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

AP2 - Projeto

A solução do problema descrito neste documento deve ser entregue até as 23h59min do dia **11/06/2023** pelo Moodle.

Leia atentamente as instruções abaixo.

Instruções:

- Este trabalho deve ser feito em **DUPLA** e deve ser implementado usando a linguagem de programação C++ **(Não aceitarei mais do que dois alunos por projeto)**
- Você tem até o dia **31 de maio de 2023** 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 do projeto 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 usando classes, como fizemos para as demais estruturas de dados programadas em aula.
- 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

Projeto retirado do livro de Nivio Ziviani [1].

1.1 Objetivos

Concretizar os conceitos de listas simplesmente encadeadas através de uma aplicação: matrizes esparsas.

1.2 Descrição

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, como Excel. 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 simplesmente encadeada circular** com um **nó sentinela**. Da mesma maneira, cada linha da matriz também será representada por uma lista simplesmente encadeada circular com um nó sentinela. Cada nó da estrutura, além dos nós-sentinela, 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. As células da matriz são indexadas a partir do número 1. Logo, $1 \leq i \leq linha$ e $1 \leq j \leq coluna$.

Como exemplo, 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.

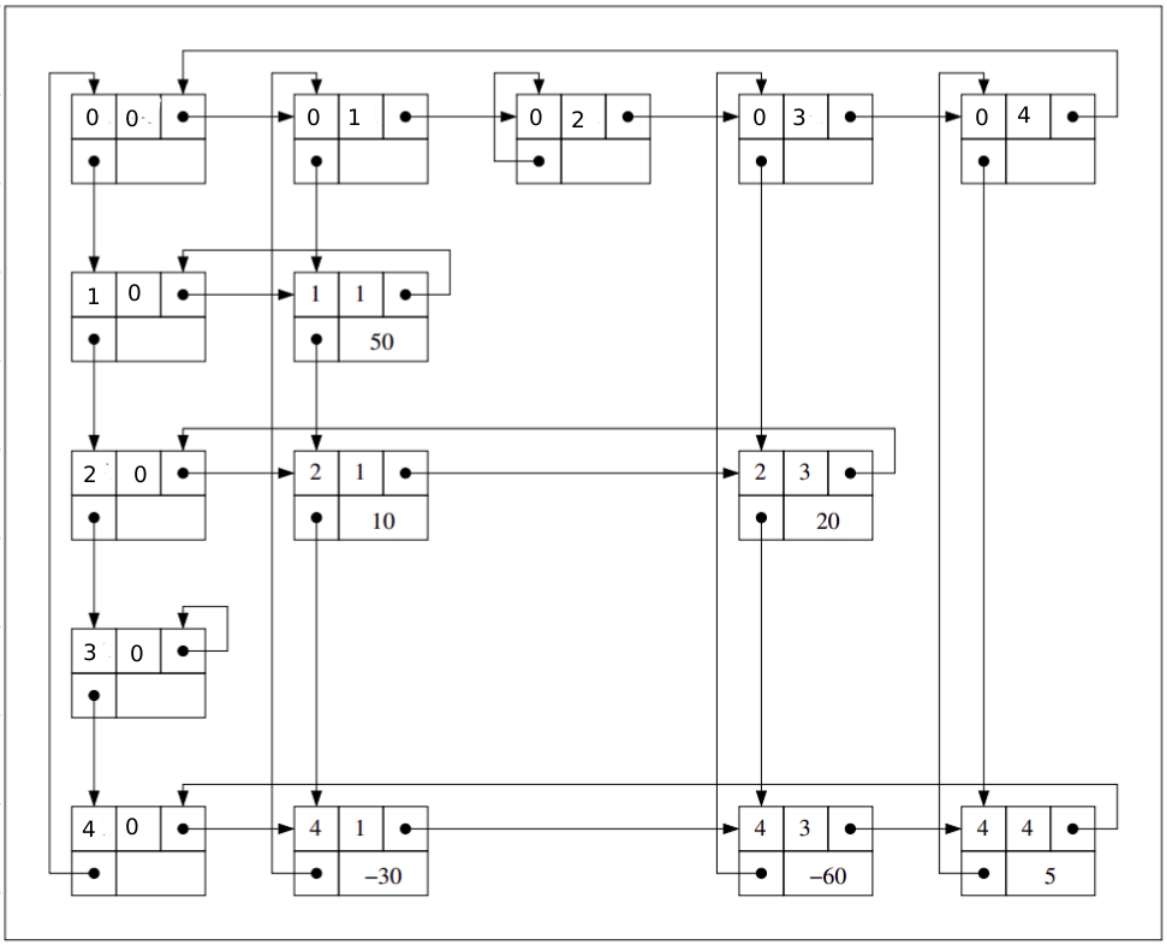

Figura 1: Exemplo de matriz esparsa.

Com essa representação, uma matriz esparsa $m \times n$ com r elementos diferentes de zero gastará $(m+n+r+1)$ 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.

1.3 TAD `SparseMatrix`

O trabalho consiste em desenvolver em C++ uma classe chamada **SparseMatrix** com (pelo menos) as seguintes operações, conforme esta especificação:

