

UNIVERSIDADE FEDERAL DE MINAS GERAIS
Instituto de Ciências Exatas
Departamento de Ciência da Computação
DCC207 -- Algoritmos 2
Prof. Renato Vimieiro
Trabalho Prático 1 - Manipulação de sequências
Guilherme de Abreu Lima Buitrago Miranda - 2018054788

1 - Introdução

Esse trabalho busca, primeiramente, abordar aspectos práticos de manipulação de sequências, como a implementação de árvores de sufixos e a busca de padrões nas mesmas. Além disso, procura auxiliar a fixação do conteúdo, trabalhando com os conceitos teóricos vistos em sala de aula na prática.

Assim, foi implementado um algoritmo que recebe uma sequência de caracteres por meio de um arquivo “.fasta” passado como argumento na linha de comandos e imprime, posteriormente, a maior subsequência que se repete no texto. Dessa forma, esse relatório busca explicar a construção desse algoritmo e discutir questões relacionadas ao seu desempenho.

2 - Implementação

A implementação do algoritmo foi feita utilizando a linguagem Python e separada em dois arquivos: main.py e suffix_tree.py. O primeiro contém os métodos gerais do programa, como a leitura do arquivo (detalhada no tópico 2.1). Nele, é chamado o segundo arquivo, que contém a implementação de duas classes: SuffixTree e Node, sendo a última interna à primeira. Além disso, nesse arquivo, também há a implementação de um método para obter o uso de memória em Megabytes, que será útil para a análise do tópico 3.

2.1 - Leitura do Arquivo

Para detalharmos a respeito da leitura do arquivo é importante, em primeiro lugar, observar como é a estrutura de um arquivo “.fasta”. Nele, a primeira linha se inicia com o caractere “>” e apresenta informações descritivas do que se trata o arquivo. No caso específico do arquivo “sarscov2.fasta”, a primeira linha informa que trata-se do genoma completo do vírus sars-cov-2 isolado em Wuhan.

Para os fins deste trabalho, a primeira linha do arquivo será ignorada, assim como os caracteres de fim de linha (“\n”) que dividem o genoma, para facilitar a leitura humana. Assim, já na linha 12 do arquivo main.py, a variável genome está completamente processada, sendo composta apenas pela sequência de caracteres referentes ao genoma do vírus. Por fim, na linha 13, o arquivo é fechado, e é iniciada a construção da árvore de sufixos.

2.2 - Construção da Árvore de Sufixos

A fim de obter-se uma boa modularização na implementação do algoritmo, utilizou-se algumas das boas práticas da programação orientada a objetos. Assim, tem-se a classe `SuffixTree`, que, como o próprio nome indica, contém a estrutura básica da árvore de sufixos construída. Além disso, ela também armazena a string original (no caso, o genoma do sars-cov-2), o tamanho dessa string e o nó raiz, que aponta para outros nós.

Os nós são implementados como uma classe interna à `SuffixTree`, armazenando os índices de início e fim da substring que ele armazena e um dicionário que mapeia caracteres para outros nós.

Com relação aos métodos da classe `SuffixTree`, além da inicialização, tem-se o `add_suffixes`, `find_biggest_repeated_substring` e `follow_nodes`, que serão detalhados nos parágrafos a seguir.

O método de inicialização, além de ser responsável pelas saídas relativas ao desempenho do algoritmo, instancia as variáveis da classe e chama o método `add_sufixes` para construir a árvore. Posteriormente, também invoca o método para encontrar a maior substring contida no texto.

O método `add_suffixes`, por sua vez, é responsável por toda a construção da árvore de sufixos que será utilizada para a busca do maior padrão que se repete. Nele, em primeiro lugar, é iniciado um `for` de 1 (pois o nó raiz já foi criado no método anterior) até o tamanho do texto e, posteriormente, iniciado um `while` que se repete até que nenhum prefixo do texto seja encontrado ou que se atinja um nó folha. Dentro dessa estrutura de repetição, há um `for` que percorre os filhos do nó atual, atualizando seu próprio valor. Em seguida, existe um `for` que encontra o maior prefixo entre a substring e o texto e o utiliza para a construção de novos nós na árvore.

