Programação Imperativa – Teste

8 de Junho de 2016 – Duração: 2h00m

Parte A

1. Apresente uma definição da função pré-definida em C `char *strcat (char s1[], char s2[])` que concatena a string s2 a s1 (retornando o endereço da primeira).

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. Defina uma função `int nivelV (ABin a, int n, int v[])` que preenche o vector `v` com os elementos de `a` que se encontram no nível `n`. Considere que a raíz da árvore se encontra no nível `1`.
   A função deverá retornar o número de posições preenchidas do array.

```
typedef struct nodo {
   int valor;
   struct nodo *esq, *dir;
} *ABin;
```

4. Apresente uma definição não recursiva da função `int addOrd (ABin *a, int x)` que adiciona um elemento a uma árvore binária de procura. A função deverá retornar `1` se o elemento a inserir já existir na árvore (e nesse caso a árvore não é alterada) ou `0` no outro caso.

Parte B

Existem domínios de aplicação onde se faz uso de matrizes de grandes dimensões em que a maior parte das respectivas entradas é "0" (matrizes esparsas). Nessas condições, é normalmente preferível armazenar apenas as entradas não nulas da matriz. Para tal considere os seguintes tipos de dados:

```
typedef struct listaC {
   int coluna;
   float valor;
   struct listaC *prox;
   } *Colunas;
```

```
typedef struct listaL {
   int linha;
   Colunas lcol;
   struct listaL *prox;
} *Mat;
```

Uma matriz é assim representada como uma lista de linhas, onde cada uma das entradas dessa lista contém por sua vez uma lista de colunas. Considere ainda que todas as listas (`Mat` e `Colunas`) estão ordenadas por ordem crescente do valor respectivo (`linha` e `coluna`). Além disso, todas as funções que constroem ou modificam matrizes **têm que manter essas listas ordenadas**. (obs: na resolução das questões que se seguem, defina as funções auxiliares que entenda necessárias)

1. Defina a função `float getEntry (Mat m, int linha, int coluna)` que retorna a entrada solicitada na matriz (obs: note que o valor das entradas que não existam em `m` é implicitamente 0).

2. Defina a função `void setEntry (Mat *m, int linha, int coluna, float valor)` que insere uma nova entrada na matriz (ou altera o valor dessa entrada, se ela já existir).

3. Defina a função `void addTo (Mat *m1, Mat m2)` que adiciona à matriz `*m1` a matriz `m2`.

4. Defina a função `void transpose (Mat *m)` que transpõe a matriz `*m`.