Algoritmos e Complexidade
LEI/LCC (2º ano)

12ª Ficha Prática

Ano Lectivo de 2009/10

1. Considere a seguinte definição para representar (as arestas de) um grafo, bem como uma implementação do algoritmo de Prim para cálculo de uma árvore geradora de custo mínimo de um grafo não orientado, pesado e ligado.

```
#define MaxV ...
#define WHITE 0
#define GRAY  1
#define BLACK 2

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

```
int primMST (Graph g, int p[], int w[]) {
int i, v, r=0, fsize, col [MaxV];
Fringe f = newFringe (MaxV);

for (i=0;i<MaxV;i++){
   p[i] = -1; col [i] = WHITE;
  }
col [0] = GREY; w [0] = 0;
f = addV (f, 0, 0);
fsize=1;
while (fsize) {
   fsize--;
   f = nextF(f, &v);
   col [f] = BLACK;´
   r += w[v];
   for (x=g[v]; x; x = x->next)
       switch (col [x->dest]) {
       case WHITE: col [x->dest] = GREY;
                   fsize++;
                   f = addV (f, x->dest, x->cost);
                   break;
       case GREY : f = updateV (f, x->dest, x->cost);
                   break;
       default   : break;
      }
  }
}
```

A função `primMST` baseia-se numa estrutura auxiliar – `Fringe` – para armazenar a orla. As funções necessárias sobre esta estrutura são `newFringe` (inicialização), `nextF` (remoção de um elemento), `addV` (adição de um elemento) e `updateV` (actualização do custo de um elemento). Considere como alternativas para implementar a orla: (A) Um vector com os pesos de cada vertice da orla, (B) uma lista com os vértices ordenados pelo seu peso, (C) uma *minheap* dos vértices ordenada pelo peso. Para cada uma destas alternativas,

(a) Apresente definições das funções referidas.

(b) Analise a complexidade (pior caso) da função `primMST` resultante.

2. Um algoritmo alternativo ao de Prim para cálculo de uma árvore geradora de custo mínimo deve-se a Joseph. B. Kruskal (1956) e pode ser descrito como *seleccionar as (n-1) arestas de menor peso do grafo que não formam ciclos*. Para isso começa-se por construir uma floresta com uma árvore (unitária) para cada vértice. De seguida vão-se acrescentando arestas por ordem crescente do seu peso (que unam árvores disjuntas, evitando assim os ciclos) juntando as respectivas árvores numa só.

   Apresente uma implementação deste algoritmo e analise o seu comportamento assimptótico no pior caso. Tenha especial atenção à estratégia usada para a escolha da aresta de menor peso.

3. Dado um grafo orientado e acíclico, uma ordenação topológica é uma sequência de vértices `[v0, v1, v2,...]` em que cada vértice só aparece na sequência depois de todos os seus antecessores.

   Uma solução para resolver este problema de uma forma eficiente (Kahn, 1962) consiste em guardar, para cada vértice o número de antecessores que ainda não apareceu na ordenação. Assim, sempre que um vértice é adicionado à ordenação, decrementa-se o contador associado a cada um dos seus sucessores (sempre que esse contador passar a zero, pode ser acrescentado à ordenação).

   Apresente uma implementação deste algoritmo e analise o seu comportamento assimptótico no pior caso.

4. Qual o comportamento do algoritmo anterior para o caso de o grafo não ser acíclico? Use esse facto para definir uma função que testa se um grafo é acíclico.