UNIVERSIDADE DA BEIRA INTERIOR

Algoritmos e Estruturas de Dados **2º Semestre**

Exame de Recurso (16 val) **2 h + 30 min** **2024/2025**

A. Análise de complexidade dos algoritmos

Considere o seguinte bloco de código em linguagem C:

```
for (k = n; k >= 1; k = k-1) {
    for (j = 0; j < k; j++)
        printf("OLA");
}
```

1. Simule o bloco de código dado, considerando que **n** é um número inteiro **muito grande**. Apresente os dados da simulação numa tabela cujo cabeçalho é o seguinte:

k	k >= 1 (S/N)	j	j < k (S/N)	printf
…	…	…	…	…

2. A partir dos resultados da simulação obtidos em **1.**, determine a **ordem de complexidade** do algoritmo associado ao bloco de código dado. Justifique, apresentando os cálculos efetuados.

B. Listas ligadas

Considere as seguintes declarações associadas a EAD Lista:

```
typedef int INFOLista;   // INFOLista = int
struct NodoLista {
    INFOLista  Elemento;
    struct NodoLista *Prox;
};
typedef struct NodoLista *PNodoLista;
```

assim como o seguinte protótipo de uma função já implementada (operação básica sobre a EAD Lista):

```
PNodoLista libertarNodoLista (PNodoLista P); // liberta a memória apontada por P e devolve NULL
```

1. Implemente uma **função recursiva em C** que

- **receba** uma lista **L** com elementos do tipo **INFOLista** (inteiros), e
- **devolva** um ponteiro para o **nodo** com o maior elemento de **L** (elemento com maior balor no campo Elemento), caso exista; se não existe, então devolver **NULL**.

2. Implemente uma função iterativa (não recursiva) em C que

- **receba** uma lista **L** de elementos do tipo **INFOLista** (inteiros), e
- **remova** todos os elementos de **L** com **valor positivo** (> 0) no campo **Elemento**.

Nota: Otimizar os algoritmos, de forma que a lista L seja percorrida apenas uma vez.

C. Pilhas e Filas

Considere as seguintes declarações de **EAD Fila** e **EAD Pilha**:

```
typedef int INFOFila;     // INFOFila = int
struct NodoFila { ... };
typedef struct NodoFila *PNodoFila;
```

e os seguintes protótipos de funções:

PNodoFila **criarFila** ();
int **filaVazia** (PNodoFila Q);
PNodoFila **juntar** (INFOFila X, PNodoFila Q);
PNodoFila **remover** (PNodoFila Q);
INFOFila **frente** (PNodoFila Q);

```
typedef INFOFila INFOPilha;
struct NodoPilha {  ... };
typedef struct NodoPilha *PNodoPilha;
```

e os seguintes protótipos de funções:

PNodoPilha **criarPilha** ();
int **pilhaVazia** (PNodoPilha S);
PNodoPilha **push** (INFOPilha X, PNodoPilha S);
PNodoPilha **pop** (PNodoPilha S);
INFOPilha **topo** (PNodoPilha S);

Usando as operações básicas sobre uma EAD Fila e uma EAD Pilha (se necessário), implemente uma **função em C** que

- **receba** uma **fila Q** com elementos do tipo INFOFila (números inteiros), e
- **devolva** a fila **Q** com os seus elementos reorganizados da seguinte forma: os elementos com valor negativo (< 0) estão na cauda da fila e os restantes elementos (>= 0) estão na frente da fila, mantendo a ordem original entre os postivos e entre os negativos.

Ex: Q = [-3, 2, 4, -5, -4, 5] (frente(Q) = **-3**) ==> Q = [2, 4, 5, -3, -5, -4] (frente(Q) = **2**)

D. Árvores Binárias de Pesquisa (ABP)

Considere as seguintes declarações associadas de EAD Árvore Binária de Pesquisa:

```
typedef int INFOABP;    // INFOABP = int
struct NodoABP {
   INFOABP  Elemento;
   struct NodoABP  *Esquerda;
   struct NodoABP  *Direita;
};
typedef struct NodoABP *PNodoABP;
```

1. Implemente uma **função recursiva** em C que

- **receba** uma ABP **T** e um número **inteiro X**, e
- **devolva** a quantidade de elementos de **T** que são **folhas** (nodos sem filhos) com valor no campo **Elemento menores que X (< X)**.

