CEFET-MG - Campus II
Departamento de Engenharia de Computação
Laboratório de Algoritmos e Estruturas de Dados II

Prática II

Implementação em JAVA do TAD Symetric Binary B-Tree (SBB)

Alunos: Antônio Augusto Diniz Sousa
Professor orientador: Thiago de Souza Rodrigues

26 de Agosto
2018

CEFET-MG - Campus II
Departamento de Engenharia de Computação
Laboratório de Algoritmos e Estruturas de Dados II

Prática II

Implementação em JAVA do TAD Symetric Binary B-Tree (SBB)

Relatório da prática II apresentado à Disciplina de Laboratório de Algoritmos e Estruturas de Dados II do Curso de Engenharia de Computação do Centro Federal de Educação Tecnológica de Minas Gerais, como requisito parcial para conclusão da disciplina.

Alunos: Antônio Augusto Diniz Sousa

Professor orientador: Thiago de Souza Rodrigues

26 de Agosto
2018

Conteúdo

1 Apresentação

A tarefa dessa prática foi testar o comportamento da Árvore SBB em diversos casos. Utilizou-se a implementação do Autor Ziviani, disponível nesse link ou acessado diretamente em `http://www2.dcc.ufmg.br/livros/algoritmos-java/implementacoes-05.php`.

Utilizando a linguagem *Java*, criou-se 18 arvores, sendo 9 delas utilizando uma inserção de dados ordenados, e as outras 9, inserção de dados aleatórios. Dentre as noves de cada modelo, a primeira possuía 10000 elementos, a segunda 20000, e assim por diante, até a 9ª, com 90000 elementos.

Analisou-se diante dessas árvores o tempo gasto para pesquisar um item ausente na estrutura e as comparações necessárias para tirar tal conclusão. Os resultados seguem adiante nesse relatório e todos os códigos fontes podem ser acessados clicando aqui ou acessando em `https://gitlab.com/antonioaads/LAEDII`.

2 Descrição de atividades

Após a execução do código, que não iremos tratar especificamente nesse relatório, pois o mesmo se encontra no repositório citado na introdução devidamente comentado e organizado, tirou-se os resultados da Tabela 1.

Tabela 1: Resultados

Número de Itens	**Ordenada**		**Aleatória**	
	Comparações	Tempo (ns)	Comparações	Tempo (ns)
10000	55	28679	37	6421
20000	58	10701	40	7705
30000	64	10273	42	8989
40000	61	10273	43	17549
50000	64	11985	46	7277
60000	67	11129	46	7277
70000	67	15409	49	17977
80000	64	8989	52	3852
90000	76	10701	52	6421

É de conhecimento que a a medida de tempo pode variar por n motivos, sendo alguns,

- Arquitetura da máquina;
- Configuração da máquina;
- Atividades rodando de maneira paralela;
- Modos de economia ou gerenciamento de energia

dentre diversos outros. A fim de especificar melhor, e tentar minimizar erros e diferenças do tempo mensurado devido aos motivos citados, utilizamos o mesmo computador para todos os testes, com especificação da Tabela 2.

Tabela 2: Hardware Utilizado

Modelo	Dell Vostro 3450
Memória RAM	8GB
Sistema Operacional	Ubuntu 18.04 LTS
Processador	Intel Core i5 6ª Geração

Utilizou-se a ferramenta o *LibreOffice Calc* para plotar os gráficos ilustrados na Figure 2 e Figure 1.

