Algorithms: Order Statistics Red Black Tree

2017-18570 Sungchan Yi
Dept. of Computer Science and Engineering

May 15th, 2019

Contents

1 Introduction **1**

2 Implementation **2**
2.1 Checker 2

3 Experiments **4**
3.1 Test Case #1 6
3.2 Test Case #2 7

4 Conclusion **9**

1 Introduction

In this homework, we implement an **order statistic red black tree** with C++ language using data structure augmentation. Compared to normal red black trees, order statistics tree will contain an additional field *size* in each node v, which keeps track of the number of nodes in the subtree with root v. Additionally, we implement a checker program for the correctness for the implemented data structure and operations. The implemented tree should support the following operations:

- OS-INSERT(T, x): Insert x into T. If x already exists in T, return 0. Otherwise, return x.
- OS-DELETE(T, x): Delete x from T. If x is not in T, return 0. Otherwise, return x.
- OS-SELECT(T, i): Select the i-th smallest key from T and return it. If i is greater than the number of keys currently in T, return 0.

- OS-RANK(T, x): Return the rank of x if x is in T. Otherwise, return 0.

The following are the restrictions on given inputs.

- Each operation will be denoted as I, D, S, R respectively.
- $1 \leq x \leq 9999$.
- Input will be given in the form "[Character representing the operation] [x]".

2 Implementation

The basic operations required for the tree was implemented as the same way in the textbook, so we omit it here.

2.1 Checker

To check the correctness, we implement a checker program. The basic idea is to use a `vector<int> vec` to keep track of the set of integers given as input.

- `check_insert(int x)` will traverse `vec` and check if `x` is already in the vector. If it is, immediately return `0`. If it is not found, `push_back x` and return `x`.

```
int check_insert(int x) {
    for(auto v : vec) {
        if(v == x) return 0;
    }
    vec.push_back(x);
    return x;
}
```

- `check_delete(int x)` will traverse `vec` and check if `x` is already in the vector. If it is, swap it with the last element of `vec`, `pop_back`, and return `x`. If it is not found, return `0`.

```
int check_delete(int x) {
    for(int i = 0; i < vec.size(); ++i) {
        if(x == vec[i]) {
            int tmp = vec[vec.size() - 1];
            vec[vec.size() - 1] = x;
            vec[i] = tmp;
            vec.pop_back();
            return x;
        }
    }
    return 0;
}
```

- `check_select(int x)` will first check if `x` is less then the size of `vec`. If it is, sort the vector and return `vec[x - 1]`. Else, return `0`.

```
int check_select(int x) {
    if(x <= vec.size()) {
        sort(vec.begin(), vec.end());
        return vec[x - 1];
    } else return 0;
}
```

- `check_rank(int x)` will sort the vector and then traverse the list. If `x` is found, return the (index of `x`) + 1. If it is not found, return `0`.

```
int check_rank(int x) {
    sort(vec.begin(), vec.end());
    for(int i = 0; i < vec.size(); ++i) {
        if(vec[i] == x) return i + 1;
    }
    return 0;
}
```

And here is the checker code. Interpret `opt_seq` and perform `check_[op]`, then compare it with the given `out_seq`. If the result doesn't match in any point in time, return `false`.

```
bool check(int opt_seq[], int in_seq[], int out_seq[], int n) {
    string str = "";
    for(int i = 0; i < n; ++i) {
        if(opt_seq[i] == 0) {
            if(check_insert(in_seq[i]) != out_seq[i]) {
                printf("%d: WA on I %d\n", i, in_seq[i]);
                return false;
            }
        } else if(opt_seq[i] == 1) {
            if(check_delete(in_seq[i]) != out_seq[i]) {
                printf("%d: WA on D %d\n", i, in_seq[i]);
                return false;
            }
        } else if(opt_seq[i] == 2) {
            if(check_select(in_seq[i]) != out_seq[i]) {
                printf("%d: WA on S %d\n", i, in_seq[i]);
                return false;
            }
        } else if(opt_seq[i] == 3) {
            if(check_rank(in_seq[i]) != out_seq[i]) {
                printf("%d: WA on R %d\n", i, in_seq[i]);
                return false;
            }
        }
```

```
    }
    return true;
}
```

3 Experiments

Here is how the test was done.

- Test environment
  - OS: Ubuntu 18.04.2 LTS
  - CPU: Intel Core i5-6200U CPU @ 2.30GHz × 4
  - RAM: 8GB
- Test Case Generator (Python)
  We keep the numbers small so that the numbers generated in the test will overlap easily, which will cause more operations to return some value other than 0.

```python
from random import randint

# Select random element from list
def rndselect(list):
    idx = randint(0, len(list) - 1)
    return list[idx]

# Number of Test Cases
N = 100000

# Operation set
op_set = ['I', 'D', 'R', 'S']

# Numbers
num = [str(i) for i in range(1, 100)]

# Create input text file
f = open('tests/input', "w")

# Write to test file
f.write(str(N) + "\n")
for i in range(N):
    f.write(rndselect(op_set))
    f.write(' ')
    f.write(rndselect(num))
    f.write('\n')
f.close()

