**TP de matemática discreta**

**Geração de fractais**

**Ian Paleta Starling – 2024005378**

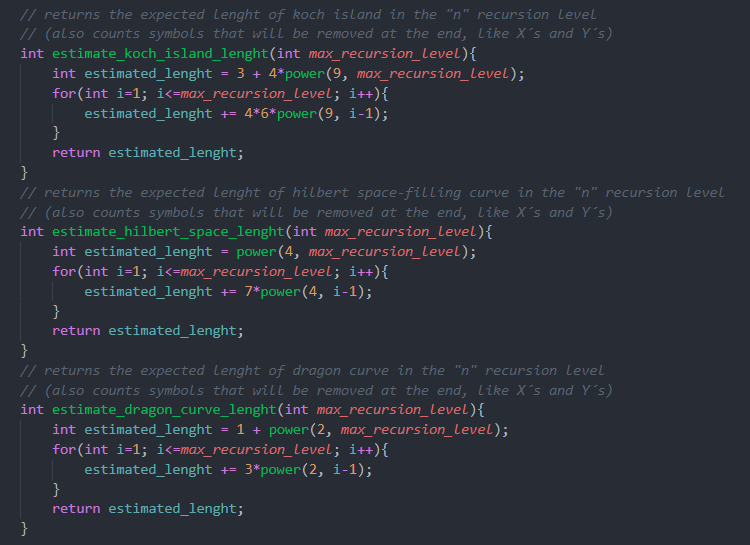
1. Iteratividade ou recursividade?

Durante a implementação do programa, a principal questão que decorre do fato de que a função do programa é gerar fractais, é: Deverá ser feita a implementação de forma recursiva ou iterativa?

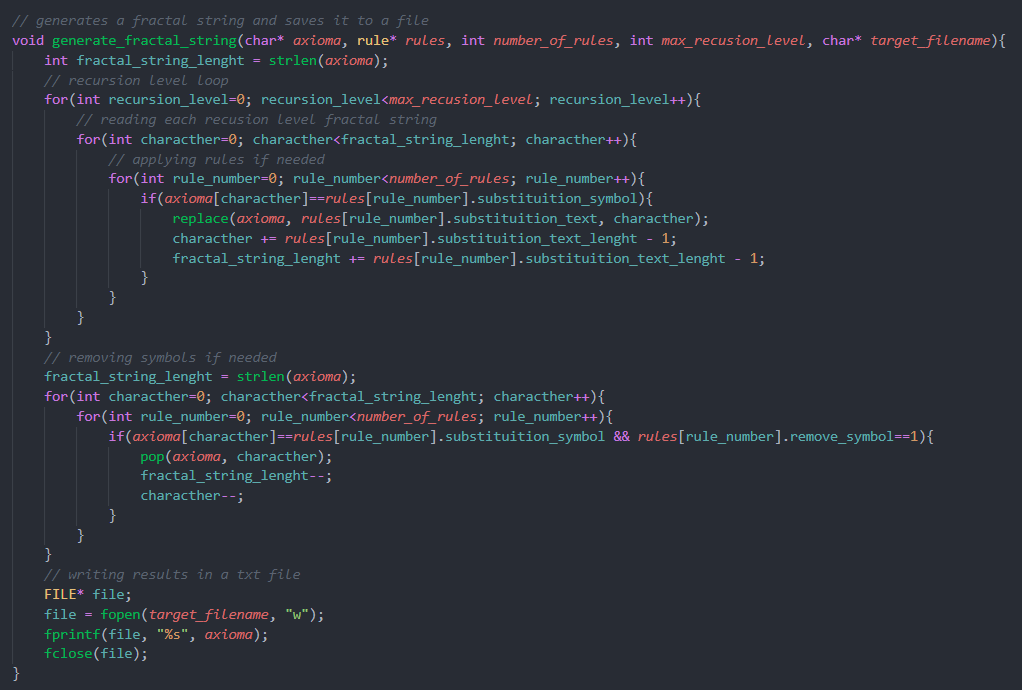
E, no caso do meu código, a resposta escolhida foi a iterativa, por diversas razões. Primeiramente, é fato que implementações iterativas, quando possíveis, são superiores às recursivas em relação à eficiência e ao custo de memória. Implementações recursivas tendem a consumir muito memória da *stack* e a serem limitadas por isso. Além disso, uma opinião pessoal minha é que um código iterativo é mais fácil de compreender do que um recursivo caso não haja conhecimento prévio sobre recursividade.

