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

TP2 - Jogo da Vida com matriz esparsa

BCC202 - Estrutura de Dados I

Jouberth Pereira e Enzo Bernardes
Professor: Pedro Henrique Lopes Silva

Ouro Preto
29 de agosto de 2024

Sumário

Lista de Códigos Fonte

1 Introdução

O objetivo desse trabalho é implementar uma simulação do jogo jogo da vida (Conway's Game of Life)[1], através de uma matriz esparsa, isto é uma matriz onde apenas as células que estão vivas são representadas e armazenadas.

O jogo da vida (Conway's Game of Life)[1], com regras fixas e simples que podem gerar comportamentos bastante complexos. O reticulado bidimensional possui células em dois estados: viva ou morta, sendo a morta, não armazenada na memória. A cada transição (ou iteração), o estado de uma célula pode ou não ser alterado de acordo com as células vizinhas. O conjunto de regras incluem:

- 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).

A codificação deve ser feita em C seguindo as boas práticas de programação. A utilização de TAD é necessária para o AutomatoCelular sendo alocada dinâmica e a função evoluirReticulado pode ser feito de recursividade ou não. A entrega deve ser feita em três arquivos tp.c, automato.h, automato.c, matriz.h e matriz.c . O programa não deve apresentar *memory leaks*. Utilizaremos o *valgrind* para verificar vazamento de memória.

Nesse relatório iremos abordar novamente os passos que utilizamos para implementar o jogo da vida porém, dessa vez, utilizaremos uma matriz esparsa. O código foi feito utilizando a linguagem C.

2 Implementação

Primeiramente, para a implementação, iniciamos o automato alocando os espaços do *"esqueleto"* da matriz esparsa, então, lemos a matriz e inserimos apenas as células vivas. (O fato de uma célula estar na matriz, significa que ela está viva.)

Para o funcionamento do jogo da vida, usamos a função *evoluirReticulado* que passa por cada posição da matriz, (esteja ela viva ou não) usando função *pesquisaCelula* em cada célula vizinha, essa função nos diz se a tal célula está viva ou morta, assim, temos o numero de vizinhos vivos.

Assim, é possível aplicar as regras do jogo da vida e decidir se tal coordenada estará viva ou morta na próxima geração. Caso a célula fique viva, adicionamos ela à um reticulado auxiliar, no fim, copiamos esse reticulado auxiliar para o automato.

3 Impressões gerais

No geral o trabalho foi interessante para aprofundar nossos conhecimentos sobre listas encadeadas e aprendemos o conceito de uma matriz esparsa.

Sobre a implementação da interface gráfica Allegro [2], grande parte do trabalho foi reaproveitado do trabalho prático anterior portanto, não tivemos dificuldade alguma na implementação.

4 Análise

Os resultados confirmam que o código é capaz de simular corretamente o "Game of Life"com qualquer entrada de dimensão do reticulado e qualquer número de gerações. Isso implica que as funções principais *evoluirReticulado* e *pesquisarCelula* funcionam perfeitamente e podem ser usadas para resolver problemas similares no futuro. Entretanto, nossas análises foram limitadas a padrões menores, e talvez seja necessário realizar algumas otimizações no código em caso de valores extraordinários para dimensões do reticulado e para o número de gerações, com o intuito de reduzir o tempo de execução.

4.1 Análise de Complexidade

O obejtivo principal nessa seção é analisar a ordem de complexidade do tempo de execução da funcão *evoluirReticulado*, porém ela faz uso de duas outras funções, por isso, também analisaremos *pesquisarCelula* e *insereCelula*.

4.1.1 Pontos base para a análise

- Nesta análise consideraremos todos os acessos à qualquer celula do reticulado como custo igual a N, tanto em acessos e em comparações, este custo é devido à natureza das listas encadeadas, onde para acessar qualquer célula é necessário percorrer todas as células anteriores para acessar ela.
- Analisaremos somente o custo do evoluirReticulado juntamente com as principais funções utilizadas dentro da mesma.
- Considere **N** o tamanho da largura e altura do reticulado.

