线段树分治

关键字：时间分治、可撤消

引例：动态图连通性(NKOJ8444)

引例：动态图连通性(NKOJ8444)

给你一张n个点的无向简单图。有m次操作，操作如下三种：

0：**添加**一条边（保证它不存在）；

1：**删除**一条边（保证它存在）；

2：**查询**两个点是否连通；

n<=5000，m<=500000 时限1s

引例：动态图连通性(NKOJ8444)

解题分析：

每条边都存在于在一定时间范围内。

比如第a号操作添加了k号边，第b号操作删除了k号边，那么k号边存在的时间就是[a,b]；

把操作顺序看作时间，用线段树来维护时间区间；

4 9		
0 1 3	操作1，添加边1	边1：[1,5]
0 2 4	操作2，添加边2	边2：[2,6]
0 1 2	操作3，添加边3	边3：[3,9]
0 2 3	操作4，添加边4	边4：[4,9]
1 1 3	操作5，删除边1	
1 2 4	操作6，删除边2	
3 1 4	操作7，询问	
0 2 4	操作8，添加边5	边5：[8,9]
3 3 4	操作9，询问	

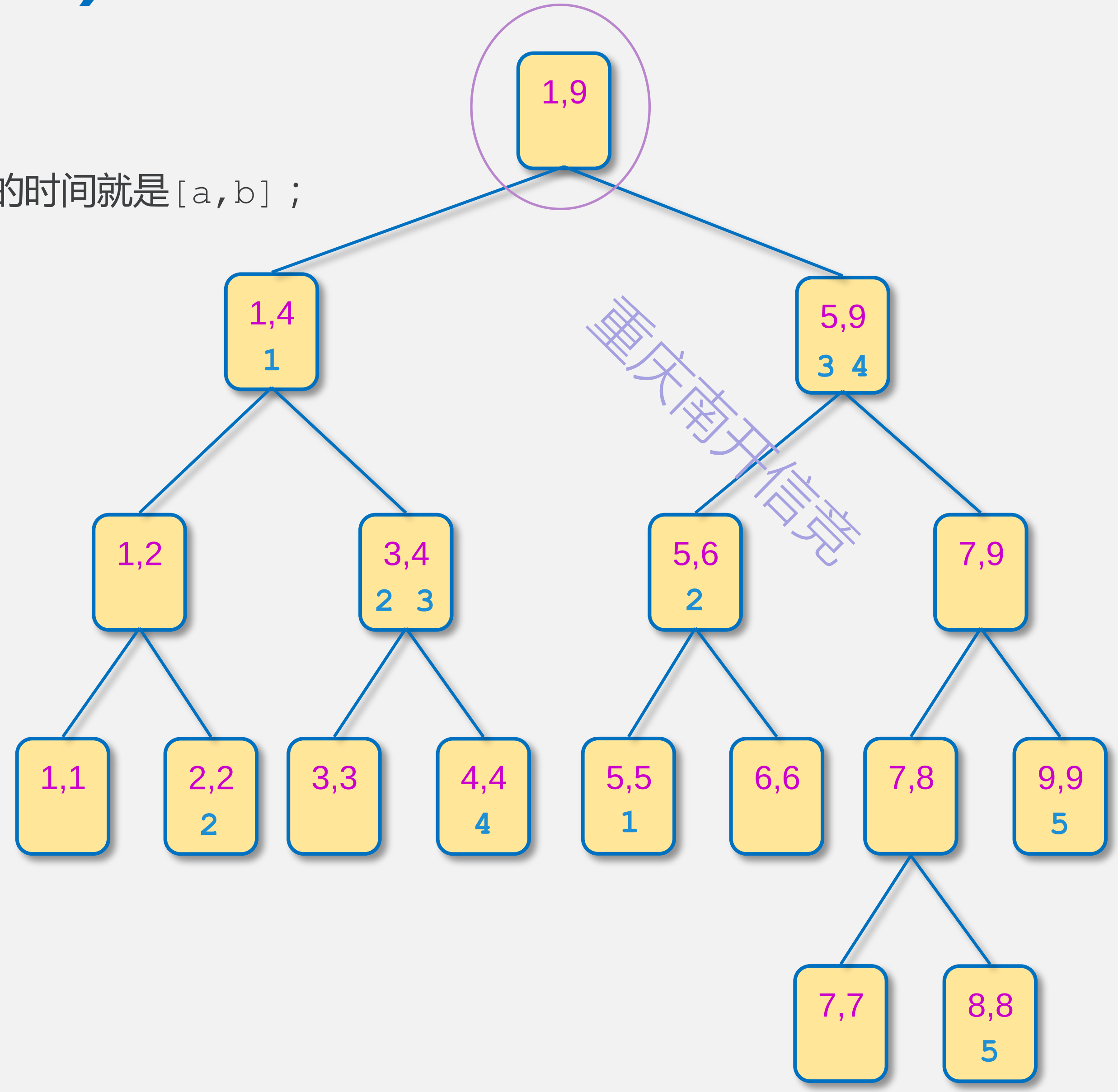
存在时间：

处理询问：

操作7：询问点1和点4
查找包含7的线段树节点 答案"N"

操作9：询问点3和点4
查找包含9的线段树节点 答案"Y"

并查集



引例：动态图连通性(NKOJ8444)

解题分析：

每条边都存在于在一定时间范围内。

在第a号操作添加了x号边，第b号操作删除了x号边，那么x号边存在的时间就是[a,b]；

在第c号操作添加了y号边，该边一直没被删除，那么y号边存在的时间就是[c,m]；

一条边删除后又被加入，算两条边。

把操作顺序看作时间，用线段树来维护时间区间。线段树的每个节点需用vector存储该点时间范围内存在的边；

