Matrizes Esparsas
Relatório do Projeto de Estrutura de Dados

Atílio Gomes Luiz[1], Jennifer Marques de Brito[2], João Guilherme Lira dos Santos[2]

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

[2]Campus Quixadá – Universidade Federal do Ceará (UFC)
Quixadá, CE – Brazil

gomes.atilio@ufc.br, jenniferbrito@alu.ufc.br, joao.lira@alu.ufc.br

Abstract. *This project, developed in the Data Structures course, explores the implementation of sparse matrices using singly linked lists, based on the project proposed by Nivio Ziviani. The developed data structure supports fundamental operations such as insertion, retrieval, addition, and multiplication of elements, along with an analysis of the worst-case complexity of some functions. The report describes the entire data structure, design decisions, challenges faced, task distribution within the pair, and the tests conducted. The proposed implementation demonstrates the efficiency of sparse matrices in optimizing computational resources.*

Resumo. *Este projeto, desenvolvido na disciplina de Estrutura de Dados, explora a implementação de matrizes esparsas por meio de listas encadeadas simples, com base no projeto proposto por Nivio Ziviani. A estrutura de dados desenvolvida suporta operações fundamentais, como inserção, recuperação, soma e multiplicação de elementos, com análise detalhada de sua complexidade no pior caso. O relatório descreve toda a estrutura de dados, as decisões de projeto, dificuldades enfrentadas, a divisão de tarefas entre a dupla, além de listar os testes realizados. A implementação proposta comprova a eficiência das matrizes esparsas para otimização de recursos computacionais.*

1. Introdução

O presente relatório descreve a implementação de uma estrutura de dados para representar matrizes esparsas, utilizando a linguagem de programação C++. Uma matriz esparsa é caracterizada por possuir um grande número de elementos nulos, sendo eficiente armazená-la sem alocar espaço para todos os elementos. A abordagem baseada em listas encadeadas com sentinelas permite manter a circularidade e simplificar operações, como inserção, remoção e atualização de valores de forma eficiente.

2. Estrutura interna

Cada elemento da matriz é representado por um nó da estrutura **Node**, que armazena o valor associado à posição na matriz, as coordenadas (linha e coluna) e ponteiros para os nós à direita e abaixo. A matriz em si é gerenciada pela classe **SparseMatrix**, que encapsula os métodos para manipulação eficiente dos dados.

2.1. Struct Node

A estrutura 'Node' é a base da implementação da matriz esparsa. Cada nó contém as seguintes informações:

Atributos:

- 'value': um valor do tipo 'double' que armazena o valor da célula da matriz (nó).
- 'direita': um ponteiro para o próximo nó na mesma linha.
- 'abaixo': um ponteiro para o próximo nó na mesma linha.
- 'linha' : um inteiro que representa o índice da linha do nó.
- 'coluna': um inteiro que representa o índice da coluna do nó.

O construtor da estrutura 'Node' inicializa esses atributos, permitindo a criação de nós que podem ser encadeados horizontal e verticalmente.

2.2. Classe SparseMatrix

A classe 'SparseMatrix' encapsula a lógica para manipulação de matrizes esparsas. Os principais atributos e métodos da classe são:

Atributos:

- 'sentinela': um ponteiro para um nó que atua como um marcador central da matriz.
- 'linsentinela' e 'colsentinela': ponteiros para sentinelas que representam as linhas e colunas da matriz, respectivamente.
- 'linhas' e 'colunas': inteiros que armazenam o número de linhas e colunas da matriz (dimensões).

Construtores:

- O construtor padrão inicializa a matriz com um sentinela circular.
- O construtor que recebe dimensões ('n' e 'm') inicializa a matriz com as dimensões especificadas e cria as sentinelas para linhas e colunas.

Métodos:

- 'insert(int i, int j, double valor)': insere um valor na posição (i,j) da matriz. Se o valor for zero, a operação é ignorada.
- 'get(int i, int j)': recupera o valor na posição (i,j). Se não houver valor armazenado, retorna zero.
- 'print()': imprime a matriz no terminal, mostrando os valores armazenados e substituindo os zeros por espaços.
- 'clear()': desaloca a memória utilizada pela matriz, liberando todos os nós.
- ' SparseMatrix()': destrutor que garante a liberação da memória ao destruir a instância da matriz.

2.3. Main.cpp

Funciona como o ponto de entrada do programa, gerenciando a interface de interação com o usuário e coordenando as operações com as matrizes esparsas.

Métodos:

- 'readSparseMatrix(SparseMatrix m, const std::string nomeDoArquivo)': lê e carrega dados de uma matriz esparsa a partir de um arquivo de texto, permitindo assim a criação dinâmica de uma matriz com base em informações externas.