Principalmente por essas razões, eu escolhi implementar o código de forma iterativa. Claro, implementações iterativas também tem seus negativos. Deixam o código menos elegante e claro (quando quem está lendo tem algum conhecimento sobre recursividade) e de uma forma geral aumentam o tamanho do código. Mas, para mim, a eficiência e não dependência de forma tão grande de memória da *stack*, favoreceram muito a escolha da iteratividade.

No PDF disponibilizado nas instruções para o TP, a sugestão da versão iterativa envolvia ler e gravar arquivos repetidamente. No caso do meu código, acredito que não havia necessidade de acessar várias vezes arquivos de texto diferentes para a gravação, e seria mais eficiente fazer todas as modificações dentro de uma *string* previamente alocada e depois inseri-la em um arquivo caso necessário.



O código descobre quantos caracteres a *string* final terá (antes da remoção de X e Y) para alocar todo o espaço necessário para a geração do fractal de uma vez.



Então, realiza operações somente dentro dessa *string* previamente alocada (mudando as posições analisadas de acordo com o número de caracteres substituídos para não entrar em *loop* infinito), aplicando as regras no axioma de acordo com o nível de recursão desejado.

Assim, não é necessário abrir e gravar arquivos, deixando o programa mais eficiente.

1. Equações de recorrência

Terminologia:

F(n) = número de Fs no nível *n* de recursão

X(n) = número de Xs no nível *n*de recursão

Y(n) = número de Ys no nível *n*de recursão

S(n) = número total de símbolos (F, X, Y, +, -) no nível *n* de recursão

T(n) = número de símbolos que não são Fs no nível *n* de recursão

M(n) = número de símbolos que não são X ou Y no nível *n* de recursão

* Ilha de Koch

F(0) = 4

F(n) = 9\*F(n-1)

T(n) = S(n) – F(n)

S(0) = 7

S(n) = 15\*F(n-1) + T(n-1) = 14\*F(n-1) + S(n-1)

* Preenchimento de Espaço de Hilbert

X(0) = 1

X(n) = 2\*X(n-1) + 2\*Y(n-1)

Y(0) = 0

Y(n) = 2\*X(n-1) + 2\*Y(n-1)

F(0) = 0

F(n) = 3\*X(n-1) + 3\*Y(n-1) + F(n-1)

T(n) = S(n) – F(n)

M(n) = S(n) – X(n) – Y(n)

S(0) = 1

S(n) = 11\*X(n-1) + 11\*Y(n-1) + M(n-1) = 10\*X(n-1) + 10\*Y(n-1) + S(n-1)

* Curva do Dragão

X(0) = 1

X(n) = X(n-1) + Y(n-1)

Y(0) = 0

Y(n) = X(n-1) + Y(n-1)

T(n) = S(n) – F(n)

M(n) = S(n) – X(n) – Y(n)

S(0) = 2

S(n) = 5\*X(n-1) + 5\*Y(n-1) + M(n-1) = 4\*X(n-1) + 4\*Y(n-1) + S(n-1)

1. Complexidade da implementação

O código trabalha através do uso de um algoritmo principal e seis algoritmos auxiliares. Os auxiliares desempenham papeis menores nos algoritmos principais, como remover um caractere, inserir uma *string*, ou elevar um uma base a um expoente. Já o algoritmo principal é o que realiza a geração do fractal.

