Teste

Algoritmos e Complexidade
2º Ano – LEI/LCC

13 de Janeiro de 2012 – Duração: 2 horas

Parte I

Esta parte do teste representa 12 valores da cotação total. Cada alínea está cotada em 2 valores. **A não obtenção de uma classificação mínima de 8 valores nesta parte implica a reprovação no teste.**

1. Suponha que se usam vectores de coeficientes para representar polinómios. O polinómio $4.2\,x^5 - 3.2\,x^2 - 0.5$ será representado por um array em que as primeiras seis componentes são `-0.5`, `0`, `-3.2`, `0`, `0` e `4.2`.

   A seguinte função calcula o valor de um polinómio `p` (de grau `n-1`) num ponto.

   ```
   float valor(float p[],int n,float x){
      int i; float r;
      i = n; r = 0;
      while (i>0) {
          i = i-1;
          r = r * x + p[i];
      }
      return r;
   }
   ```

   Para provar que esta função satisfaz a especificação

   pré-condição: $n \geq 0$

   pós-condição: $r = \sum_{k=0}^{n-1} p_k * x^k$
   (em que p_k corresponde ao valor de `p[k]`),

   vamos usar como invariante o predicado

   invariante: $(i \geq 0) \wedge (r = \sum_{k=i}^{n-1} p_k * x^k)$

   Anote correctamente o programa e, a partir do programa anotado, gere as condições de verificação necessárias para a prova da correcção **parcial** desta função (face à especificação fornecida).

2. Considere as seguintes definições para representar uma *min-heap* de inteiros.

   ```
   typedef struct minheap {
      int size;
      int used;
      int values[];
   } *MinHeap;
   ```

   (a) Suponha que numa destas estruturas está guardada uma *min-heap* com tamanho (`size`) 100, com 10 elementos (i.e., `used` tem o valor 10) e que as 10 primeiras posições do vector `values` têm os valores `4, 10, 21, 45, 13, 25, 22, 60, 100, 20`.

   Diga qual o conteúdo desse vector após a inserção do número 6. Justifique a sua resposta desenhando as árvores correspondentes à *min-heap* antes e depois da referida inserção.

   (b) Defina uma função `int minHeapOK (MinHeap h)` que testa se a min-heap está correctamente construída (i.e., se todos os caminhos da raíz até uma folha são sequências crescentes). Certifique-se que a solução que apresentou tem um custo linear no tamanho da input.

3. Considere a seguinte definição de um tipo para representar (as arestas de) um grafo em listas de adjacência.

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

   Defina uma função `int colorOK (Graph g, int color[])` que, dado um grafo não orientado `g` e um vector de inteiros `cor` verifica se essa coloração é válida. Diz-se que uma coloração é válida sse vértices adjacentes tenham cores diferentes.

4. Considere as seguintes definições para representar tabelas de *hash* dinâmicas com tratamento de colisões por *chaining*.

```
typedef struct entry {
   char key[10];
   void *info;
   struct entry *next;
} *Entry;

int hash(int hashsize, int key[]);
```

```
typedef struct hashT {
   int hashsize;
   Entry table[];
} *HashTable;
```

   Defina uma função `HashTable newTable(int hashsize)` que inicializa uma tabela de tamanho `hashsize`. Note que deve ser alocada a memória necessária e que todas as entradas da tabela devem ser inicializadas com a lista vazia.

5. Considere a seguinte função que gera (imprimindo no `stdout`) todas as combinações de N bits.

```
void bitsSeq (int N, char seq[], char *end) {
    if (N==0) {
        *end = 0;
        printf ("%s\n", seq);
    } else {
        *end = '0';
        bitsSeq (N-1, seq, end+1);
        *end = '1';
        bitsSeq (N-1, seq, end+1);
    }
}
```

   Faça a análise assimptótica do seu tempo de execução. Para isso, defina uma recorrência que exprima essa complexidade, desenhe a árvore de recursão e apresente uma solução para a recorrência apresentada.

Parte II

1. Num grafo não orientado e ligado, a **excentricidade** de um vértice define-se como a maior distância entre esse vértice e qualquer outro vértice (no caso de grafos não pesados, a distância entre dois vértices corresponde ao número de arestas do caminho mais curto entre esses vértices). Usando uma variante do algoritmo de travessia *breadth-first*, defina uma função `int excentricity (Grafo g, int v)` que calcula a excentricidade de um vértice num grafo não pesado.

2. Seja `V` um vector com `N` números inteiros diferentes. Uma forma de representar um subconjunto `X` de `V` consiste em usar um vector `x` com `N` valores booleanos em que `x[i] == 1` sse `V[i]` $\in$ `X`.

   Por exemplo para `V = {1,2,3,4,5}` o array `x={1,0,0,1,0}` representa o subconjunto $\{1, 4\}$.

   Suponha que existe a função `int teste(int V[], int N, int x[])` que testa se um determinado subconjunto `x` de `V` satisfaz uma dada propriedade. Assuma ainda que a função executa em tempo linear em `N`.

   (a) Defina uma função `int forall(int V[], int N, int x[])` que testa se a dita popriedade é válida para todos os subconjuntos de `V`. No caso de insucesso, a função deve ainda preencher o vector `x` com um conjunto que não satisfaça a propriedade.

   (b) Identifique o pior caso de execução da função apresentada na alínea anterior e analise a complexidade dessa função para esse caso.

   (c) Suponha que para um determinado problema de decisão sobre conjuntos de números inteiros, se conhece um algoritmo não deterministico que o resolve. Diga como poderia usar a função `forall` para construir um algoritmo deterministico que resolve o problema em causa.