Universidade Federal de Minas Gerais
UFMG

MATEMÁTICA DISCRETA

TRABALHO PRÁTICO

Geração de Fractais

Professor:

Antônio Loureiro

Monitor(a):

Cleiton Neves Santos

Aluno:

Gustavo Luiz Araújo Ribeiro

Belo Horizonte, Julho de 2024

Conteúdo

1 1. Código dos Fractais

1.1 Floco de neve onda quadrada de von Koch

```c
#include <stdio.h>
#include <stdlib.h>

#define AXIOMA 'F'
#define REGRAS "F-F+F+FF-F-F+F"

void firstFractal(int iterations){

    char* arquivo_atual = "i.txt";
    char* arquivo_temp = "ti.txt";

    FILE* file = fopen(arquivo_atual, "w");
    if (!file) {
        exit(EXIT_FAILURE);
    }
    fprintf(file, "%c", AXIOMA);
    fclose(file);
```

Figura 1: Importação de bibliotecas, definição de constantes e escrita do axioma no arquivo de leitura.

A primeira parte do código consiste em declarar as bibliotecas utilizadas, nesse caso as padrões do C, e a definição de constantes para o primeiro fractal, que são o axioma e as regras. Em seguida, a função é definida recebendo um número inteiro como parâmetro, que representa o número de iterações que produzirá a string ao final, e retornando vazio, pois não retorna nada. A primeira coisa que é feita dentro da função é criar dinâmicamente dois vetores do tipo char, que guardam o nome do arquivo *i.txt*, onde a string final será armazenada, e *ti.txt*, que é um arquivo temporário auxiliar que será excluido no final. Em seguida, o arquivo *i.txt* é aberto para escrita, e o axioma nele é escrito.

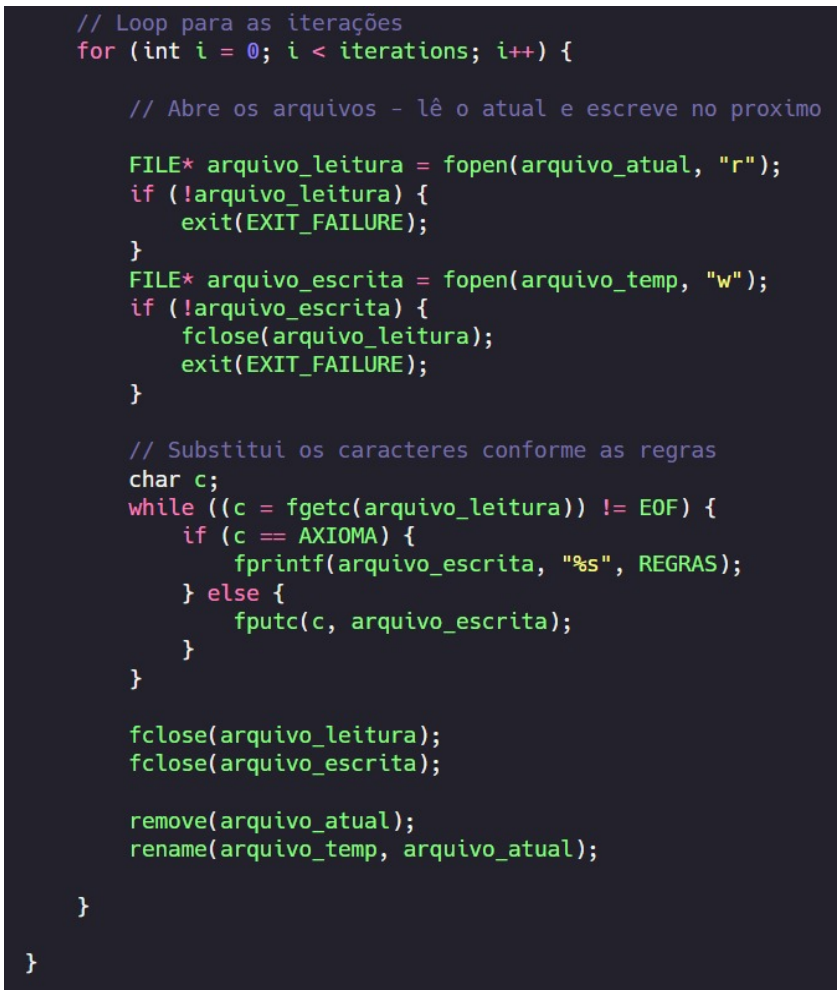

Figura 2: Leitura e escrita da string nos arquivos dentro de um laço de repetição.