Texto

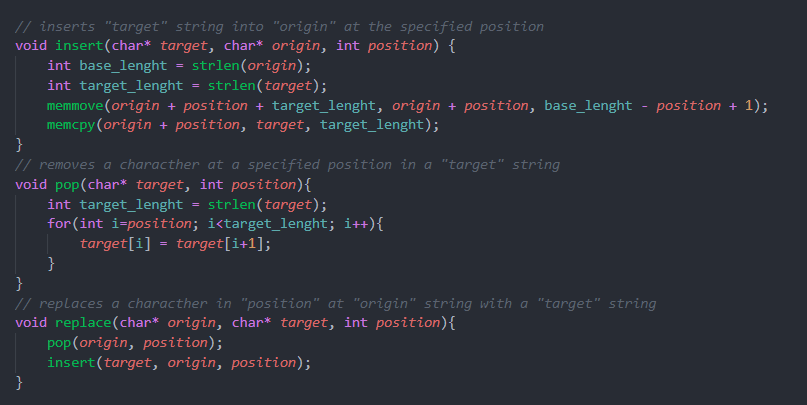
Descrição gerada automaticamente

A função *generate\_fractal\_string* recebe como parâmetros um axioma, um conjunto de regras e seu tamanho, o máximo nível de recursão desejado e um arquivo de saída. Note que essa função é capaz de gerar fractais independentemente do número de regras de cada um. Portanto, ela foi usada para a geração dos 3 fractais escolhidos.

O primeiro loop ocorre o número de vezes do nível máximo de recursão. O segundo ocorre de acordo com o número de caracteres de cada nível de recursão. O terceiro depende do número de regras do fractal (que é constante, e não muda durante a execução do algoritmo).

*O If* que compara se um caractere deve ser substituído tem um custo 1, e ambas as somas dentro do *if* terão custo 1 também. O acesso ao caractere e à regra correta tem custo 1.

As funções *pop*, *replace* e *insert* são funções auxiliares para fazer a manipulação de *strings*. Respectivamente, apaga um caractere, troca um caractere por uma *string* e insere uma *string* em outra.



A função *insert* tem o custo do tamanho de *target* e *origin* por conta do *strlen*, mais o custo de *memmove* que é o tamanho de *origin* menos a posição desejada, adicionado ao custo de *memcpy* que será o tamanho de *target*.

A função *pop* tem o custo do tamanho de target multiplicado por dois, menos a posição desejada. E a função *replace* soma ambos os custos.

Após o término do loop principal para aplicar as regras, temos a remoção de símbolos (caso seja necessária) que terá o custo do tamanho final do fractal multiplicado pelo número de regras, multiplicado novamente pelo tamanho final do fractal, por conta do uso da função *pop*.

E, por fim, o custo de abrir um arquivo, escrever nesse arquivo e fechá-lo seria O(L) onde L é o tamanho da *string* escrita.

O custo total será: O(*max\_recursion\_level*\**number\_of\_rules*\*S(*max\_recursion\_level)*)

Onde S(*max\_recursion\_level*) é o tamanho final da *string* do fractal. Note que o custo é exponencial ao parâmetro *max\_recursion\_level* uma vez que S tem o crescimento exponencial.

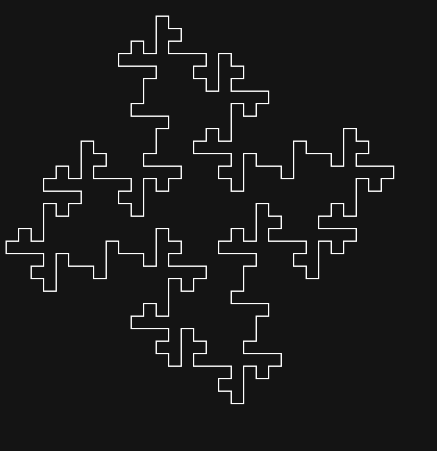
1. Ferramentas para desenhar os fractais

