**Data Structure & Algorithms Homework 3**

**Ruiyu Zhang || rz213**

**#All scripts tested under Python 3.5 environment#**

**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.**

**Solution:**

Instead of using a 2-3 tree, I choose to apply a Red-Black tree for implementation. Reason being: a typical 2-3 tree has no such property as “lean either way”(because they already are) and “black link”(this makes sense only in RB tree). In my implementation, RB-tree is implemented as a lite version:

class RBT\_Lite(object):

It does not rotate when inserting, does not check if red links are properly placed as in a real RB tree. It is so a unbalanced tree, but still works.

To implement API for symbol table, I wrote another class:

class ST(object):

which generates a table for itself while initializing.

Several functions have been implemented for symbol table, and finally a summary function is used to easily check out most functions.

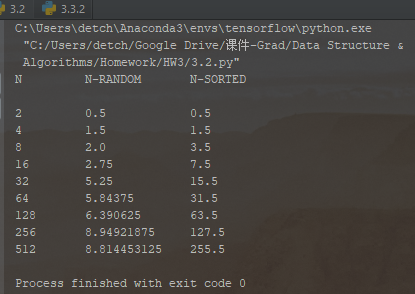


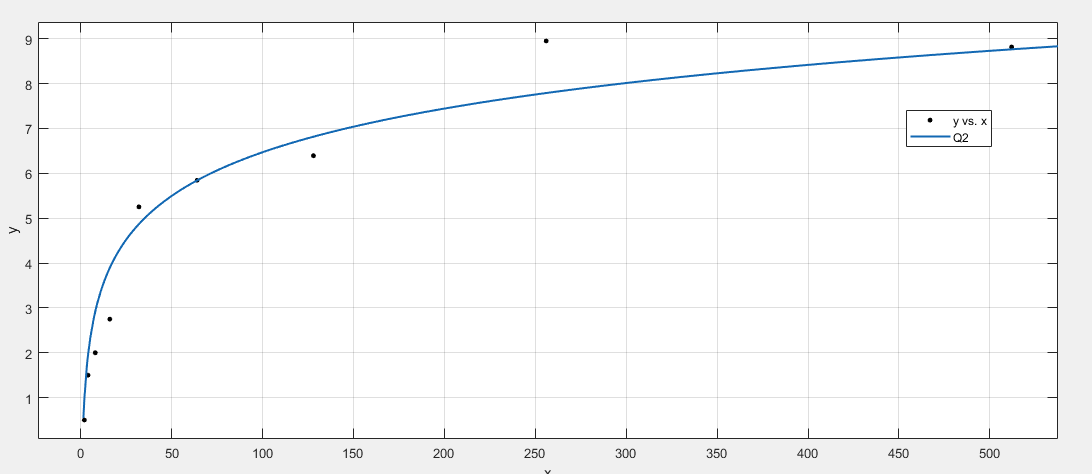
**Q2. Run experiments to develop a hypothesis estimating the average path length in a tree built from (i) N-random insertions. (ii) N-sorted insertions?**

**Solution:**

Since no tree type has been appointed here, I turn to choose BST for implementation.

Apparently, N-sorted case is the worst case for a BST, because it will generate a whole list of items which requires iteration from root (very beginning) to find a specific item. And because of that, the average path length equals to **half of tree size**, which can be told from my result shown below.

One more thing to be mentioned: the implementation entails recursion for tree traversal, and because the tree generated in N-sorted case is so bad, the tree size can not be more than 999, or python interpreter will raise error. This is the reason why my test data size is so small.  




**Curve fitting is based on Matlab-R2017b Curve Fitting Tool.**

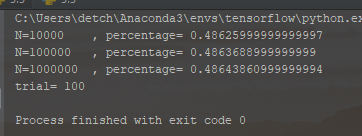
**y(N-random) ≈ 1.405log(x)**

**y(N-sorted) = 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.**

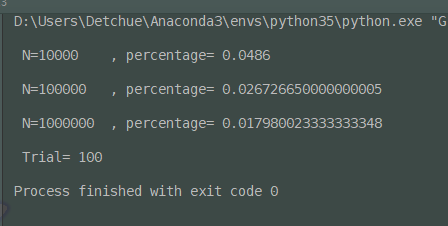
**Solution:**

In a typical RBT, the ratio of red and black nodes is be generally similar according to experience. (Not counting nil sons of course.) So the most possible percentage in most common cases should be around 0.5. This is well demonstrated in my first screenshot, where random list is used as data.



**Running with randomly-generated data set**

Then again since we are requested to use given data from **Sakai** for calculation, I used it with slight modification in formats. Because of the **highly frequent duplication** in given file, the result looks rather **uncommon**. I choose to ignore this piece of result, believing that it shows no valid trends.



