UNIVERSIDADE DA BEIRA INTERIOR

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

Frequência 1 (8 val) **1:15 h** **2023/2024**

A. [1.0 val] Análise de complexidade dos algoritmos

Considere o seguinte bloco de código em linguagem C:

```
. . .
cont = 0;
for (k = 0; k < n; k++) {
    P = k + 1;
    for (j = 1; j <= P; j++)
        cont = cont + 1;
}
. . .
```

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 < n (V/F)	P	j	j <= P (V/F)	cont
...	...	...	...	...	...

2. A partir dos resultados da simulação obtidos em 1, determine a ordem de complexidade do algoritmo associado ao bloco de código dado em cima. Incidir os cálculos sobre a operação que mais vezes é executada ("**j <= P**"). Justifique, apresentando os cálculos efetuados.

B. [1.75 val] EAD Pilha

Considere as seguintes declarações de EAD Pilha de elementos do tipo inteiro:

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

e os seguintes protótipos de funções (operações básicas):

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

Usando as operações básicas sobre uma EAD Pilha, implemente uma **função em C** que

- **receba** uma **pilha S** já preenchida com valores inteiros (parâmetro da função),
- **devolva** o elemento com valor positivo que se encontra **mais próximo** do topo da pilha **S**, mas deixando a pilha S inalterada.

Ex: S = [-3, -12, 4, -5, -4, 5], com topo(S) = -3 ==> resultados: 4 e S = [-3, -12, 4, -5, -4, 5]

C. [1.75 val] EAD Árvore Binária (AB)

Considere a seguinte declaração de EAD Árvore Binária:

```
struct NodoAB {
   int    Elemento;
   struct  NodoAB  *Esquerda;
   struct  NodoAB  *Direita;
};
typedef  struct  NodoAB *PNodoAB;
```

Implemente uma função em C que

- **receba** uma AB **T** e um número inteiro **num**,
- **devolva** o elemento (**int**) da AB **T** com **maior valor positivo** (≥ 0), se existir; caso não existam elementos com valores positivos, a função deve **devolver** um valor **negativo** qualquer.

D. [3.5 val] Listas ligadas

Considere as seguintes declarações:

```
struct Nodo {
   int    Elemento;
   struct  Nodo  *Prox;
};
typedef  struct  Nodo *PNodo;
```

Considere também os seguintes protótipos de funções já implementadas:

```
int listaVazia (PNodo L);   // devolve 1 se lista L é vazia e 0 caso contrário
PNodo libertarNodo (PNodo P);   // liberta a zona de memória apontada por P e devolve NULL
```

Usando as definições e os protótipos das funções dadas, resolva as seguintes questões:

1. **[1.5 val] Implemente** uma **função recursiva em C** que
   - **receba** uma lista não vazia **L**, com elementos do tipo **inteiro**,
   - **devolva** o elemento de **L** com **maior** valor.
2. **[2.0 val] Usando** ou **não** as funções fornecidas em cima, mas **não podendo** usar outras funções, **implemente uma função iterativa (não recursiva) em C** que
   - **receba** uma lista não vazia **L** de elementos do tipo **inteiro**,
   - **devolva** a lista L após **remover** o **segundo** elemento com valor positivo que se encontra **mais próximo** da cabeça da lista **L**, caso exista; se não existir, a lista não sofre alteração.

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

   Exemplo: Se **L** é a lista em baixo, então o elemento a remover é aquele com o número **8**, pois é o segundo elemento mais próximo da cabeça da lista (nodo com 5) com valor positivo