Aluna: Carla Beatriz Ferreira (2022097470)
Turma: Matemática Discreta - TW
Professor: Antonio Alfredo Ferreira Loureiro

UFMG
UNIVERSIDADE FEDERAL DE MINAS GERAIS

TRABALHO PRÁTICO - DOCUMENTAÇÃO

1. INTRODUÇÃO

Um fractal é uma forma ou padrão geométrico que exibe auto-semelhança em diferentes escalas ou ampliações, são muitas vezes formas complexas e irregulares que são criadas através de processos iterativos ou recursivos, o que significa que a mesma fórmula matemática ou algoritmo é aplicado repetidamente para gerar a forma. O problema proposto neste trabalho prático foi o desenvolvimento de programas que gerem os 4 primeiros estágios de 3 diferentes fractais, além da análise de sua fórmula matemática e de seu custo Θ.

2. FASES DO PROJETO E IMPLEMENTAÇÃO

2.1 - Fase 1: Entendimento do problema e sua especificação

A primeira parte do problema se baseia em entender o que foi especificado e escolher a melhor estratégia para a implementação.

Na “PARTE 1 - Fazer três programas, um para cada fractal como indicado [...]” foi pré determinado quais fractais seriam feitos por cada aluno, baseando-se no número de matrícula. Considerando o número da minha matrícula, 2022097470, os programas a serem feitos devem ser:

(i) Floco de neve onda senoidal 1 de von Koch (Σ no matrícula mod 4 = 1)

(ii) Preenchimento de espaço de Hilbert (n° de matrícula par)

e

(ii) Um fractal definido por você que gere uma cadeia de polígonos simples que tenha pelo menos duas regras como as curvas de preenchimento de espaço de Peano e Hilbert.

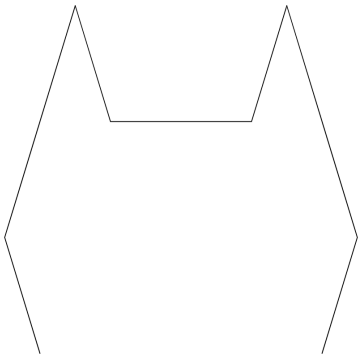
O Fractal escolhido por mim foi pensando em um floco de neve que parece um gatinho ou o Batman. O processo por trás da definição do meu próprio fractal foi:

1 - Entendimento de como era formado e desenhado os fractais de Peano e Hilbert exemplificados;

2 - Estudo de suas semelhanças e simetria, percebendo que a regra de Y era exatamente como a regra de X, substituindo ‘X’ por ‘Y’ e ‘-’ por ‘+’ (e vice versa);

3 - Desenho físico em papel de vários projetos de gatos, estudando os ângulos possíveis (principalmente 60° e 90°) e escolha de 60° por sua maior semelhança com um gato e dada a impossibilidade de utilizar dois ângulos diferentes;

4 - Definição da regra simples, sem X ou Y, apenas com F e os símbolos para que forme um gato no primeiro estágio (F = --F+FF++F-FF-F++FF+F--), como na imagem abaixo;



5 - Utilização do site/software Online Math Tools (https://onlinemathtools.com/l-system-generator) para melhor visualização das possibilidades rotacionando o gato nas próximas iterações ou criando uma simetria diferente;

6 - Escolha de X = --YF+FF++F-FF-F++FF+F-- e

Y = ++XF-FF--F+FF+F--FF-F+ (com Y com um '+' a menos na simetria para que as linhas traçadas não se cruzem e para que se feche como uma figura geométrica fechada com mais estágios)

7 - Implementação do programa em C e análise da quantidade de símbolos e caracteres para a equação de recorrência, tal qual nos outros fractais.

2.2 - Fase 2: Discussão de estratégias e implementação

Na "PARTE 2 - Discuta essas duas estratégias (iterativa ou recursiva) e outras que você considerou, analisando pontos positivos e negativos de cada uma." é sugerido duas formas de se implementar o programa e solicitado uma discussão.