先把每条边都按存在时间添加到线段树中，即如果x号边的存在时间覆盖了线段树节点i表示的时间区间，将x号边添加到i号节点的vector中

对于第i个询问，询问点x,y:

查询线段树中，所有包含时间i的节点，将每个节点中的边所连接的点都加入并查集中。总共遍历logm个点。

查询到叶子节点[i,i]后，如果x,y同在并查集里，说明连通，否则不连通。

从叶子[i,i]回溯到根的图中，还原并查集，以备后面的查询操作之用。所以需要使用**可撤消并查集。**

建好树后，遍历到每个叶子。时间复杂度O(m*logm*logn)

线段树分治

对于一类有插入、删除（撤销插入）和整体查询操作的题目，可以考虑按时间分治，就是对于每一个插入操作处理出它存在的有效时间区间，那么就不用处理删除操作了。再将这些插入操作存在区间建立一棵时间线段树，每个节点是一个vector，记录该节点时间范围内存在的插入操作。

然后从线段树根dfs到叶子经过的点上所有点vector的并就是在这个点时会对其产生影响的所有操作了。把路径上的相关信息存入一数据结构，回溯时撤消。

上面操作过程称为“线段树分治”。

CDQ分治也是对时间分治，但CDQ不支持撤销。线段树分治就是有撤销操作的时间分治。

例1：二分图(NKOJ4921)

例1：二分图(NKOJ4921)

问题描述：

n个节点的无向图。在T秒时间内一些边会出现后消失。询问每一秒这个图是否是二分图。

输入格式：

第一行是三个整数n,m,T。

第2行到第m+1行，每行4个整数u,v,start,end。

第i+1行的四个整数表示第i条边连接u,v两个点，这条边在start时刻出现，在第end时刻消失。

输出格式：

T行，在第i行中，如果第i秒时这个图是二分图，那么输出“Yes”，否则输出“No”

数据规模：n<=100000，m<=200000，T<=100000，1<=u,v<=n， 0<=start<=end<=T

时间限制：1s

例1：二分图(NKOJ4921)