A próxima etapa do código consiste em entrar em um loop *for*, que se repetirá um número n de vezes correspondente ao valor lido na entrada e passado como parâmetro da função. Em cada iteração o arquivo contendo a string referente ao axioma (para n = 0), ou referente a string da iteração passada (para n >0) é aberto para leitura, e o arquivo auxiliar é aberto para escrita. Uma variável auxiliar do tipo char é declarada, e dentro de um loop while a ela é atribuido cada um dos caracteres presentes no arquivo de leitura. Caso a variável auxiliar, que receberá como valor todos os caracteres do arquivo de leitura, seja igual ao axioma, no arquivo de escrita é escrito a regra. Caso contrário, o símbolo, que será um + ou um -, é copiado no arquivo de escrita. Por fim todos os arquivos são devidamente fechados, o arquivo atual é excluido e o arquivo temporário tem seu nome trocado para *i.txt*, que é o nome padrão de saída para o primeiro fractal.

1.2 Preenchimento de espaço de Hilbert

```c
#define AXIOMA2 'X'
#define REGRAS_X "-YF+XFX+FY-"
#define REGRAS_Y "+XF-YFY-FX+"

void secondFractal(int iterations) {

    char* arquivo_atual_2 = "ii.txt";
    char* arquivo_temp_2 = "tii.txt";

    // Grava o axioma inicial em arquivo_atual_2
    FILE* file = fopen(arquivo_atual_2, "w");
    if (!file) {
        exit(EXIT_FAILURE);
    }
    fprintf(file, "%c", AXIOMA2);
    fclose(file);
```

Figura 3: Definição de constantes e escrita do axioma no arquivo de leitura.

A primeira parte do código é idêntica a primeira parte do código anterior, com exceção da importação das bibliotecas. As constantes do fractal de preenchimento de Hilbert são declaradas e o axioma é escrito no arquivo de texto *ii.txt*.

```c
    // Loop para as iterações
    for (int i = 0; i < iterations; i++) {

        // Abre os arquivos - lê o atual e escreve no proximo

        FILE* arquivo_leitura_2 = fopen(arquivo_atual_2, "r");
        if (!arquivo_leitura_2) {
            exit(EXIT_FAILURE);
        }
        FILE* arquivo_escrita_2 = fopen(arquivo_temp_2, "w");
        if (!arquivo_escrita_2) {
            fclose(arquivo_leitura_2);
            exit(EXIT_FAILURE);
        }

        // Substitui os caracteres conforme as regras
        char c;
        while ((c = fgetc(arquivo_leitura_2)) != EOF) {
            if (c == 'X') {
                fprintf(arquivo_escrita_2, "%s", REGRAS_X);
            }
            else if (c == 'Y') {
                fprintf(arquivo_escrita_2, "%s", REGRAS_Y);
            }
             else {
                fputc(c, arquivo_escrita_2);
            }
        }

        fclose(arquivo_leitura_2);
        fclose(arquivo_escrita_2);

        remove(arquivo_atual_2);
        rename(arquivo_temp_2, arquivo_atual_2);

    }
```

Figura 4: Leitura e escrita da string nos arquivos dentro de um laço de repetição.

A segunda parte também é semelhante a do primeiro fractal. Dentro de um loop *for*, que se repete quantas vezes forem passadas na entrada, um dos arquivos é lido caractere a caractere a cada repetição, e o caractere lido é atribuido a uma variável char. Em seguida é feita uma comparação que é ligeiramente diferente do primeiro fractal. Caso a variável seja igual a X, a regra X é escrita no segundo arquivo de texto, caso a variável seja igual a Y, a regra Y é escrita no segundo arquivo de texto, e caso a variável seja igual a um símbolo, o símbolo é escrito. Por fim, todos os arquivos são fechados, o arquivo que foi lido é excluido e o arquivo que foi escrito é renomeado para *ii.txt*

```
    // Remoção de caracteres X e Y no arquivo final
    FILE* arquivo_leitura_final = fopen(arquivo_atual_2, "r");
    if (!arquivo_leitura_final) {
        exit(EXIT_FAILURE);
    }
    FILE* arquivo_escrita_final = fopen(arquivo_temp_2, "w");
    if (!arquivo_escrita_final) {
        fclose(arquivo_leitura_final);
        exit(EXIT_FAILURE);
    }

// Filtra os caracteres X e Y
    char c;
    while ((c = fgetc(arquivo_leitura_final)) != EOF) {
        if (c != 'X' && c != 'Y') {
            fputc(c, arquivo_escrita_final);
        }
    }

    fclose(arquivo_leitura_final);
    fclose(arquivo_escrita_final);

    // Substitui o arquivo original pelo filtrado
    remove(arquivo_atual_2);
    rename(arquivo_temp_2, arquivo_atual_2);
}
```

Figura 5: Filtro dos caracteres X e Y.

