USP - Universidade de São Paulo
Instituto de Ciências Matemáticas e Computação

Introdução à Ciência da Computação I
SCC0221

ADMIRÁVEL MUNDO CAÓTICO

Em memória ao matemático John Horton Conway, falecido recentemente de Covid-19 aos 82 anos.
"And as the most magical mathematician in the world, his legacy – as his life – lives on."

1 Introdução

Desde que seres humanos existem, surgiram diversos questionamentos com relação ao que todo mundo acaba adquirindo, sem nem perceber: a vida. Passamos de questionamentos sobre a existência de vida extraterrestre até a qual foi o motivo que possibilitou que organismos biológicos extremamente complexos surgissem tão bem na Terra. Esse tipo de questionamento origina muitas teorias sobre o surgimento da vida: vivemos em um mundo considerado *perfeitamente ajustado*? Qual é a probabilidade que outro planeta como o nosso surja ou exista no universo? É por isso que estudamos autômatos celulares.

1.1 Uma introdução aos autômatos celulares

A evolução da computação permitiu que a humanidade estudasse sobre a evolução da vida de uma nova forma: a partir do seu modelo biológico mais simples e primitivo – uma célula – foram criados os autômatos celulares: modelos de evolução temporal, ou *sistemas dinâmicos*, simples com capacidade para exibir um comportamento complicado. Esses modelos puderam ser simulados visualmente com a utilização de sistemas finitos.

Os autômatos celulares buscam, majoritariamente, estudar como um sistema de configurações iniciais – nesse caso, células – evolui a partir de um conjunto de regras pré-definidas (como, por exemplo, as regras que comandam o nascimento e o falecimento de cada célula em função das outras). Esses sistemas, apesar de carregarem consigo um conjunto de regras simples, muitas vezes podem reagir de maneira *caótica* – isto é, um sistema cujo comportamento futuro se torna, grosso modo, *imprevisível* – por uma perturbação mínima no seu estado inicial. Na matemática, esse estudo é chamado de *Teoria do Caos* e possui diversas aplicações na áreas da física e biologia.

1.2 John Conway e o Game of Life

Em 1970, o matemático britânico John Horton Conway apresentou seus estudos abrangendo autômatos celulares com a criação de um novo 'jogo' sem jogador chamado *Game of Life* (ou Jogo da Vida). O modelo proposto por Conway descrevia um autômato celular regido por regras que pudessem reproduzir as alterações

e mudanças em um conjunto de seres vivos – imagine um conjunto de células em um tabuleiro em constante alteração. No Game of Life, cada célula segue um conjunto finito e pequeno de regras que determinam se, a cada geração, uma célula viva irá morrer, permanecer viva ou se reproduzir, gerando mais células ao seu redor.

2 Conceitos básicos do Game of Life

Cada célula no Game of Life de Conway possui dois estados possíveis: **viva** (ou ativa) ou **morta** (ou inativa). Determinamos por uma **geração** o resultado do processamento da mudança de todas as células do tabuleiro naquele estado, seguindo um conjunto de regras que será discutido adiante. Na transição entre uma geração e outra, uma célula pode morrer, continuar viva ou nascer!

John Conway escolheu suas regras de maneira cuidadosa e elegante, após uma sessão interminável de experimentos, tais que essas satisfizessem três princípios básicos:

- Não deve haver nenhuma configuração inicial para a qual haja uma prova imediata ou trivial de que a população pode crescer indeterminadamente (de maneira *caótica*).
- Devem haver configurações iniciais que aparentemente cresçam indeterminadamente, de maneira caótica.
- Devem haver configurações iniciais simples que cresçam e mudem por um período de tempo considerável antes de terminar de duas formas:
    1. Sumindo completamente do tabuleiro (através de superpopulação ou distanciamento extremo entre as células).
    2. Estacionando em uma configuração estável que se mantém imutável para sempre ou mude seguindo um regime oscilatório infinitamente.

Em outras palavras, as regras devem induzir o sistema inicial a ter um comportamento futuro imprevisível.

2.1 Sobre a vizinhança de uma célula

Para entender melhor como acontece a transição de estados das células entre gerações, precisamos inicialmente entender o que significa a **vizinhança** de uma célula.

Define-se por **vizinhança** de uma célula o conjunto de células ao redor dela, de acordo com um padrão especificado, a qual ela sofrerá influência futura na transição entre as próximas gerações. Em autômatos celulares, usualmente costumamos utilizar de duas métricas de vizinhança: a de **Moore** e a de **Von Neumann**.