A versão iterativa, gerando um "arquivo 1" para o primeiro estágio e usando este para gerar o próximo num "arquivo 2" com o segundo estágio, que seria renomeado e utilizado para gerar o próximo de forma sequencial me pareceu a forma mais interessante dado que temos um número já pré definido de estágios que se é esperado obter e foi a mais discutida em sala na aula referente ao trabalho

prático. Além disso, mesmo que não tenha sido pedido os estágios intermediários, eu escolhi mantê-los para uma melhor visualização da aplicação e pela facilidade em se contar a quantidade de símbolos e caracteres no final. Escolhi também não utilizar só dois arquivos (ex: “final” e “auxiliar”) pelo mesmo motivo anterior, além de não precisar renomear nenhum dos meus arquivos ou mudar o arquivo para qual o ponteiro está apontando dentro do programa.

Ao considerar a versão recursiva, considerei que a mesma não me traria tanta clareza dos estágios intermediários, o que me pareceu fundamental para estudo da equação de recorrência, portanto, decidi por não utilizá-la. Para que ela me trouxesse essas informações que estava buscando, teria que salvar os estágios intermediários chamados na função recursiva em outro arquivo, o que seria mais complicado de implementar (na minha opinião) e usaria a mesma lógica da maneira iterativa, sem mais benefícios.

Outra opção também era não utilizar arquivos, mas sim apenas uma variável char com tamanho maior, mas essa opção foi desconsiderada devido ao tamanho final do quarto estágio que teria que ser definido como muito grande no início ou com alocação dinâmica durante o programa, trabalhando apenas na memória primária, não apresentando também os estágios intermediários que escolhi manter para melhor avaliação.

Ao escrever o código em C, decidi fazer vários arquivos (um para cada estágio) utilizando a memória secundária e demorando mais, mas desconsiderando este tempo pois seriam apenas 4 estágios.

Dado que a forma de entrada não estava clara até as últimas semanas antes da data original de entrega do trabalho e que foi discutido em sala e com a monitora que cada arquivo seria para apenas um exercício, não faria sentido cada main.c pedir entradas sendo que só aceitava uma entrada específica, portanto as variáveis foram definidas no próprio programa para evitar erros ao executar o programa.

As informações de compilação e execução estão apresentadas no leiame.txt junto aos programas, mas como o Trabalho prático foi desenvolvido e testado utilizando unicamente máquinas com o sistema operacional Windows 11, forneço também o link para o replit, caso queira executar o código online para o melhor resultado ("/t" diferentes do código enviado).

Cada um dos 3 arquivos .c utiliza funções que estão no próprio arquivo, sem a necessidade de uma biblioteca local (arquivo .h) extra e sem a adição de nenhum arquivo na linha de comando para compilação. Não foi acrescentado nenhum arquivo Makefile dado que é apenas um main.c para cada.

Para compilar digite no diretório correto:

gcc main.c -o main

e execute com:

./main

Links no replit:

(i) https://replit.com/join/nlvcxetjmb-carlab5

(ii) https://replit.com/join/shrbuibioe-carlab5

(iii) https://replit.com/join/thdkbbkufz-carlab5

2.3 - Fase 3: Equação de recorrência e complexidade

Na “PARTE 3 - Para cada um dos fractais implementados, apresente a equação de recorrência para calcular a quantidade de segmentos F gerados e a quantidade de símbolos existentes em cada estágio” é solicitada a equação de recorrência e aproveitando o programa em c já desenvolvido foi utilizado as saídas geradas no arquivo de texto como parte fundamental para a expansão e dedução da equação de recorrência.

Interpretando que são duas equações solicitadas, uma para a quantidade de segmentos F e uma para a quantidade de símbolos, as equações encontradas foram:

(i) Floco de neve onda senoidal 1 de von Koch:

Utilizando

```
34    //Entradas
35    char axioma = 'F';
36    int angulo = 60; //não utilizado para aplicar a regra
37    char regra[] = "F++FF--F+F"; //char regra[11]
```

Obtemos:

n	#F	#Símbolos
1	5	10
2	25	55
3	125	280
4	625	1405

e portanto podemos perceber as equações

$$\#F = 5^n$$

$$\#Símbolos = (2 \times 5^n) + \sum_{i=1}^{n-1} 5^i$$

(ii) Preenchimento de espaço de Hilbert:

