TERCEIRA PROVA DE PRINCÍPIOS DE DESENVOLVIMENTO DE ALGORITMOS
BE/BM/BMA, 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 inserimos, nesta ordem, as chaves

   `Q U E S T A O F C I L`

   em uma árvore binária de busca (ABB) inicialmente vazia. Desenhe as ABBs resultantes após cada inserção.

2. [5 pontos] Esta questão trata de árvores binárias. Suponha que temos

```
typedef struct node *link;
struct node { Item item; link l, r; int N; };
/* N armazena o número de nós (internos) na árvore enraizada naquele nó */
/* Por exemplo, N deve ser 11 na raiz de sua árvore da Questão 1 */

link rotR(link h)
  { link x = h->l; h->l = x->r; x->r = h;
    return x; }

link rotL(link h)
  { link x = h->r; h->r = x->l; x->l = h;
    return x; }

Item selectR(link h, int k)
  { int t = h->l->N;
    if (h == z) return NULLitem;
    if (t > k) return selectR(h->l, k);
    if (t < k) return selectR(h->r, k-t-1);
    return h->item;
  }
```

```
link partR(link h, int k)
  { int t = h->l->N;
    if (t > k )
      { h->l = partR(h->l, k); h = rotR(h); }
    if (t < k )
      { h->r = partR(h->r, k-t-1); h = rotL(h); }
    return h;
  }
```

O código acima supõe que estamos adotando as várias convenções que temos seguido nas aulas. Suponha que `head` aponta para a raiz de sua árvore da Questão 1.

(i) Desenhe o resultado de `rotR(head)` e `rotL(head)`. Indique em seus diagramas as novas raízes nessas árvores.

(ii) Desenhe a árvore que temos depois de executar a atribuição

```
head->l = rotL(head->l);
```

(iii) Qual é o resultado da execução de `selectR(head, 6)`?

(iv) Qual é o resultado da execução de `partR(head, 6)`?

3. [4 pontos] Dizemos que uma palavra s é um *anagrama* de uma palavra t se s pode ser obtida de t pela permutação de suas letras. Por exemplo, as palavras *triangle*, *relating*, *integral*, *altering*, e *alerting* são todas anagramas umas das outras. Descreva *cuidadosamente* o projeto de um programa que resolve o seguinte problema: a entrada é um conjunto D de palavras e a saída deve ser um conjunto S de anagramas com $S \subset D$. Ademais, o conjunto S deve ser de cardinalidade máxima.

   O seu projeto deve supor que a entrada D pode ser grande; por exemplo, D poderia ser o conjunto de palavras de um dicionário. Um programa implementado de acordo com o seu projeto deve ser tal que

   (i) sob hipóteses razoáveis, ele leva tempo basicamente proporcional ao número de palavras em D,

   (ii) ele gasta uma quantidade de memória proporcional ao número de palavras em D.

   Você deve argumentar por que o programa teria estas propriedades (em particular, você deve explicitar as hipóteses que você usa em sua análise).

   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()`).

   [*Sugestão.* Chame de *assinatura* de uma palavra s a cadeia de caracteres que obtemos ao ordenar as letras em s. Por exemplo, a assinatura de ‘assinatura’ é ‘aaainrsstu’. Use este conceito.]