CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

BRUNO NASCIMENTO DAMACENA

RAMON GRIFFO COSTA

Prática 02 – Implementação em JAVA do TAD Symetric Binary B-Tree (SBB)

Trabalho apresentado à disciplina de Laboratório de Algoritmos e Estrutura de Dados II do curso de Engenharia de Computação do Departamento de Computação do CEFET-MG

Professor Thiago de Souza Rodrigues.

Belo Horizonte
Março de 2018

1. Utilizando o Netbeans, crie um projeto chamado Prática02
2. Implemente a classe Item, como especificada abaixo para ser utilizada no T.A.D.
3. Implemente uma classe chamada ArvoreSBB para manipular uma árvore binária de pesquisa onde os nós da árvore são objetos da classe No

   Os três primeiros items foram implementados e podem ser encontrados no arquivo zip do código fonte, que foi enviado anexo a esse relatório. O código fonte também pode ser encontrado no repositório do GitHub (https://github.com/BrunoDamacena/SBBSearchTree).
4. Realizar os seguintes experimentos:
   (a) gerar árvores a partir de n elementos ORDENADOS, com n variando de 10.000 até 100.000, com intervalo de 10.000.
   (b) gerar árvores a partir de n elementos ALEATÓRIOS (long j = obj.nextInt()), com n variando de 10.000 até 100.000, com intervalo de 10.000.
   (c) Fazer um único gráfico de n x número de comparações levando em consideração as árvores geradas com inserções ordenadas e aleatórias
   (d) Fazer um único gráfico de n x tempo gasto levando em consideração as árvores geradas com inserções ordenadas e aleatórias

   As árvores foram geradas durante a execução do código. Para gerar a árvore com elementos ordenados, foi criado um ArrayList de Items ordenados de forma crescente, com tamanho *size*. Para gerar a árvore com elementos aleatórios, foi feito um *Collections.shuffle(ArrayList)* antes da inserção na árvore, de modo que os elementos foram inseridos aleatoriamente. Para avaliar a busca, foi simplesmente executada a busca pelo elemento *size + 1*.

   Os testes foram executados 10 vezes, com o resultado de cada teste disponível no arquivo .zip, no diretório *results/raw*.
   Uma vez executados os testes, os resultados foram organizados em tabelas, uma para o número de comparações, outra para o tempo de execução.

Tabela 1: Comparações por Nº de Elementos

COMPARISONS MADE											
-	Output 1	Output 2	Output 3	Output 4	Output 5	Output 6	Output 7	Output 8	Output 9	Output 10	Mean
Ordered 10000	18	18	18	18	18	18	18	18	18	18	18
Ordered 20000	19	19	19	19	19	19	19	19	19	19	19
Ordered 30000	21	21	21	21	21	21	21	21	21	21	21
Ordered 40000	20	20	20	20	20	20	20	20	20	20	20
Ordered 50000	21	21	21	21	21	21	21	21	21	21	21
Ordered 60000	22	22	22	22	22	22	22	22	22	22	22
Ordered 70000	22	22	22	22	22	22	22	22	22	22	22
Ordered 80000	21	21	21	21	21	21	21	21	21	21	21
Ordered 90000	25	25	25	25	25	25	25	25	25	25	25
Ordered 100000	22	22	22	22	22	22	22	22	22	22	22
-	-	-	-	-	-	-	-	-	-	-	-
Random 10000	15	14	13	14	13	14	15	17	14	14	14,3
Random 20000	15	16	15	16	13	13	14	13	15	15	14,5
Random 30000	14	17	14	14	16	14	14	18	16	15	15,2
Random 40000	17	13	14	16	14	17	18	14	17	14	15,4
Random 50000	17	14	19	16	19	17	17	13	16	15	16,3
Random 60000	17	15	17	17	18	16	15	16	16	14	16,1
Random 70000	17	18	16	20	18	17	16	16	20	16	17,4
Random 80000	14	16	13	16	17	16	16	17	17	14	15,6
Random 90000	16	14	17	18	15	17	16	15	15	17	16
Random 100000	19	16	20	16	15	17	15	16	17	17	16,8

