ACH2023 - Algoritmos e Estruturas de Dados I

Relatório do Exercício Programa - Indexador e Buscador de Palavras

Filipe Bison de Souza	14570653
Pedro Losso Quintans	14603120
Rodrigo Gonçalves Cardoso	14658330

Como as estruturas de dados foram pensadas:

Lista - Para a organização do index do texto em lista, considera-se alguns pontos para a escolha do tipo de lista e da forma de fazê-lo.
Primeiramente foi buscado um meio termo entre o tempo de indexação e o tempo de busca, priorizando um tempo de busca eficiente por ser realizado mais de uma vez por execução do programa. Porém, não foi deixado de lado a eficiência da indexação, dado que uma demora nesse ponto ocasionaria uma barreira pro usuário realizar operações de busca no texto desejado.
Tendo em vista os pontos anteriormente levantados, uma lista ligada simples foi a escolhida para o projeto, isso principalmente por conta de um tempo de indexação menos oneroso para o usuário.

Árvore - Para o index do texto em árvore, foi utilizado uma árvore binária de busca ordenada, devido sua melhor performance na busca. Além disso, buscando o melhor balanceamento da árvore, mas entendendo também que manter ela 100% balanceada exige um custo de tempo de indexação, definimos que a raiz da árvore seria a letra M, encontrada na metade do alfabeto, a qual seria marcada com o número de aparição zero, para não ser considerada como uma palavra na busca.

Como ocorre a indexação:

Lista - Sobre a indexação, ela é realizada da seguinte forma:
O espaço para uma lista ligada de palavras e uma lista de linhas é alocado em memória. Após isso, o arquivo de texto é percorrido linha a linha, processo necessário para armazenar todas as palavras arquivo "*.txt" na estrutura definida pra tal, tratando casos de novas palavras e palavras já existentes. Por fim, depois de todas as palavras serem devidamente indexadas, uma estrutura para guardar a lista de palavras e de linhas é alocada e populada com as respectivas estruturas.

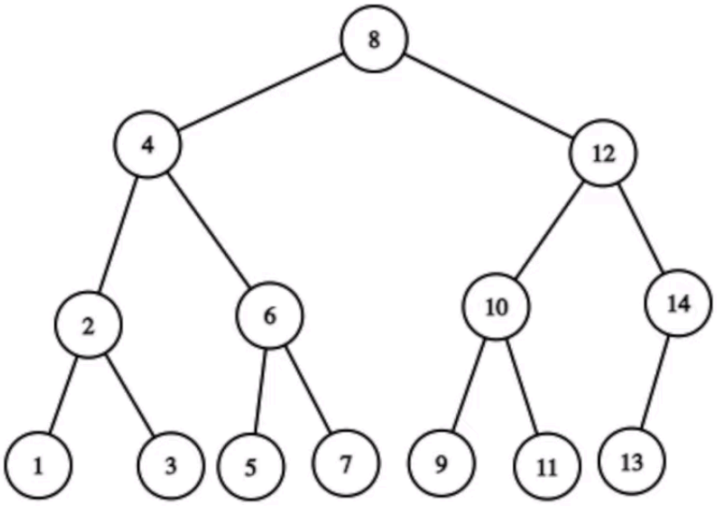
Árvore - A indexação é realizada por meio de uma iteração entre as linhas do arquivo, durante a iteração de linhas é realizado também o salvamento das linhas em uma lista sequencial, após, o código percorre pelas palavras da

linha. Cada palavra será adicionada a árvore e caso já tenha sido inserida, guarda o índice da linha em um array na estrutura, somando um ao seu número de aparições.

Questões que surgiram e como foram resolvidas:

- ***Como manter a lista mais balanceadas sem honerar a indexação?***
  Quanto mais balanceada a árvore mais rápido é possível realizar a busca, porém para manter ela balanceada temos um problema, pois em certos casos a rotação de certos nós, exige que o algorítimo altere demais a lista, sendo muito custoso para o tempo de indexação.

  Por exemplo ao tentar adicionar o zero na árvore a baixo, mantendo-a a mais balanceada possível, seria necessário trocar todos os nós de lugar.



  Visto isso, foi descartada a ideia da árvore LDL, mas para não perder o mínimo do balanceamento, foi decidido colocar o M, como raíz da árvore por ser a letra do meio do alfabeto.

  Realizamos um teste sem o uso do M, e outro com, e mesmo num texto pequeno o uso desse mecanismo em um dos testes chegou a diminuir o tempo de busca em 80%.

- ***Quando a estrutura escolhida foi a árvore, qual estrutura devemos utilizar para armazenar as linhas?***
  Para chegar a uma conclusão, colocamos no txt o texto da página do Brasil na Wikipedia e aumentamos ele até chegar em 50 mil linhas.
  .
  Em seguida, realizamos testes com diferentes tipos de estrutura:
  Armazenamento de linhas em lista sequencial

```
rodigo@PCzindoRod:~/EP-BISON$ ./meuPrograma teste.txt arvore
Tipo de indice: 'arvore'
Arquivo de texto: 'teste.txt'
Numero de linhas no arquivo: 52000
Tempo para carregar o arquivo e construir o indice: 462.467000 ms
```

Busca com linhas em lista sequencial:

```
51656 : Solnik, Alex (1987). Os pais do cruzado contam por que não deu certo. Po
Tempo de busca: 0.122000 ms
```

Armazenamento de linhas em lista dinâmica:

```
Tipo de indice: 'arvore'
Arquivo de texto: 'teste.txt'
Numero de linhas no arquivo: 51724
Tempo para carregar o arquivo e construir o indice: 470.064000 ms
```

Busca com linhas em lista dinâmica:

```
51656 : Solnik, Alex (1987). Os pais do cruzado contam por que não deu certo
Tempo de busca: 6.243000 ms
```

.