A última etapa consiste em abrir os dois arquivos novamente, o de leitura *ii.txt* e o auxiliar onde será escrito a string final *tii.txt*. Assim como feito anteriormente, dentro de um loop *While* o primeiro arquivo é lido caractere a caractere, e um a um eles são atribuidos a uma variável auxiliar do tipo char. Como queremos apenas remover todos os X e Y, verificamos se a variável é igual a eles, e caso contrário copiamos o caractere para o arquivo final, que ainda consta com nome de *tii.txt*. Por fim, fechamos todos os arquivos, deletamos o arquivo que foi lido e trocamos o nome do arquivo contendo a string final de *tii.txt* para *ii.txt*.

1.3 Fractal de Sierpiński arrowhead curve L-system

```c
#define AXIOMA3 'X'
#define REGRAS_X3 "Y-X-Y"
#define REGRAS_Y3 "X+Y+X"

void thirdFractal(int iterations) {

    char* arquivo_atual_3 = "iii.txt";
    char* arquivo_temp_3 = "tiii.txt";

    // Grava o axioma inicial em arquivo_atual
    FILE* file = fopen(arquivo_atual_3, "w");
    if (!file) {
        exit(EXIT_FAILURE);
    }
    fprintf(file, "%c", AXIOMA3);
    fclose(file);
```

Figura 6: Definição de constantes e escrita do axioma no arquivo de leitura.

```c
    // Loop para as iterações
    for (int i = 0; i < iterations; i++) {

        // Abre os arquivos - lê o atual e escreve no proximo

        FILE* arquivo_leitura_3 = fopen(arquivo_atual_3, "r");
        if (!arquivo_leitura_3) {
            exit(EXIT_FAILURE);
        }
        FILE* arquivo_escrita_3 = fopen(arquivo_temp_3, "w");
        if (!arquivo_escrita_3) {
            fclose(arquivo_leitura_3);
            exit(EXIT_FAILURE);
        }

        // Substitui os caracteres conforme as regras
        char c;
        while ((c = fgetc(arquivo_leitura_3)) != EOF) {
            if (c == 'X') {
                fprintf(arquivo_escrita_3, "%s", REGRAS_X3);
            }
            else if (c == 'Y') {
                fprintf(arquivo_escrita_3, "%s", REGRAS_Y3);
            }
             else {
                fputc(c, arquivo_escrita_3);
            }
        }

        fclose(arquivo_leitura_3);
        fclose(arquivo_escrita_3);

        remove(arquivo_atual_3);
        rename(arquivo_temp_3, arquivo_atual_3);

    }
```

Figura 7: Leitura e escrita da string nos arquivos dentro de um laço de repetição.

As duas primeiras etapas da geração do Fractal de Sierpiński arrowhead curve L-system é idêntica a do Fractal de preenchimento do espaço de Hilbert, em que um arquivo é lido caractere a caractere, e baseado em qual caractere é lido o outro arquivo de texto vai se preenchendo. A diferença entre os dois fractais é o passo final.

```
    // Remoção de caracteres X e Y no arquivo final
    FILE* arquivo_leitura_final = fopen(arquivo_atual_3, "r");
    if (!arquivo_leitura_final) {
        exit(EXIT_FAILURE);
    }
    FILE* arquivo_escrita_final = fopen(arquivo_temp_3, "w");
    if (!arquivo_escrita_final) {
        fclose(arquivo_leitura_final);
        exit(EXIT_FAILURE);
    }

// Transforma A e B em F
    char c;
    while ((c = fgetc(arquivo_leitura_final)) != EOF) {
        if (c == 'X' || c == 'Y') {
            fputc('F', arquivo_escrita_final);
        }
        else {fputc(c, arquivo_escrita_final);}
    }

    fclose(arquivo_leitura_final);
    fclose(arquivo_escrita_final);

    // Substitui o arquivo original pelo filtrado
    remove(arquivo_atual_3);
    rename(arquivo_temp_3, arquivo_atual_3);
}
```

Figura 8: Conversão de caracteres X e Y para F.