Utilizando

```
//Entradas
//char axioma = 'X';
//int angulo = 90; //não utilizado para aplicar a regra
char Regra_X[] = "-YF+XFX+FY-"; //char Regra_X[12]
char Regra_Y[] = "+XF-YFY-FX+";
char nome_regra1 = 'X';
char nome_regra2 = 'Y';
```

Obtemos

n	#F	#Símbolos	#X	#Y
1	3	7	2	2
2	15	35	8	8
3	63	147	32	32
4	255	595	128	128

e portanto podemos perceber as equações

$$\#F = 2^{2n} - 1$$

$$\#Símbolos = (2^{2n} - 1) + \sum_{i=1}^{n} 2^{2i}$$

(ii) Floco de neve que parece um gato/batman:

Utilizando

```
//Entradas
//char axioma = 'X';
//int angulo = 60; //não utilizado para aplicar a regra
char Regra_X[] = "--YF+FF++F-FF-F++FF+F--"; //char Regra_X[12]
char Regra_Y[] = "++XF-FF--F+FF+F--FF-F+";
char nome_regra1 = 'X';
char nome_regra2 = 'Y';
```

Obtemos

n	#F	#Símbolos	#X	#Y	#+	#-
1	10	22	0	1	6	6
2	20	43	1	0	11	12
3	30	65	0	1	17	18
4	40	86	1	0	22	24
5	50	108	0	1	28	30
6	60	129	1	0	33	36
7	70	151	0	1	39	42

e portanto podemos perceber as equações

$\#F = n \times 10$

$\#Símbolos = n \times 20 + 2n - \lfloor (n \div 2) \rfloor$

Para a “PARTE 4 - Apresente a complexidade de seus algoritmos considerando a notação assintótica mais precisa possível.” foi discutido em aulas e monitorias que a complexidade é em relação a equação de recorrência e não para realmente analisar o custo do código, algo que vai além da equação de recorrência.

Para ser a notação assintótica mais precisa, utilizaremos a notação teta (Θ), dado que a quantidade de F desenhados sera EXATAMENTE essa quantidade, não sendo ATÉ essa valor (como um limite superior) ou semelhante, não sendo adequado usar as notações O (“O” grande), Ω (ômega grande), o (“o” pequeno) ou ω (ômega pequeno). Para o valor deste custo, foi observado somente a equação encontrada para a quantidade de traços (F) desenhados, considerando que a mudança de ângulo (símbolos ‘+’ e ‘-’) é um tempo irrelevante para a real construção do fractal e seu desenho.

O custo encontrado está apresentado abaixo:

(i) Floco de neve onda senoidal 1 de von Koch:

Equação de recorrência: $\#F = 5^n$

Custo: $\Theta(5^n)$

(ii) Preenchimento de espaço de Hilbert:

Equação de recorrência: $\#F = 2^{2n} - 1$

Custo: $\Theta(2^{2n})$, dado que a notação Θ limita a função f(n) por fatores constantes. Escreve-se f(n) = Θ(g(n)), se existirem constantes positivas c1, c2 e n0 tais que para n ≥ n0, o valor de f(n) está sempre entre c1g(n) e c2g(n) inclusive.

Utilizando a definição apresentada no material,

➜ Pode-se dizer que $g(n)$ é um limite assintótico firme (em inglês, *asymptotically tight bound*) para $f(n)$.

$$f(n) = \Theta(g(n)),$$
$$\exists\, c_1 > 0, c_2 > 0, n_0, \mid 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0$$

(ii) Floco de neve que parece um gato/batman:

Equação de recorrência: $\#F = 10n$

Custo: $\Theta(10n)$

2.4 - Fase 4: Busca de maneiras para desenhar os fractais

Na “PARTE 5 - Investigue as opções de software (preferencialmente grátis) para desenhar os fractais gerados e discuta brevemente aqui. Apresente uma figura com os primeiros quatro estágios de pelo menos o fractal que você propôs na letra (iii) acima.” é solicitado essa pesquisa, permitindo a troca de informações e recomendações com outros alunos.

