Red-black Tree

Group 1 Wang Kedi / Shen Yunfeng / Zhao Yilei

Date: 2022-04-14

Chapter 1: Introduction

Red-black Tree is a special kind of binary tree which possesses the following properties:

```
1. Every node is either BLACK or RED;
2. The root is BLACK;
3. ALL the leaves are NULL and BLACK;
4. Each RED node must has 2 BLACK children;
5. All simple path from node X to a descendant leaf have the same number
of BLACK nodes.
```

We can define the Black-Height as the number of BLACK node in any path from the current node to descendant leaves.

Here comes the question: given a positive number N, how can we determine the number of distinct red-black trees owning exactly N internal nodes?

We solve it by applying Dynamic Programming: by dividing the sub-problems by the color of root node(RED or BLACK) , the number of internal node and the black-height of the tree, we can construct the table and solve the problem.

Chapter 2: Data Structure / Algorithm Specification

Data Structure : 2D Array

Array resultB[][] and resultR[][] store the number of distinct BLACK root / Red root read-black tree.

The first index denotes the number of internal node, and the second one denotes the black-height of the tree.

Algorithm Specification :

As a red-black tree with black-height H has at least 2^H - 1 internal nodes, it is easily to denote that the maximum & minimum black-height of a red-black tree possessing N internal nodes are:

$$\begin{cases} minBH = log_2(N+1)/2 - 1 \\ maxBH = log_2(N+1) \end{cases}$$

To solve the problem, we now define a RED ROOT red-black tree. We can easily derive from the properties of red-black tree that both the left and right subtree of the root are also red-black tree.

- Case 1: subtrees of BLACK ROOT red-black tree
    - 2 * BLACK ROOT red-black tree of the same black-height
    - 2 * RED ROOT red-black tree of the same black-height
    - 1 * BLACK ROOT red-black tree + 1 * RED ROOT red-black tree of the same black-height

  Combining all three sub-cases, we would have:

$$
\begin{aligned}
&i \; is \; the \; total \; number \; of \; internal \; nodes, \\
&j \; is \; the \; black-height \\
resultB[i][j] = resultB[i][j] \quad &+ \\
&(resultB[k][j-1] + resultR[k][j]) \quad \times \\
&(resultB[i-1-k][j-1] + resultR[i-1-k][j]) \\
&k \in [0, i-1]
\end{aligned}
$$

But in this problem, we only need to give the remainder of the result divided by 1000000007, therefore we can modify it into:

$$
\begin{aligned}
resultB[i][j] = resultB[i][j] + \\
&(\quad resultB[k][j-1] \quad + \quad resultR[k][j] \quad) \; mod \; 1000000007 \times \\
&(resultB[i-1-k][j-1] + resultR[i-1-k][j]) \; mod \; 1000000007 \\
&, \; k \in [0, i-1]
\end{aligned}
$$

- Case 2: subtrees of RED ROOT red-black tree
    - 2 * BLACK ROOT red-black tree of the same black-height

  Thus:

  $$resultR[i][j] = resultR[i][j] + (resultB[k][j-1] \times resultB[i-1-k][j-1]), \; k \in [minBh, maxBH]$$

  Similarly, we modify it into:

  $$resultR[i][j] = resultR[i][j] + (resultB[k][j-1] \times resultB[i-1-k][j-1]) \; mod \; 1000000007, \\ k \in [minBh, maxBH]$$

- Final result:

  By calculating resultB and resultR with n = 3, 4, ..., N, the total number of red-black tree possessing exactly N internal nodes would be:

  $$result = \sum_{i=N}^{0} resultB[N][i]$$

  For this problem, we modify it into:

  $$result = \sum_{i=0}^{N} resultB[N][i] \; mod \; 1000000007$$

Pseudo Code:

```
long long int DP(int n)//n is the number of internal nodes
{
    long long int resultB[501][501], resultR[501][501];
    long long int result = 0;
    //we have no need to deal with the cases which those subtrees have
no node
    resultB[1][1] = result[1][1] = 1;
    resultB[2][1] = 2;

    for(int i = 3; i <= n; i++)//i is the current total number of
internal nodes
    {
        //as n >= 2^bh -1:
        int minBh = log2(total+1)/2 - 1;
        int maxBh = log2(total+1);
```

```
        for(int bh = minBh; bh <= maxBh; bh++)
        {
            if(!bh) continue;//avod index crossing the border
            for(j = 1; j < total-1; j++)
            //j is the number of the nodes in the left subtree
            {
                resultB[total][bh] += ((resultB[j][bh-1] + resultR[j]
[bh]) % 1000000007) *
                                      ((resultB[total-j-1][bh-1] +
resultR[total-j-1][bh]) % 1000000007);
                resultR[total][bh] += ( resultB[j][bh-1] *
resultB[total-j-1][bh-1]) % 1000000007;
                resultB[total][bh] %= 1000000007;
                resultR[total][bh] %= 1000000007;
            }
        }
    }

    for(int i = 0; i <= n; i++)
        result = (result + resultB[n][i]) % 1000000007;
    return result;
}
```

