4
1	0	$\frac{12000}{1}$	$\frac{9200}{1}$	
2		0	$\frac{1200}{2}$	$\frac{1400}{3}$
3			0	$\frac{3000}{3}$
4				0

Aplicando o código no exemplo

$$u = 3 \quad \left.\begin{array}{l} i = 1 \\ j = 4 \end{array}\right\} \quad \begin{array}{l} k = 1 \Rightarrow 200 \cdot 2 \cdot 5 + m[1,1] + m[2,4] = 2000 + 0 + 1400 = 3400 \\ k = 2 \Rightarrow 200 \cdot 30 \cdot 5 + m[1,2] + m[3,4] = 30000 + 12000 + 3000 = 45000 \\ k = 3 \Rightarrow 200 \cdot 20 \cdot 5 + m[1,3] + m[4,4] = 20000 + 9200 + 0 = 29200 \end{array}$$

- $m[1,4] = 3400,\ s[1,4] = 1$

	1	2	3	4
1	0	$\frac{12000}{1}$	$\frac{9200}{1}$	$\frac{3400}{1}$
2		0	$\frac{1200}{2}$	$\frac{1400}{3}$
3			0	$\frac{3000}{3}$
4				0

Reconstruindo a expressão

- Vamos reescrever $M_{1,4}$
- $M_{i,i}$ é simplesmente M_i
- $s[1,4] = 1$

	1	2	3	4
1	0	$\frac{12000}{1}$	$\frac{9200}{1}$	$\frac{3400}{1}$
2		0	$\frac{1200}{2}$	$\frac{1400}{3}$
3			0	$\frac{3000}{3}$
4				0

- $M_{1,4} = (M_1) \times (M_{2,4})$

Reconstruindo a expressão

- Vamos reescrever $M_{1,4}$
- $M_{i,i}$ é simplesmente M_i
- $s[2,4] = 3$

	1	2	3	4
1	0	$\frac{12000}{1}$	$\frac{9200}{1}$	$\frac{3400}{1}$
2		0	$\frac{1200}{2}$	$\frac{1400}{3}$
3			0	$\frac{3000}{3}$
4				0

- $M_{1,4} = (M_1) \times ((M_{2,3}) \times (M_4))$

Reconstruindo a expressão

- Vamos reescrever $M_{1,4}$
- $M_{i,i}$ é simplesmente M_i
- $s[2,3] = 2$

	1	2	3	4
1	0	$\frac{12000}{1}$	$\frac{9200}{1}$	$\frac{3400}{1}$
2		0	$\frac{1200}{2}$	$\frac{1400}{3}$
3			0	$\frac{3000}{3}$
4				0

- $M_{1,4} = (M_1) \times (((M_2) \times (M_3)) \times (M_4))$

Reconstruindo a expressão

- Resultado final da parentização para M
- Tirando os parêntesis redundantes

	1	2	3	4
1	0	$\frac{12000}{1}$	$\frac{9200}{1}$	$\frac{3400}{1}$
2		0	$\frac{1200}{2}$	$\frac{1400}{3}$
3			0	$\frac{3000}{3}$
4				0

- $M = M_1 \times ((M_2 \times M_3) \times M_4)$
- O total de multiplicações realizadas para esta parentização é 3400.

Calculando a complexidade do algoritmo

- Como são loops, basta calcular a soma das operações:

$$
\begin{aligned}
T(n) &= \sum_{u=1}^{n-1} \sum_{i=1}^{n-u} \sum_{k=i}^{i+u-1} \Theta(1) \\
&= \sum_{u=1}^{n-1} \sum_{i=1}^{n-u} u\Theta(1) \\
&= \sum_{u=1}^{n-1} u(n-u)\Theta(1) \\
&= \sum_{u=1}^{n-1} (nu - u^2)\Theta(1)
\end{aligned}
$$

Calculando a complexidade do algoritmo

- Sabemos que:

$$\sum_{u=1}^{n-1} nu = \frac{n^3}{2} - \frac{n^2}{2}$$

e

$$\sum_{u=1}^{n-1} u^2 = \frac{n^3}{3} - \frac{n^2}{2} + \frac{n}{6}$$

ficamos com:

$$T(n) = (\frac{n^3}{6} - \frac{n}{6})\Theta(1).$$

- A complexidade é $\Theta(n^3)$
- A complexidade do espaço é $\Theta(n^2)$, já que é necessário armazenar a matriz com os valores ótimos dos subproblemas.

O Problema Binário da Mochila

Considere o seguinte problema computacional:

Dada uma mochila de capacidade W (inteiro) e um conjunto de n itens com tamanho w_i (inteiro) e valor c_i associado a cada item i, queremos determinar quais itens devem ser colocados na mochila de modo a **maximizar** o valor total transportado, respeitando sua capacidade.

- Podemos então assumir as seguintes condições:
  - $\sum_{i=1}^{n} w_i > W$; Não cabem todos os itens na mochila.
  - $0 < w_i \leqslant W$, para todo $i = 1, ..., n$; Cada item individualmente cabe na mochila.

