Programação Dinâmica

Notas de aula da disciplina IME 04-10823
ALGORITMOS E ESTRUTURAS DE DADOS II

Paulo Eustáquio Duarte Pinto
(pauloedp arroba ime.uerj.br)

novembro/2015

Programação Dinâmica

Partição de Inteiros

Dado **n** inteiro, determinar o número de maneiras de particionamento de **n**.

Exemplo 1: **n = 4** — **5 maneiras distintas:**

4
2 + 2
1 + 1 + 1 + 1
3 + 1
2 + 1 + 1

Exemplo 2: **n = 6** — **11 maneiras distintas:**

6
4 + 2
3 + 3
3 + 1 + 1
2 + 2 + 1 + 1
1 + 1 + 1 + 1 + 1 + 1
5 + 1
4 + 1 + 1
3 + 2 + 1
2 + 2 + 2
2 + 1 + 1 + 1 + 1

Programação Dinâmica

Partição de Inteiros

Formulação recursiva: dado n,

T(n, p) = número de partições onde a maior parcela ≤ p

T(n, p) = 0, se (n < 0) ou (p = 0)
T(n, p) = 1, se (n = 0)
T(n, p) = T(n – p, p) + T(n, p - 1), n > 0

Procura-se obter T(n, n). Pode-se implementar a recursão com "memorização".

Programação Dinâmica

Partição de Inteiros

T(n, p) = 0, se (n < 0) ou (p = 0)
T(n, p) = 1, se (n = 0)
T(n, p) = T(n – p, p) + T(n, p - 1), n > 0
Quer-se obter T(n, n).

A idéia da programação dinâmica é evitar a recursão, de forma análoga à memorização, mas calculando, de forma "bottom-up", todos os subproblemas menores do que o problema a ser solucionado.

Neste caso, é possível usar a idéia da PD!

Programação Dinâmica

Partição de Inteiros

T(n, p) = 0, se (n < 0) ou (p = 0)
T(n, p) = 1, se (n = 0)
T(n, p) = T(n – p, p) + T(n, p - 1), n > 0
Quer-se obter T(n, n).

Para implementar PD, calcular T(n, p) em ordem crescente por n e p, começando por qualquer um dos dois parâmetros.

Programação Dinâmica

Partição de Inteiros

T(n, p) = 0, (n < 0) ou (p = 0)
T(n, p) = 1, (n = 0)
T(n, p) = T(n – p, p) + T(n, p - 1), (n > 0)

Algoritmo 1 (por linha):

```
Para p de 0 a n:  T[0, p] ← 1;  Fp;
Para i de 1 a n:
    T[i, 0] ← 0;
    Para p de 1 a n:
        Se (i ≥ p) Então T[i, p] ← T[i, p-1] + T[i-p, p];
        Senão            T[i, p] ← T[i, p-1];
    Fp;
Fp;
```

Complexidade: $O(n^2)$

Programação Dinâmica

Partição de Inteiros

T(n, p) = 0, (n < 0) ou (p = 0)
T(n, p) = 1, (n = 0)
T(n, p) = T(n – p, p) + T(n, p - 1), (n > 0)

Algoritmo 2 (por coluna):

```
T[0,0] ← 1; Para i de 1 a n: T[i, 0] ← 0; Fp;
Para p de 1 a n:
    Para i de 0 a n:
        Se (i ≥ p) Então T[i, p] ← T[i, p-1] + T[i-p, p];
        Senão            T[i, p] ← T[i, p-1];
    Fp;
Fp;
```

Programação Dinâmica

Partição de Inteiros – Cálculo de T(5, 5)

	0	1	2	3	4	5
0	1	1	1	1	1	*1*
1	0	1	1	1	1	1
2	0	1	2	*2*	2	2
3	0	1	2	3	3	3
4	0	1	3	4	5	5
5	0	1	*3*	5	*6*	7

Programação Dinâmica

Partição de Inteiros – ExS12

Completar a tabela abaixo, para n = 7

	0	1	2	3	4	5
0	1	1	1	1	1	1
1	0	1	1	1	1	1
2	0	1	2	2	2	2
3	0	1	2	3	3	3
4	0	1	3	4	5	5
5	0	1	3	5	6	7

Programação Dinâmica

Moedas

Dados os tipos de moedas de um país, determinar o número de maneiras distintas para dar um troco de valor **n**.

Exemplo:
V = {1,5,10,25,50,100}
m = 6
n = 26

Há 13 maneiras distintas:
25, 1
10, 10, 5, 1
10, 10, 1...1
10, 5, 5, 5, 1
10, 5, 5, 1...1
10, 5, 1...1
10, 1...1
5, 5, 5, 5, 5, 1
5, 5, 5, 5, 1...1
5, 5, 5, 1...1
5, 5, 1...1
5, 1...1
1...1

Programação Dinâmica

Moedas

Formulação recursiva: dados m, n

T(p, n) = formas distintas de dar um troco n, usando os p tipos iniciais de moedas, V[1]...V[p]

T(p, n) = 0, (n < 0)
T(p, n) = 1, (n = 0)
T(p, n) = Σ T(i, n – V[i]), (n > 0), 1 ≤ i ≤ p

A solução do problema é obter T(m, n).

Programação Dinâmica

Moedas

Exemplo:

V = {1,5,10,25,50,100}
m = 6
n = 26

Há 13 maneiras distintas:

T(6, 26-100)= T(6,-74)	→	---
T(5, 26 - 50)= T(5,-24)	→	---
T(4, 26 - 25)= T(4, 1)	→	25, 1
T(3, 26 - 10)= T(3, 16)	→	10, 10, 5, 1
		10, 10, 1...1
		10, 5, 5, 5, 1
		10, 5, 5, 1...1
		10, 5, 1...1
		10, 1...1
T(2, 26 - 5)= T(2, 21)	→	5, 5, 5, 5, 5, 1
		5, 5, 5, 5, 1..1
		5, 5, 5, 1..1
		5, 5, 1..1
		5, 1..1
T(1, 26 - 1) = T(1, 25)	→	1,1...1

Programação Dinâmica

Moedas

Exemplo:

