F A C E — F U M E C
CIÊNCIA DA COMPUTAÇÃO
Estruturas de Dados II

Primeiro Trabalho Prático — 13 de Agosto de 2020

Aluno: Vinícius Dias Rodrigues

Turma: 4NA

Professor: Flávio Velloso Laper

Este trabalho contém 7 atividades, para um total de 5 pontos. Leia atentamente todas elas antes de resolvê-las.

Em diversas aplicações, principalmente científicas, são usadas estruturas denominadas *matrizes esparsas*. Uma matriz esparsa é uma matriz na qual a maioria das posições contém zero. Para estas matrizes, é possível economizar uma quantidade significativa de espaço se apenas os elementos diferentes de zero forem armazenados.

Este exercício consiste na implementação de matrizes esparsas utilizando listas ligadas. Cada linha e cada coluna da matriz será representada por uma lista contendo apenas os elementos que são diferentes de zero. Cada célula, portanto, será representada pela estrutura a seguir:

```
1  typedef struct CELULA_TAG *PONT;
2
3  typedef struct {
4      int linha, coluna;
5      double valor;
6  } ITEM;
7
8  typedef struct CELULA_TAG {
9      ITEM item;
10     PONT direita;
11     PONT abaixo;
12 } CELULA;
13
14 typedef struct {
15     PONT primeiro, ultimo;
16 } LISTA;
```

Ou seja: cada célula (linhas 11–15) contém, além do *item* armazenando os valores (linha 12), um ponteiro *direita* (linha 13) para o próximo elemento diferente de zero na mesma linha e um ponteiro *abaixo* (linha 14) para o próximo elemento diferente de zero na mesma coluna. Cada item (linhas 6–9), por sua vez, contém informações sobre a linha e coluna às quais pertence (linha 7) e sobre o valor armazenado (linha 8).

Assim, dada uma matriz A, deve haver, para cada elemento $A(i, j)$ diferente de zero, uma célula contendo um *item* com o campo *valor* contendo $A(i, j)$, o campo *linha* contendo i e o campo *coluna* contendo j. Esta célula fará parte da lista ligada da linha i e da lista ligada da coluna j *ao mesmo tempo*, utilizando para isso os ponteiros *direita* (para a lista da linha) e *abaixo* (para a lista das colunas).

A matriz propriamente dita será representada pela estrutura a seguir:

```
1  typedef struct {
2    int nLinhas, nColunas;
```

```
3   LISTA *linha;
4   LISTA *coluna;
5 } MATRIZ;
```

Os campos *nLinhas* e *nColunas* (linha 7) guardam as dimensões da matriz. O campo *linha* (linha 8) é um vetor de listas ligadas de dimensão *nLinhas*, que deve ser alocado dinamicamente. O mesmo vale para o campo *coluna* (linha 9), cuja dimensão é dada por *nColunas*. Retomando o exemplo acima, o elemento $A(i, j)$ fará parte das listas *linha[i]* e *coluna[j]*.

Baseando-se na representação acima, o trabalho consiste em desenvolver os seguinte procedimentos:

1. **void** criaMatriz(MATRIZ *a);

   Esta função apenas inicializa os atributos da matriz a com valores *default* (0, NULL).

2. **void** inicializaMatriz(MATRIZ *a, **int** linhas, **int** colunas);

   Esta função inicializa a matriz a, alocando os vetores de listas a partir da quantidade de linhas e colunas indicadas nos parâmetros. Lembre-se de inicializar cada uma das listas como uma lista vazia.

3. **void** leMatriz(MATRIZ *a);

   Esta função lê os dados da matriz a partir da entrada padrão. Estes dados devem estar no seguinte formato:

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
-1	-1	−1.0

Não são informados valores úteis para todas as células da nossa matriz, sendo necessário que interpretemos os campos afim de preencher de forma correta.

   O dados acima representam a seguinte matriz:

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

   Ou seja: a primeira linha informada traz o número de linhas e colunas da matriz. As linhas subsequentes trazem os dados de linha/coluna/valor dos elementos diferentes de zero. Um número de linha (ou coluna) negativo indica o fim das informações. *Atenção*: Não leia dados para uma matriz que não esteja vazia.

4. **void** imprimeMatriz(MATRIZ *a);

   Esta função deve imprimir a matriz recebida em um formato conveniente, tal como se a mesma não fosse uma matriz esparsa. Ou seja, a matriz deve ser exibida como uma tabela com suas linhas e colunas, mostrando inclusive os elementos iguais a zero.

5. **void** somaMatriz(MATRIZ *ma, MATRIZ *mb, MATRIZ *mc);

   Esta função recebe duas matrizes ma e mb e devolve em mc a soma das duas. Verifique se as dimensões das matrizes permitem a adição. A matriz mc deve ter sido criada, porém não inicializada.

