UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
http://www.icmc.usp.br/

Professor: Rodrigo Fernandes de Mello (mello at icmc.usp.br)

Trabalho 03: This is Esparsa!!!

1 Prazos, Especificações e Observações importantes:

- O trabalho descrito a seguir é individual e não será tolerado qualquer tipo de plágio ou cópia em partes ou totalidade do código. Caso seja detectada alguma irregularidade, os envolvidos serão chamados para conversar com o professor responsável pela disciplina e os trabalhos serão zerados.

- A interpretação desta descrição faz parte do trabalho. Leia a descrição do trabalho com atenção e várias vezes, anotando os pontos principais e as possíveis formas de resolver o problema. Comece a trabalhar o quanto antes para não ficar dúvidas e você consiga entregar o trabalho a tempo.

- O trabalho deverá ser submetido em formato zip/Makefile contendo o arquivo principal, o Makefile e possíveis TAD's implementadas (pares `.h .c`). Atente para o fato que todos os arquivos enviados deveram conter um cabeçalho contendo nome e número USP.

- A implementação é livre, crie quantas TAD's julgarem necessárias. Importante notar que a modularização do código bem como a forma como as TAD's foram criadas serão levados em consideração na atribuição final da nota.

- Compile o programa em um sistema operacional linux antes de submeter. **Trabalho em que o comando `make` e `make run` não funcione, não será corrigido e consequentemente receberá nota zero**. Caso não tenha o sistema operacional linux instalado em sua maquina, aconselha-se a utilização de uma máquina virtual. Como instalar o sistema operacional linux em uma maquina virtual está descrito no seguinte tutorial: `http://maistutoriais.com/2012/01/instalar-maquina-virtual-usando-o-`

- Referencie, com um comentário no próprio código, qualquer algoritmo ou trecho de código retirado da internet. Código copiado sem a devida referencia é considerado plágio que por sua vez é crime.

- Nenhuma saída do trabalho será em arquivo, ou seja, o resultado dos algoritmos deverá ser exibidos utilizando o stdout `printf`.

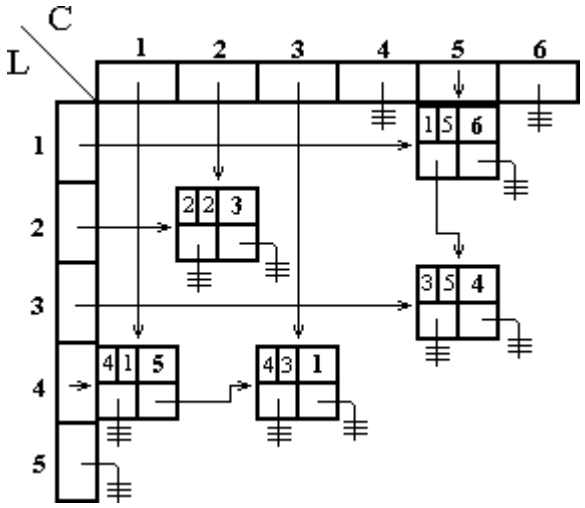
2 Descrição do Problema:

Uma matriz é dita esparsa quando possui uma grande quantidade de elementos que valem zero (ou não presentes, ou não necessários). Uma das implementações mais comuns utilizam um par de vetores estáticos que apontam para elementos diferentes de zero, ou seja, os elementos que possuem valor zero não são armazenados.

Por exemplo, seja a matriz abaixo, a qual contém 5 linhas e 6 colunas. Apenas 5 de seus 30 elementos são não nulos:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 6 & 0 \\ 0 & -3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 \\ 0 & 5 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

A representação visual dos elementos dentro de uma matriz esparsa é:



O seu trabalho é programar a TAD MatrizEsparsa. Assim como no exemplo anterior, **um nó só deverá estar na memória caso represente um valor diferente de zero, caso contrario a memória alocada por ele deverá ser liberada ou não alocada.**

Para correção no run.codes as entradas do sistema e as saídas devem obedecer padrões que serão descritos a seguir.

