DSA ECE 573
Name: Shounak Rangwala
Netid: snr85
Homework Assignment 3

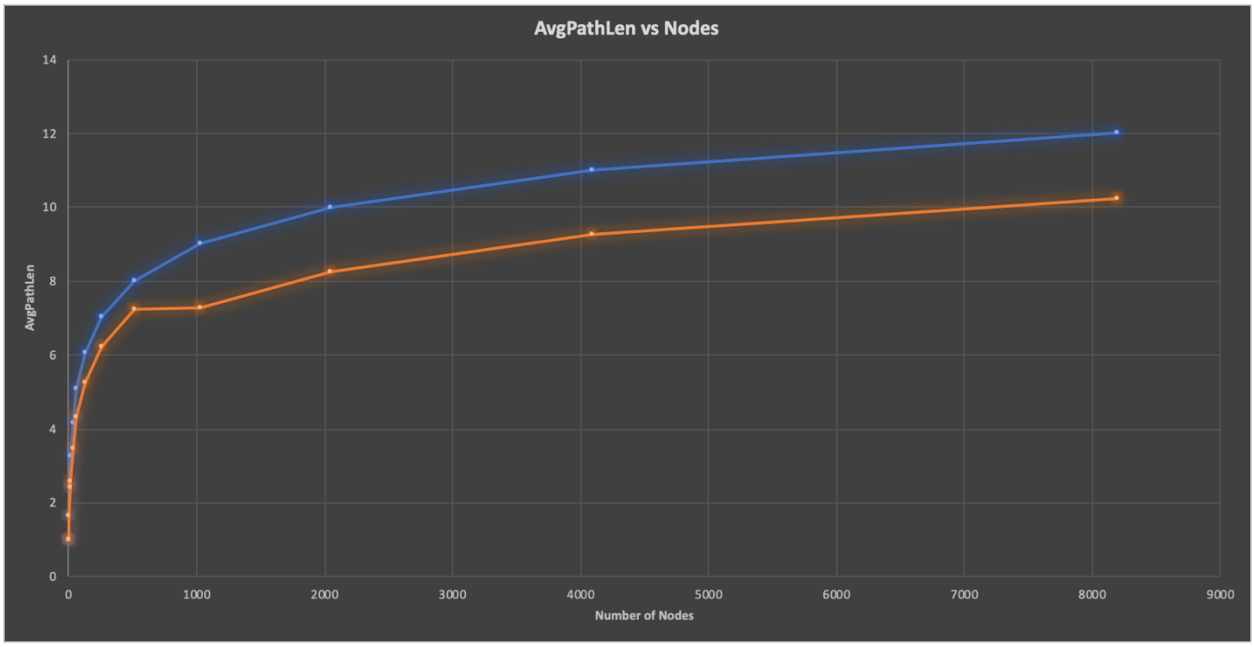
Q1)
I implemented the Left-leaning version of the Red Black trees. I developed the get() method which stays the same for normal BST and 2-3 BST's. The put() method was made by incorporating the rotateLeft() and the rotateRight() and the flipColor() methods that would help to balance the height of the tree when you add new nodes. The isRed() help to identify if the nodes have a red parent link or a black parent link.
I am using the dataset given to create the BST and displaying the inorder traversal of the tree using the inorder() function.

Q2)
Building on the red black tree made in the previous question, I added a method to calculate the avgLength(). I did not know what can be described as the average path length so I made an assumption here. The assumption I made was that the average path length would be the average of the path length from the root node to each leaf node in the red-black BST. This makes sense when you account for the balanced nature of the tree so each new node being added would not increase the height of the tree randomly.
The worst case scenario would be $\log_3 N$ where N is number of nodes in the tree.
However since we are using the 2-3 BST, we count only the black links as the actual path length. To do this, we calculate the number of red nodes in each path and deduct that from the total nodes in that path because we count them twice instead of just once. This logic is carried out in the searchBST() method in the code.
The results obtained were as follows:

| Number of Nodes (N) | Ordered insertions | Random insertions |
|---|---|---|
| 1 | 1.0 | 1.0 |
| 2 | 1.0 | 1.0 |
| 4 | 1.6666666666666667 | 1.6666666666666667 |
| 8 | 2.4285714285714284 | 2.4285714285714284 |
| 16 | 3.2666666666666666 | 2.5833333333333335 |
| 32 | 4.161290322580645 | 3.4583333333333335 |
| 64 | 5.095238095238095 | 4.326086956521739 |
| 128 | 6.05511811023622 | 5.244897959183674 |
| 256 | 7.031372549019608 | 6.248730964467005 |
| 512 | 8.017612524461839 | 7.221354166666667 |
| 1024 | 9.009775171065494 | 7.268134715025907 |
| 2048 | 10.005373717635564 | 8.2634691195795 |
| 4096 | 11.002930402930403 | 9.261125654450261 |
| 8192 | 12.001587107801246 | 10.25211038961039 |

By eyeballing the data we can conclude that the relationship between the av path length and the number of Nodes, N is of the form:
AvgPathLen = a(logN) + b

Here the value of a and b differ for random and ordered insertions.
As predicted by the algorithm the av height of the tree is logN and the av path length is to be in the same order because of the balanced nature of the red-black tree.
The result graph looks something like this:




Q3)
I worked upon the already created tree from Q1 and Q2 to add new methods of percentRed().