2. Implemente uma **função iterativa** (**não recursiva**) em C que

- **receba** uma ABP **T** e um número **inteiro X**, e
- **devolva** um ponteiro para o filho do **nodo** com valor no campo **Elemento** igual **X** com maior valor no campo **Elemento**, caso exista; caso não exista, a função deve devolver **NULL**.

Nota: otimize os algoritmos criados (código escrito) tendo em conta a **definição de ABP**.

E. BP Balanceadas/Equilibradas (AVL)

Construa uma **ABP do tipo AVL** da seguinte forma:

- **insira** os nodos com os seguintes valores (um de cada vez e pela ordem apresentada):

  70, 50, 80, 60, 55, 65 e 90. (**NOTA**: não é necessário colocar as alturas).

  Sempre que **inserir um nodo**, deve **acrescentar** o novo nodo à árvore e **verificar se continua balanceada/equilibrada**. Caso **não fique equilibrada**, deve

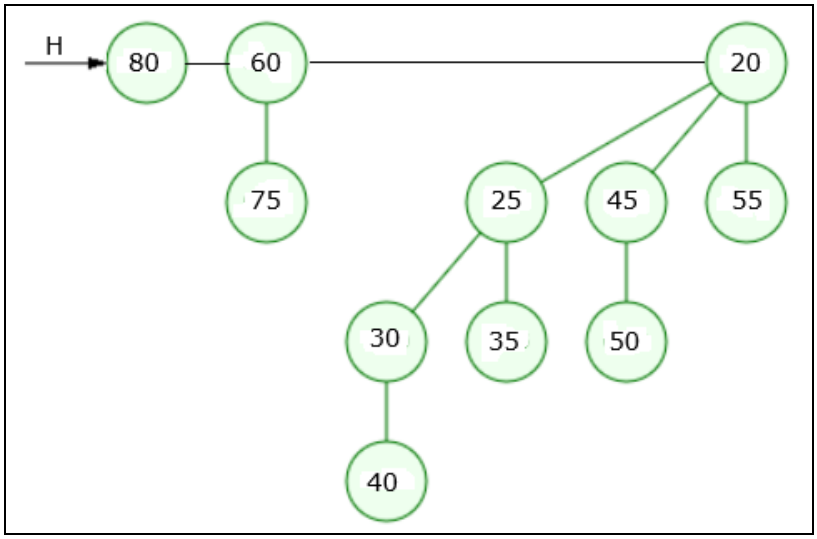
  1º) **indicar** o **nodo** onde se deteta o desequilíbrio e o **tipo de rotação** a aplicar para reequilibrá-la,

  2º) **reequilibrar** a árvore e **redesenhá-la** (apresentando todos os passos efetuados)

- **remova** o **nodo raiz** da ABP e **redesenhe** a ABP obtida, **verifique** se continua balanceada/equilibrada e **reequilibre-a**, caso seja necessário. **Desenhe a ABP final**.

F. Heaps

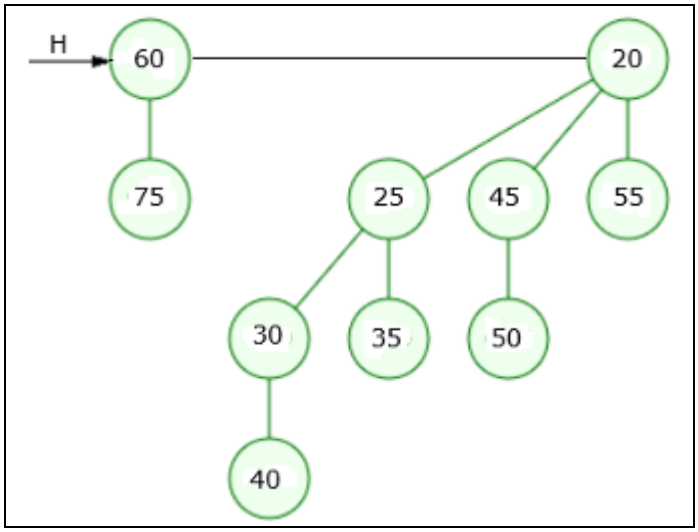
1. Considere o seguinte **heap Binomial H** com estrutura de minHeap:



Determine e **desenhe** o heap Binomial resultante da operação **"inserir"** um nodo com valor **50** no heap H. Justifique, apresentando a evolução do heap desde a sua estrutura inicial até à final.

NOTA: Sempre que redesenhar o heap Binomial, descreva o processo aplicado para o obter.