问题：

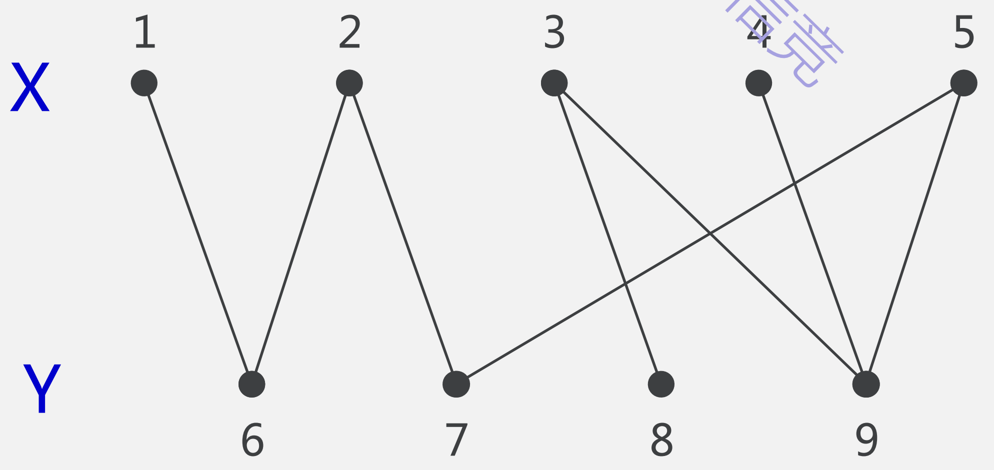
1.什么是二分图？

2.怎样判定二分图？

扩展域并查集

带权并查集

若G是一个**无向图**。G的顶点分成X和Y两部分，G中每条**边**的两个顶点**一定是一个属于X另一个属于Y**。图G称为**二分图**。



例2：数学计算(NKOJ4811 TJOI2018)

例2：数学计算(NKOJ4811 TJOI2018)

问题描述：

有一个数x，初始值为1． 有Q次操作，操作有两种类型:

"1 m"　　进行操作：x = x * m ，然后输出 x%mod;

"2 pos" 进行操作：x = x / 第pos次操作所乘的数，然后输出x%mod

（保证第pos次操作一定为类型1，对于每一个类型1的操作至多会被除一次）

Q≤ 100000, mod ≤ 1000000000

例2：数学计算(NKOJ4811 TJOI2018)

问题描述：

有一个数x，初始值为1. 有Q次操作，操作有两种类型:

"1 m"　 进行操作：x = x * m ，然后输出 x%mod;

"2 pos" 进行操作：x = x / 第pos次操作所乘的数，然后输出x%mod

暴力做法：

对于2号操作，直接计算：找到要除的数，然后除掉

这样就需要最后取mod，或求很多次逆元，不取模会爆longlong

例2：数学计算(NKOJ4811 TJOI2018)

问题描述：

有一个数x，初始值为1. 有Q次操作，操作有两种类型：

"1 m"　进行操作：x = x * m ，然后输出 x%mod;

"2 pos" 进行操作：x = x / 第pos次操作所乘的数，然后输出x%mod

线段树分治：

1.把操作顺序看作时间，用线段树维护操作时间。根节点维护操作区间[1,Q]；

2.线段树的k号节点表示区间[a,b]，维护一个Sum[k]表示第a到第b号操作数的乘积。初始时叶子的Sum值都为1；

3.若i号操作为"1 m"，将表示区间[i,i]的叶节点的Sum值设置为m

回溯时，更新所有包含i的节点的Sum值=Sum[左儿子]*Sum[右儿子]%mod

4.若j号操作为"2 pos"，将表示区间[j,j]的叶节点的Sum值设置为1(表示将原来的数值除掉了)

回溯时，更新所有包含j的节点的Sum值=Sum[左儿子]*Sum[右儿子]%mod

5.对于k号操作，更新完线段树后，根节点的Sum值%mod，即为k号操作后输出的结果

避免了求逆操作，单点修改O(logQ)，查询根节点O(1)，时间复杂度O(Q*logQ)

例3：花团 (NKOJ8446)

例3：花团 (NKOJ8446)

问题描述：

一个物品集合S初始为空，按时间递增顺序依次给出q次操作，操作如下：

1 v w e 表示在S中加入一个体积为v价值为w的物品，第e次操作结束之后移除该物品。

2 v 表示询问。你需要回答：

当前S是否存在一个子集使得子集中物品体积和为v。

当前S的所有物品体积和为v的子集中，价值和最大是多少（空集的价值和为0）。

q≤15000,maxv≤15000，

时限3s,强制在线。

例3：花团 (NKOJ8446)

