Projeto de Estrutura de Dados

matriz esparsa

Feito por: Graziele Ferreira Barbosa
matrícula: 554184

Motivação

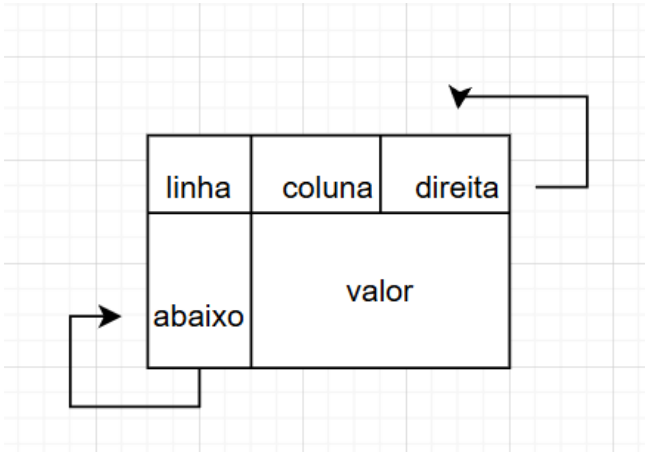
Projeto para compor a terceira nota da disciplina de Estrutura de Dados. O objetivo é implementar uma estrutura usando lista simplesmente encadeada circular para representar uma matriz esparsa.

Estrutura do Node

Cada célula da matriz, que guarda um valor diferente de zero, é representada por um Struct Node que tem os seguintes campos:

- Node* direita; // ponteiro para o próximo node (nó) ao lado
- Node* abaixo; //ponteiro para o próximo node (nó) abaixo
- int linha; // linha da matriz
- int coluna; //coluna da matriz
- double valor; // valor guardado na célula

No Struct Node há um construtor que “constrói” um nó no seguinte formato:



Se o construtor recebe direita e abaixo igual a “nullptr”, os campos direita e abaixo apontam para o próprio nó. Porém se recebe um nó, os campos especificados apontam para o nó passado no parâmetro, obedecendo uma lista simplesmente encadeada circular.

Estrutura da Sparse_matriz

A estrutura da matriz esparsa obedece o seguinte formato: a linha 0 e coluna 0, devem ser apenas de nós sentinelas. Quando um nó é criado, ele pertence a lista da coluna e a lista da linha. Para criar a matriz, utilizei um ponteiro m_head para ser a célula (0,0) da matriz e a partir dela criar os outros nós sentinelas de linha e

coluna. Também criei um inteiro linha e um inteiro coluna para auxiliar as operações na matriz.

O TAD da sparse_matriz possui as seguintes funções:

- construtor( );
- void apagar_linha(Node* *sentinela_linha);
- void inserir(int linha, int coluna, int valor);
- double get(int linha, int coluna);
- void print( );
- void redimensiona(int linha, int coluna)
- int get_linha( );
- int get_coluna;
- destrutor;

O construtor:

O construtor faz algo se somente os valores de linha e coluna forem maiores que zero. Se for, criei dois laços, um que cria os nós sentinelas da coluna e outro que cria os nós sentinelas da linha. Cada nó da coluna, o campo ‘abaixo’ aponta pra ele mesmo (enquanto vazio) e o campo ‘direita’ aponta para o próximo nó ao lado. Nos nós sentinelas da linha segue o mesmo sentido. O último nó, tanto linha quanto coluna, possui um ponteiro que aponta para o m_head.

O inserir:

A função inserir recebe como parâmetro um valor e a linha e coluna que esse valor deve ser inserido ou modificado (no caso que houver nó). Essa função só deve fazer algo se linha e coluna passados como parâmetro obedecem o intervalo já definido e se o valor passado for diferente de 0.

Usei como norte os seguintes casos: quando a coluna não tem nó e quando ela tem. Para me auxiliar, criei quatro ponteiros: um que aponta para o sentinela da coluna que vai estar o nó, um para ser o próximo (no caso o que vai caminhar a partir do campo ‘abaixo’), um que aponta para o sentinela da linha que vai estar o nó e outro que vai ser o próximo (a partir do campo ‘direita’).

Para o primeiro caso, o caso em que a coluna está vazia, criei um novo nó e ajustei os ponteiros da coluna. Já para conectar ele na linha criei um laço e dentro dele as três condições possíveis para ele ser conectado à linha (se tiver no meio, se for o primeiro e se for o último). Nos três casos, ajustar os ponteiros e saí do laço.

Para o segundo caso, o caso em que há elemento na coluna, procuro onde inserir o novo nó ou apenas atualizar o valor se já existir. Utilizei a mesma lógica anterior, em que através de um laço olho os casos possíveis para inserir ou atualizar, os casos são: se o elemento existir, se for no meio, se for o primeiro e se for o último. Em todos os casos eu ajustei os ponteiros e saio do laço após operar. Logo em seguida entro no próximo laço para ligar o novo nó na linha e sigo a mesma ideia anterior.

O get:

A função get procura um nó com as coordenadas (linha e coluna) passados no parâmetro e retorna o valor que existe naquela célula se ele existir. O primeiro passo é verificar se as coordenadas estão no intervalo correto e se tiver, a função segue a procura do valor desejado.

Através de um auxiliar, encontro o sentinela da coluna especificada e a partir dele desço até a linha especificada. Se passar é porque não existe nó e retorno 0. Se houver nó retorno o valor.

O print:

Para imprimir a função percorre cada coluna imprimindo o valor do Node se tiver, se não tiver imprime 0.