- `SparseMatrix(int m, int n);`
  Construtor da classe. Inicializa uma matriz esparsa vazia com capacidade para m linhas e n colunas. Essa função deve checar se os valores passados são válidos (se são inteiros positivos, ou seja $m > 0$ e $n > 0$); se não forem, uma exceção deve ser lançada. O esquema de uma matriz esparsa vazia é ilustrado na Figura 2.

  Atenção: Não deve ser alocado arrays nesse trabalho, apenas nós. Alocação de arrays de qualquer coisa que seja será penalizado.

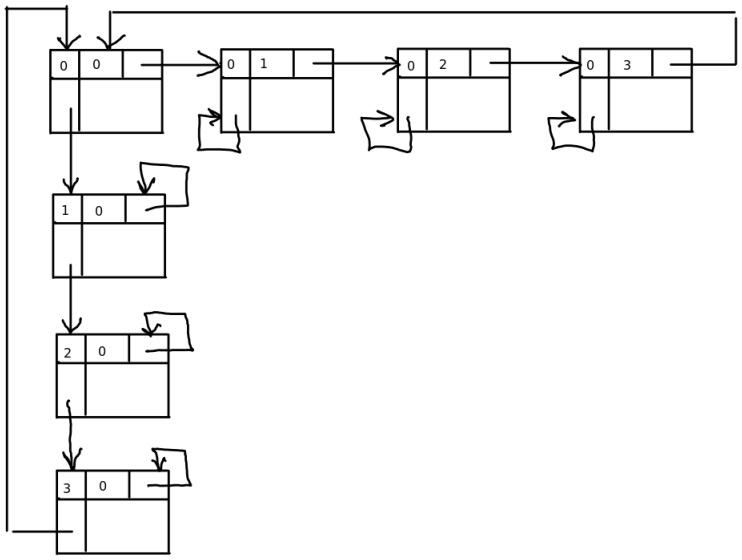

Figura 2: Esquema de uma matriz esparsa vazia.

- `~SparseMatrix();`
  Destrutor. Libera toda a memória que foi alocada dinamicamente para a matriz.

- `void insert(int i, int j, double value);`
  Esta função-membro faz o valor na célula (i, j) da matriz ser igual a `value`, onde i é a linha e j é a coluna. Se já houver um valor nessa posição, ele é atualizado para esse novo valor. Essa função deve checar se os índices i, j passados por parâmetro são válidos; se não forem válidos, uma exceção deve ser lançada. **Atenção:** No caso em que o valor do argumento `value` for igual a zero, a função deve desconsiderar e não fazer nada neste caso. Ou seja, chamadas dessa função passando o valor 0 não terão efeito algum.

- `double get(int i, int j);`
  Devolve o valor na célula (i, j) da matriz, onde i é a linha e j é a coluna. Essa função deve checar se os índices passados são válidos; se não forem válidos, uma exceção deve ser lançada.

- `void print();`
  Esta função imprime a matriz A no terminal, inclusive os elementos iguais a zero.

As funções-membro descritas acima são necessárias para possibilitar o mínimo de funcionamento da matriz. Se você identificar que precisa de funções-membro adicionais ou que precisa implementar funções auxiliares que facilitem a programação da estrutura de dados **SparseMatrix**, você tem liberdade para programá-las.