3 Entrada do Sistema:

Os dados serão inseridos no sistema seguindo a seguinte ordem:

1. '`Operação`' - A primeira informação será qual operação deverá ser executada com as matrizes. Ela poderá assumir o valor **'A'** caso a operação que deverá ser realizada for **Adição** ou **'M'** caso seja **Multiplicação**

2. '`-1 nrlinhas nrcolunas`' - Informação sobre o máximo de linhas e o número máximo de colunas que a primeira matriz terá. Observe que este comando sempre começa com o valor -1 que serve para informa ao programa que os valores em seguida são referentes ao tamanho da matriz.

3. '`linha coluna valor`' - Todos os valores presentes na primeira matriz. Eles serão inseridos em sequência e será fornecido o número da linha, o número da coluna e o valor que aquele elemento terá.

4. '`-1 nrlinhas nrcolunas`' - Informação sobre o máximo de linhas e o número máximo de colunas que a segunda matriz terá. Observe que este comando sempre começa com o valor -1 que, neste caso, serve para informa ao programa que os

valores em seguida são referentes ao tamanho da matriz 2 e não mais um elemento da matriz 1.

5. '`linha coluna valor`' - descreve os valores presentes na segunda matriz. Será fornecido o número da linha, o número da coluna e o valor que aquele elemento terá.

Assim, a entrada seguirá o seguinte padrão:

```
Operação
-1 nrlinhas nrcolunas
linha_0 coluna_0 valor
linha_0 coluna_1 valor
...
linha_0 coluna_j valor
linha_1 coluna_0 valor
....
linha_i coluna_j valor
-1 nrlinhas nrcolunas
linha_0 coluna_0 valor
linha_0 coluna_1 valor
...
linha_0 coluna_j valor
linha_1 coluna_0 valor
....
linha_i coluna_j valor
```

Um possível valor para entrada do programa é:

```
M
-1 10 10
9 7 4
3 4 6
1 3 5
-1 12 12
7 7 12
2 4 5
1 3 5
11 3 7
```

4 Saída do Sistema:

Após cada operação com as matrizes, deverá ser impresso os dados da matriz resultante. Para isso imprima primeiro o tamanho seguido dos valores de cada elemento não nulo da matriz.

1. '`-1 nrlinhas nrcolunas`' - Informação sobre o número de linhas e o número de colunas da matriz resultante. Observe que este comando sempre começa com o valor -1 que serve para informa ao programa que os valores em seguida são referentes ao tamanho da matriz. Utilize o seguinte comando para imprimir o tamanho da matriz:

```
printf("-1 %d %d\n",nrlinhas, nrcolunas);
```

2. '`linha coluna valor`' - Todos os valores presentes na matriz resultante. Para cada linha da matriz, imprima os valores não nulos nela contida. Caso uma linha não tenha elementos válidos, nada deverá ser impresso e o processo continua para a linha em sequência. Utilize o seguinte comando para imprimir os elementos da matriz:

```
printf("%d %d %d\n", linha, coluna, valor);
```

o algoritmo final para impressão dos elementos será:

```
void imprimir_matriz(MATRIZ_ESPARSA *matriz) {
    int i;
    CELULA *paux;

    printf("-1 %d %d\n", matriz->nr_linhas; matriz->nr_colunas);
    for (i = 0; i < matriz->nr_linhas; i++) {
        for (paux = matriz->linhas[i]; paux != NULL; paux = paux->direita) {
            printf("%d %d %d\n", paux->linha, paux->coluna, paux->valor);
        }
    }
}
```

Uma possível saída do programa é:

```
-1 10 10
0 5 260
1 0 -598
1 5 1508
2 1 -1170
2 5 460
2 6 -1344
2 9 1443
3 1 -1170
3 2 -30
```

5 Operações do sistema:

5.1 Adição

Em matemática a adição de matrizes é uma operação que produz a soma de duas matrizes. Duas operações distintas são definidas como a soma de matrizes: a soma termo-a-termo e a soma direta. Em particular, no nosso trabalho, iremos utilizar a soma termo-a-termo.

A adição usual de duas matrizes é definida quando elas possuem as mesmas dimensões: a soma de duas matrizes A e B de ordem mxn, denotada por $A+B$, é também uma matriz

m por n, cujos termos são a soma dos termos correspondentes das matrizes A e B. Se o termo situado na interseção da linha i com a coluna j da matriz M for denotado por M_{ij}, então a soma das matrizes A e B pode ser definida pela fórmula:

$$(A + B)_{ij} = A_{ij} + B_{ij}$$

para cada i de 1 a m e cada j de 1 a n.