Ao final dos dois primeiros processos obtemos um arquivo *iii.txt* contendo caracteres X,Y e F, além dos símbolos. Para gerar os fractais desejados, precisamos trocar todos os caracteres X e Y por caracteres F. Isso é feito de maneira idêntica a do fractal de Hilbert, lendo os caracteres de um dos arquivos e escrevendo no outro baseado no que foi lido. Dessa vez, caso os caracteres lidos sejam X ou Y, é escrito um F no lugar, e caso contrário todos os caracteres são apenas reescrios no outro arquivo. Assim, ao final de tudo teremos um arquivo *tiii.txt* contendo a string final que gera o fractal desejado. A última coisa feita é deletar o arquivo que foi lido e renomear o arquivo final com o nome *iii.txt*.

2 2. Discussão das estratégias

Existem algumas estratégias para se gerar a sequência de caracteres que formam os fractais. Dentre os possíveis algorítimos me concentrei em dois:

- Método recursivo direto:

Inicialmente ataquei o problema criando uma função que recebia como parâmetro o número de iterações, que seria lido na função main. Dentro dessa função eram declarados dois vetores (um principal e um auxiliar) de tipo char, ou seja, dois ponteiros que apontavam para endereços de memória sequenciais que seriam utilizados para armazenas variáveis do tipo char (pois em C não é possível trabalhar com strings como em C++). A minha função deveria iterar um número n de vezes e a cada iteração os caracteres armazenados em um vetor eram lidos e caso fossem iguais ao axioma eram substituidos pela regra e adicionados ao outro vetor. Para os fractais de preenchimento de espaço o processo era semelhante, embora o código fosse um pouco diferente. Após as n iterações a string final completa deveria estar armazenada em um dos vetores, e deveria ser escrita no arquivo txt correspondente.

O principal problema que encontrei ao tentar fazer o algorítimo dessa maneira foi a alocação de memória. Inicialmente tentei alocar um valor fixo de memória aos vetores, mas após 3 iterações a string já estava maior que a memória alocada. Então resolvi alocar dinâmicamente a memória, calculando através da equação matemática o número exato de caracteres que deveriam existir para n iterações e alocando esse tamanho aos vetores. Entretanto, a quantidade de memória alocada no buffer ao ponteiro estava ficando muito grande, a ponto de ultrapassar os limites do sistema em 5 iterações. Nessa hora percebi que o meu algorítimo não iria funcionar para um grande número de iterações, e resolvi tentar outro método.

- Método iterativo de escrita:

Após o fracasso da outra implementação, pensei em uma forma de contornar o problema da alocação de memória, lendo e escrevendo diretamente no arquivo de texto a cada iteração. Utilizei dois arquivos de texto, um inicialmente vazio e outro inicialmente contendo o axioma. Dentro de um laço de repetição com n iterações, os caracteres do texto contido no arquivo de texto com o axioma eram lidos um a um, e se fossem iguais ao axioma a regra era escrita no outro arquivo de texto, e se fossem diferentes o símbolo era copiado. Essa estratégia me permitiu gerar strings sem limites de tamanho, pois elas não estão sendo armazenadas em um vetor, e sim escritas em um arquivo de texto. Consegui testar para 8 iterações, e obtive uma string com 31.157.686 caracteres.

Comparando os dois métodos percebi que o primeiro era um pouco mais rápido para a produção de strings de até 4 iterações, mas para um número maior de repetições ele se mostrou inferior, o que me fez optar pelo segundo.

3 3. Equações de Recorrência

Podemos encontrar a equação de recorrência de cada um dos fractais e obter o número de caracteres "F"e de símbolos em cada estágio n. Vamos definir a seguinte notação:

- n corresponde ao número de iterações do fractal;
- F(n) corresponde ao número de segmentos (F) para um dado n;
- S(n) corresponde ao número de símbolos (+/-) para um dado n;
- T(n) corresponde ao número total de caracteres para um dado n;

3.1 Floco de neve onda quadrada de Von Koch

- Axioma: F
- Regra: F $\rightarrow$ F-F+F+FF-F-F+F
- Ângulo: 90°

Vamos tentar encontrar um padrão a partir da seguinte tabela:

n	Segmentos (F)	Símbolos (+/-)	Caracteres Totais
1	8	6	14
2	64	54	118
3	512	438	950
4	4096	3510	7606

A partir da análise dos padrões obtidos encontramos as seguintes relações:

$$\begin{cases} F(0) = 1 \\ S(0) = 0 \\ T(0) = 1 \end{cases} \tag{1}$$

$$\begin{cases} F(n) = 8 \cdot F(n-1) \\ S(n) = 8 \cdot S(n-1) + 6 \\ T(n) = 8 \cdot T(n-1) + 6 \end{cases} \tag{2}$$