O redimensiona:

Senti necessidade de criar essa função para poder me auxiliar na leitura do arquivo. Quando chamo a função que lê o arquivo na main, ela espera no parâmetro, uma matriz e o arquivo. Porém, tive a seguinte dúvida: como vou passar uma matriz sem ter lido nem saber suas dimensões? Então, antes de chamar a função ler, crio uma matriz vazia e coloco ela no parâmetro da chamada da função ler junto com o arquivo. Na função ler, chamo a função redimensiona da Sparse_matriz que deleta a matriz vazia e constrói uma nova matriz no endereço de memória onde o ponteiro this da matriz vazia está. Aprendi esse conceito chamado Placement New e o utilizei para me auxiliar.

O get_linha e get_coluna:

retornam o valor da linha e coluna da matriz. Criei essa função para utilizar na main em chamadas de função.

O destrutor e o apagar_linha:

Meu destrutor chama a função apagar_linha que recebe como parâmetro um nó sentinela da linha e apaga todos os nós alocados naquela linha. O destrutor fica responsável por enviar para o apagar_linha os nós sentinelas da linha. Ao fim do destrutor envio a linha 0 da matriz, que contém todos os nós sentinelas da coluna.

Detalhes, dificuldades e divisão

As funções auxiliares que precisei utilizar dentro da Sparse_matriz, foram descritas anteriormente e suas motivações.

Minha primeira dificuldade foi problemas com a função inserir. Fiz ela uma vez e nos testes percebi que ela não estava correta. Decidi apagar e fazer

novamente. Dessa vez, pensei em utilizar laços com as condições possíveis dentro dele e achei essa forma mais simples, tanto para explicar quanto para procurar possíveis erros. Fiquei feliz com essa conquista e por ter conseguido pensar de outras formas, além de conseguir solucionar de forma simples o problema que encontrei. Senti dificuldade ao pensar no destrutor e conversei com colegas para poder fazer ele. A princípio pensei em fazer de várias formas, como usar recursão, mas implementando achei alguns problemas. Ao final de implementar o destrutor fiquei feliz com o resultado e por conseguir fazer de uma forma simples. A próxima dificuldade foi ao ler arquivos e principalmente por ela receber uma matriz no parâmetro. Pedi auxílio ao chatgpt e ele me falou sobre o placement new, pesquisei sobre ele e decidi usar por achar que fez sentido no contexto e que resolveria meu problema. O último problema foi quando utilizei um vector para poder guardar as matrizes e poder acessar cada matriz pelo índice do vector. Implementei na main com esse formato, usando funções prontas da biblioteca do vector. Mas, por algum motivo, não estava inserindo corretamente os nós, mesmo testando a função separadamente e tendo certeza que ela estava funcionando. Decidi então, não utilizar vector e para cada operação dentre as disponíveis, você (Atílio), deve colocar o nome do arquivo quando chamar uma operação. Achei essa solução boa e ainda sim é possível realizar todas as operações possíveis e testar.

O trabalho foi feito por mim, com paciência, dedicação e calma, utilizando o que aprendi em sala de aula.

Main

A main possui 3 funções, ler, somar e multiplicar, além de um laço interativo onde é possível colocar os arquivos que contém matrizes e chamar as funções que realizam as operações.

comandos disponíveis:

1. inserir (após inserir imprime a matriz inserida)
2. pegar (devolve o valor da célula de linha e coluna passado)
3. somar (soma duas matrizes)
4. multiplicar (multiplica duas matrizes)
5. imprimir (imprime a matriz no terminal)
6. end (encerra o programa)

Para inserir é esperado uma entrada do seguinte formato:

inserir m1.txt

(m1.txt é o arquivo que o professor disponibilizou para testes)

Para pegar é esperado uma entrada do seguinte formato:

pegar m1.txt 1 1

(arquivo e linha e coluna especificado)

Para somar é esperado na entrada o seguinte formato:

somar m1.txt m2.txt

(nome da operação + nome dos arquivos)

Para multiplicar é o mesmo formato e imprimir também.

Para poder compilar os arquivos é necessário colocá-los junto do arquivo do código fonte onde tem a main.

Testes

Os testes utilizados foram feitos com os arquivos de teste que o professor disponibilizou

teste inserir e somar

```
inserir m1.txt
50      0       0       0
10      0       20      0
0       0       77      0
-30     0       -60     -5
```

```
inserir m2.txt
0       5       0       3
7       0       8       0
4       0       7       0
-1      0       -6      -5
```

resultado da soma

```
somar m1.txt m2.txt

resultado:

50      5       0       3
17      0       28      0
4       0       84      0
-31     0       -66     -10
```

arquivo m1.txt e m2.txt

teste multiplicar:

```
multiplicar m1.txt m2.txt

resultado:

0       250     0       150
80      50      140     30
308     0       539     0
-235    -150    -390    -65
```

arquivo m1.txt e m2.txt

teste pegar

```
pegar m1.txt 1 1

resultado: 50
```

(1,1) da matriz do arquivo m1.txt

Realizei os testes com os outros arquivos e funcionou.

Análise de pior caso

insert: O pior caso do insert é quando a inserção é na última célula do canto inferior direito, onde tem que chegar até o último nó sentinela da linha e coluna e chegar até o último espaço.
get: o mesmo ocorre com o get.
soma: o pior caso acontece quando as duas matrizes possuem elementos em posições diferentes, fazendo com que a matriz resultante tenha muitos nós alocados.

No arquivo enviado, terá a main.cpp. o Node.h, e o Sparse_matrix.h, além dos arquivos disponibilizados pelo professor.