Relatório do Projeto: Implementação de um Autômato Celular em C

1. Implementação

Descrição Geral

O projeto consistiu na implementação de um autômato celular para simular o jogo da vida de Conway utilizando matrizes esparsas representadas por listas encadeadas circulares. A implementação foi dividida em dois principais TADs: `MatrizEsparsa` e `AutomatoCelular`, cada um encapsulando a lógica para gerenciar a estrutura de dados e a evolução do sistema.

Estrutura do Projeto

- **matriz.h e matriz.c**: Define e implementa o TAD `MatrizEsparsa`, responsável por armazenar e manipular as células ativas da matriz. A matriz é representada por listas encadeadas circulares, o que otimiza o uso de memória para casos onde a matriz é esparsa.
- **automato.h e automato.c**: Define e implementa o TAD `AutomatoCelular`, que gerencia a evolução do autômato ao longo das gerações. Este TAD utiliza o `MatrizEsparsa` para armazenar o estado atual das células.
- **tp.c**: Contém a função principal que lê os dados de entrada, inicializa o autômato, executa as gerações, e imprime o resultado final.

Funções Principais

- **alocarMatriz**: Função responsável por alocar dinamicamente uma matriz esparsa.
- **inserirElemento**: Insere um elemento na matriz esparsa, adicionando-o à lista encadeada circular.

- **evoluirReticulado**: Função que aplica as regras do jogo da vida para evoluir o estado do autômato ao longo das gerações.
- **imprimeMatriz**: Imprime a matriz esparsa, representando os estados vivos e mortos das células.

Decisões de Implementação

- Uso de listas encadeadas circulares para representar a matriz esparsa, garantindo que a navegação nas linhas e colunas fosse eficiente.
- A função de evolução foi implementada de forma iterativa para garantir performance, já que a recursividade poderia implicar em overhead para um número elevado de gerações.
- O uso de alocação dinâmica foi essencial para garantir a flexibilidade da matriz esparsa, sendo possível adaptar o tamanho do reticulado de acordo com as necessidades da simulação.

2. Impressões Gerais

O processo de implementação foi bastante desafiador, especialmente na manipulação das listas encadeadas circulares, uma estrutura que requer atenção aos ponteiros para evitar erros como loops infinitos ou segmentation faults. Gostei de trabalhar com a otimização de memória usando matrizes esparsas, pois esse é um conceito muito aplicável em diversas áreas da computação.

3. Análise

Os resultados obtidos mostraram que a utilização de listas encadeadas circulares para representar matrizes esparsas é eficaz para economizar memória em simulações de autômatos celulares com grandes áreas vazias. O programa executou dentro do tempo limite para os casos de teste, demonstrando a eficiência da abordagem escolhida.

No entanto, percebi que, para matrizes muito densas (com muitas células vivas), a vantagem de memória pode diminuir, e o tempo de execução aumenta devido à necessidade de percorrer longas listas encadeadas.

4. Conclusão

Este trabalho foi uma boa oportunidade para aprofundar meu conhecimento sobre estruturas de dados dinâmicas e sua aplicação em problemas reais. As principais dificuldades encontradas foram na gestão dos ponteiros e na otimização das operações de inserção e busca na matriz esparsa. No entanto, a implementação foi concluída com sucesso, atendendo às especificações e alcançando os objetivos propostos.

5. Diretivas de Compilação

make

valgrind --leak-check=full -s ./exe < tests/.in

6. Modularização do projeto

matriz.h

```
//Interface do TAD MatrizEsparsa
//Esse TAD representa uma "matriz esparsa" "usando listas encadeadas
circulares"
#ifndef MATRIZ_H
#define MATRIZ_H

typedef struct Celula {
   int linha;
   int coluna;
   int valor;
   struct Celula* prox;
} Celula;

typedef struct {
   int numLinhas;
   int numColunas;
```

```
    Celula** linhas;
} MatrizEsparsa;

MatrizEsparsa* alocarMatriz(int linhas, int colunas);
void desalocarMatriz(MatrizEsparsa** matriz);
void inserirElemento(MatrizEsparsa* matriz, int linha, int coluna);
Celula* buscarElemento(MatrizEsparsa* matriz, int linha, int coluna);
void imprimeMatriz(const MatrizEsparsa* matriz);

#endif // MATRIZ_H
```

