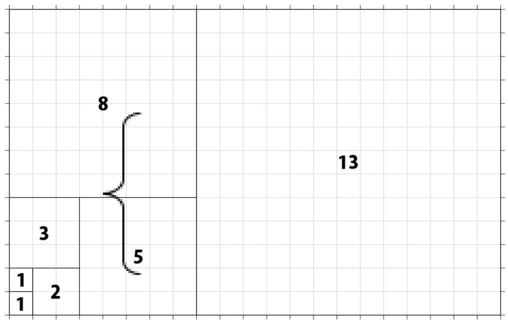
Aula prática 11

Esta aula tem como objetivo estudar e aplicar as metodologias de programação e algoritmos de caminho mais curto em grafos.

1. Sucessão de Fibonacci

1.1 – Escreva um programa que calcule o número de *Fibonacci* de ordem N. O programa deve pedir ao utilizador o número (N) e usar uma função recursiva ***unsigned int fib(unsigned int n)*** para calcular esse valor.



Relembre que a sucessão de Fibonacci pode ser calculada usando a seguinte definição:

$$fib(N) = \begin{cases} 0 & , se\ N = 0 \\ 1 & , se\ N = 1 \\ fib(N-1)+fib(N-2) & , caso\ contrário \end{cases}$$

Exemplo

```
N? 6
O numero de Fibonacci de ordem 6 e 8.
```

1.2 – Altere a função anterior para mostrar todas as chamadas à função ***fib***, bem como contabilizar o número de chamadas. Para tal, a nova função deverá ter o seguinte protótipo: ***unsigned int fib2(unsigned int n, unsigned int * nchamadas)***

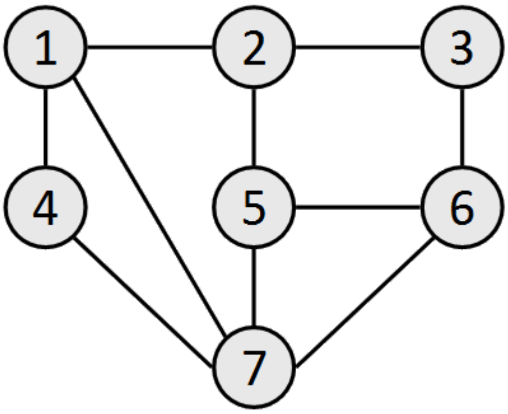
Exemplo

```
N? 2
fib(2) fib(1) fib(0)
fib() foi chamado 3 vezes.
O numero de Fibonacci de ordem 2 e 1.
```

1.3 Crie uma nova função, agora usando o protótipo ***unsigned int fib3(unsigned int n, unsigned int * nchamadas, unsigned int resultados[]),*** que utilize um vetor no qual guarda e consulta resultados já calculados. Sempre que a função necessite de um resultado que já tenha sido calculado (e portanto gravado no vetor), não deverá recalculá-lo. Nos outros casos, a função deverá calcular o valor requerido e guardá-lo no vetor de resultados. A função deve aceitar apenas valores de N entre 0 e 100.

2. Caminho mais curto em grafos

2.1 Utilizando a biblioteca de grafos disponibilizada no Moodle, construa o grafo representado na seguinte figura e determine o caminho mais curto entre os **vértices 1** e **6**.



2.2 Considere o caso em que <u>apenas uma aresta</u> tem peso diferente das restantes arestas. Implemente uma nova função que calcule o caminho mais curto neste cenário com o seguinte protótipo: ***int* caminho(grafo *g, int inicio, int fim, int *n, int peso17).*** Teste esta função para determinar o caminho mais curto entre o vértice 1 e os restantes vértices, considerando diferentes pesos para aresta que liga os **vértices 1** e **7** (parâmetro ***peso17***). <u>Nota</u>: basta recorrer às funções já disponíveis na biblioteca de grafos.

3. Triângulo de Pascal

3.1 – Escreva um programa que para calcular termos do triângulo de Pascal (dada uma linha e coluna). Cada termo pode ser calculado segundo a seguinte definição matemática:

$$p(lin,col) = \begin{cases} 0 & \text{, se } col > lin \\ 1 & \text{, se } col = 0 \\ p(lin\text{-}1,\ col\text{-}1) + p(lin\text{-}1,\ col), & \text{caso contrário} \end{cases}$$

	0	1	2	3	4	5	6	7	8
0	1	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0
2	1	2	1	0	0	0	0	0	0
3	1	3	3	1	0	0	0	0	0
4	1	4	6	4	1	0	0	0	0
5	1	5	10	10	5	1	0	0	0
6	1	6	15	20	15	6	1	0	0
7	1	7	21	35	35	21	7	1	0
8	1	8	28	56	70	56	28	8	1

Exemplo

```
Linha? 6
Coluna? 3
O elemento do triângulo de Pascal em (6, 3) é 20.
```

3.2 Imprima as primeiras 32 linhas do triângulo de Pascal (usando programação dinâmica).

Referências:

http://mathworld.wolfram.com/FibonacciNumber.html

http://mathworld.wolfram.com/PascalsTriangle.html