Documentação do Projeto Matrizes Esparsas

Gustavo Gurgel Medeiros

5 de novembro de 2022

1 Motivação

Segundo (LE MENESTREL, 2022) no artigo "Sparse Matrices: Why They Matter for Machine Learning and Data Science", quando estamos representando dados utilizando matrizes, podemos quantificar o número de espaços vazios contidos nelas. Uma matriz que é construída em sua totalidade de zeros é chamada de uma matriz esparsa.

Esse tipo de matriz pode ser empregada em ramos complexos de áreas como **engenharia** e **inteligência artificial**. Porem, uma abordagem simples inspirada em um exemplo do artigo anteriormente citado é a seguinte. Tenha em mente a matriz abaixo que repesenta o **raking de produtos** em um site hipotético:

Figura 1:Tabela de avaliação

	PRODUTO_1	PRODUTO_2	PRODUTO_3
Gustavo	★★★★		
Mário		★	
Joabe			
Pedro			★★

Fonte: Autoria própria

Imagine que dentro da matriz cada estrela/nota de cada produto é representado por um valor inteiro, e zero representa uma avaliação não feita ou nula. Tendo em conta isso, Mário por exemplo, fez apenas uma avaliação referente ao produto 2, já Joabe, nem se quer fez uma avaliação. Dessa matriz $\frac{3}{4}$ dos valores são de **avaliações nulas**, ou seja, de zeros.

Agora imagine esse exemplo em alta escala, com uma matriz de milhares de clientes e avaliações com essa densidade de valores nulos. Alocar isso iria gerar um demanda muito grande do sistema, demanda essa que não seria necessária se o site alocasse os dados em forma de uma **matriz esparsa**.

2 Requisitos do Trabalho

O trabalho se baseia na implementação de uma matriz esparsa com um TAD utilizando classes. Essa matriz em específico é composta de n_1 linhas e n_2 colunas, onde cada linha e coluna, por si só representa uma lista **encadeada circular**. Foi pedido a criação de um construtor e um destrutor, além dos métodos **insert, get e print**. Foi informado que se necessário, seria livre a criação de métodos auxiliares adicionais.

Além da própria matriz, foi requisitado a criação de um driver, para fins de teste da estrutura criada, com as seguintes funções:

- **readSparseMatrix**;
  Cria uma matriz com base em um arquivo de texto
- **sum**;
  Soma duas matrizes e retorna a matriz resultante
- **multiply**;
  Duplica duas matrizes e retorna a matriz resultante

Como é de se imaginar, a função **readSparseMatrix** necessita do conhecimento da biblioteca **fstream** (biblioteca do c++ que trabalha com operações de E/S em arquivos). Logo, deixo claro que tirei como referência a Aula 23 do Curso de C++ do Professor (SANTIAGO, 2022), onde ele fala especificamente da biblioteca **fstream** para manipulação de arquivos de texto.

3 Descrição da SparseMatrix

Nessa parte será estruturalmente descrito a SparseMatrix, além da descrição de cada médoto que ela contém. É importante ressaltar que tentei ao máximo seguir as instruções e regras que foram repassadas no arquivo "Projeto-Matrizes-Esparsas.pdf". Além disso, por escolha de projeto minha, **todos os comentários feitos nos códigos estão em português enquanto os nomes das variáveis, métodos, structs e funções estão escritos em inglês.**

3.1 Node

Antes de falar da SparseMatrix, é necessário falar sobre a estrutura **Node**, afinal, a SparseMatrix é dependente dela. Fiz a implementação dessa struct no próprio arquivo SparseMatrix.h. O Node é **estruturalmente igual** ao Node que foi passado como exemplo no "Projeto-Matrizes-Esparsas.pdf", ele contém dois ponteiros, um para um Node a **direita(right)** e outro para um Node **abaixo(down)**, além do **índice de linha**, **índice de coluna** e o **valor contido no nó**. É de conhecimento que seria possível a implementação com **templates**, porém decidi seguir a rigor a struct que foi passada nas instruções do PDF, então por padrão Node guarda um valor do tipo **double**.

