

Universidade Federal de Minas Gerais
Matemática Discreta - Trabalho Prático

1. Introdução

Este projeto consiste na geração e visualização de fractais utilizando regras de substituição em sequências de caracteres. O programa é composto por dois arquivos: um responsável pela geração do fractal e outro pela sua visualização utilizando a biblioteca SDL2.

2. Especificação e Restrições

- O programa deve receber como entrada um axioma, um ângulo e uma regra de substituição.
- O programa deve iterar sobre a sequência de caracteres a partir do axioma, substituindo os caracteres de acordo com a regra especificada.
- O número de iterações deve ser no **MÁXIMO** 4.
- O programa deve gerar uma sequência final de caracteres após as n iterações.
- O resultado final deve ser armazenado em um arquivo chamado "fractal.txt".
- O programa está limitado a um máximo de 1.000.000 de caracteres para o axioma e a regra de substituição.
- O programa assume que a entrada do usuário é válida e segue o formato esperado.

3. Projeto

3.1 Arquitetura do Sistema

O sistema é composto por dois programas em linguagem C. O primeiro programa é responsável pela geração do fractal e gera a sequência de caracteres que representa o fractal no arquivo "fractal.txt".

O segundo tem execução OPCIONAL e utiliza a biblioteca SDL2 para ler a sequência de caracteres do arquivo "fractal.txt" e desenhar o fractal em uma janela gráfica.

3.2 Algoritmo de Geração de Fractal

O algoritmo de geração de fractal é implementado no primeiro programa e segue os passos descritos no enunciado do trabalho. Ele recebe como entrada um axioma, um ângulo e uma ou duas regras de substituição, e realiza as substituições necessárias na sequência de caracteres ao longo de n iterações. Ao final das iterações, a sequência de caracteres resultante é armazenada no arquivo "fractal.txt".

3.3 Algoritmo de Desenho do Fractal (EXECUÇÃO OPCIONAL, já que as imagens serão apresentadas nesta documentação, **PODE HAVER PROBLEMAS NA INSTALAÇÃO DA BIBLIOTECA**)

O algoritmo de desenho do fractal é implementado no segundo programa, chamado "fDraw.c". Ele utiliza a biblioteca SDL2 para criar uma janela gráfica e desenhar o fractal. O algoritmo percorre a sequência de caracteres lida do arquivo "fractal.txt" e realiza as seguintes ações para cada caractere:

- Se o caractere for 'F', é desenhada uma linha na direção atual a partir da posição atual.
- Se o caractere for '+', o ângulo de direção é incrementado em um valor fixo.
- Se o caractere for '-', o ângulo de direção é decrementado em um valor fixo.
- Outros caracteres são ignorados.

O resultado é um desenho do fractal na janela gráfica, onde as linhas são desenhadas de acordo com as instruções presentes na sequência de caracteres.

4. Implementação

Ao abordar a implementação dos fractais, existem diversas estratégias possíveis. Neste documento, serão discutidas duas abordagens principais: uma versão iterativa com armazenamento em arquivo e outra versão recursiva. Além disso, será apresentada a estratégia utilizada no projeto.

4.1 Versão Iterativa com Armazenamento em Arquivo

Nessa abordagem, a geração do fractal é realizada de maneira iterativa. Os caracteres de cada estágio intermediário são gravados em um arquivo, lidos e processados para gerar um novo arquivo para o próximo estágio. Esse processo é repetido até que o estágio desejado seja alcançado.

Pontos Positivos:

- Facilidade de implementação e compreensão.
- Permite a visualização do fractal em cada estágio intermediário, auxiliando na análise e depuração.

Pontos Negativos:

- Pode ser menos eficiente em termos de desempenho, devido à leitura e escrita em arquivos durante cada iteração.
- O uso de arquivos intermediários pode ocupar espaço em disco

4.2 Versão Recursiva

Nessa abordagem, a geração do fractal é realizada de forma recursiva. A sequência de caracteres para cada estágio é gerada chamando a função recursivamente. A recursão continua até que o estágio desejado seja alcançado.

Pontos Positivos:

- Implementação mais elegante e concisa, especialmente para fractais complexos.
- Potencialmente mais eficiente em termos de desempenho, evitando o uso de arquivos intermediários.

Pontos Negativos:

- Pode ser mais difícil de compreender e depurar, especialmente para fractais com múltiplas regras de substituição.
- O uso excessivo de recursão pode levar a problemas de estouro de pilha (stack overflow) em fractais extremamente grandes.

4.3 Abordagem Utilizada

A estratégia adotada no projeto segue uma versão iterativa com armazenamento em arquivo, conforme evidenciado no código fornecido. O fractal é gerado por meio de iterações sobre a sequência de caracteres, substituindo os caracteres de acordo com as regras especificadas. O resultado final é armazenado no arquivo "fractal.txt" e, em seguida, visualizado utilizando o programa "fDraw.c" com a biblioteca SDL2.

