ACM Template

Macmillan School of Magic

Macmillan Expedition

Gene_Liu

July 15, 2019

Contents

0 常规操作

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <algorithm>
#define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
#define rep(i,a,b) for(int i = a; i <= b; i++)
#define LOG1(x1,x2) cout << x1 << ": " << x2 << endl;
#define LOG2(x1,x2,y1,y2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << endl;
#define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << " , " << z1 << ": " << z2 << endl;
typedef long long ll;
typedef double db;
const int N = 1e5+100;
const int M = 1e5+100;
const db EPS = 1e-9;
using namespace std;

/*快速读入、输出*/
inline void read(int &n)
{
  char ch=' ';int q=0,w=1;
  for(;(ch!='-')&&((ch<'0')||(ch>'9'));ch=getchar());
  if(ch=='-')w=-1,ch=getchar();
  for(;ch>='0'&&ch<='9';ch=getchar())q=q*10+ch-48;
  n=q*w;
}

inline void write(int x)
{
  static const int maxlen = 100;
  static char s[maxlen];
    if(x < 0) {putchar('-');x=-x;}
  if(!x){putchar('0'); return;}
  int len = 0; for(;x;x/=10) s[len++] = x % 10+'0';
  for(int i = len-1; i >= 0; --i) putchar(s[i]);
}

int n; read(n); wirte(n);
/*------------------------------------------------------------*/


/*离散化*/
sort(base+1,base+1+n);

int tot = unique(base+1,base+1+n)-base-1;

int find(ll x){
    return lower_bound(base+1,base+1+tot,x)-base;
}
/*------------------------------------------------------------*/


/*生成随机数*/
#include <ctime>
srand(time(0));
int x = rand()%((int)1000)+1; //1~1000范围
```

```
56 /*-------------------------------------------------------------*/
57
58
59 /*浮点数取模*/
60 void FLoat_Mod(long double a, long double b){
61     printf("%.9Lf\n",a-b*(long long)(a/b));
62 }
63 /*-------------------------------------------------------------*/
64
65 /* 相关于char[]的输入输出 */
66   char s[105];
67   scanf("%s", s); // 读入一个字符串，且首指针放到s+0
68   scanf("%s%s%s", s+1, s+21, s+41); // 读入三个字符串，且首指针分贝放到s+1、s+21、s+41，注意本
       样例若前两个字符串长度超过19，就会在内存有重叠而造成使用出错
69   scanf("\n%[^\n]%*c",s); // 读入一行(第一种形式)，首指针为s+0。也可以换为"%[^\n]%*c"，%*c表示
       忽略后一个字符
70   scanf("%[^\n]", s); getchar(); // 读入一行(第二种形式)，首指针为s+0。getchar()表示把\n取掉
71   printf("%s %s", s+0, s+1); // 分别表示从第0、1位开始输出，直到遇到'\0'，所以想截断输出一个字符
       串还可以将字符串的某个位置置为'\0'即可输出前半部分
72 /*-------------------------------------------------------------*/
73
74
75 /* 相关于string的输入输出 */
76   string s;s.resize(100); // 想用scanf读string必须预先设置大小
77   scanf("%s", &s[0]); // 读入到string
78   scanf("\n%[^\n]%*c", &s[0]); // 读入1行到string
79   char tmp[100];
80   scanf("%s", tmp+1); s=tmp+1; // 先读入到char[]再令string=char[]
81   printf("%s", s.c_str()); // 输出string
82 /*-------------------------------------------------------------*/
83
84
85 /*map中存放string*/
86   mp[s.c_str()] //最好传入string形式，避免发生错误
87 /*-------------------------------------------------------------*/
88
89
90 /*string函数*/
91   /* 数字转字符串 */
92   strings s = to_string(2323.232);
93   /* 取长度 */
94   int sz = s.size();
95   /* 取下标字符 */
96   char c = s[2];
97   /* 截取字符串 */
98   string ss = s.substr(2, 3);  // 从第2个字符开始截取长度为3的串
99   /* 查找串 */
100  size_t found = s.find("23", 0); // 从位置0开始找到第1个串"23"，找到返回下标，否则found=
       string::npos
101  size_t found = s.rfind("23", s.size()-1); // 从右侧开始找到第1个串"23"，至多找到至pos
102  size_t可以强转为int, int pos = (int)s.find("23",0);
103  int pos = (int)s.find(':');  //string也可以直接查找字符
104  /* 拼接 */
105  s += "aaaa";
106 /*-------------------------------------------------------------*/
107
108
109 /*bitset函数*/
110   /* 声明 */
```

```
111   bitset<16> a;           // 空，全零 0000000000000000
112   bitset<16> b(0x3fff);   // 整数参数 0011111111111111
113   bitset<16> c("00101");  // 字符串   0000000000000101
114
115   /* 赋值 */
116   a[0] = 1;    // 低位第0位设为1
117   a.set(0, 1); // 低位第0位设为1
118   a.set(0);    // 低位第0位设为1
119   a.set(0, 0); // 低位第0位设为0
120   a.set();     // 设为全1
121   a.reset();   // 设为全0，该函数同样有两个参数，参数1为pos，参数2位value(为空时是0)
122   a.flip(1);   // 翻转第1位
123   a.flip();    // 翻转整个01串
124   a<<=1;       // a左移1位
125
126   /* 遍历 */
127   cout << a << endl;                      // cout输出整个01串
128   printf("%s\n", a.to_string().c_str()); // printf输出整个01串，较为麻烦一些
129   cout << a[0] << endl;                   // cout输出低位第0位
130   printf("%d\n", a[0]==1);                // printf输出低位第0位
131
132   /* 判位、换型 */
133   a.count();              //统计1的个数
134   a = a1 & a2;            //按位与
135   a = a1 | a2;            //按位或
136   a = a1 ^ a2;            //按位异或
137   a = ~a1;               //按位补
138   a = a1 << 3;            //移位
139   a.text(0);                          // 判断第0位是否为1
140   a.any();                            // 判断是否至少1位为1
141   s.none();                           // 判断是否全0
142   int one = a.count;                  // 获得1的个数
143   string s = a.to_string();           // 转换为字符串
144   unsigned long = a.to_ulong();       // 转换为无符号整形
145   unsigned long long = a.to_ullong(); // 转换为无符号整形
146 /*-------------------------------------------------------------*/
147
148
149 /*vector*/
150   /* 声明 */
151   vector<int> v;
152   /* 排序 */
153   sort(v.begin(),v.end());
154   /* 遍历方式一（推荐）*/
155   for(auto &tp : v){}
156   /* 遍历方式二（不推荐）*/
157   if(v.size()){ //一定要确保v不为空，否则会RE
158     rep(i,0,v.size()-1){}
159   }
160 /*-------------------------------------------------------------*/
161
162
163 /*set函数*/
164   /* 声明 */
165   set<int> s;
166   /* 赋值 */
167   s.insert(1);
168   s.insert(2);
169   /* 遍历 */
```

