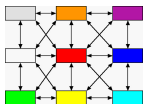


Universidade da Beira Interior
Departamento de Informática
2010/11

1. Considere que as seguintes estruturas de dados são utilizadas pela "UNICEF" para guardar informação acerca dos donativos efectuados. Dado o potencial elevado número de beneméritos, foi julgado conveniente organizá-los em estruturas binárias, por forma a acelerar a pesquisa. Dentro do nó relativo a cada benemérito, existe uma lista que regista todos os seus donativos.

```
typedef struct DONATIVO{
        int id;             //identificador do donativo
        float valor;
        int dia, mês, ano;      //As listas estão ordenadas cronologicamente
        int idDestino;          //identificação do programa UNICEF que recebeu o donativo
        struct DONATIVO *nseg;
}Donativo;

typedef struct NODO_AB{
        int BI;   //chave de ordenação na árvore. Suponha-se que todos os BIs são diferentes
        Donativo *donativos;    //lista de donativos
        struct NODO_AB *fe, *fd;
}NodoAB;
```

1a) (2 valores) Esquematize as estruturas de dados resultantes do armazenamento da seguinte informação. Considere que os nós foram inseridos nas estruturas pela ordem em que aparecem listados:

- BI: 12345678
  - Donativo 1: 10/02/2010, destino: 1, 20€
  - Donativo 2: 18/02/2010, destino: 2, 50€
  - Donativo 3: 19/03/2010, destino: 3, 70€
  - Donativo: 4 10/05/2010, destino: 1, 90€

- BI: 11112222
  - Donativo 5: 10/02/2010, destino: 1, 25€

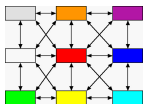
- BI: 99991111
  - Donativo 6: 25/07/2010, destino: 2, 25€
  - Donativo 7: 28/07/2010, destino: 2, 15€

- BI: 10000000
  - Donativo 8: 10/03/2010, destino: 3, 50€

1b) (3 valores) Implemente uma função que devolva o valor total de donativos feitos por determinada pessoa num ano (-1 se a pessoa não for encontrada):

Protótipo: float totalDonativos(NodoAB *A, int bi, int ano);

1c) (3 valores) Codifique a função que verifica a existência de “donativos-fraude”, isto é, donativos com "id" repetido para a mesma ou diferentes pessoas (1=Sim, 0=Não). Se achar conveniente, pode criar funções



auxiliares.

Protótipo: int existemFraudes(NodoAB *A);

2. Considere um grafo representado através de listas ligadas de adjacências, definidas pela seguinte estrutura:

```
typedef struct NODO{
        int custo;
        int idAresta;
        char idDestino; //identificador do vértice destino ('A'=posição 0, 'B'=posição 1,…)
        struct NODO *nseg;
}Nodo;
```

a) (3 valores) Implemente uma função que conte o total de lacetes no grafo.

Protótipo: int totLacetes(Nodo g, int totVertices);**

b) (3 valores) Implemente a função que verifica a adjacência de duas arestas (1=Sim, 0=Não).

Protótipo: int arestasAdjacentes(Nodo **g, int totVertices, int idAresta1, int idAresta2);

c) (3 valores) Considere a seguinte tabela, resultante da aplicação do algoritmo de Dijkstra ao grafo. Enumere os vértices que podem ser atingidos, utilizando o melhor caminho a partir do vértice "A" e sem passar pelo vértice "B".

Vértice	Custo	Predecessor
A	0	-
C	12	A
B	18	A
D	27	B
E	38	B
F	41	C
G	47	E
H	58	G
I	123	G
J	165	H

d) (3 valores) Suponha que informação similar à da tabela acima exemplificada é guardada na seguinte estrutura de dados (de acesso sequencial):

```
typedef struct AUX{
        char vertice;
        int custo;
        char predecessor;
        struct AUX* nseg;
}Aux;
```

Implemente uma função que mostre os vértices que podem ser visitados utilizando o melhor caminho a partir do vértice origem e passando no máximo por "n" arestas:

Protótipo: void mostraVerticesAcessiveisN(Aux *L, int n);