Prova com consulta (Moodle). Duração: 70min. Parte prática [15/20 valores]

No final deverá confirmar que submeteu corretamente no SIGEX o código fonte dos seus programas utilizando o nome indicado no enunciado.
Quaisquer cópias detetadas serão penalizadas com anulação da prova.

1 [8 valores] Responda às duas alíneas seguintes no ficheiro **prob1.c**. Utilize sempre que conveniente as bibliotecas de funções disponibilizadas para gestão e análise das estruturas de dados.

1.1 [5 valores] Implemente a função `max_heap` que cria uma max-heap a partir de uma min-heap dada, isto é, quando removidos, os elementos da heap retornada devem vir pela ordem inversa dos elementos da heap dada. Após a execução da função, a heap dada deve preservar todo o seu conteúdo inicial.

```
heap *max_heap(heap *h)
```

A função recebe o apontador para a heap dada e deverá retornar o apontador para a nova heap ou NULL em caso de erro. Utilize as funções disponíveis na biblioteca `heap.h/.c`.

Depois de implementada a função, o programa deverá apresentar:

```
Sequência de elementos removidos da heap original (min-heap):
A B C D E F
Sequência de elementos removidos da heap retornada (max-heap):
F E D C B A
```

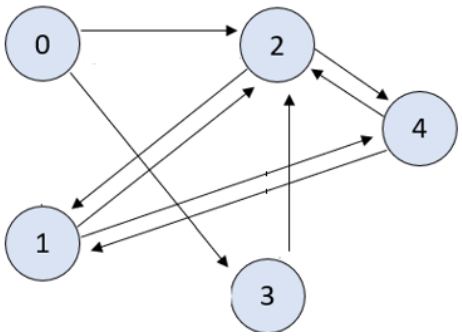
1.2 [3 valores] Suponha que tem uma árvore AVL que guarda números inteiros. Pretende criar uma função que, ao receber um inteiro **x**, retorna um vetor ordenado (por ordem crescente) contendo todos os inteiros da árvore menores ou iguais a **x**.

Explique o algoritmo que desenvolveria para esta função e indique a respetiva complexidade temporal (no pior caso).

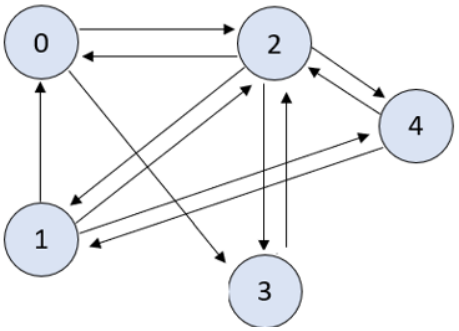
***** Submeta o ficheiro prob1.c no SIGEX *****

Prova com consulta (Moodle). Duração: 70min. Parte prática [15/20 valores]

2 [7 valores] Implemente as funcionalidades pedidas nas duas alíneas seguintes no ficheiro **prob2.c**. Utilize a biblioteca de grafos (implementada com lista de adjacências).



Grafo 1



Grafo 2

2.1 [4 valores] Implemente a função `contaVerticesComDuplaLigacao` que recebe um digrafo e retorna o número de vértices **diferentes** que têm pelo menos uma ligação dupla. Uma ligação é dupla se o nó **Vi** tiver uma aresta para outro nó **Vk** e houver também uma aresta do nó **Vk** de volta ao próprio **Vi**. A função deve retornar -1 em caso de erro.

A assinatura da função é a seguinte:

```
int contaVerticesComDuplaLigacao(grafo *)
```

Para o Grafo 1 o programa deverá apresentar:

```
O Grafo 1 tem 3 vertices duplamente ligados
```

Para o Grafo 2 o programa deverá apresentar:

```
O Grafo 2 tem 5 vertices duplamente ligados
```

NOTA: Um vértice pode ter duplas ligações para mais do que um outro vértice mas só conta uma vez. É o caso, por exemplo, do vértice 4, entre outros.

2.2 [3 valores] Implemente a função **subgrafo** que recebe dois **digrafos** e retorna: 1 se o grafo no primeiro argumento for subgrafo do segundo; 0 se o grafo no primeiro argumento **não** for subgrafo do segundo; e -1 em caso de erro. Um grafo **g1** é subgrafo de **g2 se e só se**: 1) todos os [identificadores de] vértices de **g1** existirem em **g2**; 2) todas as arestas de **g1** existirem em **g2**; 3) **g1** e **g2 não são** iguais. Todos os grafos envolvidos têm arestas sem pesos.

Considere a seguinte assinatura da função:

```
int subgrafo(grafo *g1, grafo *g2)
```

Para a chamada com g1 = **Grafo 1** e g2 = **Grafo 2** o programa deverá apresentar:

```
g1 E subgrafo de g2
```

Para a chamada com g1 = **Grafo 2** e g2 = **Grafo 1** o programa deverá apresentar:

```
g1 NAO E subgrafo de g2
```

Para a chamada com g1 = **Grafo 2** e g2 = **Grafo 2** o programa deverá apresentar:

```
g1 NAO E subgrafo de g2
```

***** Submeta o ficheiro prob2.c no SIGEX *****