Relatório - EP(AED1).

Nomes: Rafael Moura de Almeida. N° USP: 112255505. Turma 04.
Rafael Francisco de Freitas Timoteo. N° USP: 12924740. Turma 04.
Guilherme Augusto Arrabal de Souza. N° USP: 5642180. Turma 94.

Tanto com o índice "lista" ou o índice "arvore" passado como parâmetro, o programa armazena as linhas
e as palavras em estruturas diferentes. Cada linha percorrida no arquivo txt com o uso de fgets é armazenada
em uma lista ou árvore para linhas, e cada palavra de cada linha é armazenada em uma lista ou árvore para palavras.
Os nós em que são armazenadas as palavras contém um campo com a string, uma lista ligada com o índice de
todas as linhas em que a palavra aparece no texto, e a quantidade de ocorrências dessa palavra.
Depois de armazenadas em memória as linhas e palavras do texto, é possível realizar uma busca por uma palavra
específica, e o programa irá percorrer a estrutura na qual foram armazenadas as palavras em busca da palavra
digitada pelo usuário, de acordo com a estrutura "lista" ou "arvore" passada como parâmetro para a função main. Caso
encontre o nó que contém a palavra digitada pelo usuário, o programa irá acessar a informação das linhas que contém
essa palavra e percorrer a estrutura que armazena todas as linhas do texto e imprimir as linhas com a palavra digitada
pelo usuário.Dependendo da estrutura ser uma lista ou uma árvore, a velocidade com que o programa percorre a estrutura
em busca da palavra e de cada linha que contém a palavra é afetada. Se for uma lista, é necessário percorrer nó por nó
sequencialmente até encontrar a palavra, e nó por nó até encontrar cada linha com a palavra.
Se for uma árvore, no pior caso para percorrer a árvore que contém as palavras o tempo de busca é proporcional ao logarítmo
na base 2 da quantidade de palavras armazenadas, e o mesmo para percorrer a árvore binária de busca que armazena as linhas.
Esse fato faz com que a escolha da estrutura "arvore" proporcione uma busca muito mais rápida em relação à escolha
da estrutura "lista". O tempo para carregar o arquivo e construir o índice também é drasticamente afetado dependendo da estrutura
selecionada, pois para a construção do índice também são realizadas buscas, pois para cada palavra inserida deve ser feita uma busca
para verificar se aquela palavra já não foi inserida anteriormente, e caso essa busca seja sempre uma busca binária em uma árvore
binária de busca, a construção do índice será muito mais rápida do que realizar essa operação com buscas sequenciais em listas.

A seguir é feito um relatório comparando a eficiência da construção do índice e da busca em arquivos txt utilizando listas ou árvores.
Os arquivos txt são cada vez maiores, mostrando que conforme cresce a quantidade de linhas, a diferença de eficiência entre
listas e árvores fica cada vez maior.

```
Tipo de indice: 'lista'

Arquivo texto: 'texto.txt'


Numero de linhas no arquivo: 13

Tempo para carregar o arquivo e construir o indice: 1.0ms
> busca input


Tempo de busca: 1.000000 ms
> busca output


Tempo de busca: 2.000000 ms

> busca informally

Tempo de busca: 0.000000 ms

Tipo de indice: 'arvore'
Arquivo texto: 'texto.txt'

Numero de linhas no arquivo: 13

Tempo para carregar o arquivo e construir o indice: 0.0ms
> busca input


Tempo de busca: 0.000000 ms
> busca ouput

Tempo de busca: 1.000000 ms

> busca informally
Tempo de busca: 0.000000 ms

Tipo de indice: 'lista'
Arquivo texto: 'sample.txt'
Numero de linhas no arquivo: 66
Tempo para carregar o arquivo e construir o indice: 32.0ms
> busca code
Tempo de busca: 1.000000 ms
> busca the
Tempo de busca: 2.000000 ms

> busca in
Tempo de busca: 2.000000 ms

Tipo de indice: 'arvore'
Arquivo texto: 'sample.txt'
Numero de linhas no arquivo: 66
Tempo para carregar o arquivo e construir o indice: 3.0ms
```

```
> busca code
Tempo de busca: 1.000000 ms
> busca the
Tempo de busca: 2.000000 ms
> busca in
Tempo de busca: 1.000000 ms


Tipo de indice: 'lista'
Arquivo texto: 'new.txt'

Numero de linhas no arquivo: 440
Tempo para carregar o arquivo e construir o indice: 162.0ms
> busca god
Tempo de busca: 4.000000 ms

> busca the
Tempo de busca: 9.000000 ms
> busca is
Tempo de busca: 2.000000 ms

Tipo de indice: 'arvore'
Arquivo texto: 'new.txt'
Numero de linhas no arquivo: 440
Tempo para carregar o arquivo e construir o indice: 4.0ms
> busca god
Tempo de busca: 3.000000 ms
> busca the
Tempo de busca: 8.000000 ms
> busca is
Tempo de busca: 2.000000 ms

Tipo de indice: 'lista'
Arquivo texto: '2021.txt'
Numero de linhas no arquivo: 2021
Tempo para carregar o arquivo e construir o indice: 3109.0ms
> busca god
Tempo de busca: 7.000000 ms

> busca the
Tempo de busca: 45.000000 ms
> busca is
Tempo de busca: 7.000000 ms

Tipo de indice: 'arvore'
Arquivo texto: '2021.txt'
Numero de linhas no arquivo: 2021
Tempo para carregar o arquivo e construir o indice: 38.0ms
> busca god
Tempo de busca: 6.000000 ms
> busca the
Tempo de busca: 33.000000 ms
> busca is
Tempo de busca: 5.000000 ms
```

```
Tipo de indice: 'lista'
Arquivo texto: '3000.txt'
Numero de linhas no arquivo: 3000
Tempo para carregar o arquivo e construir o indice: 5552.0ms
> busca god
Tempo de busca: 10.000000 ms

> busca the
Tempo de busca: 63.000000 ms
> busca is
Tempo de busca: 10.000000 ms

ipo de indice: 'arvore'
Arquivo texto: '3000.txt'
Numero de linhas no arquivo: 3000
Tempo para carregar o arquivo e construir o indice: 73.0ms
> busca god
Tempo de busca: 6.000000 ms
> busca the
Tempo de busca: 42.000000 ms
> busca is
Tempo de busca: 7.000000 ms

Tipo de indice: 'lista'
Arquivo texto: '5000.txt'
Numero de linhas no arquivo: 5000
Tempo para carregar o arquivo e construir o indice: 15002.0ms
> busca god
Tempo de busca: 17.000000 ms

> busca the
Tempo de busca: 123.000000 ms
> busca is
Tempo de busca: 19.000000 ms

Tipo de indice: 'arvore'
Arquivo texto: '5000.txt'
Numero de linhas no arquivo: 5000
Tempo para carregar o arquivo e construir o indice: 181.0ms
> busca god
Tempo de busca: 9.000000 ms
> busca the
Tempo de busca: 67.000000 ms
> busca is
Tempo de busca: 11.000000 ms


Tipo de indice: 'lista'
Arquivo texto: 'bible.txt'
Numero de linhas no arquivo: 100182
Tempo para carregar o arquivo e construir o indice: 1667545.0ms

> busca god
```

```
Tempo de busca: 3425.000000 ms

> busca the
Tempo de busca: 33406.000000 ms
> busca is
Tempo de busca: 4929.000000 ms
> busca fire
Tempo de busca: 458.000000 ms
> busca this
Tempo de busca: 2408.000000 ms
> busca are
Tempo de busca: 2241.000000 ms
> busca if
Tempo de busca: 1301.000000 ms
> busca a
Tempo de busca: 5516.000000 ms
> busca in
Tempo de busca: 8559.000000 ms

Tipo de indice: 'arvore'
Arquivo texto: 'bible.txt'
Numero de linhas no arquivo: 100182
Tempo para carregar o arquivo e construir o indice: 228891.0ms
> busca god
Tempo de busca: 215.000000 ms
> busca the
Tempo de busca: 1769.000000 ms
> busca is
Tempo de busca: 270.000000 ms
> busca fire
Tempo de busca: 24.000000 ms
> busca this
Tempo de busca: 121.000000 ms
> busca are
Tempo de busca: 125.000000 ms
> busca if
Tempo de busca: 66.000000 ms
> busca a
Tempo de busca: 316.000000 ms
> busca in
Tempo de busca: 496.000000 ms
```