Problema de Programação Linear Inteira

- Criamos uma variável x_i para cada item: $x_i = 1$ se o item i estiver na solução ótima e $x_i = 0$ caso contrário.
- A modelagem do problema é simples:

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

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

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

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

Análise do problema

- Existem 2^n possíveis subconjuntos de itens: um algoritmo de força bruta não é prático.
- É um problema de otimização. Será que existe uma subestrutura ótima?
- Se o item n estiver na solução ótima, o valor desta solução será c_n mais o valor da melhor solução do problema da mochila com capacidade $W - w_n$ considerando-se só os $n - 1$ primeiros itens.
- Se o item n não estiver na solução ótima, o valor ótimo será dado pelo valor da melhor solução do problema da mochila com capacidade W considerando-se só os $n - 1$ primeiros itens.

Subestrutura Ótima

- Seja $z[k, d]$ o valor ótimo do problema da mochila considerando-se uma capacidade d para a mochila que contém um subconjunto dos k primeiros itens da instância original.
- A fórmula de recorrência para computar $z[k, d]$ para todo valor de d e k é:

$$
\begin{aligned}
z[0, d] &= 0 \\
z[k, 0] &= 0 \\
z[k, d] &= \begin{cases} z[k-1, d], & \text{se } w_k > d \\ \max\{z[k-1, d], z[k-1, d-w_k] + c_k\}, & \text{se } w_k \leqslant d \end{cases}
\end{aligned}
$$

Abordagem Recursiva

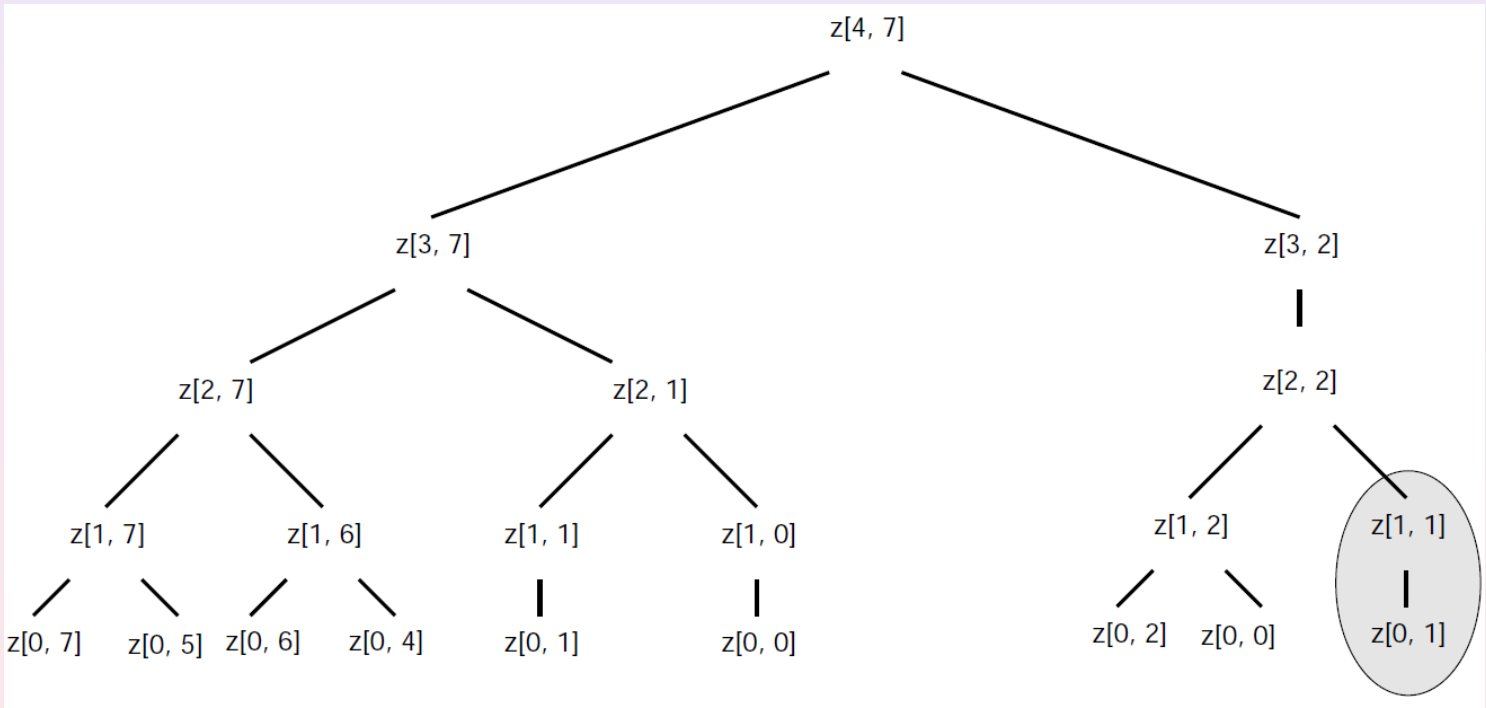
- A complexidade do algoritmo recursivo para este problema no pior caso é dada pela recorrência:

$$T(k,d) = \begin{cases} 1, & k = 0 \text{ ou } d = 0 \\ T(k-1,d) + T(k-1,d-w_k) + 1, & k > 0 \text{ e } d > 0 \end{cases}$$

- Portanto, no pior caso, o algoritmo recursivo tem complexidade $\Omega(2^n)$. Não é prático.
- Precisamos ver se existem sobreposição de subproblemas, poderemos evitar o recálculo com memorização, ou em especial com programação dinâmica.