V = {1, 5, 10, 25, 50, 100} m = 6, n = 20

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	2	2	2	2	2	3	3	3	3	3	4	4	4	4	4	5
3	1	1	1	1	1	2	2	2	2	2	4	4	4	4	4	6	6	6	6	6	9
4	1	1	1	1	1	2	2	2	2	2	4	4	4	4	4	6	6	6	6	6	9
5	1	1	1	1	1	2	2	2	2	2	4	4	4	4	4	6	6	6	6	6	9
6	1	1	1	1	1	2	2	2	2	2	4	4	4	4	4	6	6	6	6	6	9

Programação Dinâmica

Moedas

A solução por PD consiste em preencher a tabela m x n por ordem crescente do tamanho dos subproblemas, sem necessidade de recursão. Há duas formas de fazer isso:

Ex. da 1ª forma:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	2	2	2	2	2	3	3	3	3	3	4	4	4	4	4	5
3	1	1	1	1	1	2	2	2	2	2	4	4	4	4	4	6	6	6	6	6	9

Programação Dinâmica

Moedas

V = {1, 5, 10, 25, 50, 100}

Exercício: Completar a tabela abaixo para n = 26

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	2	2	2	2	2	3	3	3	3	3	4	4	4	4	4	5
3	1	1	1	1	1	2	2	2	2	2	4	4	4	4	4	6	6	6	6	6	9
4	1	1	1	1	1	2	2	2	2	2	4	4	4	4	4	6	6	6	6	6	9
5	1	1	1	1	1	2	2	2	2	2	4	4	4	4	4	6	6	6	6	6	9
6	1	1	1	1	1	2	2	2	2	2	4	4	4	4	4	6	6	6	6	6	9

Programação Dinâmica

Moedas

V = {1, 3, 5}

ExS13: Preencher a tabala para n = 10

	0	1	2	3	4	5	6	7	8	9	10
1											
2											
3											

Programação Dinâmica

Moedas

T(p, n) = 0, (n < 0) ou (p = 0)
T(p, n) = 1, (n = 0)
$T(p, n) = \Sigma_{1 \le i \le p} T(i, n - V[i])$, (n > 0)

Algoritmo, com preenchimento linha x coluna:

```
Moedas;
    Para p de 1 a  m:
        T[p, 0] ← 1;
        Para i de 1 a n:
            tot ←  0;
            Para k de 1 a p:
                Se ((i - V[k]) ≥ 0) Então
                    tot ← tot + T[k, i - V[k]];
            Fp;
            T[p, i] ← tot;
        Fp;
    Fp;
Fim;
```

Complexidade: $O(n.m^2)$

Programação Dinâmica

Moedas PD x Memorização

V = {1, 5, 10, 25, 50, 100}

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	-1
2	1	-1	-1	-1	-1	2	-1	-1	-1	-1	3	-1	-1	-1	-1	4	-1	-1	-1	-1	-1
3	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	4	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
4	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
5	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	9

Programação Dinâmica

Moedas

Outra formulação recursiva: dados m, n

T(p, n) = formas distintas de dar um troco n, usando os p tipos iniciais de moedas, V[1]...V[p]

T(p, n) = 0, (n < 0) ou (p = 0)
T(p, n) = 1, (n = 0)
T(p, n) = T(p, n – V[p]) + T(p-1, n), (n > 0)

A solução do problema é obter T(m, n).

Programação Dinâmica

Moedas

Exemplo:

V = {1, 5, 10, 25, 50, 100} m = 6, n = 20

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	2	2	2	2	2	3	3	3	3	3	4	4	4	4	4	5
3	1	1	1	1	1	2	2	2	2	2	4	4	4	4	4	6	6	6	6	6	9
4	1	1	1	1	1	2	2	2	2	2	4	4	4	4	4	6	6	6	6	6	9
5	1	1	1	1	1	2	2	2	2	2	4	4	4	4	4	6	6	6	6	6	9
6	1	1	1	1	1	2	2	2	2	2	4	4	4	4	4	6	6	6	6	6	9

Programação Dinâmica

Mochila (0/1)

Dada uma mochila com capacidade M e t ítens com peso w_i cada, verificar se existe uma combinação de itens que preencha exatamente a mochila.

Exemplo:

Dado o conjunto de ítens {7, 3, 5, 9, 15}, é possível preencher exatamente mochilas com capacidades 25 e 27, mas não é possível preencher mochila com capacidade 26.

Programação Dinâmica

Mochila (0/1)

Define-se K(q, n) = x = indicador de solução quando se usa os q itens iniciais numa mochila de capacidade n.

x =-1 não há solução,
cc indica o menor índice de ítem que que completa a mochila.

Formulação recursiva:

K(q, n) = 0, se n = 0,
K(q, n) = K(q-1, n), se K(q-1,n) ≠ -1, $0 \le n \le M$; $1 \le q \le t$;
K(q, n) = q, se K(q-1, n-p_q) ≠ -1, $0 \le n \le M$; $1 \le q \le t$;
K(q, n) = -1, nos demais casos

Programação Dinâmica Mochila (0/1)

K(q, n) = 0, se n = 0,
K(q, n) = K(q-1, n), se K(q-1,n) ≠ -1, $0 \le n \le M$; $1 \le q \le t$;
K(q, n) = q, se K(q-1, n-p_q) ≠ -1, $0 \le n \le M$; $1 \le q \le t$;
K(q, n) = -1, nos demais casos

Algoritmo:

```
K[0, 0] ← 0;  Para j de 1 a M:   K[0, j] ← -1;   Fp;
Para i de 1 a t:
        Para j de 0 a M:
                Se (K[i-1, j] ≠ -1) Então
                        K[i, j] ← K[i-1, j]
                Senão  Se (j ≥ P[i]) e (K[i-1, j-P[i]] ≠ -1) Então
                        K[i, j] ← i;
                Senão
                        K[i, j] ← -1;
        Fp;
Fp;
```

Complexidade: O(t.M)

Programação Dinâmica

Mochila (0/1)

Exemplo: P= {7, 3, 5, 9, 15} M = 20

Obs: -1 não representado

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	0																				
1 (7)	0							1													
2 (3)	0			2				1			2										
3 (5)	0			2		3		1	3		2		3			3					
4 (9)	0			2		3		1	3	4	2		3		4	3	4	4		4	
5 (15)	0			2		3		1	3	4	2		3		4	3	4	4	5	4	5