Chapter 3: Testing Result

Case 1: The miniest sample

The number of nodes is 1, therefore, the black-red tree will only have 1

Status:

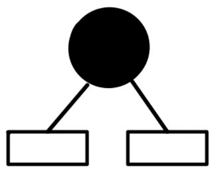
Pass

Input:

```
1
```

Output

```
1
```



Case 2: Normal sample

Status:

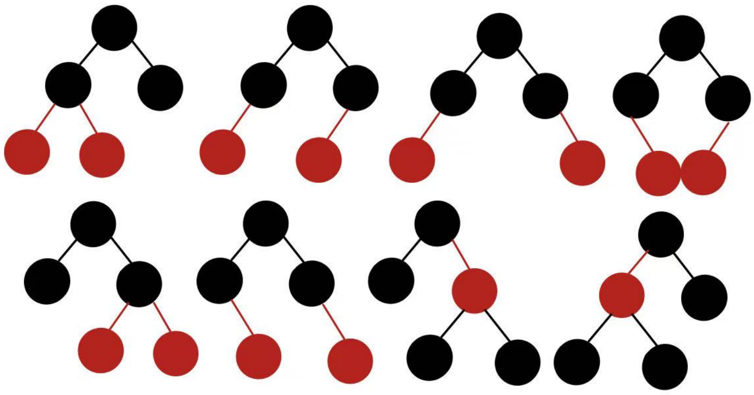
Pass

Input:

```
5
```

Output:

```
8
```



Case 3: Sample with larger input

Status:

Pass

Input:

```
100
```

Output:

```
167844408
```

Case 4: The maximum of the input

Status:

Pass

Input:

```
500
```

Output:

```
905984258
```

The time cost of the maximum sample is 20ms, which is under the time limits of the problem.

Chapter 4: Analysis and Comments

Time Complexity:

Reading the input and initialization would take O(1).

In the process of dynamic programming:

$$
\begin{aligned}
T(N) &= \sum_{i=3}^{N} 4*(i-2)*(maxBH - minBH) \\
&= \sum_{i=3}^{N} 4*(i-2)*(log_2(i+1) - (\frac{log_2(i+1)}{2} - 1)) \\
&= \sum_{i=3}^{N} 4*(i-2)*(\frac{log_2(i+1)}{2} + 1) \\
&= 2*(N+7)*(N-2) + 2*\sum_{k=4}^{N+1}(k-3)*log_2 k \\
&< 2*(N+7)*(N-2) + 2*\sum_{k=4}^{N+1}(k-3)*log_2(N+1) \\
&= 2N^2 + 10N - 28 + (N-1)*(N-4)*log_2(N+1) \\
&= O(N^2 log_2(N))
\end{aligned}
$$

And the process of calculating the sum is of O(n).

By combing all the three part, the time complexity is between O （n^2log(n)).

Space Complexity:

The size of 2D array resultB and resultR are both **n*logn**, which means it takes **2n*logn** in total. Thus, the space complexity of the whole program is O(n*logn).

Appendix: Source Code

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define mod 1000000007
long long int resultB[501][501];
```

```
long long int resultR[501][501];

int main()
{
    int n;
    scanf("%d", &n);
    resultB[1][1] = resultR[1][1] = 1;
    resultB[2][1] = 2;
    int i;
    int bh, total, minBh, maxBh;
    //when we start from 3, there's no need to deal with cases when a
subtree has 0 node
    for(total = 3; total <= n; total++) //total internal nodes
    {
        //we know total >= 2^bh - 1, bh >= h/2, (total <= 2^(h+1)-1),
log2(total+1)/2 - 1 <= bh <= 2*log2(total+1)
        minBh = log2(total+1)/2 - 1;
        maxBh = log2(total+1);
        for(bh = minBh; bh <= maxBh; bh++)
        {
            if(!bh) continue;
            for(i = 1; i < total-1; i++)    //i is the number of the
nodes in the left subtree
            {
                resultB[total][bh] += ((resultB[i][bh-1] + resultR[i]
[bh]) % mod) * ((resultB[total-i-1][bh-1] + resultR[total-i-1][bh]) %
mod);
                resultR[total][bh] += (resultB[i][bh-1] * resultB[total-
i-1][bh-1]) % mod;
                resultB[total][bh] %= mod;
                resultR[total][bh] %= mod;
            }
        }
    }
    long long int result = 0;
    for(i = 0; i <= n; i++)
        result = (result + resultB[n][i]) % mod;
    printf("%lld\n", result);

    system("pause");
    return 0;
}
```

References

[1] Introduction to Algorithms(3rd Edition), Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, The MIT Press 2009.

Author List

Name	Work
Wang Kedi	Code
Shen Yunfeng	Report-Chap 1,2,4
Zhao Yilei	Test & Report-Chap 3

Declaration

We hereby declare that all the work done in this project titled "pj4" is of our independent effort as a group.

Signatures

Each author must sign his/her name here: