Trabalho prático número 05
Matrizes esparsas

A resolução desta tarefa deverá ser postada no AVA até às
23h59min do dia **12 de novembro de 2023**.

Instrução

Uma matriz esparsa é uma estrutura de dados na qual a maioria dos elementos é igual a zero ou a algum outro valor predefinido e são comumente usadas na computação e programação para economizar memória e melhorar a eficiência das operações matriciais em situações em que a maioria dos valores de uma matriz é zero. No lugar de armazenar cada elemento da matriz, uma matriz esparsa armazena apenas os elementos diferentes de zero, juntamente com informações adicionais para indicar suas posições na matriz. Isso reduz significativamente a quantidade de memória necessária para representar a matriz e acelera as operações matriciais, especialmente quando a matriz é grande e a maioria de seus elementos é zero.

Neste trabalho prático você utilizará o tipo abstrato de dados lista (duplamente) encadeada para implementar o tipo abstrato de dados "matriz esparsa".

Nas matrizes esparsas, espaço em memória é economizado ao armazenar apenas os termos diferentes de zero e as operações usais sobre essas matrizes (somar, multiplicar, inverter, pivotar) também podem ser feitas em tempo muito menor se não armazenarmos as posições que contêm zeros.

Uma maneira eficiente de representar estruturas com tamanho variável e/ou desconhecido é com o emprego de alocação encadeada, utilizando listas. Vamos usar essa representação para armazenar as matrizes esparsas. Cada coluna da matriz será representada por uma lista linear circular com uma célula cabeça. Da mesma maneira, cada linha da matriz também será representada por uma lista linear circular com uma célula cabeça. Cada célula da estrutura, além das células cabeça, representará os termos diferentes de zero da matriz e deverá ser como no código abaixo:

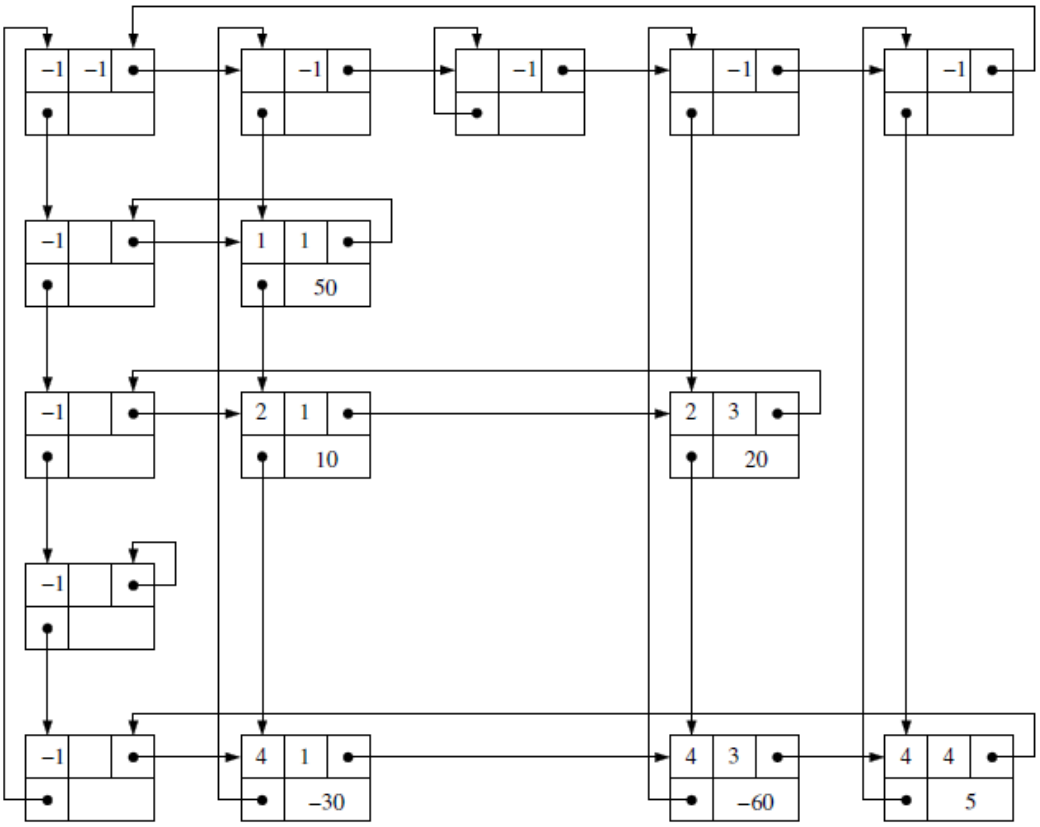
```
typedef struct Celula{
    struct Celula *direita, *abaixo;
    int linha, coluna;
    float valor;
} Celula;

typedef struct {
    Celula *cabeca;
    int nlin, ncol;
} Matriz;
```

O campo abaixo deve ser usado para referenciar o elemento diferente de zero na mesma coluna e o campo direita deve ser usado para referenciar o próximo elemento diferente de zero na mesma linha. Assim, 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. Essa célula deverá pertencer a 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 seguinte matriz esparsa:

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

A representação da matriz A pode ser vista a seguir.



Com essa representação, uma matriz esparsa m x 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 x n posições necessárias para representar a matriz toda, desde que r seja suficientemente pequeno.

Dada a representação de listas duplamente encadeadas, o trabalho consiste em desenvolver em C um tipo abstrato de dados Matriz com as seguintes operações, conforme esta especificação:

a) void imprimeMatriz(Matriz A)

Essa função imprime (uma linha da matriz por linha na saída) a matriz A, inclusive os elementos iguais a zero.

b) Matriz leMatriz()

Essa função lê, de algum arquivo texto de entrada, os elementos diferentes de zero de uma matriz e monta a estrutura especificada anteriormente. Considere que a entrada consiste dos valores de m e n (número de linhas e de colunas da matriz) seguidos de tuplas (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) Matriz somaMatrizes(Matriz A, Matriz B)

Esse método recebe como parâmetros as matrizes A e B, devolvendo uma matriz C que é a soma de A com B.

d) Matriz multiplicaMatrizes(Matriz A, Matriz B)

Esse método recebe como parâmetros as matrizes A e B, devolvendo uma matriz C que é o produto de A por B.

Observação:

- Para inserir e retirar células das listas que formam a matriz, crie métodos especiais para esse fim. Por exemplo, use **void insere(int i, int j, double v)** para inserir o valor v na linha i, coluna j da matriz A. Ademais, cabe destacar que essa função será útil tanto na função leMatriz quando na função somaMatriz.

- É obrigatório o uso de alocação dinâmica de memória para implementar as listas de adjacência que representam as matrizes.

As funções deverão ser testadas utilizando-se um programa **main** similiar ao apresentado abaixo:

```
int main(void) {
    Matriz A, B,C;
    A = leMatriz();
    imprimeMatriz(A);
    B = leMatriz();
    imprimeMatriz(B);
    C = somaMatriz(A,B);
    imprimeMatriz(C);

    // Outros testes devem ser considerados aqui...

}
```

O que deve ser feito

No desenvolvimento deste trabalho serão avaliados:

- Correção da solução proposta;
- Utilização das “structs” aqui propostas;
- Código-fonte disponibilizado e compartilhado no GitHub;
- Documentação por meio do template utilizado até o momento;
- Desenvolvimento do código-fonte em 3 arquivos (main.c, matriz.c e matriz.h);
- Inconsistências de dados de entradas;
- Condições para que as operações sejam feitas;
- Leitura das matrizes a partir de arquivos texto;