Chapter 1： Introduction

This project gives the definition of red black tree and requires us to give the composition of all red black trees with n internal nodes.

Chapter 2: Algorithm Specification

Black Trees

```
1. Each node is either red or black.
2. The root node is black.
3. Each leaf node, or nil node (or null node) is black.
4. If a node is red, its father cannot be red and its children
cannot be red. In short, two consecutive red nodes are not allowed
in the red black tree.
5. For each node, the simple path from the node to all leaf nodes
of its descendants contains the same number of black nodes.
```

Solution Manual

```
class Solution
{
    //根节点为黑色，节点数为n，黑高为bh的红黑树个数为
distinct_black_root_n[n][bh]
    std::vector<std::vector<long long>> distinct_black_root_n;
    //根节点为红色，节点数为n，黑高为bh的红黑树个数为distinct_red_root_n[n]
[bh]
    std::vector<std::vector<long long>> distinct_red_root_n;

    //由节点数计算黑高上界
    long long get_max_bh(long long node_n);

    //初始化数组
    void init(long long node_n);
```

```
    //根节点为黑的情况，左右为黑高相等的（黑+红）*（黑+红）
    //根节点为红的情况，左右为黑高相等的（黑*黑）
    //为简化动规代码，同时省略数组的最小条件初始化，这里定义两个动规过程辅助函数

    //返回n个节点，黑高为bh的黑根树+红根树个数
    long long get_r_and_b(long long n, long long bh) ;

    //返回n个节点，黑高为bh的黑根树个数
    long long get_b(long long n, long long bh) ;

    //以上两个函数包括了节点数为0的所有情况，因此动规从节点数为1开始

public:
    //用于输出时候取余数
    long long mod = 1000000007;
    //求解模块
    long long solve(long long node_n) ;
};
```

Core Solution Algorithm

```
    long long Solution::solve(long long node_n) {
        init(node_n);
        //从1个节点解到n个节点
        for (long long n = 1; n <= node_n; ++n) {   //O(N)
            //最大黑高
            auto max_bh = get_max_bh(n);    //O(logN)
            //遍历黑高
            for (long long bh = 0; bh <= max_bh; ++bh) {
                //循环开始左节点数
                long long begin = 0;    //O(N)
                //分配左右节点数
                for (
                    long long left_node_n = begin, right_node_n = n
- 1 - begin;
                    right_node_n >= begin;
                    ++left_node_n, --right_node_n
                    )
                {
                    //至此总共三层循环，总时间复杂度O(N^2logN),在计算的时候
就直接求余，防止溢出
```

```
                    distinct_black_root_n[n][bh + 1] +=
get_r_and_b(left_node_n, bh) * get_r_and_b(right_node_n, bh) % mod;
                    distinct_red_root_n[n][bh] +=
get_b(left_node_n, bh) * get_b(right_node_n, bh) % mod;
                    distinct_black_root_n[n][bh + 1] %= mod;
                    distinct_red_root_n[n][bh] %= mod;
                }
            }
        }
        //我们只需要根节点为黑的解
        long long ans = 0;
        auto max_bh = get_max_bh(node_n);
        for (long long bh = 0; bh <= max_bh; ++bh) {
            ans += get_b(node_n, bh);
            ans %= mod;
        }
        return ans;
    }
```

Chapter3: Testing Result

> *Attention: The green value will be input, the white is output of program.*

Sample:

```
5
8
```

Min N(positive N must start from 1):

```
1
1
```

- In fact, when there is only one node, there is only one possibility

Easy Case:

```
2
2
```

```
3
2
```

```
3
2
```

- These cases can be verified manually

Complex Case

```
10
164
```

- The procedures of the same group members have been tested and the results are the same

Max N(lower than 500)

```
500
905984258
```

- The procedures of the same group members have been tested and the results are the same

- Since the following data is too large for manual verification, after the team members' algorithms are verified each other, the topic scoring system of PTA system is used for verification, which can be AC

Chapter 4: Analysis and Comments

Time complexity:

- get_ r_ and_ B and get_ B the time complexity of the two functions is O (1) because they directly obtain the return data from the dynamic programming array
- In the problem-solving algorithm, a total of three loops are called. Firstly, n loops are calculated from n = 1 to n = n (input); Traverse all black heights. Since the maximum black height should be log (n), log (n) cycles are carried out here; The number of left and right nodes is allocated and N cycles are carried out, so the overall time complexity should be $O(N^2log_2(N))$

Space complexity:

- When initializing the array, a total of $2Nlog_2(N)$ long long int spaces are opened up, So the spatial complexity is $O(Nlog_2(N))$

Declaration

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