As comparações de eficiência entre o uso de listas e árvores para construir o índice e efetuar as buscas foram feitas começando com arquivos
txt com poucas linhas e depois arquivos cada vez maiores. Nas primeiras comparações não há grande diferença com relação à busca, pois os arquivos são muito pequenos, mas no segundo arquivo utilizado, 'sample.txt', com apenas 66 linhas, já pode ser observada uma diferença mais acentuada na construção do índice:
32 ms utilizando listas e 3 ms com árvores. Na terceira comparação, essa diferença fica ainda mais acentuada, em que foi utilizado o arquivo 'sample.txt' com 440 linhas:

162 ms para carregar o arquivo e construir o índice com listas e 4 ms com árvores. Isso se deve ao fato de que durante a construção do índice a operação que mais vezes é realizada
é a operação de busca. Como já foi constatado, para toda inserção de uma palavra é feita uma busca para verificar se aquela palavra já não foi inserida anteriormente. No caso
de a estrutura utilizada para armazenar as palavras ser uma lista, sempre que uma palavra nova é inserida todas as palavras inseridas até aquele ponto devem ser percorridas, o que
faz com que a complexidade temporal cresça no pior caso proporcionalmente ao quadrado da quantidade de palavras diferentes no texto. Caso a estrutura seja uma árvore binária de busca,
qualquer inserção de uma nova palavra exige que apenas log n comparações sejam feitas. Ou seja, a complexidade temporal da construção do índice utilizando
árvore binária de busca como estrutura é, no pior caso, proporcional ao logaritmo na base 2 da quantidade de palavras diferentes no texto. E a diferença entre complexidade quadrática e
complexidade logaritmica na base 2 é gritante, o que justifica a diferença de 158 ms para construção do índice com o arquivo 'sample.txt'. Por outro lado, em relação à busca de uma palavra específica após a construção do índice, apesar de essa operação sempre ser mais rápida com árvores binárias de busca, a diferença não é
tão acentuada mesmo com um arquivo relativamente grande de 5000 linhas, no caso '5000.txt', em que a busca da palavra god leva 123 ms para ser feita com listas e 67 ms com árvores binárias
de busca. Essa diferença de eficiência só pode ser realmente constada com um arquivo muito grande, como o arquivo 'bible.txt' com 100182 linhas, em que para buscar a palavra 'the' e mostrar
todas as ocorrências no texto são necessários 33406 ms utilizando listas, e apenas 1769 ms com árvores. Ou seja, nesse caso a busca com listas foi 18 vezes mais demorada.
E o pior é o tempo levado para construir o índice utilizando listas: um milhão e seiscentos e sessente e sete milissegundos, o que dá quase meia hora, enquanto que a mesma operação com
o uso de árvores levou duzentos e vinte e oito mil milissegundos, que não dá nem 4 minutos. Todos esses fatos evidenciam como a eficiência da árvore binária de busca para realizar buscas
é muito superior às buscas em listas.