Trabalho 1 – Encadeamentos

Integrantes:
- Edryck Freitas Nascimento – a2727617
- Samuel Oliveira – a2727820

Objetivo:

Desenvolver funções com base nos conceitos de listas e encadeamento para simular e manipular uma matriz padrão bidimensional através de encadeamento e seus ponteiros.

Resolução:

Definimos duas estruturas de dados principais no arquivo matriz.h para fazer a alocação de uma matriz bidimensional de inteiros com o uso de listas duplamente encadeadas em quatro direções.

- **struct elemento**: Esta estrutura representa um nó (célula) da matriz. Ela contém:
  - **int valor**: O dado inteiro armazenado naquela posição, por padrão inicia como 0.
  - **struct elemento* d**ir: Ponteiro para o nó vizinho à direita.
  - **struct elemento* esq**: Ponteiro para o nó vizinho à esquerda.
  - **struct elemento* cima**: Ponteiro para o nó vizinho acima.
  - **struct elemento* baixo**: Ponteiro para o nó vizinho abaixo.

```
struct elemento {
    int valor;
    struct elemento* dir;
    struct elemento* esq;
    struct elemento* cima;
    struct elemento* baixo;
};
```

- **struct matriz:** Esta é a estrutura que representa a matriz como um todo. Ela contém:
  - **struct elemento* inicio**: Um ponteiro para o primeiro nó da matriz, correspondente à coordenada (0, 0).
  - **int linhas**: Armazena o número total de linhas da matriz.
  - **int colunas**: Armazena o número total de colunas da matriz.

```
struct matriz {
    struct elemento* inicio;
    int linhas;
    int colunas;
};
```

Utilizamos essas estruturas para simular uma matriz convencional, onde cada elemento acessa seus vizinhos através desses ponteiros.

Funções criadas:

Foram criadas um total de 13 funções, tendo as seguintes funções como principais para a manipulação e criação da matriz:

- **cria_matriz**: Foi a mais complexa de se fazer, devido as ligações dos ponteiros.

  Protótipo: Matriz* cria_matriz(int linha, int coluna);

  Como funciona: A função aloca dinamicamente toda a estrutura da matriz, criando cada nó e estabelecendo as quatro conexões (cima, baixo, esquerda, direita) para cada um deles.
  Inicialmente, ela verifica se os valores informados para linha e coluna são menores ou iguais a zero, se forem, retorna *NULL*, pois uma matriz com dimensões inválidas não pode ser criada.
  Em seguida, aloca memória para a estrutura principal Matriz (o ponteiro m), se a alocação falhar, retorna *NULL*.
  O ponteiro m->início é definido como *NULL* (pois nenhum nó foi criado ainda) e os campos m->linhas e m->colunas são preenchidos com os valores recebidos.
  A função utiliza dois *for*: um externo para percorrer as linhas (i) e um interno para percorrer as colunas (j).
    - **linha_ant_ini**: Um ponteiro que aponta para o primeiro nó (coluna 0) da linha anterior. É usado para criar as ligações verticais.
    - **esq**: Um ponteiro que aponta para o nó criado anteriormente na mesma linha (o vizinho à esquerda).
    - **cima**: Um ponteiro que aponta para o nó vizinho na linha de cima. Ele é inicializado com linha_ant_ini e depois avança para a direita junto com o loop de colunas.
    - **linha_atual_ini**: Um ponteiro que guarda o endereço do primeiro nó (coluna 0) da linha que está sendo criada.

  Dentro do loop interno, um novo Elemento (no) é alocado.
  Se a alocação do nó falhar, a função chama libera_matriz(m) para desalocar tudo que foi criado até o momento e retorna *NULL* para evitar vazamento de memória.
  O no->valor é inicializado com 0, e seus ponteiros dir e baixo com *NULL*.
  A parte principal da função é a parte de ligar os ponteiros. Para cada no criado:
    - no->esq aponta para esq para a ligação horizontal. Se esq não for nulo, esq->dir é atualizado para apontar de volta para nó.
    - no->cima aponta para cima, para a ligação vertical. Se cima não for nulo, cima->baixo é atualizado para apontar de volta para nó.
    - esq é atualizado para no (o nó atual se torna o "esquerdo" da próxima iteração).
    - cima avança para cima->dir, preparando-se para a próxima coluna.

- Se for o primeiro nó (0,0), m->inicio é definido como no.
- Se for o primeiro nó da linha (j == 0), linha_atual_ini é definido como no.

No final do *for* de dentro, o ponteiro linha_ant_ini é atualizado para linha_atual_ini, preparando para a próxima linha ser ligada à que acabou de ser criada.

