



**ENGENHARIA DE SOFTWARE / GRADUAÇÃO**

**Sistema de Criptografia Educacional Baseado em  
Árvores Binárias de Busca**

**MARICÁ 2025**

## **UNIVERSIDADE DE VASSOURAS**

Sistema de Criptografia Educacional Baseado em Árvores Binárias de Busca

**ENIGMA-BST**

**Professor:** Márcio Garrido

**Alunos:** Eduardo Corrêa de Lima-202413753

Kariny Cunha -202423500

Monique E R Gomes -202514946

Rodrigo Monteiro -202413846

Guilherme Rodrigues -202322061

## Sumário

1. INTRODUÇÃO .....	4
2. OBJETIVOS .....	5
2.1 Objetivo Geral.....	5
2.2 Objetivos Específicos.....	5
3. FUNDAMENTAÇÃO TEÓRICA.....	6
3.1 Criptografia Educacional .....	6
3.2 Estruturas de Dados .....	6
3.3 Matemática Aplicada .....	6
4. METODOLOGIA .....	7
4.1 Processo de Desenvolvimento .....	7
4.2 Ferramentas e Tecnologias.....	7
4.2.1 Stack Tecnológico Principal.....	7
4.2.2 Bibliotecas Utilizadas.....	7
4.3 Estrutura do Projeto.....	7
5. RESULTADOS PARCIAIS .....	9
5.1 Módulos Implementados e Validados.....	9
5.1.1 Grupo A – Criptografia .....	9
5.1.2 Grupo B – Descriptografia .....	9
5.2 Fluxo de Processamento Atual.....	9
5.3 Especificações Técnicas dos Módulos.....	10
5.3.1 Módulo Integral Processor .....	10
5.3.2 Módulo JSON Reader .....	10
6. DISCUSSÃO .....	11
6.1 Análise de Desempenho.....	11
6.2 Robustez e Confiabilidade .....	11
6.3 Alinhamento com Objetivos Pedagógicos .....	11
7. TRABALHOS FUTUROS .....	12
7.1 Desenvolvimentos Imediatos .....	12
7.2 Melhorias de Longo Prazo .....	12
8. CONCLUSÕES .....	13
REFERÊNCIAS.....	14
APÊNDICE A .....	14

# **RELATÓRIO TÉCNICO**

## **1. INTRODUÇÃO**

Este relatório técnico tem como objetivo apresentar o desenvolvimento do Sistema Enigma-BST, um projeto acadêmico voltado para a aplicação prática de conceitos de estruturas de dados, programação orientada a objetos e criptografia didática. Inspirado na Máquina Enigma, o sistema utiliza Árvores Binárias de Busca (BinarySearchTrees - BST) como mecanismo de embaralhamento e simula um ambiente gamificado, no qual o Grupo A (Criptografia) é responsável pela geração de mensagens criptografadas e o Grupo B (Descriptografia) pelo processo inverso de decodificação. O projeto integra disciplinas de Estruturas de Dados Avançadas e Processo de Desenvolvimento de Software, atendendo aos critérios avaliativos que incluem implementação técnica robusta e documentação metodológica detalhada.

## 2. OBJETIVOS

### 2.1 Objetivo Geral

Desenvolver um sistema modular de criptografia educacional reversível, utilizando árvores binárias de busca como mecanismo central de embaralhamento de dados.

### 2.2 Objetivos Específicos

- Implementar modularização em Python com foco na reutilização de código;
- Aplicar estruturas de dados avançadas (BST com travessia em pós-ordem);
- Utilizar algoritmos determinísticos de transformação matemática;
- Desenvolver funcionalidades de serialização/desserialização em JSON;
- Documentar o projeto conforme boas práticas de Engenharia de Software.
- Implementar uma dinamicagamificada entre grupos.
- Simular o funcionamento da máquina enigma.

### 3. FUNDAMENTAÇÃO TEÓRICA

#### 3.1 Criptografia Educacional

A criptografia didática tem como foco demonstrar conceitos computacionais, sem objetivo de segurança real em sistemas de informação. O enfoque está na clareza algorítmica e na reversibilidade determinística.

#### 3.2 Estruturas de Dados

O projeto utiliza Árvores Binárias de Busca (BST), uma das estruturas fundamentais da Ciência da Computação. Sua aplicação envolve operações de inserção e travessia em pós-ordem, empregadas como técnica de embaralhamento de dados e manipulação de sequências de caracteres e valores.

#### 3.3 Matemática Aplicada

O módulo de criptografia aplica cálculos integrais sobre valores ASCII, garantindo operações reversíveis por meio de transformações matemáticas definidas pelas fórmulas:

##### **Criptografia:**

$$R = (\text{num} * \text{key}^2) / 2$$

##### **Descriptografia:**

$$\text{num} = (2 * R) / \text{key}^2$$

## 4. METODOLOGIA

### 4.1 Processo de Desenvolvimento

O desenvolvimento seguiu uma metodologia ágil, estruturada em ciclos iterativos organizados no Trello, conforme o fluxo:

Backlog → Análise → Implementação → Testes → Documentação → Revisão

### 4.2 Ferramentas e Tecnologias

#### 4.2.1 Stack Tecnológico Principal

Linguagem: Python 3.13

IDE: Visual Studio Code

Controle de Versão: Git + GitHub

Gestão de Projeto: Trello

#### 4.2.2 Bibliotecas Utilizadas

```
sympy==1.14.0      # Cálculos simbólicos e integrais
mpmath==1.3.0       # Aritmética de alta precisão
json               # Manipulação de dados estruturados
argparse          # Interface de linha de comando
typing             # Anotações de tipo para qualidade de código
tkinter            # Interface Gráfica
Pytest             #Para a suíte de testes unitários e de integração.
Pytest-Mock        #Plugin para simular comportamentos durante os testes.
```

