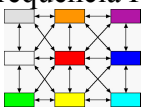
Estruturas de Dados
2015/16
Frequência F2



Universidade da Beira Interior
Departamento de Informática

1. Suponha que a UBI decidiu implementar uma nova estrutura de dados para guardar informação sobre os seus alunos. Para efeitos de optimização de desempenho, optou-se por manter a informação numa árvore binária de pesquisa, utilizando como atributo-chave o número de aluno.

```
typedef struct ALUNO{
        int numero;     //critério de ordenação na árvore
        char nome[80];
        int codigoCurso;
        struct ALUNO *fe, *fd;
}Aluno;
```

1a) (2 valores) Mediante a estrutura apresentada, esquematize a organização da informação resultante do armazenamento dos seguintes itens, inseridos segundo a ordem pela qual aparecem abaixo.

- Aluno 2, José Andrade, curso 1;
- Aluno 3, Vitor Coelho, curso 2;
- Aluno 4, Andreia Soares, curso 3;
- Aluno 1, Paula Correia, curso 1;
- Aluno 5, João Raposo, curso 1;
- Aluno 7, Sandra Henrique, curso 3;

1b) (2 valores) Como pode verificar, a inserção dos elementos na árvore pela ordem apresentada leva a que a estrutura não fique balanceada no seu estado final. Na sua opinião, qual o maior problema que resulta da utilização de estruturas não-balanceadas? Apresente uma ordem para a inserção dos respectivos elementos que tornaria a estrutura resultante balanceada.

1c) (2 valores) Implemente uma função que imprima o nome dos alunos inscritos num determinado curso.

Protótipo: void mostraAlunosCurso(Aluno *A, int codigoCurso);

1d) (2 valores) Codifique uma função que retorne (através de parâmetros "fe" / "fd" passados por referência) o total de ligações "fe" e "fd" activas na árvore (isto é, diferentes de NULL).

Protótipo: void totalApontadoresActivos(Aluno *A, int *fe, int *fd);

1e) (3 valores) Implemente uma função que conte de forma eficiente o total de alunos com numero superior a um valor introduzido por parâmetro.

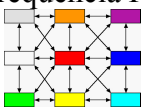
Protótipo: int contaAlunos(Aluno *A, int numero);

Considere que tem à sua disposição a seguinte função:

```
int contaNos(Aluno *A);
//Devolve o total de nós numa árvore binária
```

2) Considere a seguinte estrutura de dados relativa à implementação de um grafo, utilizando listas de adjacência. O grafo serve para registar as ligações de "Amigos" na rede social "*LivroDeCaras*", cuja característica mais distintiva é o facto das relações de amizade poderem não ser bidireccionais, isto é, "X" ser amigo de "Y", mas o contrário não se verificar. Assim, no respectivo grafo, uma aresta "X→Y" significa que "X" é amigo de "Y".

```
typedef struct AMIGO{
        int verticeAmigo;
        int custo;    //a utilizar na alínea c)
        struct AMIGO *nseg;
} Amigo;
```



2a) (3 valores) Implemente uma função que imprima o ID das pessoas (vértices) que são amigas de todas as outras pessoas registadas.

Protótipo: void amigoTodos(Amigo **G, int tv);

2b) (3 valores) Implemente uma função que verifique se existem relações de amizade repetidas, isto é, casos em que "X→Y" mais que uma vez. A função deve retornar "1" em caso afirmativo ou "0" caso contrário.

Protótipo: int amigosRepetidos(Amigo **G, int tv);

2c) (3 valores) Implemente uma função que imprima o ID dos amigos de grau até "n" de um determinado elemento. Por exemplo, no caso de "X→Y" e "Y→Z", considera-se que "X" é amigo de "Y" em grau 1 e amigo de "Z" em grau 2.

Considere que tem à sua disposição a seguinte função:

int** Dijkstra(Amigo **G, int tv, int vo, int vd);

Esta função devolve a tabela que resulta da execução do algoritmo de Dijkstra, isto é, uma matriz com "tv" linhas e 3 colunas, em que a primeira coluna descreve os vértices explorados, a segunda o respectivo custo e a terceira o predecessor de cada vértice ("-1" indica "sem predecessor").

Protótipo: void mostraAmigosGrauN(Amigo **G, int tv, int codigoPessoa, int grauN);