4.4 Tecnologias Usadas na Implementação

O projeto foi implementado utilizando a linguagem C e as seguintes ferramentas e tecnologias:

- Compilador GCC (GNU Compiler Collection) versão 9.3.
 - Biblioteca SDL2 (Simple DirectMedia Layer) versão 2.0.
-

5. Equações de Recorrência

Serão apresentadas a seguir, cada uma das equações de recorrência usadas para calcular a quantidade de segmentos F gerados e a quantidade de símbolos existentes em cada estágio.

A lógica usada para calcular cada uma delas foi a mesma, foi desenvolvido um código que contava a ocorrência dos caracteres F e os todos os caracteres a cada iteração e a partir disso foi desenvolvido o raciocínio.

5.1 Ilha de Koch

Axioma : F

$$\Theta = \pi / 2$$

$$F \rightarrow F + F - F - FFF + F + F - F$$

Iterações(n)	#F	#Símbolos
0	4	7
1	36	63
2	324	567
3	2916	5103
4	26244	45927

Observando os dados fornecidos, podemos notar que a quantidade de segmentos 'F' na iteração n é igual à quantidade de segmentos 'F' na iteração anterior multiplicada por 9:

$$S(0) = 4$$

$$S(n) = 9 * S(n-1)$$

Além disso, a quantidade total de símbolos na iteração n é igual à quantidade total de símbolos na iteração anterior multiplicada por 9

$$T(0) = 7$$

$$T(n) = 9 * T(n-1)$$

5.2 Preenchimento de espaço de Hilbert

Axioma : X

$X \rightarrow -YF+XFX+FY-$

$$\Theta = \pi / 3$$

$Y \rightarrow +XF-YFY-FX+$

Iterações(n)	#F	Símbolos
0	0	1
1	3	11
2	15	51
3	63	211
4	255	851

Mais uma vez, observando os dados fornecidos, temos que:

Equação de recorrência para a quantidade de segmentos 'F' (S):

$$S(0) = 0$$

$$S(n) = 4 * S(n-1) + 3$$

Equação de recorrência para a quantidade total de símbolos (T):

$$T(0) = 1$$

$$T(n) = (4^n - 1 / 3) * 7$$

5.3 Fractal Criado por mim

Axioma : X

$X \rightarrow YF+XF+Y$

$$\Theta = \pi / 3$$

$Y \rightarrow XF-YF-X$

Iterações(n)	#F	Símbolos
0	0	1
1	2	7
2	8	25
3	26	79
4	80	241

Por fim, observando os dados fornecidos, temos que:

Equação de recorrência para a quantidade de segmentos 'F' (S):

$$S(0) = 0$$

$$S(n) = 3*S(n-1) + 2$$

Equação de recorrência para a quantidade total de símbolos (T):

$$T(0) = 1$$

$$T(n) = 3 \cdot T(n-1) + 4$$

6. Complexidade Assintótica

6.1 Ilha de Koch

A equação de recorrência dada é $T(n) = 9 \cdot T(n-1)$, com $T(0) = 7$.

Podemos resolver essa equação de recorrência para obter uma fórmula fechada para $T(n)$.

$$T(n) = 9 \cdot T(n-1) = 9 \cdot (9 \cdot T(n-2)) = 9^2 \cdot T(n-2) = 9^3 \cdot T(n-3) = \dots = 9^n \cdot T(0) = 9^n \cdot 7$$

Portanto, a solução da equação de recorrência é $T(n) = 7 \cdot 9^n$.

A complexidade do algoritmo representado por essa equação de recorrência é **$O(9^n)$** .

6.2 Preenchimento de espaço de Hilbert

A equação de recorrência dada é $T(n) = ((4^n - 1) / 3) \cdot 7$, com $T(0) = 1$.

A fórmula fechada para $T(n)$ é $T(n) = ((4^n - 1) / 3) \cdot 7$.

Para determinar a complexidade assintótica dessa função, podemos analisar o crescimento da expressão $(4^n - 1) / 3$ à medida que n aumenta.

Quando n aumenta, o termo $(4^n - 1)$ tende a dominar o comportamento da função, uma vez que 4^n cresce exponencialmente. Portanto, podemos considerar apenas esse termo para analisar a complexidade assintótica.

O termo $(4^n - 1)$ cresce exponencialmente em relação a n , e podemos dizer que

sua complexidade é $O(4^n)$.

Assim, a complexidade assintótica da função $T(n) = ((4^n - 1) / 3) * 7$ é $O(4^n)$.

CONSIDERANDO QUE TEMOS OS TERMOS X E Y NA STRING FINAL.

