UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E COMPUTAÇÃO

SCC 202 - Algoritmos e Estruturas de Dados I

Lista 4 de Exercícios
(Listas Generalizadas, Listas Cruzadas)

1. Sobre listas generalizadas, responda os seguintes itens:
(a) Quais suas vantagens?
(b) Quais suas desvantagens?
(c) Dê exemplos de uso.

2. Faça uma função recursiva que determine se duas listas generalizadas são iguais em termos de estrutura.

3. Mostre graficamente a representação em lista dos seguintes polinômios e listas generalizadas:
(a) $P(x, y, z) = 7y^2z^3 + 6x^3 – 5yz^2 + 2z^3 + 3x – 1$;
(b) $P(x, y) = 10x^2y^2 – 8xy^2 + 7x^2y + 5x^2 + 3y + 2$;
(c) *A = (a, (b, c), (d, (e)))*;
(d) *B = (f, (b, c), ((a)))*;
(e) *C = (A, B, C)*, onde *A, B* e *C* não são átomos, mas as listas deste item e dos anteriores.

4. Desenvolva uma função para avaliar um polinômio em 3 variáveis, ou seja, dados *P(x, y, z)* e *a*, *b* e *c*, calcular o valor de *P(a, b, c)*. Pense como a representação de listas generalizadas pode ajudar nesta solução.

5. Refaça o exercício anterior, generalizando para *n* variáveis.

6. Escreva um procedimento recursivo — *inv(l)* — que inverte uma lista generalizada não-recursiva e todas as suas sub-listas.

Por exemplo: *inv((a, (b, c), d)) = (d, (c, b), a)*.

7. Considerando a representação de matrizes esparsas proposta em aula (**listas cruzadas**), pede-se (na forma de *TAD*):

(a) um procedimento que converta uma matriz X_{mxn} "tradicional" para a nova representação (os valores não nulos de X são introduzidos na forma de triplas (linha, coluna, valor));
(b) um procedimento que elimina (zera) o elemento x_{ij};
(c) um procedimento de "leitura" de elemento, retornando x_{ij}, dados *X*, *i* e *j*;
(d) um procedimento de "escrita" de elemento, fazendo $x_{ij} \leftarrow val$, dados *X*, *i*, *j* e *val*;
(e) um procedimento que some duas matrizes de mesma ordem, resultando uma terceira matriz;
(f) idem para multiplicar duas matrizes

8a . Uma maneira eficiente de representar estruturas com tamanho variável e/ou desconhecido é através de alocação encadeada, utilizando listas. Vamos usar esta representação para armazenar as matrizes esparsas. **Cada coluna da matriz será representada por uma lista linear circular com nó cabeça. Da mesma maneira, cada linha da matriz também será representada por uma lista linear circular com nó cabeça.** Cada célula da estrutura, além dos nós-cabeça, representará os termos diferentes de zero da matriz e devem ter o seguinte tipo (Pascal):

```
type
                Apontador  = ^ TipoCelula;

                TipoCelula = record
                                        Direita,
                                        Abaixo                  : Apontador;
                                        Linha,
                                        Coluna                   : integer;
                                        Valor           : real;
                             end;
```

ou em C:

```
typedef struct MatrAux {
  int linha, coluna;
  float valor;
  struct MatrAux *direita, *abaixo;
} RegMatriz, *ImplMatriz;
```

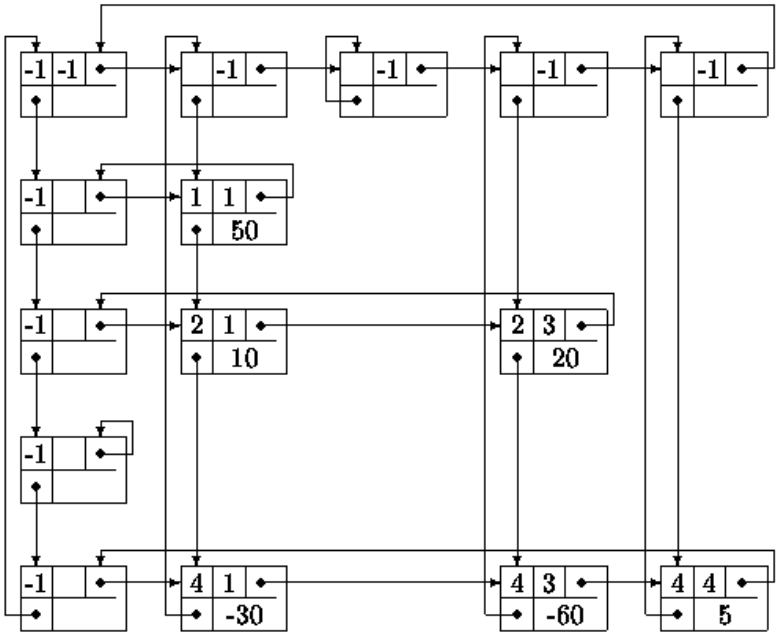
Onde: *RegMatriz* é uma estrutura que armazena um elemento da matriz (ou um dos nós sentinelas; e *direita, esquerda e ImplMatriz* são ponteiros para essa estrutura (RegMatriz), sendo que o ponteiro para o nó (super-)cabeça representará toda a matriz.

