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

Automato Celular

BCC202 - Estrutura de Dados 1

Lucas Chagas, Nicolas Mendes, Pedro Morais
Professor: Pedro Henrique Lopes Silva

Ouro Preto
6 de janeiro de 2023

Sumário

Lista de Figuras

1 Introdução

O objetivo deste trabalho pratico foi implementar o "Game of Life"proposto por "John Horton Conway".

1.1 Especificações do problema

O programa do "Game of Life", consiste em criar uma matriz quadrada que contem "celulas vivas", que iniciarão uma sequencia de açoes em cadeia. Sendo essas açoes definidas por meio da posiçao de "celulas vivas"perante outras "celulas vivas".

1.2 Considerações iniciais

Algumas ferramentas foram utilizadas durante a criação deste projeto:

- Ambiente de desenvolvimento do código fonte: Visual 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:

- *CLANG*: ferramentas de análise estática do código.
- *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 3700x.
- Memória RAM: 8 Gb.
- Sistema Operacional: Ubuntu.

1.5 Instruções de compilação e execução

Para a compilação do projeto, basta digitar:

Compilando o projeto

```
gcc automato.c tp.c -o exe -std=c99 -Wall -pg -g
```

Usou-se para a compilação as seguintes opções:

- *-std=99*: para usar-se o padrão ANSI C 99, conforme exigido.
- *-g*: para compilar com informação de depuração e ser usado pelo Valgrind.
- *-Wall*: para mostrar todos os possível *warnings* do código.
- *-pg*: para gerar o arquivo para fazer-se o profiling para identificar gargalos no programa.

Para a execução do programa basta digitar:

```
./exe caminho_até_o_arquivo_de_entrada opcao
```

Onde "opcao" pode ser: "p" para fazer a, "d" para fazer b e "n" para fazer c.

[1] Vscode está disponível em https://code.visualstudio.com/
[2] Disponível em https://www.overleaf.com/

2 Desenvolvimento

As funções auxiliares implementadas no código(localizadas no arquivo "automato.c"), consiste em manipular matrizes do tipo abstrato "AutomatoCelular", sendo elas: "alocarReticulado", "desalocaReticulado", "LeituraReticulado", "imprimeReticulado", "zeramatriz"e "**Reticuladoarq".

As funções Principais do codigo sao responsaveis pela logica do reticulado onde as células residem, a função "verifica"analisa quantas células vivas existem com relação a uma coordenada do reticulado e a função "evoluirReticulado"aplica as regras do "automato celular"e atualiza o reticulado para uma nova geração.

2.1 Regras do automato celular

- Uma célula viva se mantém viva se tiver 2 (duas) ou 3 (três) células vizinhas vivas.
- Uma célula viva torna-se morta se houver mais de 3 (três) células vizinhas vivas por superpopulação (sufocamento).
- Uma célula viva torna-se morta se houver menos que 2 (duas) células vizinhas vivas por subpopulação (solidão).
- Uma célula morta torna-se viva se houver exatamente 3 (três) células vizinhas vivas, por reprodução (renascimento).

2.2 Funções principais

- "verifica": Esta função recebe uma matriz do tipo "AutomatoCelular"e uma coordenada desta mesma matriz, com esta coordenada a função percorre pelas coordenadas ao redor da coordenada recebida e conta quantas "células vivas"existem em volta desta coordenada. E por fim, retorna este valor como um inteiro.
- "evoluirReticulado": Esta função recebe uma matriz do tipo "AutomatoCelular"e utiliza da função "verifica"para aplicar as "regras do autômato celular"e preparar uma nova geração do reticulado. Novas gerações são armazenadas no atributo "estadofuturo", que posteriormente é transacionado para o atributo "estadoatual".

2.3 Funções auxiliares

- "alocarReticulado": Esta função cria uma matriz quadrada do tipo "AutomatoCelular" baseada em um número inteiro que é recebido por parâmetro.
- "LeituraReticulado": Esta função recebe uma matriz do tipo"AutomatoCelular"e recebe os valores que serão colocados na matriz pelo usuários por intermédio do comando "scanf".
- "zeramatriz": recebe uma matriz do tipo "AutomatoCelular" e zera todos os componentes presentes de cada elemento da matriz, componentes esses que pertencem ao tipo abstrato citado.
- "imprimeReticulado": Esta função recebe uma matriz do tipo "AutomatoCelular" e imprime a matriz no terminal por meio do comando "printf".
- "desalocaReticulado": recebe uma matriz de "Automatocelular"e a desaloca usando o comando "free".
- "**Reticuladoarq":Le um arquivo e retorna uma matriz do tipo "Automatocelular".

2.4 Imagens

A seguir algumas imagens de alguns casos de teste:

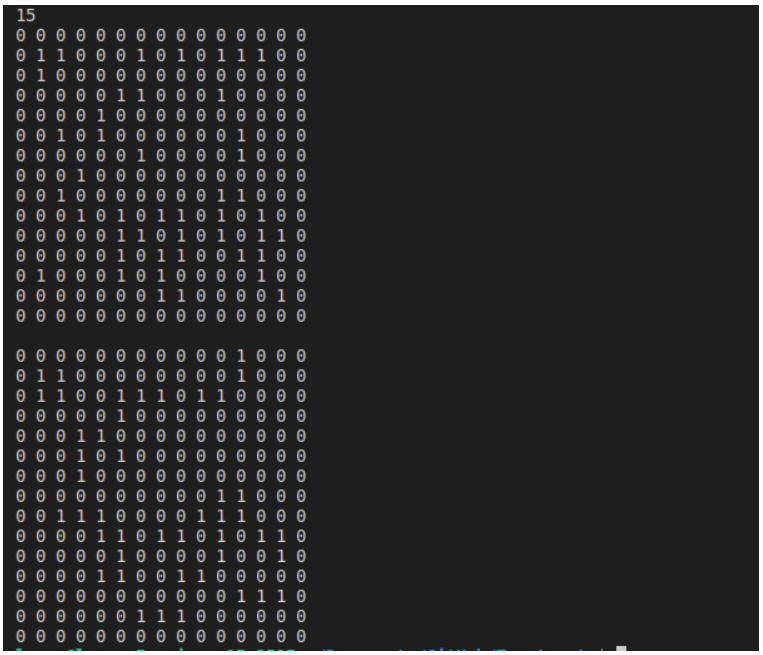

Figura 1: Primeiro caso de teste(input e output).

