Exame de Programação Imperativa

LCC/MIEF/MIEI

Época Especial – 4 de Setembro de 2020

Parte A

1. Defina uma função `int maiorPrefixo (char s1 [], char s2 [])` que calcula o comprimento do maior prefixo comum entre as duas strings.

2. Defina uma função `int remRep (char x[])` que elimina de uma string todos os caracteres que se repetem sucessivamente deixando lá apenas uma cópia. A função deverá retornar o comprimento da string resultante. Assim, por exemplo, ao invocarmos a função com uma vector contendo `"aaabaaabbbaaa"`, o vector deve passar a conter a string `"ababa"` e a função deverá retornar o valor `5`.

3. Considere o tipo `LInt` para representar listas ligadas de inteiros.

```
typedef struct lligada {
    int valor;
    struct lligada *prox;
} *LInt;
```

   Defina uma função `void splitQS (LInt l, int x, LInt *mx, LInt *Mx)` que, dada uma lista ligada `l` e um inteiro `x`, parte a lista em duas (retornando os endereços dos primeiros elementos das listas em `*mx` e `*Mx`): uma com os elementos de `l` menores do que `x` e a outra com os restantes. Note que esta função não deverá criar cópias dos elementos da lista.

4. Apresente uma definição da função `int removeDups (LInt *)` que remove os valores repetidos de uma lista (deixando apenas a primeira ocorrência).

Parte B

Relembre o problema de guardar para cada palavra de um texto o número de vezes que palavra ocorre.
Uma das soluções estudadas consistia em usar uma lista na qual, sempre que uma palavra ocorria, o nodo da lista correspondente era movido para o início da lista.
Uma outra solução estudada consistia em usar uma árvore binária de procura (ordenada pela palavra).
Neste problema vamos combinar estas duas estratégias, usando uma árvore binária de procura em que o acesso a uma dada palavra faz com que ela seja movida para a raíz da árvore.

```
typedef struct dict {
    char *palavra;
    int ocorr;
    struct dict *esq, *dir;
} *Dict;
```

1. Considere a função ao lado que faz uma *rotação à direita* de uma árvore.

   - Identifique os casos para os quais esta função não está definida, originando um acesso inválido à memória.
   - Apresente um exemplo de funcionamento desta função. Para isso comece por apresentar uma árvore com altura 3 para a qual esta função esteja definida e apresente o resultado de invocar a função com essa árvore.

```
Dict rodaDireita (Dict d) {
    Dict r = d->esq;
    d->esq = r->dir;
    r->dir = d;
    return r;
}
```

2. Defina a função `Dict rodaEsquerda (Dict d)` inversa da anterior, que faz uma *rotação à esquerda* de uma árvore. Mais uma vez, identifique os casos em que essa função não está definida.

3. As funções `rodaDireita` e `rodaEsquerda` mantêm a ordenação da árvore, *promovendo* um dos nodos do nível 2 para a raíz (nível 1).

   Usando repetidamente a função `rodaDireita` defina uma função `Dict promoveMenor (Dict d)` que coloca o menor elemento da árvore na raíz (continuando a árvore a estar ordenada). Serão valorizadas soluções que não aumentem a altura da árvore em mais do que uma unidade.

   Apresente o resultado de aplicar a função que definiu ao exemplo apresentado na alínea 1.

4. Defina a função `Dict acrescenta (Dict d, char *pal)` que regista uma nova ocorrência da palavra `pal` no dicionário `d`. A função deve garantir que a árvore resultante tem como raíz a palavra `pal`. **Sugestão:** Use as funções `rodaEsqueda` e `rodaDireita` para promover o nodo correspondente para o nível 1 (raíz).