Programação Dinâmica

Mochila (0/1)

P = {7, 3, 5, 9, 15}

ExS14: Completar a tabela abaixo para M = 27

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	0																				
1 (7)	0							1													
2 (3)	0			2				1			2										
3 (5)	0			2		3		1	3		2		3			3					
4 (9)	0			2		3		1	3	4	2		3		4	3	4	4		4	
5 (15)	0			2		3		1	3	4	2		3		4	3	4	4	5	4	5

Programação Dinâmica

Mochila (0/1) – Apresentação de uma solução

Solução:

```
    j ← M;
    Enquanto (j ≠ 0):
        Escrever (P[K[t, j]]):
        j ← j - P[K[t,j]];
    Fe;
Fim;
```

Para M = 17, a saída seria: 9 5 3

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	0																				
1 (7)	0							1													
2 (3)	0			2				1			2										
3 (5)	0			2		3		1	3		2		3			3					
4 (9)	0			2		3		1	3	4	2		3		4	3	4	4		4	
5 (15)	0			2		3		1	3	4	2		3		4	3	4	4	5	4	5

Programação Dinâmica

Mochila (0/1)

P = {7, 3, 5, 9, 15}

Exercício: Encontrar a solução para M = 19

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	0																				
1 (7)	0							1													
2 (3)	0			2				1			2										
3 (5)	0			2		3		1	3		2		3			3					
4 (9)	0			2		3		1	3	4	2		3		4	3	4	4		4	
5 (15)	0			2		3		1	3	4	2		3		4	3	4	4	5	4	5

Programação Dinâmica

Mochila (0/1) – outras versões

a) usar dois valores booleanos em cada célula, o primeiro indicando se há solução e o segundo se o ítem atual é o de menor índice para a solução

Exemplo: P = {7, 3, 5, 9, 15} M = 20

Obs: célula em branco com (F,F)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	VF																				
1 (7)	VF							VV													
2 (3)	VF			VV				VF			VV										
3 (5)	VF			VF		VV		VF	VV		VF		VV			VV					
4 (9)	VF			VF		VF		VF	VF	VV	VF		VF		VV	VF	VV	VV		VV	
5 (15)	VF			VF		VF		VF	VF	VF	VF		VF		VF	VF	VF	VF	VV	VF	VV

Programação Dinâmica

Mochila (0/1) – outras versões

b) usar um vetor ao invés de uma matriz

Exemplo: P = {7, 3, 5, 9, 15} M = 20

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	-1	-1	2	-1	3	-1	1	3	4	2	-1	3	-1	4	3	4	4	5	4	5

Algoritmo:

```
K[0] ← 0;   Para j de 1 a M:   K[j] ← -1;   Fp;
Para i de 1 a t:
    Para j de M descendo a P[i]:
        Se (K[j] = -1) e (K[j-P[i]] ≥ 0) Então
            K[j] ← i;
    Fp;
Fp;
```

Programação Dinâmica

Mochila (0/1) – outras versões

b) usar um vetor ao invés de uma matriz

Exemplo: P = {7, 3, 5, 9, 15} M = 20

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	-1	-1	2	-1	3	-1	1	3	4	2	-1	3	-1	4	3	4	4	5	4	5

Exercício: mostrar a situação do vetor acima após o processamento de cada ítem, a partir da situação inicial:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Programação Dinâmica

Mochila (0/1) – outras versões

ExS15: mostrar a situação do vetor para o problema mochila para M = 20, e 5 ítens: {2, 3, 3, 5, 6}:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0																				

Programação Dinâmica

Mochila (0/1) – outras versões

b) usar um vetor ao invés de uma matriz

Problema Ferry Loading (Val 12601)

Tem-se um "ferry" de comprimento lf, duas linhas para carros e uma fila dada de carros, com seus comprimentos lc_i. Quer-se saber quantos carros podem ser carregados, obedecendo-se a fila.

Dados: n carros de comprimentos lc_i dados e o ferry com comprimento lf.

lf = 9

Fila: 2, 5, 4, 3, 3, 3, 1, 2

Sol: 5 (2/1, 5/2, 4/2, 3/1, 3/1)

Programação Dinâmica

Mochila (0/1) – outras versões

b) usar um vetor ao invés de uma matriz

Problema Ferry Loading (Val 12601)

Tem-se um "ferry" de comprimento lf, duas linhas para carros e uma fila dada de carros, com seus comprimentos lc_i. Quer-se saber quantos carros podem ser carregados, obedecendo a fila.

lf = 9, lc = (2, 5, 4, 3, 3, 3, 1, 2)

	0	1	2	3	4	5	6	7	8	9	tot	SIT
0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	
1	0	-1	1	-1	-1	-1	-1	-1	-1	-1	2	V
2	0	-1	1	-1	-1	2	-1	2	-1	-1	7	V
3	0	-1	1	-1	3	2	3	2	-1	3	11	V
4	0	-1	1	-1	3	2	3	2	4	3	14	V
5	0	-1	1	-1	3	2	3	2	4	3	17	V
6	0	-1	1	-1	3	2	3	2	4	3	20	F

Programação Dinâmica

Mochila (0/1) – outras versões

Problema Ferry Loading (Val 12601) - Escolha dos carros

Algoritmo:

```
K[*] ← -1;    K[0] ← 0;  F[*] ← 0;
i ← 1;   lotado ← F;   tot ← 0;
Enquanto (i ≤ t) E (não lotado):
    tot ← tot + lc[i];   lotado ← V;
    Para j  descendo de lf a lc[i]:
        Se (K[j-lc[i]] > -1) E ((tot-j) ≤ lf) Então
            lotado ← F;   F[i] ← 1;
            Se (K[j] = -1) Então k[j] ← i;
    Fp;
    i ← i+1;
Fe;
Fim;
```

Programação Dinâmica

Mochila (0/1) – outras versões

Problema Ferry Loading (Val 12601) - Determinação das filas

Algoritmo:

```
i ← lf;  Enquanto (K[i] = -1): i ← i-1;  Fe;
Enquanto (i > 0):
    j ← K[i];  F[j] ← 2;   i ← i-lc[j];
Fe;
Fim;
```