问题描述：

一个物品集合S初始为空，按时间递增顺序依次给出Q次操作，操作如下：

1 v w e 表示在S中加入一个体积为v价值为w的物品，第e次操作结束之后移除该物品。

2 v 表示询问。你需要回答：

当前S是否存在一个子集使得子集中物品体积和为v。当前S的所有物品体积和为v的子集中，价值和最大是多少（空集的价值和为0）。

Q≤15000,maxv≤15000，时限3s,强制在线。

解题分析：

1.本题已经事先给出了每个物品的存活时间，所以是假强制在线；

2.线段树维护操作时间，点i维护的区间[Li,Ri]表示第Li到第Ri号操作。
  每个节点维护一个物品vector,就该区间内存在的物品们；

3.对于第j号询问，如果已知当前存活的物品，就可用背包DP算出结果。
  直接查询线段树中，包含j号操作的节点，一直查询到叶子，所经过路径上的节点们存储的物品，都是j号操作可用的物品；

时间复杂度：O(Q*v*logQ)

例4：Addition on Segments (CF981E)

例4：Addition on Segments (NKOJXXXX CF981E)

问题描述：

有一个长度为n的整数数列，初始时，数列元素全为0。
有q次操作，每次操作形如“x y z”，表示把数列[x,y]区间内每个数都增加z。
有n次询问，第k次询问数字k，**问能否在q次操作中选出若干操作来执行，使得数列元素的最大值恰好为k**？

1<=n,q<=10000
1<=z,k<=n

例4：Addition on Segments (NKOJXXXX CF981E)

问题描述：有一个长度为n的数列，初始时数列元素全为0。有q次操作，每次操作形如“x y z”，表示把数列[x,y]区间内每个数都增加z。有n次询问，第k次询问数字k，**问能否在q次操作中选出若干操作来执行，使得数列元素的最大值恰好为k**？

动态规划1：

f[i][j]表示前i个操作中选若干，能否使得位置j的值为k（能为1，否则为0）

f[i][j]=f[i-1][j-z[i]]　　条件：x[i]<=j<=y[i]

时间复杂度$O(n^2q)$

例4：Addition on Segments (NKOJXXXX CF981E)

问题描述：有一个长度为n的数列，初始时数列元素全为0。有q次操作，每次操作形如“x y z”，表示把数列[x,y]区间内每个数都增加z。有n次询问，第k次询问数字k，**问能否在q次操作中选出若干操作来执行，使得数列元素的最大值恰好为k**？

动态规划2：考虑特殊情况**所有操作区间都是[1,n]**

设前3次操作的增加值分别为**1 1 2**，那么可以得到的数值为**0，1，2，3，4**

我们用二进制来表示得到的数值**s=**$(000000011111)_2$

设第4次操作的增加值为7:

1.把s往左移动7位：s1=s<<7,此时s1=$(111110000000)_2$

2.s=s|s1=$(111110011111)_2$

即在原有0，1，2，3，4的基础上，每个数再加7，得到的数值有**0，1，2，3，4，7，8，9，10，11**

f[i]表示前i个操作能得到的数字集合

f[i+1]=f[i]|(f[i]<<z[i+1])

问题1：本题数值范围0到10000，要记录10^4位的二进制，普通状态压缩行不通，怎么办？

用bitset:

`bitset<10005>f[10005];`

问题2：上式前提是操作的x=1,y=n,每个操作都可以作用在当前f[]上。任意区间的操作”x y z”怎么处理？

对于操作x y z ，把它看作只在时间[x,y]内有效，于是就转换成了线段树分治。

例4：Addition on Segments (NKOJXXXX CF981E)

知识回顾：bitset

bitset类似数组，它每一个元素只能是0或1，每个元素占1bit空间。相当于二进制数组，各种位运算操作可用。

```
bitset<4> bs1;          //无参构造，长度为4，默认每一位为0
bs1=12;

bitset<8> bs2(12);      //长度为8，二进制保存，前面用0补充，等效于bs2=12;

string s = "100101";
bitset<10> bs3(s);      //长度为10，高位用0自动填充。不可表示为bs3=s;

char s2[] = "10101";
bitset<13> bs4(s2);      //长度为13，高位用0自动填充。不可表示为bs4=s2;

cout << bs1 << endl;     //输出“1100”
cout << bs2 << endl;     //输出“00001100”
cout << bs3 << endl;     //输出“0000100101”
cout << bs4 << endl;     //输出“0000000010101”
```