Analisando a sobreposição de problemas

Vamos considerar: $w = 2, 1, 6, 5$ e $W = 7$

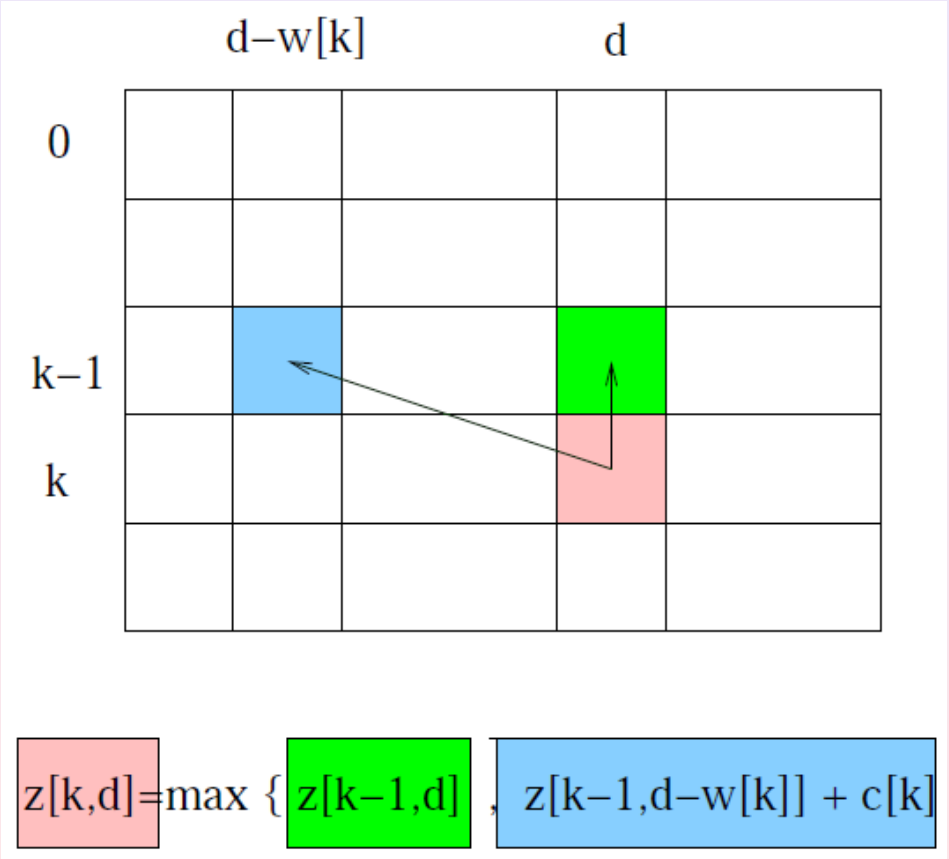


O subproblema $z[1, 1]$ foi computado 2 vezes.

Construindo a solução por programação dinâmica

- O número total máximo de subproblemas a serem computados é nW.
- Tanto o tamanho dos itens quanto a capacidade da mochila são inteiros!
- Podemos usar a programação dinâmica para evitar o recálculo de subproblemas.
- Como o cálculo de $z[k, d]$ depende de $z[k-1, d]$ e $z[k-1, d-w_k]$, preenchemos a tabela linha a linha.
- Ou seja, cada linha representa a consideração de um objeto, podendo ser inserido ou não em qualquer volume da mochila (volumes inteiros), o que for mais vantajoso.

Varrendo a tabela



O algoritmo da Mochila: Problema Binário da Mochila

Entrada: Vetores c e w com o valor e tamanho de cada item, capacidade W da mochila e o número de itens n.
Saída: O valor máximo do total de itens na mochila.

Algoritmo MOCHILA(c,w,W,n)
  para $d \leftarrow 0$ **até** W **faça**
    $z[0, d] \leftarrow 0$
  para $k \leftarrow 1$ **até** n **faça**
    $z[k, 0] \leftarrow 0$
  para $k \leftarrow 1$ **até** n **faça**
    para $q \leftarrow 1$ **até** W **faça**
      $z[k, d] \leftarrow z[k-1, d]$
      se $w_k \leqslant d$ **e** $c_k + z[k-1, d-w_k] > z[k, d]$ **então**
        $z[k, d] \leftarrow c_k + z[k-1, d-w_k]$
  Retorna $(z[n, W])$

Exemplo do Algoritmo

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

k \ 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							

Exemplo do Algoritmo

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

k \ 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							

Exemplo do Algoritmo

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

k \ 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							

Exemplo do Algoritmo

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

k \ 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							

Exemplo do Algoritmo

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

k \ 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

Exemplo do Algoritmo

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

k \ 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

Análise do Algoritmo

- Baseado nos dois loops, encontramos que o algoritmo é $O(nW)$.
- É um algoritmo *pseudo-polinomial* pois depende do valor de W, que é parte da entrada do algoritmo.
- Embora não há indicação no algoritmo do *subconjunto de itens escolhidos*, a escolha é fácil de se recuperar pela matriz gerada.