Vetor K

	0	1	2	3	4	5	6	7	8	9	SIT
	0	-1	1	-1	3	2	3	2	4	3	F

Vetor F

i	1	2	3	4	5	6	7	8
9	1	1	2	1	1	0	0	0
5	1	2	2	1	1	0	0	0
0	1	2	2	1	1	0	0	0

Programação Dinâmica

Mochila (0/1) – outras versões

c) contar o número de soluções

Exemplo: P = {2, 3, 2, 2, 5} M = 15

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1															
1	1		1													
2	1		1	1		1										
3	1		2	1	1	2		1								
4	1		3	1	3	3	1	3		1						
5	1		3	1	3	4	1	6	1	4	3	1	3		1	

Programação Dinâmica

Mochila (0/1) – outras versões

c) contar o número de soluções

Algoritmo:

```
K[0] ← 1;   Para j de 1 a M:    K[j] ← 0;   Fp;
Para i de 1 a t:
   Para j decrescendo de M a P[i]:
       K[j] = K[j] + K[j-P[i]];
   Fp;
Fp;
```

Programação Dinâmica

Mochila (0/1) – outras versões

ExS16: mostrar o preenchimento do vetor para contar o número de soluções para M = 20, e 5 ítens:
{2, 3, 3, 5,6}:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1																					

Programação Dinâmica

Mochila (0/1) – outras versões

d) contar o número de itens na solução

Exemplo: P = {7, 3, 5, 9, 15} M = 20

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	-1			2		3		1		4						5					
2	-1								3		2		3		4		4		5		5
3	-1															3		4		4	
4																					
5																					

Programação Dinâmica

Mochila (0/1) – outras versões

d) contar o número de itens na solução

Exemplo: P = {7, 3, 5, 9, 15} M = 20

Algoritmo:

```
K[0,0] ← 0;   Para j de 1 a M:    K[0,j] ← -1;   Fp;
Para i de 1 a t:
    Para j de 0 a M:    K[i,j] ← -1;   Fp;
    Para r decrescendo de i a 1:
        Para j de P[i] a M:
            Se (K[r, j] = -1) e (K[r-1,j-P[i]] ≥ 0) Então
                K[r, j] ← i;
        Fp;
    Fp;
Fp;
```

Programação Dinâmica

Mochila (0/1) – outras versões

d) contar o número de itens na solução

Problema Tug of War (Val 10032)

Tem-se n competidores, e são dados os pesos de cada um. Quer-se dividir as pessoas em dois times tal que o número de competidores difira no máximo em 1 e a soma dos pesos deve ser mínima. Indicar os pesos de cada grupo.

Dados:
n = 9
Pesos: 100, 65, 70, 82, 95, 71, 71, 66, 84
Sol: 350 354 (1, 2, 2, 2, 1, 1, 2, 2, 1)

Programação Dinâmica

Mochila (0/1) – outras versões

d) contar o número de itens na solução

Problema Joys of Farming (Val 11331)

Tem-se n casas, cada uma com 2 quartos, em cada quarto um número variável de camas. Quer-se distribuir m moças e r rapazes pelas casas, tal que em cada quarto só tenha ou moças ou rapazes e não podendo os dois quartos de uma mesma casa serem ocupados pelo mesmo sexo. É possível fazer a distribuição?

Dados:
n = 5, m = 32, r = 40
Quartos: 3/8, 5/6, 4/7, 6/10, 11/13
Sol: S (moças: 3, 6, 4, 6, 13
rapazes: 8, 5, 7, 10, 10)

Programação Dinâmica

Mochila (0/1) -

ExS17: mostrar o preenchimento da matriz para mostrar o número de ítens na solução M = 20, e 5 ítens:
{2, 3, 3, 5,6}:

Programação Dinâmica

Mochila (0/1) – outras versões

e) Mochila com peso e valor: a cada ítem, além do peso(p) é associado um valor(v). O objetivo passa a ser determinar o valor máximo que pode comportar a mochila

Ex: P = {(7,3), (3,7), (5,11), (9,12), (15,15)} M = 20

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	-1	-1	2	-1	3	-1	1	3	4	2	-1	4	-1	4	3	4	4	5	4	5
0	0	0	7	0	11	0	3	18	12	10	0	19	0	23	21	15	30	22	22	26

Programação Dinâmica

Mochila (0/1) – outras versões – Peso e Valor

Ex: P = {(7,3), (3,7), (5,11), (9,12), (15,15)} M = 20

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0							1													
0							3													
0			2				1			2										
0			7				3			10										
0			2		3		1	3		2		3			3					
0			7		11		3	18		10		14			21					
0			2		3		1	3	4	2		4		4	3	4	4		4	
0			7		11		3	18	12	10		19		23	21	15	30		22	
0			2		3		1	3	4	2		4		4	3	4	4	5	4	5
0			7		11		3	18	12	10		19		23	21	15	30	22	22	26

Programação Dinâmica

Mochila (0/1) – outras versões – Peso e Valor

e) Mochila com peso e valor: a cada ítem, além do peso(W) é associado um valor(V). O objetivo passa a ser determinar o valor máximo que pode comportar a mochila

Ex: P, V = {(7,10), (3,6), (5,11), (9,12), (15,15)} M = 20

Algoritmo:

```
K[0].me ← 0;  K[0].vm ← 0;
Para j de 1 a M: K[j].me ← -1;  K[j].vm ← 0;  Fp;
Para i de 1 a t:
      Para j descendo de  M a P[i]:
          Se (K[j-P[i]].me ≥ 0) e (K[j].vm < (K[j-P[i]].vm+V[i])) Então
                K[j].me ← i;   K[j].vm ← K[j-P[i]].vm+V[i];
      Fp;
Fp;
```

Obs: K[j].me = menor índice que obtém o valor máximo
K[j].vm = valor máximo

Programação Dinâmica

Mochila (0/1)– outras versões - Peso e Valor

ExS18: mostrar a situação do vetor para os seguintes ítens, M = 10.