3.2 Preenchimento do espaço de Hilbert

- Axioma: X
- Regra 1: X → -YF+XFX+FY-
- Regra 2: Y → +XF-YFY-FX+
- Ângulo: 90°

n	Segmentos (F)	Símbolos (X,Y,+,-)	Caracteres Totais
1	3	8	11
2	15	36	51
3	63	148	211
4	255	596	851

A partir da análise dos padrões obtidos encontramos as seguintes relações:

$$\begin{cases} F(0) = 0 \\ S(0) = 1 \\ T(0) = 1 \end{cases} \tag{3}$$

$$\begin{cases} F(n) = 4 \cdot F(n-1) + 3 \\ S(n) = 4 \cdot S(n-1) + 4 \\ T(n) = 4 \cdot T(n-1) + 7 \end{cases} \tag{4}$$

3.3 Fractal de Sierpiński arrowhead curve L-system

A construção desse fractal é baseada apenas em X e Y, sendo que ao final das iterações teremos uma string contendo X e Y, que serão ambos transformados em F. Portanto, para realizar as contas considerarei a quantidade de F = X + Y a cada iteração, assim como a quantidade total de caracteres.

- Axioma: X
- Regra 1: X → Y-X-Y
- Regra 2: Y → X+Y+X
- Ângulo: 60°

n	Segmentos (X)	Segmentos (Y)	Segmento F (X+Y)	Símbolos (+/-)	Caracteres Totais
1	1	2	3	2	5
2	5	4	9	8	17
3	13	14	27	26	53
4	41	40	81	80	161

$$\begin{cases} F(0) = 1 \\ S(0) = 0 \\ T(0) = 1 \end{cases} \tag{5}$$

$$\begin{cases} F(n) = 3 \cdot F(n-1) \\ S(n) = 3 \cdot S(n-1) + 2 \\ T(n) = 3 \cdot T(n-1) + 2 \end{cases} \tag{6}$$

4 4.Complexidade dos Algorítimos

Para cada algorítimo implementado podemos analisar sua complexidade e obter a notação assintótica mais precisa possível.

4.1 Floco de neve onda quadrada de Von Koch

Para calcular a complexidade desse algorítimo devemos considerar o número de caracteres lidos/escritos em cada iteração. Pela fórmula de recorrência temos que o número de caracteres totais ao fim de uma dada iteração n é:

$$T(n) = 8 \cdot T(n-1) + 6 \tag{7}$$

Isso quer dizer que em uma dada iteração n são lidos *T(n-1)* caracteres, que resultam em *T(n)* caracteres escritos. Assim, podemos calcular a complexidade baseada no número de caracteres escritos em cada iteração, que sempre vai ser superior ao número de caracteres lidos.

Resolvendo a equação temos:

$$T(n) = \frac{13 \cdot 8^n}{7} - \frac{6}{7} \tag{8}$$

Portanto a complexidade do algorítimo é $\Theta(8^n)$.

4.2 Preenchimento do espaço de Hilbert

Analogamente, podemos obter a complexidade através da resolução da expressão:

$$T(n) = 4 \cdot T(n-1) + 7 \tag{9}$$

que pode ser resolvido gerando:

$$T(n) = 4^n \cdot \frac{10}{3} - \frac{7}{3} \tag{10}$$

Portanto a complexidade do algorítimo é $\Theta(4^n)$.

4.3 Fractal de Sierpiński arrowhead curve L-system

Para esse fractal temos:

$$T(n) = 3 \cdot T(n-1) + 2 \tag{11}$$

que é resolvido em:

$$T(n) = 2 \cdot 3^n - 1 \tag{12}$$

Portanto a complexidade do algorítimo é $\Theta(3^n)$.

5 Desenho dos Fractais

Para gerar uma representação gráfica dos fractais a partir das strings obtidas após n iterações, criei um página web simples utilizando HTML, CSS e Javascript. Dentro da página é possível abrir um documento de texto e selecionar o ângulo de geração do fractal.

- Link da página para gerar imagens.
- Acessar Github.