- 'salvarMatrizEmArquivo(const SparseMatrix matriz, const string nomeDoArquivo)': salva uma matriz esparsa em um arquivo de texto.
- 'sum(const SparseMatrix A, const SparseMatrix B)': realiza a soma de duas matrizes esparsas que possuam dimensões compatíveis.
- 'multiply(const SparseMatrix A, const SparseMatrix B)': realiza a multiplicação de duas matrizes esparsas, desde que o número de colunas da primeira matriz seja igual ao número de linhas da segunda.

3. Aplicações

A função main() é responsável pelas aplicações da estrutura implementada, testa as principais funcionalidades. Dentro dela, são realizadas as seguintes operações:

1. **Criação da matriz esparsa:** Inicializa uma instância da classe SparseMatrix.
2. **Inserção de valores:** Insere os elementos em posições específicas para demonstras a capacidade da estrutura em armazenar dados de forma eficiente.
3. **Recuperação de valores:** Demonstra o acesso aos elementos armazenados através do métodos get().
4. **Impressão da matriz:** Exibe a matriz com seus valores armazenados e elementos nulos como zero, usando o método print().
5. **Operações matemáticas:** Testa a soma e a multiplicação de matrizes esparsas, mostrando os resultados das operações no terminal.
6. **Limpeza de memória:** Chama o método clear() para desalocar a memória ocupada pela matriz.
7. **Salvamento:** Permite exportar uma matriz esparsa para um arquivo de texto no formato padrão, onde a primeira linha contém as dimensões da matriz (número de linhas e colunas), e as linhas seguintes contêm as células não nulas com suas respectivas coordenadas e valores. Além disso, registra o nome do arquivo no log de testes.

3.1. Direcionamento de comandos:

1. **Help** - Exibe um painel de ajuda com a listagem de todos os comandos e suas funções.
   Como usar: *Digite* ***help*** *ou* ***Help***
2. **Exit** - encerra o programa.
   Como usar: *Digite* ***exit*** *ou* ***Exit***
3. **Create** - Cria uma nova matriz esparsa com o número de linhas e colunas especificados.
   Como usar: *Digite* ***create*** *ou* ***Create***
4. **Show** - Imprime a matriz de um índice específico na tela.
   Como usar: *Digite* ***show*** *ou* ***Show*** *seguido do índice da matriz (por exemplo, show 0).*
5. **Showidx** - Mostra os índices (e dimensões) de todas as matrizes atualmente armazenadas.
   Como usar: *Digite* ***showidx*** *ou* ***Showidx***
6. **Sum** - Realiza a soma de duas matrizes esparsas que possuem as mesmas dimensões.
   Como usar: *Digite* ***sum*** *ou* ***Sum*** *seguido dos índices das duas matrizes (por exemplo, sum 0 1). Obs.: Após a soma, há a possibilidade de armazenar a matriz resultando.*

7. **Multiply** - Realiza a multiplicação de duas matrizes esparsas, sendo necessário que o número de colunas da primeira seja igual ao número de linhas da segunda.
   ***como usar**: Digite **multiply** ou **Multiply** seguido dos índices das matrizes (por exemplo, multiply 0 1). Obs.: Assim como na soma o usuário poderá escolher se deseja armazenar o resultado.*
8. **Clear** - Limpa (zera) a matriz armazenada no índice informado, removendo todos os elementos.
   ***Como usar:** Digite **clear** ou **Clear** seguido do índice da matriz (por exemplo, clear 0).*
9. **Read** - Lê uma matriz a partir do arquivo "mtx.txt" e armazena ela a estrutura.
   ***Como usar:** Digite **read** ou **Read** seguido do nome do arquivo, se for diferente do padrão (por exemplo, read mtx.txt).*
10. **Update** - Atualiza o valor de uma célula específica em uma matriz.
    ***Como usar:** Digite **update** ou **Update** seguido do índice da matriz, número da linha, número da coluna e o novo valor (por exemplo, update 0 2 3 5.6 atualiza a posição (2,3) da matriz de índice 0 para 5.6).*
11. **Erase** - Apaga todas as matrizes atualmente armazenadas no programa, liberando a memória utilizada.
    ***Como usar:** Digite **erase** ou **Erase**.*
12. **Save** - Salva a matriz armazenada no índice informado em um arquivo.
    ***Como usar:** Digite **save** ou **Save**.*

4. Divisão do trabalho

A divisão de trabalho entre a dupla foi feita de forma colaborativa e equilibrada. Ambos participaram ativamente em todas as etapas do desenvolvimento, desde a criação das funções fundamentais até a implementação das funcionalidades de leitura, soma, multiplicação e gerenciamento de memória. Se um implementava uma função, o outro sempre revisava e, se necessário, realizava ajustes para melhorar a performance ou corrigir erros.

