

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
25 de julho de 2024

Sumário

1	Introdução	1
1.1	Especificações do problema	1
1.2	Considerações iniciais	1
1.3	Ferramentas utilizadas	1
1.4	Especificações da máquina	1
1.5	Instruções de compilação e execução	1
2	Desenvolvimento e Principais Funções	2
2.1	Função calculaProximaGeracao()	2
2.2	Função evoluirReticulado()	2
3	Testes	4
4	Conclusão	6

Lista de Figuras

1	Matriz 5x5-1 geração entrada.	4
2	Matriz 5x5-1 geração saída.	4
3	Matriz 20x20-1 geração entrada.	4
4	Matriz 20x20-1 geração saída.	5

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. 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 L^AT_EX. 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.>

²Disponível em <https://www.overleaf.com/>

2 Desenvolvimento e Principais Funções

A implementação do programa se deu por base de 2 arquivos.c e um arquivo.h, sendo eles o tp.c, automato.c e a automato.h. As funções mais importantes do programa são a 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 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 mas agora para contar quantas células vivas tem ao redor da célula que os primeiros laços de repetição estão passando (linha 5). Em seguida, temos duas estruturas condicionais IF para verificar se a célula vizinha está dentro do limite da matriz, caso não esteja, ela não é válida para ser comparada, caso esteja, incremente o contador de células vivas (caso há alguma viva).

```
1 for (int i = 0; i < automato->dimensao; i++) {
2     for (int j = 0; j < automato->dimensao; j++) {
3         int vivos = 0;
4         for (int linhasVizinhas = -1; linhasVizinhas <= 1; linhasVizinhas
5             ++){
6             for (int colunasVizinhas = -1; colunasVizinhas <= 1;
7                 colunasVizinhas++){
8                 if (linhasVizinhas == 0 && colunasVizinhas == 0) continue;
9                 atual
10                int ni = i + linhasVizinhas;
11                int nj = j + colunasVizinhas;
12                if (ni >= 0 && ni < automato->dimensao && nj >= 0 && nj <
13                    automato->dimensao) {
14                    vivos += automato->grade[ni][nj];
15                }
16            }
17        }
```

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.

```
1 if (automato->grade[i][j] == 1) {
2     if (vivos < 2 || vivos > 3) {
3         novaGrade[i][j] = 0;
4     } else {
5         novaGrade[i][j] = 1;
6     }
7 } else {
8     if (vivos == 3) {
9         novaGrade[i][j] = 1;
10    } else {
11        novaGrade[i][j] = 0;
12    }
13 }
14 }
15 }
16 }
17 }
```

2.2 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.

```

1 if (geracoesRestantes == 0){
2     return 0;
3 }

```

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

```

1 int **novaGrade = (int **)malloc(automato->dimensao * sizeof(int *));
2 for (int i = 0; i < automato->dimensao; i++) {
3     novaGrade[i] = (int *)malloc(automato->dimensao * sizeof(int));
4 }

```

Agora chamamos a função auxiliar para calcular a próxima geração.

```

1 calcularProximaGeracao(automato, novaGrade);

```

Após isso atualiza a matriz antiga pela nova gerada.

```

1 for (int i = 0; i < automato->dimensao; i++) {
2     for (int j = 0; j < automato->dimensao; j++) {
3         automato->grade[i][j] = novaGrade[i][j];
4     }
5 }

```

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.

```

1 for (int i = 0; i < automato->dimensao; i++) {
2     free(novaGrade[i]);
3 }
4 free(novaGrade);
5
6 return evoluirReticulado(automato, geracoesRestantes - 1) + 1;

```

3 Testes

Figura 1: Matriz 5x5-1 geração entrada.

```
5 1
0 0 0 0 0
0 0 1 0 0
0 1 0 1 0
0 0 0 0 0
0 0 0 0 0
```

Figura 2: Matriz 5x5-1 geração saída.

```
root@Gutin:~/gutim/ed1/tp1# ./exe <tests/5-1.in
0 0 0 0 0
0 0 1 0 0
0 0 1 0 0
0 0 0 0 0
0 0 0 0 0
```

Figura 3: Matriz 20x20-1 geração entrada.

```
20 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0
0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 1 0 1 0 0
0 0 0 0 1 1 0 0 0 1 0 0 0 0 1 1 0 0 0 0
0 0 1 1 1 1 0 0 0 0 0 1 1 0 0 0 0 1 0 0
0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 1 0 0 0 1 1 1 0 1 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0
0 1 1 0 0 1 0 0 1 0 0 0 1 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0
0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0
0 1 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 1 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Figura 4: Matriz 20x20-1 geração saída.

```

● root@Gutin:~/gutin/ed1/tp1# ./exe <tests/20.in
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0
0 0 0 1 1 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0
0 0 1 0 0 1 0 0 1 1 1 0 0 0 1 1 0 0 0 0
0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 0 0 1 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 0 0 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

```

4 Conclusão

Neste trabalho, o objetivo principal era demonstrar o Autômato Celular conhecido como jogo da vida desenvolvido por John Horton Conway. Para atingir esse objetivo, foram implementadas uma função recursiva e uma função auxiliar.

A utilização do TAD AutomatoCelular foi fundamental 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 e a utilização de uma função recursiva que deixou o código mais desafiador.

Em resumo, o trabalho propôs a implementação de recursão 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.