# RELATÓRIO

MAC0323

## Unique substrings of length L

Este programa resolve o exercício 5.2.2 Unique Substrings of Length L do Algorithms 4th ed. do Sedgewick e inclui mais alguns estudos divididos nas seguintes partes:

1. Maior L-completo com números gerados aleatoriamente
2. Maior L-completo com os dígitos de $\pi$

Para simplificar os dados mostrados posteriormente, segue a definição de uma sequência L-completa:

*UMA SEQUÊNCIA S DE DÍGITOS É L-COMPLETA SE TODAS AS $10^L$ SEQUÊNCIAS DE DÍGITOS OCORRE EM S*

Para todos os fins, qualquer L apresentado aqui significa maior L para o qual uma sequência dada é L-completa

- **Nota:** Com o intuito de facilitar os testes, este programa foi implementado com uma interface *ITabelaSimbolo* que implementa as funções necessárias para o exercício. Assim, as estruturas *RedBlackBST*, *SeparateChainingHashST*, *TST* e *DigitTrieST (10-way Trie)* puderam implementar essa interface de modo que a programação fosse única e independente da estrutura de dados utilizada

# Relatório

**UNIQUE SUBSTRINGS OF LENGTH L**

## CAPÍTULO 1 | MAIOR L-COMPLETO COM NÚMEROS GERADOS ALEATORIAMENTE

O método de geração de número aleatórios foi *random.nextInt(10)*. Sendo N o número de dígitos contidos no texto, ou seja, o comprimento total do texto. Segue os dados obtidos para:

### $N = 10^6$

| *Estrutura de dados* | *Tempo (s)* |
|---|---|
| TST | 1.533 |
| RedBlackBST | 1.732 |
| SeparateChainingHashST | 1.621 |
| DigitTrieST | 0.684 |

- L achado: 4

### $N = 10^7$

| *Estrutura de dados* | *Tempo (s)* |
|---|---|
| TST | 15.418 |
| RedBlackBST | 23.963 |
| SeparateChainingHashST | 19.228 |
| DigitTrieST | 9.797 |

- L achado: 5

**O maior N que consegui achar foi $10^7$. O programa rodou bem e poderia continuar sendo testado para número maiores, mas o mesmo estourava por memória (quando chegava a 100mb de chars usados) e não consegui achar como aumentar o limite de memória do Java.**

# CAPÍTULO 2 | MAIOR L-COMPLETO COM OS DÍGITOS DE PI

O estudo foi realizado com o primeiro milhão e os primeiros 10 milhões de dígitos de $\pi$, respectivamente. Com a leitura inteira da String, o programa não conseguia alocar memória suficiente e parava de rodar antes mesmo de começar a executar o exercício. Para contornar este problema, foi feita a leitura dígitos a dígito ganhando uma performance bastante considerável.

Contudo, estes testes foram feitos utilizando a IDE Eclipse, e a leitura de caracteres depende da interrupção CTRL+Z no console indicando o fim do Standard Input. Assim, os resultados obtidos podem ser um pouco discrepantes em relação ao tempo, principalmente em casos onde o programa é executado rapidamente (caso do 1 milhão de dígitos).

## 1 milhão de dígitos

| *Estrutura de dados* | *Tempo (s)* |
|---|---|
| TST | 2.374 |
| RedBlackBST | 1.971 |
| SeparateChainingHashST | 1.756 |
| DigitTrieST | 1.092 |

- L achado: 4

## 10 milhões de dígitos

| *Estrutura de dados* | *Tempo (s)* |
|---|---|
| TST | 17.306 |
| RedBlackBST | 29.334 |
| SeparateChainingHashST | 19.306 |
| DigitTrieST | 13.379 |

- L achado: 5