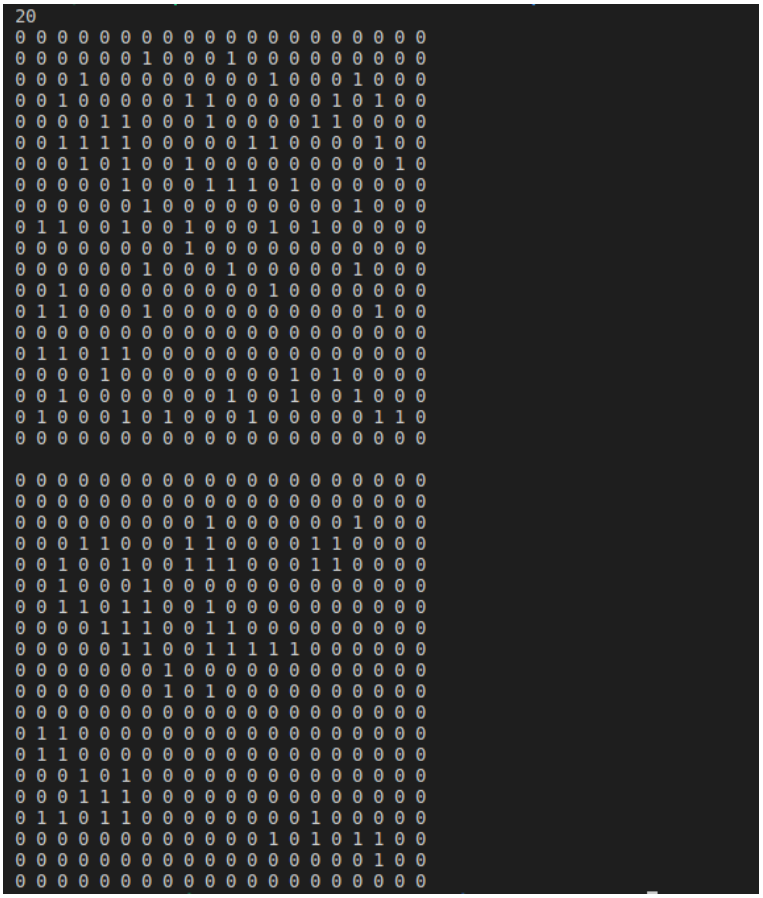

Figura 2: Segundo caso de teste(input e output).

2.5 Trechos do código

A seguir estão as implementações das funções principais do código e a struct usada..

```
struct automatocelular
{
    //variaveis para fazer a troca entre as geracoes do reticulado
    int estadoatual;
    int estadofuturo;

};

//funcao que com base em uma posicao de uma matriz, retorna a quantidade de casas ao redor possuem o valor 1 em relacao a posicao dada


int verifica(AutomatoCelular**M,int size,int row,int col){
    int count = 0;


    for(int i = row-1 ; i<=row+1 ; i++){

        for(int j = col-1 ; j<=col+1 ; j++){

            if((i == row && j == col) || (i<0||j<0) || (i>=size || j>=size)){
                continue;
            }
            else if(M[i][j].estadoatual==1){
                count++;
            }
        }
    }


    return count;
}


//funcao responsavel por atualizar um reticulado(matriz) em sua proxima geracao
void evoluirReticulado(AutomatoCelular **M,int size){
    int aux;
    for(int i = 0; i<size;i++){
        for(int j = 0; j<size;j++){
            aux = verifica(M,size,i,j);
            //condicional de vida de uma celula
            if( (M[i][j].estadoatual == 1) && (aux == 2||aux == 3) ){
                M[i][j].estadofuturo = 1;
            }
            //condicional de morte de uma celula
            else if( (M[i][j].estadoatual == 1) && (aux > 3||aux<2)){
                M[i][j].estadofuturo = 0;
            }
            //condicional de ressurreicao de uma celula
            else if((M[i][j].estadoatual == 0) && (aux == 3)){
                M[i][j].estadofuturo = 1;
            }
        }

    }

    //loop para atualizar as geracoes
```

```
57      for(int i = 0; i<size;i++){
58          for(int j = 0; j<size;j++){
59              M[i][j].estadoatual = M[i][j].estadofuturo;
60          }
61
62      }
63
64  }
```

3 Experimetos e Resultados

Foram realizados vários testes com diversos tipos de input, com isso conseguimos corrigir alguns erros ao longo do processo, porém o problema "conditional jump or move depends on unitialised value(s)"identificado pelo valgrind foi o que mais se destacou. Para resolvermos este problema implementamos uma função para "setar"todos os valores da matriz em "0".

4 Considerações Finais

O código foi capaz de lidar com todos os casos de teste propostos no "runcodes"e não apresentou nenhum erro na verificação do valgrind. O código ficou suficientemente intuitivo, sendo comentado e indentado em toda sua constituição.