Além disso, a elaboração do relatório também foi um esforço conjunto, cada parte do código e as decisões de projeto adotadas foram discutidas e alinhadas.

4.1. Dificuldades

Durante o desenvolvimento do projeto, foi encontrado algumas dificuldades. Entre elas: implementar a alocação e desalocação dinâmica de memória para os nós da matriz esparsa e garatir que não houvesse vazamentos de memória ou acessos inválidos. Visando evitar esses impasses houve a implementação da função clear() para auxiliar a liberação da memória alocada, além dos construtores e destrutores.

Lidar com situações como inserção de elementos em posições já ocupadas, remoção de elementos inexistentes e manipulação de linhas ou colunas inteiras vazias também foi um desafio durante a implementação.

No entanto, uma das principais dificuldades enfrentadas foi na implementação da função readSparseMatrix, responsável por ler e carregar os dados de uma matriz esparsa a partir de um arquivo de texto. Apesar de a função ter sido projetada para facilitar a importação de dados externos, encontramos problemas que impediam seu funcionamento

correto, resultando em erros durante a execução. Ao tentar utilizar essa função, o programa apresentava mensagens de erro indicando falhas na leitura dos dados ou na inserção dos elementos na matriz.

5. Testes executados

6. Listagem de Testes Executados

A seguir, a listagem de testes feitos para validar as funcionalidades da implementação da matriz esparsa:

- **TC001 - Criação da Matriz**
  - **Funcionalidade:** Criação de uma matriz esparsa de 5x5.
  - **Entrada/Comando:** `create 5 5`.
  - **Resultado Esperado:** Nova matriz esparsa de 5x5 criada com sucesso.
  - **Resultado Obtido:** Matriz criada conforme esperado.
  - **Status:** Sucesso.
- **TC002 - Inserção de Elemento**
  - **Funcionalidade:** Inserir um elemento na posição (1,1) com valor 5.0.
  - **Entrada/Comando:** `update 0 1 1 5.0`.
  - **Resultado Esperado:** Elemento 5.0 inserido na posição (1,1).
  - **Resultado Obtido:** Elemento inserido com sucesso.
  - **Status:** Sucesso.
- **TC003 - Recuperação de Elemento**
  - **Funcionalidade:** Recuperar o valor armazenado na posição (1,1).
  - **Entrada/Comando:** Chamada ao método `get(1,1)`.
  - **Resultado Esperado:** Valor 5.0 retornado.
  - **Resultado Obtido:** Valor retornado corretamente.
  - **Status:** Sucesso.
- **TC004 - Impressão da Matriz**
  - **Funcionalidade:** Exibir a matriz com os elementos inseridos e zeros para as posições não definidas.
  - **Entrada/Comando:** `show 0`.
  - **Resultado Esperado:** Matriz impressa com os valores corretos.
  - **Resultado Obtido:** Matriz exibida conforme esperado.
  - **Status:** Sucesso.
- **TC005 - Soma de Matrizes**
  - **Funcionalidade:** Realizar a soma de duas matrizes esparsas.
  - **Entrada/Comando:** `sum 0 1`.
  - **Resultado Esperado:** Matriz resultante com a soma elemento a elemento.
  - **Resultado Obtido:** Soma realizada corretamente.
  - **Status:** Sucesso.
- **TC006 - Multiplicação de Matrizes**
  - **Funcionalidade:** Realizar a multiplicação de duas matrizes esparsas.
  - **Entrada/Comando:** `multiply 0 1`.
  - **Resultado Esperado:** Matriz resultante da multiplicação.
  - **Resultado Obtido:** Multiplicação realizada corretamente.
  - **Status:** Sucesso.
- **TC007 - Leitura de Arquivo**