De uma forma mais precisa, podemos diferenciá-las pelo padrão que abrangem:

- No modelo de **Moore**, as células que abrangem sua vizinhança são as células imediatamente ao redor da célula selecionada, em um raio de 1.
- No modelo de **Von Neumann**, as células que abrangem sua vizinhança são as células que formam uma cruz com a célula selecionada no meio, com cada lado da cruz abrangendo duas células.

Observação: Considere que, nas bordas do tabuleiro, **a região de vizinhança se propaga para o lado oposto. Isto é, as bordas do tabuleiro estão interligadas de maneira circular.** Caso a vizinhança de uma célula 'exceda' alguma das bordas do tabuleiro, ela deve ser propagada para o lado oposto ao qual a célula está.

A figura abaixo apresenta uma representação visual de ambas as métricas de vizinhança citadas, onde o ponto cinza representa uma célula viva e os pontos vermelhos representam as células que abrangem sua vizinhança.

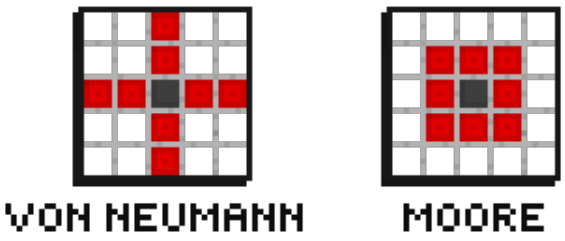

Figura 1: Modelos de vizinhança de Moore e Von Neumann

2.2 Regras básicas sobre a autonomia das células

Cada célula deve seguir um conjunto de quatro regras para determinar seu estado futuro para a próxima geração, sendo essas:

- Qualquer célula viva com **menos de dois vizinhos vivos** morre de solidão.
- Qualquer célula viva com **mais de três vizinhos vivos** morre de superpopulação.
- Qualquer célula morta com **exatamente três vizinhos vivos** se torna uma célula viva (nasce).
- Qualquer célula viva com **dois ou três vizinhos vivos** continua no mesmo estado para a próxima geração.

É importante enfatizar que todos os nascimentos e mortes devem ocorrer simultaneamente. Isto é, na transição entre uma geração e outra, todas as células devem ser processadas de acordo com esse conjunto de regras que determinará seu próximo estado. Uma geração, então, pode ser interpretada como um "instante"na história da vida completa da configuração inicial de células que foi descrita.
Observação: Em todos os exemplos abaixo, estamos considerando o modelo de vizinhança de Moore para a troca de gerações. O mesmo pode ser aplicado para a vizinhança de Von Neumann, mas como previsto o resultado ao longo das gerações será diferente.

Figura 2: Exemplificação do funcionamento do jogo da vida em três gerações.

Na figura anterior, os pontos cinzas escuros representam as células que estão vivas, os pontos cinzas claros representam as células que foram mortas na geração anterior (apenas para meio de visualização) e os pontos brancos representam as células mortas por padrão. Nesse caso a configuração inicial se encontra na geração 1 e perceba que, com essa configuração, o autômato termina com todas as células mortas na geração 3.

De maneira semelhante, podemos utilizar de configurações iniciais que não nos dizem com certeza o futuro do autômato (se ele acabará com todas as células mortas, se ele acabará em um regime oscilatório, etc) para a simulação, como exemplifica a imagem a seguir.

Figura 3: Exemplificação do jogo da vida com uma configuração inicial mais complexa.

As regras criadas por Conway nos garantem que, para o final da simulação, o sistema possui 3 possibilidades: não possuir nenhuma célula viva (como na figura 2), permanecer em um regime oscilatório infinito e constante, ou ficar estático de forma que não nasçam/morram mais células.

3 Modelo de entrega

Interpretando o tabuleiro do Game of Life como uma matriz, você deverá implementar um código que consiga processar geração por geração a partir de uma configuração inicial de células (dada de entrada). Seu código deve ser capaz de processar e simular as regras do Game of Life de acordo com um dos modelos de vizinhança (dado de entrada). Nesse caso, o caractere `'x'` representará uma célula viva e o caractere `'.'` representará uma célula morta.

- Para a entrega no `run.codes`, seu programa deve exibir na tela **apenas a última configuração** do número de gerações requisitada de entrada.
- É válido dizer que o mais interessante desse experimento é vê-lo funcionando visualmente como uma simulação. Lhe ensinaremos como fazer isso para visualizar no seu próprio terminal no final do documento (vide Tópico 8). No entanto, o código que deverá ser enviado ao `run.codes` deve seguir o critério proposto acima. :)

4 Orientações para confecção do código

- Se certifique de utilizar **alocação dinâmica** para armazenamento da matriz em memória e de outros dados de tamanho considerável que você achar conveniente (como *structs*), uma vez que esta poderá vir a ser grande. Para variáveis primitivas e simples, não há necessidade de utilização de alocação dinâmica.