O campo **Abaixo** deve ser usado para apontar o próximo elemento diferente de zero na mesma coluna. O campo **Direita** deve ser usado para apontar o próximo elemento diferente de zero na mesma linha. Dada uma matriz A, para um valor A(i,j) diferente de zero, deverá haver uma célula com o campo **Valor** contendo A(i,j), o campo **Linha** contendo i e o campo **Coluna** contendo j. Esta célula deverá pertencer à lista circular da linha i e também deverá pertencer à lista circular da coluna j. Ou seja, cada célula pertencerá a duas listas ao mesmo tempo. Para diferenciar as células cabeça, coloque -1 nos campos Linha e Coluna destas células. Considere a matriz esparsa seguinte:

$$A = \begin{pmatrix} 50 & 0 & 0 & 0 \\ 10 & 0 & 20 & 0 \\ 0 & 0 & 0 & 0 \\ -30 & 0 & -60 & 5 \end{pmatrix}$$

A representação da matriz *A* pode ser vista na Figura 1.

Figura 1: Exemplo de Matriz Esparsa



Com esta representação, uma matrix esparsa $m \times n$ com *r* elementos diferentes de zero gastará (*m* + *n* + *r*) células. É bem verdade que cada célula ocupa vários bytes na memória; no entanto, o total de memória usado será menor do que as $m \times n$ posições necessárias para representar a matriz toda, desde que *r* seja suficientemente pequeno.

Dada a representação acima, desenvolva cinco procedimentos (funções que retornam void) em C, conforme especificação abaixo:

- **procedure** LeMatriz (**var** A: Matriz); Este procedimento lê de algum arquivo de entrada os elementos diferentes de zero de uma matriz e monta a estrutura especificada acima. Considere que a entrada consiste dos valores de *m* e *n* (número de linhas e de colunas da matriz) seguidos de triplas (*i*, *j*, *valor*) para os elementos diferentes de zero da matriz. Por exemplo, para a matriz acima, a entrada seria:

```
4,  4
1,  1,    50.0
2,  1,    10.0
2,  3,    20.0
4,  1,   −30.0
4,  3,   −60.0
4,  4,     5.0
```

- **procedure** ApagaMatriz (**var** A: Matriz); Este procedimento devolve todas as células da matriz *A* para a área de memória disponível (use a **procedure** *dispose*).

- **procedure** SomaMatriz (**var** A, B, C: Matriz); Este procedimento recebe como parâmetros as matrizes *A* e *B*, devolvendo em *C* a soma de *A* com *B*.

- **procedure** MultiplicaMatriz (**var** A, B, C: Matriz); Este procedimento recebe como parâmetros as matrizes *A* e *B*, devolvendo em *C* o produto de *A* por *B*.

- **procedure** ImprimeMatriz (**var** A: Matriz); Este procedimento imprime (uma linha da matriz por linha da saída) a matriz *A*, *inclusive* os elementos iguais a zero.

Para inserir e retirar células das listas que formam a matriz, crie procedimentos especiais para este fim. Por exemplo, um procedimento

```
procedure Insere (i, j: integer; v: real; var A: Matriz);
```

para inserir o valor *v* na linha *i*, coluna *j* da matriz *A* será útil tanto na **procedure** LeMatriz quanto na **procedure** SomaMatriz. As matrizes a serem lidas para testar os procedimentos são:

- A mesma da Figura 1 deste enunciado;

- $\begin{pmatrix} 50 & 30 & 0 & 0 \\ 10 & 0 & -20 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -5 \end{pmatrix}$