```
  if(s.find(1) != s.end()) printf("had");
  for(set<int>::iterator it=s.begin();it!=s.end();it++) printf("%d", *it); // 全部遍历
  for(auto &x : s) printf("%d\n", x);
  int sz = s.size(); // 大小
  /* 清空 */
  s.clear();
  s.erase(x); //x为s中存的内容
  *(s.begin()); //取出set中首位元素
  multiset<int>::iterator it1 = st.begin(), it2 = st.end();
  it2--; //取出set/multiset首尾元素
  set<pair<int,int> >::iterator it = st1.lower_bound(make_pair(-pos,0)); //it是指针
  lower_bound: 找到第一个大于等于这个数的值，若要找第一个小于的可以考虑往set里丢负值
  /* multiset删除 */
  st.erase(st.find(x)) //只删一个x
  st.erase(x) //删除所有x
/*------------------------------------------------------------------*/


/*regex分割*/
  regex re("[\\.,!\\? ]"); //全局变量，.与?需要转义，此处分割的内容有'.'、','、'!'、'?'、' '
  //分割之后，最后的一个字符串可能会含有大量空格
  /* 切割单词后放入一个vector */
  vector<string> ans{
      sregex_token_iterator(yy.begin()+pos+1, yy.end(), re, -1),
      sregex_token_iterator()
  };
  for(auto &it: ans) printf("%s\n",it.c_str());
/*------------------------------------------------------------------*/


/*二进制中01操作*/
template<class T> int getbit(T s, int i) { return (s >> i) & 1; }
template<class T> T onbit(T s, int i) { return s | (T(1) << i); }
template<class T> T offbit(T s, int i) { return s & (~(T(1) << i)); }
template<class T> int cntbit(T s) { return __builtin_popcount(s);}
/*------------------------------------------------------------------*/
```

1 数学

1.1 快速幂与快速乘

1.1.1 快速幂

```
//快速幂
#include <cstdio>
#include <iostream>
typedef long long LL;
using namespace std;

LL poww(LL a,LL b)
{
    LL base = a,ans = 1;
    while(b!=0)
    {
        if(b&1)
            ans *= base;
        base *= base;
        b >>= 1;
    }
    return ans;
}

LL poww(LL a,LL b,LL p)
{
    LL base = a%p,ans = 1;
    while(b!=0)
    {
        if(b&1)
            ans = (ans*base)%p;
        base = (base*base)%p;
        b >>= 1;
    }
    return (ans%p);
}

int main()
{
    LL h = poww(11,11);
    cout<<h<<endl;
    return 0;
}
```

1.1.2 矩阵快速幂

```
#include <cstdio>
#include <iostream>
#include <cstring>
using namespace std;
const int mod = 9973;
typedef long long ll;

int n,m,t;
//i、k、j只能作为临时变量
struct Matrix{
    ll a[35][35];
    Matrix(){memset(a,0,sizeof(a));}
```

```
13
14      Matrix operator * (const Matrix y)
15      {
16          Matrix ans;
17          for(int i = 1;i <= n;i++)   //行
18              for(int j = 1;j <= n;j++)  //列
19                  for(int k = 1;k <= n;k++)
20                      ans.a[i][j] = (ans.a[i][j]+a[i][k]*y.a[k][j])%mod;
21          return ans;
22      }
23
24      Matrix operator + (const Matrix y)
25      {
26          Matrix ans;
27          for(int i = 1;i <= n;i++)
28              for(int j = 1;j <= n;j++)
29                  ans.a[i][j] = (a[i][j]+y.a[i][j])%mod;
30          return ans;
31      }
32  };
33
34  Matrix q_pow(Matrix x,int k)
35  {
36      Matrix ans;
37      for(int i = 1;i <= n;i++)   ans.a[i][i] = 1;
38      while(k)
39      {
40          if(k&1)
41              ans = ans*x;
42          x = x*x;
43          k = k>>1;
44      }
45      return ans;
46  }
47
48  int main()
49  {
50      cin>>t;
51      while(t--)
52      {
53          scanf("%d%d",&n,&m);
54          Matrix p;
55          for(int i = 1;i <= n;i++)
56              for(int j = 1;j <= n;j++)
57                  scanf("%lld",&p.a[i][j]);
58          Matrix ans = q_pow(p,m);
59          long long ans1 = 0;
60          for(int i = 1;i <= n;i++)
61              ans1 = (ans.a[i][i]+ans1)%mod;
62          printf("%lld\n",ans1);
63      }
64      return 0;
65  }
```

1.1.3 快速乘取膜

方法一:
a b p 范围都在 1e18

求 a*b 的结果
为了不让结果爆 ll，则将 a 进行二进制分解，然后递推 *b 即可

方法二:
用 long double 存数, long double 的有效数字有 18-19 位

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <algorithm>
#define rep(i,a,b) for(int i = a; i <= b; i++)
using namespace std;
typedef long long ll;

ll mul(ll a,ll b,ll p)
{
    a %= p, b %= p;
    long long c = (long double)a*b/p;
    long long ans = a*b-c*p;
    //此处会溢出，但是由于二者相差的部分不会太大，所以前面的数字都是一样的
    //因此溢出不但不影响答案，还是符合我们的要求的
    if(ans < 0) ans+=p;
    else if(ans >= p) ans-=p;
    return ans;
}

int main()
{
    ll a,b,p;
    while(~scanf("%lld%lld%lld",&a,&b,&p))
    {
        printf("%lld\n",mul(a,b,p));
    }
    return 0;
}
```

1.2 筛

1.2.1 埃氏筛

```
#include <cstdio>
#include <iostream>
#include <algorithm>
#include <cstring>
#define rep(i,a,b) for(int i = a;i <= b;i++)
using namespace std;
const int N = 1.5*1e7+1000;

int b[N],tot,prime[N];

int main()
{
    tot = 0; //素数个数

    //埃氏筛
    for(int i = 2; i <= 1e5; i++)
    {
        if(b[i] == 0)
```

```
            prime[++tot] = i; //哪几个是素数
        else continue;
        int base = i;
        while(base <= 1e5){
            base += i;
            b[base] = 1; //这个数是不是素数, 为1则非素数
        }
    }
    return 0;
}
```

