Jogo da Vida

João Vitor Coelho Oliveira
Matrícula: 23.1.4133

24 de julho de 2024

Universidade Federal de Ouro Preto
Ciência da Computação
Estrutura de Dados I

Resumo

Neste trabalho prático de Estrutura de Dados I, será abordado implementação do Jogo da Vida de John Conway em C. O foco está na metodologia utilizada, os resultados obtidos e a análise de desempenho, incluindo o uso de Valgrind para observação do tempo de execução.

1 Introdução

O jogo da vida é um autômato celular desenvolvido pelo matemático britânico John Horton Conway em 1970. O jogo foi criado de modo a reproduzir, através de regras simples, as alterações e mudanças em grupos de seres vivos, tendo aplicações em diversas áreas da ciência.

1.1 Especificações do Problema

Em resumo, a cada iteração, as células podem viver, morrer ou se reproduzir seguindo regras específicas baseadas no número de vizinhos vivos. O programa recebe uma configuração inicial e o número de iterações, exibindo o estado final do grid após a execução das iterações.

`Regras:`

1. `Q`ualquer célula viva com menos de dois vizinhos vivos morre de solidão.

2. Qualquer célula viva com mais de três vizinhos vivos morre de superpopulação.

3. Qualquer célula com exatamente três vizinhos vivos se torna uma célula viva.

4. Qualquer célula com dois vizinhos vivos continua no mesmo estado para a próxima geração.

1.2 Especificações da Máquina

```
Processador : Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz
2.71 GHz

Memória RAM  : 16,0 GB

Sistema Operacional : Windows 10
```

2 Considerações Iniciais

- Ambiente de desenvolvimento do código fonte: Visual Studio Code
- Linguagem Utilizada: C
- Ambiente de desenvolvimento da documentação: Overleaf / LaTeX

Para a implementação foram usadas ferramentas como Valgrind, para analisar dinâmicamente o código e Gcc como compilador. Também foi usado WSL para executar um sistema de arquivos linux para realizar testes em outros tipos de sistemas operacionais.

3 Metodologia

A implementação foi dividida em três partes principais: as funções de manipulação, lógica do jogo e o arquivo principal, que utiliza as funções definidas nos outros arquivos para executar o autômato.

3.1 Detalhes da Implementação

A estrutura strDados é utilizada para armazenar a matriz principal (matrizPrincipal) e o tamanho da matriz (tamanhoMat). Esta estrutura permite encapsular as informações relevantes para a operação do autômato celular.

Figura 1: Estrutura strDados

Para atingir o objetivo, foi construído um Tipo Abstrato de Dados (TAD) `Automato` como representação do reticulado. O TAD implementa as seguintes operações:

- `alocarReticulado`: Aloca memória para uma matriz quadrada de tamanho 'tamanhoMat x tamanhoMat'.

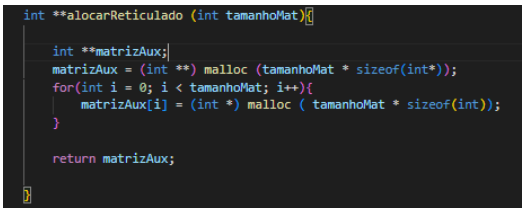

Figura 2: Função alocarReticulado

- `desalocarReticulado`: Libera a memória alocada para a matriz.

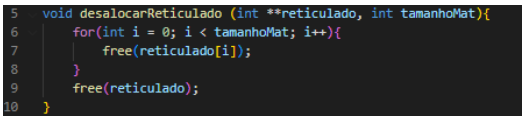

Figura 3: Função desalocarReticulado

- `leEntradaConsole`: Lê os valores da matriz a partir do terminal.

```c
void leEntradaConsole (int *qntMov, strDados *dados){
    scanf("%d", &dados->tamanhoMat);
    scanf("%d", qntMov);

    dados->matrizPrincipal = alocarReticulado(dados->tamanhoMat);

    for(int i = 0; i < dados->tamanhoMat; i++) {
        for(int j = 0; j < dados->tamanhoMat; j++) {
            scanf("%d", &dados->matrizPrincipal[i][j]);
        }
    }
}
```

Figura 4: Função leEntradaConsole

- `evoluirReticulado`: Implementa a lógica de evolução do autômato celular ao longo de várias gerações, conforme as regras do jogo da vida.

```c
void evoluirReticulado (strDados *dados, int qntMov){

    int tamanhoMat = dados->tamanhoMat;

  for (int geracao = 0; geracao < qntMov; geracao++) {
     int **novaMatriz = alocarReticulado(tamanhoMat);

     for (int i = 0; i < tamanhoMat; i++) {
        for (int j = 0; j < tamanhoMat; j++) {
            int vivos = 0;

            for (int x = -1; x <= 1; x++) {
                for (int y = -1; y <= 1; y++) {
                    if (x == 0 && y == 0) continue;
                    int ni = i + x, nj = j + y;
                    if (ni >= 0 && ni < tamanhoMat && nj >= 0 && nj < tamanhoMat) {
                        vivos += dados->matrizPrincipal[ni][nj];
                    }
                }
            }

            if (dados->matrizPrincipal[i][j] == 1) {
                if (vivos < 2 || vivos > 3) {
                    novaMatriz[i][j] = 0;
                } else {
                    novaMatriz[i][j] = 1;
                }
            } else {
                if (vivos == 3) {
                    novaMatriz[i][j] = 1;
                } else {
                    novaMatriz[i][j] = 0;
                }
            }
        }
     }

     desalocarReticulado(dados->matrizPrincipal, tamanhoMat);
     dados->matrizPrincipal = novaMatriz;
  }
}
```

