Universidade Federal de Minas Gerais

Matemática Discreta - Trabalho Prático

Lucas Cassio Costa

---

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

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

S(0) = 4

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

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

T(0) = 7

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

5.2 Preenchimento de espaço de Hilbert

Axioma : X

X ⟶-YF+XFX+FY-

Θ = π / 2

Y ⟶+XF-YFY-FX+

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

CONSIDERANDO QUE TEMOS OS TERMOS X E Y NA STRING FINAL.
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 ⟶YF+XF+Y

Θ = π / 3

Y ⟶ XF-YF-X

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

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

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*T(n-1) + 4

6. Complexidade Assintótica

Para calcular a complexidade assintótica dos algoritmos a estratégia utilizada será a seguinte:

Será calculado a complexidade a partir da equação de recorrência que define a quantidade de traços, isto é, a notação que define a quantidade de segmentos “F” gerados.

Isso acontece porque o tempo de execução do algoritmo se baseia, ESSENCIALMENTE, na quantidade de traços que eu farei para desenhar meu eventual fractal. Isso implica que será usado SEMPRE a notação **Θ.**

6.1 Ilha de Koch

A equação de recorrência dada é $T(n) = 9 * T(n-1)$, com $T(0) = 4$, para todos os traços “F”.

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

$T(n) = 9 * T(n-1) = 9 * (9 * T(n-2)) = 9^2 * T(n-2) = 9^3 * T(n-3) = ... = 9^n * T(0) = 9^n * 4$

Portanto, a solução da equação de recorrência é $T(n) = 4 * 9^n$.

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

6.2 Preenchimento de espaço de Hilbert

A equação de recorrência dada é: $S(n) = 4 * S(n-1) + 3$, para todos os traços “F”.

Podemos resolver essa equação de recorrência usando o método de substituição. Vamos substituir n por n-1, n-2, n-3, etc., para obter uma ideia do padrão:

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

$= 4 * (4 * S(n-2) + 3) + 3$

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

$= ...$

Podemos simplificar essa equação expandindo-a:

$S(n) = 4^k * S(n-k) + 3 * (4^{(k-1)}) + 3 * (4^{(k-2)}) + ... + 3$

Agora, vamos considerar o caso base, onde $n - k = 0$. Isso ocorre quando $k = n$, o que significa que k é igual a n. Portanto, substituindo k por n na equação acima, temos:

$S(n) = 4^n * S(0) + 3 * (4^{(n-1)}) + 3 * (4^{(n-2)}) + ... + 3$

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

$= 3 * (4^{(n-1)}) + 3 * (4^{(n-2)}) + ... + 3$

Agora, vamos analisar o termo $3 * (4^{(n-1)}) + 3 * (4^{(n-2)}) + ... + 3$. Podemos simplificar esse termo fatorando 3:

$3 * (4^{(n-1)}) + 3 * (4^{(n-2)}) + ... + 3$

$= 3 * (4^{(n-1)} + 4^{(n-2)} + ... + 1)$

Podemos reescrever essa soma como uma série geométrica:

$4^{(n-1)} + 4^{(n-2)} + ... + 1 = (4^n - 1) / (4 - 1)$

Portanto, podemos substituir essa soma de volta na equação para obter:

$S(n) = 3 * (4^{(n-1)} + 4^{(n-2)} + ... + 1)$

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

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

$= 4^n - 1$

Agora, podemos analisar o custo assintótico do algoritmo. Como a fórmula final é $S(n) = 4^n - 1$, o custo assintótico é dominado pelo termo 4^n. Portanto, o custo assintótico desse algoritmo é **$\Theta(4^n)$.**

6.3 Fractal criado por mim

A equação de recorrência dada é: $S(n) = 3 * S(n-1) + 2$, para todos os traços "F".