3.2 Atributos Privados da SparseMatrix

- **head**;
  Head é um ponteiro para o nó de índice (0,0), esse nó é de grande importância para estrutura, afinal é ele que da o ponto de partida para chegar a qualquer lugar da matriz
- **row_s**;
  Pode ser interpretado como rows_size, refere-se a quantidade de linhas da matriz
- **col_s**;
  Pode ser interpretado como column_size, refere-se a quantidade de coluna da matriz

3.3 Construtor

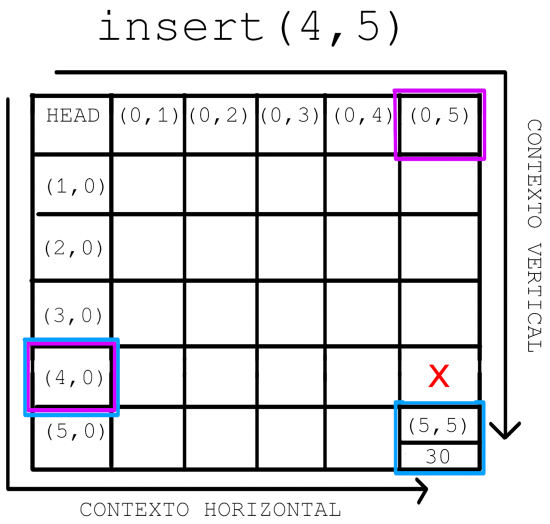
O construtor da classe é relativamente simples. De forma resumida, sé forem passados valores nulos, negativos ou maiores que 30000 na criação da matriz, o construtor lança um erro do tipo invalid_argument. Se os valores forem validos, head(o nó principal) é criado, assim como dois loops fazem o papel de alocar todos os sentinelas da linha 0 e coluna 0, além de organizar os Nodes down e right para manter a circularidade. **Veja o teste de estrutura do construtor na seção 5.1.**

3.4 Insert

Vou tentar explica esse método de forma resumida, pois de longe ele é um dos mais complexos da classe. Basicamente, a ideia principal do método insert é trabalhar com dois contextos, sendo eles, o **vertical** e o **horizontal** (Basicamente como se fossem duas listas separadas). Para ambos os contextos meu método faz a busca de dois Nodes, sendo um o **back** (Node que deve vir antes do valor a ser inserido) e o outro o **front** (Node que deve vir depois). O primeiro contexto a ser analisado é o vertical, nele é feita a possível alocação do novo Node. Eu me refiro como possível, pois esse método tem casos especiais. Um exemplo é o caso em que o método insert quer colocar um zero aonde não existe valor. Nesse caso, o método não altera nenhum node e não aloca memória nenhuma. Outro exemplo de caso especial é quando é necessário inserir zero em um local que **já existe um nó**. Nesse em específico, meu método remove as referências ao valor existente e o deleta.

A ilustração abaixo de um pequeno exemplo de como o método inserte funciona para um caso de alocação. Perceba que meu nó back(horizontal/vertical) **está na cor roxa** e meu nó front(horizontal/vertical) **está na cor azul**, assim como o local de insert está na **cor vermelha**.

Figura 2: Exemplificação do insert em uma matriz 5x5



Fonte: Autoria própria

Veja o teste de validação da função insert na seção **5.2**

3.5 Destrutor

Esse método tem o papel de desalocar todos os Nodes que foram alocados, incluindo os próprios sentinelas. Esse método tem o funcionamento relativamente simples. O que acontece é que o método passa por todas as linhas **deletando horizontalmente todos os Nodes**. Quando resta apenas um Node (que no caso é o primeiro da linha), esse método guarda a referência desse Node, pula para próxima linha, e remove essa referência. Esse **ciclo** se repete até chegar a o primeiro Node da última linha. Quando o método chega nesse ponto, ele deleta esse Node e termina.

