Matriz Esparsa

Guilherme Lima Moretti e Vitória Ashiley Lopes Ferreira

June 11, 2023

1 Introdução

Este documento faz referencia às funcionalidades de uma Matriz Esparsa, que é uma matriz a qual a maioria de seus elementos possui um valor padrão (como zero) ou são nulos ou faltantes.

Neste caso, para evitar desperdício de memória, existe uma estrutura específica para gerenciar matrizes esparsas, de forma a alocar memória apenas para valores não nulos ou diferentes de 0.

Portanto, este documento tem por intuito mostrar como isto é feito de forma prática, trazendo explicações sobre como a matriz foi desenvolvida e alguns exemplos.

2 Componentes de uma Matriz Esparsa

2.1 Node.h

A classe Node.h em uma matriz esparsa serve para criar nós, que são como os localizadores dos elementos insividuais armazenados em uma matriz. Essa classe possui atributos e um construtor, que serão essenciais na desenvoltura da matriz. Com os Nodes em uma matriz esparsa, é possivel localizar elementos não nulos e ignorar os zeros presentes, ajudando assim a não desperdiçar memória desnecessária.

Os atributos da classe são:
Node* right
Node* bottom
unsigned line
unsigned column
double value

2.2 Matrix.h

A classe Matrix.h é um cabeçalho para um objeto chamado SparseMatrix, a classe permite manipular as matrizes esparsas adicionadas. Ela possui como membros private: lines, columns e mhead e metodos public: getLines, getColumns, verifyCoord, insert, get e print.

Os atributos da classe são:
unsigned line
unsigned column
Node* mhead

2.3 Matrix.cpp

A Matriz.cpp é a função executável da classe Matriz.h, ela implementa todas as funções criadas para que a matriz possa ser construida, destruida, tenha um elemento inserido, seja retornada e seja printada na tela.

Na Matriz.cpp, cada função tem uma complexidade.

A Função SparseMatrix deve ser capaz de construir os nós da matriz.

A Função SparseMatrix deve ser capaz de destruir a matriz e seus nós.

A Função VerifyCoord verifica se as coordenadas passadas existem dentro da matriz, caso não, emite uma mensagem de erro.

A Função Insert deve ser capaz de inserir ou modificar um elemento na matriz.
A Função Get deve retornar os elementos da matriz.
A Função Print deve printar a matriz na tela.

2.4 Main.cpp

O Arquivo Main.cpp possui algumas funções como: readSparseMatrix,showAllMatrices, sum e multiply. Além disso, também possui a função main, que é onde toda a matriz pode ser testada e executada, de forma a apresentar erros ou não.

A função readSparseMatrix deve ser capaz de ler uma matriz de um arquivo .txt e criá-la. A função showAllMatrices deve ser capaz de exibir todas as matrizes existentes no vector. A função createMatrix() deve ser capaz de criar uma matriz vazia de acordo com o valor de linhas e colunas que o usuario desejar.

A função sum deve ser capaz de ler duas matriz e, caso sejam de mesmo tamanho (ex: 3 x 3), somar seus elementos.

A função multiply deve ser capaz de ler duas matriz e, caso a linha da primeira matriz seja igual a coluna da segunda matriz (ex: (3,2) * (2,3), multiplicará seus elementos resultando em outra matriz.

A função main, apenas chama o menu principal.

3 Complexidade

Cada função tem seu nível de complexidade, que descreve quanta memória é necessária para executar uma sessão de código no seu pior caso. Nas funções insert, get e sum, seus níveis de complexidade são:

Insert - O(n+m): O pior caso é quando o usuario escolhe o ultimo elemento da coluna e da linha da matriz. Onde a linha e a coluna estão preenchidas de elementos(nós).

Get - O(n+m): O pior caso é quando o usuario escolhe o ultimo elemento da coluna e da linha da matriz. Onde a linha e a coluna estão preenchidas de elementos(nós).

Sum - O(m*n(m+n)): O pior caso dessa função ocorre quando as matrizes de entrada possuem o mesmo número de linhas e colunas, pois é necessário percorrer todos os elementos dessas matrizes para realizar a soma e inserir os valores na matriz resultante.

4 Dificuldades

As dificuldades presentes se deram em funções específicas, foram elas: destrutor, insert e print.

4.1 Destrutor

A dificuldade presente no destrutor foi a exclusão dos elementos dentro da matriz e a exclusão dos nós a direta do m-head (ou seja, os nós das colunas). A dificuldade se deu porquê não se sabia uma forma de excluir todos ao mesmo tempo.

A solução encontrada foi percorre a matriz, começando pelo nó cabeça (mhead), e exclui cada nó individualmente, juntamente com os nós adjacentes a ele. O processo continua até que todos os nós tenham sido excluídos. No final, o mhead também é excluído.

4.2 Insert

A maior dificuldade encontrada em fazer a função insert foi conseguir inserir um elemento (criar um nó) em uma localização onde os ponteiros right e bottom não apontavam para o nó cabeça. Por exemplo, tem-se uma matriz 3x3 totalmente preenchida com elementos diferentes de zero, exceto a posição 2x2, para criar um novo nó é necessário chegar no nó anterior da posição que desejo inserir (no caso o 2x1 e 1x2), criar o novo nó e fazer com que os ponteiros apontem para o novo elemento. Nessa situação, ao criar o nó, outros elementos da matriz estavam sendo modificados juntos (devido o erro na manipulação de ponteiros), o que não condizia com a finalidade do programa.