1.2.2 线性筛

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <algorithm>
#define rep(i,a,b) for(int i = a; i <= b; i++)
using namespace std;
const int N = 1e7+1000;

int v[N],prime[N],tot;  //v[i]: i的最小质因子

void primes(int x)  //线性筛
{
    memset(v,0,sizeof v);
    tot = 0;
    rep(i,2,x){
        if(!v[i]) v[i] = i, prime[++tot] = i;
        rep(j,1,tot)
            if(prime[j] > v[i] || prime[j] > x/i) break;
            else v[i*prime[j]] = prime[j];
    }
}

int main()
{
    int n = 1e5;
    primes(n);
    rep(i,1,tot) printf("%d\n",prime[i]);
    return 0;
}
```

1.2.3 杜教筛

给定一个 n ($n \leq 2^31-1$), 求 ans_1 和 ans_2

$ans_1 = \sum_{i=1}^{n} \varphi(i)$

$ans_2 = \sum_{i=1}^{n} \mu(i)$

```
#include<bits/stdc++.h>
#include<tr1/unordered_map>
#define N 6000010
using namespace std;
template<typename T>inline void read(T &x)
{
```

```
    x=0;
    static int p;p=1;
    static char c;c=getchar();
    while(!isdigit(c)){if(c=='-')p=-1;c=getchar();}
    while(isdigit(c)) {x=(x<<1)+(x<<3)+(c-48);c=getchar();}
    x*=p;
}
bool vis[N];
int mu[N],sum1[N],phi[N];
long long sum2[N];
int cnt,prim[N];
tr1::unordered_map<long long,long long>w1;
tr1::unordered_map<int,int>w;
void get(int maxn)
{
    phi[1]=mu[1]=1;
    for(int i=2;i<=maxn;i++)
    {
        if(!vis[i])
        {
            prim[++cnt]=i;
            mu[i]=-1;phi[i]=i-1;
        }
        for(int j=1;j<=cnt&&prim[j]*i<=maxn;j++)
        {
            vis[i*prim[j]]=1;
            if(i%prim[j]==0)
            {
                phi[i*prim[j]]=phi[i]*prim[j];
                break;
            }
            else mu[i*prim[j]]=-mu[i],phi[i*prim[j]]=phi[i]*(prim[j]-1);
        }
    }
    for(int i=1;i<=maxn;i++)sum1[i]=sum1[i-1]+mu[i],sum2[i]=sum2[i-1]+phi[i];
}
int djsmu(int x)
{
    if(x<=6000000)return sum1[x];
    if(w[x])return w[x];
    int ans=1;
    for(int l=2,r;l>=0&&l<=x;l=r+1)
    {
        r=x/(x/l);
        ans-=(r-l+1)*djsmu(x/l);
    }
    return w[x]=ans;
}
long long djsphi(long long x)
{
    if(x<=6000000)return sum2[x];
    if(w1[x])return w1[x];
    long long ans=x*(x+1)/2;
    for(long long l=2,r;l<=x;l=r+1)
    {
        r=x/(x/l);
        ans-=(r-l+1)*djsphi(x/l);
    }
    return w1[x]=ans;
```

```
}
int main()
{
    int t,n;
    read(t);
    get(6000000);
    while(t--)
    {
        read(n);
        printf("%lld %d\n",djsphi(n),djsmu(n));
    }
    return 0;
}
```

1.3 GCD 与逆元

1.3.1 GCD

```
typedef long long ll;

ll gcd(ll a,ll b)
{
    return b == 0 ? a:gcd(b,a%b);
}
```

1.3.2 Exgcd 与逆元

```
#include <cstdio>
#include <iostream>
#include <algorithm>
#include <cstring>
#define rep(i,a,b) for(int i = a;i <= b;i++)
using namespace std;
typedef long long ll;
const ll mod = 9973;

//求ax+by = gcd(a,b)的特解
ll exgcd(ll a,ll b,ll &x,ll &y)
{
    if(b == 0) {x = 1,y = 0; return a;}  //此处a为gcd(a,b)
    ll d = exgcd(b,a%b,x,y);
    ll z = x; x = y; y = z-y*(a/b);
    return d;
}
// ax+by = c的通解可以表示为 x = (c/d)x1+k*(b/a) 【d为gcd(a,b)】
//                          y = (c/d)y1-k*(a/d) 【k为正整数】

//调用mod_reverse就是求逆元，求a在%n意义下的逆元
ll mod_reverse(ll a,ll n)
{
    ll x,y;      //求a的逆元，a*ai 与 1 % mi同余，则a*ai-mi*k = 1；a与mi已知，求ai
    ll d = exgcd(a,n,x,y);
    if(d == 1) return (x%n+n)%n;  //保证逆元为正
    else return -1;
}
```

费马小定理求逆元
1. 模数 p 为素数

$a^{-1} = a^{p-2}$
2. 模数 p 不为素数
$a^{-1} = a^{\phi(p)-1}$

```

int main()
{
    ll x;
    while(~scanf("%lld",&x))
    {
        printf("%lld\n",mod_reverse(x,mod));
    }
    return 0;
}
```

1.3.3 O(n) 打表求逆元

```
#include <cstdio>
#include <cmath>
using namespace std;
typedef  long long ll;
const int N = 1e5 + 5;

int inv[N];

void inverse(int n, int p) { //O(n)求1-n所有逆元
    inv[1] = 1;
    for (int i=2; i<=n; ++i) {
        inv[i] = (ll) (p - p / i) * inv[p%i] % p;
    }
}
```

1.4 唯一分解

```
#include <cstdio>
#include <iostream>
#include <algorithm>
#include <cstring>
#define rep(i,a,b) for(int i = a;i <= b;i++)
using namespace std;
const int N = 1.5*1e7+1000;

int b[N],c[N][2]; //c[i][0] —— 表示第 i 个唯一分解数
//c[i][1] —— 表示第 i 个唯一分解数的指数
int prime[N],tot;
int n;

//唯一分解
void getFactors(ll x)
{
    ll tmp = x;
    for(ll i = 1; prime[i]*prime[i] <= tmp; i++)
    {
        if(tmp%prime[i] == 0)
        {
            c[++tot][0] = prime[i];
            while(tmp%prime[i] == 0){
```

```
                tmp /= prime[i];
                c[tot][1]++;
            }
        }
    }
    if(tmp != 1){
        c[++tot][0] = tmp;
        c[tot][1] = 1;
    }
}

int main()
{
    scanf("%d",&n);
    tot = 0;