matriz.c

```
//Implementacao do TAD MatrizEsparsa
#include <stdio.h>
#include <stdlib.h>
#include "matriz.h"

//Aloca dinamicamente uma estrutura MatrizEsparsa
MatrizEsparsa* alocarMatriz(int linhas, int colunas) {
    MatrizEsparsa* matriz =
(MatrizEsparsa*)malloc(sizeof(MatrizEsparsa));
    matriz->numLinhas = linhas;
    matriz->numColunas = colunas;
    matriz->linhas = (Celula**)malloc(linhas * sizeof(Celula*));
    for (int i = 0; i < linhas; i++) {
        //Todos os ponteiros sao definidos como NULL, indicando que nao
ha celulas nessa linha
        matriz->linhas[i] = NULL;
    }
    return matriz;
}

//Libera a memoria alocada para a MatrizEsparsa e suas celulas
void desalocarMatriz(MatrizEsparsa** matriz) {
    if (matriz == NULL || *matriz == NULL) return;

    //Para cada linha, se ha celulas alocadas, percorre-se a lista
    //encadeada circular e libera cada celula
    for (int i = 0; i < (*matriz)->numLinhas; i++) {
        Celula* atual = (*matriz)->linhas[i];
        if (atual != NULL) {
```

```
            Celula* inicio = atual;
            do {
                Celula* temp = atual;
                atual = atual->prox;
                free(temp);
            } while (atual != inicio);
        }
    }
    free((*matriz)->linhas);
    free(*matriz);
    *matriz = NULL;
}

//Insere um elemento na matriz esparsa na posicao especificada por
linha e coluna
void inserirElemento(MatrizEsparsa* matriz, int linha, int coluna) {
    if (linha >= matriz->numLinhas || coluna >= matriz->numColunas)
return;

    Celula* nova = (Celula*)malloc(sizeof(Celula));
    nova->linha = linha;
    nova->coluna = coluna;
    nova->valor = 1;

    Celula* atual = matriz->linhas[linha];

    if (atual == NULL) { //Se a linha ainda nao possui elementos
        nova->prox = nova; //A nova celula e adicionada como o unico
elemento da lista circular
        matriz->linhas[linha] = nova;

    //insere a nova celula na posicao correta na lista circular,
garantindo que as celulas
    //estejam ordenadas pela coluna
    //Depois, a lista circular e atualizada para manter a referencia
correta entre as celulas,
    //seja ao inserir no inicio ou no meio da lista
    } else {
        Celula* anterior = NULL;
        do {
            if (atual->coluna >= coluna) break;
            anterior = atual;
            atual = atual->prox;
```

```
        } while (atual != matriz->linhas[linha]);

        nova->prox = atual;
        if (anterior != NULL) {
            anterior->prox = nova;
        } else {
            matriz->linhas[linha] = nova;
        }
    }
}

//Busca por uma celula especifica na matriz esparsa
Celula* buscarElemento(MatrizEsparsa* matriz, int linha, int coluna) {
    if (linha >= matriz->numLinhas || coluna >= matriz->numColunas)
return NULL;

    Celula* atual = matriz->linhas[linha];
    if (atual == NULL) return NULL;

    //Percorre a lista circular na linha especificada ate encontrar a
celula
    //que corresponde a coluna desejada
    do {
        if (atual->coluna == coluna) return atual;
        atual = atual->prox;
    } while (atual != matriz->linhas[linha]);

    return NULL;
}

//Imprime a matriz esparsa na forma de uma matriz densa
//A lista circular e percorrida conforme necessario para
//verificar a presenca de celulas ativas em cada coluna
void imprimeMatriz(const MatrizEsparsa* matriz) {
    for (int i = 0; i < matriz->numLinhas; i++) {
        Celula* atual = matriz->linhas[i];
        for (int j = 0; j < matriz->numColunas; j++) {
            if (atual != NULL && atual->coluna == j) {
                printf("1 ");
                atual = atual->prox;
            } else {
                printf("0 ");
            }
```

```
        }
        printf("\n");
    }
}
```

automato.h

```
//Interface do TAD AutomatoCelular
//Usando listas encadeadas para representar um automato
#ifndef AUTOMATO_H
#define AUTOMATO_H

#include "matriz.h"

typedef struct AutomatoCelular {
    MatrizEsparsa* matriz;
} AutomatoCelular;

AutomatoCelular* alocarReticulado(int linhas, int colunas);
void desalocarReticulado(AutomatoCelular** automato);
void leituraReticulado(AutomatoCelular* automato, FILE* arquivo);
void evoluirReticulado(AutomatoCelular* automato, int geracoes);
void imprimeReticulado(const AutomatoCelular* automato);

#endif // AUTOMATO_H
```