Ex: P, V = {(2,2), (3,4), (2,4), (3,7)} M = 10

Programação Dinâmica

Mochila (0/1)– outras versões - Peso e Valor

f) Mochila múltiplos itens: existem infinitos objetos de cada tipo de ítem

Ex: P = {7, 3, 5, 9, 15} M = 20

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	-1	-1	2	-1	3	2	1	3	2	2	3	2	2	1	2	2	2	2	2	2

Programação Dinâmica

Mochila – outras versões

f) Mochila múltiplos itens: existem infinitos objetos de cada tipo de ítem

Ex: P = {7, 3, 5, 9, 15} M = 20

Algoritmo:

```
K[0] ← 0;   Para j de 1 a M: K[j] ← -1;  Fp;
Para i de 1 a t:
      Para j de P[i] a M:
            Se (K[j-P[i]] ≥ 0) e (K[j] = -1) Então
                  K[j] ← i;
      Fp;
Fp;
```

Programação Dinâmica

Mochila – outras versões

g) Mochila com múltiplos itens, cada um com peso e valor: a cada ítem, além do peso(p) é associado um valor(v). O objetivo passa a ser determinar o valor máximo que pode comportar a mochila

Ex: W = {(7,10), (3,6), (5,11), (9,12), (15,15)} M = 20

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	-1	-1	2	-1	3	2	1	3	2	3	3	2	3	3	3	3	3	3	3	3
0	0	0	6	0	11	12	10	17	18	22	23	24	28	29	33	34	35	39	40	44

0	-1	-1	2	-1	3	-1	1	3	4	2	-1	3	-1	4	3	4	4	5	4	5
0	0	0	6	0	11	0	10	17	12	16	0	21	0	23	27	22	29	21	28	26

(valores da versão 0/1)

Programação Dinâmica

Mochila – outras versões

g) Mochila com múltiplos itens, cada um com peso e valor: a cada ítem, além do peso(W) é associado um valor(V). O objetivo passa a ser determinar o valor máximo que pode comportar a mochila

Ex: P, V = {(7,10), (3,6), (5,11), (9,12), (15,15)} M = 20

Algoritmo:

```
K[0].me ← 0;  K[0].vm ← 0;
Para j de 1 a M: K[j].me ← -1;  K[j].vm ← 0;  Fp;
Para i de 1 a t:
      Para j de P[i] a M:
            Se (K[j-P[i]].me ≥ 0) e (K[j].vm < (K[j-P[i]].vm+V[i]) Então
                  K[j].me ← i;   K[j].vm ← K[j-P[i]].vm+V[i];
      Fp;
Fp;
```

Programação Dinâmica

Mochila (0/1)– Ítens múltiplos

ExS19: mostrar a situação do vetor para os seguintes ítens, M = 10,

Ex: P, V = {(2,2), (3,4), (2,4), (3,7)} M = 10

a) Sem considerar o valor

b) Considerando o valor

Programação Dinâmica

Mochila – outras versões

h) Mochila com ítens fracionáveis: cada ítem pode ser fracionado. Será visto adiante (Guloso)

i) Mochila mista: parte dos itens não fracionáveis e parte fracionáveis. Será visto adiante (PD+Guloso)

Programação Dinâmica

Mochila – PD x Backtracking

-Muitos itens pequenos

S = {3, 3, 4, 7, 9, 15, 16, 55, 57, 58, 100, 111, 115, 125, 150, 201, 337, 442, 503, 712, 1111}

é melhor PD

-Poucos itens grandes

S = {1.023, 19.992, 220.043, 401.327, 899.206, 1.203.427, 1.234.567.806, 2.997.200.025}

é melhor BK

Programação Dinâmica

Problemas com paradigma da "diagonal"

Alguns problemas com solução por PD requerem que se examine todos os subproblemas relativos a intervalos consecutivos de um vetor, ou seja intervalos (1,1), (1,2)...(1,n),(2,3).....(n-1,n)(n,n).
Neste caso, o resultado ótimo para cada intervalo (i,j), com j ≥ i é guardado na célula (i,j) de uma matriz $M_{n \times n}$, que geralmente é preenchida por diagonais, onde cada diagonal representa dada diferença de índices j-i, dos diversos intervalos. Só é preenchido o "triângulo" superior da matriz,
Problemas apresentados são: Produto de Matrizes e Jogo da Soma.

1	2	3	4

	1	2	3	4
1				
2	x			
3	x	x		
4	x	x	x	

Programação Dinâmica

Produto de Matrizes

Dada uma sequência de matrizes que devem ser multiplicadas, determinar a ordem ótima de multiplicação (aquela que requer o menor número de operações), considerando que o produto é associativo.

Exemplo:

M_1 (5 x 20) M_2(20 x 50) M_3(50 x 5) M_4(5 x 100)
O produto pode ser feito de inúmeras maneiras, dentre as quais:

$((M_1 \times M_2) \times (M_3 \times M_4))$

$(M_1 \times ((M_2 \times M_3) \times M_4))$

Programação Dinâmica

Produto de Matrizes – Associatividade

$M_1(a \times b) \times M_2(b \times c) \times M_3(c \times d)$

a) $((M_1 \times M_2) \times M_3) = M_4(a \times d)$, $(M_1 \times M_2) = M_5\ (a \times c)$

$$M_{4ij} = \Sigma_{1 \le k \le c} M_{5ik} \times M_{3kj} = \Sigma_{1 \le k \le c}(\Sigma_{1 \le t \le b} M_{1it} \times M_{2tk}) \times M_{3kj} = \Sigma_{1 \le k \le c} \Sigma_{1 \le t \le b} M_{1it} \times M_{2tk} \times M_{3kj}$$

b) $(M_1 \times (M_2 \times M_3)) = M_4(a \times d)$, $(M_2 \times M_3) = M_6\ (b \times d)$

$$M_{4ij} = \Sigma_{1 \le t \le b} M_{1it} \times M_{6tj} = \Sigma_{1 \le t \le b} M_{1it} (\Sigma_{1 \le k \le c} M_{2itk} \times M_{3kj}) = \Sigma_{1 \le k \le c} \Sigma_{1 \le t \le b} M_{1it} \times M_{2tk} \times M_{3kj}$$