    //埃氏筛
    for(int i = 2; i <= 1e5; i++)
    {
        if(b[i] == 0)
            prime[++tot] = i;
        else continue;
        int base = i;
        while(base <= 1e5){
            base += i;
            b[base] = 1;
        }
    }
    rep(i,1,n)
    {
        int x;
        scanf("%d",&x);
        getFactors(x);
    }
    return 0;
}
```

1.5 中国剩余定理

1.5.1 中国剩余定理

```
//中国剩余定理
#include <cstdio>
#include <iostream>
#include <algorithm>
#include <cstring>
#define rep(i,a,b) for(int i = a;i <= b;i++)
using namespace std;

//求ax+by = gcd(a,b)的特解
ll exgcd(ll a,ll b,ll &x,ll &y)
{
    if(b == 0) {x = 1,y = 0; return a;}  //此处a为gcd(a,b)
    ll d = exgcd(b,a%b,x,y);
    ll z = x; x = y; y = z-y*(a/b);
    return d;
}
// ax+by = c的通解可以表示为 x = (c/d)x1+k*(b/a) 【d为gcd(a,b)】
//                          y = (c/d)y1-k*(a/d) 【k为正整数】
```

```

//调用mod_reverse就是求逆元，求a在%n意义下的逆元
ll mod_reverse(ll a,ll n)
{
    ll x,y;      //求a的逆元，a*ai 与 1 % mi同余，则a*ai-mi*k = 1；a与mi已知，求ai
    ll d = exgcd(a,n,x,y);
    if(d == 1) return (x%n+n)%n;  //保证逆元为正
    else return -1;
}

//求ans%m1=a1,ans%m2=a2,...中的ans，其中m1,m2,m3...互质
int CRT(int a[],int m[],int cn){
    int M = 1;
    int ans = 0;
    for(int i=1; i<=cn; i++)
        M *= m[i];          //M为所有的除数相乘
    for(int i=1; i<=cn; i++){
        int Mi = M / m[i];
        int x = mod_reverse(Mi, m[i]);  //求Mi在%m[i]意义下的逆元
        ans = (ans + Mi * x * a[i]) % M; //最后求出的结果在1-M之间
        //注意此处的Mi*x*a[i]如果有爆long long的风险，则需要调用快速乘来进行乘法运算
    }
    if(ans < 0) ans += M;  //ans是特解，ans的通解 = ans+K*M (K为整数)
    return ans;
}
```

1.5.2 中国剩余定理拓展

```
//拓展中国剩余定理 —— m1,m2,m3之间不互质
#include<iostream>
#include<cstdio>
#include<climits>
#include<cstring>
#include<algorithm>
using namespace std;
#define LL long long
const int maxn=1e5+5;
int n;
LL exgcd(LL a,LL b,LL &x,LL &y){
    if(!b){x=1,y=0;return a;}
    LL re=exgcd(b,a%b,x,y),tmp=x;
    x=y,y=tmp-(a/b)*y;
    return re;
}
LL m[maxn],a[maxn];
LL work(){
    LL M=m[1],A=a[1],t,d,x,y;int i;
    for(i=2;i<=n;i++){
        d=exgcd(M,m[i],x,y);//解方程
        if((a[i]-A)%d)return -1;//无解
        x*=(a[i]-A)/d,t=m[i]/d,x=(x%t+t)%t;//求x
        A=M*x+A,M=M/d*m[i],A%=M;//日常膜一膜（划掉）模一模，防止爆
    }
    A=(A%M+M)%M;
    return A;
}
int main()
{
```

```
    int i,j;
    while(scanf("%d",&n)!=EOF){
        for(i=1;i<=n;i++)scanf("%lld%lld",&m[i],&a[i]);
        printf("%lld\n",work());
    }
    return 0;
}
```

1.6 求组合数

1.6.1 Lucas (模数为质数)

```
#include <cstdio>
#include <iostream>
#include <algorithm>
#include <cstring>
#define rep(i,a,b) for(int i = a;i <= b;i++)
using namespace std;
typedef long long ll;

//求ax+by = gcd(a,b)的特解
ll exgcd(ll a,ll b,ll &x,ll &y)
{
    if(b == 0) {x = 1,y = 0; return a;}  //此处a为gcd(a,b)
    ll d = exgcd(b,a%b,x,y);
    ll z = x; x = y; y = z-y*(a/b);
    return d;
}
// ax+by = c的通解可以表示为 x = (c/d)x1+k*(b/a) 【d为gcd(a,b)】
//                          y = (c/d)y1-k*(a/d) 【k为正整数】

//调用mod_reverse就是求逆元，求a在%p意义下的逆元
ll mod_reverse(ll a,ll p)
{
    ll x,y;      //求a的逆元，a*ai 与 1 % mi同余，则a*ai-mi*k = 1；a与mi已知，求ai
    ll d = exgcd(a,p,x,y);
    if(d == 1) return (x%p+p)%p;  //保证逆元为正
    else return -1;
}

ll fact(ll n, ll p){//n的阶乘 % p
    ll ret = 1;
    for (int i = 1; i <= n ; i ++) ret = ret * i % p ;
    return ret ;
}

ll comb(ll n, ll m, ll p){//C(n, m) % p，将n和m限制在1-p之内
    if (m < 0 || m > n) return 0;
    //调用逆元和求解阶乘
    return fact(n, p) * mod_reverse(fact(m, p), p) % p * mod_reverse(fact(n-m, p), p) %
     p;
}

ll Lucas(ll n, ll m, ll p)  //求组合数C(n,m)%p
{
    return m ? Lucas(n/p, m/p, p) * comb(n%p, m%p, p) % p : 1;
}


```

```
int main()
{
/*注意事项:
    C(n,m)%p用Lucas定理求解，p必须为质数，并且p < 1e5，这个限制主要是针对如果p大于1e5，那就算n和m在1-p之内，如果n和m大于1e5，也难以求解
    本模板还可以进一步进行加速，那就是预处理每个数的阶乘
*/
    ll n,m,p;
    int T;
    scanf("%d",&T);
    while(T--)
    {
        scanf("%lld%lld%lld",&n,&m,&p);
        printf("%lld\n",Lucas(n+m,m,p));
    }
    return 0;
}
```

1.6.2 递推求组合数

$C(n,m) = C(n-1,m-1) + C(n-1,m)$