6.3 Fractal criado por mim

Para analisar a complexidade desse algoritmo, vamos observar a relação de recorrência dada:

$$T(n) = 3 * T(n-1) + 4$$

Vamos expandir a relação para alguns valores de n para identificar um padrão:

$$\begin{aligned} T(0) &= 1 \\ T(1) &= 3 * T(0) + 4 = 3 * 1 + 4 = 7 \\ T(2) &= 3 * T(1) + 4 = 3 * 7 + 4 = 25 \\ T(3) &= 3 * T(2) + 4 = 3 * 25 + 4 = 79 \\ T(4) &= 3 * T(3) + 4 = 3 * 79 + 4 = 241 \end{aligned}$$

Podemos observar que cada termo $T(n)$ depende do termo anterior $T(n-1)$ multiplicado por 3 e acrescido de 4. Portanto, podemos reescrever a relação de recorrência como:

$$T(n) = 3^n * T(0) + 4 * (3^{(n-1)} + 3^{(n-2)} + \dots + 3 + 1)$$

A soma dos termos $3^{(n-1)} + 3^{(n-2)} + \dots + 3 + 1$ é uma série geométrica finita com a razão 3 e n termos. Podemos utilizar a fórmula da soma de uma série geométrica para simplificar:

$$3^{(n-1)} + 3^{(n-2)} + \dots + 3 + 1 = (3^n - 1) / (3 - 1) = (3^n - 1) / 2$$

Substituindo essa expressão na relação de recorrência, temos:

$$T(n) = 3^n * T(0) + 4 * (3^n - 1) / 2 = 3^n + 2 * (3^n - 1) = 3^n + 2 * 3^n - 2 = 3 * 3^n - 2$$

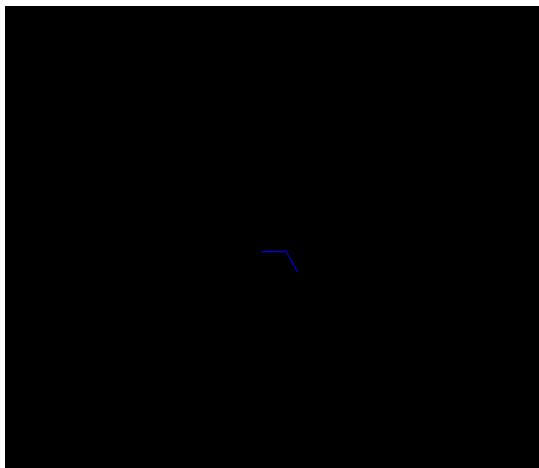
A complexidade assintótica mais precisa possível é, portanto, $O(3^n)$.

CONSIDERANDO QUE TEMOS OS TERMOS X E Y NA STRING FINAL.

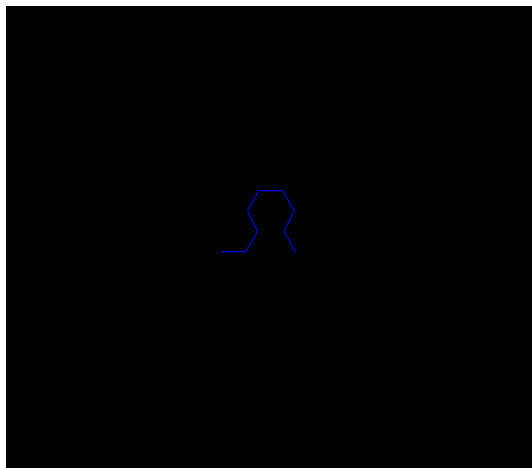
7. Representação Gráfica do Fractal

Para representar graficamente os fractais, eu usei um código desenvolvido por mim em linguagem C com a biblioteca SLD2. Ele pode ser executado, se for de interesse. Para isso, o “readme.txt” especifica como executá-lo;

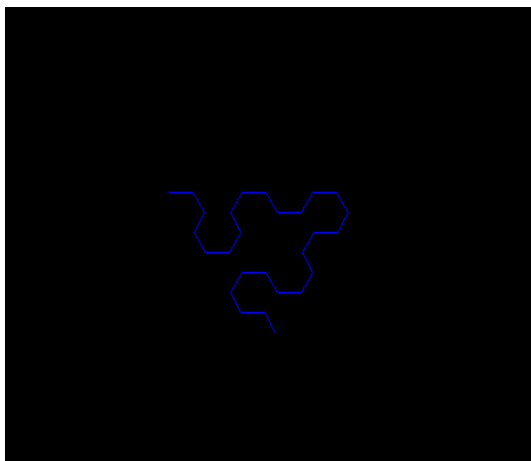
Seguem as imagens do meu fractal em cada estágio.



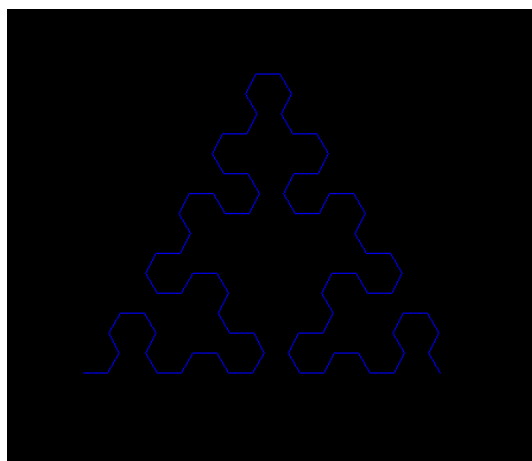
fractal na primeira iteração



fractal na segunda iteração



fractal na terceira iteração



fractal na quarta iteração