2. Considere o seguinte **heap Binomial H** com estrutura de minHeap:

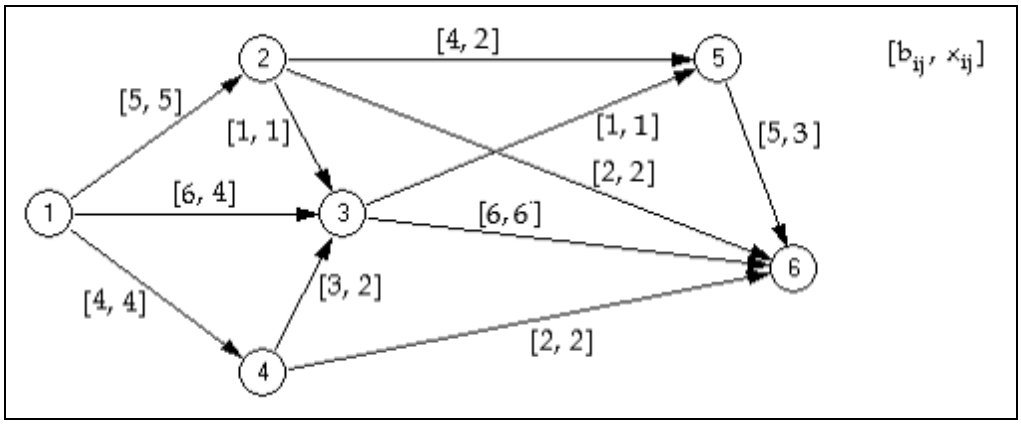


Determine e **desenhe** o heap Binomial resultante da operação **"remover"** aplicada ao heap H dado. Justifique, apresentando a evolução do heap desde a sua estrutura inicial até à final.

NOTA: Sempre que redesenhar o heap Binomial, descreva o processo aplicado para o obter.

G. Grafos

Considere a seguinte rede G = (A, N, C), onde o valor de cada arco corresponde ao seu custo ($\mathbf{C_{ij}}$):



1. Determine o fluxo atual da rede entre os nós **1** e **6**. Justifique.

2. Determine o **fluxo máximo** que pode ser enviado do nó **1** para o nó **6**, simulando o algoritmo de **Ford-Fulkerson**. Apresente os dados da simulação: todos os valores das estruturas de dados **R**, **Nós rotulados** e **Nós rotulados varridos**, e das variáveis **j** e **k**.

Esquema para apresentar os dados da simulação:

	1	2	3	4	5	6
R						

Nós **rotulados** ←

Nós **rotulados varridos** ←

j ← ?

k ← ...

k ← ...

Algoritmo de Ford-Fulkerson:

Passo 1.
R(S) ← [S^+, ∞] (S é rotulado com S^+ e ∞)
S fica rotulado e não varrido

Passo 2. (processo de rotulação)
j (rotulado e não varrido) ← [i^+, Q(j)] ou [i^-, Q(j)]
para (todo **k** ∈ **N** não rotulado, tal que (j, k) ∈ A e $x_{jk} < b_{jk}$) **repetir**
R(k) ← [j^+, Q(k)], com Q(k) = min{ Q(j), $b_{jk} - x_{jk}$ }
fim_para
para (todo **k** ∈ **N** não rotulado, tal que (k, j) ∈ A e $x_{kj} > 0$) **repetir**
R(k) ← [j^-, Q(k)], com Q(k) = min{ Q(j), x_{kj} }
fim_para
j fica **rotulado** e **varrido**
todos os nós **k** ficam **rotulados** e **não varridos**
se (T está rotulado) **ou** (não é possível rotular T) **então**
(foi determinado um c.a.f. ⇒ **passar** ao **Passo 3**) **ou**
(não existe c.a.f. — o fluxo atual é máximo ⇒ **PARAR**)
senão
regressar ao **Passo 2** (início)
fim_se

Passo 3. (mudança de fluxo)
como (R(T) = [k^+, Q(T)]) **então**
x_{kT} ← x_{kT} + Q(T)
enquanto (k ≠ S) **repetir**
se (R(k) = [j^+, Q(k)]) **então**
x_{jk} ← x_{jk} + Q(T)
fim_se
se (R(k) = [j^-, Q(k)]) **então**
x_{kj} ← x_{kj} – Q(T)
fim_se
k ← j
fim_enquanto
apagar os rótulos
regressar ao **Passo 1**