```
C[1][0] = C[1][1] = 1;

for (int i = 2; i < N; i++){
    C[i][0] = 1;
    for (int j = 1; j < N; j++)
        C[i][j] = (C[i - 1][j] + C[i - 1][j - 1]);
}
```

1.7 常见数学式子

1. $F(n) = \sum_{x=1}^{n} x * 2^x$
错位相减，直接求出答案 $F(n) = (n-1) * 2^{n+1} + 2$。

2. $F(n) = \sum_{x=1}^{n} 2^x * x^2$
先构造 $f(x) = \sum_{i=1}^{n} x^i * i^2$，即直接求出 x 的通式，然后再将 x 替换成 2。
利用高数计算无穷级数的方法，先构造 $\frac{f(x)}{x}$，然后先积分再求导，得出答案 $F(n) = 2^{n+1} * (n^2 - 2 * n + 3) - 6$。

3. 常见组合数公式
$\sum_{i=1}^{n} 2^i i^2 = (n^2 - 2n + 3)2^{n+1} - 6$

$\sum_{i=1}^{n} 2^i i = (n-1)2^{n+1} + 2$

$1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(n+1)(2*n+1)}{6}$

$1^3 + 2^3 + 3^3 + \cdots + n^3 = \frac{n^2(n+1)^2}{4}$

$\sum_{i=0}^{k} C_{n+i}^{n} = C_{n+k+1}^{n+1}$

$\sum_{i=1}^{n} iC_n^i = n * 2^{n-1}$

$\sum_{i=1}^{n} i^2 C_n^i = n * (n+1) * 2^{n-2}$

$\sum_{i=1}^{n}(-1)^{i-1}\frac{C_n^i}{i}=\sum_{i=1}^{n}\frac{1}{i}$

$\sum_{i=0}^{n}(C_n^i)^2=C_{2n}^n$

```
return 0;
```

2 博弈

2.1 SG 函数

题意：有 n 堆魔法珠，第 i 堆有 a_i 颗。选择 n 堆魔法珠中数量大于 1 的任意一堆。记该堆魔法珠数量为 p，p 有 $b_1\ b_2\ b_3\ ...\ b_m$ 这 m 个小于 p 的约数。然后将这一堆魔法珠变成 m 堆，每堆各有 $b_1\ b_2\ b_3\ ...\ b_m$ 颗魔法珠。最后选择这 m 堆的任意一堆，令其消失，不可操作者输，问谁能获胜。$(1 \leq n \leq 100, 1 \leq a_i \leq 1000)$

思路：不难发现整个游戏就是由多个魔法珠堆组成的，因此 $SG[x]$ 表示某一堆魔法珠，个数为 x 时的 SG 值。求 $SG[x]$ 时，若 x 有 $b_1\ b_2\ b_3\ ...\ b_m$ 这 m 个小于 x 的约数，则可以达到的后继状态一共有 m 个，即去掉任意一堆达到的状态。而每一个状态中仍有 $m-1$ 个堆，即 $m-1$ 个子游戏，因此每一个状态的 SG 值为这 $m-1$ 个子游戏 SG 值的异或和。具体过程见代码。

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <algorithm>
#define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
#define rep(i,a,b) for(int i = a; i <= b; i++)
#define LOG1(x1,x2) cout << x1 << ": " << x2 << endl;
#define LOG2(x1,x2,y1,y2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << endl
    ;
#define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2
    << " , " << z1 << ": " << z2 << endl;
typedef long long ll;
typedef double db;
const int N = 1000+10;
const int M = 1e5+100;
const db EPS = 1e-9;
using namespace std;

int sg[N],n;

int solve(int x)
{
    if(sg[x] != -1) return sg[x];
    int vis[2010]; memset(vis,0,sizeof vis);
    int ans[2010]; //记录每一个约数
    int tp = 0, ct = 0;
    rep(i,1,x-1){
        if(x%i == 0){
            ans[++ct] = solve(i);
            tp ^= ans[ct];
        }
    }
    rep(i,1,ct) vis[tp^ans[i]] = 1; //枚举每一个子状态
    rep(i,0,2000)
        if(vis[i] == 0) return sg[x] = i;
}

int main()
{
    memset(sg,-1,sizeof sg);
    sg[1] = 0;
    while(~scanf("%d",&n)){
        int ans = 0;
        rep(i,1,n){
            int xx; scanf("%d",&xx);
```

```
44              ans ^= solve(xx);
45          }
46          if(ans == 0) printf("rainbow\n");
47          else printf("freda\n");
48      }
49      return 0;
50  }
```

3 动态规划

3.1 01 背包

题意：一个比赛有 n 个裁判，每个裁判有一个权值 ai，并且给出通过或不通过。有一个判断值 p，若所有给出通过的裁判的权值和大于等于 p，那么整体就算通过。给出第二组裁判，权值为 bi，判断值为 q，判断所有 2^n 种判断下这两组给出的结果是否完全相同，如果不相同，给出一种不相同的情况。

思路：比赛的时候完全没思路，一个劲的贪心... 赛后才知道这题其实是个 dp，dp[i][j] 表示前 i 个人，序列 a 中选的人的值到达 j 的所有情况中，序列 b 中对应人的权值之和的最大值。因此当 dp[i][j] >= q，而 j < p 时，即是错误的情况。然后再判断一遍 b 序列即可。此处还有一个地方，记错误方案时如果是用 pre 数组不断回溯的话，会 T。正确做法是对于每个 dp[j] 记一个 bitset 的状态，检查出错误后，直接输出即可。

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <bitset>
#include <algorithm>
#define rep(i,a,b) for(int i = a; i <= b; i++)
#define LOG1(x1,x2) cout << x1 << ": " << x2 << endl;
#define LOG2(x1,x2,y1,y2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << endl
    ;
typedef long long ll;
typedef double db;
const db EPS = 1e-9;
using namespace std;
const int N = 200;
const int M = 1e6+1000;

int n,a[N],b[N],dp[M],am,bm,tp[M],ans[N];
//dp[i]表示a组中，价值为i的所有组合中，b组最大价值的一组
bitset<110> base1[M],base2[M];

void solve()
{
    rep(i,1,n) ans[i] = 0;
    rep(i,1,n){
        for(int j = am-1; j >= a[i]; j--){
            if(j == a[i]){
                if(dp[j] < b[i]) dp[j] = b[i], base1[j].reset(),base1[j].set(i);
            }
            if(dp[j] < dp[j-a[i]]+b[i] && dp[j-a[i]]!=0){
                dp[j] = dp[j-a[i]]+b[i];
                base1[j] = base1[j-a[i]];
                base1[j].set(i);
            }
            if(dp[j] >= bm){
                printf("NO\n");
                rep(i,1,n) printf("%d",(int)base1[j][i]);
                printf("\n");
                return;
            }
        }
    }
    rep(i,1,n){
        for(int j = bm-1; j >= 1; j--){
            if(j == b[i]){
                if(tp[j] < a[i]) tp[j] = a[i], base2[j].reset(),base2[j].set(i);
```