### 4.3 Estrutura do Projeto

```
enigma/
├── docs/
│   ├── orientacao.txt
│   └── projeto.pdf
└── module/
    ├── __init__.py
    ├── ascii_converter.py # String → ASCII decimal → String
    ├── IntegralProcessor.py # Aplica integral
    ├── binary_converter.py # Decimal → Binário → Decimal
    ├── input_text.py # recebe a entrada de texto
    ├── Tree.py # criação e desmonte da árvore
    └── json_exporter.py # Gera JSON + exporta |
├── testes/
    ├── __init__.py
    ├── conftest.py
    ├── test_ascii_convertes.py
    ├── test_binary_converter.py
    ├── test_input_text.py
    ├── test_IntegralProcessor.py
    ├── test_json_exporter.py
    ├── test_Tree.py
    └── test_main_integration.py |
├── .gitignore
└── README.md
└── requirements.txt
```

## 5. RESULTADOS PARCIAIS

### 5.1 Módulos Implementados e Validados

#### 5.1.1 Grupo A – Criptografia

- `entrada_string_curta.py`: Conversão de texto em lista de caracteres, com suporte a múltiplos encodings;
- `ascii_converter.py`: Mapeamento bidirecional ASCII ↔ caracteres (0–127);
- `integral_processor.py`: Transformação criptográfica integral reversível com cálculo simbólico.

#### 5.1.2 Grupo B – Descriptografia

- `ljson_reader.py`: Leitura avançada de JSON (arquivo, string, stdin), validação robusta e tratamento de erros;
- `3binary_decimal_converter.py`: Conversão binário ↔ decimal com suporte a fracionários e arredondamento.

### 5.2 Fluxo de Processamento Atual

- Criptografia (Grupo A):
- Texto → Lista de Caracteres → ASCII → Transformação Integral → [Binário → BST → JSON]
- Descriptografia (Grupo B):
- JSON → Leitura Estruturada → Binário → Decimal → [Transformação Inversa → ASCII → Texto]

## 5.3 Especificações Técnicas dos Módulos

### 5.3.1 Módulo Integral Processor

```
classIntegralProcessor:  
def encrypt(self, numbers: List[int]) -> List[float]:  
    # Implementa: R=(num * key2 )/ 2  
    # Complexidade: O(n)  
  
def decrypt(self, numbers: List[float]) -> List[int]:  
    # Implementa: num = (2 *R) / key2  
    # Complexidade: O(n)
```

### 5.3.2 Módulo JSON Reader

- Suporte a múltiplas fontes (arquivo, string, stdin);
- Validação robusta com mensagens de erro detalhadas;
- Fallback de encoding (UTF-8 → Latin-1);
- Formatação adaptativa da saída.

## 6. DISCUSSÃO

### 6.1 Análise de Desempenho

- Complexidade computacional:  $O(n)$  para os algoritmos principais (integrais e conversão binária).
- Consumo de memória: Linear em função do tamanho da mensagem.
- Precisão numérica: Garantida pela aritmética simbólica (Sympy).

### 6.2 Robustez e Confiabilidade

- Validação de entradas: JSON com fallback de encodings;
- Tratamento de erros: Conversão binária com mensagens detalhadas;
- Testabilidade: Módulos independentes permitem testes unitários focados.

### 6.3 Alinhamento com Objetivos Pedagógicos

- Modularização do código com arquitetura reutilizável
- Estruturas de dados avançadas prontas para BST
- Criptografia didática com transformações reversíveis
- Documentação e versionamento conforme padrões profissionais

## 7. TRABALHOS FUTUROS

### 7.1 Desenvolvimentos Imediatos

- Implementação da BST com travessia em pós-ordem;
- Módulo de visualização gráfica da árvore (tree\_visualizer.py);
- Desenvolvimento do ascii\_decoder.py;
- Geração e validação de chaves determinísticas.

### 7.2 Melhorias de Longo Prazo

- Testes unitários (pytest) com cobertura mínima de 80%;
- Criação de interface gráfica (GUI) para uso educacional;
- Otimização de performance para grandes volumes de dados;
- Expansão para múltiplos algoritmos criptográficos.

## 8. CONCLUSÕES

O Sistema Enigma-BST atingiu aproximadamente 50% de implementação dos módulos planejados, consolidando um pipeline funcional para criptografia parcial (Grupo A) e descriptografia parcial (Grupo B).

A arquitetura modular mostrou-se eficaz para o desenvolvimento colaborativo, permitindo que componentes fossem implementados e testados independentemente.

A integração da matemática computacional com estruturas de dados provou-se viável no contexto educacional, fornecendo uma base sólida para o ensino de algoritmos criptográficos e princípios de engenharia de software.

Com a futura implementação da BST e conclusão do pipeline, o sistema poderá operar como uma plataforma robusta de experimentação em criptografia acadêmica, atendendo plenamente aos objetivos pedagógicos.

## REFERÊNCIAS

- PYTHON SOFTWARE FOUNDATION. *Python 3.13 Documentation*. 2025.
- SYMPY DEVELOPMENT TEAM. *SymPy: Python library for symbolic mathematics*. 2025.
- CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. *Algoritmos: Teoria e Prática*. 3. ed. Rio de Janeiro: Elsevier, 2012.
- KNUTH, D. E. *The Art of Computer Programming*. Vol. 3: Sorting and Searching. 2. ed. Boston: Addison-Wesley, 1998.

## APÊNDICE A

Códigos-fonte disponíveis em:

[https://github.com/\[repositório\]/enigma-bst](https://github.com/[repositório]/enigma-bst)