4.1.2 *pesquisaCelula*

Esta função, verifica a existência de uma célula com as coordenadas passadas por parâmetro, retornando *true* caso encontre ou *false* caso contrário. Como as linhas são listas encadeadas, teremos uma complexidade que depende do número de itens nessa lista, sendo **N** a quantidade de itens nessa lista, a complexidade de tempo seria apenas **O(n)**.

```
    bool pesquisaCelula( Matriz* matriz, int x, int y )
{
    Celula* aux = matriz->vetLinhas[y].pCabeca->direitaProx;

    while (aux != NULL && aux->x <= x)
    {
        if (aux->x == x)
        {
            return true;
        }
        aux = aux->direitaProx;
    }

    return false;
}
```

Código 1: Código onde acontece a pesquisa dada uma cordenada do reticulado.

4.1.3 *evoluirReticulado*

Esta função passa por cada célula do reticulado verificando as posições adjacentes de uma dada coordenada com a função *pesquisaCelula* e incrementando o número de vizinhos vivos. E, caso necessário, utilizados o inserirCelula para adicionar uma nova celula sempre no final da lista.

Por fim, adicionamos todo o conteúdo da *novaGeração* ao reticulado do automato. Assimm, a nossa função *evoluirReticulado* e todas funções internas necessárias para o funcionamento da mesma tem uma ordem de complexidade de tempo **O(n^8) x Gerações**.

```
void evoluirReticulado( Automato* automato )
{
    for (int i = 0; i < automato->geracao; i++)
    {
        Matriz* novaGeracao = iniciaMatriz(automato->tam);

        for (int j = 0; j < automato->tam; j++)
        {
            for (int k = 0; k < automato->tam; k++)
            {
```

```
                int vizinhos = 0;

                for (int l = -1; l <= 1; l++)
                {
                    for (int m = -1; m <= 1; m++)
                    {
                        if (l == 0 && m == 0)
                        {
                            continue;
                        }

                        int x = k + m;
                        int y = j + l;


                        if (podeAcessar(x, y, automato->tam) && pesquisaCelula
                            (automato->reticulado, x, y))
                        {
                            vizinhos++;
                        }
                    }
                }

                if (vizinhos == 3 || (vizinhos == 2 && pesquisaCelula(
                    automato->reticulado, k, j ) ) )
                {
                    insereCelula(novaGeracao, k, j);
                }
            }
        }

        desalocaMatriz(automato->reticulado, automato->tam);

        automato->reticulado = novaGeracao;
    }
}
```

Código 2: Loop principal da função *evoluirReticulado* onde acontecem as checagens.

5 Diretrizes de compilação

Entrar na pasta desejada (**tp** ou **alegro**) e utilizar o comando make no terminal, depois rodar o programa utilizando um arquivo de entrada como exemplificado abaixo.

```
    $~ make
    $~ ./exe < {nome_do_arquivo_de_entrada.in}
```

Código 3: Código para rodar o programa.

6 Conclusão

Neste trabalho, utilizamos a linguagem C e suas várias bibliotecas para implementar as funções necessárias para as regras do Jogo da Vida. Aprendemos sobre oque é uma matriz esparsa e como implementa-lá. O trabalho apresentou um bom nível de dificuldade e um tópico interessante a ser explorado.

Referências

[1] Mathematical Games. The fantastic combinations of john conway's new solitaire game "life" by martin gardner. *Scientific American*, 223:120–123, 1970.

[2] Marcos Oliveira Terminal Root. Aprenda a criar jogos com allegro c/c++ no windows e linux. pages https://terminalroot.com.br/2022/12/aprenda–a–criar–jogos–com–allegro–c–cpp–no–windows–e–linux.html, 2022.