- **libera_matriz:** Libera toda a memória alocada para a matriz, incluindo a estrutura principal e todos os nós.

  Protótipo: void libera_matriz(Matriz *m);

  Como funciona: A função primeiro verifica se o ponteiro da matriz m é nulo; se for, retorna imediatamente. Caso contrário, ela inicia um ponteiro linha no m->inicio. A função entra em um loop while que percorre todas as linhas (enquanto linha não for NULL). Dentro desse loop, um ponteiro próximo guarda a referência para a linha de baixo (linha->baixo), pois a linha atual será liberada.
  Em seguida, um ponteiro auxiliar aux é iniciado no começo da linha atual e um loop while interno percorre todas as colunas dessa linha. Em cada iteração do loop interno, um ponteiro no armazena o aux atual, aux avança para a direita (aux->dir), e a memória de no é liberada usando free(no).
  Quando o loop interno termina, todos os nós da linha foram liberados. O ponteiro linha é então atualizado para próxima (a linha de baixo). Ao final do loop externo, quando todas as linhas forem liberadas, a função libera a própria estrutura da matriz com free(m).

- **insere_matriz:** Insere ou atualiza um valor em uma posição específica (x, y) da matriz.

  Protótipo: int insere_matriz(Matriz* m, int valor, int x, int y);

  Como funciona: A função retorna 0 (falha) se a matriz m ou seu ponteiro inicio forem nulos. Ela também valida os índices x (linha) e y (coluna), retornando 0 se estiverem fora dos limites da matriz.
  Se as verificações passarem, um ponteiro no é iniciado em m->inicio. A função então navega até a posição correta: primeiro, um loop for move o ponteiro no para baixo x vezes (no = no->baixo); em seguida, um segundo loop for move o ponteiro no para a direita y vezes (no = no->dir). Ao final dos loops, no aponta para o elemento exato na coordenada (x, y). O valor desse nó (no->valor) é atualizado com o valor passado como parâmetro, e a função retorna 1 (sucesso).

- **remove_especifico:** Remove um valor de uma posição (x, y), substituindo por 0 (zero), que é o valor padrão. A função não desaloca o nó.

  Protótipo: int remove_especifico(Matriz* m, int x, int y);

  Como funciona: A função realiza as mesmas verificações iniciais de insere_matriz: retorna 0 (falha) se m ou m->inicio forem nulos, ou se os índices x e y estiverem fora dos limites.
  Ela então usa um ponteiro p, iniciado em m->inicio, para navegar até a coordenada (x, y) usando dois loops for (um para descer x vezes, outro para avançar y vezes).
  Se a qualquer momento p se tornar nulo durante a navegação, a função retorna 0. Ao encontrar o nó correto, seu valor (p->valor) é definido como 0. A função então retorna 1 (sucesso).

- **imprime_matriz:** Exibe todos os valores da matriz no console, formatados visualmente como uma matriz.

  Protótipo: int imprime_matriz(Matriz* m);

  Como funciona: Retorna 0 (falha) se m ou m->inicio forem nulos. A função usa dois loops for, um dentro do outro, para percorrer a estrutura. O loop externo, controlado por um ponteiro linha (que começa em m->inicio e se move com linha = linha->baixo), itera pelas linhas.
  O loop interno, controlado por um ponteiro col (que começa em linha e se move com col = col->dir), itera pelas colunas de cada linha. Dentro do loop interno, o valor col->valor é impresso. É usado um operador ternário para imprimir um espaço " " após o número, a menos que seja o último elemento da coluna.
  Após o término do loop interno (fim de uma linha), um caractere de nova linha (\n) é impresso. A função retorna 1 (sucesso) após a impressão completa.

- **matriz_vazia:** Verifica se todos os elementos da matriz possuem o valor 0.

  Protótipo: int matriz_vazia(const Matriz* m);

  Como funciona: Se m ou m->inicio forem nulos, a função considera a matriz como vazia e retorna 1 (verdadeiro), caso contrário, ela percorre todos os nós usando dois loops for, um dentro do outro, (com ponteiros linha e col).
  Em cada nó, ela verifica se col->valor != 0.
  Se qualquer elemento tiver um valor diferente de 0, a função retorna 0 (falso) imediatamente.

Se os loops terminarem sem encontrar nenhum valor diferente de 0, significa que a matriz está vazia, e a função retorna 1 (verdadeiro).