4.3 Print

Não houveram gandes dificuldades, mas na função print dentro do linux, a impressão estava ocorrendo em notação científica, então, para consertar o problema, foi necessário o uso da biblioteca stringstream ,que ajuda na manipulação de strings, sendo assim, podendo tratar o valor no momento de exibir, é possivél resolver o problema.

Outra dificuldade foi para manter os elemntos da matriz alinhados, para isso foi utilizada a função "setw() e left", que ajudou no melhor alinhamento para uma melhor visualização da matriz.

5 Divisão do Trabalho entre a dupla

A divisão do trabalho foi feita entre a dupla de maneira igualitária, o código ficou dividido para que um criasse certas funções enquanto o outro criaria o resto.

Vitória: Responsável pelo corpo do código e a implementação das funções SparseMatrix(construtor), SparseMatrix(destrutor), get, readSparseMatrix e as classes Node.h e Matriz.h. Também ficou responsável pela revisão do documento de relatório.

Guilherme: Responsável por implementar as funções insert, print, sum, multiply, ShowAllMatrices e a main. Também ficou responsável por criar o "Menu Principal" na função main, que guia a pessoa que está interagindo com o programa.

Os dois participantes ficaram responsáveis por relatar sua parte no relatório final e fazer juntos a introdução, relato de testes e divisão entre a dupla.

6 Comandos

6.1 Menu Principal

Ao acessar o código, irão aparecer na tela algumas opções, as quais deve ser escolhida uma por vez, serão elas:

(c) criar matriz vazia
(a) criar matriz por meio de um arquivo
(l) listar matrizes
(e) editar matrizes
(s) somar matrizes
(m) multiplicar matrizes
(q) sair do programa

6.2 Menu Matriz

Ao escolher uma das opções, irá aparecer uma nova tela com o que deseja ser feito, por exemplo, ao escolher ((c) criar matriz vazia), será exibida uma tela onde o usuário poderá informar qual o tamanho da matriz que deseja.

Em seguida, aparecerá o menu de funções disponíveis para a matriz, serão elas:

(p) exibir matriz
(e) exibir elemento da matriz
(m) modificar elemento da matriz
(d) deletar matriz
(s) voltar para menu principal

Ao escolher uma nova opção, uma função nova será realizada.

Ao fim, se desejar realizar algo mais, basta retornar ao "Menu principal", se não, basta sair do programa.

7 Testes

7.1 Teste 1- Inserir:

Exemplo 1: Inserir na matriz vazia

Matriz inicial:		Insert:
0.0000	0.0000	1 1 3
0.0000	0.0000	2 2 6

Matriz final:

3.0000	0.0000
0.0000	6.0000

Exemplo 2: Inserir na matriz com elementos

Matriz inicial:		Insert:
3.0000	0.0000	1 1 7
0.0000	6.0000	2 1 4

Matriz final:

7.0000	0.0000
4.0000	6.0000

7.2 Teste 2- Somar:

Exemplo 1:

Entrada 1	Entrada 2
4 4	4 4
1 2 4	1 1 1
4 4 5	1 2 2
2 3 7	1 3 3
1 1 6	1 4 4
3 2 9	2 1 6
1 4 0	2 2 0
2 1 0	2 3 0
4 1 0	2 4 7
4 2 3	3 1 1
	3 2 9
	3 3 9
	3 4 2
	4 1 0
	4 2 5
	4 3 0
	4 4 0

Saída:

7.0000	6.0000	3.0000	4.0000

6.0000 0.0000 7.0000 7.0000
1.0000 18.0000 9.0000 2.0000
0.0000 8.0000 0.0000 5.0000

Exemplo 2:

Entrada 1
3 3
1 1 4
1 2 5
1 3 0
2 1 13
2 2 95
2 3 0
3 1 0
3 2 81
3 3 0

Entrada 2
3 3
1 1 0
1 2 8
1 3 45
2 1 0
2 2 6
2 3 0
3 1 0
3 2 67
3 3 9

Saída:

4.0000 13.0000 45.0000
13.0000 101.0000 0.0000
0.0000 148.0000 9.0000

7.3 Teste 3 - Multiply

Exemplo 1:

Entrada 1
2 2
1 1 3
1 2 7
2 1 1
2 2 0

Entrada 2
2 2
1 1 2
1 2 3
2 1 4
2 2 6

Saída:

34.0000 51.0000
2.0000 3.0000

Exemplo 2:

Entrada 1
2 2
1 1 7
1 2 2
2 1 0
2 2 4

Entrada 2
2 4
1 1 0
1 2 7
1 3 2
1 4 2
2 1 0
2 2 5
2 3 0
2 4 6

Saída:

0.0000 59.0000 14.0000 26.0000

0.0000 20.0000 0.0000 24.0000

7.4 Teste 4 - Destrutor

Entrada:

7.0000 0.0000
4.0000 6.0000

(d) Deletar matriz

Saída:

Matriz foi apagada com sucesso.