Algoritmos e Complexidade

MIEI, LCC 2º ano

12 de Janeiro de 2019 – Duração: 90 min

1. Considere o problema de, dado um array `v` com `N` elementos determinar o `k`-ésimo maior elemento (com $k < N$), isto é, o elemento que estaria na posição `N-k` caso o array estivesse ordenado por ordem crescente.

   Uma forma de resolver este problema consiste em percorrer o array guardando numa estrutura auxiliar apenas os k maiores elementos lidos até ao momento. No final o elemento pretendido é o menor dos elementos armazenados.

   Defina uma função `int kmaior (int v[], int N, int k)` que implementa o algoritmo descrito, usando como estrutura auxiliar uma *min-heap* (com `k` elementos). Admita que existem disponíveis as funções `void bubbleDown (int h[], int N)` (que faz o *bubble down* do elemento `h[0]` na heap `h` com `N` elementos) e `void bubbleUp (int h[], int N)` (que faz o *bubble up* do elemento `h[N-1]` na heap `h`).

   Identifique o pior caso da solução apresentada e diga, justificando, qual a complexidade dessa solução nesse caso.

2. Apresente a evolução de uma tabela de hash de inteiros (`(x) = x%HSize` e tratamento de colisões por linear probi (por esta ordem) os elementos:

   30, 18, 43, 25, 60, 35,

   Use a simulação feita para calcular o número médio de inserção, neste caso.

0									
1		18	18	18	18	18	18	18	18
2						35	35	35	35
3								20	20
4									
5									
6									
7							40	40	40
8				25	25	25	25	25	25
9			43	43	43	43	43	43	43
10					60	60	60	60	60
11									41
12									
13	30	30	30	30	30	30	30	30	30
14									
15									
16									

3. Considere uma estrutura de dados sobre a qual é executada uma sequência de operações. É sabido que o tempo de execução da n-ésima operação da sequência é dado por $T_{op}(n) = n^2$ quando n é uma potência de 2 (i.e., $n = 2^k$ para algum $k \in \mathbf{N}$), e $T_{op}(n) = 1$ para todos os outros valores de n.

   Efectue a análise amortizada do tempo de execução de uma operação arbitrária desta sequência, utilizando para isso **duas técnicas** à sua escolha.

4. Dado um grafo G um caminho $\langle v_0, v_1, \ldots, v_n \rangle$ diz-se um caminho simples sse não contiver elementos repetidos. Defina uma função `int simplePath (Graph g, int v[], int k)` que testa se a sequência ⟨`v[0],v[1],...,v[k-1]`⟩ é um caminho simples. A função deve verificar que se trata realmente de um caminho e que não tem vértices repetidos.
   Indique, justificando, a complexidade da solução apresentada.

```
#define N ...
typedef struct edge {
    int dest, cost;
    struct edge *next;
} * Graph [N];
```

5. Considere o grafo pesado e não orientado representado na matriz de adjacência à direita (`(-1)` é usado para marcar que **não há** aresta).
   Relembre o algoritmo de Dijkstra para calcular o caminho mais curto entre um par de vértices.

```
int dijkstraSP (GMat g, int o, int d,
                int pesos[], int pais[]);
```

   Apresente a evolução do algoritmo de Dijkstra para calcular o caminho mais curto entre os vértices 2 e 3, i.e., a invocação `dijkstraSP (g1, 2, 3, pesos, pais);`
   Nomeadamente, apresente a evolução dos arrays `pais` e `pesos`.

```
#define N 6
typedef int GMat [N][N];

GMat g1 =
  // 0  1  2  3  4  5
  {{-1, 6, 7,-1, 3,-1}, // 0
   { 6,-1,-1, 5, 4, 2}, // 1
   { 7,-1,-1,-1, 8,-1}, // 2
   {-1, 5,-1,-1,-1, 2}, // 3
   { 3, 4, 8,-1,-1, 3}, // 4
   {-1, 2,-1, 2, 3,-1}} // 5
```

pesos = {6,5,7,-1,-1,-1} pais = {1,3,0,-1,-1,-1}