Universidade Federal do Ceará - Campus Quixadá
QXD0010 – Estruturas de Dados – Turma 05A
Prof. Atílio Gomes Luiz

Primeiro Projeto

As soluções das questões descritas neste documento devem ser entregues até a meia-noite do dia **28/02/2021** pelo Moodle.

Leia atentamente as instruções abaixo.

Instruções:

- Este trabalho pode ser feito em **dupla** ou **individualmente** e deve ser implementado usando a linguagem de programação C++ **(Não aceitarei mais do que dois alunos por projeto)**
- Você tem até o dia 9 de fevereiro para definir se vai fazer o trabalho em dupla ou individualmente. Me envie um email (`gomes.atilio@ufc.br`) com o nome completo e matrícula dos integrantes da sua dupla assim que ela for definida (basta um integrante me comunicar).
- Coloque a solução de cada questão em uma pasta específica. O seu trabalho deve ser compactado (.zip, .rar, etc.) e enviado para o Moodle na atividade correspondente ao Projeto 01.
- Identifique o seu código-fonte colocando o **nome** e **matrícula** dos integrantes da dupla como comentário no início de seu código.
- Indente corretamente o seu código para facilitar o entendimento. **Trabalhos com códigos maus indentados sofrerão redução na nota.**
- As estruturas de dados devem ser implementadas como TAD. Deve haver um arquivo de cabeçalho (.h) e um arquivo de implementação (.cpp) para cada estrutura de dado programada.
- Os programas-fonte devem estar devidamente organizados e documentados.
- Observação: Lembre-se de desalocar os endereços de memória alocados quando os mesmos não forem mais ser usados.
- **Observação: Qualquer indício de plágio resultará em nota ZERO para todos os envolvidos.**

DICA: COMECE O TRABALHO O QUANTO ANTES.

1 Matrizes esparsas

Matrizes esparsas são matrizes nas quais a maioria das posições é preenchida por zeros. Para essas matrizes, podemos economizar um espaço significativo de memória se apenas os termos diferentes de zero forem armazenados. As operações usuais sobre essas matrizes (somar, multiplicar, inverter, pivotar) também podem ser feitas em tempo muito menor se não armazenarmos as posições que contêm zeros.

Há numerosos exemplos de aplicações que exigem o processamento de matrizes esparsas. Muitas se aplicam a problemas científicos ou de engenharia que só são facilmente entendidos por peritos. No entanto, há uma aplicação muito familiar que usa matriz esparsa: um programa de planilha. Muito embora a matriz de uma planilha comum seja muito grande, digamos 999 por 999, apenas uma porção da matriz pode realmente estar sendo usada em um dado momento. Planilhas usam a matriz para guardar fórmulas, valores e strings associados a cada posição. Em uma matriz esparsa, o armazenamento para cada elemento é alocado de um espaço de memória livre (heap) conforme se torne necessário. Embora apenas uma pequena porção dos elementos esteja realmente sendo usada, a matriz pode parecer muito grande – maior do que o que normalmente caberia na memória do computador.

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 um **nó cabeça**. Da mesma maneira, cada linha da matriz também será representada por uma lista linear circular com um nó cabeça. Cada nó da estrutura, além dos nós cabeça, representará os termos diferentes de zero da matriz e deverá ser como no código abaixo:

```
1 struct Node {
2     Node *direita;
3     Node *abaixo;
4     int linha;
5     int coluna;
6     double valor;
7 };
```

O campo `abaixo` do `struct Node` deve ser usado para referenciar o próximo elemento diferente de zero na mesma coluna. O campo `direita` deve ser usado para referenciar o próximo elemento diferente de zero na mesma linha. Dada uma matriz A, para um valor $A(i, j)$ diferente de zero, deverá haver um nó com o campo `valor` contendo $A(i, j)$, o campo `linha` contendo i e o campo `coluna` contendo j. Esse nó deverá pertencer à lista circular da linha i e também deverá pertencer à lista circular da coluna j. Ou seja, cada nó pertencerá a duas listas ao mesmo tempo. Para diferenciar os nós cabeça, coloque -1 nos campos `linha` e `coluna` desses nós.

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 na Figura 1.

