Samara Paloma Lopes Ribeiro
Bruno Matheus Valério de Souza

Relatório: Implementação de um Autômato Celular

1. Implementação

Descrição Geral: Este projeto consiste na implementação de um autômato celular conhecido como "Jogo da Vida" de John Horton Conway. O programa lê um estado inicial de um reticulado bidimensional a partir de um arquivo, evolui o reticulado por um número especificado de gerações e imprime o estado final.

Principais Funções e Procedimentos:

- Arquivo tp.c
  - Função main:
    - Verifica o número de argumentos passados.
    - Chama a função leituraReticulado para ler o estado inicial do reticulado a partir de um arquivo.
    - Chama a função evoluirReticulado para evoluir o reticulado pelo número de gerações especificado.
    - Chama a função imprimeReticulado para imprimir o estado final do reticulado.
    - Desaloca a memória utilizada chamando desalocarReticulado.

- Arquivo automato.h
  - Estrutura AutomatoCelular:
    - Define a estrutura do autômato celular, contendo a dimensão
  - Protótipos das funções necessárias.

- Arquivo automato.c
  - Função alocarReticulado:
    - Aloca memória para a estrutura do autômato celular e suas matrizes.
  - Função desalocarReticulado:
    - Libera a memória alocada para a estrutura do autômato celular e suas matrizes.
  - Função leituraReticulado:
    - Lê as dimensões do reticulado e o número de gerações a partir de um arquivo.
    - Inicializa a matriz de células com os valores lidos do arquivo.
  - Função contarCelulasVivasVizinhas:
    - Conta o número de células vivas vizinhas de uma célula específica, considerando as células ao redor.
  - Função evoluirReticulado:
    - Evolui o reticulado por um número especificado de gerações, aplicando as regras do Jogo da Vida.

- Utiliza recursão para iterar sobre as gerações.
- Função imprimeReticulado:
- Imprime o estado atual do reticulado na tela.

2. Impressões Gerais

Processo de Implementação: Gostei de como o projeto exigiu a implementação de um TAD (Tipo Abstrato de Dados) para organizar a estrutura do autômato celular. Isso ajudou a manter o código modular e organizado.

3. Análise

Resultados Obtidos: O programa funciona corretamente para os casos de teste fornecidos, evoluindo o reticulado conforme esperado. A implementação recursiva da função evoluirReticulado foi eficiente e cumpriu o limite de tempo estabelecido. A modularização do código facilitou a depuração e a manutenção.

4. Conclusão

Comentários Gerais: A implementação deste projeto foi uma boa oportunidade para aplicar conceitos de alocação dinâmica de memória, modularização de código e recursão. As principais dificuldades encontradas foram relacionadas à manipulação correta das matrizes e ao gerenciamento da memória. No geral, o projeto foi bem-sucedido e proporcionou um bom entendimento sobre o funcionamento dos autômatos celulares.

Diretivas de Compilação

- **make**

- **valgrind --leak-check=full -s ./ex < casoteste.in ou ./ex < casoteste.in**

- **python3 corretor.py**

TP.C

```c
#include "automato.h"
#include <stdio.h>
#include <stdlib.h>

int main() {
   int numGeracoes;

   // Ler o automato celular da entrada padrao
   AutomatoCelular* automato = leituraReticulado(stdin, &numGeracoes);

   // Evolui o reticulado pelo numero de geracoes especificado
   evoluirReticulado(automato, numGeracoes);
```

```
    // Imprimi o estado atual do reticulado apos as geracoes
    imprimeReticulado(automato);

    // Desaloca memoria antes de encerrar o programa
    desalocarReticulado(automato);

    return EXIT_SUCCESS;
}
```

AUTOMATO.H

```
#ifndef AUTOMATO_H
#define AUTOMATO_H

#include <stdio.h>

// Declaracao da estrutura que representa o automato
typedef struct {
    // Representa o automato celular
    int dimensao;
    int **celulas; // matriz para armazenar o estado das celulas
    int **proximaGeracao; // matriz para armazenar o estado da proxima
geracao
} AutomatoCelular;

// Funcoes publicas do TAD AutomatoCelular
AutomatoCelular* alocarReticulado(int dimensao);
void desalocarReticulado(AutomatoCelular* automato);
AutomatoCelular* leituraReticulado(FILE* entrada, int *numGeracoes);
int contarCelulasVivasVizinhas(AutomatoCelular* automato, int linha,
int coluna);
void evoluirReticulado(AutomatoCelular* automato, int geracoes);
void imprimeReticulado(AutomatoCelular* automato);

#endif
```