Algoritmo de recuperação da solução

Algoritmo MochilaSolucao(z,n,W)
  para $i \leftarrow 1$ **até** n **faça**
    $MochilaSolucaoAux(x, z, n, W)$
  Retorna x

Algoritmo MochilaSolucaoAux(x,z,k,d)
  se $k \neq 0$ **então**
    se $z[k, d] = z[k - 1, d]$ **então**
      $x[k] \leftarrow 0$
      $MochilaSolucaoAux(x, z, k - 1, d)$
    senão
      $x[k] \leftarrow 1$
      $MochilaSolucaoAux(x, z, k - 1, d - w_k)$

Exemplo do Algoritmo: Recuperação da solução

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

k \ 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

Exemplo do Algoritmo: Recuperação da solução

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

k \ 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

Exemplo do Algoritmo: Recuperação da solução

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

k \ 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

Exemplo do Algoritmo: Recuperação da solução

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

k \ 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

Exemplo do Algoritmo: Recuperação da solução

- $c = \{10, 7, 25, 24\}$, $w = \{2, 1, 6, 5\}$, e $W = 7$
- $x[1] = 1$, $x[2] = 0$, $x[3] = 0$, $x[4] = 1$

k \ 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

Problema do Troco em Moedas

Considere o seguinte problema computacional:

> Dado um conjunto base de n valores de moedas $M = \{M_1, M_2, ..., M_n\}$, desejo o menor conjunto de moedas que sirva para compor um troco de valor T.

- Poderíamos pensar em um ALGORITMO GULOSO, por exemplo sempre pegar a maior moeda menor que T.
- Veja a seguinte situação: $M = \{1, 2, 5, 7, 10\}$ e $T = 14$.
- Se escolhessemos pela maior moeda, o troco seria composto por: $\{10, 2, 2\}$, no entanto, $\{7, 7\}$ é o menor conjunto.

Problema de Otimização com subestrutura Ótima

- É um problema de otimização, queremos o menor conjunto.
- Também possui uma subestrutura ótima, pois se a k-ésima moeda fizer parte da solução ótima para o problema, $T - M_k$ é um problema cuja solução ótima é parte da solução ótima do problema original.
- Novamente encontramos um algoritmo recursivo.
  - Para um determinado valor T, se possuímos k faces de moedas, a k-ésima moeda pode fazer parte (uma ou mais vezes) ou não do troco, neste caso ficaríamos com:
    $m[T, k] = m[T - M_k, k] + 1$ se a k-ésima moeda fizer parte do troco, ou
    $m[T, k] = m[T, k - 1]$ se a k-ésima moeda não fizer parte do troco.
    De qualquer forma, caímos em um subproblema menor

- A fórmula de recorrência para o problema é:

$$m[T,k] = \begin{cases} 0 & , T = 0 \\ m[T - M_k, k] + 1 & , T > 0 \text{ e } k = 1 \\ m[T, k-1] & , T > 0 \text{ e } k > 1 \text{ e } M_k > T \\ \min\{m[T, k-1], m[T - M_k, k] + 1\} & , T > 0 \text{ e } k > 1 \text{ e } M_k \leqslant T; \end{cases}$$

- Observe que $k \neq 0$, sempre, pois a solução seria ∞, mais ainda, para uma base válida para qualquer valor de T, $k = 1 \rightarrow M_k = \$1$.
- Qual o tempo deste algoritmo?
- $C(T,k) = \begin{cases} 1 & , T = 0 \\ C(T - M_k, k) + C(T, k-1) + 1 & , T > 0 \end{cases}$
- Este algoritmo é pior que o da mochila binária.

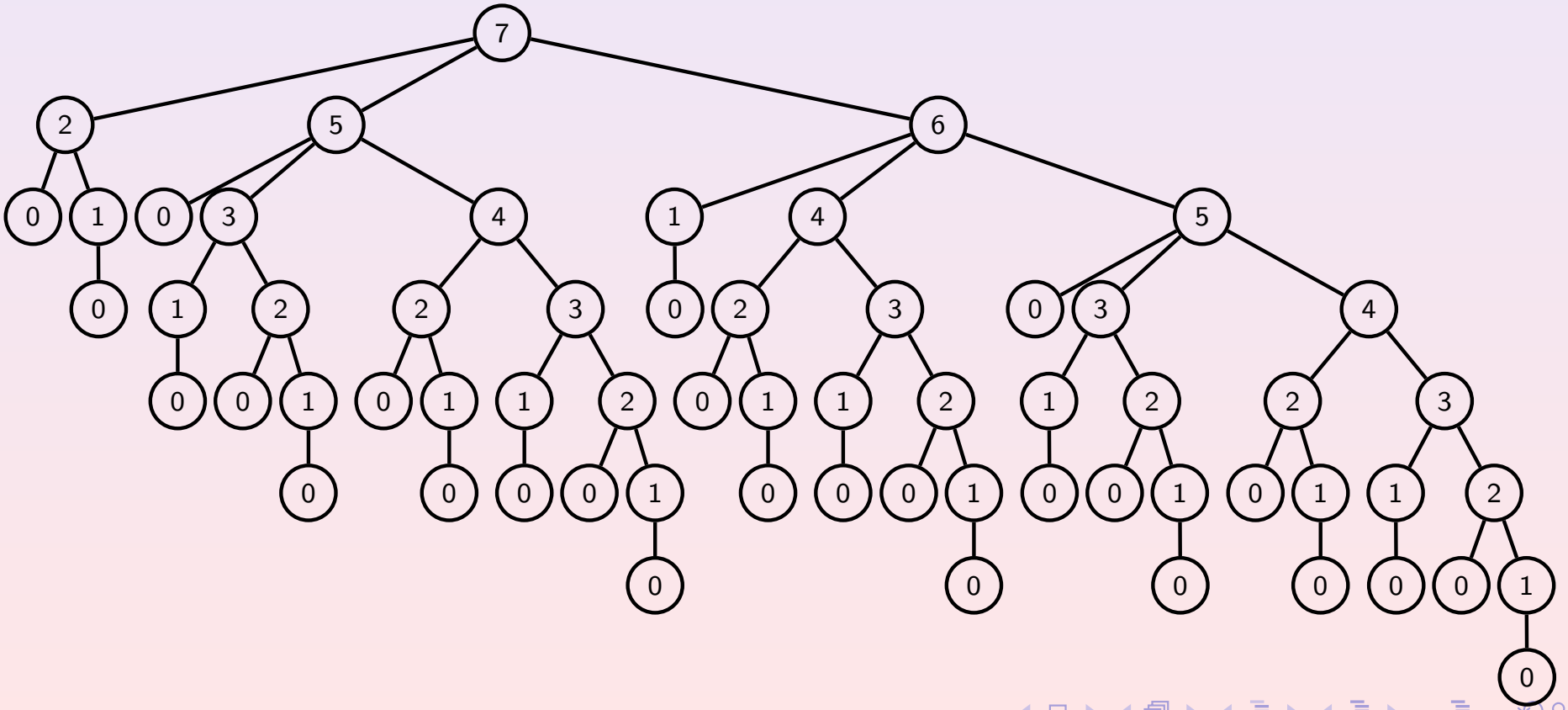
- A fórmula de recorrência para o problema é:

