Matriz Esparsa

Elysson A. Lacerda[1], **Gustavo A. Monteiro**[2]

[1]Universidade Federal do Ceará (UFC)
Quixadá - Ceará - Brazil

Abstract. *This document consists of describing the implementation of a Sparse Matrix formed by simply linked circular lists. Work developed in C++ and uses Object Orientation concepts.*

Resumo. *Este documento consiste em descrever a implementação de uma Matriz Esparsa formada por listas circulares simplesmente encadeadas. Trabalho desenvolvido em* ***C++*** *e utiliza conceitos de Orientação a Objeto.*

1. Apresentação

As matrizes esparsas são matrizes que contêm uma grande quantidade de elementos nulos, o que as diferencia das matrizes densas. Devido a esse comportamento particular, a forma como as matrizes esparsas são armazenadas em memória difere da representação das matrizes densas. Existem diversos exemplos que ilustram a utilização dessa estrutura, uma aplicação comum é o uso de mapas de bits em imagens, onde a representação da imagem é feita por meio de uma matriz esparsa, aproveitando o fato de que a maioria dos pixels em uma imagem são nulos (sem informação). Isso permite economizar espaço de armazenamento e acelerar o processamento de imagens, outra aplicação é o armazenamento de dados em planilhas. Muitas vezes, as células em uma planilha contêm valores nulos, especialmente quando há uma grande quantidade de linhas e colunas. Utilizar uma representação de matriz esparsa nesse contexto ajuda a reduzir o consumo de memória e otimizar as operações realizadas nas planilhas.

Esses são apenas alguns exemplos que ilustram a utilização de matrizes esparsas em diferentes áreas. A capacidade de aproveitar a estrutura esparsa em dados economiza espaço de armazenamento, melhora a eficiência computacional e possibilita o processamento de grandes conjuntos de dados de forma mais eficiente.

1.1. Lista Circular Simplesmente Encadeada

Uma forma eficiente de representar matrizes esparsas é por meio de uma lista circular encadeada. Nessa estrutura, objetos do mesmo tipo são conectados pelo endereço do próximo objeto, formando uma lista circular. Essa abordagem é escolhida devido à sua melhor utilização de memória, pois a quantidade de elementos não nulos é significativamente menor que a quantidade de elementos nulos. Apesar da pesquisa linear necessária, essa estrutura de dados é mais eficiente para representar matrizes esparsas.

1.2. Incentivo

Para criar essa essa estrutura, é necessário criar um tipo de objeto (Node.h) onde armazena o número da coluna e da linha onde, dois ponteiros, um que guarda o endereço do próximo objeto horizontalmente e outro o endereço do próximo objeto verticalmente, e um campo

do tipo double para armazenar o valor. Esses objetos são partes da listas circulares que se cruzam, portanto, um objeto está presente em duas listas, para criar isso, foram utilizados ”nós cabeça” para marcar o início de cada uma das listas e além disso, os objetos que representam os elementos nulos não devem ser criados. O TAD foi feito em **C++** e possui as seguintes operações : construtor, destrutor, inserção de elementos, remoção de elementos, acesso a elementos e impressão de matrizes, foi implementada uma forma de criação em que, a cada chamada, é criada uma matriz com um identificador iniciando de 0. Além da manipulação na **Main** para somar e multiplicar matrizes. O projeto conta com o TAD Node.h que representa cada objeto dessa matriz

2. Node.h

Nesse arquivo de cabeçalho contém a estrutura **Node** onde essa estrutura representa um nó na matriz esparsa. A estrutura Node possui os seguintes membros, nextdireita: Um ponteiro para o próximo nó na mesma linha da matriz, nextbaixo: Um ponteiro para o próximo nó na mesma coluna da matriz, linha: Um inteiro que representa o número da linha em que o nó está localizado, coluna: Um inteiro que representa o número da coluna em que o nó está localizado, valor: Um número real (double) que representa o valor armazenado no nó. Além disso, o arquivo de cabeçalho contém a declaração do construtor da estrutura Node. O construtor recebe como parâmetros o valor, a linha e a coluna do nó, bem como os ponteiros para os próximos nós na mesma linha e mesma coluna.

Essa estrutura é usada para construir uma matriz esparsa, na qual os nós representam os elementos não nulos da matriz. Através dos ponteiros nextdireita e nextbaixo, é possível percorrer a matriz e realizar operações específicas, como inserção, remoção e busca de elementos.

3. Classe SparseMatrix

Utilizamos o paradigma da programação orientada a objetos, sendo uma forma de organizar o código para melhor entender e manter. Na criação da classe SparseMatrix, escolhemos separar os métodos e atributos dessa classe em 2 arquivos diferentes, um para a declaração da classe e outro para a implementação dos métodos dessa classe.

