Algoritmos e Complexidade — Universidade do Minho
Exame 8 de Fevereiro de 2022

Número: _____ Nome: _____

1. Análise de algoritmos (7.5 valores)

Considere as duas seguintes funções. Ambas contam o número de ocorrências de um elemento `x` num array `u` de comprimento `size`.

```
int count(int u[], int x, int size) {
  // pre: size >= 0;
  int i = 0, c = 0;
  while (i < size) {
    if (u[i] == x) { c++; }
    i++;
  }
  // pos: c == count_l(u, x, size) ;
  return c;
}

int  count_l (int u[], int x, int n) =
  if (n<=0) return 0 ;
  if (x==u[n-1]) return 1+count_l(u, x, n-1) ;
  return count_l(u, x, n-1) ;
}
```

a. Analise o tempo de execução de `count`, com base no número de operações de comparação `(u[i] == x)` efetuadas.

b. Apresente uma recorrência adequada para analisar o tempo de execução de `count_l`. Resolva-a e identifique o tempo de execução.

c. Atente na especificação (pré- e pós-condição) para o comportamento da função `count`. Note que a função `count_l` é utilizada na pós-condição, uma vez que não existe notação matemática específica para exprimir a contagem de elementos.

   Escreva um **invariante** e um **variante** de ciclo que permitam provar a correcção total de `count` face a esta especificação (deverá para isso utilizar igualmente `count_l`).

Algoritmos e Complexidade — Universidade do Minho
Exame 8 de Fevereiro de 2022

Número: _____ Nome: _____

2. Estruturas de Dados e Análise de Caso Médio (5 valores)

a. Simule a evolução de uma **min-heap** ao longo da seguinte sequência de operações de inserção e remoção. Deve desenhar a heap que resulta de cada operação.

```
Insert(15) ;
Insert(17) ;
Insert(12) ;
Insert(6) ;
ExtractMin ;
Insert(20) ;
Insert(14) ;
Insert(4) ;
ExtractMin ;
```

b. A função seguinte insere um elemento `x` numa **min-heap** de tamanho `n`, representada num array `a`, através da habitual operação de *bubble-up*. Assume-se que o índice `n` pertence ainda ao array.

```
void insertHeap (Elem a[], int x, int n) {
   int p = (n-1)/2;
   a[n] = x;
  while (n>0 && a[n]<a[p]) {
     swap(a, n, p);
     n = p;
     p = (n-1)/2;
   }
 }
```

Analise o tempo de execução deste algoritmo no **caso médio,** tendo em conta o número de comparações `a[n]<a[p]` efectuadas. Para isso assuma que numa heap a probabilidade de um elemento ser uma folha (i.e. de se encontrar no último nível da árvore) é ½, o que significa que a probabilidade de um novo elemento, escolhido aleatoriamente, ser inserido nesse nível é também de ½.

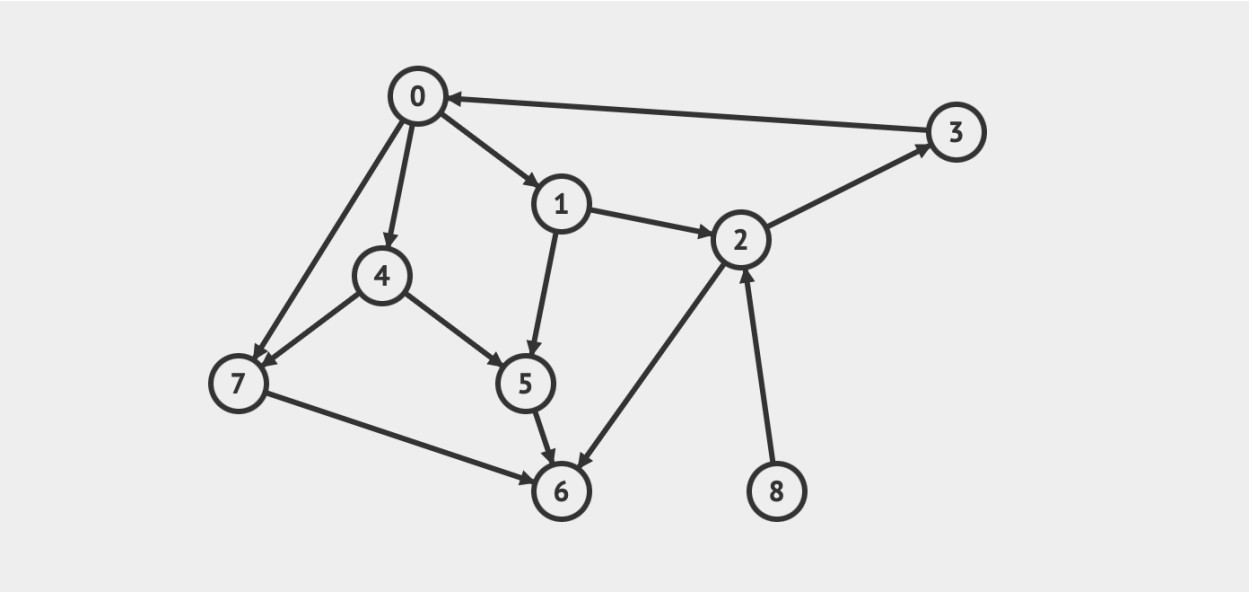
Algoritmos e Complexidade — Universidade do Minho
Exame 8 de Fevereiro de 2022

Número: _____ Nome: _____

3. Grafos (2.5 valores)

Considere o seguinte grafo orientado sem pesos. **Desenhe** a árvore que resulta de uma **travessia em profundidade** feita a partir do vértice 0.

Com base na árvore que construiu, justifique que o algoritmo de travessia em profundidade não pode ser utilizado para encontrar caminhos mais curtos entre pares de vértices.



Algoritmos e Complexidade — Universidade do Minho
Exame 8 de Fevereiro de 2022

Número: _____ Nome: _____

4. Grafos (5 valores)

Considere os seguintes tipos de dados para a representação de grafos por listas de adjacências:

```
struct edge {
  int dest;
  struct edge *next;
};
typedef struct edge *Graph[MAX];
```

a. Defina a função

   ```
   int reachable (Graph g, int x, int y, int n, int V) ;
   ```

   que determina (devolvendo 0 ou 1) se y é alcançável a partir de x **em n passos ou menos**, i.e. por um caminho com comprimento inferior ou igual a n.

b. Um grafo orientado diz-se *fortemente ligado* se para qualquer par de vértices u, v, existem caminhos quer de u para v quer de v para u (i.e. cada vértice é alcançável a partir do outro). Defina a função

   ```
   int stronglyConnected (Graph g, int n);
   ```

   que testa se um grafo orientado (representado por listas de adjacências) é ou não fortemente ligado. Analise o tempo de execução da função no pior caso.