$$m[T,k] = \begin{cases} 0 & , T = 0 \\ m[T - M_k, k] + 1 & , T > 0 \text{ e } k = 1 \\ m[T, k-1] & , T > 0 \text{ e } k > 1 \text{ e } M_k > T \\ \min\{m[T, k-1], m[T - M_k, k] + 1\} & , T > 0 \text{ e } k > 1 \text{ e } M_k \leqslant T; \end{cases}$$

- Observe que $k \neq 0$, sempre, pois a solução seria ∞, mais ainda, para uma base válida para qualquer valor de T, $k = 1 \rightarrow M_k = \$1$.
- Qual o tempo deste algoritmo?
- $C(T,k) = \begin{cases} 1 & ,T = 0 \\ C(T - M_k, k) + C(T, k-1) + 1 & ,T > 0 \end{cases}$
- Este algoritmo é pior que o da mochila binária.

Sobreposição de Subproblemas

- Vamos considerar um caso simples, troco para 7 centavos, que usaria moedas de 1, 2 e 5 centavos apenas.



Aplicando a técnica de Programação Dinâmica

- Aplicando a recorrência de forma iterativa: programação dinâmica:

Algoritmo TrocoOtimo(T,M,n)

para $k \leftarrow 1$ **até** n **faça**

$m[0,k] \leftarrow 0$ ▷ $T = 0$

para $t \leftarrow 1$ **até** T **faça**

$m[t,1] \leftarrow t$ ▷ $k = 1$

para $k \leftarrow 2$ **até** n **faça**

para $t \leftarrow 1$ **até** T **faça**

$m[t,k] = m[t,k-1]$

se $M_k \leqslant t$ **e** $(m[t-M_k,k]+1) < m[t,k]$ **então**

$m[t,k] = m[t-M_k,k]+1$

Retorna $m[T,n]$

Aplicando o Algoritmo no exemplo

- $M = \{1, 2, 5, 7, 10\}$, $n = 5$, $T = 14$

M_k \ T	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$1 k_1															
\$2 k_2															
\$5 k_3															
\$7 k_4															
\$10 k_5															

Aplicando o Algoritmo no exemplo

- $M = \{1, 2, 5, 7, 10\}$, $n = 5$, $T = 14$

M_k \ T	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$1 k_1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$2 k_2	0														
\$5 k_3	0														
\$7 k_4	0														
\$10 k_5	0														

para $k \leftarrow 1$ **até** n **faça**
    $m[0, k] \leftarrow 0$

para $t \leftarrow 1$ **até** T **faça**
    $m[t, 1] \leftarrow t$

Aplicando o Algoritmo no exemplo

- $M = \{1, 2, 5, 7, 10\}$, $n = 5$, $T = 14$

M_k \ T	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$1 k_1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$2 k_2	0	1													
\$5 k_3	0														
\$7 k_4	0														
\$10 k_5	0														

para $k \leftarrow 2$ **até** n **faça**
  para $t \leftarrow 1$ **até** T **faça**
    $m[t, k] = m[t, k-1]$
    se $M_k \leqslant t$ **e** $(m[t - M_k, k] + 1) < m[t, k]$ **então**
      $m[t, k] = m[t - M_k, k] + 1$

Aplicando o Algoritmo no exemplo

- $M = \{1, 2, 5, 7, 10\}$, $n = 5$, $T = 14$

M_k \ T	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$1 k_1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$2 k_2	0	1	1 (+1)												
\$5 k_3	0														
\$7 k_4	0														
\$10 k_5	0														

para $k \leftarrow 2$ **até** n **faça**
  para $t \leftarrow 1$ **até** T **faça**
    $m[t, k] = m[t, k-1]$
    se $M_k \leqslant t$ **e** $(m[t - M_k, k] + 1) < m[t, k]$ **então**
      $m[t, k] = m[t - M_k, k] + 1$