3.1. SparseMatrix.h

No arquivo SparseMatrix.h tem-se a declaração da classe SparseMatrix e alguns atributos e métodos da classe.

3.2. SparseMatrix.cpp

No arquivo SparseMatrix.cpp contém a implementação dos métodos da classe SparseMatrix, sendo eles, **constutor**, **destrutor**, **insert** (inserir elemento na matriz), **remove** (remover elemento na matriz), **get** (retornar número de colunas da matriz), **print** imprimir a matriz, **getColunas** (retornar o número de colunas), **getLinhas** (retorna o número de linhas), **copia** (cria uma cópia da matriz esparsa atual) .

3.2.1. SparseMatrix

O construtor da classe SparseMatrix recebe o número de linhas e colunas da matriz e inicializa os atributos da classe com esses valores, cria um nó cabeça para cada linha e coluna da matriz, e os conecta entre si, inicializando as listas, sendo sua complexidade O(1).

3.2.2. Destrutor

O destrutor da classe SparseMatrix percorre a matriz e deleta todos os nós da matriz, começando deletando os nós das listas verticais e em seguida os nós cabeça restantes, sendo sua complexidade O(n).

3.2.3. .insert()

Esse método recebe uma linha, uma coluna e um valor e insere esse valor na matriz na posição (linha, coluna). Caso já existe uma valor na posição (linha, coluna), ele é substituído pelo novo valor, caso o valor seja zero, o nó que contém esse valor é removido da matriz. Portanto no pior caso a matriz é densa, com muitos elementos nulos a complexidade é O(n) com n sendo o número de elementos não nulos na matriz .

3.2.4. .remove()

Esse método remove o valor em uma posição específica da matriz esparsa, recebendo uma linha, uma coluna. Ela realiza uma busca para encontrar o nó anterior na mesma linha e na mesma coluna e, em seguida, remove o nó correspondente. A complexidade dessa função depende do número de nós na matriz esparsa. No pior caso, em que a posição não existe na matriz, a complexidade é O(n), onde n é o número de nós na matriz esparsa.

3.2.5. .get()

Esse método recebe como parâmetro a linha e coluna do elemento a ser retomado. O método percorre a matriz até encontrar a linha coluna desejada, e então retorna o valor do elemento. Caso o elemento não exista, o método retorna 0, sendo sua complexidade O(n).

3.2.6. .print()

Esse método imprime a matriz na tela. Ele deve percorrer toda a matriz, campo a campo imprimindo o valor de cada campo ou zero, caso uma posição tenha o valor nulo. Portanto, no pior caso as linhas e colunas são iguais, pois o número de campos a serem percorridos é quadrático em relação ao tamanho da matriz, sendo sua complexidade O(m*n).

3.2.7. .getColunas()

Esse método retorna o número de colunas da matriz, a criação do métodos foi pensada em auxiliar outras funções onde esse informação é necessária, essa função não acessada diretamente ela, apenas as funções sum e multiply.

3.2.8. .getLinhas()

Esse método retorna o número de linhas da matriz, a criação do métodos foi pensada em auxiliar outras funções onde esse informação é necessária, essa função não acessada diretamente ela, apenas as funções sum e multiply.

3.2.9. .copiar()

Esse método faz uma cópia da matriz esparsa selecionada pelo usuário,iterando sobre todos os nós da matriz esparsa atual e insere os valores correspondentes na matriz esparsa copiada. A complexidade dessa função é O(n^2), onde n^2 é o número de nós na matriz esparsa selecionada.

4. Main.cpp

A função main apresenta funções onde, quando executada, trazendo comandos nos quais se apresentam o criar (possível criar uma matriz) onde é criado um array de matrizes a cada escolha do usuário, começando a partir de 0, copiar, inserir(insert), remover (remove), somar(sum). multiplicar, get, show (imprimir na tela o array com as matrizes criadas pelo usuário, junto com o índice de cada uma), carregar (readSparseMatrix) e por fim sair (para fechar o programa).

4.1. Função readSparseMatrix

A função readSparseMatrix lé uma matriz esparsa de um arquivo de texto e retorna um ponteiro para a matriz lida (SparseMatrix*). A função deve lançar uma exceção caso o arquivo não exista ou não esteja no formato correto. Além disso, ela usa a função insert que retorna um erro caso a posição seja inválida, sua análise de complexidade pode variar a depender do número de linhas do arquivo lido.