- $\begin{pmatrix} 3 & 0 & 0 \\ 0 & -1 & 0 \end{pmatrix}$

Os procedimentos deverão ser testados com o seguinte programa:

```
program TestaMatrizesEsparsas;
...
...
begin
                ...
        LeMatriz (A); ImprimeMatriz (A);
        LeMatriz (B); ImprimeMatriz (B);
        SomaMatriz (A, B, C); ImprimeMatriz (C); ApagaMatriz (C);
        MultiplicaMatriz (A, B, C); ImprimeMatriz (C);
        ApagaMatriz (B); ApagaMatriz (C);
        LeMatriz (B);
        ImprimeMatriz (A); ImprimeMatriz (B);
        SomaMatriz (A, B, C); ImprimeMatriz (C);
        MultiplicaMatriz (A, B, C); ImprimeMatriz (C);
        MultiplicaMatriz (B, B, C);
        ImprimeMatriz (B); ImprimeMatriz (B); ImprimeMatriz (C);
        ApagaMatriz (A); ApagaMatriz (B); ApagaMatriz (C);
                ...
end. { TestaMatrizesEsparsas }
```

8 b) Implementem matrizes esparsas com listas circulares com nós cabeças.
Para este exercício serão fornecidos três arquivos:
esparsas.h: contendo o cabeçalho dos métodos e o tipo **Matriz** que será utilizada para o gerenciamento de matrizes esparsas.
esparsas.c: arquivo contendo a ED do tipo Matriz e a implementação dos métodos especificados em *esparsas.h* e, potencialmente, métodos auxiliares.
testaesparsas.c: arquivo principal (a partir do qual será gerado o progama executável) que utiliza os métodos implementados em esparsas.c para a leitura, soma e impressão de matrizes esparsas.