Aplicando o Algoritmo no exemplo

- $M = \{1, 2, 5, 7, 10\}$, $n = 5$, $T = 14$

M_k \ T	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$1 k_1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$2 k_2	0	1	1	2											
\$5 k_3	0														
\$7 k_4	0														
\$10 k_5	0														

para $k \leftarrow 2$ **até** n **faça**
  para $t \leftarrow 1$ **até** T **faça**
    $m[t, k] = m[t, k-1]$
    se $M_k \leqslant t$ **e** $(m[t - M_k, k] + 1) < m[t, k]$ **então**
      $m[t, k] = m[t - M_k, k] + 1$

Aplicando o Algoritmo no exemplo

- $M = \{1, 2, 5, 7, 10\}$, $n = 5$, $T = 14$

M_k \ T	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$1 k_1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$2 k_2	0	1	1	2	2										
\$5 k_3	0														
\$7 k_4	0														
\$10 k_5	0														

para $k \leftarrow 2$ **até** n **faça**
  para $t \leftarrow 1$ **até** T **faça**
    $m[t, k] = m[t, k-1]$
    se $M_k \leqslant t$ **e** $(m[t - M_k, k] + 1) < m[t, k]$ **então**
      $m[t, k] = m[t - M_k, k] + 1$

Aplicando o Algoritmo no exemplo

- $M = \{1, 2, 5, 7, 10\}$, $n = 5$, $T = 14$

M_k \ T	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$1 k_1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$2 k_2	0	1	1	2	2	3									
\$5 k_3	0														
\$7 k_4	0														
\$10 k_5	0														

para $k \leftarrow 2$ **até** n **faça**
  para $t \leftarrow 1$ **até** T **faça**
    $m[t, k] = m[t, k-1]$
    se $M_k \leqslant t$ **e** $(m[t - M_k, k] + 1) < m[t, k]$ **então**
      $m[t, k] = m[t - M_k, k] + 1$

Aplicando o Algoritmo no exemplo

- $M = \{1, 2, 5, 7, 10\}$, $n = 5$, $T = 14$

M_k \ T	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$1 k_1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$2 k_2	0	1	1	2	2	3	3								
\$5 k_3	0														
\$7 k_4	0														
\$10 k_5	0														

+1

para $k \leftarrow 2$ **até** n **faça**
  para $t \leftarrow 1$ **até** T **faça**
    $m[t, k] = m[t, k-1]$
    se $M_k \leqslant t$ **e** $(m[t - M_k, k] + 1) < m[t, k]$ **então**
      $m[t, k] = m[t - M_k, k] + 1$

Aplicando o Algoritmo no exemplo

- $M = \{1, 2, 5, 7, 10\}$, $n = 5$, $T = 14$

M_k \ T	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$1 k_1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$2 k_2	0	1	1	2	2	3	3 +1	4							
\$5 k_3	0														
\$7 k_4	0														
\$10 k_5	0														

para $k \leftarrow 2$ **até** n **faça**
  para $t \leftarrow 1$ **até** T **faça**
    $m[t, k] = m[t, k-1]$
    se $M_k \leqslant t$ **e** $(m[t - M_k, k] + 1) < m[t, k]$ **então**
      $m[t, k] = m[t - M_k, k] + 1$

Aplicando o Algoritmo no exemplo

- $M = \{1, 2, 5, 7, 10\}$, $n = 5$, $T = 14$

M_k \ T	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$1 k_1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$2 k_2	0	1	1	2	2	3	3	4	4						
\$5 k_3	0														
\$7 k_4	0														
\$10 k_5	0														

+1

para $k \leftarrow 2$ **até** n **faça**
  para $t \leftarrow 1$ **até** T **faça**
    $m[t, k] = m[t, k-1]$
    se $M_k \leqslant t$ **e** $(m[t - M_k, k] + 1) < m[t, k]$ **então**
      $m[t, k] = m[t - M_k, k] + 1$

Aplicando o Algoritmo no exemplo

- $M = \{1, 2, 5, 7, 10\}$, $n = 5$, $T = 14$

M_k \ T	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$1 k_1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$2 k_2	0	1	1	2	2	3	3	4	4	5					
\$5 k_3	0														
\$7 k_4	0														
\$10 k_5	0														

+1

para $k \leftarrow 2$ **até** n **faça**
  para $t \leftarrow 1$ **até** T **faça**
    $m[t, k] = m[t, k-1]$
    se $M_k \leqslant t$ **e** $(m[t - M_k, k] + 1) < m[t, k]$ **então**
      $m[t, k] = m[t - M_k, k] + 1$

Aplicando o Algoritmo no exemplo

- $M = \{1, 2, 5, 7, 10\}$, $n = 5$, $T = 14$

M_k \ T	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$1 k_1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$2 k_2	0	1	1	2	2	3	3	4	4	5	5				
\$5 k_3	0														
\$7 k_4	0														
\$10 k_5	0														

