Algoritmos e Complexidade

1º Teste

28 de Novembro de 2012 – Duração: 90 min

Parte I

1. Apresente as condições de verificação necessárias à prova da correcção parcial do seguinte programa anotado que calcula o número representado num array de (N) bits.

```
// N >= 0
n = 0; i = 0;
// N>=0 && n == 0 && i == 0
while (i<N) {
    // n = sum{k=0}{k=i-1} b[k] * (2^k)  && i<=N
    n = n*2 + b[i];
    i = i+1;
}
// n = sum{k=0}{k=N-1} b[k] * (2^k)
```

2. Defina uma função `break_list` que parta uma lista ligada em dois segmentos de igual comprimento, devolvendo o endereço da lista correspondente ao segundo. Se a lista inicial tiver comprimento ímpar, o segundo segmento terá mais um elemento do que o primeiro. Por exemplo, se a lista original tiver os elementos `1,2,3,4,5,6,7`, após a execução da função, a lista passa a ter apenas os valores `1,2,3` e o resultado da função deverá ser a lista com os elementos `4,5,6,7`.

   A função a definir só terá que produzir resultados válidos para listas de comprimento superior ou igual a 2.
   Certifique-se que a solução apresentada executa em tempo linear no tamanho da lista e que não aloca memória adicional.

```
typedef struct lnode {
  int info;
  struct lnode *next;
} Lnode, *List;
```

3. Escreva uma recorrência e diga qual o tempo de execução assimptótico da função seguinte, que constrói uma árvore binária a partir de uma lista ligada de inteiros (usando a função `break_list` da alínea anterior).

```
typedef struct tnode {
  int info;
  struct tnode *left, *right;
} Tnode, *BTree;
```

```
Tnode* mkTree (Lnode *l) {
  Tnode *new;
  Lnode *l2;
  if (!l) return NULL;
  new = malloc(sizeof(Tnode));
  if (!l->next) {
    new -> left = new -> right = NULL;
    new -> info = l -> info;
    free (l); return new;
  }
  l2 = break_list(l);
  new -> info = l2 -> info;
  new -> left = mkTree (l);
  new -> right = mkTree (l2->next);
  free (l2); return new;
}
```

Parte II

1. Considere o seguinte programa, parcialmente anotado, para calcular a representação binária de um número inteiro positivo.

```
// n == n0 >= 0
i = 0;
// n == n0 >= 0 && i == 0
while (n > 0) {
    // ...
    b[i] = n%2;
    i = i+1; n = n/2;
}
// n0 = sum{k=0}{k=i-1} b[k] * (2^k)
```

   Determine um invariante que lhe permita provar a correcção parcial deste excerto de código. Mostre que o invariante que apresentou é preservado (em cada iteração).

2. Relembre a seguinte função de consulta de uma árvore binária de procura:

```
int elem (BTree a, int x) {
    while (a != NULL)
        if (a->info == x) break;
        else if (a->valor > x) a = a->left;
        else a = a->right;
    return (a != NULL)
}
```

   Admitindo que se trata de uma árvore perfeitamente balanceada,

   (a) Determine o tempo médio de execução desta função, no caso de o elemento pertencer à árvore. Admita que o valor a procurar está com igual probabilidade em qualquer posição da árvore. Note que uma árvore balanceada com N nodos tem aproximadamente $\log_2 N$ níveis.

   (b) Calcule o tempo de execução desta função no caso de insucesso (i.e., no caso de o elemento não existir na árvore). O que pode concluir sobre o comportamento assimptótico médio desta função?