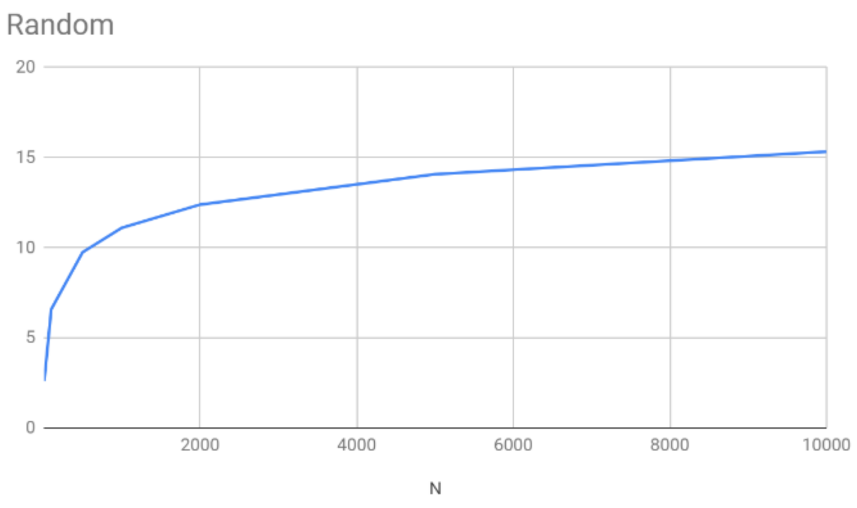
DSA HW3
Yuhang Zhou
yz853
4/2/2019

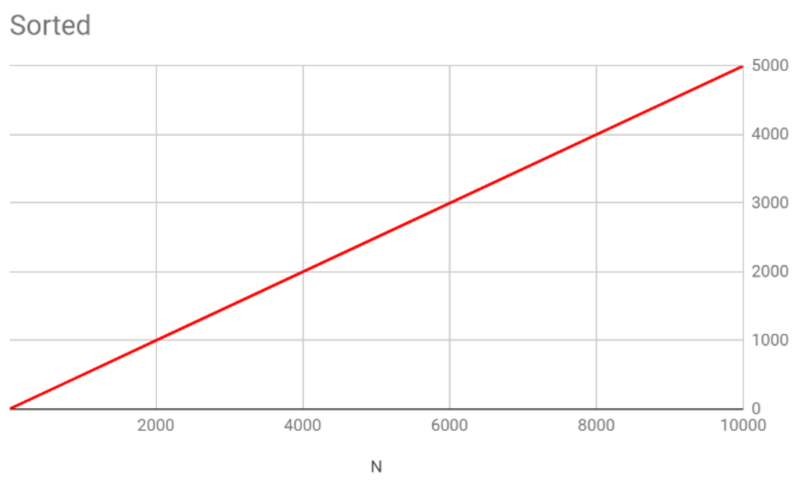
Q1.
The 2-3 tree can lean either way and no necessarily balanced, so that it's basically a red black tree and the position of the red node doesn't matter. But at least a red node won't connect to another node. Details in the source file.

Q2.
In the N-random insertions case. After insertion, the best case is that the becomes a balanced tree, so that the longest path length with be log(n). The worst case is that all the nodes relies on one side, so that the path length will be N.
In the N-sorted insertions case. Because the insertions are sorted, the inserted data will be larger than any data in the existing tree. So the inserted nodes will rely on one side, the longest path length will be N.

In the experiment, I made a very basic BST and inserts these data into the tree. Then I get the length to get every data and add them together. At last, the total length will be divided by the size of the tree.
Results

N	10	100	500	1000	2000	5000	10000
Random	2.6	6.59	9.746	11.104	12.3905	14.0784	15.3301
Sorted	4.5	49.5	249.5	499.5	999.5	2499.5	4999.5





As I said before, the results are very close to my hypothesis.
In the random one, length is almost proportional to log(n), but it's a little larger than log(n). While in the sorted one, obviously the length is N/2. Because the longest length is N, and the shortest is 0, also the lengths are an arithmetic sequence, so that the average length is N/2, which make sense.