- Ao utilizar alocação dinâmica, é responsabilidade do programador **desalocar todos os blocos de memória que foram alocados manualmente**, não deixando vazamentos. A ferramenta `valgrind` pode te ajudar nessa tarefa.
- **Evite a utilização de memória em excesso.** Aloque somente o necessário para a execução correta do programa.
- **Modularize seu código com a utilização de funções para separar o código em blocos com funcionalidades distintas.** Funções também são muito úteis em casos que um mesmo trecho de código é repetido várias vezes.
- **Atente-se à estruturação e identação do código**. A identação serve como uma maneira de dividir o código em subconjuntos e facilitar sua leitura e manutenção.
- Junto com a modularização, não se esqueça de **documentar as partes mais relevantes do código, incluindo as funções que foram criadas com um cabeçalho acima dela**. Lembre-se, comentários demais também prejudicam a leitura de código!

5 Entrada

Será informado, de entrada, as dimensões m e n (linhas e colunas, respectivamente) da matriz do tabuleiro do Game of Life na mesma linha. Logo após, será dado um número p de gerações do Game of Life que devem ser processadas pelo seu algoritmo e, em outra linha, o tipo v de vizinhança que deverá ser utilizada para o processamento. Por último, será dada uma matriz $M_{m \times n}$ de entrada contendo caracteres `'x'` ou `'.'` em todas as posições dela, representando a configuração inicial do autômato.

- Os valores de m, n e p devem ser inteiros maiores do que zero.
- O valor de v pode ser o caractere `'M'` para o modelo da vizinhança de Moore e o caractere `'N'` para a o modelo de vizinhança de Von Neumann.
- Cada linha da matriz $M_{m \times n}$ será separada por uma quebra de linha (`'\n'`). Não há espaços entre as colunas da matriz.
- Não há necessidade de checar se cada um dos elementos inseridos na matriz é algum dos dois caracteres citados.

6 Saída

A saída do seu programa deverá ser **apenas o tabuleiro do Game of Life na geração p requisitada de entrada**, com o sistema inicial tendo sido modificado consistentemente seguindo as regras do Game of Life. Considere a geração inicial passada como sendo a geração 0. O processamento deve ser feito considerando o modelo de vizinhança escolhido.

- Caso algum dos valores dados de entrada forem inválidos, a mensagem `'Dados de entrada apresentam erro.\n'` deve aparecer na tela e o programa não deve seguir com sua execução.
- Cada linha da matriz deve ser separada por uma quebra de linha (`'\n'`).
- Não deve haver espaços entre os elementos da matriz exibida na tela.

7 Exemplos de Entrada e Saída

- Exemplo 1: Vizinhança de Moore

Entrada

```
7 9
3
M
.........
.........
.........
....xxx..
...xxx...
.........
.........
```

Saída

```
.........
.........
.....x...
...x..x..
...x..x..
....x....
.........
```

- Exemplo 2: Vizinhança de Von Neumann

Entrada

```
9 20
19
N
....................
....................
........x.xxx.......
.....x.xx...........
............x.......
.......xx.x.........
.......xx.x.x.......
....................
....................
```

Saída

```
....................
....................
....................
....................
..........x.........
....................
....................
....................
....................
```

- Exemplo 3: Dados de entrada inválidos

Entrada

```
20 48
-4
N
```

Saída

```
Dados de entrada apresentam erro.
```

8 Simulando o Game of Life

O mais interessante do Game of Life é vê-lo funcionando como uma simulação, o que pode ser feito de forma simples no seu próprio terminal. Para isto, basta imprimir geração por geração no seu terminal conforme elas forem sendo calculadas e, após cada impressão feita, você pode chamar uma função que faça seu programa "dormir"por alguns milissegundos. Dessa forma, você terá uma animação rodando no seu terminal.

Iremos disponibilizar esta função num arquivo separado no run.codes, copie ela para seu código e inclua a biblioteca time.h. Ao chamar a função, basta passar um inteiro com a duração em milissegundos que deseja para cada frame (é sugerido um valor de cerca de 50 para uma animação rápida e 500 para uma lenta).

Se desejar, você também pode limpar a tela do terminal antes de cada nova impressão utilizando o comando `system("clear")`; no Linux ou `system("cls")`; no Windows.

Bom trabalho! :)