Tabela 2: Tempo de Execução por Nº de Elementos

EXECUTION TIME (in ms)											
-	Output 1	Output 2	Output 3	Output 4	Output 5	Output 6	Output 7	Output 8	Output 9	Output 10	Mean
Ordered 10000	224	253	213	271	234	339	265	356	252	268	267,5
Ordered 20000	131	125	125	134	126	126	132	124	127	146	129,6
Ordered 30000	125	117	141	131	116	117	150	119	123	160	129,9
Ordered 40000	598	150	134	143	154	134	134	130	142	161	188
Ordered 50000	166	139	171	501	132	607	160	605	499	154	313,4
Ordered 60000	119	227	112	112	114	114	145	112	127	127	130,9
Ordered 70000	136	138	134	132	140	135	131	138	136	137	135,7
Ordered 80000	114	114	108	110	108	127	112	111	115	113	113,2
Ordered 90000	114	149	112	269	113	137	113	134	119	184	144,4
Ordered 100000	116	116	113	115	110	113	114	111	117	117	114,2
-	-	-	-	-	-	-	-	-	-	-	-
Random 10000	114	163	111	109	111	110	112	114	113	108	116,5
Random 20000	186	202	178	155	150	196	208	179	114	159	172,7
Random 30000	163	161	169	170	161	129	161	154	362	282	191,2
Random 40000	278	522	272	264	260	236	262	262	99	289	274,4
Random 50000	105	101	98	107	97	98	99	98	100	99	100,2
Random 60000	104	110	102	98	162	99	100	96	199	108	117,8
Random 70000	103	103	101	96	95	101	100	112	101	97	100,9
Random 80000	133	133	133	128	129	133	121	129	135	126	130
Random 90000	101	108	96	103	99	98	97	97	142	96	103,7
Random 100000	99	163	106	161	97	107	98	96	106	119	115,4

Com os dados no gráfico, calculamos a média dos testes e reproduzimos os gráficos. O eixo das abscissas representa o número de elementos da árvore, enquanto o eixo das ordenadas representa, no primeiro gráfico, o número de comparações feitas durante a busca, e no segundo gráfico, o tempo de execução da busca.