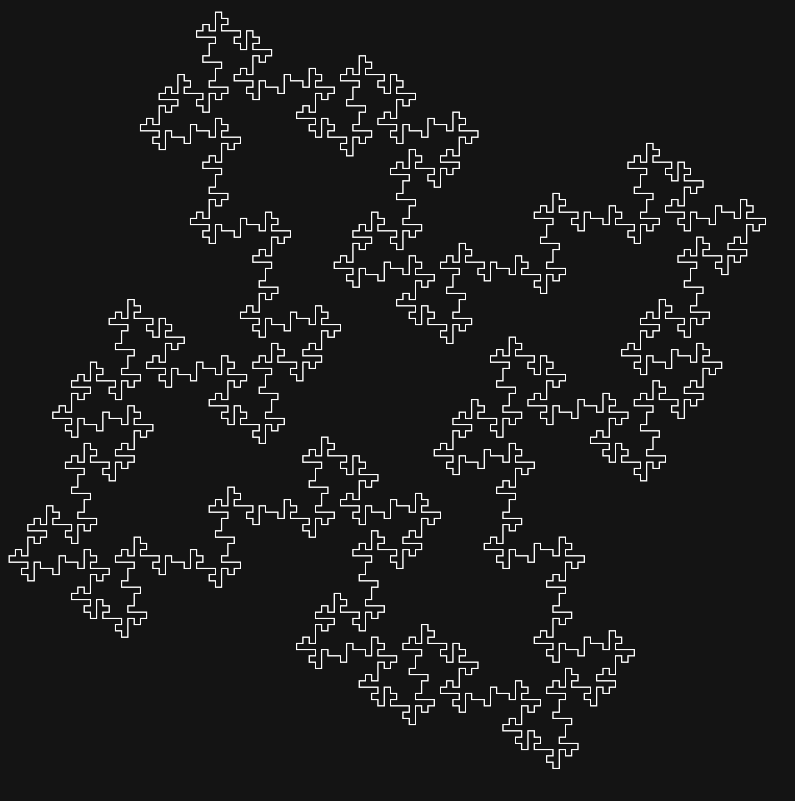
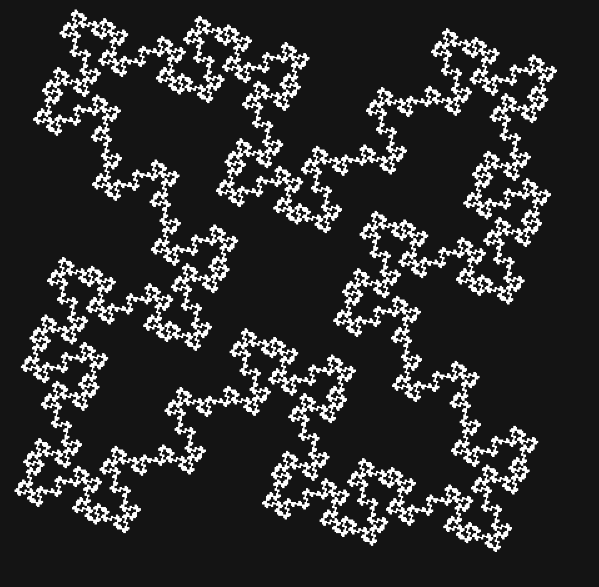
Existem diversas ferramentas para desenhar os fractais. Sites online podem gerar os desenhos derivados do *l-system*, e bibliotecas como o *turtle* ou o *matplotlib* podem ser usadas para a geração das figuras.

Aqui, usarei um site disponível em <https://piratefsh.github.io/p5js-art/public/lsystems/> para gerar as figuras.

Ilha de Koch (níveis de recursão 1, 2, 3 e 4)

Texto

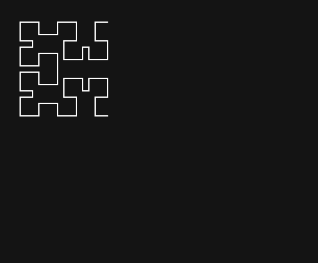
Descrição gerada automaticamente com confiança baixa



Preenchimento de Espaço de Hilbert (níveis de recursão 1, 2, 3 e 4)

Uma imagem contendo Forma

Descrição gerada automaticamenteUma imagem contendo Texto

Descrição gerada automaticamente

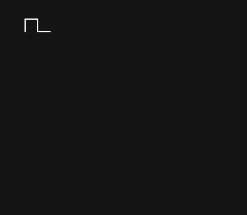
Desenho em preto e branco

Descrição gerada automaticamente com confiança média

Curva do Dragão (níveis de recursão 1, 2, 3, 4 e 10)

Uma imagem contendo Forma

Descrição gerada automaticamente

Forma

Descrição gerada automaticamente com confiança baixaUma imagem contendo Forma

Descrição gerada automaticamente