Trabalho sobre Matrizes Esparsas

Estruturas de Dados I
Prof. Roney Pignaton da Silva
Data de Entrega: 30/04/2019
Meio: enviar arquivos para roney.silva@ufes.br

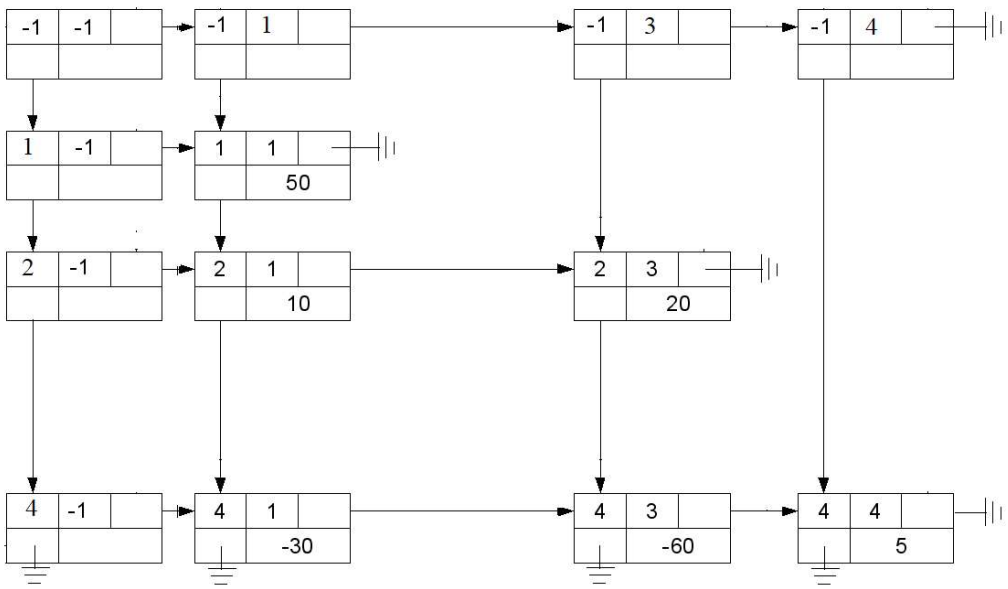
Matrizes esparsas são matrizes nas quais a maioria das posições é preenchida por zeros. Para estas matrizes, podemos economizar um espaço significativo de memória se apenas os termos diferentes de zero forem armazenados. As operações usuais sobre estas matrizes (somar, multiplicar, inverter, etc) também podem ser feitas em tempo muito menor se não armazenadas as posições que contém zeros. Uma maneira eficiente de representar estruturas com tamanho variável e/ou desconhecido é com o emprego de alocação encadeada, utilizando listas. Vamos usar esta representação para armazenar as matrizes esparsas. Cada coluna da matriz será representada por uma **lista linear** com uma **célula cabeça**. Da mesma maneira, cada linha da matriz também será representada por uma lista linear com uma célula cabeça. Cada célula da estrutura, além das células cabeça, representará os termos diferentes de zero da matriz e devem ser como na listagem abaixo:

```
typedef struct Tcelula
{
      struct TCelula *Direita, *Abaixo;
      int Linha, Coluna;
      double Valor;
}Celuna;
```

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 dessas células.
Considere a matriz esparsa seguinte:

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

Ela pode ser representada da seguinte forma:



Com esta representação, uma matriz esparsa $m \times n$ com r elementos diferentes de zero gastará (1+ r + x) células, sendo x o número de linhas mais o número de colunas com elementos diferentes de zero na matriz. 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, o trabalho consiste em desenvolver cinco funções em C, conforme seguinte especificação:

A) void ImprimeMatriz(Matriz * A): Esta função imprime a matriz A, *inclusive* os elementos iguais a zero. Na formatação da saída, deve ser impresso uma linha da matriz por cada linha de saída.

B) void LeMatriz (Matriz * A): Esta função 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 anterior, 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

C) void ApagaMatriz(Matriz * A): Esta função devolve todas as células da matriz A para a área de memória disponível (usando a função *free()*).

D) void SomaMatriz (Matriz *A, Matriz *B, Matriz *C): Esta função recebe como parâmetros as matrizes A e B, devolvendo em C a soma de A com B.

E) void MultiplicaMatriz (Matriz *A, Matriz *B, Matriz *C): Esta função recebe como parâmetros as matrizes A e B, devolvendo em C o produto de A por B.

Para inserir e retirar células das listas que formam a matriz, crie funções especiais para este fim. Por exemplo, a função:

F) void insere(int i, int j;double v; Matriz * A): para inserir o valor v na linha i, coluna j da matriz A será útil tanto na função LeMatriz quanto na função SomaMatriz.

As matrizes a serem lidas para testar as funções podem ser as seguintes:

a) a mesma matriz apresentada anteriormente como exemplo;

b) $$\begin{Bmatrix} 50 & 30 & 0 & 0 \\ 10 & 0 & -20 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -5 \end{Bmatrix}$$

c) $$\begin{Bmatrix} 3 & 0 & 0 \\ 0 & -1 & 0 \end{Bmatrix}$$

O que deve ser apresentado (entregue):

1) Arquivo matriz.h, com a definição da estrutura do TAD matriz esparsa, bem como as assinaturas das funções sobre esse TAD;

2) Arquivo matriz.c, com a implementação das funções do TAD matriz esparsa;
3) Arquivo mainMatriz.c com o programa que testa as operações sobre o TAD.
4) Três arquivos com matrizes de teste que serão lidas

É obrigatório o uso de alocação dinâmica de memória para implementar as listas de adjacências que representam as matrizes. Você também deverá implementar todas as funções necessárias a manipulação da matriz esparsa, conforme necessidade.

Um exemplo de programa para testar as funções está listado abaixo:

```
int main (void) {
...
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);
MutiplicaMatriz(B, B, C);
ImprimeMatriz(B); ImprimeMatriz(B); ImprimeMatriz(C);
ApagaMatriz(A);ApagaMatriz(B); ApagaMatriz(C);
...}
```