UNIVERSIADE FEDERAL DE OURO PRETO – *UFOP*

CURSO DE CIÊNCIA DA COMPUTAÇÃO

Gabriel Fernandes Niquini

Filipe Ramos

Rafael Diniz

Trabalho Prático I – Pesquisa Externa

Ouro Preto
Setembro de 2020

Gabriel Fernandes Niquini

Filipe Ramos

Rafael Diniz

Trabalho Prático I – Pesquisa Externa

Ouro Preto
Setembro de 2020

RESUMO

O trabalho consiste em criar a lógica e desenvolver um programa, em linguagem escolhida C/C++, para exemplificar e praticar o conteúdo de pesquisa externa, além de um estudo sobre a complexidade de desempenho dos métodos: Acesso sequencial indexado, Árvore Binaria, Árvore B e Árvore B*.

1 INTRODUÇÃO

O trabalho consiste em primeiramente desenvolver códigos em linguagem C/C++ de logicas de pesquisa externa, sendo elas: Acesso sequencial indexado, Árvore Binaria, Árvore B e Árvore B*. Logo após isso os testes para cada um dos métodos com 100, 1000, 10000, 100000 e 1000000 de registros em ordem ascendente, descendente e desordenados aleatoriamente. Um total de 60 testes diferentes.

Como não foi possível a realização de teste por problemas de segmentação de memória aos quais não tiveram solução a tempo do prazo de entrega, não existem dados para serem comparados.

1- Acesso sequencial indexado:

Método teoricamente menos eficaz que os outros para grandes volumes de dados, já que sua lógica consiste em procurar bloco a bloco até encontrar o registro desejado. Seu melhor caso é onde você encontra-o no primeiro bloco e o pior no último.

2- Árvore binaria:

Diferentemente do método anterior, esse método é muito mais eficaz, já que sua complexidade é bem menor no quesito busca, como ela é dividida em lados maiores e menores que o registro anterior a busca é feita de forma logarítmica onde sempre é reduzida mais e mais até achar o registro desejado, Sendo ótima tanto para muitos quanto para pouco números de registros.

3- Árvore B:

Quando falando da árvore B e suas possibilidades, percebemos que, por se tratar de um método onde os registros podem ser organizados de maneiras mais livre, claro que seguindo uma lógica, a disposição dos mesmos tende a facilitar a procura já que uni o melhor dos dois mundo, uma separação entre lados maiores e maiores registros por bloco e uma sequência de registros em cada um desses blocos, fazendo com que a procura seja mais rápida para grandes quantidade de dados.

4- Árvore B*:

Por fim, o método que mais agrega características dos outros. Ela tem uma constituição muito parecida com a árvore B, porém todas as divisões de blocos, menos a última, são somente uma indexação para os verdadeiros registros, que ficam armazenados no final da árvore B*. Ela por sua vez é tão facilmente navegável que as comparações maiores acontecem somente ao chegar no final dela, antes disso são somente “orientações de qual lado seguir”, necessitando de bem menos espaço de memória, fazendo com que seja ótima para gigantes quantidades de dados.