Para exemplificar a operação de adição, considere as seguintes matrizes A e B respectivamente:

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}_{3\times4}$$

$$B = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 \end{pmatrix}_{3\times4}$$

o resultado da soma das matrizes:

$$A + B = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 2 & 0 \end{pmatrix}_{3\times4}$$

A entrada do sistema:

```
A
-1 3 4
1 1 3
-1 3 4
0 0 1
2 2 2
```

e a saída esperada:

```
-1 3 4
0 0 1
1 1 3
2 2 2
```

5.2 Multiplicação

Em matemática, o produto de duas matrizes é definido somente quando o número de colunas da primeira matriz é igual ao número de linhas da segunda matriz. Se A é uma matriz mxn e B é uma matriz nxp, então seu produto é uma matriz mxp definida como AB (ou por $A \cdot B$). O produto é dado por

$$(A + B)_{ij} = \sum_{r=1}^{n} a_{ir} b_{rj} = a_{i1} b_{1j} + a_{i2} b_{2j} + ... + a_{in} b_{nj}$$

para cada par i e j com $1 \leq i \leq m$ e $1 \leq j \leq p$.

Para exemplificar a operação de multiplicação, considere as seguintes matrizes A e B respectivamente:

$$A = \begin{pmatrix} 0 & 0 \\ 0 & 3 \end{pmatrix}_{2\times 2}$$

$$B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 2 \end{pmatrix}_{2\times 3}$$

o resultado da soma das matrizes:

$$A \cdot B = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 6 \end{pmatrix}_{2\times 3}$$

A entrada do sistema:

```
A
-1 2 2
1 1 3
-1 2 3
0 0 1
1 2 2
```

e a saída esperada:

```
-1 2 3
1 2 6
```

6 main.c

```
#include <stdio.h>
#include <stdlib.h>

#include "matriz_esparsa.h"

int main(){
char op;
int nr_linhas, nr_colunas, valor;

//definindo a operação
scanf("%c", &op);

//lendo a matriz a
scanf("%d %d %d", &valor, &nr_linhas, &nr_colunas);
MATRIZ_ESPARSA * A = criar_matriz(nr_linhas, nr_colunas);
while (scanf("%d %d %d", &nr_linhas, &nr_colunas, &valor) && nr_linhas != -1){
set_matriz(A, nr_linhas, nr_colunas, valor);
}

//lendo a matriz b
MATRIZ_ESPARSA * B = criar_matriz(nr_colunas, valor);
while (scanf("%d %d %d", &nr_linhas, &nr_colunas, &valor) != EOF){
set_matriz(B, nr_linhas, nr_colunas, valor);
}

MATRIZ_ESPARSA *C = NULL;
if (op == 'M'){
C = multiplicar_matriz(A,B);
} else if (op == 'A'){
C = somar_matriz(A,B);
}

imprimir_matriz(C);

apagar_matriz(&A);
apagar_matriz(&B);
apagar_matriz(&C);

return 0;
}
```

7 matriz_esparsa.h

```
#ifndef MATRIZ_ESPARSA_H
#define MATRIZ_ESPARSA_H

typedef struct matriz_esparsa MATRIZ_ESPARSA;

MATRIZ_ESPARSA *criar_matriz(int nr_linhas, int nr_colunas);
void apagar_matriz(MATRIZ_ESPARSA **matriz);

int set_matriz(MATRIZ_ESPARSA *matriz, int lin, int col, int val);

MATRIZ_ESPARSA *multiplicar_matriz(MATRIZ_ESPARSA *m1, MATRIZ_ESPARSA *m2);
MATRIZ_ESPARSA *somar_matriz(MATRIZ_ESPARSA *m1, MATRIZ_ESPARSA *m2);

void imprimir_matriz(MATRIZ_ESPARSA *matriz);

#endif  /* MATRIZ_ESPARSA_H */
```