Dentre as diversas opções de software disponíveis para desenhar os fractais duas se destacaram, sendo elas o site “onlinemathtools” e uma biblioteca sugerida

pelo colega de sala Eduardo Correia, que além de compartilhar essa biblioteca com a turma, se dispôs a criar um repositório no github onde explicou o seu funcionamento e como implementá-la. Por questões de praticidade, a escolha final foi o site previamente citado.

O “onlinemathtools” é uma página na internet que fornece diversas ferramentas matemáticas para os mais diversos usos, dentre elas a geração de imagens de fractais.

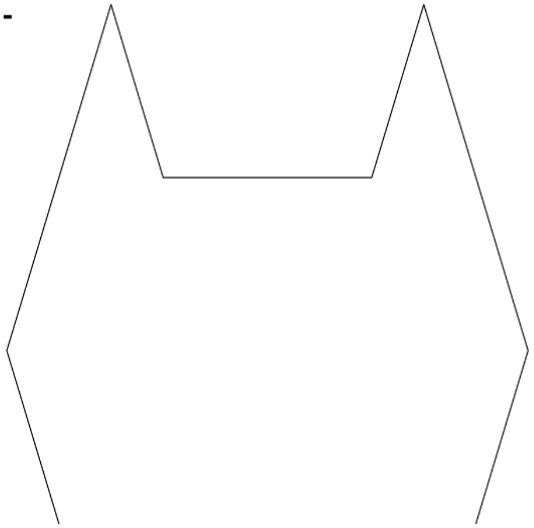
Seu funcionamento é extremamente simples e prático, sendo necessário fornecer a ele somente as seguintes informações:

- Os símbolos a serem desenhados;
- O axioma inicial;
- Quais as regras do fractal;
- Quantas iterações deseja que sejam feitas;
- O ângulo de rotação.

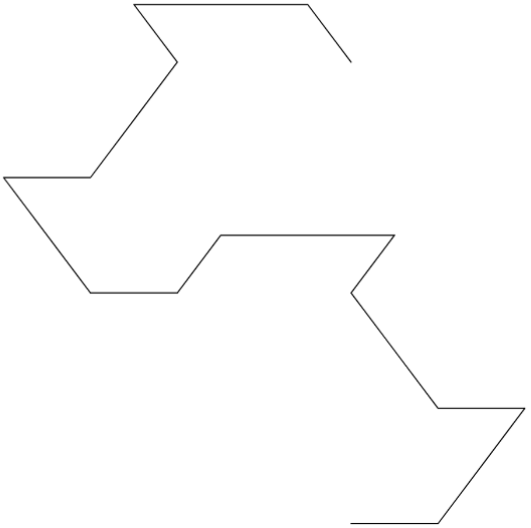
Além disso, o site apresenta algumas outras funções de personalização como dimensionamento do desenho, espessura das retas e cores do desenho e possui uma interface de fácil utilização.

Os primeiros estágios do fractal criado por mim (gato/batman) estão apresentados na figura a seguir, além do link para o site já configurado.

