Yilin Yang

Data Structures HW 3 Report

Q1.

An API implementation of a 2-3 Tree was developed in Python3. Methods are provided to print the tree, sort tree nodes, insert new elements, delete elements, split 4-nodes which may temporarily be created during insertion, search for elements, and path length calculation. See submitted file "q1.py" for specifications.

Q2.

Table of Average Path Lengths in a 2-3 Tree of N-elements that have been Randomized or Sorted

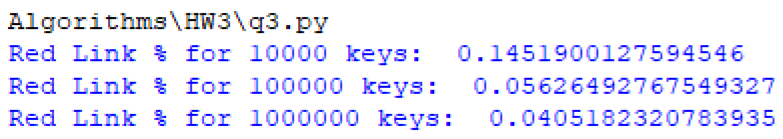
Problem Size (N)	Random	Sorted
1k	7	8
2k	8	9
4k	9	10
8k	10	11
16k	11	12
32k	11 (not a typo)	13
64k	12	14
128k	13	15

In the experimental procedure, a list was prepared that holds integers ranging from 1 to 128 thousand in ascending (sorted) order. The user may input a problem size (N) and specify whether they want the input data sorted or randomized. If the data is requested to be randomized, the list will then be shuffled. This data is then input into a 2-3 tree and the average path length is then calculated.

Based on the results collected, it is hypothesized that path length is roughly proportional to log(N)-2 for sorted inputs and log(N)*0.75 for randomized inputs.

Q3.

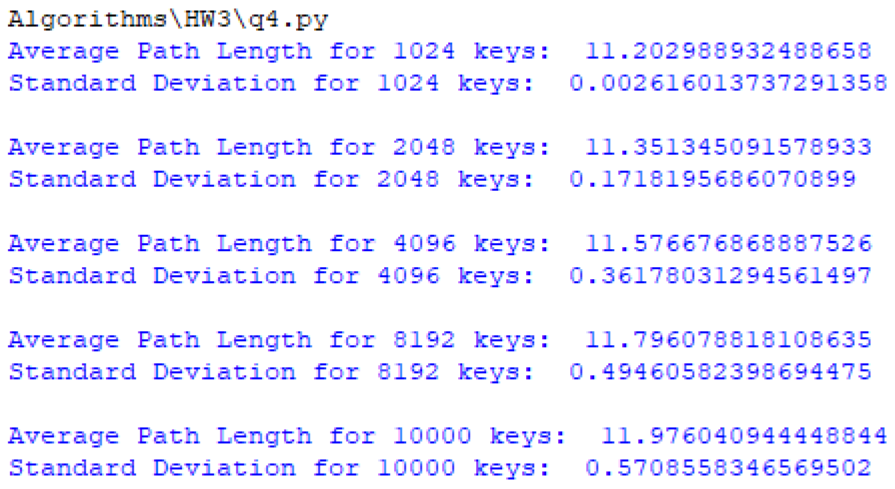
An API implementation of a Red Black Tree was developed in Python3, similar to the API developed for the 2-3 Tree. This implementation calculates the percentage of links in the Red Black Tree that are colored red. Using the data set provided, three different experiments were run using problem sizes (N) of 10 thousand, 100 thousand, and 1 million. For each experiment, the data set from the external text file is read into a list and the list is shuffled. A Red Black Tree is then assembled using this input and the number of red links are counted and divided by the total number of links. This procedure is repeated 40 times per experiment and the results averaged together for a total of 120 trials. A screenshot of the results is shown below:



```
Algorithms\HW3\q3.py
Red Link % for 10000 keys:  0.1451900127594546
Red Link % for 100000 keys:  0.05626492767549327
Red Link % for 1000000 keys:  0.0405182320783935
```

Q4.

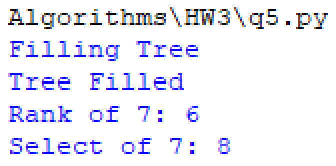
The API of the Red Black Tree developed for Q3 was reused for this problem. Rather than measure the percentage of red links in the tree, this time the number average path length to all leaf nodes was calculated. Using the provided dataset once again, a Red Black Tree was constructed and then the tree was traversed to count the path lengths to each leaf. The sum of all internal path lengths is then divided by the tree size. This process is repeated for 1000 trials. The outcome of each individual trial is compiled in a list. The contents of the list are used to calculated final average and standard deviation. Five different problem sizes were used: 1024, 2048, 4096, 8192, and 10000 (the maximum possible size). Each problem size underwent 1000 trials. The outcome of the trials is shown beow.



```
Algorithms\HW3\q4.py
Average Path Length for 1024 keys:  11.202988932488658
Standard Deviation for 1024 keys:  0.002616013737291358

Average Path Length for 2048 keys:  11.351345091578933
Standard Deviation for 2048 keys:  0.1718195686070899

Average Path Length for 4096 keys:  11.576676868887526
Standard Deviation for 4096 keys:  0.36178031294561497

Average Path Length for 8192 keys:  11.796078818108635
Standard Deviation for 8192 keys:  0.49460582398694475

Average Path Length for 10000 keys:  11.976040944448844
Standard Deviation for 10000 keys:  0.5708558346569502
```

Q5.

A program that implements the rank() and select() operations for a BST was developed. This particular implementation ignores duplicate keys. Using the dataset provided, rank(7) and select(7) were evaluated.



```
Algorithms\HW3\q5.py
Filling Tree
Tree Filled
Rank of 7: 6
Select of 7: 8
```