Pat Wongwiset

10/10/17

Post-lab 5 - Analysis

section 2 - 3.15 pm

testfile4.txt

a bird called doves eats food goat hen ink jelly kite lion moon nine orange pineapple queen rat sequence toe unix van women xyrine yoyo zebra

Why the testfile4.txt file works

testfile4.txt can illustrate the superior implementation of AVL tree to binary search tree in case that all the data are in sorted order. Therefore, the worst case when the user needs to find the last sorted word, ‘zebra’ in this case, would take a linear time to do that. On the other hand, the AVL tree could balance the tree to prevent the worst case (linear time) so the file can shows that the worst case will take log(n) time or the number of total nodes in which AVL tree passing through is less than that of BST.

Actual numerical results

Please enter the name of a file of words: **testfile1.txt**
we can't solve problems by using the same kind of thinking we used when we created them - albert einstein
19 words in this text

BST:
Left links followed = 0
Right links followed = 0
Total number of nodes = 17
Avg. node depth = 3

AVL Tree:
Left links followed = 0
Right links followed = 0
Total number of nodes = 17
Single Rotations = 6
Double Rotations = 2
Avg. node depth = 2

Enter word to lookup > s
s was not found in testfile1.txt
BST:
Left links followed = 3
Right links followed = 2
Total number of nodes = 17
Avg. node depth = 3

AVL Tree:
Left links followed = 2
Right links followed = 2
Total number of nodes = 17
Single Rotations = 6
Double Rotations = 2
Avg. node depth = 2

Please enter the name of a file of words: **testfile2.txt**
a bee caught dung everywhere flying greatly higher in mauve skies than we had ever flown
16 words in this text

BST:
Left links followed = 0
Right links followed = 0
Total number of nodes = 16
Avg. node depth = 6

AVL Tree:
Left links followed = 0
Right links followed = 0
Total number of nodes = 16
Single Rotations = 9

Double Rotations = 0
Avg. node depth = 2

Enter word to lookup > we
Word was found: we
BST:
Left links followed = 0
Right links followed = 12
Total number of nodes = 16
Avg. node depth = 6

AVL Tree:
Left links followed = 0
Right links followed = 3
Total number of nodes = 16
Single Rotations = 9
Double Rotations = 0
Avg. node depth = 2

Please enter the name of a file of words: **testfile3.txt**
zany cobwebs littered the clockwork orange landscape like misty works of surreal art
13 words in this text

BST:
Left links followed = 0
Right links followed = 0
Total number of nodes = 13
Avg. node depth = 3

AVL Tree:
Left links followed = 0
Right links followed = 0
Total number of nodes = 13
Single Rotations = 5
Double Rotations = 2
Avg. node depth = 2

Enter word to lookup > misty
Word was found: misty
BST:
Left links followed = 3
Right links followed = 2
Total number of nodes = 13
Avg. node depth = 3

AVL Tree:
Left links followed = 2
Right links followed = 1
Total number of nodes = 13
Single Rotations = 5
Double Rotations = 2
Avg. node depth = 2

Please enter the name of a file of words: **testfile4.txt**
a bird called doves eats food goat hen ink jelly kite lion moon nine orange pineapple queen rat sequence toe unix van women xyrine yoyo zebra
26 words in this text

BST:
Left links followed = 0
Right links followed = 0
Total number of nodes = 26
Avg. node depth = 12

AVL Tree:
Left links followed = 0
Right links followed = 0
Total number of nodes = 26
Single Rotations = 21
Double Rotations = 0
Avg. node depth = 3

Enter word to lookup > zebra
Word was found: zebra
BST:
Left links followed = 0
Right links followed = 25
Total number of nodes = 26
Avg. node depth = 12

AVL Tree:
Left links followed = 0
Right links followed = 4
Total number of nodes = 26
Single Rotations = 21
Double Rotations = 0
Avg. node depth = 3

A characterization of situations where AVL trees are preferable to BSTs

From running all four .txt files, AVL trees can run faster and more efficient that BSTs especially in case that the data is in sorted order. Since BSTs would act as a list on the case when the data is already sorted, the time complexity becomes linear. While AVL tree implementation would be self-balancing trees, the complexity time is confirmed to be log(n) which is faster. It also confirms that the AVL trees are more compact, so the distance from root to the defined node would be less than that of BSTs. The path of find() method in AVL tree passed less number of nodes compared to BST in average case observed from experiments. Thus, AVL trees are also more preferable when a program needs to mostly implement the find() method (searching) with a few amount of insertion and deletion.

A discussion of the costs incurred in an AVL implementation

The implementation of AVL trees is more complexed and difficult to code than BST, so the coder has to spend time on coding and debugging it more than BSTs. Some operations i.e. printing elements both AVL trees and BSTs would implement in the same time complexity, linear, so BST might be the better choice if users require only storage and printing. Moreover, AVL trees implementation is not a good choice in case that each depth representing hierarchy (level of importance) because the rotations will disturb it. Even though AVL trees are worth a lot in term of time saving when the program is mostly used in searching data, the insertion might take much longer time if the size of data is big since single and double rotations make the program run slower. In each single rotation, there are two nodes which have to change their pointers to balance factor numbers to be -1, 0 or 1 so that the tree will be balance and compact i.e. decreasing the distance from root to leaves, so the program needs to spend more time on this complexity. In worst case, the program can be slower than BST if there exist many rotations when it is inserting or removing many elements.