HW3

Topics: Tree

Course: DSA

Xinyu Lyu(xl422)

Q1 Develop an implementation of the basic symbol-table API that uses 2-3 trees that are not necessarily balanced as the underlying data structure. Allow 3-nodes to lean either way. Hook the new node onto the bottom with a black link when inserting into a 3-node at the bottom.

Description:

For this problem, we need to make some changes of the usual RedBlackTree API, like the put and deletion. For put function, we do not need to consider the rotate and flip, because there is no such occasion considering a unnecessary balanced 2-3 tree. It is the same to the deletion functions. Because, basically, an unbalanced 2-3 tree is a BST apart from mark some nodes as reds or blacks. We don't need to make changes of the other basic RedBlackTree API, for they don't need the filp or rotate operations.

For test of the built tree, I write functions to test the basic API such as isempty(), put(), Value get(key), contains(key), int size(). I also test if it is a 2-3 tree or if it is balanced.

Test results:

```
Test of the built tree:
is 2-3 tree
not balanced
Test of size() function:1000
Test of get(3) function:3
Test of contains(2) function:
contains 2
Test of delete(2) function:
after delete(2) not contains 2
```

Q2. Run experiments to develop a hypothesis estimating the average path length in a tree built from (i) N-random insertions. (ii) N-sorted insertions?
Descriptions:

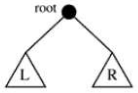
Apart from some basic API from Q1, I implement double getresult() function to return the average length to a random node in the built 2-3 tree. In the double getresult(Node x) function, I invoke the int getInterlength(Node x) to add the size of every node into the list. There are two ways to implement this function. Originally, first I traverse the tree inorder, then set each node as the target node. Use the recursion functions to get the path length from the root to the target node then add it to the list and get the average value. But the algorithm complexity is too high, for there are too much recursions. Therefore, I use an easier way to implement. In this way, it basically follows the definition of the internal path length. For each height of nodes, multiple the height value and the number of nodes in that height level, then sum up all the results of them with different height levels. Then divide that value by the N (total amounts of nodes). To implement, for each random node, use the recursion function to add its size and its left node size and its right node size together, then add the value into the list. At last, when sum up all the values in the list

and divide by the length of the list, you can get the average internal path length to a random node. (https://web.cs.wpi.edu/~cs504/s00m/classes/class12/Class12.html)

Total Path Length

One way to calculate the expcted value of the path length in a tree is to calculate the total path length **P** to reach all nodes in the tree and divide by **N**, the number of nodes.

We can model a tree as a node plus two subtrees. The nice thing about this definition is that it is recursive. If we get the calculation right at the root, it will apply to all nodes in the tree.



Look at a randomly grown tree. When you see "random" you natrurally think of working with random variables and expected values. Assign a random variable P_N which represents the total path length to all nodes in a tree of size N. To account for randomly grown trees, we just find the expected value of P_N for all possible binary trees with N nodes. And the average path length to any node in a randomly-grown binary tree of size N is just that expected value divided by N.

$$\frac{E(P_N)}{N}$$

So let's see how to find the expected value $E(P_N)$.

Each subtree will have it's own total path length, measured from it's own "root" node. The total path length for the entire tree can be calculated from the sum of the path lengths of the subtrees.

$$P_N = N + P_L + P_R$$

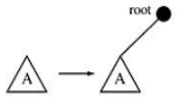
The N comes from the fact that each node in the two subtrees has a path length one more than when P_L and P_R were calculated because they are now subtrees off the root. And, there are N-1 nodes in the subtrees. So, the first term should be N-1 but we made it N because the math becomes easier. This is an approximation which works for large N. Now look at the case where there are K nodes in the left subtree and N-K-1 in the right subtree. We want to calculte the expected value of P_N over all possible values of K from 0 to N-1. So, we can transform the above equation into an equation involving expected values.