- **matriz_cheia:** Verifica se todos os elementos da matriz possuem um valor diferente de 0.
  Protótipo: int matriz_cheia(const Matriz* m);
  Como funciona: É a lógica oposta da matriz_vazia. Se m ou m->inicio forem nulos, retorna 0 (falso).
  A função percorre todos os nós com dois loops for, um dentro do outro. Em cada nó, ela verifica se col->valor == 0. Se qualquer elemento for igual a 0, a função retorna 0 (falso) imediatamente. Se os loops terminarem sem encontrar nenhum 0, significa que a matriz está cheia, e a função retorna 1 (verdadeiro).

- **consulta_valor:** Permite ao usuário consultar o valor armazenado em uma coordenada (x, y) específica.
  Protótipo: int consulta_valor(Matriz* m, int x, int y, int* valor_out);
  Como funciona: A função retorna 0 (falha) se m, m->inicio ou o ponteiro de saída valor_out forem nulos, ou se os índices x e y estiverem fora dos limites. Ela usa a mesma lógica de navegação de insere_matriz para encontrar o nó (x, y) usando um ponteiro p. Se p se tornar nulo durante a navegação, retorna 0. Ao encontrar o nó, o valor p->valor é copiado para o local apontado por valor_out (usando *valor_out = p->valor). A função retorna 1 (sucesso).

- **tamanho_total:** Retorna o número total de elementos que a matriz pode armazenar (capacidade total).
  Protótipo: int tamanho_total(const Matriz* m);
  Como funciona: Esta função não percorre a lista. Ela apenas verifica se m é nulo (retornando 0 nesse caso) e, se não for, retorna o produto de m->linhas * m->colunas, valores que estão armazenados na própria estrutura Matriz.

- **tamanho_usado:** Conta quantos elementos da matriz são diferentes de 0.
  Protótipo: int tamanho_usado(Matriz* m);
  Como funciona: Retorna 0 se m ou m->inicio forem nulos.
  Inicializa um contador count em 0. Em seguida, percorre toda a matriz com dois loops for, um dentro do outro (ponteiros linha e col).
  Para cada nó visitado, verifica se col->valor != 0, se for diferente de 0, o

contador count é incrementado.
No final dos loops, a função retorna o valor total de count.

- **imprime_vizinhos:** Mostra na tela os valores dos quatro vizinhos (cima, baixo, esquerda, direita) de um nó na posição (x, y).
  Protótipo: void imprime_vizinhos(Matriz* m, int x, int y);
  Como funciona: A função retorna se m for nulo ou se x ou y forem negativos.
  Ela utiliza a função consulta_no(m, x, y) para obter um ponteiro direto para o elemento (x, y), se esse ponteiro for NULL (posição inválida), a função retorna, caso contrário, ela verifica cada um dos quatro ponteiros do nó (cima, esq, dir, baixo).
  Para cada ponteiro, se ele for NULL, imprime uma mensagem indicando que é uma borda, se não for NULL, imprime o valor do vizinho (ex: no->cima->valor) e sua coordenada relativa.

- **busca_valor:** Procura a primeira ocorrência de um valor específico na matriz.
  Protótipo: Elemento* busca_valor(Matriz* m, int valor);
  Como funciona: Retorna NULL se m for nulo.
  A função percorre a matriz com dois loops for, um dentro do outro (ponteiros linha e col).
  Em cada nó, compara col->valor com o valor buscado, se forem iguais, a função retorna o ponteiro col imediatamente, se os loops terminarem sem encontrar o valor, a função retorna NULL.

- **consulta_no:** Retorna um ponteiro direto para o Elemento na posição (x, y), é uma função auxiliar para facilitar a implementação de outras funções.
  Protótipo: Elemento* consulta_no(Matriz* m, int x, int y);
  Como funciona: Retorna NULL se m for nulo ou se os índices x e y estiverem fora dos limites da matriz.
  Inicia um ponteiro no em m->inicio, usa um loop for para mover no para baixo x vezes e um segundo loop for para movê-lo para a direita y vezes, se em qualquer ponto da navegação no se tornar NULL (o que não deveria acontecer em uma matriz bem formada dentro dos limites), a função retorna NULL.
  No final, retorna o ponteiro no apontando para o elemento (x, y).

Principais desafios: No decorrer da criação do trabalho tivemos como principal desafio a parte de ligar os ponteiros na criação da matriz, algo

relativamente fácil após quebrar muito a cabeça. Nas outras funções basicamente foi todas baseadas na insere_matriz, tendo que mudar uma coisa ou outra.
No início também foi muito discutido sobre como seria a estrutura da matriz, no final foi decidido usar essa que foi apresentada.