Programação Dinâmica

Produto de Matrizes – Quantidade de operações

M_1 (a x b) x M_2(b x c) ⇒ a x b x c produtos
a x (b-1) x c somas

Produto de Matrizes – Mais de duas matrizes

M_1 (5 x 20) M_2(20 x 50) M_3(50 x 5) M_4(5 x 100)

a) $((M_1 \times M_2) \times (M_3 \times M_4))$
5 x 20 x 50 + 50 x 5 x 100 + 5 x 50 x 100 =
= 55000

b) $(M_1 \times ((M_2 \times M_3) \times M_4))$
20 x 50 x 5 + 20 x 5 x 100 + 5 x 20 x 100 =
= 25000

Programação Dinâmica

Produto de Matrizes – Formulação recursiva

Vetor de dimensões: M_i $(r_{i-1} \times r_i)$

0	1	2	...	...	n
r_0	r_1	r_2	...	...	r_n

T[i, j] = núm. mínimo de operações p/ obter $M_{i....}M_j$

$T[i, j] = \min_{i \le k < j}\{T[i, k]+T[k+1, j] + r_{i-1} \times r_k \times r_j\}$

T[i, i] = 0

Programação Dinâmica

Produto de Matrizes – Implementação com PD

	M_1	M_2	M_3	M_4
0	1	2	3	4
5	20	50	5	100

A idéia é calcular os produtos ótimos $M_{i....}M_j$ em ordem crescente da diferença j – i:
T[1,1], T[2,2], T[3,3], T[4,4]
T[1,2], T[2,3], T[3,4]
T[1,3], T[2,4]
T[1,4], a solução buscada!

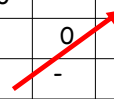
Programação Dinâmica

Produto de Matrizes – Implementação com PD

M_1	M_2	M_3	M_4	
5	20	50	5	100

j – i = 0

	1	2	3	4
1	0			
2	-	0		
3	-	-	0	
4	-	-	-	0



Programação Dinâmica

Produto de Matrizes – Implementação com PD

M_1	M_2	M_3	M_4	
5	20	50	5	100

j – i = 1

	1	2	3	4
1	0	5.000		
2	-	0	5.000	
3	-	-	0	25.000
4	-	-	-	0

Programação Dinâmica

Produto de Matrizes – Implementação com PD

M_1	M_2	M_3	M_4	
5	20	50	5	100

j – i = 2

	1	2	3	4
1	0	5.000	5.500	
2	-	0	5.000	
3	-	-	0	25.000
4	-	-	-	0

T[1,3]

k = 1
$T[1,1]+T[2,3]+r_0.r_1.r_3$ =
0 + 5000+5.20.5 =
5.500

k = 2
$T[1,2]+T[3,3]+r_0.r_2.r_3$ =
0 + 5000+5.50.5 =
6.250

Programação Dinâmica

Produto de Matrizes – Implementação com PD

M_1	M_2	M_3	M_4	
5	20	50	5	100

j – i = 2

	1	2	3	4
1	0	5.000	5.500	
2	-	0	5.000	15.000
3	-	-	0	25.000
4	-	-	-	0

T[2,4]

k = 2
$T[2,2]+T[3,4]+r_1.r_2.r_4$ =
0 + 25000+20.50.100 =
125.000

k = 3
$T[2,3]+T[4,4]+r_1.r_3.r_4$ =
5000 + 0+20.5.100 =
15.000

Programação Dinâmica

Produto de Matrizes – PD - j – i = 3

M_1	M_2	M_3	M_4	
5	20	50	5	100

	1	2	3	4
1	0	5.000	5.500	8.000
2	-	0	5.000	15.000
3	-	-	0	25.000
4	-	-	-	0

T[1,4]

k = 1
$T[1,1]+T[2,4]+r_0.r_1.r_4$ =
0 + 15000+5.20.100 =
25.000

k = 2
$T[1,2]+T[3,4]+r_0.r_2.r_4$ =
5000 + 25000+5.50.100
= 55.000

k = 3
$T[1,3]+T[4,4]+r_0.r_3.r_4$ =
5500 + 0+5.5.100 =
8.000

Programação Dinâmica

Produto de Matrizes – PD - Solução

Custo

	1	2	3	4
1	0	5.000	5.500	8.000
2	-	0	5.000	15.000
3	-	-	0	25.000
4	-	-	-	0

Melhor k

	1	2	3	4
1	0	1	1	3
2	-	0	2	3
3	-	-	0	3
4	-	-	-	0

Multiplicação ótima: $((M_1 \times (M_2 \times M_3)) \times M_4)$

Programação Dinâmica Produto Matrizes

$T[i, j] = \min\{T[i, k]+T[k+1,j] + r_{i-1} \times r_k \times r_j\} \quad i \leq k < j$
$T[i, i] = 0$
Quer-se encontrar $T[1, n]$.

Algoritmo:

```
Para k de 1  a  n:    T[k, k]  ← 0;    Fp;
Para d de 1  a  n – 1:
     Para i de 1  n – d:
       j  ←  i  +  d;      T[i, j] ← ∞;
       Para  k de i até j – 1:
          Se  ((T[i, k] + T[k+1, j] + r[i-1].r[k].r[j])  <  T[i, j])  Então
               T[i, j]  ←  T[i, k]  +  T[k+1, j]  + r[i-1].r[k].r[j];
               MK[i, j]  ←  k;
       Fp;
     Fp;
Fp;
```

Complexidade: $O(n^3)$

Programação Dinâmica

Produto de Matrizes

M_1	M_2	M_3	M_4	
5	20	50	5	100

ExS20: Calcular a pior maneira de multiplicar as matrizes acima.

Programação Dinâmica Produto Matrizes

Impressão da expressão: baseada em uma recorrência sobre a matriz MK

Algoritmo:

```
Expressao(i,j)  { retorna String }
       Se (i = j) Então
              Retornar 'M' + i;
       Senão
              Retornar '(' + Expressao(i,MK[i,j])+ 'x' +
                             Expressao(MK[i,j]+1,j) + ')';
Fim;
```

Programação Dinâmica

Jogo da Soma

Dado um vetor V contendo inteiros, determinar ao melhor resultado para o seguinte jogo: os jogadores retiram, em turnos, quantos elementos quiserem do vetor, desde que seja de uma das pontas.