Figura 1: Representação do gráfico n x Número de comparações

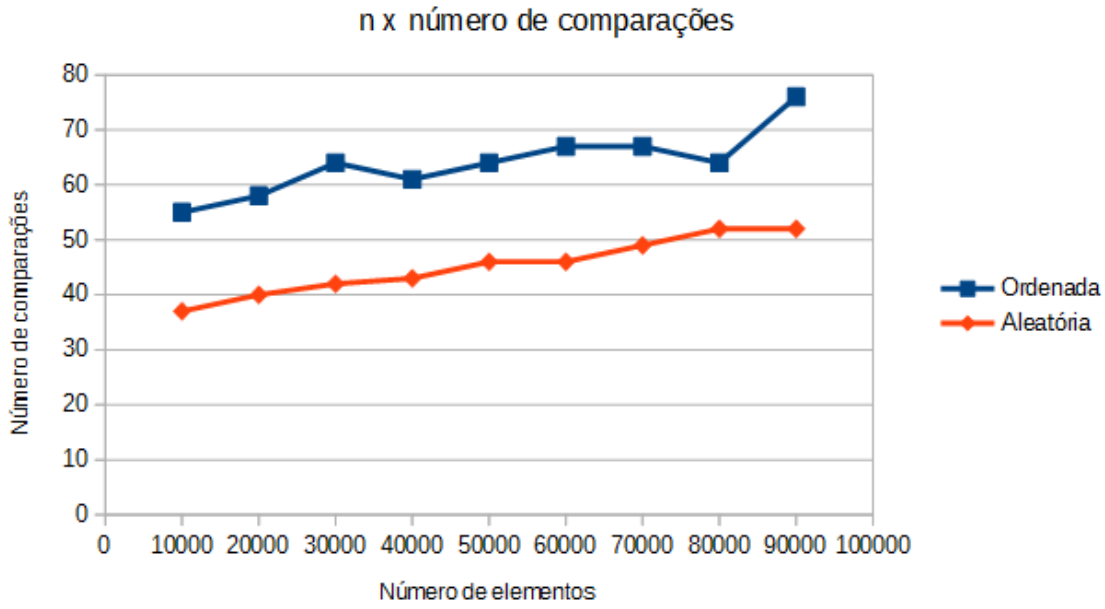
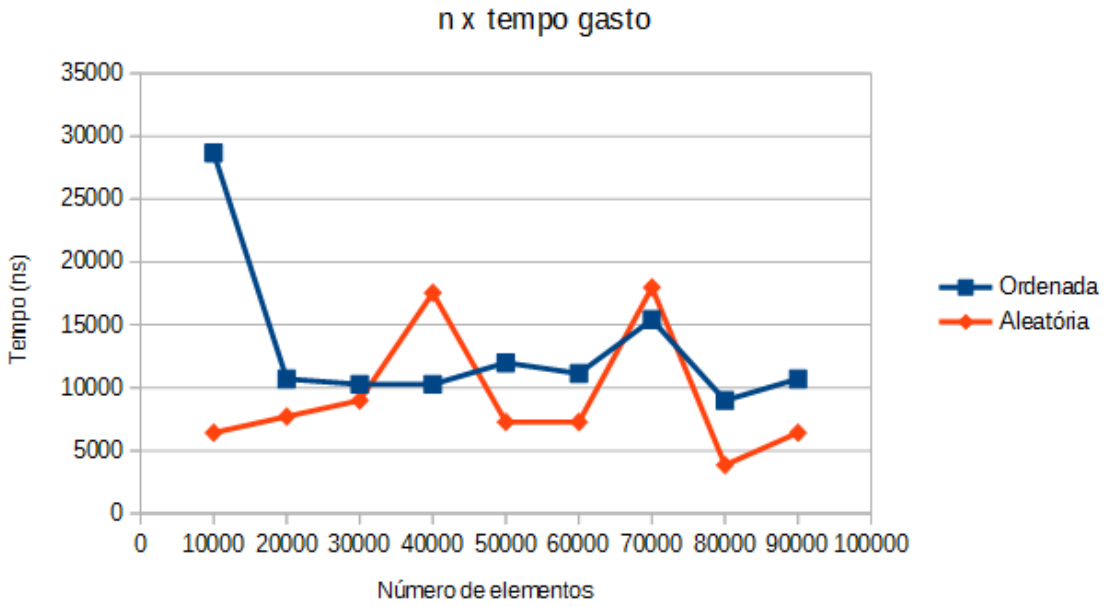


Figura 2: Representação do gráfico n x Tempo Gasto



2.1 Análise

A Arvore Binária SBB possui uma complexidade de tempo para busca $O(logn)$ tanto para o **caso médio** quanto para o **pior caso**.

Diante disso, percebemos que a onda do número de comparações dos dados ordenados é bem semelhante aos dados aleatórios, com a presença de um pequeno *offset*, porém mantendo semelhança a uma curva característica da função $logn$.

O gráfico que relaciona o tempo gasto e o número de elementos não ficou muito apresentável, por motivo que imagina-se ser relacionado aos problemas citados anteriormente. Mesmo assim, podemos concluir que não há muita diferença de tempo dos dados ordenados com os aleatórios, motivo muito bom, pois mantêm-se um custo com crescimento $logn$ para quaisquer entradas.

3 Comparação com a Prática 1

A diferença principal na prática da árvore binária de busca com a SBB se dá no pior caso, que no caso da árvore binária é $O(n)$ e no caso da SBB é $O(logn)$.

O número de comparações cresce de maneira muito mais suave na SBB do que na binária de busca, mesmo para dados com proporções maiores. O motivo disso é algoritmo por trás da SBB, que tenta balancear a árvore conforme é inserido os dados, deixando a árvore, em geral, menos profunda que a árvore binária, logo, mais veloz para buscar um item.

4 Conclusão

A SBB possui uma complexidade maior no algoritmo e na inserção de dados, porém, dependendo da aplicação pode funcionar de maneira muito mais eficiente do que a árvore binária de busca.

A escolha depende da aplicação, e deve ser escolhida minunciosamente, analisando custo para produção do algoritmo, custo de implementação, dentre outros diversos parâmetros.