```
45              }
46              if(j < b[i]) continue;
47              if(tp[j] < tp[j-b[i]]+a[i] && tp[j-b[i]]!=0){
48                  tp[j] = tp[j-b[i]]+a[i];
49                  base2[j] = base2[j-b[i]];
50                  base2[j].set(i);
51              }
52              if(tp[j] >= am){
53                  printf("NO\n");
54                  rep(i,1,n) printf("%d",(int)base2[j][i]);
55                  return;
56              }
57          }
58      }
59      printf("YES\n");
60  }
61
62  int main()
63  {
64      scanf("%d",&n);
65      scanf("%d",&am);
66      rep(i,1,n) scanf("%d",&a[i]);
67      scanf("%d",&bm);
68      rep(i,1,n) scanf("%d",&b[i]);
69      solve();
70      return 0;
71  }
```

3.2 完全背包

题意:
音符格式转换成长度最小且字典序最小的格式。一共有 7 种音节，分别是 $R1$、$R2$、$R4$、$R8$、$R16$、$R32$、$R64$，分别表示 1、$\frac{1}{2}$、$\frac{1}{4}$、$\frac{1}{8}$、$\frac{1}{16}$、$\frac{1}{32}$、$\frac{1}{64}$ 拍。对于 $R1R2$ 即为 $\frac{3}{2}$ 拍，也可以表示成 $R1.$。即可将相邻的一个音节化为 '.'，因此 $R2...$ 表示 $R2R4R16R32$。

此题给出一个音符表示格式，要求转化为长度最短且字典序最小的方式。

例如 $R1R4R16$ 转化为 $R16R1R4$ 。

思路:
比赛的时候遇到这题，错误地当成了贪心问题进行考虑，导致始终无法 AC。

现在我们来观察这个问题，这个表达式所有音节的拍数相加之和是不变的，而每一个音节单元的拍数和长度也都是确定的，因此等价于一个完全背包问题。

因此如果此题没有要求给出一个字典序最小的方案，只要求最短长度的话。$dp[i]$ 表示音符表达式拍数为 i 所需的最短长度，然后直接跑一个完全背包即可。

但是这题还要求求出长度最小时字典序最小，因此我们需要考虑状态方程。我一开始的错误做法是两维 for 循环，第一维枚举用到了第 i 个音节单元，第二维枚举当前拍数 j，然后在转移时若是小于，直接转移，若是等于，则比较当前音节单元字典序与 dp[j] 的最后一个音节单元字典序。这样会 wa，因为无法保证字典序最小。

但是如果换一种枚举方式就可以通过此题，第一维 for 循环枚举音节拍数 i，第二维枚举使用了第 j 个音节单元，然后 == 部分的转移仍比较当前 j 与 $dp[i]$ 的上一个音节单元字典序即可通过此题。原因在于求取 $dp[i]$ 的时候，已经使用了所有的音节单元，而前一种枚举方式并没有用上所有的音节单元所以导致出错。

当然此题也可以字典序倒序枚举前 i 个音节单元, 也可以通过此题。

处理完这部分之后，就是完全背包的路径输出了，由于完全背包 $dp[i][j]$ 是由 $dp[i][j-w[i]]$ 转移而来，即同维转移而来，因此不需要不需要 01 背包的二维记录路径，直接使用一维进行路径记录，即上一次由哪个音节单元转移而来即可。

总结:
此题考查了完全背包的两个内容,一是完全背包求取过程中加上了对于字典序最小的求解,二是完全背包的一维路径输出。

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <string>
#include <bitset>
#include <algorithm>
#define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
#define rep(i,a,b) for(int i = a; i <= b; i++)
#define LOG1(x1,x2) cout << x1 << ": " << x2 << endl;
#define LOG2(x1,x2,y1,y2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << endl;
#define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << " , " << z1 << ": " << z2 << endl;
typedef long long ll;
typedef double db;
const int N = 3200000+10;
const int M = 1e5+100;
const db EPS = 1e-9;
using namespace std;

char s[N];
int len,n,tot,dp[N],pre[N],ttp[N];
// int str[N][33];
// string str[N];
//v: 拍数, w: 长度
//len: 总拍数
//用最少的长度凑满总拍数
//dp[i]: 表示凑满拍数为i所需的最少长度

struct Node{
    int num,sum,ctt,v,w; //num: 编号, sum: .个数, ctt: 选了多少个, v: 拍数, w: 字符长度
    string ss;
    bool operator < (Node xx) const {
        if(num == xx.num) return sum > xx.sum;
        else return (ss < xx.ss);
    }
}ans[30];

void solve()
{
    memset(dp,0x3f,sizeof dp);
    dp[0] = 0;
    rep(j,1,len) //音符长度
        rep(i,1,tot){ //枚举所有音符单元
            if(j >= ans[i].v){
                if(dp[j] > dp[j-ans[i].v]+ans[i].w){
                    dp[j] = dp[j-ans[i].v]+ans[i].w;
                    pre[j] = i;
                }
                else if(dp[j] == dp[j-ans[i].v]+ans[i].w){
                    if(pre[j] > i){ //之前的字典序更大
```

```
                        dp[j] = dp[j-ans[i].v]+ans[i].w;
                        pre[j] = i;
                    }
                }
            }
        }
    int tp = len;
    while(tp > 0){ //完全背包一维记录路径, 直接回溯
        ans[pre[tp]].ctt++;
        tp = tp-ans[pre[tp]].v;
    }
    rep(i,1,tot)
        rep(j,1,ans[i].ctt)
            cout << ans[i].ss;
    cout << endl;
}

int main()
{
    // freopen("e.in","r",stdin);
    // freopen("e.out","w",stdout);
    scanf("%s",s+1); n = strlen(s+1);
    int num;
    //获取拍数总长度
    rep(i,1,n){
        num = 0;
        if(s[i] == 'R'){
            i++;
            while(s[i] != 'R' && i <= n && s[i] != '.')
            {
                num = num*10+s[i]-'0';
                i++;
            }
        }
        int tt = 64/num;
        // LOG2("i",i,"tt",tt);
        len += tt;
        if(s[i] == '.'){
            while(s[i] != 'R' && i <= n){
                tt /= 2;
                len += tt;
                i++;
            }
        }
        i--;
    }
    //构造所有音符单元
    rep(i,1,7){
        rep(j,0,i-1){
            int tp = 1<<(i-1);
            ++tot;
            ans[tot].v += tp;
            rep(k,1,j){
                tp /= 2, ans[tot].v += tp;
            }
            ans[tot].w = j+1;
            if(i <= 3) ans[tot].w += 2;
            else ans[tot].w += 1;
            ans[tot].sum = j;
```

```
            ans[tot].num = 1<<(7-i);
        }
    }
    rep(i,1,tot){
        ans[i].ss = "\0";
        string tpp;
        tpp += 'R';
        tpp += to_string(ans[i].num);
        rep(j,1,ans[i].sum) tpp += ".";
        ans[i].ss = tpp;
    }
    sort(ans+1,ans+1+tot);
    //按字典序构造音符单元 —— 非关键操作，也可直接打表输入
    Node tp = ans[7];
    rep(i,8,10) ans[i-1] = ans[i];
    ans[10] = tp;
    // rep(i,1,tot)
        // LOG3("i",i,"ss",ans[i].ss,"v",ans[i].v);
    solve();
    return 0;
}
```