Internal and External Path Lengths

A node's *path length* is the number of links (or branches) required to get back to the root. The root has path length zero and the maximum path length in a tree is called the tree's *height*. The sum of the path lengths of a tree's internal nodes is called the *internal path length* and the sum of the path lengths of a tree's external nodes is called the *external path length*. For the tree shown above, the internal and external path lengths are:

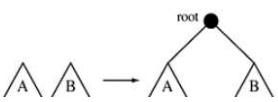
$$L_I = 0+1+1+2 = 4$$
$$L_E = 2+2+2+3+3 = 12$$

Consider a full binary tree *A* with N_A nodes, internal path length $L_{I,A}$ and external path length $L_{E,A}$. If we make this tree the left subtree of a root node, then each path length, internal and external, in the left subtree increases by one:



$$N_A \rightarrow N_A + 1$$
$$I_A \rightarrow I_A + 1$$
$$E_A \rightarrow E_A$$
$$L_{I,A} \rightarrow L_{I,A} + I$$
$$L_{E,A} \rightarrow L_{E,A} + E = L_{E,A} + I + 1$$

Note the substitution of the identity between the number of internal and external nodes in the last equation. Now add another tree, B, as the right subtree of the root node:

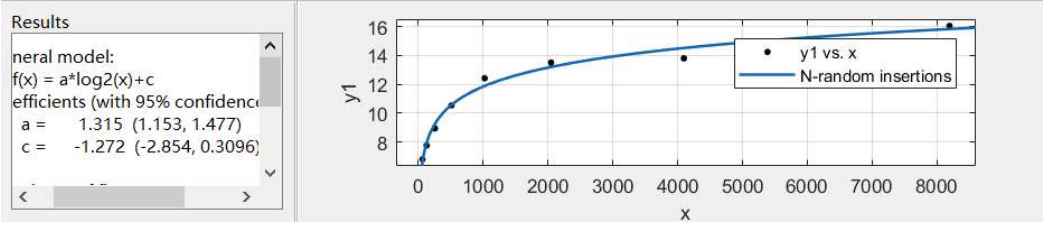


Test Results:

```
64 sorted insertions: average length path:32.5
64 random insertions: average length path:6.203125
128 sorted insertions: average length path:64.5
128 random insertions: average length path:7.3828125
256 sorted insertions: average length path:128.5
256 random insertions: average length path:9.14453125
512 sorted insertions: average length path:256.5
512 random insertions: average length path:10.408203125
1024 sorted insertions: average length path:512.5
1024 random insertions: average length path:11.943359375
2048 sorted insertions: average length path:1024.5
2048 random insertions: average length path:16.0771484375
4096 sorted insertions: average length path:2048.5
4096 random insertions: average length path:15.05908203125
8192 sorted insertions: average length path:4096.5
8192 random insertions: average length path:16.2989501953125
```

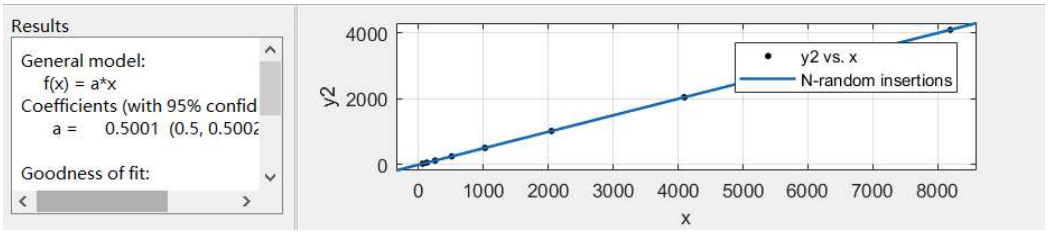
And I use matlab to do the curvefitting for the test result:

(1) N-random insertions (x is N, y1 is average path length)



The formula is y1=1.315log(N)-1.272

(1) N-random insertions (x is N, y2 is average path length)



The formula is y2=0.5N

Q3. Write a program that computes the percentage of red nodes in a given red-black tree. Test program by running at least 100 trials of the experiment of increasing N random keys into an initially empty tree for N=10^4, 10^5 and 10^6 and formulate a hypothesis.

Descriptions:

To calculate the percentage of the rednodes in RedBlackTree, first I traverse the tree nodes inorder, then add 1 to the list each time visit a red node. When sum up all the values in the list, we get the total number of rednodes, then divide by M (the total

numbers of nodes in the tree), we will get the percentage value of rednodes. And the results reveal that it tends to be 25%.

