Mike Koch
04/23/2015
EECS 2510 | Non-Linear Data Structures

Comparisons between an AVL Tree and a B-Tree with Nodes Stored on Disk

Using a text file containing all of Shakespeare's works, a C++ program was compiled and ran, building an AVL tree and a B-Tree. Using different values of "t" for the B-Tree instance, metric information was gathered from each tree, which contained the following: Height of the tree

- Number of nodes
- Number of keys used, keys available, and load factor (B-Tree only)
- Number of unique words
- Number of words, including duplicates
- Height of Tree
- Disk reads/writes
- File size
- Elapsed time
- Elapsed time, excluding overhead

These metrics were recorded for the AVL tree and several B-Trees with different "t" values (in square brackets), as shown in the table below.

Tree	Height	Number of nodes	Unique words	Total Words	Disk Reads	Disk Writes
AVL	18	38,004	38,004	883,299	9,952,498	1,073,309
B-Tree [2]	11	21,810	38,004	883,299	8,194,604	948,694
B-Tree [16]	3	1767	38,004	883,299	3,341,408	888,589
B-Tree [32]	2	882	38,004	883,299	2,654,308	885,937
B-Tree [500]	1	62	38,004	883,299	1,800,595	883,480

Tree Type	File Size	Keys Used/Total		Load Factor	Elapsed Time[1]	
AVL	2.74 MB				102,580	96,643
B-Tree [2]	5.58 MB	38,004	65,430	58.08%	100,167	94,230
B-Tree [16]	3.32 MB	38,004	54,777	69.38%	84,420	79,114
B-Tree [32]	3.30 MB	38,004	55,566	68.39%	103,167	97,888
B-Tree [500]	3.65 MB	38,004	61,938	61.36%	819,386	813,699

As expected, the AVL tree required the most reads and writes compared to any of the B-Tree configurations, as there is only one key per node, while the B-Tree configurations had multiple keys per node. However, the AVL tree had the lowest file size of all of the trees, which makes sense as there are no "empty" spots that keys have not filled yet, as each node has exactly one key and there cannot be any "empty" spaces. The number of unique words and total words were consistent across all trees as expected. As the value for t increased for the B-Tree, the number of nodes decreased, which in turn decreased the height of the tree. However, there appears to be a certain value of t that will either make the tree more efficient than other values, as the B-Tree [500] had a larger size than B-Tree [32], which were both smaller than B-Tree [2]. More investigating into B-Tree t values is included in the appendix.

[1] Units are in clock cycles. The value to the left is the total time including overhead. The value to the right is the total time, not counting for overhead (overhead differed during each execution). Times achieved were ran using an Intel® Core™ i5 processor (3.20 GHz)

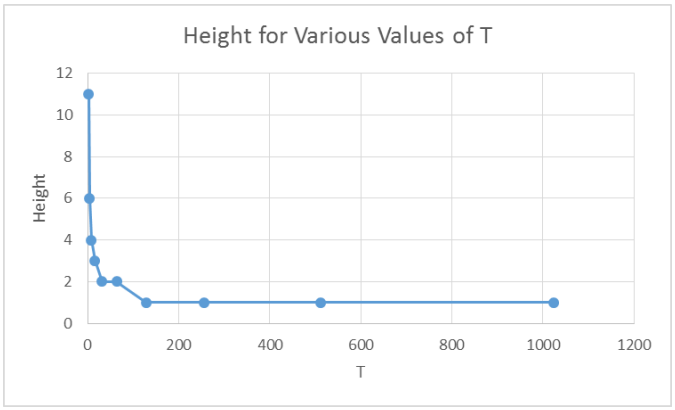
Appendix: B-Tree Analysis For Various Values of T