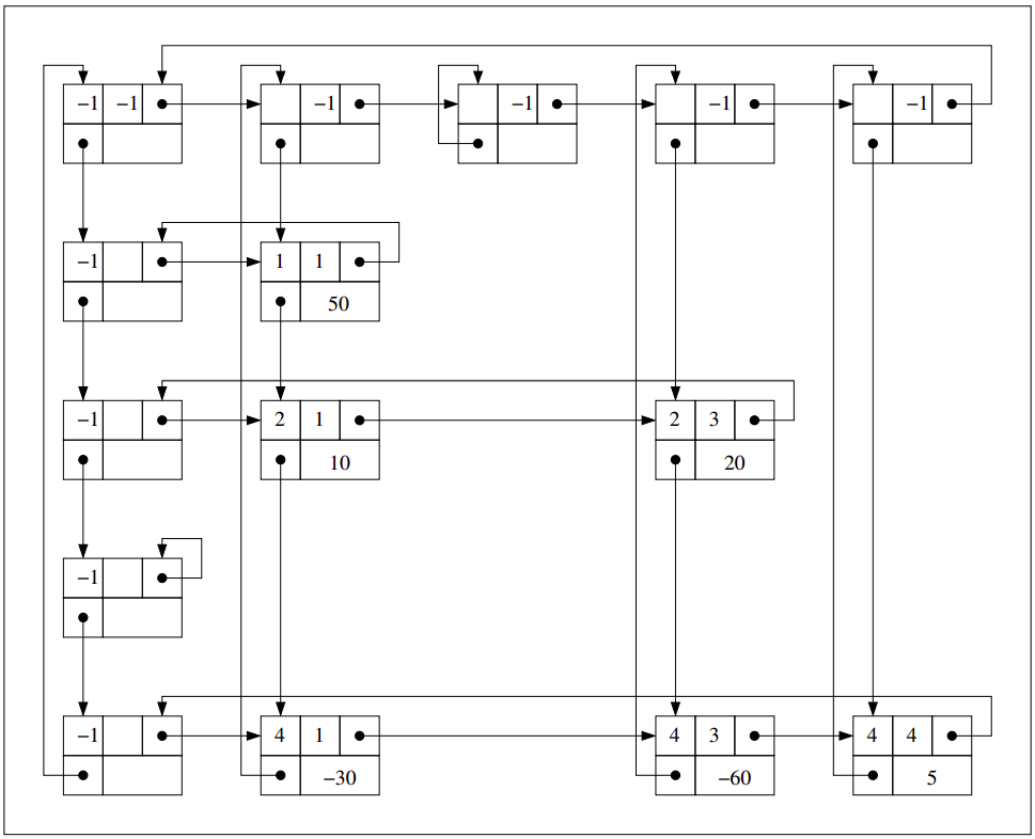

Figura 1: Exemplo de matriz esparsa.

Com essa representação, uma matriz esparsa $m \times n$ com r elementos diferentes de zero gastará $(m + n + r)$ nós. É bem verdade que cada nó 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.

O trabalho consiste em desenvolver em C++ um tipo abstrato de dados **Matriz** com as seguintes operações, conforme esta especificação:

- `void lerMatriz(std::string nome_do_arquivo);`
  Esse método lê, de um arquivo 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 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

- `void imprime();`
  Esse método imprime (uma linha da matriz por linha na saída) a matriz A, inclusive os elementos iguais a zero.

Para inserir e retirar nós das listas que formam a matriz, crie métodos especiais para esse fim. Por exemplo, você pode criar o método

- `void insere(int i, int j, double v)`
  Esse método insere o valor v na linha i, coluna j da matriz A.

Além do TAD, você deve implementar as seguintes funções:

- `Matriz *soma(Matriz *A, Matriz *B)`
  Essa função recebe como parâmetro as matrizes A e B, devolvendo uma matriz C que é a soma de A com B.

- `Matriz *multiplica(Matriz *A, Matriz *B)`
  Esse método recebe como parâmetro as matrizes A e B, devolvendo uma matriz C que é o produto de A por B.

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

Além da matriz A outras matrizes podem ser lidas para testar os métodos, como, por exemplo:

$$B = \begin{pmatrix} 50 & 30 & 0 & 0 \\ 10 & 0 & -20 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -5 \end{pmatrix} \qquad C = \begin{pmatrix} 3 & 0 & 0 \\ 0 & -1 & 0 \end{pmatrix}$$

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

```
1  void main() {
2      ...
3      A->lerMatriz("A.txt"); A->imprime();
4      B->lerMatriz("B.txt"); B->imprime();
5      C = soma(A,B); C->imprime();
6      B->lerMatriz("B2.txt");
7      A->imprime(); B->imprime();
8      C = soma(A,B); C->imprime();
9      C = multiplica(A,B); C->imprimeMatriz();
10     C = multiplica(B,B);
11     A->imprime(); B->imprime(); C->imprime();
12     ...
13 }
```

1.1 O que deve ser submetido

- **Deverá ser submetido:**
  - Um **relatório do trabalho** realizado, contendo a especificação completa das estruturas de dados utilizadas; e as decisões tomadas relativas aos casos e detalhes de especificação que porventura estejam omissos no enunciado.
  - Uma seção descrevendo como o trabalho foi dividido entre a dupla, se for o caso; além das dificuldades encontradas.
  - Os programas fonte devidamente organizados e documentados.

- Um dos parâmetros utilizados na avaliação da qualidade de uma implementação consiste na constatação da presença ou ausência de comentários. Comente o seu código. Mas também não comente por comentar, forneça bons comentários.

- Outro parâmetro de avaliação de código é a *portabilidade*. Dentre as diversas preocupações da portabilidade, existe a tentativa de codificar programas que sejam compiláveis em qualquer sistema operacional. Como testarei o seu código em uma máquina que roda Linux, não use bibliotecas que só existem para o sistema Windows como, por exemplo, a biblioteca `conio.h` e outras tantas.