O que foi feito para testar esse método foi criar um contador e incrementar a cada remoção efetuada pele método. No final, era feito a comparação do total de remoções e o total de Nodes (contando com os sentinelas). Em todos os testes os **valores finais foram os esperados**, o que comprovou a remoção completa dos Nodes.

3.6 Get

Basicamente esse método "anda"até o sentinela de coluna referente a coordenada que se quer pegar o valor. Assim que chega, ela "anda"até achar um elemento que **tenha um índice de linha maior ou igual** ao passado como parâmetro, ou até um elemento que aponta para um sentinela(que no caso é o último/único elemento da coluna). Após isso, um if analisa sé ela consegui o índice pedido, sé sim, ela retorna o valor do Node onde ela parou, se não, ela retorna 0.

3.7 Print

Provavelmente o método mais simples. Ele intera a matriz toda utilizando o método get e da um cout nos valores.

4 Descrição da main (interativa)

Essa seção é dedicada a descrição da main (arquivo principal).

4.1 Comandos

- **create [quantidade_linhas] [quantidade_colunas]**
  Cria uma matriz esparsa com os valores passados e guarda ela em um vetor de matrizes esparsas. Importante ressaltar que esse comando confirma as dimensões passadas e informa o **índice** atribuído matriz no vetor.
- **create_by_sum [índice_da_matriz1] [índice_da_matriz2]**
  Cria uma nova matriz baseada na soma de duas outras matrizes e guarda ela em um vetor de matrizes esparsas.
- **create_by_mult [índice_da_matriz1] [índice_da_matriz2]**
  Cria uma nova matriz baseada na multiplicação de duas outras matrizes e guarda ela em um vetor de matrizes esparsas.
- **create_by_read [nome_do_arquivo]**
  Cria uma nova matriz baseada em um arquivo de texto e guarda ela em um vetor de matrizes esparsas.

- **delete [índice_da_matriz]**
  Remove uma matriz do vetor de matrizes esparsas.
- **insert [índice_da_matriz] [linha] [coluna] [valor]**
  Insere um valor na linha e coluna de uma matriz específica.
- **print [índice_da_matriz]**
  Printa uma matriz específica.
- **printAll**
  Printa todas as matrizes.
- **sum [índice_da_matriz1] [índice_da_matriz2]**
  Printa a soma de duas matrizes.
- **mult [índice_da_matriz1] [índice_da_matriz2]**
  Printa a multiplicação de duas matrizes.
- **exit**
  Encerra o programa.

4.2 Funções

4.2.1 readSparseMatrix

Essa função utiliza a biblioteca fstream para ler um arquivo de texto. Os dois primeiros valores do arquivo de texto são referentes ao tamanho da matriz. Das dimensões em diante, todos os valora são organizados na forma linha coluna valor. O que a função faz é alocar uma matriz esparsa com as dimensões passadas e em seguida inserir todos os valores passados. Ao final ela retorna o endereço da matriz esparsa alocada com os valores inseridos. Veja sobre o teste dessa função na seção **5.3**

4.2.2 sum & multiply

Para a criação dessas funções, minha principal referência foi as funções "soma"e "multiplica"implementadas na atividade de [TAD] Matriz. Soma e multiplicação de matrizes em códigos são problemas clássicos. Basicamente a única adaptação necessárias foi utilizar as funções get e insert para pegar e inserir valores nas células. Além da utilização de alocação dinâmica, afinal, a função tem como retorno o endereço para uma matriz de resultado.