This was pretty simple to calculate because I was using the number of red nodes in Q2 to deduct from the paths to get the path length. For this question all we did. Was calculate total number of nodes, total number of red nodes, deduct the latter from the former and take percentage from this new result.

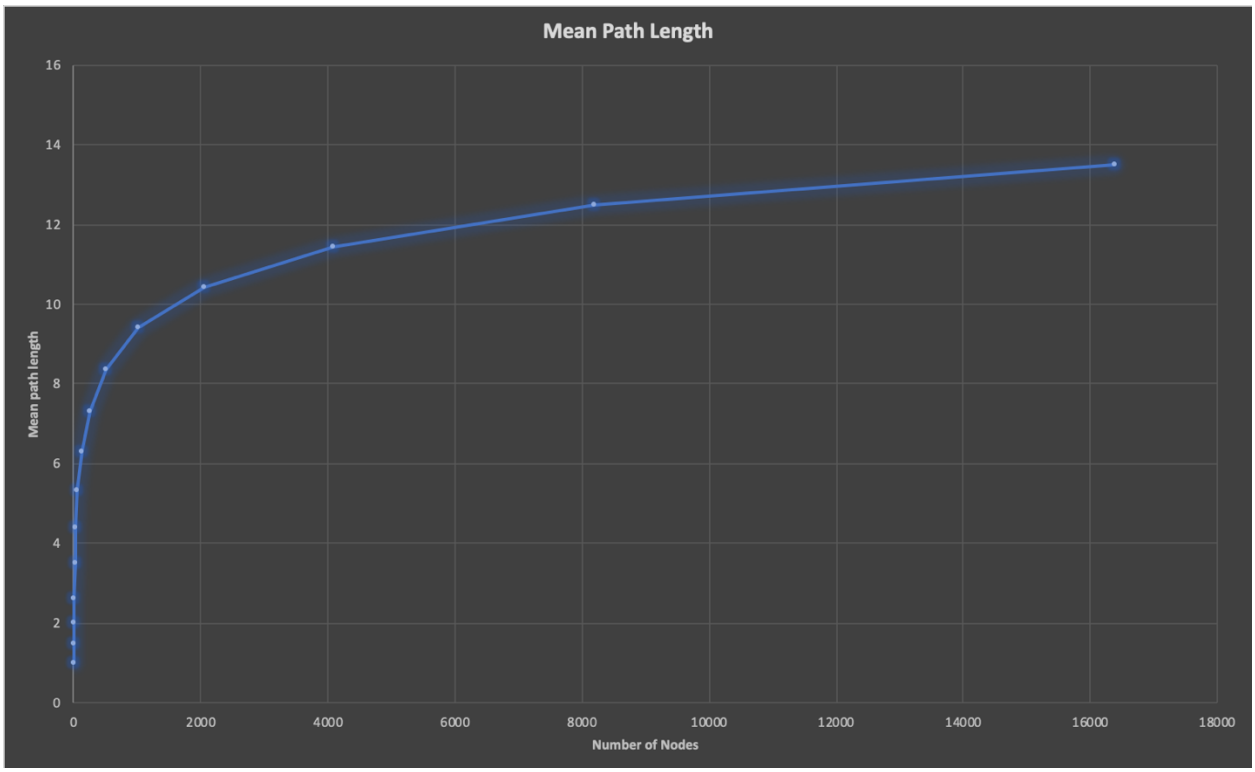
The calculation for $10^6$ took almost 30 minutes.
The result obtained from using different N is:

| | $10^4$ | $10^5$ | $10^6$ |
|---|---|---|---|
| % | 25.054882 | 25.77432 | 25.34557 |

Q4)
Building on the same BST used in the previous questions, we already have the avg path length from the previous question.
In the main() function for this program I run a loop from N=1 to $2^{14}$ . I increase N exponentially by powers of 2 because otherwise the computation time of this code could take beyond 10 hours on my laptop.
For each N value, I run a nested for loop 1000 times. Each time I calculate the avg path length, I store it in an array. After the 1000 values are recorded in the array, I would calculate the average of the 1000 values and the standard deviation in that population. The output of this average and standard deviation is being pushed into a csv file to maintain a record.
I couldn't run 10000000 computations because the processing power on my laptop is very limited.

```
nodes = 14
    arr = 2**np.arange(0,15,1)
    with open('Q4results.csv', 'w', newline='') as file:
        writer = csv.writer(file)
        writer.writerow(["N","Mean Path Length","Standard Deviation"])
        for i in arr:
            print("Starting on node "+str(i))
            pathL = 0
            pathDev = []
            for j in range(1000):
                b_random = randomInsert(i)
                pathL += b_random.getIntPath()
                pathDev.append(b_random.getIntPath())
            meanL = pathL/ 1000.0
            stdDev = 0
            s = 0
            for j in range(1000):
                s+=(pathDev[j] - meanL)**2
            stdDev = math.sqrt(s/1000)
            writer.writerow([i, meanL, stdDev])
```

This is the graph of the final results that are recorded in the csv file.




Q5)
For this question, I implemented a normal BST with the get(), put(), rank() and select() ordered implementations. I used the dataset given to make the BST and the output for the select(7) and rank(7) is as follows:

```
In [9]:  1 b.select(7)
Out[9]: 8

In [10]:  1 b.rank(7)
Out[10]: 6
```