Red-black Tree

Date: 2022-04-22

chapter 1: Introduction

In computer science, a red-black tree is a self-balance binary search tree. Each node stores an extra bit of representing "color" ("red" or "black"). By constraining the color of each node on the simple path from the root node to the the leaf node, it is ensured that the tree is balanced during insertion and deletion. It can find, insert, and delete in O(lg n) time, where N is the number of nodes in the tree. The precise definition of the red and black tree is as follows:

Each node of the tree contains the attributes color, key, left, right, and p. If a child or the parent of a node does not exist, the corresponding pointer contains the value NIL. We regard NIL as the external node of the red-black tree. The red-black tree satisfy the following five properties:

1. Every node is either red or black.
2. The root is black.
3. Every leaf (NIL) is black
4. If a node is red, then both its children are black.
5. For each node, all simple paths from the node to descendant leaves contain the same number of black nodes.

Due to these properties of the red-black tree. There are more than one structure with N internal nodes. Our project is to answer how many distinct red-black trees are there that consist of exactly N internal nodes. Since the answer may be very large, we will output the remainder of it divided by 1000000007.

chapter 2: Algorithm Specification

```
long long vr[maxlen][maxlen], vb[maxlen][maxlen];//vr is red node, vb is black
node
int N;//all nodes
long long sum = 0;//sum of sulutions
//the initialized conditions
vb[1][1] = vr[0][1] = 1;
vb[1][2] = 2;
//use dp to calculate
for loop1: i = 0~100
    for loop2: j = 2 ~ N-1
        for loop3: k = 1 ~ j;
            long long m1=(vb[i][k] * vb[i][j - k]) % OVERFLOW;
            long long m2=(vr[i][k] + vb[i][k]) * (vr[i][j - k] + vb[i][j - k]) %
OVERFLOW;
            vr[i][j + 1] += m1 % OVERFLOW;
            vb[i + 1][j + 1] += m2 % OVERFLOW;
            vr[i][j + 1] %= OVERFLOW;
            vb[i + 1][j + 1] %= OVERFLOW;
//add up all solutions
for(int i = 0; i <= N; i++) sum = (sum + vb[i][N]) % OVERFLOW;
```

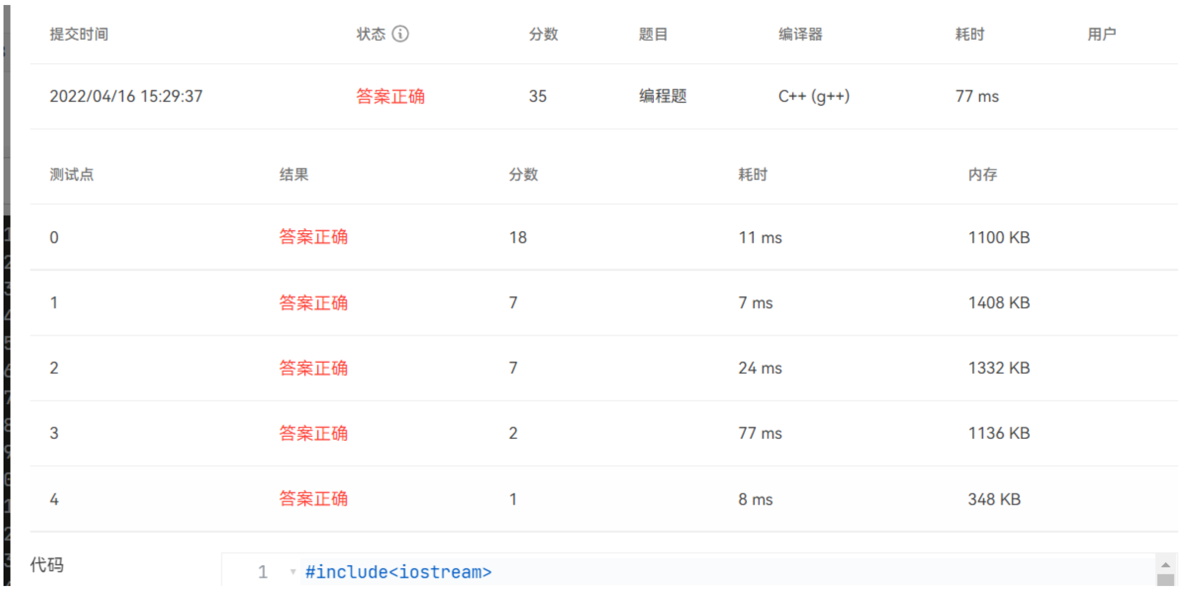
Let's focus on the dp algorithm, firstly set the vb [1] [1] = vr[0] [1] = 1; vb[1] [2] = 2, these are the conditions when there are one black, one red, one red + one black. then use DP in the for loop. Loop i stands for the black height of the tree, since the N is below, the BK is definitely no larger then 100, so i from 0~100 is enough. Loop j stands for the internal nodes, since we have already considered the one node condition, here j starts from 2. k is used to scan the array and add up the conditions.

Since the sum may overflow out of the large amount of solutions, here we use result % OVERFLOW to keep the correctness of the result.

Testing results

Test for correctness

our code past the PAT (Top Level) Practice 1007



提交时间	状态	分数	题目	编译器	耗时	用户
2022/04/16 15:29:37	答案正确	35	编程题	C++ (g++)	77 ms	

测试点	结果	分数	耗时	内存
0	答案正确	18	11 ms	1100 KB
1	答案正确	7	7 ms	1408 KB
2	答案正确	7	24 ms	1332 KB
3	答案正确	2	77 ms	1136 KB
4	答案正确	1	8 ms	348 KB

代码

```
#include<iostream>
```

Comment and analysis

The time complexity:
  There is a there-for loop of the DP, the i loop is consider as O(log(2*N+c)), the j loop is O(N), the k loop is O(N), so the total time complexity is O(N^2 * log(N))

The space complexity:
  there are two longlong [n] [n] matrix and some contents, so the space complexity is O(N^2)

Appendix : Source code

```
#include<iostream>
#include<vector>
#define maxlen 501
#define OVERFLOW 1000000007
using namespace std;

long long vr[maxlen][maxlen], vb[maxlen][maxlen];
//vr stands for red-root node conditions; vb stands for black-root node
conditions;

int main(){
    int N;//the number of internal nodes
    long long sum = 0;//the sum of all the conditions
    cin>>N;
    vb[1][1] = vr[0][1] = 1;//define the conditions when there is at least a nil child of the root
    vb[1][2] = 2;
    for(int i = 0; i < 100; i++){//i stands for the black height
        for(int j = 2; j < N; j++){
            //j stands for the internal nodes, since we have already considered the one_node condition, here j starts from 2
            for(int k = 1; k < j; k++){//k is used to scan the array and add up the conditions
                long long m1=(vb[i][k] * vb[i][j - k]) % OVERFLOW;//the child nodes of the red-root
                long long m2=(vr[i][k] + vb[i][k]) * (vr[i][j - k] + vb[i][j - k]) % OVERFLOW;//the child nodes of the black-root
                vr[i][j + 1] += m1 % OVERFLOW;
                vb[i + 1][j + 1] += m2 % OVERFLOW;
                vr[i][j + 1] %= OVERFLOW;
                vb[i + 1][j + 1] %= OVERFLOW;
            }
        }
    }
    for(int i = 0; i <= N; i++) sum = (sum + vb[i][N]) % OVERFLOW;//since the red-black-tree must have a black root, count all the conditions in array vb[]
    cout<<sum;
    return 0;
}
```

Declaration

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

Signatures