automato.c

```
//Implementacao do TAD AutomatoCelular
#include <stdio.h>
#include <stdlib.h>
#include "automato.h"
#include "matriz.h"

//Aloca e inicializa uma estrutura AutomatoCelular, que representa o
reticulado do automato celular
AutomatoCelular* alocarReticulado(int linhas, int colunas) {
```

```
    //Internamente, essa estrutura contem uma matriz esparsa, que e
alocada pela funcao alocarMatriz
    AutomatoCelular* automato =
(AutomatoCelular*)malloc(sizeof(AutomatoCelular));
    automato->matriz = alocarMatriz(linhas, colunas);
    return automato;
}

//Libera a memoria alocada para o AutomatoCelular, bem como para a
matriz esparsa associada
void desalocarReticulado(AutomatoCelular** automato) {
    if (automato == NULL || *automato == NULL) return;
    desalocarMatriz(&(*automato)->matriz);
    free(*automato);
    *automato = NULL;
}

//Le o estado inicial do automato celular a partir de um arquivo
void leituraReticulado(AutomatoCelular* automato, FILE* arquivo) {
    //Para cada posicao na matriz, o valor lido (0 ou 1) e utilizado
para decidir se deve
    //ser inserida uma celula na matriz esparsa
    for (int i = 0; i < automato->matriz->numLinhas; i++) {
        for (int j = 0; j < automato->matriz->numColunas; j++) {
            int valor;
            fscanf(arquivo, "%d", &valor);
            //Se o valor for 1, a funcao inserirElemento e chamada para
inserir a
            //celula ativa na posicao correspondente
            if (valor == 1) {
                inserirElemento(automato->matriz, i, j);
            }
        }
    }
}

//Evolucao do automato celular por um numero especificado de geracoes
void evoluirReticulado(AutomatoCelular* automato, int geracoes) {
    for (int g = 0; g < geracoes; g++) {
        //Uma nova matriz esparsa (novaMatriz) e alocada para armazenar
o estado atualizado
```

```
        MatrizEsparsa* novaMatriz =
alocarMatriz(automato->matriz->numLinhas,
automato->matriz->numColunas);


        //Para cada celula da matriz atual, e contada a quantidade de
celulas vivas em
        //suas vizinhancas (oito celulas adjacentes)
        for (int i = 0; i < automato->matriz->numLinhas; i++) {
            for (int j = 0; j < automato->matriz->numColunas; j++) {
                int vivos = 0;
                int deltas[] = {-1, 0, 1};
                for (int di = 0; di < 3; di++) {
                    for (int dj = 0; dj < 3; dj++) {
                        if (!(deltas[di] == 0 && deltas[dj] == 0)) {
                            int ni = i + deltas[di];
                            int nj = j + deltas[dj];
                            if (ni >= 0 && ni <
automato->matriz->numLinhas && nj >= 0 && nj <
automato->matriz->numColunas) {
                                if (buscarElemento(automato->matriz, ni,
nj)) {
                                    vivos++;
                                }
                            }
                        }
                    }
                }

                //As regras de atualizacao sao aplicadas
                Celula* celulaAtual = buscarElemento(automato->matriz,
i, j);
                if (celulaAtual) {
                    if (vivos == 2 || vivos == 3) {
                        inserirElemento(novaMatriz, i, j);
                    }
                } else {
                    if (vivos == 3) {
                        inserirElemento(novaMatriz, i, j);
                    }
                }
            }
        }
```

```
        //Apos processar todas as celulas, a matriz antiga e desalocada,
        //e a nova matriz se torna a matriz atual do automato
        desalocarMatriz(&automato->matriz);
        automato->matriz = novaMatriz;
    }
}

//Imprime o estado atual do automato celular
void imprimeReticulado(const AutomatoCelular* automato) {
    imprimeMatriz(automato->matriz);
}
```

tp.c

```
//Funcao principal que controla a execucao do programa
#include <stdio.h>
#include <stdlib.h>
#include "automato.h"

int main(int argc, char* argv[]) {
    int linhas, geracoes;

    //Le a primeira linha do arquivo que contem a dimensao da matriz e o
numero de geracoes
    fscanf(stdin, "%d %d", &linhas, &geracoes);

    AutomatoCelular* automato = alocarReticulado(linhas, linhas);
    leituraReticulado(automato, stdin); //Le o reticulado a partir do
stdin

    evoluirReticulado(automato, geracoes);

    imprimeReticulado(automato);

    desalocarReticulado(&automato);

    return 0;
}
```