Relatório de Fractais e *L-System*

Adriel Costa, Eduardo Marinho, Edson Costa, Fernanda Lustosa, Samuel Morais

Dezembro 2024

Sumário

1 Objetivos

Este relatório é produto de um trabalho da disciplina **DIM0606 - Linguagens Formais e Autômatos**, ministrada pelo Dr. Martin Alejandro Musicante, professor do Departamento de Informaática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte, e tem o objetivo de descrever o cumprimento dos seguintes objetivos:

1. Identificar um alfabeto suficientemente rico para descrever *L-Systems*. A alfabeto deve conter, pelo menos, primitivas para empilhar e desempilhar a posição do cursor;
2. Propor primitivas para extender um *L-System* na prática, de forma a considerar diferentes tamanhos e cores de traços, assim como diferentes ângulos de giro para o cursor;
3. Criar um algoritmo e um programa de computador que dado um número natural n e um *L-System* enriquecido com os dados do ítem anterior, gere uma cadeia obtida pela n-ésima rescrita do L-System;
4. Criar um algoritmo que transforme a cadeia obtida no ítem anterior num programa que use uma biblioteca de gráficos de tartaruga para desenhar a curva descrita pela cadeia;
5. Criar um programa de computador que receba um um *L-System* (ou a gramática dequivalente), uma *string* e um número natural representando o número de iterações usado para produzir a *string* e devolva **Verdadeiro** ou **Falso**, indicando se essa string corresponde à n-ésima derivação no *L-System*;
6. A resolução deve ser feita utilizando um Autômato com Pilha.

Neste link se encontra o repositório com o código fonte do projeto.

2 Alfabeto

O seguinte alfabeto foi utilizado para o desenvolvimento da atividade:

$$\mathrm{A} = \{F, f, G, g,], [, r, g, b, +, -, T, t\}$$

Além disso, é importante mencionar que as regras de derivação devem ser informadas pelo usuário no momento em que o código for executado.

Na tabela abaixo encontram-se descrições mais detalhadas sobre cada elemento do alfabeto.

Tipo	Valor	Descrição
Variaveis	F ou G	Desenha um traço de tamanho 1
	f ou g	Desenha um traço de tamanho meio
Constantes	[	Empilha
	]	Desempilha
	r	Cor vermelha
	g	Cor verde
	b	Cor azul
	$+$	Altera o ângulo do traço para esquerda
	$-$	Altera o ângulo do traço para direita
	T	Aumenta a espessura do traço
	t	Diminui a espessura do traço
Regras		Definidas pelo usuário durante a execução.

Tabela 1: Tabela do alfabeto

3 Códigos

A linguagem de programação utilizada foi *Python3*, com auxílio de algumas bibliotecas externas:

- `turtle`: Para implementar a parte gráfica, ou seja, "desenhar"o fractal à partir de uma cadeia de caracteres;
- `sys`: Para receber parâmetros do usuário ao executar o programa à partir de um emulador de terminal;
- `collections` (apenas o módulo `namedtuple`): Para armazenar o estado do programa durante a renderização.

3.1 Estrutura do Diretório

A solução desenvolvida se utiliza de um diretório (*src*) com 4 arquivos em linguagem *Python3* dispostos da seguinte maneira:

```
src/
├── LSystem.py
├── alphabet.py
├── main.py
├── renderer.py
└── test.py
```

3.1.1 `main.py`

- Responsável por:

  - Ler os argumentos de entrada da linha de comando;
  - Solicitar dados do usuário (número de iterações, regras de reescrita, ângulo e axioma);
  - Determinar a funcionalidade desejada: checar ou renderizar.

- Funções:
  - `read_render_input()`: Lê dados necessários para renderização;
  - `read_check_input()`: Lê dados necessários para validação de um *L-System*;
  - `read_rules()`: Solicita ao usuário a definição das regras de reescrita.

3.1.2 `alphabet.py`

Define o alfabeto do *L-System*, mapeando caracteres para ações específicas usando a biblioteca `turtle`.

3.1.3 `LSystem.py`

- Implementa a lógica central do *L-System*:
  - Validação de strings geradas;
  - Aplicação das regras de produção ao axioma para gerar novas *strings*;
  - Verificação se uma *string* fornecida é consistente com o *L-System* esperado;

- Funções:
  - `validate(l_system)`: Verifica se uma string contém apenas caracteres válidos no alfabeto;
  - `check(axiom, string, rules, n_iterations)`: Faz algumas manipulações nos parâmetros de entrada para poder chamar corretamente função `check_aux()`;
  - `check_aux(string, rules, n_iterations, stack)`: Valida se a *string* fornecida corresponde ao resultado das reescritas do axioma. Isso é feito utilizando uma pilha;
  - `apply_production_rules(l_system, rules, n_iterations)`: Faz algumas manipulações nos parâmetros de entrada para poder chamar corretamente a função `apply_rules()`;
  - `apply_rules(string, rules, start_index)`: Aplica recursivamente as regras de produção ao *L-System*;
  - `add_prefix(prefix, string_list)`: Adiciona um prefixo a cada *string* em uma lista de *strings*.

3.1.4 `renderer.py`

Classe responsável por renderizar o *L-System* utilizando a biblioteca `turtle`. Implementa operações gráficas como:

- Salvamento e restauração do estado da tartaruga usando uma pilha;
- Desenho baseado nos caracteres definidos no alfabeto.