Figura 5: Função evoluirReticulado

- `imprimeReticulado`: Imprime a matriz principal no console.

```c
void imprimeReticulado(strDados *dados){
    int tamanhoMat = dados->tamanhoMat;

    for(int i = 0; i < tamanhoMat; i++){
        for(int j = 0; j < tamanhoMat; j++){
            printf("%d ", dados -> matrizPrincipal[i][j]);
        }
        printf("\n");
    }
}
```

Figura 6: Função imprimeReticulado

3.2 Entrada

A entrada foi fornecida por meio de arquivos. A primeira linha especifica as dimensões do reticulado D e o número de gerações a serem processadas.

Em seguida, é apresentada uma matriz de valores binários que reproduz o reticulado do jogo da vida a ser analisado.

3.3 Saída

A saída é uma matriz com a mesma dimensão da entrada, contendo o estado das células na próxima geração, com base no conjunto de regras do jogo da vida.

3.4 Casos de Teste

Para verificar a performance e corretude do código, foram utilizados vários casos de teste com diferentes tamanhos de grids e configurações iniciais de células vivas, juntamente com um corretor em python fornecido nas informações do trabalho. Esses testes garantiram que o programa responde corretamente às regras do jogo e que todas as células evoluem conforme esperado.

Os casos teste foram definidos em arquivos de texto contendo a dimensão do grid, o número de iterações e o estado inicial.

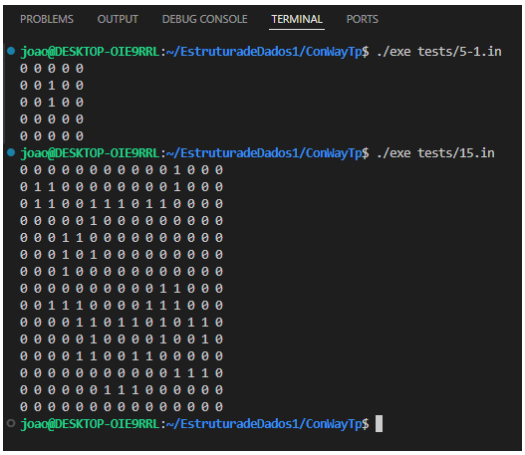

Figura 7: Testes de arquivo com dimensoes 5x5 e 15x15

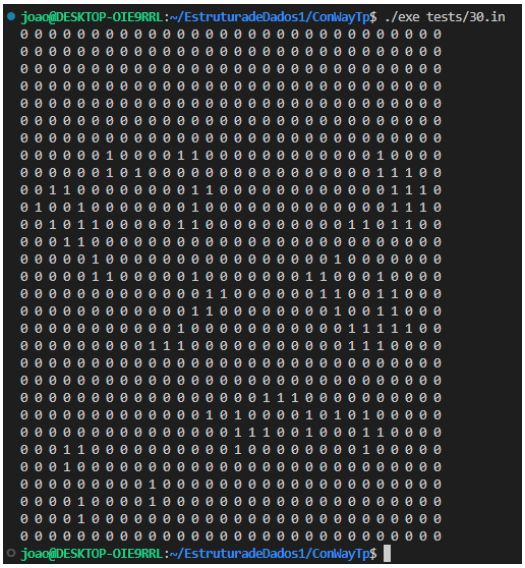

Figura 8: Teste de arquivo com dimensão 30x30

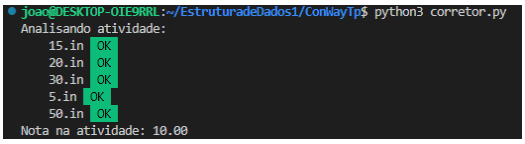

Figura 9: Teste utilizando corretor.py

3.5 Resultados

Para avaliar o desempenho da implementação, utilizamos o Valgrind para monitorar o tempo de execução e a utilização de memória, juntam. A seguir, será apresentado os resultados obtidos.