2 Listas Circulares Duplamente Encadeadas

A estrutura de lista simplesmente encadeada, vista durante a aula, caracteriza-se por formar um encadeamento simples entre os nós: cada nó armazena um ponteiro para o próximo elemento da lista. Dessa forma, não temos como percorrer eficientemente os elementos em ordem inversa. O encadeamento simples também dificulta a retirada de um elemento da lista. Mesmo se tivermos o ponteiro do elemento que desejamos retirar, temos de percorrer a lista, elemento por elemento, para encontrar o elemento anterior, pois, dado o ponteiro para um determinado elemento, não temos como acessar diretamente seu elemento anterior.

Para solucionar esses problemas, podemos formar o que chamamos de **listas duplamente encadeadas**. Nelas, cada elemento tem um ponteiro para o próximo elemento e um ponteiro para o elemento anterior. Assim, dado um elemento, podemos acessar os dois elementos adjacentes: o próximo e o anterior. A lista duplamente encadeada pode ou não ter um nó cabeça e pode ou não ser circular, conforme as conveniências do programador. Uma **lista circular duplamente encadeada com Nó Cabeça** é uma lista duplamente encadeada na qual o último elemento da lista passa a ter como próximo o primeiro elemento, que, por sua vez, passa a ter o último como anterior. A Figura 2 ilustra uma lista duplamente encadeada com estrutura circular e a presença de um nó cabeça.

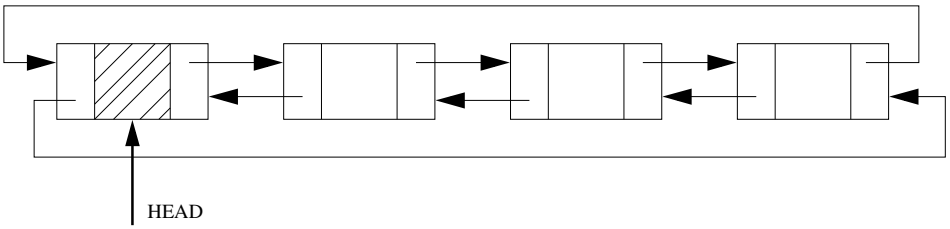

Figura 2: Lista circular duplamente encadeada com nó cabeça.

Problema: Implemente em C++ o Tipo Abstrato de Dados Lista Linear usando como base a estrutura de dados Lista Circular Duplamente Encadeada. A sua estrutura de dados deve ser encapsulada por uma classe chamada `List`, que deve suportar as seguintes operações:

- `List()`: Construtor da classe. Deve iniciar todos os atributos da classe com valores válidos.
- `~List()`: Destrutor da classe. Libera memória previamente alocada.
- `void push_back(int key)`: Insere um inteiro *key* ao final da lista.
- `int pop_back()`: Remove elemento do final da lista e retorna seu valor.
- `void insertAfter(int key, int k)`: Insere um novo nó com valor *key* após o *k*-ésimo nó da lista.
- `void remove(int key)`: Remove da lista a primeira ocorrência do inteiro *key*
- `void removeAll(int key)`: Remove da lista todas as ocorrências do inteiro *key*.

- `void print()`: Imprime os elementos da lista.
- `void printReverse()`: Imprime os elementos da lista em ordem reversa.
- `bool empty()`: Retorna `true` se a lista estiver vazia e `false` caso contrário.
- `int size()`: Retorna o número de nós da lista.
- `void clear()`: Remove todos os elementos da lista e deixa apenas o nó cabeça.
- `void concat(List *lst)`: Concatena a lista atual com a lista `lst` passada por parâmetro. Após essa operação ser executada, `lst` será uma lista vazia, ou seja, o único nó de `lst` será o nó cabeça.
- `List *copy()`: Retorna um ponteiro para uma cópia desta lista.
- `void copyArray(int n, int arr[])`: Copia os elementos do array *arr* para a lista. O array *arr* tem n elementos. Todos os elementos anteriores da lista são mantidos e os elementos do array *arr* devem ser adicionados após os elementos originais.
- `bool equal(List *lst)`: Determina se a lista passada por parâmetro é igual à lista em questão. Duas listas são iguais se elas possuem o mesmo tamanho e o valor do k-ésimo elemento da primeira lista é igual ao k-ésimo elemento da segunda lista.

2.1 Interface com o usuário

Escreva um programa principal (`main.cpp`) com **um menu de comandos** para que o usuário possa digitar os comandos e testar TODAS as operações da estrutura `List` que você implementou. Os comandos podem ser similares aos comandos utilizados nas atividades do Moodle.

2.2 O que deve ser submetido

- **Deverá ser submetido:**
  - Um **relatório do trabalho** realizado, contendo a especificação completa da estrutura de dados; Além disso, deve apresentar uma análise da complexidade das funções implementadas.
  - Uma seção descrevendo como o trabalho foi dividido entre a dupla, se for o caso; além das dificuldades encontradas.
  - Os programas fonte devidamente organizados e documentados.
- Um dos parâmetros utilizados na avaliação da qualidade de uma implementação consiste na constatação da presença ou ausência de comentários. Comente o seu código. Mas também não comente por comentar, forneça bons comentários.
- Não serão aceitos trabalhos submetidos após o prazo final.