Lista de Exercícios Sobre Programação Dinâmica

Questão 0.1

Caminho dos Presentes. Suponha um caminho de três vias no qual a cada trecho i do caminho, três presentes estão dispostos, um em cada uma das três vias. Os presentes têm valores diferenciados. Suponha que você esteja inicialmente na posição marcada com x e o caminho vai até a posição y (posição final). A partir de x, você percorrerá os n trechos da pista, recolhendo um presente a cada trecho. Em cada movimento de um trecho i ao trecho $i+1$, você apenas pode se mover na mesma pista, ou para uma pista acima, ou para uma pista abaixo. Ou seja, considere as três pistas 1, 2 e 3. Se você estiver na pista 2 no trecho i, você pode permanecer na pista 2 no trecho $i+1$, ou mover-se para as pistas 1 ou 3. Mas caso você estiver na pista 1, você pode permanecer nela, ou mudar-se para a 2. Se você estiver na 3, pode permanecer nela ou mudar-se para a 2. Considere que estando num trecho i você não pode retornar a um trecho $i-1$. O objetivo do problema é maximizar a soma dos presentes que são recolhidos no caminho entre x e y.
Considere o exemplo abaixo, onde as três vias são representadas pelas três linhas e os n trechos são representados pelas colunas :

		Índice da coluna								
		1	2	3	4	5	6	7	8	
1		4	2	3	1	6	3	9	0	
2	**x**	12	8	5	5	2	11	1	4	**y**
3		12	8	8	6	3	0	4	5	

Projete um algoritmo que resolva o problema para matrizes de 3 linhas e n colunas.

a) Qual a soma máxima de presentes que você poderia recolher no exemplo acima? 63

b) Apresente uma definição recursiva para o problema.

$$c(i,j) = \begin{cases} 0 & \text{se j=1} \\ c(i,j) + \max(c(i,j-1), c(i-1,j-1)) & \text{se i=1} \\ c(i,j) + \max(c(i,j-1), c(i-1,j-1), c(i+1,j-1)) & \text{se i=2} \\ c(i,j) + \max(c(i,j-1), c(i+1,j-1)) & \text{se i=3} \end{cases}$$

c) Elabore um algoritmo de programação dinâmica que resolve o problema (escreva o pseudo-código) do algoritmo.

NewSort (S,i,j)

Entrada n e a matrix c com os valores dos presentes

Saída soma dos valores recolhidos

Objetivo retornar a soma máxima possível

```
1  c[1,0]:=0; c[2,0]:=0;c[3,0]:=0; //inicializações
2  para i=1 até n faça
3      se i=1 então
4          c[i,j] := c[i,j] + max(c[i,j-1],c[i-1,j-1]);
5      senão se i=2 então
6          c[i,j] := c[i,j] + max(c[i,j-1],c[i-1,j-1],c[i+1,j-1]);
7      senão i=3 então
8          c[i,j] := c[i,j] + max(c[i,j-1],c[i+1,j-1]);
9  return max(c[1,n],c[2,n],c[3,n]);
```

d) Qual a complexidade de tempo e espaço de pior caso do seu algoritmo? O(n) para tempo e espaço.

Questão 0.2 (Robótica)
Um robô está no ponto A da matriz

$$\begin{array}{cccc} \mathbf{A}: 10 & 5 & 8 & 13 \\ 3 & 19 & 14 & 8 \\ 2 & 4 & 6 & 8 \\ 3 & 5 & 7 & \mathbf{B}: 11 \end{array}$$

e quer chegar no ponto B. Em cada passo, ele pode se mover somente para direita ou para baixo. Ao mesmo tempo, ele coleciona todos items que estão disponíveis ao longo do caminho. Projete um algoritmo que resolva o problema para matrizes arbitrárias.

(a) O caminho com o maior número de pontos é D, B, D, D, B, B com 75 pontos.

(b) Uma solução com programação dinâmica pode ser obtida usando uma matriz $C = (c_{ij})$, sendo o elemento c_{ij} o número máximo de itens que pode ser colecionado a partir do ponto (i, j) até ponto $B = (n, m)$. $P = (p_{ij})$ é a matriz de pontos, os c_{ij} satisfazem a recorrência

$$c_{ij} = \begin{cases} \max\{c_{i(j+1)}, c_{(i+1)j}\} + p_{ij} & \text{se } i < n \wedge j < m \\ p_{ij} + c_{i(j+1)} & \text{se } i = n \wedge j < m \\ p_{ij} + c_{(i+1)j} & \text{se } i < n \wedge j = m \\ p_{ij} & \text{se } i = n \wedge j = m \end{cases}$$

e a solução desejada é c_{11}. No exemplo, temos a matriz

$$C = \begin{pmatrix} 75 & 65 & 49 & 40 \\ 63 & 60 & 41 & 27 \\ 31 & 29 & 25 & 19 \\ 26 & 23 & 18 & 11 \end{pmatrix}$$

que leva ao caminho ótimo D, B, D, D, D, B acima. Uma implementação seria

Caminho mais lucrativo

Entrada Um matriz de pontos $P = (p_{ij})$ de tamanho $n \times m$.

Saída O número máximo de pontos em qualquer caminho do elemento $(1, 1)$ até elemento (n, m) com passos para direita ou para baixo.

```
C := 0
c_nm := p_nm
for i := n-1, ..., 1 do c_im := p_im + c_(i+1)m
for j := m-1, ..., 1 do c_nj := p_nj + c_n(j+1)
for i := n-1, ..., 1 do
   for j := m-1, ..., 1 do
      c_ij := m_ij + max{c_(i+1)j, c_i(j+1)}
   end for
end for
return c_11
```

(c) O algoritmo é correto porque ele tem uma subestrutura ótima: (i) Temos (ao máximo) dois subproblemas de um caminho ótimo de (i, j) até B: os caminhos de $(i+1, j)$ e de $(i, j+1)$. (ii) Uma solução ótima também contém um solução ótima para um subcaminho. A complexidade do algoritmo é $O(mn)$.