Gráfico 1: Comparações por Nº de Elementos

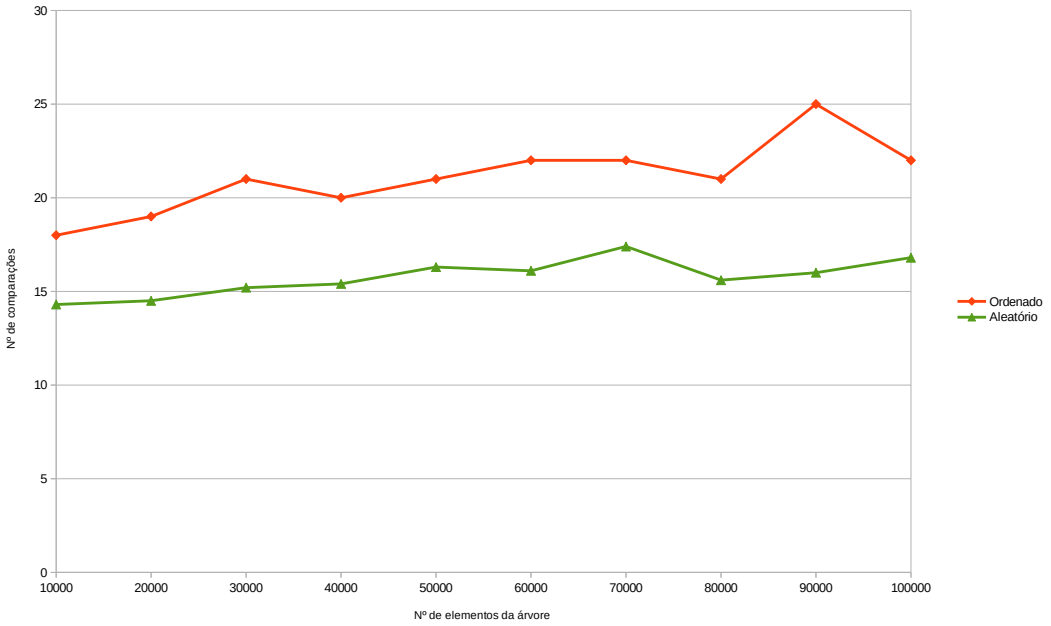
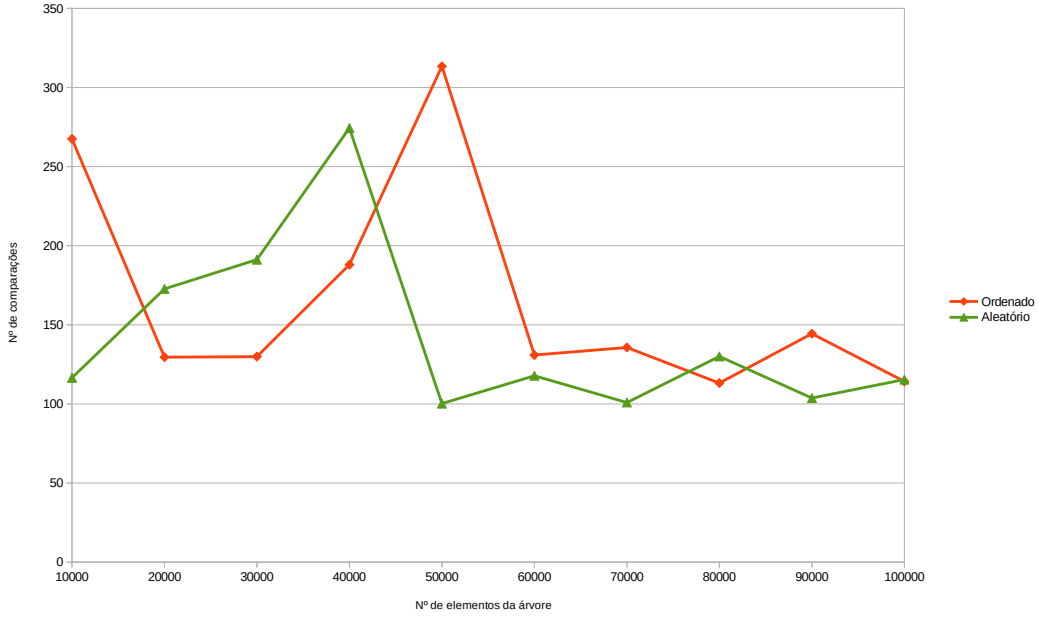


Gráfico 2: Tempo de Execução por Nº de Elementos



5. Explique o comportamento dos gráficos gerados

   No Gráfico 1, é mostrado como o número de comparações é bem próximo, tanto na árvore de inserção ordenada, quanto na árvore de inserção aleatória, ambos próximos de *ln(size)*. Isso acontece porque na árvore SBB, é feito o balanceamento na inserção, tornando a pesquisa sempre *O(ln(n))*.

   No Gráfico 2, fica evidente novamente a imprecisão da utilização do tempo como fator. Graças a uma série de fatores como armazenamento em memória vs. armazenamento em cache, uso do processador do computador utilizado para os testes, imprecisão do método nanoTime(), estes citados também na Prática 01, entre outros, ficou visível que o resultado do gráfico foi inconclusivo.

6. Compare com o resultados obtidos na prática 01

   Para inserção ordenada, fica óbvio a vantagem da árvore SBB sobre a árvore binária, pelo seu fator de balanceamento. Já na inserção aleatória, o resultado da busca na árvore binária foi ligeiramente melhor que na árvore SBB. Por fim, como a diferença no número médio de trocas foi aproximadamente 5 a mais para a árvore SBB, podemos concluir que o desempenho das duas é similar. A nível computacional, a diferença em tempo de execução é indistinguível.