2 METODOS

2.1 Teste ASI | 100 | ascendente = ?

2.2 Teste ASI | 1000 | ascendente = ?

2.3 Teste ASI | 10000| ascendente = ?

2.4 Teste ASI | 100000 | ascendente = ?

2.5 Teste ASI | 1000000 | ascendente = ?

2.6 Teste ASI | 100 | descendente = ?

2.7 Teste ASI | 1000 | descendente = ?

2.8 Teste ASI | 10000| descendente = ?

2.9 Teste ASI | 100000 | descendente = ?

2.10 Teste ASI | 1000000 | descendente = ?

2.11 Teste ASI | 100 | aleatório = ?

2.12 Teste ASI | 1000 | aleatório = ?

2.13 Teste ASI | 10000 | aleatório = ?

2.14 Teste ASI | 100000 | aleatório = ?

2.15 Teste ASI | 1000000 | a aleatório = ?

2.16 Teste Árvore binaria | 100 | ascendente = ?

2.17 Teste Árvore binaria | 1000 | ascendente = ?

2.18 Teste Árvore binaria | 10000| ascendente = ?

2.19 Teste Árvore binaria | 100000 | ascendente = ?

2.20 Teste Árvore binaria | 1000000 | ascendente = ?

2.21 Teste Árvore binaria | 100 | descendente = ?

2.22 Teste Árvore binaria | 1000 | descendente = ?

2.23 Teste Árvore binaria | 10000| descendente = ?

2.24 Teste Árvore binaria | 100000 | descendente = ?

2.25 Teste Árvore binaria | 1000000 | descendente = ?

2.26 Teste Árvore binaria | 100 | aleatório = ?

2.27 Teste Árvore binaria | 1000 | aleatório = ?

2.28 Teste Árvore binaria | 10000 | aleatório = ?

2.29 Teste Árvore binaria | 100000 | aleatório = ?

2.30 Teste Árvore binaria | 1000000 | a aleatório = ?

2.31 Teste Árvore B | 100 | ascendente = ?

2.32 Teste Árvore B | 1000 | ascendente = ?

2.33 Teste Árvore B | 10000| ascendente = ?

2.34 Teste Árvore B | 100000 | ascendente = ?

2.35 Teste Árvore B | 1000000 | ascendente = ?

2.36 Teste Árvore B | 100 | descendente = ?

2.37 Teste Árvore B | 1000 | descendente = ?

2.38 Teste Árvore B | 10000| descendente = ?

2.39 Teste Árvore B | 100000 | descendente = ?

2.40 Teste Árvore B | 1000000 | descendente = ?

2.41 Teste Árvore B | 100 | aleatório = ?

2.42 Teste Árvore B | 1000 | aleatório = ?

2.43 Teste Árvore B | 10000 | aleatório = ?

2.44 Teste Árvore B | 100000 | aleatório = ?

2.45 Teste Árvore B | 1000000 | a aleatório = ?

2.46 Teste Árvore B* | 100 | ascendente = ?

2.47 Teste Árvore B* | 1000 | ascendente = ?

2.48 Teste Árvore B* | 10000| ascendente = ?

2.49 Teste Árvore B* | 100000 | ascendente = ?

2.50 Teste Árvore B* | 1000000 | ascendente = ?

2.51 Teste Árvore B* | 100 | descendente = ?

2.52 Teste Árvore B* | 1000 | descendente = ?

2.53 Teste Árvore B* | 10000| descendente = ?

2.54 Teste Árvore B* | 100000 | descendente = ?

2.55 Teste Árvore B* | 1000000 | descendente = ?

2.56 Teste Árvore B* | 100 | aleatório = ?

2.57 Teste Árvore B* | 1000 | aleatório = ?

2.58 Teste Árvore B* | 10000 | aleatório = ?

2.59 Teste Árvore B* | 100000 | aleatório = ?

2.60 Teste Árvore B* | 1000000 | a aleatório = ?

3 CONSIDERAÇÕES FINAIS

Todos os métodos têm suas utilidades e funções, mesmo que uns sejam melhores que outros para funções específicas, sendo assim, não é possível descartar nenhum dos métodos para atividades diversas.

Na teoria as arvores B e B* tem a vantagem sobre os outros métodos, por serem mais uteis na maioria das vezes, já que são capazes de lidar melhor com maiores quantidades de dados do que os outros.

O acesso sequencial indexado é o “pior” deles por não ser otimizado para grandes volumes de informações, mesmo que seja uma ótima maneira de separar dados simples de forma fácil.

Já a árvore binaria, a mais conhecida de todas, é utilizada em situações onde as outras árvores são de mais e quando o acesso sequencial indexado é insuficiente.