A solução é maximizar a diferença entre os dois jogadores, levando em conta que os dois jogam de maneira ótima.

Para o vetor abaixo, a maior diferença possível é 5: o primeiro retira 1 e 1; o segundo retira 1, o primeiro retira -1 e o segundo -5, com a diferença (1+1-1)-(1-5) = 5.

1	2	3	4	5
-1	-5	1	1	1

Programação Dinâmica

Jogo da Soma - Recorrência

$M(i,i) = V(i)$
$M(i,j) = \max(Ac(i,j),$
$\quad Ac(i,k) - M(k+1,j),\ i \leq k < j,$
$\quad Ac(k,j) - M(i, k-1),\ i< k \leq j,$
$\quad)$

O primeiro termo da maximização corresponde à retirada de todos os números i a j; o segundo, às retiradas da ponta esquerda e o trceiro, às retiradas da ponta direita do vetor.

Programação Dinâmica

Jogo da Soma - Algoritmo

```
Para k de 1  a  n:    T[k, k]  ← V[k];    Fp;
VA[0] ← 0; Para k de 1  a  n:    VA[k] ← VA[k-1]+V[k];    Fp;
Para d de 1  a  n – 1:
     Para i de 1  n – d:
        j  ←  i  +  d;      T[i, j] ← VA[j]-VA[i-1]; MI[i, j]  ← (i, j);
        Para  k de i até j – 1:
           Se  ((T[i, j] < (VA[k]-VA[i-1]-T[k+1,j]))  Então
               T[i, j] ← VA[k]-VA[i-1]-T[k+1,j];
               MI[i, j]  ← (i, k);
        Fp;
        Para  k de i+1 até j:
           Se  ((T[i, j] < (VA[j]-VA[k-1]-T[i, k-1]))  Então
               T[i, j] ← VA[j]-VA[k-1]-T[i,k-1];
               MI[i, j]  ← (k, j);
         Fp;
     Fp;
Fp;
```

Programação Dinâmica

Jogo da Soma - Exemplo:

V

1	2	3	4	5
-1	-5	1	1	1

	1	2	3	4	5
1	-1				
2	x	-5			
3	x	x	1		
4	x	x	x	1	
5	x	x	x	x	1

Diagonal de diferença de índices=0

Programação Dinâmica

Jogo da Soma - Exemplo:

V

1	2	3	4	5
-1	-5	1	1	1

Diagonal de diferença de índices=1

	1	2	3	4	5
1	-1	4			
2	x	-5	6		
3	x	x	1	2	
4	x	x	x	1	2
5	x	x	x	x	1

M(1,2) = max(-6,
-1-(-5),
-5-(-1)) =4
M(2,3) = max(-4,
-5-(1),
1-(-5)) =6
M(3,4) = max(2,
1-(1),
1-(1)) =2
M(4,5) = max(2,
1-(1),
1-(1)) =2

Programação Dinâmica

Jogo da Soma - Exemplo:

V

1	2	3	4	5
-1	-5	1	1	1

Diagonal de diferença de índices=2

	1	2	3	4	5
1	-1	4	-3		
2	x	-5	6	7	
3	x	x	1	2	3
4	x	x	x	1	2
5	x	x	x	x	1

M(1,3) = max(-5,
-1-(6), -6-(1),
1-(4), -4-(-1) =-3
M(2,4) = max(-3,
-5-(2), -4-(1),
1-(6), 2-(-5)) =7
M(3,5) = max(3,
1-(2), 2-(1),
1-(2), 2-(1)) =3

Programação Dinâmica

Jogo da Soma - Exemplo:

V

1	2	3	4	5
-1	-5	1	1	1

Diagonal de diferença de índices=3

	1	2	3	4	5
1	-1	4	-3	4	
2	x	-5	6	7	8
3	x	x	1	2	3
4	x	x	x	1	2
5	x	x	x	x	1

M(1,4) = max(-4,
-1-(7), -6-(2),-5-(1),
1-(-3),2-(4), -3-(-1))=4
M(2,5) = max(-2,
-5-(3), -4-(2), -3-(1),
1-(7),2-(6),3-(-5)) =8

Programação Dinâmica

Jogo da Soma - Exemplo:

V

1	2	3	4	5
-1	-5	1	1	1

	1	2	3	4	5
1	-1	4	-3	4	5
2	x	-5	6	7	8
3	x	x	1	2	3
4	x	x	x	1	2
5	x	x	x	x	1

Diagonal de diferença de índices=4

M(1,5) = max(-3,
-1-(8), -6-(3), -5-(2),-4-(1),
1-(4), 2-(-3), 3-(4), 02-(-1))
=5

Programação Dinâmica

Jogo da Soma - Exemplo:

V

1	2	3	4	5
-1	-5	1	1	1

Maior diferença

	1	2	3	4	5
1	-1	4	-3	4	5
2	x	-5	6	7	8
3	x	x	1	2	3
4	x	x	x	1	2
5	x	x	x	x	1

Melhor intervalo

	1	2	3	4	5
1	1-1	1-1	2-4	4-4	4-5
2	x	2-2	3-3	3-4	3-5
3	x	x	3-3	3-4	3-5
4	x	x	x	4-4	4-5
5	x	x	x	x	5-5

Programação Dinâmica

Distância de Edição

Dados dois strings A e B, quer-se determinar a menor sequência de operações p/ transformar A em B.
Os tipos de operação são:

- -inserção de um caracter
- -deleção de um caractér
- -substituição de um caractér

Exemplo: ERRO transforma-se em ACERTO mediante 3 operações:

-inserção do A	AERRO
-inserção do C	ACERRO
-substituição do R	ACERTO

Programação Dinâmica

Distância de Edição - Formulação recursiva

Dados os substrings A_i (primeiros i caracteres) e B_j

D(i, j) = distância de edição entre A_i e B_j =

$D(i-1, j-1)$, se $a_i = b_j$
$\min(D(i-1, j), D(i, j-1), D(i-1, j-1)) + 1$, se $a_i \neq b_j$