A geração da imagem se dá em um elemento do HTML chamado canvas, que permite que o usuário "desenhe". O processo de renderização começa com a leitura de um arquivo de entrada que contém a sequência do fractal, composta por símbolos como F, + e -. Cada símbolo tem um papel específico: F indica a necessidade de desenhar uma linha reta, enquanto + e - modificam a direção do desenho, rotacionando-a em um ângulo definido pelo usuário. O código javascript utilizado para implementar a página é simples e pode ser acessado através do repositório github. Basicamente o programa lê o arquivo de texto, e para cada letra F encontrada traça um segmento de reta. Caso ele encontre um + ou um

- a direção do traço é alterada baseada no ângulo definido. Algumas funcionalidades extras como zoom e recentralização foram implementadas para uma melhor manipulação, mas não são necessárias para visualizar as imagens.

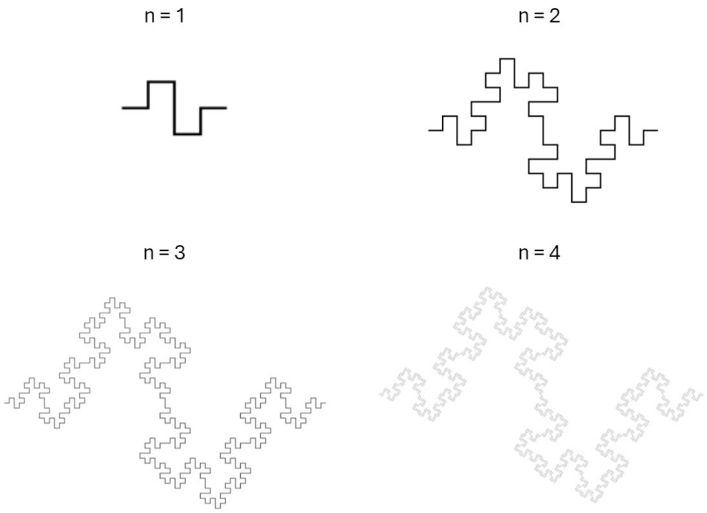

Figura 9: Primeiras 4 iterações do fractal floco de neve onda quadrada de Von Koch.

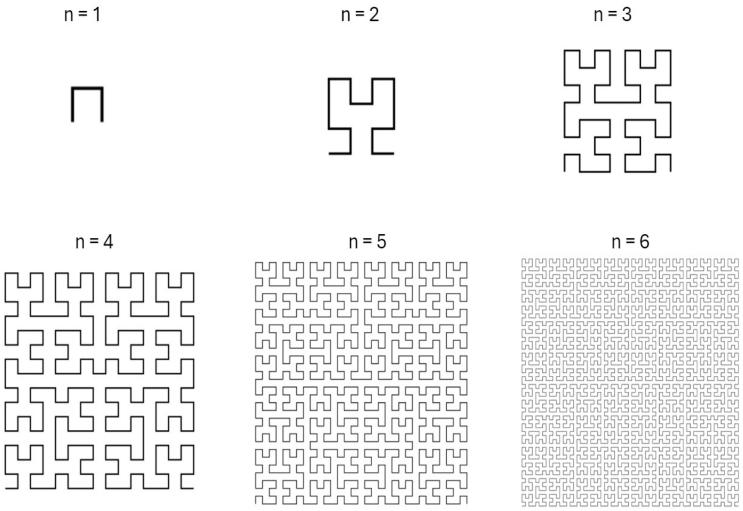

Figura 10: Primeiras 6 iterações do fractal de preenchimento do espaço de Hilbert.

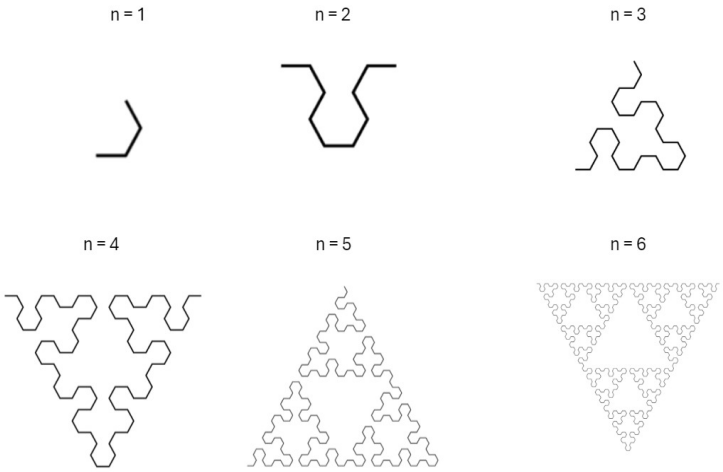

Figura 11: Primeiras 6 iterações do Fractal de Sierpiński arrowhead curve L-system.