4.2. Função sum

A função sum recebe duas matrizes esparsas e retorna uma terceira matriz esparsa que é a soma das primeiras. As matrizes só podem ser somadas se tiverem o mesmo número de linhas e colunas, caso isso não aconteça, é disparada uma exceção e vai ser impressa a mensagem ”As matrizes não podem ser somadas”. No pior caso a matriz do resultado da operação é a soma de duas matrizes esparsas com os elementos diferentes de zero, sendo sua complexidade O(n^2). Vale salientar que ela não cria uma nova matriz que só existe enquanto a função sum estiver sendo executada, onde não é possivel acessar essa matriz fora da função sum.

4.3. Função multiply

A função multiply recebe duas matrizes esparsas A e B e retorna uma nova matriz esparsa resultante da multiplicação das matrizes recebidas, obedecendo as regras para tal operação acontecer, caso contrário, é lançada uma exceção caso as matrizes não possam ser multiplicadas. Vale salientar que ela não cria uma nova matriz que só existe enquanto a função multiply estiver sendo executada, onde não é possivel acessar essa matriz fora da função multiply.

4.4. Comandos

Dentro da **Main** temos um sistema de escolha sobre qual ação será feita, criar uma matriz, copiar uma matriz, inserir elementos, remover elementos, somar matrizes, multiplicar matrizes, retornar valores de posições específicas, mostra matriz ou matrizes criadas, carregar arquivos .txt com testes e por fim sair do programa.

5. Divisão de trabalho

Inicialmente procuramos entender o problema proposto, então dividimos as atividades da seguinte forma, Elysson : Costrutor, Destrutor, insert(), getColunas(), getLinhas(), sum(), multiply(), e Gustavo com remove(), get(), print() e copiar(), load(), menu de comandos da main.cpp Para a produção deste documento, resolvemos fazer juntos, usamos uma extensão disponível no VsCode chamada Live Share, onde, escolhemos uma máquina e através dessa extensão, ambos puderam editar o código do projeto, porém, mesmo com funções divididas, algum concluía uma implementação, o outro revisava, testava e modificava, se fosse necessário.

6. Dificuldades

No decorrer da produção do projeto tivemos algumas dúvidas com loops infinitos dentro das funções, fora dúvidas com a complexidade do código visando melhorar, mas em conjuntos conseguimos superar com êxito.

7. Testes Realizados

7.1. Exemplo 1

Matriz 0:

$$\begin{bmatrix} 1 & 2 \\ 5 & 4 \end{bmatrix}$$

Matriz 1:

$$\begin{bmatrix} 1 & 2 \\ 5 & 4 \end{bmatrix}$$

Soma:

$$\begin{bmatrix} 2 & 4 \\ 10 & 8 \end{bmatrix}$$

Multiplicar

$$\begin{bmatrix} 11 & 10 \\ 25 & 26 \end{bmatrix}$$

7.2. Exemplo 2

Matriz 0:

$$\begin{bmatrix} 0 & 1 & 2 \\ 4 & 0 & 1 \end{bmatrix}$$

Matriz 1:

$$\begin{bmatrix} 4 & 2 & 3 \\ 3 & 5 & 9 \end{bmatrix}$$

Soma:

$$\begin{bmatrix} 4 & 3 & 5 \\ 7 & 5 & 10 \end{bmatrix}$$

Multiplicar:

Não e possível multiplicar as matrizes

7.3. Exemplo 3

Criamos dois arquivos **.txt**, A.txt e B.txt, nelas temos no começo o número de linhas e colunas respectivamente, após isso coloca-se os elementos da matriz obedecendo as dimensões dadas na primeira linha, em seguida executamos a função ”carregar”, escohemos os arquivos desejados :

A.txt :

3 3
5 6 10
9 8 30
14 10 19.

B.txt :

3 3
10 9 20
24 5 11
8 9 6

A.txt que seria a Matriz 0:

$$\begin{bmatrix} 5 & 6 & 10 \\ 9 & 8 & 30 \\ 14 & 10 & 19 \end{bmatrix}$$

B.txt que seria a Matriz 1:

$$\begin{bmatrix} 10 & 9 & 20 \\ 24 & 5 & 11 \\ 8 & 9 & 6 \end{bmatrix}$$

Soma:

$$\begin{bmatrix} 15 & 15 & 30 \\ 33 & 13 & 41 \\ 22 & 19 & 25 \end{bmatrix}$$

Multiplicação:

$$\begin{bmatrix} 274 & 165 & 226 \\ 522 & 391 & 448 \\ 532 & 347 & 504 \end{bmatrix}$$

8. Referências

Para a produção desse projeto utilizamos o livro [Calle 2017], e para complementar esse trabalho também utilizamos o site www.leetcode.com para entender melhor ponteiros e suas aplicações.

References

[Calle 2017] Calle, P. D. (2017). ***Introdução à Programação Orientada para Objetos em Linguagem C++***. Universidade de São Paulo - USP, 1th edition.