+1

para $k \leftarrow 2$ **até** n **faça**
  para $t \leftarrow 1$ **até** T **faça**
    $m[t, k] = m[t, k-1]$
    se $M_k \leqslant t$ **e** $(m[t - M_k, k] + 1) < m[t, k]$ **então**
      $m[t, k] = m[t - M_k, k] + 1$

Aplicando o Algoritmo no exemplo

- $M = \{1, 2, 5, 7, 10\}$, $n = 5$, $T = 14$

M_k \ T	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$1 k_1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$2 k_2	0	1	1	2	2	3	3	4	4	5	5	6			
\$5 k_3	0														
\$7 k_4	0														
\$10 k_5	0														

para $k \leftarrow 2$ **até** n **faça**
  para $t \leftarrow 1$ **até** T **faça**
    $m[t, k] = m[t, k-1]$
    se $M_k \leqslant t$ **e** $(m[t - M_k, k] + 1) < m[t, k]$ **então**
      $m[t, k] = m[t - M_k, k] + 1$

Aplicando o Algoritmo no exemplo

- $M = \{1, 2, 5, 7, 10\}$, $n = 5$, $T = 14$

M_k \ T	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$1 k_1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$2 k_2	0	1	1	2	2	3	3	4	4	5	5	6 +1	6		
\$5 k_3	0														
\$7 k_4	0														
\$10 k_5	0														

para $k \leftarrow 2$ **até** n **faça**
  para $t \leftarrow 1$ **até** T **faça**
    $m[t, k] = m[t, k-1]$
    se $M_k \leqslant t$ **e** $(m[t - M_k, k] + 1) < m[t, k]$ **então**
      $m[t, k] = m[t - M_k, k] + 1$

Aplicando o Algoritmo no exemplo

- $M = \{1, 2, 5, 7, 10\}$, $n = 5$, $T = 14$

M_k \ T	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$1 k_1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$2 k_2	0	1	1	2	2	3	3	4	4	5	5	6	6	7	
\$5 k_3	0														
\$7 k_4	0														
\$10 k_5	0														

para $k \leftarrow 2$ **até** n **faça**
  para $t \leftarrow 1$ **até** T **faça**
    $m[t, k] = m[t, k-1]$
    se $M_k \leqslant t$ **e** $(m[t - M_k, k] + 1) < m[t, k]$ **então**
      $m[t, k] = m[t - M_k, k] + 1$

Aplicando o Algoritmo no exemplo

- $M = \{1, 2, 5, 7, 10\}$, $n = 5$, $T = 14$

M_k \ T	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$1 k_1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$2 k_2	0	1	1	2	2	3	3	4	4	5	5	6	6	7 (+1)	7
\$5 k_3	0														
\$7 k_4	0														
\$10 k_5	0														

para $k \leftarrow 2$ **até** n **faça**
  para $t \leftarrow 1$ **até** T **faça**
    $m[t, k] = m[t, k-1]$
    se $M_k \leqslant t$ **e** $(m[t - M_k, k] + 1) < m[t, k]$ **então**
      $m[t, k] = m[t - M_k, k] + 1$

Aplicando o Algoritmo no exemplo

- $M = \{1, 2, 5, 7, 10\}$, $n = 5$, $T = 14$

M_k \ T	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$1 k_1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$2 k_2	0	1	1	2	2	3	3	4	4	5	5	6	6	7	7
\$5 k_3	0	1	1	2	2	1	2	2	3	3	2	3	3	4	4
\$7 k_4	0														
\$10 k_5	0														

para $k \leftarrow 2$ **até** n **faça**
  para $t \leftarrow 1$ **até** T **faça**
    $m[t, k] = m[t, k-1]$
    se $M_k \leqslant t$ **e** $(m[t - M_k, k] + 1) < m[t, k]$ **então**
      $m[t, k] = m[t - M_k, k] + 1$

Aplicando o Algoritmo no exemplo

- $M = \{1, 2, 5, 7, 10\}$, $n = 5$, $T = 14$

M_k \ T	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$1 k_1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$2 k_2	0	1	1	2	2	3	3	4	4	5	5	6	6	7	7
\$5 k_3	0	1	1	2	2	1	2	2	3	3	2	3	3	4	4
\$7 k_4	0	1	1	2	2	1	2	1	2	2	2	3	2	3	2
\$10 k_5	0														

para $k \leftarrow 2$ **até** n **faça**
  para $t \leftarrow 1$ **até** T **faça**
    $m[t, k] = m[t, k-1]$
    se $M_k \leqslant t$ **e** $(m[t - M_k, k] + 1) < m[t, k]$ **então**
      $m[t, k] = m[t - M_k, k] + 1$

Aplicando o Algoritmo no exemplo

- $M = \{1, 2, 5, 7, 10\}$, $n = 5$, $T = 14$

M_k \ T	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$1 k_1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$2 k_2	0	1	1	2	2	3	3	4	4	5	5	6	6	7	7
\$5 k_3	0	1	1	2	2	1	2	2	3	3	2	3	3	4	4
\$7 k_4	0	1	1	2	2	1	2	1	2	2	2	3	2	3	2
\$10 k_5	0	1	1	2	2	1	2	1	2	2	1	2	2	3	2

para $k \leftarrow 2$ **até** n **faça**
  para $t \leftarrow 1$ **até** T **faça**
    $m[t, k] = m[t, k-1]$
    se $M_k \leqslant t$ **e** $(m[t - M_k, k] + 1) < m[t, k]$ **então**
      $m[t, k] = m[t - M_k, k] + 1$

Complexidade do Algoritmo

- A contagem dos *loops* nos fornece complexidade $O(nT)$.
- Algoritmo pseudo-polinomial, T é valor, não quantidade.
- Na recuperação usamos a matriz $m[n, T]$.