```

- Checker Code

  This is the checker code I implemented, to check the correctness of my implementation. An helper function `printSideways` was used to examine the tree structure easily.

```cpp
#include <cstdio>
#include "hw2_cpp.h"
#define N 101010

extern Tree tree; // Tree defined in hw2_cpp.h

int opt_seq[N], in_seq[N], out_seq[N];

int main() {
    string str = "";

    freopen("./tests/input", "r", stdin);
    string op;
    int x, t;
    cin >> t;
    for(int i = 0; i < t; ++i) {
        cin >> op >> x;
        in_seq[i] = x;
        printf("%d. %c %d : ", i + 1, op[0], x);
        if(op[0] == 'I') {
            opt_seq[i] = 0;
            out_seq[i] = os_insert(x);
        } else if(op[0] == 'D') {
            opt_seq[i] = 1;
            out_seq[i] = os_delete(x);
        } else if(op[0] == 'S') {
            opt_seq[i] = 2;
            out_seq[i] = os_select(x);
        } else if(op[0] == 'R') {
            opt_seq[i] = 3;
            out_seq[i] = os_rank(x);
        }
        printf("%d\n", out_seq[i]);
        tree.printSideways(tree.getRoot(), str);
    }

    puts("Checking...");
    if(check(opt_seq, in_seq, out_seq, t)) puts("correct");
    else puts("incorrect");
}
```

- `printSideways` function

Prints the *data* and *size* of each node, with colors embedded.

```
void Tree::printSideways(Node* node, string& indent) {
    string str = indent + "        ";
    if(node != NIL) {
        printSideways(node -> right, str);
        if(node -> color == RED) {
            cout << indent;
            printf("\033[1;31m(%d / %d)\033[0m\n", node -> data,
    node -> size);
        } else {
            cout << indent;
            printf("\033[1;30m(%d / %d)\033[0m\n", node -> data,
    node -> size);
        }
        printSideways(node -> left, str);
    }
}
```

- Test case generator code was saved to `tester.py`, the checker code was saved to `os_test.cpp`.
  - Generate Test: `python3 tester.py`
  - Compile: `g++ os_test.cpp -o test`
  - Run: `./test`

3.1 Test Case #1

I tried a small test case first, with small numbers.
10 I 3 I 13 I 2 S 1 R 6 R 5 D 10 I 17 I 7 I 8
Here is the structure of the tree after each operation. Each node is represented by (*key*/*size*). The leftmost node is the root, and by each column, the nodes in the next level are shown.

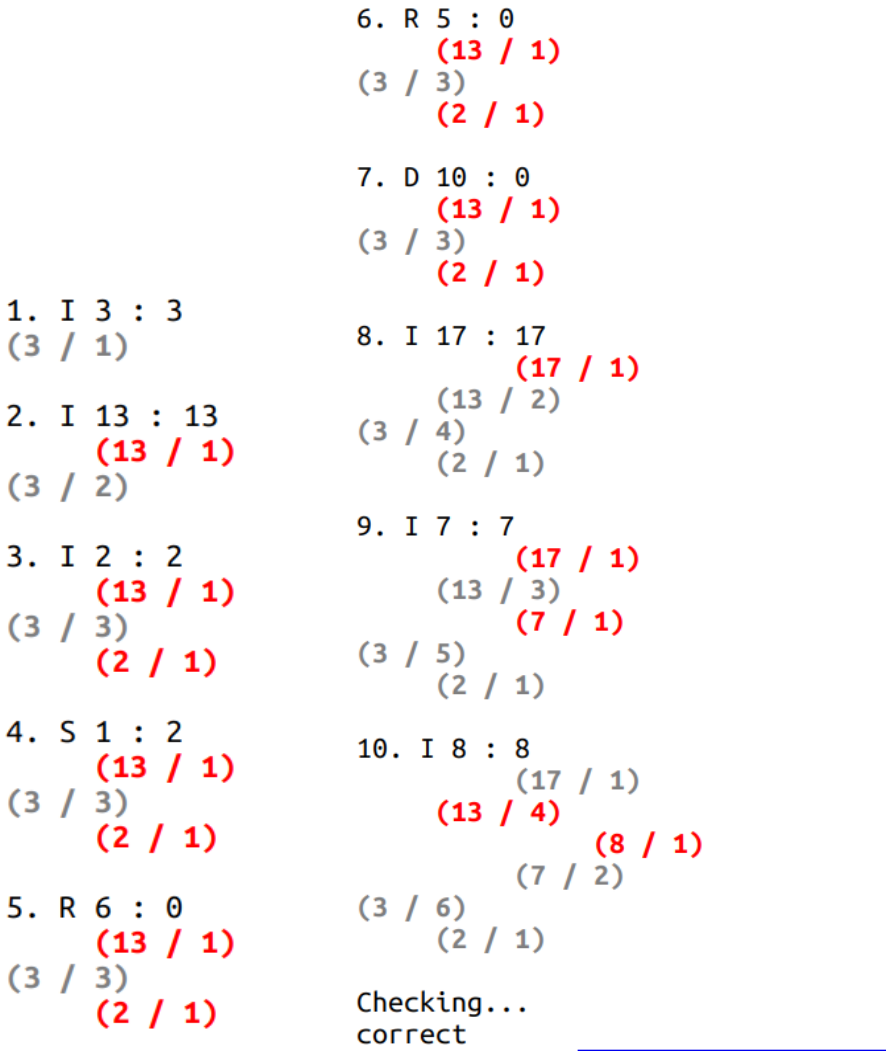

Figure 1: Result of Test 1

3.2 Test Case #2

For the second case, I tried a case with size 30.

30 I 9 I 1 S 12 R 19 R 14 S 6 S 11 R 9 I 3 D 10 S 5 S 17 D 13 I 19 I 19 S 6 R 3 S 7 I 1 S 10 R 8 I 3 I 10 I 2 I 11 I 15 I 13 I 16 R 15 D 6

Figure 2: Result of Test 2

I ran many other cases, up to 100000 operations. They were all correct.

4 Conclusion

In this homework, I learned how to augment data structures to contain additional information, that will help me retrieve particular information. With the augmented data structure, the additional field could be calculated in $O(1)$ time, since it only needs the information from left/right child. In total, `rank`, `select` takes $O(\log n)$ time with the order statistics tree, since the height of a red black tree is at most $O(\log n)$.