Test Results:

```
M random insertions:10000.0 red node percentages25.45159999999999%
M random insertions:100000.0 red node percentages25.38397%
M random insertions:1000000.0 red node percentages25.390409%
```

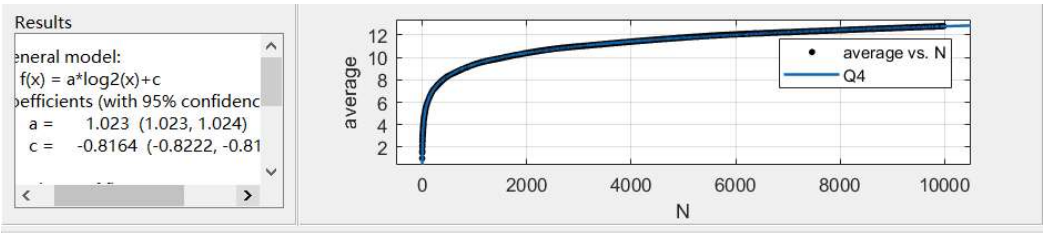
Q4. Run empirical studies to compute the average and std deviation of the average lenght of a path to a random node (internal path length divided by tree size) in a red-black BST built by insertion of N random keys into an initially empty tree, for N from 1 to 10,000. Do at least 1,000 trials for each size.

Descriptions:

The function of get internal oath length is the same as the Q2, just look at the Q2 to see the statement of the implementations of such function. After running 1000 trails of each N, I calculate the average value of 1000 trails of each N. Then calculate the deviation of such 1000 trails with size N. Then I write the N, average, deviation values to both the csv file in order. And do the curvefitting with matlab to develop the formular. Also I save the test results to txt files for check.

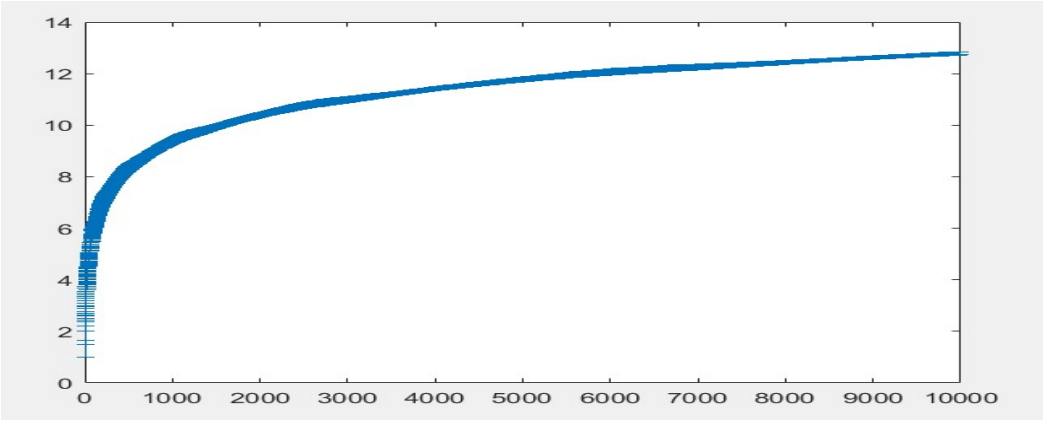
Test Results:

(1) Test results for the average of the average path length



The formula is average=1.023log(N)-0.8164

(2) Test results for the average of the average path length adding the std deviation of the average path length



Q5. Implement the rank() and select() ordered operations for a BST. Use data set linked below. (i) What is the value of select(7) for the data set? (ii) What is the value of rank(7) for the data set? [10 points]

Descriptions:

Basically follow the rules of the rank() and select() as follows to implement.

Selection. Suppose that we seek the key of rank k (the key such that precisely k other keys in the BST are smaller). If the number of keys t in the left subtree is larger than k, we look (recursively) for the key of rank k in the left subtree; if t is equal to k, we return the key at the root; and if t is smaller than k, we look (recursively) for the key of rank k - t - 1 in the right subtree.

Rank. If the given key is equal to the key at the root, we return the number of keys t in the left subtree; if the given key is less than the key at the root, we return the rank of the key in the left subtree; and if the given key is larger than the key at the root, we return t plus one (to count the key at the root) plus the rank of the key in the right subtree.

Therefore, select(7) is to return the key of the nodes which has 7 nodes smaller than it. 8 has 1-7 seven nodes smaller than it, so select(7) should be 8. Rank(7) is to return the amount of nodes smaller then 7, which is from 1-6 comes as 6.

Test Results:

```
rank(7):6
select(7):8
```