TERCEIRA PROVA DE PRINCÍPIOS DE DESENVOLVIMENTO DE ALGORITMOS
BCC, 2o. SEMESTRE DE 2006

Instruções:

1. Não destaque as folhas do caderno de soluções.
2. A prova pode ser feita a lápis. Cuidado com a legibilidade.
3. Não é permitido o uso de folhas avulsas para rascunho.
4. Não é necessário apagar rascunhos no caderno de soluções.
5. Asserções imprecisas valem pouco. Justifique suas asserções (dentro do razoável!).

1. [3 pontos] Suponha que estamos utilizando uma tabela de hashing com `M` entradas, com resolução de colisões por encadeamento (as `M` listas não são mantidas em ordem). Suponha que usamos a função de hashing que devolve $11k \bmod \texttt{M}$ quando a entrada é a k-ésima letra do alfabeto (por exemplo, `C` corresponde a $k = 3$). Suponha que inserimos em uma tabela inicialmente vazia, nesta ordem, as chaves

   `E A S Y Q U T I O N`

   onde $\texttt{M} = 5$. Desenhe diagramas que ilustram este processo de inserção.

2. [4 pontos] Esta questão trata de árvores binárias. Suponha que temos

```
typedef struct node *link;
struct node { Item item; link l, r; int N; };
/* item nao sera usado nesta questão */
/* N armazena o número de nós na árvore enraizada naquele nó */
/* Por exemplo, N = 1 nos nós sem filhos */
```

   Suponha ainda que não temos nós externos em nossas árvores (a árvore vazia é representada com o ponteiro `head = NULL`). A *profundidade* $\text{prof}(x)$ de um nó x em uma árvore com raiz r é a 'distância' de r a x na árvore. Alternativamente, a profundidade de um nó x é definida como sendo 0 se $x = r$ e, se $x \neq r$, então $\text{prof}(x) = \text{prof}(x') + 1$, onde x' é o pai de x na árvore. O *comprimento interno* I de uma árvore T é definido como

$$I = \sum_x \text{prof}(x), \tag{1}$$

   onde a soma é sobre todo nó x em T.

   (i) Desenhe todas as árvores binárias com 4 nós (são 14).

   (ii) Determine o comprimento interno de todas as árvores em (i) (várias delas têm $I = 6$).

   (iii) Escreva uma função em C que recebe uma árvore (isto é, um ponteiro para a raiz de uma árvore) e que devolve o comprimento interno dessa árvore. Sua função deve ter complexidade de tempo proporcional ao número de nós na árvore.

3. [4 pontos] Descreva o projeto de um programa que recebe uma cadeia de caracteres T como entrada e devolve o trecho repetido mais comprido neste texto. Por exemplo, se o texto de entrada for `aacabcabcbabcbabac`, então a saída de seu programa deve ser `abcbab`.

   Faça uma descrição cuidadosa de seu projeto de programa: descreva a estrutura de dados a ser usada e dê os protótipos das funções principais, com uma descrição de o que estas funções devem fazer e como elas poderiam ser implementadas (não esqueça do `main()`).

   O seu projeto deve supor que a entrada T pode ser grande; por exemplo, T poderia um livro todo. Um programa implementado de acordo com o seu projeto deve ser tal que

   (i) sob hipóteses razoáveis, ele leva tempo basicamente proporcional a $n \log n$, onde n é o número de caracteres em T,

   (ii) ele gasta uma quantidade de memória proporcional ao número de caracteres em T.

   Você deve argumentar por que o programa teria estas propriedades (em particular, você deve explicitar as hipóteses que você usa em sua análise). [*Sugestão.* Um *sufixo* de uma cadeia de caracteres T é um 'segmento final' dela. Os sufixos de `abcde` são `abcde`, `bcde`, `cde`, `de`, `e`, e a cadeia vazia, com 0 caracteres (6 sufixos no total). Considere os sufixos de T.]