Deleção de a_i | Inserção de b_j | Substituição de a_i por b_j

Programação Dinâmica

Cálculo da Distância de Edição

	0	1 A	2 C	3 E	4 R	5 T	6 O
0	0	1	2	3	4	5	6
1 E	1	1	2	2	3	4	5
2 R	2	2	2	3	2	3	4
3 R	3	3	3	3	3	3	4
4 O	4	4	4	4	4	4	3

Programação Dinâmica

Distância de Edição

$D(i, j) = D(i-1, j-1) = 0$, se $a_i = b_j$
$\min(D(i-1, j), D(i, j-1), D(i-1, j-1)) + 1$, se $a_i \neq b_j$

Algoritmo:

```
Para i de 0 a n: D[i, 0] ← i;  Fp;
Para i de 0 a m: D[0, i] ← i;  Fp;
Para i de 1 a n:
   Para j de 1 a m:
      Se (A[i] = B[j]) Então D[i, j] ← D[i-1, j-1];
      Senão D[i, j] ← min(D[i-1, j-1], D[i-1, j], D[i, j-1])+1;
   Fp;
Fp;
```

Complexidade: O(n.m)

Programação Dinâmica

Distância de Edição - Apresentando a transformação

	0	1 A	2 C	3 E	4 R	5 T	6 O
0	0	1	2	3	4	5	6
1 E	1	1	2	2	3	4	5
2 R	2	2	2	3	2	3	4
3 R	3	3	3	3	3	3	4
4 O	4	4	4	4	4	4	3

Programação Dinâmica

Distância de Edição - Apresentando a transformação

	0	1 M	2 A	3 R	4 R	5 O	6 M
0	0	1	2	3	4	5	6
1 A	1	1	1	2	3	4	5
2 M	2	1	2	2	3	4	4
3 O	3	2	2	3	3	3	4
4 R	4	3	3	2	3	4	4
5 A	5	4	3	3	3	4	5

Programação Dinâmica

Distância de Edição - Apresentando a transformação

Algoritmo:

```
Transf(i,j);
  Se (i>0) ou (j>0)
      Se (A[i] = B[j]) Então
           Transf(i-1,j-1);   Escrever 'Manteve' A[i];
       Senão Se ((j > 0) e D[i,j] = D[i,j-1]+1) Então
           Transf(i,j-1);   Escrever 'Inseriu' B[j];
       Senão Se ((i > 0) e D[i,j] = D[i-1,j]+1) Então
           Transf(i-1,j);   Escrever 'Deletou' A[i];
       Senão
           Transf(i-1,j-1);
           Escrever 'Transformou' A[i] 'em' B[j];
       Fs;
Fim;
Externamente:  Transf(n,m);
```

Programação Dinâmica

Distância de Edição

ExS21: Determinar a distância de edição do seu ALSTRING(8 letras iniciais do nome) para o do colega. Mostrar duas transformações mínimas do primeiro string para o segundo.

Programação Dinâmica

Exercício 1 -

a) Resolver intuitivamente o seguinte problema:

Qual o valor máximo possível da soma dos números obtidos pela concatenação dos dígitos do particionamento do vetor de dígitos em segmentos consecutivos de 1 a 3 dígitos?

2	6	3	1	7	8	9	0	9	5

Programação Dinâmica

Exercício 1 -

b) Escrever uma recorrência para o problema:

Qual o valor máximo possível da soma dos números obtidos pela concatenação dos dígitos do particionamento do vetor de dígitos em segmentos consecutivos de 1 a 3 dígitos?

2	6	3	1	7	8	9	0	9	5

Programação Dinâmica

Exercício 1 -

c) Escrever um algoritmo de PD para o problema:

Qual o valor máximo possível da soma dos números obtidos pela concatenação dos dígitos do particionamento do vetor de dígitos em segmentos consecutivos de 1 a 3 dígitos?

2	6	3	1	7	8	9	0	9	5

Programação Dinâmica

Exercício 1 -

d) Escrever um algoritmo de PD para o problema:

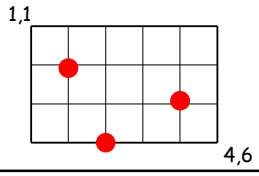
Qual a maneira de particionar um vetor de dígitos em partições de 1 a 3 dígitos, tal que a soma dos números resultantes da concatenação dos dígitos de cada partição seja máxima?

2	6	3	1	7	8	9	0	9	5

Programação Dinâmica

Exercício 2-

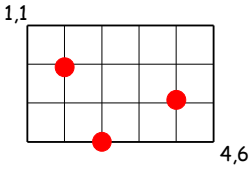
a) Escrever a recorrência que indica o número de caminhos mínimos distintos entre os pontos (1,1) e (n,m) em um grid de dimensões n x m, onde há obstáculos em alguns pontos de cruzamento (no máximo n-1 obstáculos).



Programação Dinâmica

Exercício 2

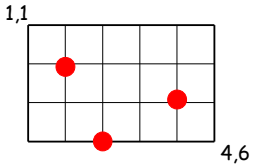
b) Escrever um algoritmo de PD para a formulação anterior.



Programação Dinâmica

Exercício 2

c) Mostrar o preenchimento da matriz para o exemplo abaixo.

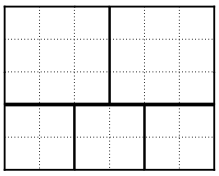


Programação Dinâmica

Exercício 3-

a) Escrever a recorrência que indica o número mínimo de quadrados, T(a,b) que podem ser obtidos com cortes transversais em uma chapa de dimensões a x b.

Exemplo: a = 5, b=6

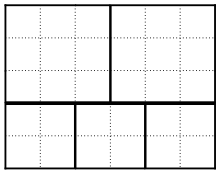


Programação Dinâmica

Exercício 3-

b) Escrever um algoritmo de PD para o problema descrito.

Exemplo: a = 5, b=6

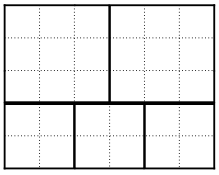


Programação Dinâmica

Exercício 3-

c) Preencher a matriz 6 x 6 relativa ao exemplo.

Exemplo: a = 5, b=6



Programação Dinâmica

FIM