**Running with data set provided (with more trials and higher accuracy)**

**#not taken into consideration when forming hypothesis**

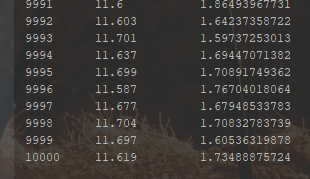
**hypothesis:**

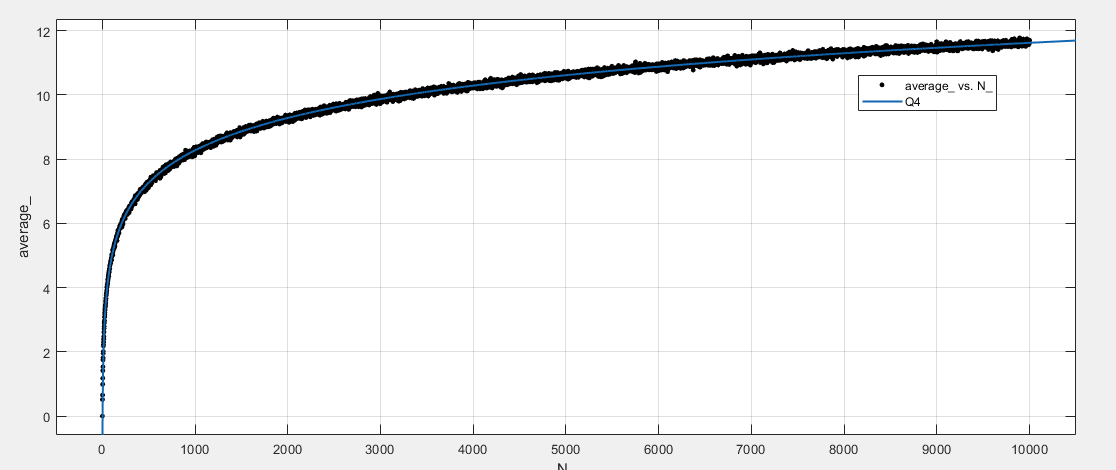
In a Red-Black Tree, the quantity of red and black nodes are most likely similar to each other in numbers. But black nodes are slightly more than red nodes, around 3% more.

**Q4. Run empirical studies to compute the average and std deviation of the average length 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.**

**Solution:**

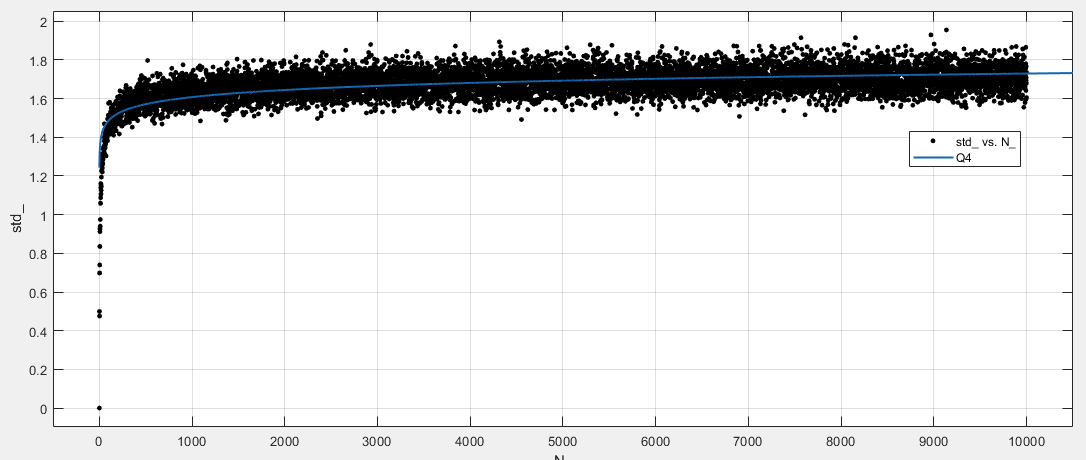
Results are printed in txt file (3.4\_result\_1-10000.txt) attached. A screenshot of last several results are shown below.



**‘N-size’, ‘Average path length’, ‘STD Deviation’**  


**Average vs N**

**Average ≈ 1.453\*ln(x) - 1.768**



**std-deviation vs N**

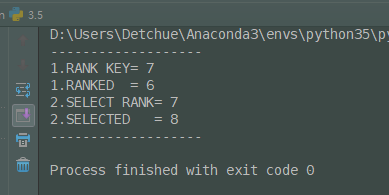
std goes quite stable after N goes beyond 1000.

**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?**

**Data Set for Q3 is: https://sakai.rutgers.edu/access/content/group/f73f2fd4-280d-4e7c-8cf2-9cc34bcffcff/HW-DataSet/select-data.txt**

**Solution：**

Since the given data file does not show high variety (but with high duplication), the final tree is pretty much like a range() list. This explains why whatever rank-key or select-key is given, the corresponding value is close to the given key. (e.g. here rank(7)=6, select(7)=8)



**rank(7)=6 and select(7)=8**