5 Listagem de Testes

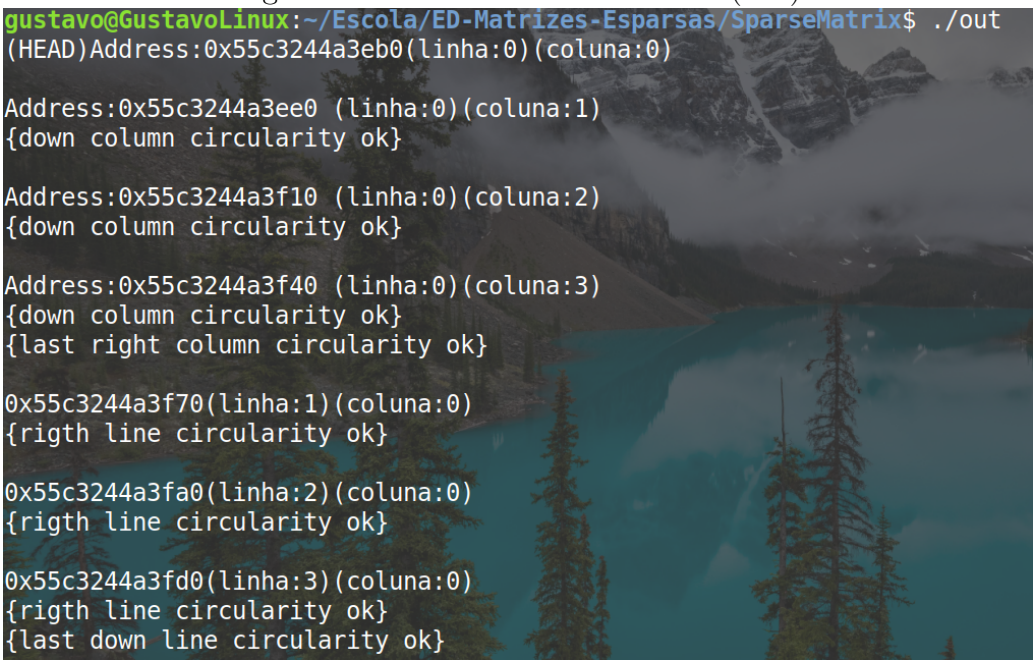
Essa seção é dedicada a listagem de testes efetuados durante a criação da estrutura de dados e seu script principal. É importante ressaltar que deixei uma pasta dedicada a esses scripts, para caso de averiguação.

5.1 Teste de estrutura do construtor

Nesse teste foi feito um script que passa por todos os sentinelas da linha 0, assim como todos os sentinelas da coluna 0. Para todos os sentinelas é observado se ele aponta para si próprio e se o k_n último sentinela aponta para head.

Foi observado, que tanto para matrizes pequenas (exemplo uma 1x1), quanto para matrizes maiores os testes funcionaram perfeitamente.

Figura 3: Teste de estrutura matriz (3x3)

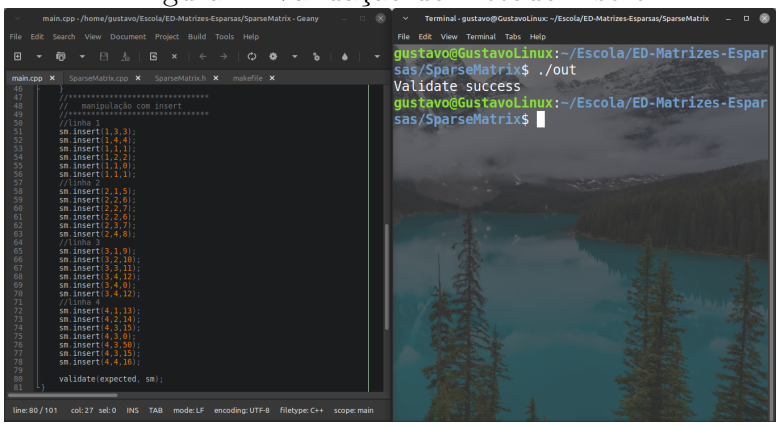


Fonte: Autoria própria

5.2 Teste de insert

Esse teste é um teste de comparação. Basicamente um matriz estática é feita com números específicos. Em seguida, um matriz esparsa de mesmo tamanho da estática é criada. O script começa a manipular a matriz esparsa tentando quebrar de alguma maneira o método insert. A matriz esparsa é manipulada de diversas formas até ficar igual em valores à matriz estática. A função validate recebe as duas matrizes e testa todos os índices para ver se elas são iguais. Além disso a função é dependente que a circularização da matriz esparsa esteja perfeita, ou então o validate irá falhar. Nó final o script de valida ou não se as matizes são iguais. Observe que após diversas manipulações método insert não falha e as matrizes são iguais, assim obtendo a validação da função.

