Algoritmos e Complexidade
LEI/LCC (2º ano)

11ª Ficha Prática

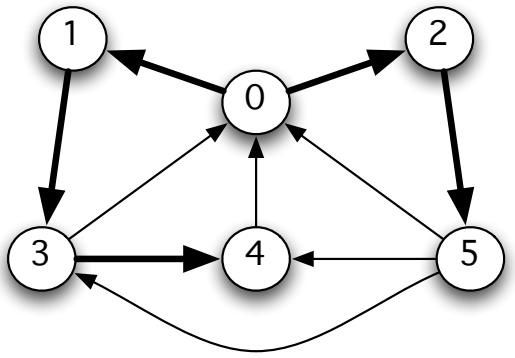
Ano Lectivo de 2009/10

Considere a seguinte definição para representar (as arestas de) um grafo.

```
#define MaxV ...
#define MaxE ...

typedef struct edge {
     int dest;
     int cost;
     struct edge *next;
} Edge, *Graph [MaxV];
```

Uma árvore pode ser vista como um caso particular de um grafo: trata-se de um grafo ligado e acíclico. Numa árvore, cada vértice (com excepção da raiz) tem exactamente um antecessor. Daí que seja comum representar uma árvore num vector de antecessores (em que para cada vértice se guarda o índice do seu antecessor; na componente correspondente à raiz, guarda-se -1).
No grafo que abaixo se apresenta, a árvore a *bold* é representada no vector da direita.



0	1	2	3	4	5
-1	0	0	1	3	2

Cada um destes vectores pode ainda ser usado para representar um conjunto (disjunto) de árvores, a que é costume referir-se por *floresta*.

1. Defina uma função `int raiz (int f[], int v)` que, dada uma floresta `f` e um vértice `v` determine (o índice d)a raiz da árvore a que o vértice pertence.

2. Usando a função anterior, defina as seguintes funções:

   (a) `int inTree (int f[], int a, int b)` que testa se dois vértices estão na mesma árvore.

   (b) `void joinTree (int f[], int v1, int v2)` que, dada uma floresta `f` e dois vértices pertencentes a árvores distintas, junta essas árvores ligando a raiz da árvore de `v1` a `v2`.

3. Defina uma função `int custo (Graph g, int f[])` que calcula o custo total de uma árvore num grafo (a soma dos custos de todos os arcos). A função deve retornar 0 se algum dos ramos da árvore não for um ramo do grafo. Certifique-se que a função que definiu não tem uma complexidade assimptótica superior a $\mathcal{O}(V + E)$ em que V e E são os números de vértices e arestas do grafo.

4. Defina uma função `int altura (int f[])` que calcula a altura de uma árvore representada num vector de antecessores. Analise o tempo de execução da função apresentada em função do número de vértices. Identifique o melhor e pior casos.

5. Considere a seguinte definição de uma procura *depth-first* (que testa se um vértice `d` é alcançável a partir de um vértice `o` num grafo `g`).

```
int DF_aux (Graph g, int o, int d, int v[]) {
    int r = 0;
    Edge *aux;
    v[o] = 1;
    if (o==d) r = 1;
    else for (aux=g[o]; (aux && !r); aux = aux->next)
              if (! v [aux->dest]) r = DF_aux (g,aux->dest,d,v);
    return r;
}

int DF_search (Graph g, int o, int d){
    int i, vis [MaxV];
    for (i=0;(i<MaxV); vis[i++] = 0);
    return (DF_aux(g,o,d,vis));
}
```

   Apresente uma definição da função `int travessia_DF (Graph g, int o, int f[])` que percorre um grafo a partir de um dado vértice segundo uma estratégia *depth-first*. Esta função deverá retornar o número de vértices alcançáveis a partir do vértice dado. Deverá ainda preencher o vector `f[]` com a árvore correspondente à travessia (i.e., com as arestas usadas).