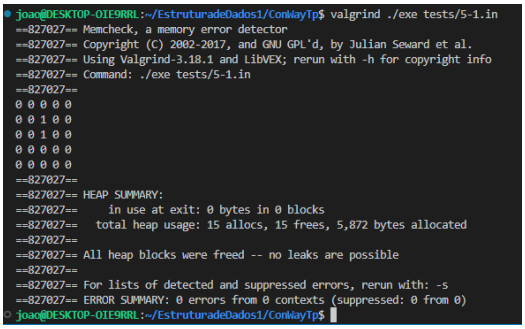

Figura 10: Testes de arquivo usando Valgrind (5x5)

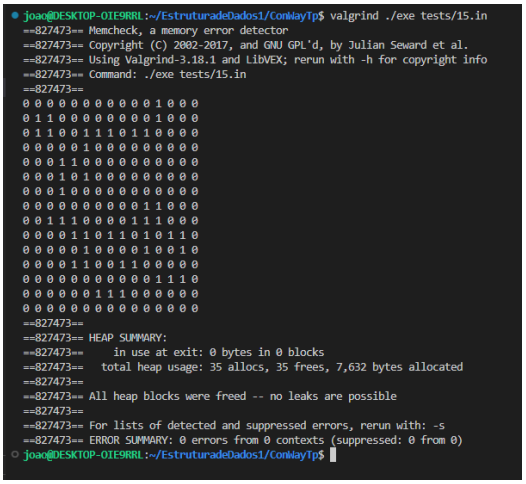

Figura 11: Teste de arquivo usando Valgrind (15x15)

4 Lógica de Resolução do Problema

A função evoluirReticulado é responsável por implementar a lógica de evolução do autômato celular ao longo de várias gerações, conforme as regras do jogo. O primeiro loop itera sobre o número de gerações especificado por qntMov. Em cada iteração, a matriz principal evolui uma geração.

```
54      for (int geracao = 0; geracao < qntMov; geracao++) {
```

Figura 12: Loop de gerações especificado por qntMov

Para cada geração, uma nova matriz (novaMatriz) é alocada para armazenar o estado atualizado da matriz principal. Isso permite que as atualizações sejam feitas de forma independente, sem interferir na matriz atual.

```
55          int **novaMatriz = alocarReticulado(tamanhoMat);
```

Figura 13: atualização da `novaMatriz`

Dois loops aninhados percorrem todas as células da matriz principal. i e j são os índices das células na matriz.

```
57          for (int i = 0; i < tamanhoMat; i++) {
58              for (int j = 0; j < tamanhoMat; j++) {
```

Figura 14: Loops aninhados

Para cada célula (i, j), são contados os vizinhos vivos. Isso é feito usando dois loops que percorrem a vizinhança 3x3 ao redor da célula.

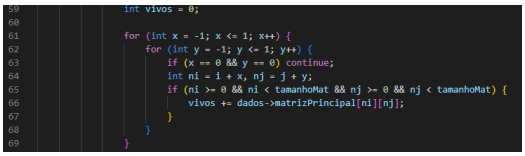

Figura 15: Loops para contagem de vizinhos

1. As variáveis x e y percorrem os valores -1, 0 e 1, representando os deslocamentos para verificar os vizinhos ao redor da célula atual.
2. A condição `if (x == 0 && y == 0) continue;` é usada para ignorar a própria célula.
3. As variáveis ni e nj são usadas para calcular as posições dos vizinhos.
4. A condição `if (ni >= 0 && ni < tamanhoMat && nj >= 0 && nj < tamanhoMat)` verifica se os índices dos vizinhos estão dentro dos limites da matriz.
5. O contador vivos é incrementado para cada vizinho vivo encontrado `(dados -> matriz-Principal[ni][nj])`.

Figura 16: Loop para encontrar vizinhos

As regras do jogo da vida são aplicadas para determinar o novo estado da célula (i, j) na nova matriz:

- Célula Viva:
  - Se a célula está viva (`dados -> matrizPrincipal[i][j] == 1`):
    - Ela se torna viva (novaMatriz[i][j] = 1) se tiver menos de 2 ou mais de 3 vizinhos vivos (subpopulação ou superpopulação).

        * Caso contrário, ela continua viva (novaMatriz[i][j] = 1).

- Célula Morta:
    - Se a célula está morta (`dados -> matrizPrincipal[i][j] == 0`):
        * Ela se torna morta (novaMatriz[i][j] = 0) se tiver exatamente 3 vizinhos vivos (reprodução).
        * Caso contrário, ela permanece morta (novaMatriz[i][j] = 0).

Após atualizar todas as células, a matriz principal antiga é desalocada usando a função desalocarReticulado. Em seguida, a nova matriz (novaMatriz) é atribuída à matriz principal `(dados->matrizPrincipal)`.

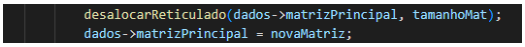

Figura 17: chamada da `novaMatriz para a matriz principal`

5 Conclusão

O núcleo do projeto é a função evoluirReticulado, que aplica com sucesso as regras do jogo da vida e permite observar como padrões surpreendentes podem ser criados por interações locais entre células. A estrutura modular do código facilitou a leitura, manutenção e expansão do programa. Isso organizou o processo de desenvolvimento e diminuiu a probabilidade de erros.

Esse projeto, além da parte técnica, enfatiza o gerenciamento de recursos eficaz, especialmente a memória, para garantir que a implementação seja não apenas funcional, mas também eficiente.

6 Referências

- Gardner, M. (1970). Mathematical games: The fantastic combinations of John Conway's new solitaire game "life". Scientific American, 223(4):120-123.
- Valgrind. Disponível em: `http://valgrind.org/`
- Visual Studio Code. Disponível em: `https://code.visualstudio.com/`