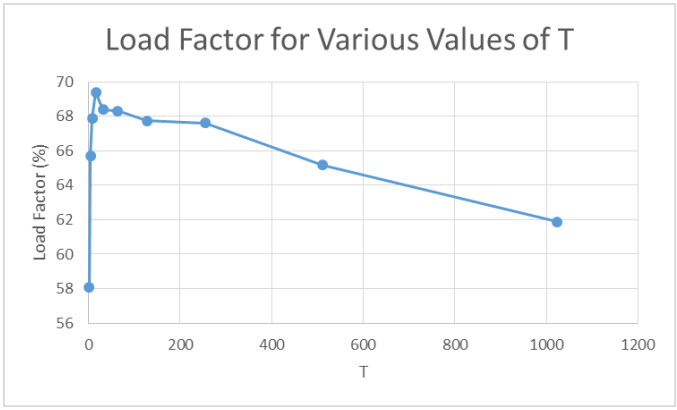
To further investigate how efficient B-Trees can be, various version of the B-Tree program were compiled with different values of T. A PowerShell script than ran each version of the program and the results were stored in a text file. Each run, along with basic information about the tree is displayed in the table below.

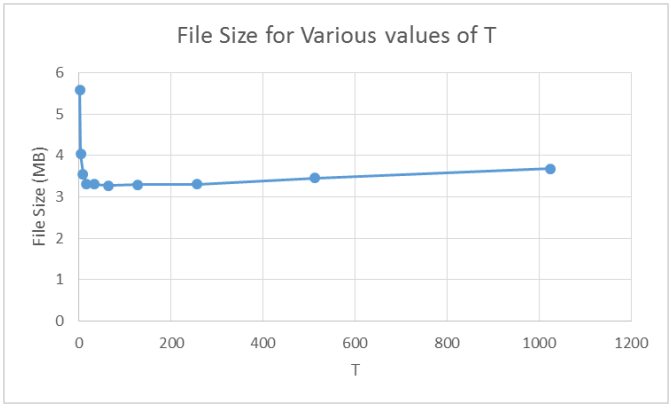
Stat	T=2	T=4	T=8	T=16	T=32	T=64	T=128	T=256	T=512	T=1024
Height	11	6	4	3	2	2	1	1	1	1
Load Factor (%)	58.1	65.7	67.9	69.4	68.39	68.32	67.74	67.61	65.17	61.89
File Size (MB)	5.58	4.03	3.55	3.32	3.30	3.27	3.29	3.30	3.45	3.68
Execution Time	90521	78710	67160	76183	91331	154296	244007	420270	794286	1512871

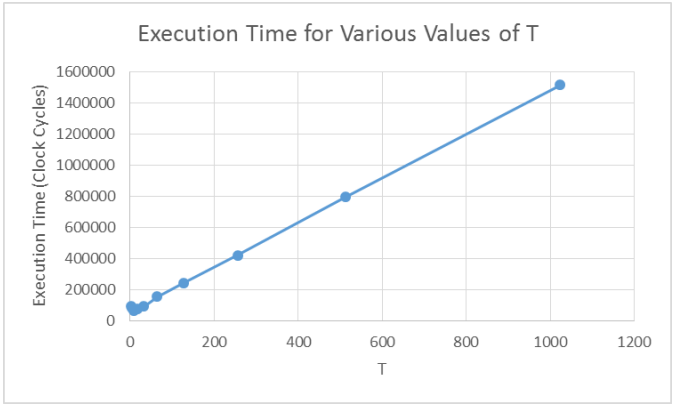
Execution time is counted in clock cycles. Overhead was not calculated.

Looking at each statistic graphically:









The height for each tree drops quickly as t increased from 2 to 8 (the height went from 11 to 6 and to 4, respectively), and then stays at 1 for 128 – 1024. The only way the height could be reduced to 0 is if t = 19,003, setting the max number of keys for the B-Tree at 38,005, allowing all keys to be in the root node and leaving one blank space. The load factor starts low when t = 2, but jumps to ~66% and goes as high as ~68% when t=8. After that point, the load factor starts going down steadily. Looking at file size, the file size is the largest when t = 2, then drops down to a little above 3 MB and then starts to slowly creep up in space as t increases. Lastly, the execution time drops slightly around t = 4 and t = 8, and then starts to rise in a linear fashion as t increases.

Looking at all of these graphs, it appears that time-wise a value of t between 2 and 20 would be the most ideal number, as the height will be low, file size is near its lowest, and execution time is the fastest. However, the load factor would be around 60%, which may be considered high, depending on the "ideal" load factor.