Algoritmos e Complexidade – Exame

6 de Fevereiro de 2014 – Duração: 150 min

Parte A

1. Considere a seguinte função recursiva:

```
int contaAux (int cont[], int n)  {
  if (n == 0) return 0;
  else if (cont[n] >= n) return n;
       else return contaAux(cont, n-1);
}
```

   Calcule o seu tempo de execução assimptótico no melhor e pior caso, identificando bem esses casos, e apresentando as recorrências e árvores de recursão apropriadas para ambos os casos.

2. Considere a seguinte função que testa se um grafo representado numa matriz de adjacência é **não orientado**.

```
int undirected (int G[MaxV][MaxV]) {
  int i, j;
  for (i=0;i<MaxV;i++)
    for (j=i+1; j<MaxV; j++)
      if (G[i] != G[j]) return 0;
  return 1;
}
```

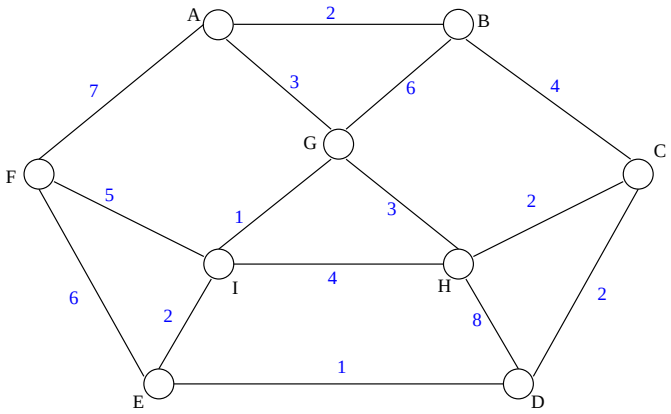
   (a) Identifique o melhor e pior casos da execução da função apresentada e, para o pior caso, calcule o número de comparações entre elementos de `G`.

   (b) Suponha agora que o grafo está representado usando listas de adjacência. Apresente uma alternativa eficiente para a função acima nessa representação. Identifique o custo assimptótico da solução que apresentou, *em termos de tempo e de espaço*, no melhor e no pior caso.

3. Considere uma representação por listas de adjacências de grafos orientados. Escreva uma função que detecte se um dado grafo é ou não cíclico. Para isso, adapte um dos procedimentos de ordenação topológica que conhece.

4. Escreva uma função `replace(int h[], int x, int k, int N)` que recebe uma *minheap* `h` com `N` elementos, e *substitui* o elemento contido na posição `k` pelo valor `x`, realizando naturalmente os ajustes necessários para que o *array* resultante continue a representar uma *minheap*.

5. Considere o seguinte grafo. Simule, apresentando todos os passos, a execução do algoritmo de Prim para a construção de uma sua árvore geradora mínima. Considere o vértice F como inicial.



Parte B

1. Considere o extracto de programa usado para calcular o quociente e resto da divisão inteira entre dois números inteiros e positivos.

```
// x >= 0 && y > 0 && p = 2^N && dy = y*p && dy >= x
q = 0; r = x;
while (N >= 0) {
  if (r >= dy) {
    r = r - dy; q = q + p;
  }
  dy = dy/2; p=p/2; N = N-1;
}
// 0 <= r < y && q * y + r = x
```

   Anote o programa de forma a provar a sua correcção parcial e apresente as condições de verificação correspondentes à prova da preservação do invariante.

2. Considere uma tabela de hash de tamanho N com open addressing e linear probing. Considere ainda que o factor de carga da tabela é 0.5. Assuma que a função de hash é uniforme (i.e., que a probabilidade de $hash(x) = k$ é $1/N$ para todo o k). Calcule o número médio de posições consultadas na tabela aquando da inserção de uma nova chave nas seguintes duas configurações:

   (a) As posições ocupadas correspondem à primeira metade da tabela (i.e., desde a posição 0 até à posição $N/2$).

   (b) As posições ocupadas são as posições pares (i.e., 0, 2, 4, ...).