AUTOMATO.C

```
#include "automato.h"
#include <stdlib.h>
#include <stdio.h>
```

```c
//Esse arquivo contera as implementacoes das funcoes que operam sobre a estrutura do automato

// Aloca um reticulado
AutomatoCelular* alocarReticulado(int dimensao) {
   AutomatoCelular* automato = (AutomatoCelular*)malloc(sizeof(AutomatoCelular));
   if (automato == NULL) {
       fprintf(stderr, "Erro ao alocar memoria para o reticulado.\n");
       exit(EXIT_FAILURE);
   }

   automato->celulas = (int**)malloc(dimensao * sizeof(int*));
   if (automato->celulas == NULL) {
       fprintf(stderr, "Erro ao alocar memoria para as celulas.\n");
       exit(EXIT_FAILURE);
   }

   for (int i = 0; i < dimensao; i++) {
       automato->celulas[i] = (int*)malloc(dimensao * sizeof(int));
       if (automato->celulas[i] == NULL) {
           fprintf(stderr, "Erro ao alocar memoria para as celulas.\n");
           exit(EXIT_FAILURE);
       }
   }

   automato->proximaGeracao = (int**)malloc(dimensao * sizeof(int*));
   if (automato->proximaGeracao == NULL) {
       fprintf(stderr, "Erro ao alocar memoria para a proxima geracao.\n");
       exit(EXIT_FAILURE);
   }

   for (int i = 0; i < dimensao; i++) {
       automato->proximaGeracao[i] = (int*)malloc(dimensao * sizeof(int));
       if (automato->proximaGeracao[i] == NULL) {
           fprintf(stderr, "Erro ao alocar memoria para a proxima geracao.\n");
           exit(EXIT_FAILURE);
       }
   }
```

```c
    automato->dimensao = dimensao;

    return automato;
}

// Desaloca um reticulado
void desalocarReticulado(AutomatoCelular* automato) {
    // Libera memoria das matrizes de celulas e da estrutura
AutomatoCelular
    for (int i = 0; i < automato->dimensao; i++) {
        free(automato->celulas[i]);
        free(automato->proximaGeracao[i]);
    }
    free(automato->celulas);
    free(automato->proximaGeracao);
    free(automato);
}

// Ler o reticulado inicial a partir do arquivo
AutomatoCelular* leituraReticulado(FILE* entrada, int *numGeracoes) {
    int dimensao;
    fscanf(entrada, "%d %d", &dimensao, numGeracoes);

    AutomatoCelular* automato = alocarReticulado(dimensao);

    for (int i = 0; i < dimensao; i++) {
        for (int j = 0; j < dimensao; j++) {
            fscanf(entrada, "%d", &automato->celulas[i][j]);
        }
    }

    return automato;
}

// Funcao auxiliar para contar o numero de celulas vivas vizinhas de
uma celula especifica
int contarCelulasVivasVizinhas(AutomatoCelular* automato, int linha,
int coluna) {
    int contador = 0;
    int dimensao = automato->dimensao;

    // Verifica as 8 celulas vizinhas ao redor da celula atual
```

```
    for (int i = linha - 1; i <= linha + 1; i++) {
        for (int j = coluna - 1; j <= coluna + 1; j++) {
            // Verifica se a celula esta dentro dos limites do
reticulado
            if (i >= 0 && i < dimensao && j >= 0 && j < dimensao) {
                // Ignora a propria celula
                if (i != linha || j != coluna) {
                    // Se a celula vizinha estiver viva, incrementar o
contador
                    if (automato->celulas[i][j] == 1) {
                        contador++;
                    }
                }
            }
        }
    }
    return contador;
}

// Evolui o reticulado por um numero especificado de geracoes
void evoluirReticulado(AutomatoCelular* automato, int geracoes) {
    // Verifica se ainda ha geracoes a serem evoluidas
    if (geracoes > 0) {
        int dimensao = automato->dimensao;

        // Itera sobre cada celula do reticulado
        for (int i = 0; i < dimensao; i++) {
            for (int j = 0; j < dimensao; j++) {
                // Conta o numero de celulas vivas vizinhas
                int celulasVivasVizinhas =
contarCelulasVivasVizinhas(automato, i, j);

                // Aplica as regras do jogo da vida para determinar o
estado da celula na proxima geracao
                if (automato->celulas[i][j] == 1) {
                    if (celulasVivasVizinhas < 2 || celulasVivasVizinhas
> 3) {
                        // Morte por solidao ou superpopulacao
                        automato->proximaGeracao[i][j] = 0;
                    } else {
                        // Celula viva permanece viva
                        automato->proximaGeracao[i][j] = 1;
                    }
```

```
                } else {
                    if (celulasVivasVizinhas == 3) {
                        // Renascimento de celula morta
                        automato->proximaGeracao[i][j] = 1;
                    } else {
                        // Celula morta permanece morta
                        automato->proximaGeracao[i][j] = 0;
                    }
                }
            }
        }

        // Atualiza o estado do reticulado para refletir a proxima
geracao
        for (int i = 0; i < dimensao; i++) {
            for (int j = 0; j < dimensao; j++) {
                automato->celulas[i][j] =
automato->proximaGeracao[i][j];
            }
        }

        // Chama recursivamente a funcao para evoluir as proximas
geracoes
        evoluirReticulado(automato, geracoes - 1);
    }
}

// Imprime o reticulado atual
void imprimeReticulado(AutomatoCelular* automato) {
    for (int i = 0; i < automato->dimensao; i++) {
        for (int j = 0; j < automato->dimensao; j++) {
            printf("%d ", automato->celulas[i][j]);
        }
        printf("\n");
    }
}
```