|  |  |
| --- | --- |
|  | Relatório |
|  |  |
| MAC0323 | Unique substrings of length L |
|  | Este programa resolve o exercício [5.2.2 Unique Substrings of Length L](http://algs4.cs.princeton.edu/52trie/) 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   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 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 =

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

* L achado: 4

## N =

|  |  |
| --- | --- |
| 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 . 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 , 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