Relatório sobre Matrizes Esparsas

Letícia Pinheiro d. Oliveiral[1], **Lucas Mauricio Braga**[2]

[1]Universidade Federal do Ceará (UFC)
Av. José de Freitas Queiroz, 5003 – Cedro – Quixadá – Ceará 63902-580

(leticiapinheiro3137,lucasmb.7)@gmail.com

Abstract. *This work aimed to create a program in the C++ language, which allows the creation and manipulation of sparse matrices. It is able to create arrays, add, query, sum and multiply the values present in them, and erase them.*

Resumo. *Este trabalho teve como objetivo criar um programa na linguagem C++, que permite a criação e manipulação de matrizes esparsas. Ele é capaz de criar matrizes, adicionar, consultar, somar e multiplicar os valores presentes nas mesmas, além de apagá-las.*

1. Introdução

Matrizes esparsas são matrizes nas quais a maioria das posições é preenchida por zeros, ou são nulos ou faltantes. Para essas matrizes, podemos economizar um espaço significativo de memória se apenas os termos diferentes de zero forem armazenados, pois seria um desperdício gastar m x n posições de mémoria, sendo que apenas uma pequena parcela dos elementos é diferente de zero.

As operações usuais sobre essas matrizes (inserir, somar, multiplicar, printar, verificar, ler matriz) também podem ser feitas em tempo muito menor se não armazenarmos as posições que contêm zeros. Uma maneira eficiente de representar estruturas com tamanho variável e/ou desconhecido é com o emprego de alocação encadeada, utilizando listas, a fim de evitar, também, o desperdício de mémoria, um arranjo com listas ligadas é usado.

Cada coluna da matriz será representada por uma lista linear circular com uma célula cabeça. Da mesma maneira, cada linha da matriz também será representada por uma lista linear circular com uma célula cabeça.

2. Desenvolvimento

O projeto contém quatro arquivos, sendo o SparserMatrix.cpp, a Main.cpp, o Node.h e o SparserMatrix.h. O SparserMatrix.cpp guarda a classe construtora, que é responsável por criar uma matriz m x n, onde cada linha da matriz será representada por uma lista ligada que só conterá elementos com valores diferentes de zero, assim, teremos um arranjo de listas ligadas. Para inicializar nossa matriz esparsa, nós precisamos: acertar o valor dos campos linhas e colunas, isto é, a ordem da matriz passada pelo usuário. Precisamos também criar o arranjo de listas ligadas e iniciar cada posição do arranjo com o valor NULL - indicando que cada lista está vazia.

Guarda também o destrutor que, como visto em POO, destrói o objeto criado anteriormente. Ainda dentro desse arquivo, é possível encontrar uma função para verificar, do tipo booleana, que retorna falso se os valores inseridos forem menores que zero e verdadeiro se os valores forem maiores que o tamanho de linhas e colunas.

Na função DeleteNode, basicamente vai de 0,0 até 0,i, então ele vai procurar o node por toda essa coluna i, caso ele ache o node, uma flag é ativada, apaga ele nessa posição da coluna e faz o antecessor apontar para o sucessor. Se a flag estiver ativa, significa que ele existe e o programa faz essa mesma busca na coluna j, só que agora ele deleta esse node, ou seja, só vai deletar se a flag estiver ativa e isso só acontece se ele existir.

Há, também, uma função de insert, que é responsável por inserir numéros na matriz, o usuário nos passa: o endereço da matriz, a linha, a coluna e o valor a ser colocado na respectiva posição da matriz. Se não houver nenhum nó na posição e o valor for diferente de zero temos que inserir um novo nó na respectiva lista ligada.

Foi criada uma função get, do tipo double, que retorna um nó, caso ele exista na posição m e n, ou 0, caso não exista

A função getRow serve para retornar um inteiro referente ao número de linhas da matriz. O getColumn retorna um inteiro referente ao número de colunas da matriz.

A função print irá imprimir toda a matriz (m x n) com a ajuda de dois nodes, um deles (pos.node) irá percorrer toda a coluna (0, j) enquanto o aux.node irá percorrer as linhas (i,j) imprimindo o valor do nó i, j caso ele exista e 0.0 caso não exista. E com essa função, encerra-se o arquivo SparseMatrix.

Indo para o arquivo da Main.cpp. Nesse arquivo, é possível encontrar a função readSparseMatriz, Sum, Multiply e main.

