Algoritmos e Complexidade
2º Ano – LEI/LCC

17 de Setembro de 2010 – Duração: 2 horas

Exame – Época Especial

Parte I

Esta parte 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 exame.**

1. Calcule as condições de verificação necessárias à prova de correcção parcial do seguinte programa (anotado em comentário `//`)

```
// s = 0 && i = 0
while (i < N) {
  // s = sum (k=0..i-1) a[k] && i <= N
  s = s + a[i];
  i = i+1;
}
// s = sum (k=0..N-1) a[k]
if (s > 10) res = 1;
else res = 0;
// sum (k=0..N-1) a[k] > 10 <==> res = 1
```

2. Suponha que o ciclo while do programa anterior é alterado para

```
while (i < N) { a[i] = 10;
                s = s + a[i];
                i = i+1; }
```

   Explique porque é que a especificação do programa (anotações da alínea anterior) permanece válida. Apresente uma especificação alternativa para o programa original, que exclua esta solução claramente contra-intuitiva.

3. Considere a seguinte função sobre uma árvore binária, que devolve o elemento alcançável através de um determinado caminho.

```
int walk(TreeNode *root, ListNode *path) {
      if (! root) return (-1);
   if (! path) return (root->val);
   if (path -> direction == LEFT) {
      return walk(root->left,path->next);
      } else {
          return walk(root->right,path->next);
      }
}
```

   Analise o seu tempo de execução, no melhor e no pior caso, tendo em conta que o tamanho do input é dado pelo número de elementos da árvore N e pelo comprimento do caminho L. Utilize equações de recorrência na sua análise.

4. Recorde o que estudou sobre árvores AVL. Implemente a função rdir que efectua uma rotação simples para a direita. A função deverá retornar 0 em caso de sucesso, e -1 no caso de a rotação ser impossível devido à estrutura da árvore. Não é necessário ajustar os factores de balanceamento. Note que a função recebe a árvore passada por referência (struct avlnode **tptr).

```
typedef struct avlnode {
    int value,balance;
    struct avlnode *esq,dir;
} *AVL;

int rdir(AVL *tptr);
```

5. Recorde o que estudou sobre Tabelas de Hash. Suponha que alteramos a implementação do mecanismo de resolução de colisões denominado *chaining*, substituindo as listas ligadas armazenadas em cada posição por árvores binárias de pesquisa. Efectue uma análise do tempo de execução das operações de inserção e pesquisa na tabela de hash, no melhor e no pior caso. Os resultados vão de encontro à sua intuição? Justifique a sua resposta.

6. Dados os seguintes tipos de dados para a representação de grafos orientados,

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

   Escreva uma função `int reachable(Graph g, int source, int n)` que conta os nodos alcançáveis a partir do nó source.

Parte II

Relembre a função de partição usado no algoritmo de *quicksort*.

```
int partition (int v[], int l, int u) {
int i, j;
i = j = l;
while j<u {
   if v[j] < v[u] { swap (v, i, j); i++; }
   j++;}
swap (v,i,u);
return i;
}
```

1. Apresente um invariante e um variante adequados à prova da correcção total desta função, face à seguinte especificação:

   pré-condição $l \leq u$

   pós-condição $(\forall_k . l \leq k < i \Rightarrow v[k] \leq v[i]) \wedge (\forall_k . i < k \leq u \Rightarrow v[i] \leq v[k])$

2. Usando este invariante e variante, apresente uma versão anotada desta função de acordo com as regras de anotação estudadas.

3. A função de partição apresentada pode ser usada para definir uma função que determina qual o k-ésimo menor elemento de um vector:

   > Se o resultado `p` de invocar esta função for menor do que `k`, esse elemento estará antes da posição `p`; se for maior estará depois de `p`; finalmente se for igual está encontrado o valor pretendido.

   Apresente uma definição da função `int kesimo (int v[], int n, int k)` que calcula o `k`-ésimo elemento do vector `v` (com `n` elementos).

4. Assumindo que o valor de retorno da função `partition (v,l,u)` é `(l+u)/2` apresente uma relação de recorrência que caracterize a complexidade desta função em função do número de elementos do vector.

Algumas Regras em Lógica de Hoare

Atribuição

$$\frac{P \Rightarrow (Q[x \setminus E])}{\{\, P \,\}\ x = E\ \{\, Q \,\}} \quad \text{(Atrib2)}$$

Sequência

$$\frac{\{\, P \,\}\ S_1\ \{\, R \,\} \quad \{\, R \,\}\ S_2\ \{\, Q \,\}}{\{\, P \,\}\ S_1 ;\ S_2\ \{\, Q \,\}} \quad (;)$$

Ciclo

$$\frac{P \Rightarrow I \quad \{\, I \wedge c \,\}\ S\ \{\, I \,\} \quad (I \wedge \neg c) \Rightarrow Q}{\{\, P \,\}\ \texttt{while}\ c\ S\ \{\, Q \,\}} \quad \text{(while-2)}$$