8 matriz_esparsa.c

```
#include "matriz_esparsa.h"

#include <stdlib.h>
#include <stdio.h>

typedef struct CELULA {
    int linha;
    int coluna;
    int valor;
    struct CELULA *direita;
    struct CELULA *abaixo;
} CELULA;

struct matriz_esparsa {
    CELULA **linhas;
    CELULA **colunas;
    int nr_linhas;
    int nr_colunas;
};

MATRIZ_ESPARSA *criar_matriz(int nr_linhas, int nr_colunas) {
    MATRIZ_ESPARSA *mat = (MATRIZ_ESPARSA *) malloc(sizeof (MATRIZ_ESPARSA));

    if (mat != NULL) {
        int i;
        mat->nr_colunas = nr_colunas;
        mat->nr_linhas = nr_linhas;
        mat->colunas = (CELULA **) malloc(sizeof (CELULA *) * nr_colunas);
        mat->linhas = (CELULA **) malloc(sizeof (CELULA *) * nr_linhas);

        if (mat->colunas != NULL && mat->linhas != NULL) {
            for (i = 0; i < nr_colunas; i++) {
                mat->colunas[i] = (CELULA *) malloc(sizeof (CELULA));
                mat->colunas[i]->abaixo = NULL;
            }

            for (i = 0; i < nr_linhas; i++) {
                mat->linhas[i] = (CELULA *) malloc(sizeof (CELULA));
                mat->linhas[i]->direita = NULL;
            }
        }
    }

    return mat;
}
```

```
void apagar_matriz(MATRIZ_ESPARSA **matriz) {
    int i;
    for (i = 0; i < (*matriz)->nr_linhas; i++) {
        CELULA *paux = (*matriz)->linhas[i]->direita;
        while (paux != NULL) {
            CELULA *prem = paux;
            paux = paux->direita;
            free(prem);
        }

        free((*matriz)->linhas[i]);
    }

    free((*matriz)->linhas);
    free((*matriz)->colunas);
    free((*matriz));
    *matriz = NULL;
}

int set_matriz(MATRIZ_ESPARSA *matriz, int lin, int col, int val) {
    if (lin < matriz->nr_linhas && col < matriz->nr_colunas) {
        CELULA *paux = matriz->linhas[lin];
        while (paux->direita != NULL && paux->direita->coluna <= col) {
            paux = paux->direita;
        }

        if (paux->coluna == col) {
            paux->valor = val;
        } else {
            CELULA *pnovo = (CELULA *) malloc(sizeof (CELULA));
            if (pnovo != NULL) {
                pnovo->linha = lin;
                pnovo->coluna = col;
                pnovo->valor = val;
                pnovo->direita = paux->direita;
                paux->direita = pnovo;

                paux = matriz->colunas[col];
                while (paux->abaixo != NULL && paux->abaixo->linha <= lin) {
                    paux = paux->abaixo;
                }

                pnovo->abaixo = paux->abaixo;
                paux->abaixo = pnovo;
            }
        }
        return 1;
    }
    return 0;
}
```

```
int get_matriz(MATRIZ_ESPARSA *matriz, int lin, int col) {
    if (lin < matriz->nr_linhas && col < matriz->nr_colunas) {
        CELULA *paux = matriz->linhas[lin];
        while (paux->direita != NULL && paux->direita->coluna <= col) {
            paux = paux->direita;
        }

        if (paux->coluna == col) {
            return paux->valor;
        }
    }
    return 0;
}

MATRIZ_ESPARSA *multiplicar_matriz(MATRIZ_ESPARSA *m1, MATRIZ_ESPARSA *m2) {
    return NULL;
}

MATRIZ_ESPARSA *somar_matriz(MATRIZ_ESPARSA *m1, MATRIZ_ESPARSA *m2) {
    return NULL;
}

void imprimir_matriz(MATRIZ_ESPARSA *matriz) {
    int i;
    CELULA *paux;

    printf("-1 %d %d\n", matriz->nr_linhas; matriz->nr_colunas);
    for (i = 0; i < matriz->nr_linhas; i++) {
        for (paux = matriz->linhas[i]; paux != NULL; paux = paux->direita) {
            printf("%d %d %d\n", paux->linha, paux->coluna, paux->valor);
        }
    }
}
```