**Vijos1448校门外的树 题解**

**描述：**

校门外有很多树，有苹果树，香蕉树，有会扔石头的，有可以吃掉补充体力的……

如今学校决定在某个时刻在某一段种上一种树，保证任一时刻不会出现两段相同种类的树，现有两个操作：

K=1，K=1，读入l、r表示在区间[l,r]中种上一种树，每次操作种的树的种类都不同

K=2，读入l,r表示询问l~r之间能见到多少种树

（l,r>0）

**输入格式：**

第一行n,m表示道路总长为n，共有m个操作

接下来m行为m个操作

**输出格式：**

对于每个k=2输出一个答案

**样例输入：**

5 4

1 1 3

2 2 5

1 2 4

2 3 5

**样例输出：**

1

2

**数据范围：**

20%的数据保证，n,m<=100

60%的数据保证，n <=1000,m<=50000

100%的数据保证，n,m<=50000

原题地址：[https://vijos.org/p/1448](https://vijos.org/p/1448" \t "https://www.cnblogs.com/shadowland/p/_blank)

~~~~~~~~~~~~~~~~~~~~~~~~~~~分割线~~~~~~~~~~~~~~~~~~~~~~~~~~~

**分析：**

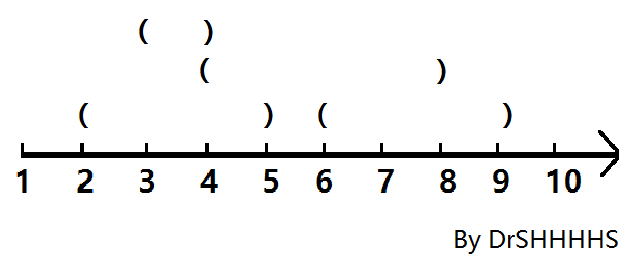
这道题是一个典型的区间问题，考虑到数据量较大，使用线段树完成这个操作。由于树的种类很多，不难想到用线段树暴力维护的方法。但是暴力维护一定会超时，那么这么解决这个问题呢？

这里介绍一种十分机智的想法——**括号序列**。

假设有一个长度为10的数轴，我们要将区间[ 2 , 5 ]中种树，这时，我们将 2 处放一个左括号 " ( "  ,5处放一个 " )"  ，表示区间 [ 2 , 5 ]种了树。

查询某个区间树的种类，如区间[ 3 , 10]，只需统计10之前（包括10）有多少个‘（’，统计3之前有多少个‘）’，（不包括3）。

如下图所示：

  
以上就是括号序列的过程。简单的说，就是更新区间[a,b]时，点a记录左括号数，点b记录右括号数，查询区间[a,b]时，即为b之前（包括b）的左括号数-a之前的右括号数。

下面贴注释代码：

#include "bits/stdc++.h"

#define maxN 50010

using namespace std ;

typedef long long QAQ ;

struct Tree

{

int l , r ;

QAQ liml , limr ;//左括号右括号

};

Tree tr[maxN << 2];

void Build\_Tree ( int x , int y , int i )//建树

{

tr[i].l = x ;

tr[i].r = y ;

if( x == y )return ;

else

{

QAQ mid = (tr[i].l + tr[i].r) >> 1 ;

Build\_Tree ( x , mid , i << 1);

Build\_Tree ( mid + 1 , y , i << 1 | 1);

return ;

}

}

void Update\_left ( int w , int i )

{

if( w == tr[i].l && w == tr[i].r )tr[i].liml++;//找到目标节点

else

{

QAQ mid = (tr[i].l + tr[i].r) >> 1 ;

if( w > mid )Update\_left( w , i << 1 | 1);//找右儿子

else if( w <= mid)Update\_left( w , i << 1 );//找左儿子

tr[i].liml = tr[i << 1].liml + tr[i << 1 | 1].liml ;//回溯更新

}

}

void Update\_right ( int w , int i )//同Update\_left

{

if( w == tr[i].l && w == tr[i].r )tr[i].limr++;

else

{

QAQ mid = (tr[i].l + tr[i].r) >> 1 ;

if( w > mid )Update\_right( w , i << 1 | 1);

else if( w <= mid)Update\_right( w , i << 1 );

tr[i].limr = tr[i << 1].limr + tr[i << 1 | 1].limr ;

}

}

QAQ Query\_left ( int q , int w , int i )//同Query\_right

{

if( q <= tr[i].l && w >= tr[i].r )return tr[i].liml ;

else

{

QAQ mid = (tr[i].l + tr[i].r) >> 1 ;

if ( q > mid )return Query\_left ( q , w , i << 1 | 1);

else if ( w <= mid ) return Query\_left ( q , w , i << 1);

else return Query\_left ( q , w , i << 1 | 1) + Query\_left ( q , w , i << 1);

}

}

QAQ Query\_right ( int q , int w , int i )

{

if( q <= tr[i].l && w >= tr[i].r )return tr[i].limr ;//找到目标区间直接返回

else

{

QAQ mid = (tr[i].l + tr[i].r) >> 1 ;

if ( q > mid )return Query\_right ( q , w , i << 1 | 1);//找右儿子

else if ( w <= mid ) return Query\_right ( q , w , i << 1);//找左儿子

else return Query\_right ( q , w , i << 1 | 1) + Query\_right ( q , w , i << 1);//左右儿子都查找

}

}

int main()

{

int N, M, op, ll, rr ;

scanf("%d %d", &N, &M);

Build\_Tree ( 1 , N , 1 ) ;//建树

while(M--)

{

scanf("%d%d%d", &op, &ll, &rr);

if( op == 1 )

{

Update\_left ( ll , 1);//添加左括号

Update\_right ( rr , 1 );//添加右括号

}

else

{

QAQ ans = Query\_left( 1 , rr , 1);

if (ll != 1)ans -= Query\_right(1 , ll - 1 , 1);//当ll不等于1时再相减，否则栈会炸

printf("%I64d\n", ans);

}

}

return 0 ;

}