Project

Red-black Tree

Group 17

Date:2022-04-23

Chapter 1: Introduction

Red Black Tree is a self-balancing binary search tree, a data structure used in computer science, typically used to implement associative arrays.

It has the following 5 properties:

> (1) Every node is either red or black.
>
> (2) The root is black.
>
> (3) All the leaves are NULL nodes and are colored black.
>
> (4) Each red node must have 2 black descends (may be NULL).
>
> (5) All simple paths from any node x to a descendant leaf have the same number of black nodes.

We call a non-NULL node an internal node. From property 5 we can define the black-height of a red-black tree as the number of nodes on the simple path from the root (excluding the root itself) to any NULL leaf (including the NULL leaf). And we can derive that a red-black tree with black height H has at least $2^{H}-1$ internal nodes.

For *N* internal nodes, there are many distinct red-black trees. In this project, we are trying to find the number of distinct red-black trees with *N* internal nodes and compare the time it used to find for different *N*.

For the testing purpose, we just abandon the input, and give out all output corresponded.

Chapter 2: Algorithm Specification

Main Program

The main program use dfs by enumerating the black height of the tree to get the answer.

```
signed main(signed, char**, char**) {
    Input n;

    For i = 1 To n Step 1
        ans := ans + dfs(n, i, BLACK);    // BLACK is referred to as 1
        ans := ans % MOD;    // MOD is a constant value 1000000007

    Output ans;
}
```

mul

This part is to multiply the two given parameters, using long long to avoid overflow.

```
int mul(long long a, long long b) {
    return a * b % MOD;
}
```

dfs

This part is to search the subtree and then memorize the number of different trees with size **len**, black height **bht**, root color **color**.

```
Procedure dfs(len:integer, bht:integer, color:integer)
    // Memorized Search on a subtree. parameters:
    // len: number of nodes in that subtree;
    // bht: black height of that subtree;
    // color: the color of that root.

    if (bht < 0) then
        return 0;    // BHT<0, impossible
    if (len = 0) then
        return color = BLACK && bht = 0;
                    // NULL node, must be BLACK and BHT=0
    int& ret := res[len][bht][color];
    if (calc[len][bht][color])
        then
            return ret;
        else
            calc[len][bht][color] := 1;    // If calculated, return
    bht := bht - color;                    // Get BHT of its sons(SBHT)
    double lb := pow(2, bht) - 1;
    For i = lb to len - lb - 1 Step 1 {
        // Enumerate number of nodes of its two sons
        if (color = RED)
            then {
                ret := ret +
                    mul(dfs(i, bht, BLACK), dfs(len - i - 1, bht, BLACK));
                ret := ret % MOD;
                // Two sons must be BLACK, SBHT=BHT, get the result
            }
            else {
                ret := ret +
                    mul(dfs(i, bht, BLACK), dfs(len - i - 1, bht, BLACK));
                ret := ret % MOD;
                ret := ret +
                    mul(dfs(i, bht, RED), dfs(len - i - 1, bht, BLACK));
                ret := ret % MOD;
                ret := ret +
                    mul(dfs(i, bht, BLACK), dfs(len - i - 1, bht, RED));
                ret := ret % MOD;
                ret := ret +
                    mul(dfs(i, bht, RED), dfs(len - i - 1, bht, RED));
                ret := ret % MOD;
                // Two sons can be any color, SBHT=BHT-1, add up the result
            }
    }
    return ret;    // return the caculated result
}
```

