Universidade Federal de Ouro Preto - UFOP
Instituto de Ciências Exatas e Biológicas - ICEB
Departamento de Computação - DECOM
Ciência da Computação

Autômato Celular

BCC202 - Estrutura de Dados

Augusto Luna
Professor: Pedro Silva

Ouro Preto
29 de agosto de 2024

Sumário

Lista de Figuras

1 Introdução

Este relatório descreve o desenvolvimento de um código que tem como função demonstrar um Autômato Celular utilizando linguagem de programação C. Um Autômato Celular é definido por seu espaço celular e suas regras de transição. O Autômato Celular utilizado neste experimento foi o proposto pelo matemático John Horton Conway. Esse Autômato foi conhecido como jogo da vida (game of life), que consiste em uma matriz bidimensional que possui células em dois estados: viva ou morta. A cada geração uma célula pode ou não ser alterada de acordo com as células vizinhas.

1.1 Especificações do problema

O programa consiste em receber uma matriz bidimensional inicial inicializada com zeros e uns, sendo zero uma célula morta e um viva. Porém agora, utilizando uma matriz esparsa para armazenar somente células vivas, economizando memória. Após receber o número de gerações desejadas e seguindo as regras do jogo da vida, o código utilizará recursividade para aplicar essas regras e retornar a geração desejada imprimindo a matriz final correta.

1.2 Considerações iniciais

Algumas ferramentas foram utilizadas durante a criação deste projeto:

- Ambiente de desenvolvimento do código fonte: VSCode(Virtual Studio Code). [1]
- Linguagem utilizada: C.
- Ambiente de desenvolvimento da documentação: Overleaf LaTeX. [2]

1.3 Ferramentas utilizadas

Algumas ferramentas foram utilizadas para testar a implementação, como:

- *Valgrind*: ferramentas de análise dinâmica do código.

1.4 Especificações da máquina

A máquina onde o desenvolvimento e os testes foram realizados possui a seguinte configuração:

- Processador: Ryzen 7-5800H.
- Memória RAM: 16Gb.
- Sistema Operacional: Linux.

1.5 Instruções de compilação e execução

Para a compilação do projeto, basta digitar:

Compilando o projeto

```
gcc -c automato.c -Wall gcc -c tp.c -Wall gcc tp.o automato.o -o exe -lm
```

Usou-se para a compilação as seguintes opções:

- -Wall: para mostrar todos os possíveis warnings do código.

Para a execução do programa basta digitar:

```
.exe <diretório/teste.in
```

[1] ????? está disponível em https://www.

[2] Disponível em https://www.overleaf.com/

2 Desenvolvimento e Principais Funções

A implementação do programa se deu por base de 3 arquivos.c e 2 arquivo.h, sendo eles o tp.c, matriz.c, automato.c, automato.h e matriz.h. As funções mais importantes do programa são a void inserirElemento(MatrizEsparsa *matriz, int linha, int coluna, int valor), que insere os elementos não nulos na matriz esparsa, calcularProximaGeracao(AutomatoCelular *automato, int **novaGrade), que aplica as regras do jogo da vida e a evoluirReticulado(AutomatoCelular *automato, int geracoesRestantes) que é uma função recursiva que retorna a geração desejada.

2.1 Função inserirElemento()

Primeiramente, temos uma estrutura concional IF para identificar os elementos diferentes de 0.

```
    if (valor == 0){
        return;
    }
```

Após isso, definimos as linhas, as colunas e os valores da matriz esparsa.

```
   Celula *nova = (Celula *)malloc(sizeof(Celula));
    nova->linha = linha;
    nova->coluna = coluna;
    nova->valor = valor;
    nova->prox = matriz->primeira;
    matriz->primeira = nova;
}
```

2.2 Função calculaProximaGeracao()

Primeiramente, temos dois laços de repetição do tipo for aninhados para percorrer toda a matriz inicial, após isso, temos mais dois laços de repetição aninhados para contar a quantidade de células viva, chamando a função obterElemento.

```
for (int i = 0; i < automato->dimensao; i++) {
        for (int j = 0; j < automato->dimensao; j++) {
            int vivos = 0;
            for (int linhasVizinhas = -1; linhasVizinhas <= 1; linhasVizinhas
                ++) {
                for (int colunasVizinhas = -1; colunasVizinhas <= 1;
                    colunasVizinhas++) {
                    if (linhasVizinhas == 0 && colunasVizinhas == 0) continue;
                    int ni = i + linhasVizinhas;
                    int nj = j + colunasVizinhas;
                    if (ni >= 0 && ni < automato->dimensao && nj >= 0 && nj <
                        automato->dimensao) {
                        vivos += obterElemento(automato->grade, ni, nj);
                    }
                }
            }
```