Na função readSparseMatriz, é passado por parâmetro uma string que lê o nome do arquivo, que será o SparseMatriz, caso não ache o arquivo, a função irá retornar uma mensagem dizendo que não achou o arquivo.

Na função Sum, vai ser passado duas matrizes por parâmetro e utilizando o node como auxiliar, onde irá ser chamada a função getRow para ler as linhas e depois a getColumn para ler as colunas, em seguida criando uma nova matriz para receber a soma das duas iniciais.

Em questão da função Multiply, são passados por parâmetro duas matrizes, primeiro é feito uma verificação sobre o número de colunas e linhas, caso seja diferente, não será possível realizar a operação. Então, é pego uma matriz c, com número de colunas e linhas iguais e feito a operação de multiplicação.

Por último, a função principal, na main é chamada as outras funções, um menu interativo é criado e é impresso a matriz.

Os arquivos auxiliares, SparseMatrix.h e Node.h, auxiliam nas funções principais, e neles contém a matriz e suas funções em uma classe e no outro arquivo, node.h, a struct node.

3. Desafios Encontrados

A dupla fez metade, metade, os dois foram se auxiliando e pedindo ajuda aos monitores e colegas. Foi um projeto, um tanto quanto complexo de ser executado que, no entanto, acrescentou muito na compreensão da matéria de Estrutura de Dados. Ensinou que para ter um bom programa, que é funcional, deve existir o equilíbrio no uso de memória e

tempo gasto. Para as operações foi compreendido e percebeu-se que esse equilíbrio é uma das bases de Estrutura de Dados.

Procuramos ajuda dos monitores e de colegas que já fizeram a cadeira, foram feitos grupos de estudo para podermos tirar nossas dúvidas. Acreditamos que a parte mais complexa foi a das listas circulares, que teve muita pesquisa e visualização de vários vídeos para entendermos e podermos desenvolver o projeto. O monitor Gabriel foi bastante importante, nos ajudou e explicou como funcionava as listas circulares e como poderíamos criá-las para desenvolver o código.

4. Resultados

Matriz esparsa é considerada como uma solução para o problema de representar uma matriz 2-D com a maioria dos elementos nulos. Podemos usar representação de matriz ou representação de lista encadeada para armazenar elementos de tal matriz e diminuir a complexidade, assim como o tempo do programa. Além disso, podemos economizar muito espaço armazenando apenas 0 elementos.

Usamos alguns testes para verificarmos se o programa estava rodando bem.

Matriz A:

```
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
```

Matriz B:

```
4 4
1 1 50.0
1 2 10.0
2 1 30.0
2 3 -20.0
4 4 -5.0
```

Matriz C:

```
2 3
1 1 2.0
2 1 3.0
1 2 1.0
2 2 0.0
1 3 4.0
2 3 5.0
```

Matriz D:

```
2 2
```

1 1 3.0
1 2 2.0
2 1 1.0
2 2 4.0

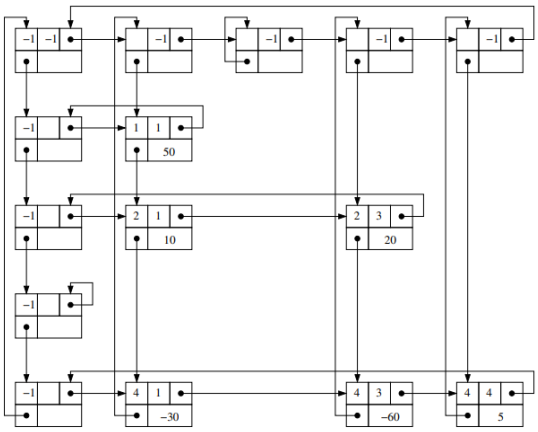

Figure 1. Matriz esparsa

5. Referências

Usamos como refêrencias alguns documentos e vídeos, os [1], [2] e [3] foram alguns dos documentos que usamos no desenvolvimento do nosso projeto.

References

[1] DOS SANTOS, L. F. Método dual simplex para problemas canalizados com estruturas de dados eficientes.

[2] MENOTTI, D. Trabalho prático 2 listas/filas/pilhas.

[3] PAULA, F., SILVA, E., AND MESQUITA, R. Sistema de matrizes esparsas orientado por objetos em c+. In *CBmag'95-Congresso Brasileiro de Eletromagnetismo* (1995), pp. 14–17.