Figura 4: Validação do método insert



Fonte: Autoria própria

5.3 Teste da função readSparseMatrix

Nesse teste foi feito uma main que ler uma quantidade n de arquivos de texto. Para todos os arquivos essa main utiliza a função readSparseMatrix, printa o valor da matriz recentemente lida e no final deleta ela.

Ao final das contas, esse teste acabou sendo efetivo não só para testar a função readSparseMarix, mas como também para testar as **funções a qual ela depende**. Foi possível analisar nesse teste, que tanto a função insert, como a função print, estavam adequadamente funcionado.

6 Analise de pior caso

6.1 insert

Quando estamos fazendo uma análise de pior caso, sempre damos maior atenção para os **laços de repetição**, afinal, a maioria deles tem maior influência do que uma **linha de comando simples**. No caso da função insert, temos majoritariamente 4 laços. Dois que andam no contexto vertical e outros dois que andam no contexto horizontal, sendo que desses quatro, dois devem andar por todos os sentinelas, um nos verticais e outro nos horizontais. Pensando nesse dois laços citados, o pior dos casos é ter que anda para r_k última linha e c_k última coluna da matriz, ou seja, **andar até o canto inferior direito**. Pensando nisso, agora vamos para os dois laços que faltaram, esse em questão são **dependentes** dos valores alocados na estrutura de dados. Tendo isso em mente, uma matriz de r_k linhas e c_k colunas sem elementos alocados, independente da posição, não rodara se quer uma única vez esse laço. Porém, uma matriz com **todas** todas as linhas e colunas com valores alocados, rodaram esses laços referente ao **módulo da distância** do sentinela até a célula desejada. Somando isso ao que já foi citado, é visível que o **pior caso** para essa função é ter que inserir um valor na r_k última linha, c_k última coluna tendo todas as células devidamente alocadas.

6.2 get

Como já citado na análise de pior caso anterior, é de grande importância a análise dos laços de repetição do método. Tendo isso em mente, na questão de pior caso, o método **get** se assemelha muito ao insert, porém ele só possui dois laços. Um dos laços anda até o sentinela de coluna (contexto vertical) desejado, enquanto o outro **tenta** chegar até o valor pedido. Do mesmo modo da função insert, esse último laço citado, quando uma matriz não possui nenhum valor não roda se quer uma única vez. Porém, se a matriz estivar cheia, então esse laço rodará o número de vezes referente ao **módulo da distância** do sentinela até a célula desejada. Logo, o pior caso desse método é o mesmo do método insert, ou seja, ter que ler um valor na r_k última linha, c_k última coluna tendo todas as células devidamente alocadas.

6.3 sum

A função sum, em seu "core"utiliza outros dois métodos da classe **SparseMatrix** no seu funcionamento, sendo eles os métodos get e insert. Como já citado anteriormente, esse dois métodos possuem a mesma conclusão de pior caso. Então é fácil de perceber que função sum **"herdará" parcialmente** o pior caso desses métodos, afinal, os dois laços presente na função sum rodam sempre $r_n \times c_n$ vezes. Logo, o pior cenário de soma é se foram passadas

duas matrizes **completamente preenchidas**, qualquer outro caso diferente desse, tendo como referência uma matriz de mesmo tamanho, rodara de forma mais rápida.

Referências

LE MENESTREL, Thomas. Sparse Matrices: Why They Matter for Machine Learning and Data Science. en. **Towards Data Science**, v. 1, n. 1, p. 1, 2022.

SANTIAGO, Judson. Programação de Computadores. pt. **Curso Completo de Programação C++**, v. 0, n. 0, p. 1, 2022.