**Red-black Tree**

### Chapter 1: Introduction

This project is aim to find how many distinct red-black tree are there given that there are N(<500) internal nodes in the red-black tree.(The answer should be the remainder of it divided by 1000000007 )

First of all , you should know what is red-black tree. Red-black tree is a kind of binary tree that has 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.

From last property 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 2H– 1 internal nodes.

With the property above this project should ues dynamic programming to solve the problem.

### Chapter 2: Algorithm Specification

To use dynamic programming method , we should find out whether this problem has overlapping subproblems and optimal substructure. It’s clear that if BH(aka black-height,we will call it BH from now on) satisfy the property above then given a root ,its left child tree and right child tree have the same BH ,otherwise it will have contradiction. So here comes to mind that if a red-black tree’s BH is x,then its left subtree and right subtree both have x-1 as its BH.

However, its sub tree may have a red node as its root , so here we define another kind of red-black tree whose property is the same with original red-black tree except it has red node as its root.

After a new structure was defined ,we can define our dp states. In this project I ues two dimension to describe its states , BH and N. dp\_b[BH][N] means the number of distinct red-black trees which have BH as its black-height and N as its numbers of internal nodes. dp\_r[BH][N] is the same except its root is red node.

So we can have our dp function below:

dp\_b[BH][N] = (dp\_b[BH-1][i]+dp\_r[BH-1][i])\*(dp\_b[BH-1][N-1-i]+dp\_r[BH][N-1-i]) i=0,1,2,3……N-1

dp\_r[BH][N] = dp\_b[BH][i]\*dp\_b[BH][N-1-i] i=0,1,2,3……N-1

(Notice that red root tree must have two black root tree as its subtree)

The initial condition: dp\_b[0][0] = 1 , dp\_b[0][i] = 0 i=1,2,3……N

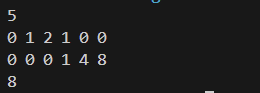
dp\_r[0][1] =1 ,dp\_r [0][i] = 0 i= 0,2,3……N

Because of the huge number of distinct trees are there so each answer in the dp table should be moded 1000000007 , which can avoid overflowing.

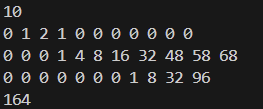
### Chapter 3: Testing Results

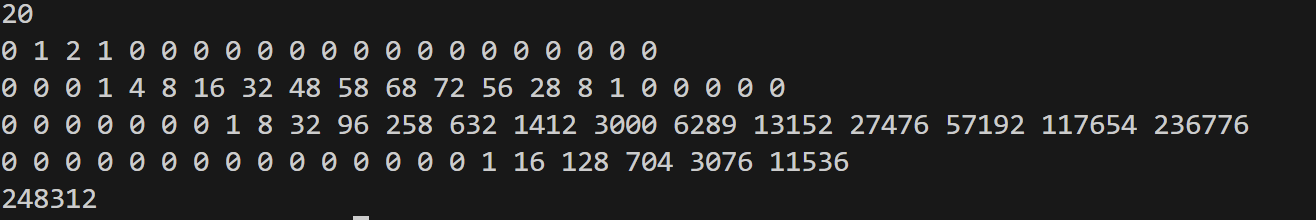
Each case output the dp\_b table and the final result

Eg input:



other input:





N=500 input:(The dp table is too large to print,so I didn’t print it out)



### Chapter 4: Analysis and Comments

Let assume that the input num is N, and the BH is M,its obvious that the time complexity is O(M\*N2) ,because there are total M line to scan ,and each line has N numbers , we should add 0 +N-1 , 1+ N-2 … N-1 + 0 for a sigle num which means for a num we need O(N) and a line O(N2).So total is O(M\*N2) . And by the property of red-black tree we know that if a red-black tree has BH then its minimal internal nodes number is 2BH-1 ,which means if we have N then the maximal BH is logN. So the real time complexity is O(N2logN).

Because the dp table is M\*N , so the space complexity is O(N\*logN)

### Appendix: Source Code (in C)

#include <stdio.h>

#include <stdlib.h>

int main(){

int n,i,j,k;

scanf("%d",&n);

long long int\*\* dp\_b = (long long int\*\*)malloc((n+1)\*sizeof(long long int\*));

long long int\*\* dp\_r = (long long int\*\*)malloc((n+1)\*sizeof(long long int\*));

for(i=0;i<n+1;i++){

dp\_b[i] = (long long int\*)malloc((n+1)\*sizeof(long long int));

dp\_r[i] = (long long int\*)malloc((n+1)\*sizeof(long long int));

}

//初始化

for(i=0;i<n+1;i++){

for(j=0;j<n+1;j++){

if(i==0&&j==0)dp\_b[i][j]=1;//根节点不存在的情况下才有可能

else dp\_b[i][j]=0;

if(i==0&&j==1)dp\_r[i][j]=1;//根节点为红色的情况下才可能

else dp\_r[i][j]=0;

}

}

//dp状态转移

int twosquare = 1;

int minnode = twosquare-1;

for(i=1;i<n+1;i++){

twosquare \*= 2;

minnode = twosquare-1;

if(minnode>n)break;

for(j=0;j<n+1;j++){

for(k=0;k<j;k++){

dp\_b[i][j] = (dp\_b[i][j]+(((dp\_b[i-1][k]+dp\_r[i-1][k])%1000000007)\*((dp\_b[i-1][j-k-1]+dp\_r[i-1][j-k-1])%1000000007))%1000000007)%1000000007;//更新黑表

dp\_r[i][j] = (dp\_r[i][j]+((dp\_b[i][k]%1000000007)\*(dp\_b[i][j-k-1]%1000000007))%1000000007)%1000000007; //更新红表

}

// printf("%d ",dp\_b[i][j]);

}

// printf("\n");

}

int sum = 0;

for(j=0;j<i;j++){

sum+=dp\_b[j][n];

sum%=1000000007;

}

printf("%d\n",sum);

}

### Declaration

***I hereby declare that all the work done in this project titled "* *Red-black Tree" is of my independent effort.***