例4：Addition on Segments (NKOJXXXX CF981E)

知识回顾：bitset

bitset类似数组，它每一个元素只能是0或1，每个元素占1bit空间。相当于二进制数组，各种位运算操作可用。

```
bitset<2> bs1(12);    //12的二进制为1100（长度为4），但bs1的size=2，只取低位部分，即"00"

string s = "100101";
bitset<4> bs2(s);     //s的长度为6，而bitset的size=4，只取高位部分，即1001

char s2[] = "11101";
bitset<4> bs3(s2);    //与bs2同理，只取高位部分，即1110

cout << bs1 << endl;    //输出00
cout << bs2 << endl;    //输出1001
cout << bs3 << endl;    //输出1110
```

例4：Addition on Segments (NKOJXXXX CF981E)

知识回顾：bitset

bitset类似数组，它每一个元素只能是0或1，每个元素占1bit空间。相当于二进制数组，各种位运算操作可用。

```
bitset支持所有位运算操作
    bitset<4> a (string("1001"));
    bitset<4> b (string("0011"));

    cout << (a^=b) << endl;          // 1010 (a对b按位异或后赋值给a)
    cout << (a&=b) << endl;          // 0010 (按位与后赋值给a)
    cout << (a|=b) << endl;          // 0011 (按位或后赋值给a)

    cout << (a<<=2) << endl;          // 1100 (左移2位，低位补0，有自身赋值)
    cout << (a>>=1) << endl;          // 0110 (右移1位，高位补0，有自身赋值)

    cout << (~b) << endl;             // 1100 (按位取反)
    cout << (b<<1) << endl;           // 0110 (左移，不赋值)
    cout << (b>>1) << endl;           // 0001 (右移，不赋值)

    cout << (a==b) << endl;           // false (0110==0011为false)
    cout << (a!=b) << endl;           // true  (0110!=0011为true)

    cout << (a&b) << endl;            // 0010 (按位与，不赋值)
    cout << (a|b) << endl;            // 0111 (按位或，不赋值)
    cout << (a^b) << endl;            // 0101 (按位异或，不赋值)
```

例4：Addition on Segments (NKOJXXXX CF981E)

知识回顾：bitset

bitset类似数组，它每一个元素只能是0或1，每个元素占1bit空间。相当于二进制数组，各种位运算操作可用。

可以通过 下标[ ] 访问元素（类似数组），最低位下标为0，如下：

```
bitset<4> bs("1011");
bs[3]=0;

cout << bs[0] << endl;    //1
cout << bs[1] << endl;    //1
cout << bs[2] << endl;    //0
cout << bs[3] << endl;    //0
```

例4：Addition on Segments (NKOJXXXX CF981E)

知识回顾：bitset

bitset类似数组，它每一个元素只能是0或1，每个元素占1bit空间。相当于二进制数组，各种位运算操作可用。

bitset还支持一些有用的函数，比如：

```
bitset<8> bs ("10011011");

cout << bs.count() << endl;     //输出5, count函数用来求bitset中1的位数，bs中共有5个1
cout << bs.size() << endl;      //输出8, size函数用来求bitset的大小，一共有8位

cout << bs.test(0) << endl;     //输出true, test函数用来查下标处的元素是0还是1，并返回false或
                                 true，此处bs[0]为1，返回true

cout << bs.test(2) << endl;     //输出false,同理，bs[2]为0，返回false

cout << bs.any() << endl;       //输出true, any函数检查bitset中是否有1
cout << bs.none() << endl;      //输出false, none函数检查bitset中是否没有1
cout << bs.all() << endl;       //输出false, all函数检查bitset中是全部为1
```

例4：Addition on Segments (NKOJXXXX CF981E)

