ACH2023 – Algoritmo e Estrutura de Dados I
EACH-USP – Segundo Semestre de 2023
Turma 94 – Prof. Flávio Luiz Coutinho

Dimitri Prado
Kevin Tamayose
Rodrigo Lyusei Suguimoto

Relatório – Exercício Programa

Desenvolvemos o nosso programa utilizando uma lista ligada ordenada e uma árvore binária de busca. Usando a função “strcasecmp” foi possível comparar as strings/palavras e usar seu resultado para ordenar alfabeticamente, indiferentemente das letras maiúsculas ou minúsculas. Para a lista ligada, usamos a ordem alfabética crescente, e para a árvore, a partir da raiz de uma árvore/subárvore, os filhos à esquerda são alfabeticamente anteriores e os filhos à direita são alfabeticamente posteriores.

Implementamos os nós tanto da lista como da árvore guardando a palavra, as linhas onde ela aparece, e a quantidade de vezes que aparece (pode ser diferente da quantidade de linhas). Assim, tentamos manter o máximo de semelhança para medir a diferença no desempenho causado somente pelo tipo da estrutura adotada.

Como exemplo, temos os seguintes de resultados usando o texto da bíblia:

```
Tipo de indice: 'lista'
Arquivo texto: 'data/bible.txt'
Numero de linhas no arquivo: 100182
Tempo para carregar o arquivo e construir o indice: 274047 ms
> busca god
Existe(m) 4472 ocorrencia(s) da palavra 'god' na(s) seguinte(s) linha(s):
Tempo de busca: 00570 ms
> busca jesus
Existe(m) 983 ocorrencia(s) da palavra 'jesus' na(s) seguinte(s) linha(s):
Tempo de busca: 00104 ms
> busca love
Existe(m) 311 ocorrencia(s) da palavra 'love' na(s) seguinte(s) linha(s):
Tempo de busca: 00034 ms
> busca a
Existe(m) 8233 ocorrencia(s) da palavra 'a' na(s) seguinte(s) linha(s):
Tempo de busca: 00846 ms
> busca the
Existe(m) 64185 ocorrencia(s) da palavra 'the' na(s) seguinte(s) linha(s):
Tempo de busca: 04954 ms
> fim

Tipo de indice: 'arvore'
Arquivo texto: 'data/bible.txt'
Numero de linhas no arquivo: 100182
Tempo para carregar o arquivo e construir o indice: 243305 ms
> busca god
Existe(m) 4472 ocorrencia(s) da palavra 'god' na(s) seguinte(s) linha(s):
Tempo de busca: 00471 ms
> busca jesus
Existe(m) 983 ocorrencia(s) da palavra 'jesus' na(s) seguinte(s) linha(s):
Tempo de busca: 00102 ms
> busca love
Existe(m) 311 ocorrencia(s) da palavra 'love' na(s) seguinte(s) linha(s):
Tempo de busca: 00034 ms
> busca a
Existe(m) 8233 ocorrencia(s) da palavra 'a' na(s) seguinte(s) linha(s):
Tempo de busca: 00769 ms
> busca the
Existe(m) 64185 ocorrencia(s) da palavra 'the' na(s) seguinte(s) linha(s):
Tempo de busca: 05060 ms
> fim
```

Ocultamos as linhas de ocorrência pois são muitas e não são necessários para análise do desempenho nesse exemplo. A primeira diferença que podemos notar é no tempo para carregar o arquivo e construir o índice, sendo mais rápido no caso da árvore, cerca de 30 segundos de diferença. No geral, buscando as mesmas palavras nos dois tipos de índices, a árvore é mais rápida mas não no nível tão considerável.

Podemos ver que na busca, o tempo empregado varia por uma outra razão vendo os seguintes resultados:

```
Tipo de indice: 'lista'
Arquivo texto: 'data/bible.txt'
Numero de linhas no arquivo: 100182
Tempo para carregar o arquivo e construir o indice: 297677 ms
> busca god
Existe(m) 4472 ocorrencia(s) da palavra 'god' na(s) seguinte(s) linha(s):
Tempo de busca: 00002 ms
> busca jesus
Existe(m) 983 ocorrencia(s) da palavra 'jesus' na(s) seguinte(s) linha(s):
Tempo de busca: 00001 ms
> busca love
Existe(m) 311 ocorrencia(s) da palavra 'love' na(s) seguinte(s) linha(s):
Tempo de busca: 00001 ms
> busca a
Existe(m) 8233 ocorrencia(s) da palavra 'a' na(s) seguinte(s) linha(s):
Tempo de busca: 00002 ms
> busca the
Existe(m) 64185 ocorrencia(s) da palavra 'the' na(s) seguinte(s) linha(s):
Tempo de busca: 00001 ms
> fim

Tipo de indice: 'arvore'
Arquivo texto: 'data/bible.txt'
Numero de linhas no arquivo: 100182
Tempo para carregar o arquivo e construir o indice: 247396 ms
> busca god
Existe(m) 4472 ocorrencia(s) da palavra 'god' na(s) seguinte(s) linha(s):
Tempo de busca: 00000 ms
> busca jesus
Existe(m) 983 ocorrencia(s) da palavra 'jesus' na(s) seguinte(s) linha(s):
Tempo de busca: 00001 ms
> busca love
Existe(m) 311 ocorrencia(s) da palavra 'love' na(s) seguinte(s) linha(s):
Tempo de busca: 00000 ms
> busca a
Existe(m) 8233 ocorrencia(s) da palavra 'a' na(s) seguinte(s) linha(s):
Tempo de busca: 00000 ms
> busca the
Existe(m) 64185 ocorrencia(s) da palavra 'the' na(s) seguinte(s) linha(s):
Tempo de busca: 00000 ms
> fim
```

Aqui, invés de ocultar as linhas de ocorrências, comentamos a parte de impressão no código fonte em si. Ou seja, podemos concluir que no caso da busca, o que demandava tempo era a impressão, já que o mesmo era proporcional a quantidade de ocorrências. Podemos ver uma diferença de 1 ms na busca que é bem pequena, mas pode ser maior no caso de computadores com menor desempenho. O grande impacto está então na criação dos índices do arquivo de texto.

Realizamos outros testes, mudando a primeira palavra que aparece no texto. Como a nossa árvore não é AVL, observamos uma melhora ou piora no desempenho devido ao balanceamento da árvore resultante. Quando a primeira palavra do texto fica alfabeticamente no início ou no final em comparação as outras palavras, a árvore fica desbalanceada e o tempo para criação dos índices é muito pior.

Tivemos um teste que não executou o programa apenas no caso da árvore, usando o arquivo “br-sem-acentos.txt” que é o dicionário português sem acentuação. Como ela

lista as palavras uma em cada linha, no total havia 245.366 linhas, e supomos que o problema seja com o limite de memória do programa, já que colocando uma impressão para o número da iteração atual, o programa fecha quando chega próximo de 15.000, faltando ainda muitas palavras para ser inserida. Curiosamente no caso da lista ela é executada sem problemas, e a busca funciona corretamente.

Um outro ponto a ressaltar é a compatibilidade com a língua portuguesa, testamos alguns métodos como a biblioteca “locale.h” e “wchar.h”. Foi possível resolver para o windows mas mesmo dentro dela, o problema varia de acordo com a configuração do terminal. Já no caso do linux não encontramos uma solução global que funcione corretamente.