3.1.5 `test.py`

Arquivo utilizado para fazer alguns testes simples.

3.2 Checando uma *String*

Verifica se uma *string* corresponde ao resultado esperado de um *L-System* após um certo número de reescritas. Para checar uma *string* são necessárias algumas entradas por parte do usuário. São elas:

1. Número de reescritas, na forma de um inteiro (por exemplo, **4**);
2. Axioma criado à partir do alfabeto presente na seção 2 (por exemplo, **F-fbfg-f**);
3. *String* que se deseja checar (por exemplo, **r-rrrg-r**);
4. Regras de derivação escritas no formato **char->string** (por exemplo, **F->rT**). Podem ser escritas tantas regras quanto forem necessárias.

```
*[main][~/Code/Projects/LSystem]$ python3.13 src/main.py checar
Número de reescritas: 4
L-System: F-fbFg-f
String: r-rrrg-r
Escreva as regras a seguir no seguinte formato: char→string
Quando não houver mais regras deixe a linha em branco!
Regra: F→b
Regra: b→r
Regra: f→F
Regra:
Verdadeiro: O L-System esperado é equivalente ao L-System após 4 reescritas
```

Figura 1: Exemplo de checagem de uma *string*

3.2.1 Funcionamento

1. Primeiramente, o código irá "puxar"da linha de comando o argumento (**checar**);
2. O código entra numa condicional;

3. Dentro da condicional, os parâmetros `n_iterations` (número de iterações), `axiom` (axioma do *L-System*) e `string` (*string* final que se deseja checar) são lidos e armazenados;

4. As regras de reescrita são lidas e armazenadas;

5. Por fim, a função `check()` do arquivo `L-System.py` é chamada para verificar se a *string* fornecida é resultado da reescrita do *L-System* com o número de iterações e regras fornecidos.

3.3 Renderizando Fractais

Para desenhar o fractal use o comando **pyhton3.13 src/main.py renderizar**. Os seguintes argumentos serão necessários:

1. Número de reescritas, na forma de um inteiro (por exemplo, **4**);

2. Ângulo de virada em graus, também na forma de um inteiro (por exemplo, **90**);

3. Axioma criado à partir do alfabeto presente na seção 2 (por exemplo, **F-fbfg-f**);

4. Regras de derivação escritas no formato **char->string** (por exemplo, **F->rT**). Podem ser escritas tantas regras quanto forem necessárias.

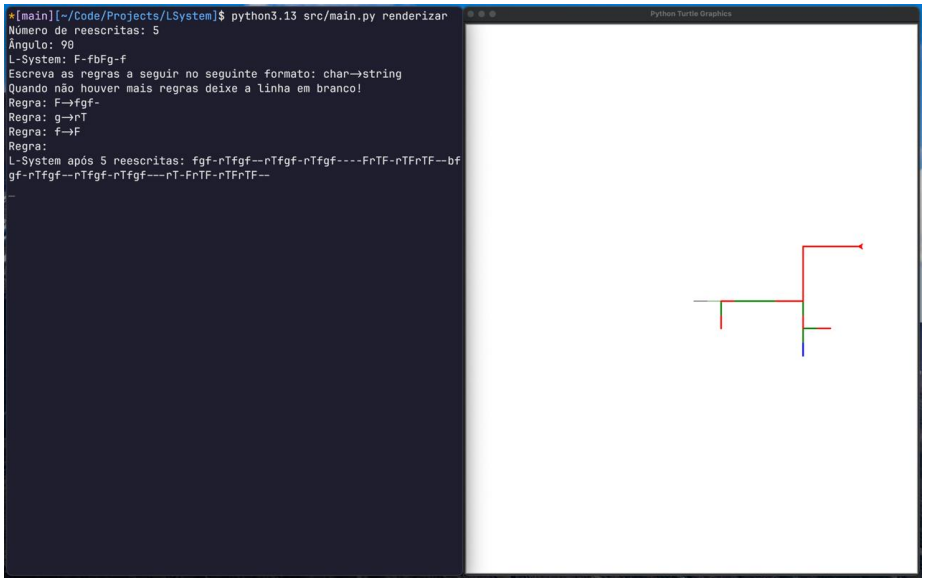

Figura 2: Exemplo de renderização de um *L-System*

3.3.1 Funcionamento

1. Primeiramente, o código irá "puxar"da linha de comando o argumento (**renderizar**);

2. O código entra numa condicional;

3. Dentro da condicional, a primeira coisa a ser feita é a checagem do axioma, ou seja, verificar se a cadeia de caracteres está de acordo com o alfabeto;

4. Caso tudo esteja correto, a função `apply_production_rules()` é chamada para construir o *L-System* final;

5. O *L-System* final é mostrado e, em seguida é renderizado na tela.

4 Conclusão

Os modelos teóricos utilizados podem cumprir os objetivos propostos e os códigos que implementaram os modelos funcionam como esperado. Portanto, o trabalho foi feito com êxito.

5 Tabela de Participação

Tabela de Participação	
Nome	**Participação (de 0 a 10)**
Adriel Costa	**10**
Eduardo Marinho	**10**
Edson Costa	**10**
Fernanda Lustosa	**10**
Samuel Morais	**10**