  - **Funcionalidade:** Carregar dados de uma matriz esparsa a partir de um arquivo.
  - **Entrada/Comando:** `read mtx.txt`.
  - **Resultado Esperado:** Dados importados conforme o conteúdo do arquivo.
  - **Resultado Obtido:** Dados importados conforme esperado (ou mensagem de erro se o arquivo não for encontrado).
  - **Status:** Sucesso/Erro*.
- **TC008 - Limpeza de Matriz**
  - **Funcionalidade:** Limpar (zerar) a matriz removendo todos os elementos.
  - **Entrada/Comando:** `clear 0`.
  - **Resultado Esperado:** Matriz limpa, sem dados antigos.
  - **Resultado Obtido:** Matriz limpa conforme esperado.
  - **Status:** Sucesso.
- **TC009 - Apagar Todas as Matrizes**
  - **Funcionalidade:** Remover todas as matrizes armazenadas e liberar a memória.
  - **Entrada/Comando:** `erase`.
  - **Resultado Esperado:** Todas as matrizes removidas e memória liberada.
  - **Resultado Obtido:** Operação realizada com sucesso.
  - **Status:** Sucesso.
- **TC010 - Comando de Ajuda**
  - **Funcionalidade:** Exibir o painel de ajuda com a listagem de comandos.
  - **Entrada/Comando:** `help`.
  - **Resultado Esperado:** Painel de ajuda exibido.
  - **Resultado Obtido:** Painel de ajuda apresentado.
  - **Status:** Sucesso.
- **TC011 - Encerramento do Programa**
  - **Funcionalidade:** Finalizar a execução do programa.
  - **Entrada/Comando:** `exit`.
  - **Resultado Esperado:** Programa encerrado sem erros.
  - **Resultado Obtido:** Programa finalizado corretamente.
  - **Status:** Sucesso.

Observações

- Cada teste possui um identificador único (por exemplo, TC001, TC002, etc.), o que facilita a referência e a discussão dos casos de teste.
- Se houver discrepâncias entre o resultado esperado e o obtido, o status deve ser alterado para **Falha** e deve-se incluir uma breve explicação.
- O teste TC007 pode apresentar um status de Sucesso ou Erro, dependendo se o arquivo de dados está correto ou se ocorreu algum problema na leitura.

7. Análise de Complexidade

7.1. insert

Teorema: A função insert possui complexidade O(n+m), sendo n o número de linhas e m o número de colunas.

Prova: As duas principais operações realizadas pela função insert são a busca na linha, onde percorre n linhas no pior dos casos para encontrar a linha requerida pela função. A segunda operação é a busca pela coluna, onde percorre uma quantidade m de colunas no pior dos casos para encontrar a celula correta. As demais operações possuem complexidade constante e são irrelevantes para n e m grandes o suficiente.

conclusão: A complexidade da função é O(n+m).

7.2. get

Teorema: A função get() possui complexidade O(n+m).

Prova: A função get() realiza duas principais operações. Ao fazer a busca pela celula (i, j), a função realiza uma busca pela linha, onde percorre um total de n linhas no pior dos casos. A segunda operação consiste na busca de colunas, que percorre um total de m colunas no pior dos casos até encontrar a celula correta.

conclusão: A complexidade da função é O(n+m).

7.3. sum

Teorema: a função Sum tem complexidade O(n×m×(m+n)), sendo n o número de linhas e m de colunas de duas matrizes esparsas A e B.

Prova:

1. Função Sum: A função sum realiza a soma de duas matrizes esparsas A e B, que possuem tamanhos iguais de dimensão n×m. Para cada celula (i, j), a função executa duas operações ”get()”, que possuem complexidade O(m+n), e uma operação ”insert()” em outra matriz, que possui complexidade O(m+n).
2. Numero de células: A matriz A e a matrix B possuem m×n celulas, e o loop passa por cada celula ao percorrer n linhas e m colunas.
3. Como o total de operações é n×m e como cada iteração possui complexidade O(m+n), a complexidade total pode ser escrita como O(n×m×(m+n))

Conclusão: complexidade = O(n×m×(m+n))

References

[1] Pantuza, G. (2016). Tipos abstratos de dados - Lista encadeada (Linked list). Recuperado de `https://blog.pantuza.com/artigos/tipos-abstratos-de-dados-lista-encadeada-linked-list`

[2] Baptista, C. S. Estruturas de Dados. Recuperado de `https://www.facom.ufu.br/~claudio/Cursos/Antigos/EDxxxx/Artigos/bcc-ed01.pdf`

[3] Carvalho, A. C. P. L. F. Matrizes Esparsas. Recuperado de `http://www.each.usp.br/digiampietri/ed/aula14.pdf`

[4] UNIVESP. (n.d.). Matrizes Esparsas em C [Vídeo]. YouTube. Recuperado de `https://www.youtube.com/watch?v=C_ePgrEbLs0`

[5] Modesto Comp. (n.d.). Matrizes Esparsas [Vídeo]. YouTube. Recuperado de `https://www.youtube.com/watch?v=bREMw479MS8`

[6] Cassol, V. (n.d.). C: Leitura de Arquivo Texto. Medium. Recuperado de https://vinicassol.medium.com/c-leitura-de-arquivo-texto-bae9d91febe4

[7] Deitel, P., Deitel, H. (2016). C: How to Program with an Introduction to C++ (8ª ed.). Pearson Education.