GUIÃO 06 – PROGRAMAÇÃO DINÂMICA

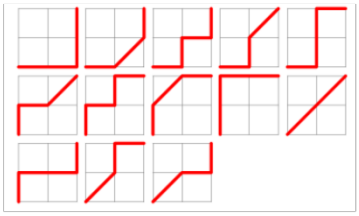
1 – Números de Delannoy

Os números de Delannoy representam o número de caminhos definidos numa grelha, desde o canto inferior esquerdo – posição (0, 0) – até qualquer outra posição (m, n), sendo permitidos apenas movimentos para cima (N), para a direita (E) ou na diagonal (NE).

As figuras seguintes apresentam os caminhos que é possível definir até às posições (1, 1), (2, 2) e (3, 3).

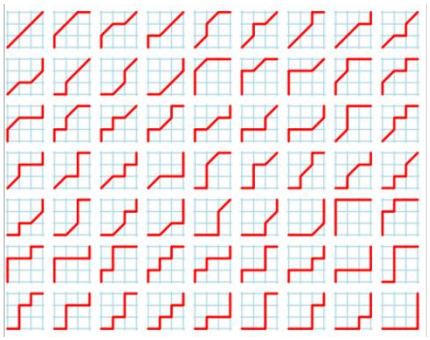
D(1,1)

D(2,2)



[Mathworld]

D(3,3)



[Wikipedia]

Os números de Delannoy são definidos pela seguinte relação de recorrência:

D(m, n) = 1, se m = 0 ou n = 0

D(m, n) = D(m – 1, n) + D(m – 1, n – 1) + D(m, n – 1)

Tarefas

- Implemente uma **função recursiva**, para calcular o número de Delannoy D(m, n).
- Qual é o **maior número D(k, k)** que consegue determinar em tempo útil, usando o seu computador?
- Implemente uma **função iterativa**, usando **Programação Dinâmica** e um **array 2D local**, para calcular o número de Delannoy D(m, n).
- Implemente uma **função recursiva**, usando **memoization** e um **array 2D global**, para calcular o número de Delannoy D(m, n).
- Para analisar o esforço computacional requerido por cada uma das funções, construa uma tabela como a da figura seguinte e conte o **número de adições realizadas** para calcular cada um dos seus elementos. Classifique as funções de acordo com a sua **ordem de complexidade**.

Delannoy's Matrix - Recursive Function

1	1	1	1	1	1	1	1	1	1	1
1	3	5	7	9	11	13	15	17	19	21
1	5	13	25	41	61	85	113	145	181	221
1	7	25	63	129	231	377	575	833	1159	1561
1	9	41	129	321	681	1289	2241	3649	5641	8361
1	11	61	231	681	1683	3653	7183	13073	22363	36365
1	13	85	377	1289	3653	8989	19825	40081	75517	134245
1	15	113	575	2241	7183	19825	48639	108545	224143	433905
1	17	145	833	3649	13073	40081	108545	265729	598417	1256465
1	19	181	1159	5641	22363	75517	224143	598417	1462563	3317445
1	21	221	1561	8361	36365	134245	433905	1256465	3317445	8097453

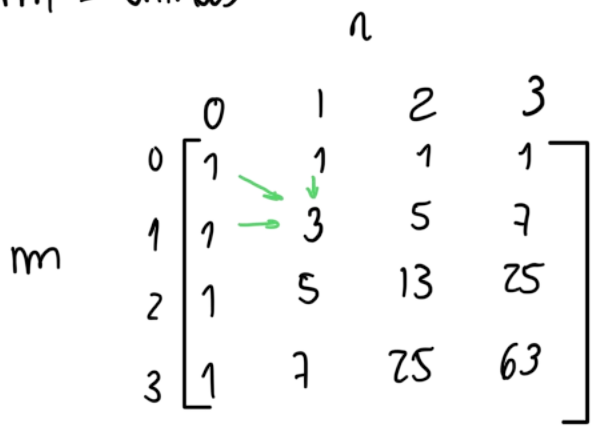
Função Recursiva normal;

$$T(n) = \begin{cases} \\ \\ \\ \end{cases} T$$

Delannoy [3] [3];

n → Colunas

m → Linhas



2 – O Problema da Fileira de Moedas ("The Coin Row Problem")

Seja dada uma sequência de **n moedas** de **valores inteiros** – $\mathbf{c_1, c_2, \ldots, c_n}$ –, com possíveis repetições.

Pretende-se resolver o seguinte problema de otimização combinatória:

- determinar o **valor de um subconjunto de moedas com o maior valor total**, com a **restrição** de que esse subconjunto **não contém moedas que sejam adjacentes na sequência** dada.

Note que podem ocorrer mais do que um subconjunto de moedas com o valor ótimo (i.e., máximo), que são designados soluções ótimas equivalentes.

Para uma sequência de n moedas, o **valor de uma solução ótima** pode ser determinado pela seguinte relação de recorrência, em que **V(i), i = 0, 1, 2, …, n,** representa o valor de uma solução ótima considerando apenas as **primeiras i moedas**.

$$\mathbf{V(0) = 0}$$
$$\mathbf{V(1) = c_1}$$
$$\mathbf{V(n) = \max \{\ c_n + V(n-2),\ V(n-1)\ \},\ para\ n > 1}$$

Tarefas

- Considere a sequência de moedas de valor **5, 1, 2, 10, 6, 2**. Calcule manualmente o valor da correspondente **solução ótima**.

- Implemente uma **função recursiva** que, dada uma sequência de n moedas, cujos valores estão armazenados num array, calcula o valor V(n) de uma sua solução ótima.

- Qual é o **tamanho da maior sequência** que consegue processar em tempo útil, usando o seu computador?

- Implemente uma **função iterativa**, usando **Programação Dinâmica** e um **array local**, para calcular o valor V(n) de uma solução ótima para uma sequência de n moedas.

- Implemente uma **função recursiva**, usando **memoization** e um **array global**, para calcular o valor V(n) de uma solução ótima para uma sequência de n moedas.

- Para analisar o esforço computacional requerido por cada uma das funções, conte o **número de comparações realizadas** para determinar o valor da solução ótima, para sequências de moedas sucessivamente mais longas. Classifique as funções de acordo com a sua **ordem de complexidade**.

Tarefa adicional

- Para a função iterativa que usa **Programação Dinâmica**, desenvolva uma estratégia que, além de determinar o valor de uma solução ótima, **identifique as moedas** que constituem essa solução.