Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Santa Helena
Bacharelado em Ciência da Computação

Disciplina:	ALGORITMOS E ESTRUTURA DE DADOS III		
Aluno:	Marciano Kuhn	**R.A.:**	2677245
Docente:	Thiago Naves		

ATIVIDADE:Trabalho Prático

Introdução

Este trabalho implementa uma matriz bidimensional dinâmica utilizando listas encadeadas, onde cada elemento pode acessar seus vizinhos esquerda, direita, cima e baixo por meio de ponteiros. Essa abordagem permite a manipulação da matriz de forma eficiente, sem necessidade de alocação estática.A matriz é representada por nós encadeados, onde cada nó contém:**um valor inteiro**,ponteiros para **esquerda**,**direita**,**cima** e **baixo**.

Arquivos criados

O código foi organizado em 3 arquivos de cabeçalho que separam diferentes funcionalidades do código.

- **Matriz.h**: Contém os protótipos das funções e as estruturas utilizadas para a definição dos dados da matriz e dos elementos.

- **Matriz.c**: Contém todas as funções responsáveis pelas operações do código.

- **Main.c**:É o ponto de entrada do código,onde todas as funcionalidades são integradas e executadas.

As funções implementadas

- **Criar matriz:**

  Cria dinamicamente uma matriz de elementos onde cada elemento guarda um valor, tem ponteiros que o ligam aos seus **vizinhos**: cima, baixo, esquerda, e direita.

  Primeiramente é alocado memória para a matriz,e inicializados seus ponteiros.

  Depois é alocado memória de um vetor de ponteiros para as linhas,e alocada cada linha(coluna) com um loop for.

  Configura as posições dos elementos para receber 0 e faz a ligação entre os elementos, e por fim define o inicio da matriz com 0,0.

●**Insere um valor:**

A função recebe como parâmetro um ponteiro para a estrutura da matriz, a posição da linha e coluna onde o valor será inserido, e o valor inteiro que será armazenado naquela posição.Primeiro verifica se existe a matriz,depois aloca memória para o novo elemento,percorre a matriz até a posição desejada e atribui o valor desejado.

●**Consulta valor pela posição:**

A função recebe como parâmetro um ponteiro para a estrutura da matriz, a posição da linha e da coluna.Primeiro verifica se existe a matriz, depois percorre a matriz até achar a posição desejada, se a posição não existir retorna posição inválida,se existir imprime o valor.

●**Consulta posição pelo valor:**

A função recebe como parâmetro um ponteiro para a estrutura da matriz, o valor a ser recebido,um ponteiro para guardar a quantidade se houver mais de um valor,e um vetor para as posições.Primeiro verifica se existe a matriz, depois percorre a matriz até achar o valor desejado, e armazena a posição no vetor e incrementa a quantidade,retorna 1 se encontrou e 0 se não.

●**Imprime os 4 vizinhos:**

A função recebe como parâmetro um ponteiro para a estrutura da matriz, a posição da linha e da coluna.Primeiro verifica se existe a matriz, depois percorre a matriz até achar a posição desejada,se a posição não existir retorna. Se existir,imprime os valores apontados pelo ponteiro que está na posição.

●**Imprime matriz:**

A função recebe como parâmetro um ponteiro para a estrutura da matriz.Primeiro verifica se existe a matriz,depois percorre todas as linhas e colunas, se o valor for 0 imprime ‘M’, caso contrário imprime o valor.

●**Remove valor:**

A função recebe como parâmetro um ponteiro para a estrutura da matriz, e o valor a ser recebido.Primeiro verifica se existe a matriz, depois percorre a matriz até achar a posição desejada,comparando o valor da posição com o valor recebido,e atribuindo o valor 0 para ele,com isso removendo ele da matriz.Se não achar retorna 0.

●**Libera matriz:**

A função recebe como parâmetro um ponteiro para a estrutura da matriz. Se a matriz existir libera a memória de cada linha da matriz que são vetores de elementos,depois libera o vetor de ponteiros que estavam apontando para cada linha e por fim libera a estrutura matriz.

Desafios

As principais dificuldades foram na criação da matriz com o gerenciamento correto da memória para garantir que cada nó fosse corretamente ligado ao seu vizinho,estabelecer ligações corretas ou esquecer conexões foram um desafio.A liberação de memória também se mostrou difícil,como cada elemento é um nó encadeado a liberação correta foi um desafio.

Em todas as funções lidar com as ligações do nós foi o mais difícil,porque um nó sem uma ligação para o próximo já não funcionava corretamente a função, e o gerenciamento de memória para cada elemento.