1.4 Arquivo `main.cpp`

Além do TAD SparseMatrix, você deve implementar três funções adicionais no arquivo **main.cpp** (não são funções-membro da SparseMatrix, são funções externas à classe):

- `SparseMatrix* readSparseMatrix(std::string nome_do_arquivo);`
  Essa função lê, de um arquivo de entrada, os elementos diferentes de zero de uma matriz esparsa e monta a estrutura especificada anteriormente, devolvendo um ponteiro para a SparseMatrix alocada dinamicamente. Considere que a primeira linha do arquivo de entrada consiste dos valores m e n (número de linhas e de colunas da matriz), e as demais linhas do arquivo são constituídas de triplas $(i, j, valor)$ para os elementos diferentes de zero da matriz. Por exemplo, para a matriz A anterior, o conteúdo do arquivo de 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
  ```

  Dica: O C++ possui a biblioteca `fstream` que possibilita abrir arquivos, ler e escrever neles de forma intuitiva, muito parecida com a leitura e escrita usando `cin` e `cout`. Pesquise sobre essa biblioteca em livros ou na internet. Existem muitos livros de C++ que possuem um capítulo exclusivo ensinando como ler e escrever em arquivos-texto. Ache o melhor livro que você puder e estude o tal capítulo a fim de aprender como manipular arquivos usando o C++.

  ATENÇÃO: As matrizes podem ter até 30000 linhas e 30000 colunas.

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

- `SparseMatrix *multiply(SparseMatrix *A, SparseMatrix *B);`
  Essa função recebe como parâmetro as matrizes A e B, devolvendo uma matriz C que é a multiplicação 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:

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

Observação: Desenvolva uma main interativa, com menu ou com comandos (fique à vontade para escolher o melhor formato), porém, não esqueça de documentar os comandos no relatório final, para que eu tenha um lugar para saber quais comandos usar a fim de executar o seu programa com sucesso. Uma possibilidade adicional é colocar um helper no seu programa a fim de disponibilizar no próprio programa uma listagem dos comandos que o usuário pode utilizar.

1.5 O que deve ser submetido

- **Deverá ser submetido:**

  (1) Um **relatório do trabalho** realizado, contendo:
  - (i) a descrição completa da estrutura de dados que foi implementada, suas aplicações, e tudo o mais que sirva de motivação;
  - (ii) as decisões tomadas relativas aos casos e detalhes de especificação que porventura estejam omissos no enunciado; por exemplo, se você precisar escrever funções auxiliares.
  - (iii) Uma seção descrevendo como o trabalho foi dividido entre a dupla, se for o caso;
  - (iv) As dificuldades encontradas;
  - (v) Listagem dos testes executados;
  - (vi) O relatório deve conter também uma análise da complexidade de pior caso das funções: `insert`, `get` e `sum`.
  - (vii) Referências bibliográficas, tutoriais, vídeos ou outros materiais consultados.

  (2) O **código-fonte** devidamente organizado e documentado. Além disso, os arquivos das matrizes que foram utilizadas como teste. A equipe pode desenvolver diversos testes que ficam a critério da equipe.

  (3) **Relatório escrito em Latex:** 0,2 ponto extra.
  O Latex é um programa de edição de texto que tem uma linguagem de marcação própria. No Moodle, eu coloquei links para minicursos de introdução ao Latex usando o Overleaf. Acho que vale a pena ver.

1.6 Comentários Gerais

- Comece a fazer este trabalho logo, enquanto o problema está fresco na memória e o prazo para terminá-lo está tão longe quanto jamais poderá estar.
- Clareza, identação e comentários no programa também vão valer pontos.

- O trabalho é individual ou em dupla.
- Trabalhos copiados receberão nota **ZERO**, para ambas as partes, a que cedeu e a que copiou.
- 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.

Obs.: Não serão aceitos trabalhos submetidos após o prazo final.

Referências

[1] N. Ziviani, F. C. Botelho. *Projeto de Algoritmos com implementações em Java e C++*, Editora Cengage Learning, 2006.