Assignment: Balanced Binary Search Tree

LI Xiao Yang

SID: 56638660

Email: xyli45_c@my.cityu.edu.hk

1 Background

AVL tree and splay tree will be chosen in the assignment. All findings and discussions in this report will base on these two data structures.

In the following discussion, operation cost and performance will be measured in the number of rotations and the number of nodes visited in one single operation.

The compiling commands for generator programs are provided as follows.

```
g++ generator1.cpp
./a.out
```

```
g++ generator2.cpp
./a.out
```

```
g++ generator3.cpp
./a.out
```

```
g++ generator4.cpp
./a.out
```

```
g++ generator5.cpp
./a.out
```

2 Generators and Datasets

Operation type:
Type 1: insert
Type 2: delete
Type 3: find rank
Type 4: find nodes by rank

Type 5: find precursor

Type 6: find successor

x is the value of operand in operations above, where $x \in [-10^7, 10^7]$.

1) D1: Fake random data

n is the number of operations in one single test, where $n = 10^3, 10^4, 10^5$.

generator 1:

The occurrence of each and every type of operation will be roughly equal in this data set.

2) D2: Sequential data

n is the number of operations in one single test, where $n = 2 \times 10^3, 2 \times 10^4, 2 \times 10^5$.

Only operation 1 and operation 3 will be involved in this dataset.

Since operations 3 and 4 are essentially similar, only operation 3 are considered in this dataset for simplicity.

generator 2:

Sequentially insert $\frac{n}{2}$ numbers $x_0, x_1, ..., x_{\frac{n}{2}-1}$ in ascending order, where $x_i \in [-10^7, 10^7]$.

Sequentially do operation 3 $\frac{n}{2}$ times for $\frac{n}{2}$ numbers $x_0, x_1, ..., x_{\frac{n}{2}-1}$.

generator 3:

Sequentially insert $\frac{n}{2}$ numbers $x_0, x_1, ..., x_{\frac{n}{2}-1}$ in ascending order, where $x_i \in [-10^7, 10^7]$.

Sequentially do operation 3 $\frac{n}{2}$ times for $\frac{n}{2}$ numbers $x_{\frac{n}{2}-1}, x_{\frac{n}{2}-2}, ..., x_0$, in descending order.

generator 4:

Insert x_i and then do operation 3 for x_i for i = 0, 1, ..., $\frac{n}{2}$-1.

3) D3: Local random data

p is the number of nodes in one single test.

q is the number of operation 3 in one single test.

r is the number of pivot nodes in one single test.

For the same reason, only operation 1 and operation 3 will be involved in this dataset.

generator 5:

Generate p numbers and insert them into BBST. For each and every pivot node, generate roughly $\frac{q}{r}$ operation 3 in its neighboring interval.

3 Comparison and Analysis

The time required with different input test cases is as follows. The time in the table is measured 5 times in seconds. (The time program reads data and outputs data is included.)

1) D1: Fake random data

generator 1:

n	10^3	10^4	10^5
AVL tree	0.555	0.565	0.704
Splay tree	0.559	0.566	0.791

With relatively random data, the performance of AVL tree is obviously better than splay tree. Since the data and operations are random with poor time locality of data access, the cache strategy of splay tree is difficult to achieve. Splay tree has no manual balancing measures, thereby visiting a large number of nodes in the process. In contrast, the balance condition of AVL is relatively strict, and the low tree height ensures the time complexity of one single operation.

2) D2: Sequential data

generator 2:

n	10^3	10^4	10^5
AVL tree	0.576	0.605	0.748
Splay tree	0.559	0.579	0.689

generator 3:

n	10^3	10^4	10^5
AVL tree	0.556	0.565	0.712
Splay tree	0.560	0.567	0.652

generator 4:

n	10^3	10^4	10^5

n	10^3	10^4	10^5
AVL tree	0.566	0.568	0.769
Splay tree	0.543	0.563	0.663

For a large number of sequential data access, splay tree has better performance than AVL tree. This type of dataset makes the cache strategy of splay tree play a rather important role since the target node is close to the root node, and the number of nodes that need to be accessed is limited. In this case, the relatively stable single operation cost of AVL tree becomes a disadvantage.

For test case where $n = 10^7$, the average number of node access for one single operation is as follows:

generator	g2	g3	g4
AVL tree	323.78	323.77	313.77
Splay tree	54.02	54.04	1

The difference is quite obvious, which confirms the analysis above.

3) D3: Local random data

The performance of AVL tree is slightly better than that of splay tree for this type of data.

With the increase of r, the locality of data access decreases, and both performances of AVL tree and splay tree decreases slightly, while AVL tree is less affected.

p/q/r	$10^5/10^4/10^2$	$10^6/10^5/10^3$	$10^7/10^6/10^4$
AVL tree	12.74	12.91	13.35
Splay tree	12.77	13.05	14.64

4 Conclusion

In the case of uncertain data locality or random data, AVL tree is supposed to be choosen, which has relatively stable complexity of one single operation and small time constant. In the case with strong data locality, splay tree should be choosen.

AVL tree:

strength:

1. The performance of AVL tree is better than that of splay tree with random data.
2. The time complexity of one single operation is better because of strict balance conditions and lower tree height.
3. The worst time complexity is $o(\log n)$ for insert, deletion and query operation.
   weakness:
4. The performance of AVL tree with sequential data is not as good as that of splay tree.
5. There is possibly a difference between the measured time complexity and the theoretical time complexity. The cost of rotation after insertion and deletion is expensive. After the delete operation, the scale of rotations is up to $o(\log n)$ times. (see reference)

Splay tree:

strength:

1. The performance of splay tree is better than that of AVL tree in terms of dataset with high locality.

NOTE: By locality, we mean that
1)The node that has just been accessed is very likely to be accessed again very soon.
2)The next node to be accessed is very likely to be close to a node that has been accessed previously.

2. The amortized time complexity of one single operation is $o(\log n)$.

weakness:

The time complexity of one single operation is worse and the performance is worse in terms of random test cases.

5 Reference

Amani, M., Lai, K., & Tarjan, R. (2016). Amortized rotation cost in AVL trees. Information Processing Letters, 116(5), 327-330.

Russo, L. (2019). A study on splay trees. Theoretical Computer Science, 776, 1-18.