Q3.
The source code of red black tree is from Wikipedia Chinese version
https://zh.wikipedia.org/wiki/%E7%BA%A2%E9%BB%91%E6%A0%91#C++%E7%A4%BA%E4%BE%8B%E4%BB%A3%E7%A0%81
I made some changes inside the codes to count the number of red nodes
Add an int redn in the public members of the trees class, and set the redn=0 in the class constructor.
Add two functions void coutrend() and void traversecount() to traverse all the nodes and count the number of red nodes
Result:

N	Percentage
10000	0.486649
100000	0.486635
1000000	0.486512

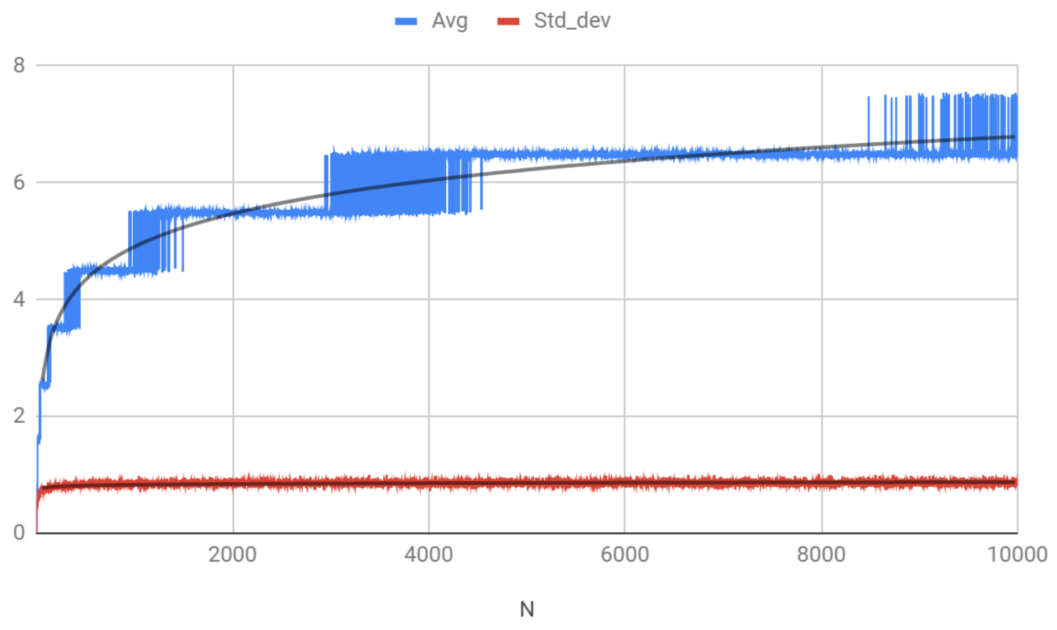
Hypothesis:
The percentage of the red nodes in a red-black tree is around 48.5% when the insertions are random and not duplicated.
I guess the reason is that the red-black tree I used is a little different from the one we learned in the class. In the class, the red black tree we learned is left-leaning, while the tree I use doesn't have this restriction. So in this case, the red node will be similar with the black node. It makes sense that the percentage of red node is close to 50%. So if the red black tree I used is left-leaning, the percentage maybe around 25%.

Q4.

The source code of red black tree is the same as the one I used in Q3. Based on that, I add another function int length(int), which can find the node and return the length to this node. Details in the head file.

The results are saved in the Q4results.csv file.

I draw a chart using the results data. The black line is the trend line.



So the average length looks like is proportional to log(n), and the std deviation is always very small.

Q5.

rank(7)=6

select(7)=8

Most of the data in the dataset is duplicated, but it covers from1-999. Btw, my BST can not accept duplicated nodes, or the result will be much lager.

The rank() means find the node, and count the number of nodes whose key is smaller than the founded one. So there are 1-6, 6 nodes are smaller than 7.

The select() means find the node, the number of nodes whose key is smaller should be the inserted number. So the smallest 7 nodes are from 1-7. So the next node should be 8.