6. **void** multiplicaMatriz(MATRIZ *ma, MATRIZ *mb, MATRIZ *mc);

   Esta função recebe duas matrizes ma e mb e devolve em mc o produto das duas. Verifique se as dimensões das matrizes permitem a multiplicação. A matriz mc deve ter sido criada, porém não inicializada.

7. **void** apagaMatriz(MATRIZ *ma);

   Devolve para o sistema *todas* as áreas de memória alocadas dinamicamente para a matriz, e volta seus atributos para os valores default. Ou seja: o estado da matriz, após o apagamento, deve ser o de uma matriz que foi criada, porém não inicializada.

Para sua conveniência, os arquivos com as estruturas a utilizar são reproduzidos a seguir:

Estrutura das listas *(lista.h)*:

```
typedef struct CELULA_TAG *PONT;

typedef struct {
    int linha, coluna;
    double valor;
} ITEM;

typedef struct CELULA_TAG {
    ITEM item;
    PONT direita;
    PONT abaixo;
} CELULA;

typedef struct {
    PONT primeiro, ultimo;
} LISTA;
```

Estrutura da matriz *(matriz.h)*:

```
#include "lista.h"

typedef struct {
  int nLinhas, nColunas;
  LISTA *linha;
  LISTA *coluna;
} MATRIZ;
```

Um exemplo de programa principal é também listado a seguir:

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

#include "matriz.h"

int main(int argc, char *argv[])
{
  MATRIZ a, b, c;

   printf("Primeiro grupo de matrizes\n");
   criaMatriz(&a); leMatriz(&a); imprimeMatriz(&a);
   criaMatriz(&b); leMatriz(&b); imprimeMatriz(&b);
```

```
13    printf("Soma\n");
14    criaMatriz(&c); somaMatriz(&a, &b, &c); imprimeMatriz(&c);
15    printf("Produto\n");
16    apagaMatriz(&c); multiplicaMatriz(&a, &b, &c); imprimeMatriz(&c);
17
18    printf("Segundo grupo de matrizes\n");
19    apagaMatriz(&a); apagaMatriz(&b);apagaMatriz(&c);
20    leMatriz(&a); imprimeMatriz(&a);
21    leMatriz(&b); imprimeMatriz(&b);
22    printf("Soma\n");
23    criaMatriz(&c); somaMatriz(&a, &b, &c); imprimeMatriz(&c);
24    printf("Produto\n");
25    apagaMatriz(&c); multiplicaMatriz(&a, &b, &c); imprimeMatriz(&c);
26
27    apagaMatriz(&a);
28    apagaMatriz(&b);
29    apagaMatriz(&c);
30
31    return 0;
32 }
```

As matrizes a seguir pode ser utilizadas com ele. Para o primeiro grupo:

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

Para o segundo grupo:

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

Observações importantes:

1. *Formato de saída:* a saída dos programas deve consistir apenas das informações mencionadas no texto acima. Não imprima títulos, versões, ou qualquer outro tipo de informação. Não procure formatar a tela: apenas envie os dados para a saída padrão.

2. *Entrega:* a entrega do trabalho consistirá dos arquivos fonte *comentados* e do código executável do programa. Os comentários devem descrever, em linhas gerais, o algoritmo empregado para a solução do problema. Entregue este material através do sistema acadêmico. O programa poderá ser testado contra diferentes entradas para comprovar sua correção.

3. Faça as consistências que julgar necessárias sobre as informações fornecidas pelo usuário. Lembre-se que seu programa será testado e deverá se comportar de forma consistente para diferentes entradas. Não confie apenas na execução correta da função *main* fornecida.

4. Identifique-se: coloque o seu nome e turma em destaque em um comentário no início dos fontes do programa.

5. A pontuação do trabalho será feita considerando os seguintes critérios:

   - Correção.
   - Formato de entrada.

- Formato de saída.
- Material a entregar.
- Prazo de entrega.

Trabalhos que não obedecerem aos critérios estipulados serão desvalorizados (ou até mesmo desconsiderados).

6. Incluído no material fornecido está um programa *matriz.exe*. Utilize-o como exemplo, principalmente para os formatos de entrada e saída.
7. O trabalho é individual. Você pode discutí-lo com seus colegas, mas deve fazer a sua própria implementação. Cópias não serão toleradas.

BOA SORTE.

1) Mudar a entrada padrão não é uma tarefa difícil, da pra se organizar da seguinte maneira pelo terminal:

./myProgram < entrada.txt

Ainda tenho algumas dificuldades pra entender a estrutura do nosso projeto;

2) O "Matriz.exe" já tem a resposta do programa. Descobri como fazer para redirecionar a entrada padrão mas não consegui realizar isso para o arquivo teste.c;

Duvidas Vinicius::

Ao compilar gera um executável?
Como posso fazer isso?
Qual ide devo usar?