3.3 多重背包

```
#include <cstdio>
#include <iostream>
#include <algorithm>
#include <cstring>
#define rep(i,a,b) for(int i = a;i <= b;i++)
using namespace std;
const int N = 500;

int v[N],w[N],num[N];  //每袋的价格、重量和袋数
int dp[N];  //dp[i]:总经费为i时所能买的最多重量的大米
int n,total;  //大米种类、总经费

void ZeroOnePack(int cost,int weight)
{
    for(int i = total; i >= cost;i--)
        dp[i] = max(dp[i],dp[i-cost]+weight);
}

//完全背包，代价为cost，获得的价值为weight
void CompletePack(int cost,int weight)
{
    for(int i = cost;i <= total;i++) //total为上界
        dp[i] = max(dp[i],dp[i-cost]+weight);
}

//多重背包，代价为cost，获得的价值为weight
void MultiplePack(int cost,int weight,int amount)
{
    if(cost*amount >= total) CompletePack(cost,weight);  //相当于取任意袋数，变为完全背包
    else{
        int k = 1;
        while(k < amount){
            ZeroOnePack(k*cost,k*weight);  //二进制拆分
            amount -= k;
```

```
35              k<<=1;
36          }
37          ZeroOnePack(amount*cost,amount*weight);  //把剩余个数用01背包求
38          //这个很重要，不要忘记了
39      }
40  }
41
42  int main()
43  {
44      int T;
45      scanf("%d",&T);
46      while(T--)
47      {
48          memset(dp,0,sizeof dp);  //memset别忘了，很重要!
49          scanf("%d%d",&total,&n);
50          rep(i,1,n)
51              scanf("%d%d%d",&v[i],&w[i],&num[i]);
52          rep(i,1,n)
53              MultiplePack(v[i],w[i],num[i]);
54          printf("%d\n",dp[total]);
55      }
56      return 0;
57  }
```

3.4 状压 dp

题意：
一共 N 个人，给出任意两个人之间的胜负关系，你的编号是 M。现在需要安排一棵竞赛树使得 M 能够胜出，问使竞赛树高度最小且 M 获胜的安排方案一共有多少个。$(1 \leq N \leq 16)$

思路：
根据题意以及数据范围，可以很明显的发现这是一个状压 dp，因此我们来考虑 dp 的状态。

既然是状态，那肯定要记录当前的状态，即选了哪些人，然后还要记录当前的胜利者，以及当前树的高度，因此 $dp[i][j][k]$ 表示 i 状态下，胜出者为 j，树高度为 k 的安排方案数。

然后采用记忆化搜索，求取 $dp[i][j][k]$ 时将 i 分为两个子状态 x、y，然后递归求取 $dp[x][j][k-1]$，枚举 m 为 y 状态下的胜利者并且会输给 j，求取 $dp[y][m][k]$。具体细节见代码。

```
1   #include <cstdio>
2   #include <iostream>
3   #include <cstring>
4   #include <cmath>
5   #include <algorithm>
6   #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
7   #define rep(i,a,b) for(int i = a; i <= b; i++)
8   #define LOG1(x1,x2) cout << x1 << ": " << x2 << endl;
9   #define LOG2(x1,x2,y1,y2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << endl
        ;
10  #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2
        << " , " << z1 << ": " << z2 << endl;
11  typedef long long ll;
12  typedef double db;
13  const int N = 1e5+100;
14  const int M = 1e5+100;
15  const db EPS = 1e-9;
```

```
using namespace std;

template<class T> int getbit(T s, int i) { return (s >> i) & 1; }
template<class T> T onbit(T s, int i) { return s | (T(1) << i); }
template<class T> T offbit(T s, int i) { return s & (~(T(1) << i)); }
template<class T> int cntbit(T s) { return __builtin_popcount(s);}

int c[20][20],n,win,h[20];
ll dp[1<<16][17][7];

ll solve(int stat,int m,int height){
    if(dp[stat][m][height] != -1) return dp[stat][m][height];
    if(cntbit(stat) == 1){
        if(getbit(stat,m-1)) return dp[stat][m][height] = 1ll;
        else return dp[stat][m][height] = 0;
    }
    for(int i = (stat-1)&stat; i >= 1; i = (i-1)&(stat)){ //枚举子集
        int x = i, y = stat-i;
        if(!getbit(x,m-1) || h[cntbit(x)] > height-1 || h[cntbit(y)] > height-1)
    continue;
        // if(height-1 > cntbit(x) || height-1 > cntbit(y)) continue;
        ll ans1 = solve(x,m,height-1);
        ll ans2 = 0;
        rep(j,1,n)
            if(getbit(y,j-1) && c[m][j] == 1) ans2 += solve(y,j,height-1);
        if(dp[stat][m][height] == -1) dp[stat][m][height] = ans1*ans2;
        else dp[stat][m][height] += ans1*ans2;
    }
    return max(0ll,dp[stat][m][height]);
}

int main()
{
    freopen("f.in","r",stdin);
    freopen("f.out","w",stdout);
    scanf("%d%d",&n,&win);
    rep(i,1,n)
        rep(j,1,n) scanf("%d",&c[i][j]);
    rep(i,1,16) h[i] = (log(i-0.5)/log(2))+2;
    memset(dp,-1,sizeof dp);
    printf("%lld\n",max(0ll,solve((1<<n)-1,win,h[n])));
    return 0;
}
```