Optamos, então, por utilizar lista sequencial neste caso.

- ***Qual tipo de lista deve ser utilizada para armazenar as palavras?***
  A primeira ideia foi utilizar uma lista sequencial ordenada, para que fosse possível usar a busca binária e obter grande eficiência na busca.
  Porém, na hora de realizar a indexação, seria necessário passar duas vezes pelo txt, o que ocasionou em um tempo muito grande para tal tarefa:

```
Tipo de indice: 'lista'
Arquivo de texto: 'teste.txt'
Numero de linhas no arquivo: 51724
Tempo para carregar o arquivo e construir o indice: 20506.153000 ms
```

A busca estava realmente boa:

```
51656: Solnik, Alex (1987).
Tempo de busca: 0.124000 ms
```

.

Realizamos os testes com lista ligada.

Indexação:

```
Tipo de indice: 'lista'
Arquivo de texto: 'teste.txt'
Numero de linhas no arquivo: 51723
Tempo para carregar o arquivo e construir o indice: 12504.478000 ms
```

Busca:

```
51656 : Solnik, Alex (1987).
Tempo de busca: 0.602000 ms
```

.

Optamos então por utilizar a lista ligada como estrutura para armazenar as palavras, uma vez que 7 segundos (diferença no tempo de indexação) é mais perceptível que 0.5 milissegundos (diferença no tempo de busca).
Portanto, a lista acabou ficando pior para indexar e para buscar.

Testes de tempo

Metodologia: indexação e pesquisa de diferentes palavras em diferentes tamanhos de texto.
Com ajuda de uma IA elaboramos um texto de 1000 linhas e repetimos ele até chegar nos tamanhos desejados.
Para saber se o tamanho do texto tem influência ou não na busca, independente da impressão das linhas, colocamos uma palavra (“comando”) na linha 500 (metade do texto antes de se repetir) para analisar e comparar os resultados. A palavra foi colocada no meio do texto para simular um caso médio na lista.
(Na indexação com 500 linhas, a palavra “comando” foi colocada na linha 250)
.

N.º Linhas	Indexação - Lista (em ms)	Indexação - Árvore (em ms)	Palavra	Busca - Lista (em ms)	Busca - Árvore (em ms)
50k	4735.619000	429.649000	eclipses	1.655000	0.378000
50k	-	-	comando	0.266000	0.037000
40k	3436.607000	295.998000	cinema	5.657000	4.583000
40k	-	-	comando	0.223000	0.048000
30k	2517.975000	217.166000	areia	1.573000	1.085000
30k	-	-	comando	0.216000	0.050000
20k	1750.386000	162.471000	pilotos	0.348000	0.076000
20k	-	-	comando	0.260000	0.052000
10k	1006.781000	68.260000	habitats	0.393000	0.188000
10k	-	-	comando	0.203000	0.052000
5k	416.182000	35.969000	mágicas	0.342000	0.130000
5k	-	-	comando	0.203000	0.051000
1000	85.079000	7.705000	selvagem	0.137000	0.060000
1000	-	-	comando	0.205000	0.054000

N.º Linhas	Indexação - Lista (em ms)	Indexação - Árvore (em ms)	Palavra	Busca - Lista (em ms)	Busca - Árvore (em ms)
500	36.138000	3.462000	jovens	0.120000	0.095000
500	-	-	comando	0.147000	0.054000

Discussão dos resultados obtidos

Analisando os resultados das buscas, podemos concluir que:

- A árvore é mais rápida que a lista na busca, as buscas das palavras, em especial “comando”, evidenciaram isso.
- O tempo de busca está relacionado com a posição da palavra na estrutura, e não com o tamanho do arquivo. Os tempos de busca da palavra “comando” evidenciaram isso.
- A velocidade de busca da lista é totalmente dependente da posição em que a palavra se encontra no texto, se ela demora para realizar a primeira aparição ou não. Diferente da árvore binária de busca, que por ter uma lógica de ordenação, depende dessa lógica.
- O tempo de impressão (percorrer a lista sequencial de linhas e imprimir elas) costuma demorar mais do que o tempo da busca de fato.

Analisando os resultados das indexações, podemos concluir que:

- A velocidade de indexação em árvore é mais rápida do que a de lista.
- O tempo de indexação é diretamente proporcional ao tamanho do arquivo.

Conclusão

Através dos dados obtidos em nosso trabalho, pode-se concluir que a árvore binária de busca é uma estrutura mais eficiente que a lista ligada, tanto na indexação quanto na busca.
Porém, os resultados envolvem muitas variáveis que devem ser sempre consideradas na hora de escolher uma estrutura ou outra.