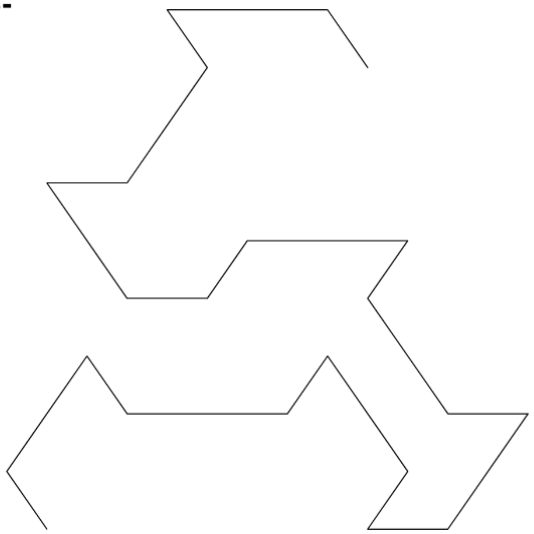
1-



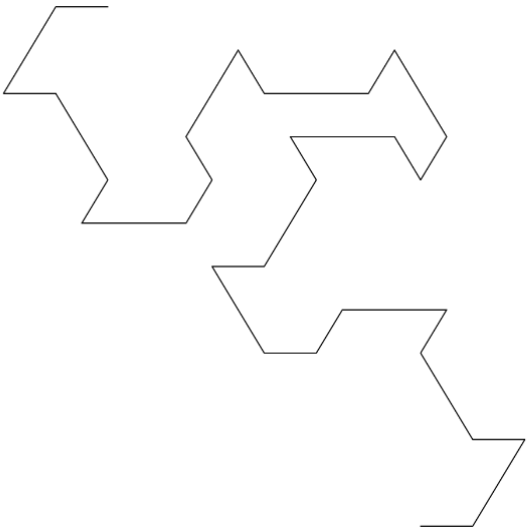
2-



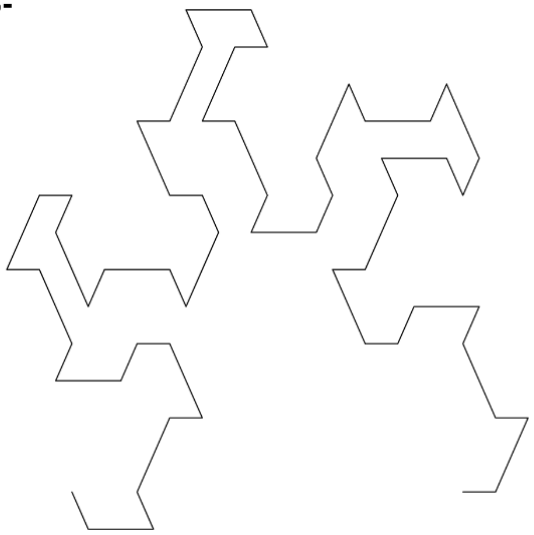
3-



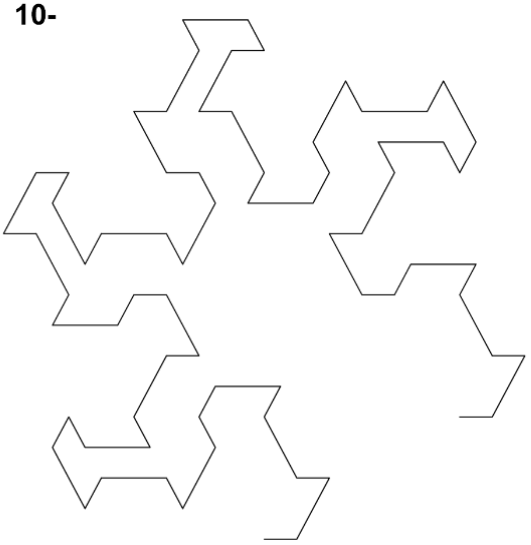
4-



8-



10-



Link para configuração com 4 estágios já definida:
https://onlinemathtools.com/l-system-generator?draw=F&skip=&axiom=X&rule-1=X%20%3D%20--YF%2BFF%2B%2BF-FF-F%2B%2BFF%2BF--&rule-2=Y%20%3D%20%2B%2BXF-FF--F%2BFF%2BF--FF-F%2B&rule-3=&rule-4=&rule-5=&width=800&height=800&iterations=4&angle=60&direction=right&line-width=2&padding=5&background-color=rgb(255%2C%20255%2C%20255)&line-segment-color=rgb(0%2C%200%2C%200)

DISCUSSÃO E CONCLUSÃO:

A solução implementada, apesar de possuir alto custo operacional para compilar e executar o programa, permite ao usuário deduzir fórmulas e analisar a quantidade de segmentos e de símbolos da sequência gerada em diferentes estágios do fractal, para se certificar de que a resposta final selecionada é válida para o problema. Com isso, temos um gasto de tempo que foi analisado para a escolha da estratégia, mas mais informações no arquivo final.

Além disso, a parte mais desafiadora e desconhecida deste trabalho prático e que exigiu mais pesquisa foi a busca por opções de software para desenhar os fractais gerados, mas que ao final se tornou uma aliada na conferência da sequência gerada pelo programa em C, ao ser possível verificar visualmente o que é esperado.