知识回顾：bitset

bitset类似数组，它每一个元素只能是0或1，每个元素占1bit空间。相当于二进制数组，各种位运算操作可用。

bitset还支持一些有用的函数，比如：

```
bitset<8> bs ("10011011");

cout << bs.flip(2) << endl;    //10011111  , flip函数传参数时，用于将参数位取反，本行代码将bs下
                                 标2处"反转"，即0变1，1变0
cout << bs.flip() << endl;     //01100000  , flip函数不指定参数时，将bitset每一位全部取反

cout << bs.set() << endl;       //11111111  , set函数不指定参数时，将bitset的每一位全部置为1
cout << bs.set(3,0) << endl;   //11110111  , set函数指定两位参数时，将第一参数位的元素置为第二参
                                  数的值，本行对bs的操作相当于bs[3]=0
cout << bs.set(3) << endl;     //11111111  , set函数只有一个参数时，将参数下标处置为1

cout << bs.reset(4) << endl;   //11101111  , reset函数传一个参数时将参数下标处置为0
cout << bs.reset() << endl;    //00000000  , reset函数不传参数时将bitset的每一位全部置为0
```

例4：Addition on Segments (NKOJXXXX CF981E)

知识回顾：bitset

bitset类似数组，它每一个元素只能是0或1，每个元素占1bit空间。相当于二进制数组，各种位运算操作可用。

一些类型转换的函数：

```
bitset<8> bs ("10011011");

string s = bs.to_string();                  //将bitset转换成string类型
unsigned long a = bs.to_ulong();            //将bitset转换成unsigned long类型
unsigned long long b = bs.to_ullong();      //将bitset转换成unsigned long long类型

cout << s << endl;    //10011011
cout << a << endl;    //155
cout << b << endl;    //155
```

例4：Addition on Segments (NKOJXXXX CF981E)

问题描述：有一个长度为n的数列，初始时数列元素全为0。有q次操作，每次操作形如“x y z”，表示把数列[x,y]区间内每个数都增加z。
有n次询问，第k次询问数字k，**问能否在q次操作中选出若干操作来执行，使得数列元素的最大值恰好为k**？

动态规划2：考虑特殊情况**所有操作区间都是[1,n]**

设前3次操作的增加值分别为**1 1 2**，那么可以得到的数值为**0，1，2，3，4**

我们用二进制来表示得到的数值**s=(000000011111)**$_2$

设第4次操作的增加值为7:

1.把s往左移动7位：s1=s<<7,此时s1=(111110000000)$_2$

2.s=s|s1=**(111110011111)**$_2$

即在原有0，1，2，3，4的基础上，每个数再加7，得到的数值有**0，1，2，3，4，7，8，9，10，11**

f[i]表示前i个操作能得到的数字集合

f[i+1]=f[i]|(f[i]<<z[i+1])

问题1：本题数值范围0到10000，要记录10^4位的二进制，普通状态压缩行不通，怎么办？

用bitset:

bitset<10005>f[10005];

问题2：上式前提是操作的x=1,y=n,每个操作都可以作用在当前f[]上。任意区间的操作”x y z”怎么处理？
对于操作x y z ，把它看作只在时间[x,y]内有效，于是就转换成了线段树分治。

例4：Addition on Segments (NKOJXXXX CF981E)

```
//main函数

bitset<10005>Ans,Tmp;
vector<int>G[80000];

cin>>n>>q;
for(int i=1;i<=q;i++)
{
      cin>>x>>y>>z;
      Modify(x,y,z,1,n,1);
}
Tmp[0]=1;       //bitset作为参数传递
Query(1,n,1,Tmp);
for(int i=1;i<=n;i++)
     if(ans[i])cout<<i<<endl;
```

```
void Modify(int x,int y,int z,int l,int r,int p)
{
     if(l>=x&&r<=y)
     {
          G[p].push_back(z);   //记录档期区间内可用的数字
          return;
     }
     int mid=(l+r)>>1;
     if(x<=mid)Modify(x,y,z,l,mid,p<<1);
     if(y>mid)Modify(x,y,z,mid+1,r,p<<1|1);
}
```

```
void Query(int l,int r,int p,bitset<10005>t)
{    //bitset数组t记录上一层能得到的所有数值
     bitset<10005>bs=t;   //bs记录当前层能得到的所有数值
     for(int i=0;i<G[p].size();i++)
          bs|=(bs<<G[p][i]);   //bs继承了上一层t的状态
     int mid=(l+r)>>1;
     if(l==r)ans|=t;   //到了叶子，从根到叶子路径上的所有数字都可已被讨论
     else Query(l,mid,p<<1,bs),Query(mid+1,r,p<<1|1,bs);
}
```

奋斗吧少年

习题：CF1140F CF603E Pastoral Oddities
nkoj5474,4346,8445