Algoritmo TrocoCerto(T,n,m)

$t \leftarrow T$; $k \leftarrow n$
$Troco \leftarrow \emptyset$
enquanto $m[t, k] > 0$ **faça**
  se $k > 1$ **e** $m[t, k] = m[t, k-1]$ **então**
    $k \leftarrow k - 1$
  senão
    $Troco \leftarrow Troco \cup M_k$
    $t \leftarrow t - M_k$

Recuperando o Troco

- $M = \{1, 2, 5, 7, 10\}$, $n = 5$, $T = 14$ $M = \{\ \}$

M_k \ T	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$1 k_1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$2 k_2	0	1	1	2	2	3	3	4	4	5	5	6	6	7	7
\$5 k_3	0	1	1	2	2	1	2	2	3	3	2	3	3	4	4
\$7 k_4	0	1	1	2	2	1	2	1	2	2	2	3	2	3	2
\$10 k_5	0	1	1	2	2	1	2	1	2	2	1	2	2	3	2

enquanto $m[t, k] > 0$ **faça**
  se $k > 1$ **e** $m[t, k] = m[t, k-1]$ **então**
    $k \leftarrow k - 1$
  senão
    $Troco \leftarrow Troco \cup M_k$
    $t \leftarrow t - M_k$

Recuperando o Troco

- $M = \{1, 2, 5, 7, 10\}$, $n = 5$, $T = 14$ $M = \{\ \}$

M_k \ T	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$1 k_1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$2 k_2	0	1	1	2	2	3	3	4	4	5	5	6	6	7	7
\$5 k_3	0	1	1	2	2	1	2	2	3	3	2	3	3	4	4
\$7 k_4	0	1	1	2	2	1	2	1	2	2	2	3	2	3	2
\$10 k_5	0	1	1	2	2	1	2	1	2	2	1	2	2	3	2

enquanto $m[t, k] > 0$ **faça**
  se $k > 1$ **e** $m[t, k] = m[t, k-1]$ **então**
    $k \leftarrow k - 1$
  senão
    $Troco \leftarrow Troco \cup M_k$
    $t \leftarrow t - M_k$

Recuperando o Troco

- $M = \{1, 2, 5, 7, 10\}$, $n = 5$, $T = 14$ $M = \{\$7\}$

M_k \ T	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$1 k_1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$2 k_2	0	1	1	2	2	3	3	4	4	5	5	6	6	7	7
\$5 k_3	0	1	1	2	2	1	2	2	3	3	2	3	3	4	4
\$7 k_4	0	1	1	2	2	1	2	1	2	2	2	3	2	3	2
\$10 k_5	0	1	1	2	2	1	2	1	2	2	1	2	2	3	2

enquanto $m[t, k] > 0$ **faça**
  se $k > 1$ **e** $m[t, k] = m[t, k-1]$ **então**
    $k \leftarrow k - 1$
  senão
    $Troco \leftarrow Troco \cup M_k$
    $t \leftarrow t - M_k$

Recuperando o Troco

- $M = \{1, 2, 5, 7, 10\}$, $n = 5$, $T = 14$ $M = \{\$7\}$

M_k \ T	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$1 k_1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$2 k_2	0	1	1	2	2	3	3	4	4	5	5	6	6	7	7
\$5 k_3	0	1	1	2	2	1	2	2	3	3	2	3	3	4	4
\$7 k_4	0	1	1	2	2	1	2	1	2	2	2	3	2	3	2
\$10 k_5	0	1	1	2	2	1	2	1	2	2	1	2	2	3	2

enquanto $m[t, k] > 0$ **faça**
  se $k > 1$ **e** $m[t, k] = m[t, k-1]$ **então**
    $k \leftarrow k - 1$
  senão
    $Troco \leftarrow Troco \cup M_k$
    $t \leftarrow t - M_k$

Recuperando o Troco

- $M = \{1, 2, 5, 7, 10\}$, $n = 5$, $T = 14$ $M = \{\$7, \$7\}$

M_k \ T	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$1 k_1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$2 k_2	0	1	1	2	2	3	3	4	4	5	5	6	6	7	7
\$5 k_3	0	1	1	2	2	1	2	2	3	3	2	3	3	4	4
\$7 k_4	0	1	1	2	2	1	2	1	2	2	2	3	2	3	2
\$10 k_5	0	1	1	2	2	1	2	1	2	2	1	2	2	3	2

enquanto $m[t, k] > 0$ **faça**
  se $k > 1$ **e** $m[t, k] = m[t, k-1]$ **então**
    $k \leftarrow k - 1$
  senão
    $Troco \leftarrow Troco \cup M_k$
    $t \leftarrow t - M_k$

Atividades baseadas no CLRS.

1. Ler capítulos 15 (prefácio), 15.2 e 15.3 (pulamos o 15.1, mas se desejar, pode ler).
2. Exercícios: 15.2-1, 15.2-2
3. Resolver a lista 9