Podemos resolver essa equação de recorrência usando o método de substituição. Vamos substituir n por n-1, n-2, n-3, etc., para obter uma ideia do padrão:

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

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

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

$= ...$

Podemos simplificar essa equação expandindo-a:

$S(n) = 3^k * S(n-k) + 2 * (3^{(k-1)}) + 2 * (3^{(k-2)}) + ... + 2$

Agora, vamos considerar o caso base, onde n - k = 0. Isso ocorre quando k = n, o que significa que k é igual a n. Portanto, substituindo k por n na equação acima, temos:

$S(n) = 3^n * S(0) + 2 * (3^{(n-1)}) + 2 * (3^{(n-2)}) + ... + 2$

$= 3^n * 0 + 2 * (3^{(n-1)}) + 2 * (3^{(n-2)}) + ... + 2$

$= 2 * (3^{(n-1)}) + 2 * (3^{(n-2)}) + ... + 2$

Agora, vamos analisar o termo 2 * (3^(n-1)) + 2 * (3^(n-2)) + ... + 2. Podemos simplificar esse termo fatorando 2:

$2 * (3^{(n-1)}) + 2 * (3^{(n-2)}) + ... + 2$

$= 2 * (3^{(n-1)} + 3^{(n-2)} + ... + 1)$

Podemos reescrever essa soma como uma série geométrica:

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

Portanto, podemos substituir essa soma de volta na equação para obter:

$S(n) = 2 * (3^{(n-1)} + 3^{(n-2)} + ... + 1)$

= 2 * ((3^n - 1) / (3 - 1))

= 2 * ((3^n - 1) / 2)

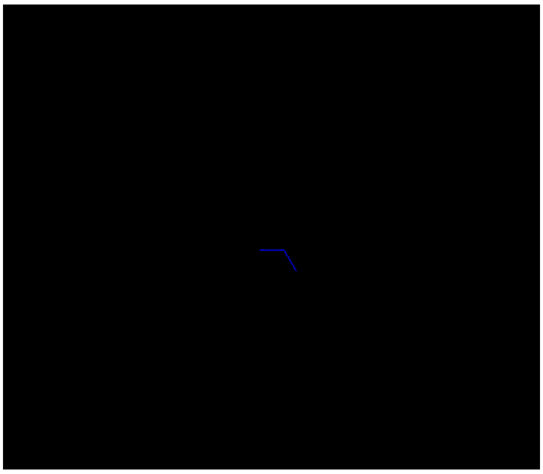
= 3^n - 1

Agora, podemos analisar a complexidade do algoritmo. Como a fórmula final é S(n) = 3^n - 1, a complexidade é dominada pelo termo 3^n. Portanto, a complexidade desse algoritmo é **Θ(3^n).**

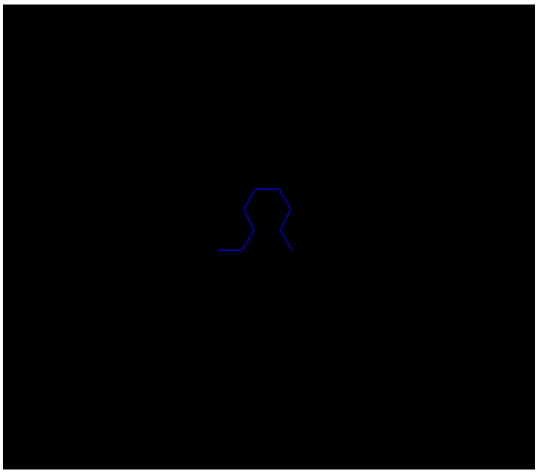
7. Representação Gráfica do Fractal

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” específica como executá-lo;