Quando o algoritmo sai do primeiro `while` mencionado, ou seja, quando nenhum prefixo `for` encontrado ou quando se atinge um nó folha, é feita uma verificação que observa se ainda existem caracteres não inseridos na árvore. Caso seja verdadeira, é criado um nó na árvore com esses caracteres restantes.

2.3 - Busca pela maior substring que se repete

Para buscar a maior substring na árvore, utilizam-se os dois métodos mencionados no tópico anterior. O primeiro (`find_biggest_repeated_substring`) é responsável apenas por chamar o método principal (`follow_nodes`) e, posteriormente, retornar os valores encontrados para serem impressos como resultado.

Assim, no método `follow_nodes`, iniciam-se duas variáveis, `start` e `end`, sendo o `first_index` e o `last_index` do nó atual. Além disso, é feita uma verificação: caso o nó atual seja a raiz, a variável `start` terá sido inicializada com “None”, e, então, a ambas as variáveis é atribuído o valor 0. Caso contrário, haveria o lançamento de uma exceção posteriormente na execução.

Em seguida, verifica-se também se o nó atual é um nó folha. Se sim, é feito o retorno, impedindo o método de entrar em loop infinito. Caso contrário, continua-se

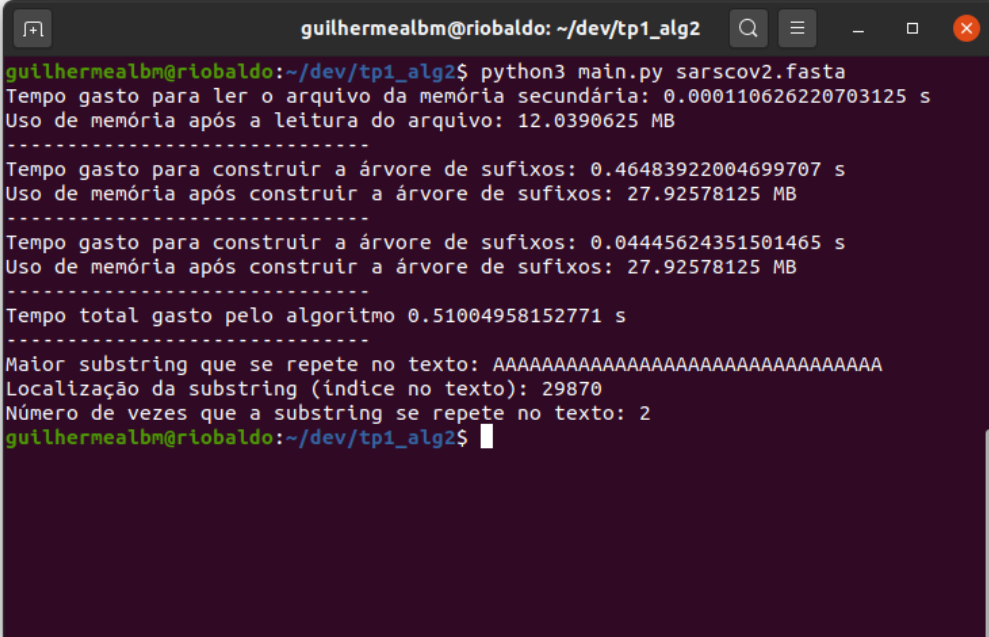
a execução. Assim, recursivamente, o método percorre a árvore à procura da maior substring, retornando ao método `find_biggest_repeated_substring`, finalmente, a maior substring, o índice de sua primeira ocorrência e o números de ocorrência dela no texto, baseando-se no caminho feito na árvore.