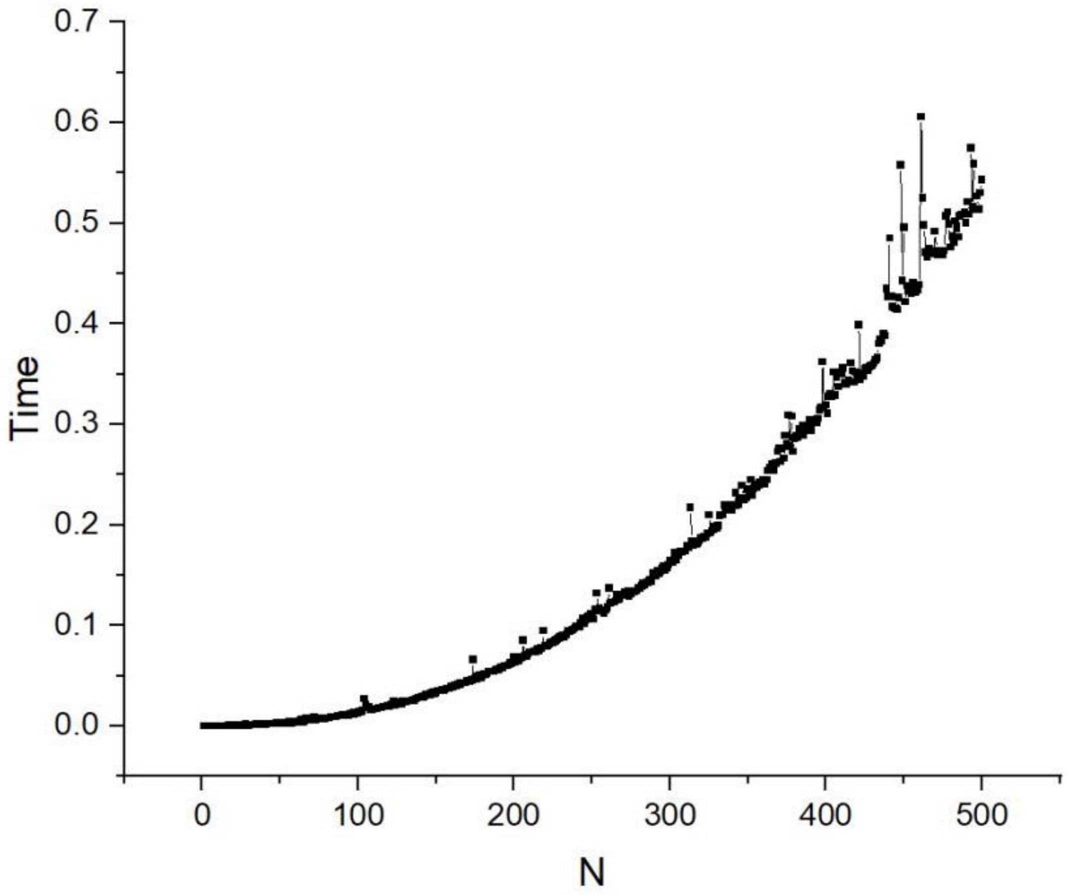
Chapter 3: Testing Results

Since the number of distinct red-black trees with *N* (≤500) internal nodes may be very large, we will output the remainder of it divided by 1000000007. And for the time, we use the average time of running it for 10 times.

Testing results

The running results have been stored in out.txt, and you can look up it to find the results. These results have been verified to be correct. And you can test its correntness in PTA-Top 1007.

Compare the time it used to find the number of distinct red-black trees with *N* (≤500) internal nodes for different N:



Apparently it nearly forms a parabola.

Chapter 4: Analysis and Comments

Space Complexity

- The program use $O(N^2)$ space to store the results.

Time Complexity

Memorized Search can be converted to Dynamic Programming.

using 3 parameters to describe the state, we have $O(N^2)$ states. Each state relies on $O(N)$ other states. So in that theory the time complexity shall be $O(N^3)$.

However, as BHT is limited to arround $[\frac{log(LEN)}{2}, log(LEN)]$, we can expect most of the states cannot even get 1 transmission from other states. Apparently we have about $O(NlogN)$ states to be calculated.

Each state rely on ($LEN - 2^{BHT+1}$) states. That's $O(N - \sqrt{N})$ - also $O(N)$. So we can actually expect the time complexity $O(N^2 logN)$.

Appendix(README.txt)

```
We suppose read the source file to get more comments. The file:
main.cpp is used to solve the problem. This program is by Zuo.

You can see the output in out.txt. We can guarantee its correctness.

The report is by Han and Shi.

We recommend using DEV-C++ v5.15 to open the project, and TDM-
GCC9.2.0 to compile.
The C++ standard shall be C++11 or above.
```