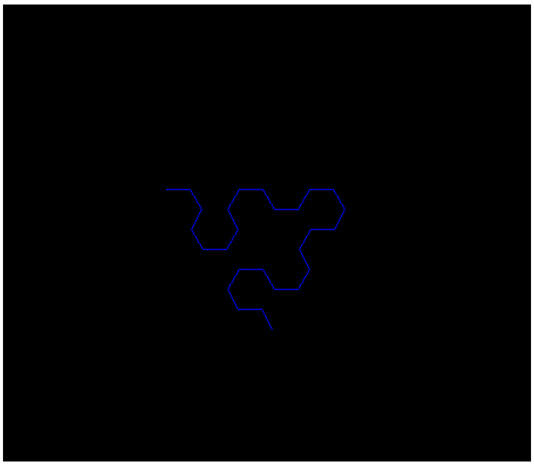
Seguem as imagens do meu fractal em cada estágio:

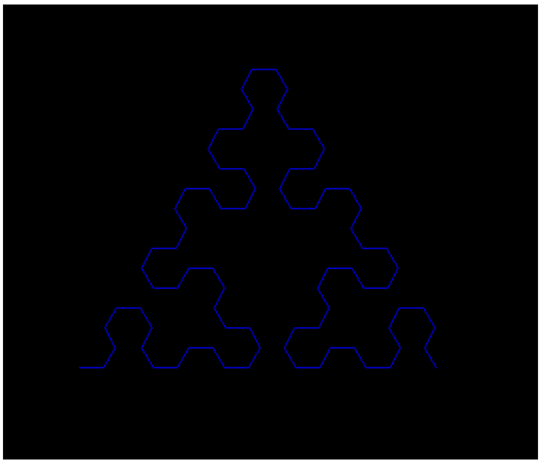


Fractal na primeira iteração



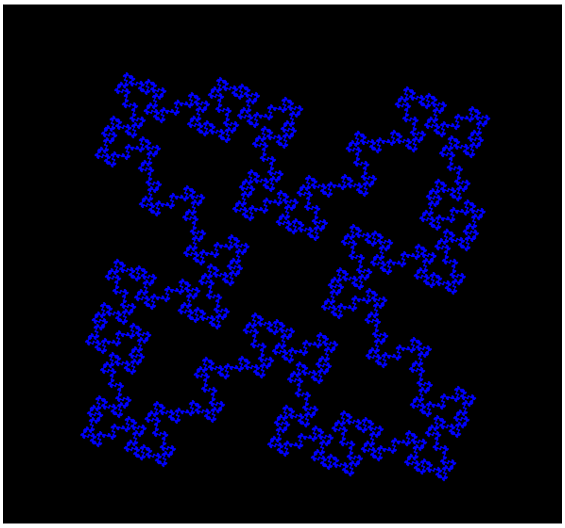
Fractal na segunda iteração



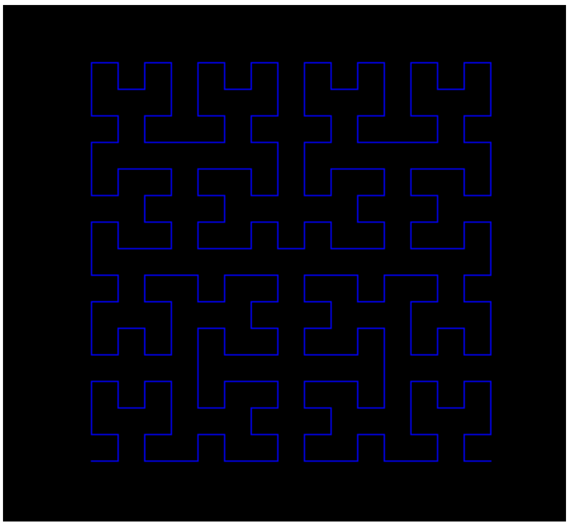


Fractal na terceira iteração ***Fractal na quarta iteração***

A seguir colocarei a imagem do quarto estágio dos outros fractais, apenas a título de curiosidade:



Quarto estágio do Fractal Ilha de Koch.



Quarto estágio do Fractal Preenchimento de Espaço de Hilbert.