Logo depois, temos mais estruturas condicionais IF para aplicarmos as regras do jogo da vida de Conway. Uma célula morre por solidão caso tenha menos que duas células vizinhas e por superpopulação caso tenha mais que 3 células vizinhas vivas. Uma célula morta se torna viva caso tenha exatamente 3 células vizinhas vivas.

```
int valorAtual = obterElemento(automato->grade, i, j);
            if (valorAtual == 1) {
                if (vivos < 2 || vivos > 3) {
                    inserirElemento(novaGrade, i, j, 0);
                } else {
                    inserirElemento(novaGrade, i, j, 1);
                }
            } else {
```

```
                if (vivos == 3) { // torna-se viva
                    inserirElemento(novaGrade, i, j, 1);
                } else {
                    inserirElemento(novaGrade, i, j, 0);
                }
            }
        }
    }
}
```

2.3 Função evoluirReticulado()

Esta é a nossa função recursiva. Inicialmente ela tem sua condição de parada, que é quando o número de gerações restantes chega em 0.

```
if (geracoesRestantes == 0){
        return 0;
    }
```

Em seguida, criamos uma matriz esparsa substituir a matriz antiga por ela.

```
    MatrizEsparsa *novaGrade = criarMatrizEsparsa(automato->dimensao);
```

Agora chamamos a função auxiliar para calcular a próxima geração.

```
    calcularProximaGeracao(automato, novaGrade);
```

Por fim, liberamos a memória da nova matriz e chamamos a função recursivamente. Chamamos o parâmetro (geracoesRestantes -1) para fazer todas as gerações necessárias e chegar no caso base e adicionamos +1 para contar quantas gerações foram feitas.

```
     liberarMatrizEsparsa(automato->grade);
    automato->grade = novaGrade;

    return evoluirReticulado(automato, geracoesRestantes - 1) + 1;
}
```

3 Testes

Figura 1: Matriz 5x5-1 geração entrada.

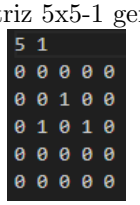


Figura 2: Matriz 5x5-1 geração saída.

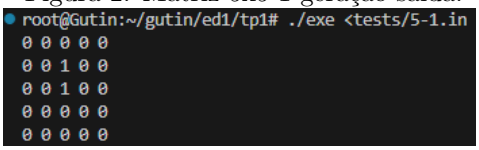


Figura 3: Matriz 20x20-1 geração entrada.

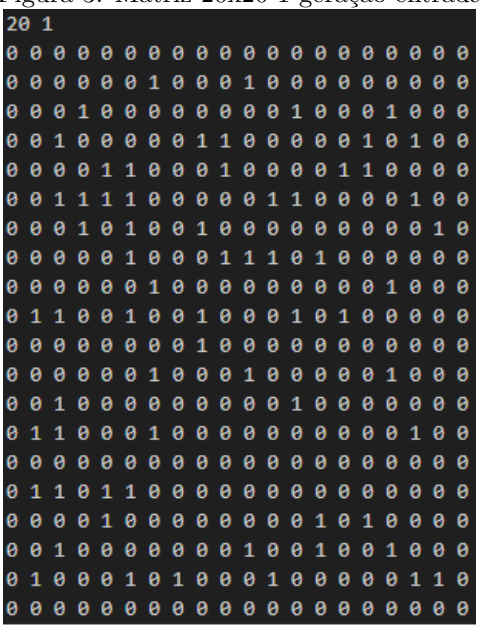
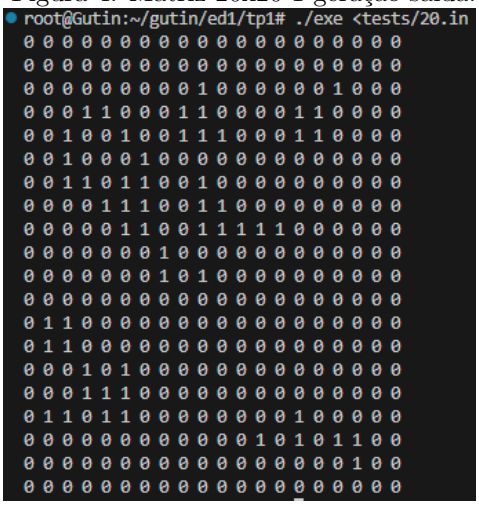


Figura 4: Matriz 20x20-1 geração saída.



4 Conclusão

Neste trabalho, o objetivo principal era demonstrar o Autômato Celular conhecido como jogo da vida desenvolvido por John Von Neumann. Para atingir esse objetivo, foram implementadas listas encadeadas para armazenar elementos não nulos em uma matriz esparsa.

A utilização dos TADs AutomatoCelular e TAD MatrizEsparsa foram fundamentais para facilitar a leitura das entradas e o entendimento do código. Com a realização deste trabalho, alguns conhecimentos foram adquiridos e melhorados, como a alocação dinâmica para a criação de uma matriz de qualquer tamanho desejado, a utilização de uma função recursiva e a utilização de listas encadeadas que deixaram o código bem desafiador.

Em resumo, o trabalho propôs a implementação de listas encadeadas para desafiar a nossa capacidade de criar uma solução para o desafio proposto, assim conseguindo criar inúmeras gerações para o jogo da vida de Conway.