Trabalho Prático I - Algoritmos II

Gabriel Victor Carvalho Rocha

Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG) – Belo Horizonte, MG – Brasil

gabrielcarvalho@dcc.ufmg.br - 2018054907

1. Introdução

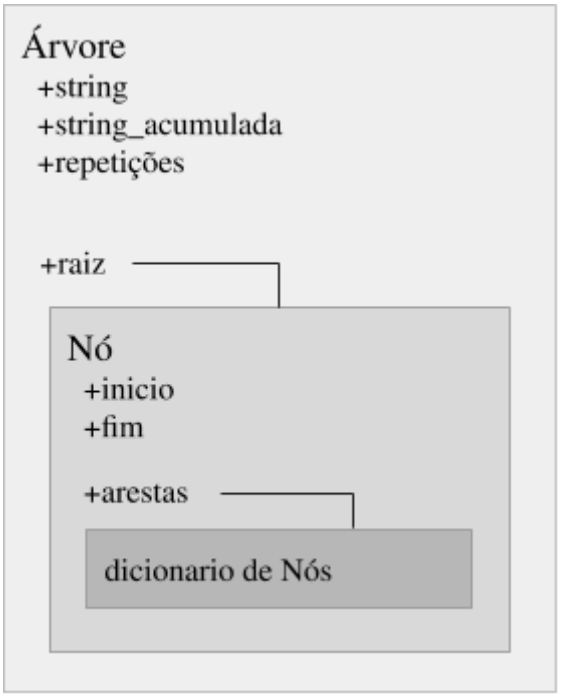
O objetivo principal do trabalho é encontrar a maior substring que ocorre pelo menos duas vezes no genoma do vírus Sars-CoV-2, reportando-o no final. Para isso, foi necessária a implementação de uma estrutura de árvore de sufixo compacta, na qual foi construída a partir da entrada de um arquivo *".fasta"* do genoma citado anteriormente.

2. Implementação

Todo o algoritmo foi desenvolvido na linguagem Python 3.8.5.

2.1. Estruturas de dados

A estrutura de dados utilizada foi uma árvore de sufixo compacta, armazenando a string de entrada (somada ao símbolo de fim *"$")*, a string acumulada, o número de repetições da string acumulada e o Nó raiz da árvore. Cada Nó possui o inicio e o final da sua substring correspondente, além de um dicionário representando suas arestas para outros Nós. Como pode ser observado na imagem abaixo:



2.2. Construção da árvore de sufixo

A construção da árvore inicia-se pegando cada sufixo (do maior para o menor) e chamando uma função recursiva para tratar a inserção a partir da raiz, possuindo três casos possíveis de inserção:

No primeiro caso, quando o Nó atual não possuir aresta para um Nó que tenha prefixo em comum com a *substring*, iremos entrar no primeiro *else*, criando então um Nó com o *slice* correspondente de toda esta *substring*.

Já no segundo caso, existe um Nó que compartilha o prefixo, porém este não corresponde à *substring* inteira deste Nó, então criaremos outros dois Nós, sendo o primeiro a *substring* sem o prefixo e o segundo a *substring* do Nó sem o prefixo, ambos tendo como pai o Nó anterior, contendo apenas o prefixo, entretanto, o segundo Nó precisa preservar os seus filhos anteriores.

O terceiro ocorre quando a *substring* compartilha prefixo com a *substring* do Nó, porém nesse caso o prefixo compartilhado é a *substring* inteira do Nó. Então, entramos recursivamente passando como parâmetro o Nó atual e além disso retirando da *substring* o prefixo comum, logo, nesse caso ele irá tratar a inserção até que se chegue no primeiro ou no segundo caso, onde ela ocorrerá.

2.3. Encontrando a maior *substring* que se repete

Esse algoritmo consiste em analisar cada ramo da raiz separadamente, ou seja, iremos caminhar recursivamente para cada filho da raiz.

Quando entramos na função recursiva, checamos se a *substring* não termina com o símbolo de fim *"$"*, isto é, estamos olhando apenas os filhos que são Nós internos (que possuem dois ou mais filhos). Em seguida recursivamente acumulamos a *substring* em uma variável acumulo_local. Então, após chegar ao último filho interno, precisamos checar se o acúmulo é maior do que o já armazenado pela classe Árvore, caso positivo, string_acumulada é alterada, porém se não for maior, apenas ignoramos e seguimos o resto da recursão procurando sempre um valor maior.

3. Instruções de execução

A execução é feita em qualquer ambiente que suporte Python, sendo realizada da seguinte forma:

python main.py [*nome_arquivo*].fasta

Onde [*nome_arquivo*] é o nome do arquivo do tipo fasta que se deseja passar como entrada, tal como o sarscov2.fasta que foi fornecido. A saída consiste em:

Maior string que se repete: *string_acumulada*
Número de repetições: *repeticoes*
1º ocorrência: [*início, final*]
…
*repeticoes*º ocorrência: [*início, final*]

4. Análise

4.1. Resultados

Após executar o algoritmo para a entrada sarscov2.fasta fornecida, foram obtidos os seguintes valores:

> Maior string que se repete: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
> Número de repetições: 2
> 1º ocorrência: [29870, 29901]
> 2º ocorrência: [29871, 29902]

4.2. Análise empírica

O experimento baseia-se na análise do tempo para execução e da memória utilizada, foi utilizado um computador rodando Windows 64 bits, 8 GB de RAM e um processador i5, obtendo os seguintes valores para cada etapa do algoritmo:

> Início:
>
> Tempo: 0.013 segundos; Memória: 4.04 MB

> Processamento dos sufixos:
>
> Tempo: 0.614 segundos; Memória: 20.52 MB

> Localização da *substring*:
>
> Tempo: 0.795 segundos; Memória: 20.52 MB

Podemos verificar que foram gastos 16.48 MB para armazenar a árvore de sufixo compacta e foi construída em 0.601 segundos. Para realizar a busca na árvore para encontrar a maior *substring* que se repete, foi gasto apenas 0.181 segundos. O tempo total da execução do algoritmo foi de 0.795 segundos.