Dica: Utilize os valores linha e coluna do nó super-cabeça para armazenar a ordem da matriz.

```
/* Arquivo esparsas.h */

typedef RegMatriz* Matriz;

Matriz IniciaMatriz(int m, int n);
/* Aloca, inicializa e devolve uma matriz com 'm' linhas e
   'n' colunas.  Aproveita os campos do nó super-cabeça para
   guardar estas dimensões */

void LiberaMatriz(Matriz a);
/* Libera o espaço da matriz 'a' */

void AtribuiMatriz(Matriz a, int i, int j, float x);
/* Simula 'a[i][j] = x' */

float ValorMatriz(Matriz a, int i, int j);
/* Devolve o equivalente a 'a[i][j]' */

void OrdemMatriz(Matriz a, int *m, int *n);
/* Retorna as dimensões de 'a' em 'm' e 'n' */

int NumeroNos(Matriz a);
/* Devolve o número total de nós alocados para representar
   a matriz 'a' */

int TamanhoNo();
/* Devolve o tamanho de um nó usado na representação,
   em bytes */

Matriz SomaMatriz(Matriz a, Matriz b) ;
/* Devolve a+b; resultado nulo 'NULL' para ordens incompatíveis */
/* Esta implementação não precisa ser eficiente (isto é, não precisa levar
em consideração o fato das matrizes serem esparsas nem a sua  implementação
*/

/* Arquivo esparsas.c */

#include "esparsas.h"
#include <malloc.h>

typedef struct MatrAux {
  int linha, coluna;
  float valor;
  struct MatrAux *direita, *abaixo;
} RegMatriz, *ImplMatriz;


ImplMatriz InsereMatriz(ImplMatriz pL, ImplMatriz pC, int lin, int col,
float val) {
/* Insere um nó como sucessor de 'pL' (linha) e 'pC' (coluna) , com os
   respectivos campos; predecessores vazios indicam nó-cabeça da lista;
devolve o apontador para o nó criado. */
  /* Completar! */
} /* InsereMatriz */

void RemoveMatriz(ImplMatriz pL, ImplMatriz pC) {
/* Remove e libera o nó sucessor de 'pL' (linha) e e 'pC'
   (coluna) */
```

```
  /* Completar! */
} /* RemoveMatriz */


Matriz IniciaMatriz(int m, int n){
/* Aloca, inicializa e devolve uma matriz com 'm' linhas e
   'n' colunas.  Aproveita os campos do nó super-cabeça para
   guardar estas dimensões */

  int i;
  ImplMatriz p;
  ImplMatriz mat=InsereMatriz(NUL,NUL,m,n,0.0);

  p = mat;
  for (i=1;i<=m;i++)
    p = InsereMatriz(NUL,p,i,-1,0.0);

  p = mat;
  for (i=1;i<=n;i++)
    p = InsereMatriz(p,NUL,-1,i,0.0);

  return (Matriz) mat;

} /* IniciaMatriz */

void LiberaMatriz(Matriz a) {
/* Libera todos os nós da matriz 'a' */

  /* Completar! */

} /*  LiberaMatriz */

void AtribuiMatriz(Matriz a, int i, int j, float x){
/* Simula 'a[i][j] = x' */

  ImplMatriz p = (ImplMatriz) a;
  ImplMatriz q = (ImplMatriz) a;
  ImplMatriz pp, qq;
  int k;

  for (k=1;k<=i;k++)
    p = p->abaixo;

  for (k=1;k<=j;k++)
    q = q->direita;

  do {
    pp = p;  p = p->direita;
  } while ((p->coluna<j) && (p->coluna!=-1));
  do {
    qq = q; q = q->abaixo;
  } while ((q->linha<i) && (q->linha!=-1));

  if ((p->coluna)==j)
    if (x!=0.0) {
      p->valor = x;
      return;
    }
    else {
      RemoveMatriz(pp,qq);
      return;
    }
```

```
  else
    if (x!=0.0) {
      InsereMatriz(pp,qq,i,j,x);
      return;
    }
} /* AtribuiMatriz */

float ValorMatriz(Matriz a, int i, int j) {
/* Devolve o equivalente a 'a[i][j]' */

  ImplMatriz p = (ImplMatriz) a;
  int k;

  for (k=1;k<=i;k++)
    p = p->abaixo;
  do {
    p = p->direita;
  } while ((p->coluna<j) && (p->coluna!=-1));
  if (p->coluna==j)
    return p->valor;
  else
    return 0.0;
} /* ValorMatriz */

void OrdemMatriz(Matriz a, int *m, int *n) {
/* Retorna as dimensões de 'a' em 'm' e 'n' */
  /* Completar! */
} /* OrdemMatriz */

int NumeroNos(Matriz a) {
/* Devolve o número total de nós alocados para representar
   a matriz 'a', incluindo as cabeças das listas */
  /* Completar! */
} /* NumeroNos */

int TamanhoNo() {
    return sizeof(RegMatriz);
} /* TamanhoNo */

Matriz SomaMatriz(Matriz a, Matriz b) {
/* Devolve a+b; resultado nulo 'NULL' para ordens incompatíveis */
/* Esta implementação não precisa ser eficiente (isto é, não precisa levar
   em consideração o fato das matrizes serem esparsas nem a sua
implementação */

  /* Completar! */

} /* SomaMatriz */

/* Arquivo testaesparsas.c */
#include <stdio.h>
#include "esparsas.c"

void LeMatriz(Matriz *a) {
/* Inicializa e lê os elementos de 'a' */
  int m,n,i,j;
  float x;
  scanf("%d %d\n",&m,&n);
  *a = IniciaMatriz(m,n);
  do {
   scanf("%d %d %f\n",&i,&j,&x);
   if (i!=0)
```

```
      AtribuiMatriz(*a,i,j,x);
  } while (i!=0);
}

void EscreveMatriz(Matriz a) {
  int m,n,i,j;
  float x;
  OrdemMatriz(a,&m,&n);
  printf("%3d %3d\n",m,n);
  for (i=1;i<=m;i++)
    for (j=1;j<=n;j++) {
      x = ValorMatriz(a,i,j);
      if (x!=0.0)
        printf("%3d %3d %6.1f\n",i,j,x);
    }

}

float Eficiencia(Matriz a) {
/* Calcula a eficiência da representação esparsa em
   comparação com a usual, em percentagem */
  int m,n;
  OrdemMatriz(a,&m,&n);
  if (m*n!=0)
    return 100.0*(NumeroNos(a)*TamanhoNo())
                 /(m*n*sizeof(float));
  else
    return 0;
}

int main() {
  Matriz a,b,c;
  LeMatriz(&a);
  EscreveMatriz(a);
  printf("Eficiencia: %6.1f%%\n",Eficiencia(a));
  LeMatriz(&b);
  EscreveMatriz(b);
  printf("Eficiencia: %6.1f%%\n",Eficiencia(b));
  c = SomaMatriz(a,b);
  if (c==NULL)
    printf("Soma de matrizes incompativeis\n");
  else {
    EscreveMatriz(c);
    printf("Eficiencia: %6.1f%%\n",Eficiencia(c));
    LiberaMatriz(c);
  }
  LiberaMatriz(a);
  LiberaMatriz(b);
  return 0;
}
```

9. Usando a estrutura de dados do exercício 7 ou 8 para armazenar matrizes esparsas quadradas NxN, escreva funções para verificar se uma dada matriz é:
a. Simétrica
b. Diagonal
c. Triangular superior

10. Para a mesma estrutura de dados do exemplo anterior faça rotinas que informem a linha e a coluna da matriz com maior valor