3 - Resultados e Desempenho

Como resultado da execução do algoritmo para a sequência genômica do sars-cov-2, tem-se a substring “AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA” como maior encontrada, repetindo-se duas vezes no texto e ocorrendo, pela primeira vez, no índice 29870 do genoma. Além disso, o programa em questão também gera uma série de saídas, para que seja observado o tempo e o uso de memória em cada uma das etapas da execução. Para fins de comparação, as configurações da máquina utilizada para processar o algoritmo são as seguintes:

- CPU: Intel Core i5-8250U @ 1.6Ghz;
- Memória RAM: 8GB;
- Dispositivo de Armazenamento (memória secundária): SSD Western Digital Green M.2 SATA III até 545 MB/s de leitura;
- Sistema Operacional: Ubuntu 20.04 LTS;

Assim, tem-se que:



```
guilhermealbm@riobaldo: ~/dev/tp1_alg2
guilhermealbm@riobaldo:~/dev/tp1_alg2$ python3 main.py sarscov2.fasta
Tempo gasto para ler o arquivo da memória secundária: 0.000110626220703125 s
Uso de memória após a leitura do arquivo: 12.0390625 MB
-----
Tempo gasto para construir a árvore de sufixos: 0.46483922004699707 s
Uso de memória após construir a árvore de sufixos: 27.92578125 MB
-----
Tempo gasto para construir a árvore de sufixos: 0.04445624351501465 s
Uso de memória após construir a árvore de sufixos: 27.92578125 MB
-----
Tempo total gasto pelo algoritmo 0.51004958152771 s
-----
Maior substring que se repete no texto: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Localização da substring (índice no texto): 29870
Número de vezes que a substring se repete no texto: 2
guilhermealbm@riobaldo:~/dev/tp1_alg2$
```

Como pode-se observar, o menor tempo é gasto para ler o arquivo da memória secundária. Posteriormente, tem-se a execução mais custosa para o algoritmo: construir a árvore de sufixos, com 0.46s. Por fim, tem-se a busca na árvore, que consome 0.04s. Com relação ao uso de memória, vê-se que o algoritmo consome relativamente pouco espaço. Em especial, observa-se que a maior parte da memória é consumida para a construção dos nós da árvore, sendo que o

restante fica por conta da alocação do próprio Python e do armazenamento do genoma como texto.

Os tempos obtidos corroboram com o fato de que a maior complexidade assintótica observada nos métodos é na construção da árvore de sufixos, com $O(n^2)$. Assim, 91,1% do tempo de execução é despendido no método em questão.

4 - Conclusão

Em primeiro lugar, é bastante interessante observar, na prática, as aplicações dos algoritmos para manipulação de sequências. Em particular, é importante observar, também, como a complexidade assintótica de um algoritmo interfere diretamente no tempo que sua execução gasta. Exatamente por esse motivo, é ainda mais interessante pensar o quanto algoritmos mais rápidos podem acelerar a resolução de problemas em outras áreas. Mais especificamente em relação ao problema discutido neste trabalho, o algoritmo de Ukkonen certamente foi de grande valia a áreas como a bioinformática, já que, certamente, agiliza o trabalho dos pesquisadores ao manipular grandes genomas.

Por fim, o trabalho também foi de grande importância para sedimentar os conhecimentos a respeito dos algoritmos para manipulação de sequências vistos em aula. Em especial, proporcionou uma grande visão a respeito das dificuldades para a implementação desses algoritmos em uma linguagem de programação, assim como à atenção aos tempos de execução de cada parte da solução de um problema e suas respectivas complexidades assintóticas.

5 - Referências Bibliográficas

Cormen, T., Leiserson, C, Rivest R., Stein, C. Introduction to Algorithms, Third Edition, MIT Press, 2009. Versão Traduzida: Algoritmos – Teoria e Prática 3a. Edição, Elsevier, 2012;

M. Goodrich, R. Tamassia, M. Goldwasser. Data Structures and Algorithms in Java, John Wiley & Sons; 6a Edição, 2014.

Wikipedia. Formato FASTA. Disponível em:
https://pt.wikipedia.org/wiki/Formato_FASTA