3.5 区间 dp

3.5.1 区间 dp 例题 1

题意：

给定一个序列。将序列中的一个数字消去的代价是与这个数字相邻的两个数字的 gcd，问将所有数字消去的最小代价。注意这个序列是环形的。

思路：

首先我们比较容易发现这是一个区间 DP 问题，于是问题就变成了如何列区间 DP 状态。一开始考虑的是 dp[i][j] 表示区间 [i,j] 全部消去的最小代价，然后在区间 [i,j] 中枚举第一个消去的 k 进行更新。然后会发现一个问题，如何先消的是 j ，那么 j 是右端点，因此 j 两端的元素是不确定的，因此这个转移方程不对。

因此我们来重新考虑这道题。由于左右端点不确定，因此我们重新定义 dp 状态，dp[i][j] 表示区间 [i, j] 中

所有数全部消除，最后剩下 i 和 j 的最小代价。则在区间中枚举 k ， dp[i][j] = dp[i][k]+dp[k][j]+gcd(i,j) 。由于是个环形序列，因此需要将长度扩展两倍进行 dp。最后的答案就是枚举最后剩下的两个点，然后找最小值即可。

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <algorithm>
#define rep(i,a,b) for(int i = a; i <= b; i++)
#define LOG1(x1,x2) cout << x1 << ": " << x2 << endl;
#define LOG2(x1,x2,y1,y2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << endl;
typedef long long ll;
typedef double db;
const db EPS = 1e-9;
using namespace std;
const int N = 300;

int n,a[N],dp[N][N];

int gcd(int a,int b)
{
    return b == 0 ? a:gcd(b,a%b);
}

int main()
{
    while(~scanf("%d",&n))
    {
        if(n == 0) break;
        rep(i,1,n) scanf("%d",&a[i]);
        rep(i,n+1,2*n) a[i] = a[i-n];
        rep(len,1,n)
            rep(j,1,2*n){
                int x = j, y = x+len-1;
                if(y > 2*n) continue;
                if(y == x+1 || y == x) dp[x][y] = 0;
                else{
                    dp[x][y] = 10000;
                    rep(k,x+1,y-1) dp[x][y] = min(dp[x][y],dp[x][k]+dp[k][y]+gcd(a[x],a[y]));
                }
                // printf("dp[%d][%d]:%d\n",x,y,dp[x][y]);
            }
        int ans = 10000;
        rep(i,1,n)
            rep(j,i+1,i+n-1){
                if(ans > dp[i][j]+dp[j][i+n]+gcd(a[i],a[j])){
                    ans = dp[i][j]+dp[j][i+n]+gcd(a[i],a[j]);
                    // LOG1("ans",ans);
                    // LOG2("i",i,"j",j);
                }
            }
        printf("%d\n",ans);
    }
    return 0;
}
```

3.5.2 区间 dp 例题 2

题意：
一个序列，选手 A、B 轮流从序列中从左端或者右端选一段区间，然后区间和加到自己的权值中。两个选手都会按照最优的方式进行选取，问先手 A 最多可以比 B 多拿多少。$(1 \leq n \leq 100)$

思路：
很明显这是一道 DP 问题，又因为只能从左端点或右端点拿，因此不难想到用区间 DP 的方法来解决此题。

既然是区间 DP，那么最常见的状态就是 $DP[i][j]$ 表示对于区间 $[i,j]$ ，先手最多领先后手多少。又因为区间和是一定的，因此已知选手 A 获得的价值就可以知道选手 B 获得的价值，因此修改状态为 $DP[i][j]$ 表示区间 $[i,j]$，先手最多可以获得多少价值。

因此 $DP[i][j] = max(sum[i][j] - DP[x][j], sum[i][j] - DP[i][y])\ i < x, y < j$，由于 n 比较小，直接枚举区间长度，从小区间到大区间进行转移即可。

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <algorithm>
#define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
#define rep(i,a,b) for(int i = a; i <= b; i++)
#define LOG1(x1,x2) cout << x1 << ": " << x2 << endl;
#define LOG2(x1,x2,y1,y2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << endl
    ;
typedef long long ll;
typedef double db;
const db EPS = 1e-9;
using namespace std;
const int N = 200;

int n,a[N],sum[N],dp[N][N];
//dp[i][j]: 表示区间[i,j]先手能够获得的最大价值

int main()
{
    while(~scanf("%d",&n))
    {
        if(!n) break;
        memset(dp,0,sizeof dp);
        rep(i,1,n) scanf("%d",&a[i]);
        rep(i,1,n) sum[i] = sum[i-1]+a[i];
        rep(len,1,n)
            rep(i,1,n-len+1){
                int j = i+len-1;
                dp[i][j] = sum[j]-sum[i-1];
                rep(k,1,len-1){
                    dp[i][j] = max(sum[j]-sum[i-1]-dp[i+k][j],dp[i][j]);
                    dp[i][j] = max(sum[j]-sum[i-1]-dp[i][j-k],dp[i][j]);
                }
            }
        int ans = dp[1][n]-(sum[n]-sum[0]-dp[1][n]);
        printf("%d\n",ans);
    }
    return 0;
}
```

3.6 数位 dp

题意:

给定 k 与 b，求出所有 k 在 0 (2^b-1) 范围内的倍数，将这些倍数二进制中的 1 求 sum 和，模 1e9+9 输出。

思路:

首先，这是一个在数位上的 dp，重点就在于如何描述每个数的状态。

发现数的范围很大，想要直接描述是不可能的。但是 k 的范围很小，只有 1000，因此考虑存储这个数

然后就可以列出 dp 方程，dp[i][j] 表示前 i 个二进制位，mod k = j 的个数，再用 ans[i][j] 表示前 i 个二进制位，mod k = j 的每一种情况二进制拆分后 1 的总和。

因为在 mod 意义下的加减都是可以的。因此对于第 i 个位置，我们只需考虑此处为 0 还是 1，只有两个状态，然后就可以列出转移方程。

dp[i][j] = dp[i-1][j]+dp[i-1][(j-poww[i]+k)
ans[i][j] = ans[i-1][j]+ans[i-1][(j-poww[i]+k)

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <algorithm>
#define rep(i,a,b) for(int i = a; i <= b; i++)
using namespace std;
const int mod = 1e9+9;

int k,b,dp[150][1100],poww[200],ans[150][1100];

int main()
{
    scanf("%d%d",&k,&b);
    poww[1] = 1%k;
    rep(i,2,130) poww[i] = (poww[i-1]*2)%k;
    dp[1][0] = 1, dp[1][1] = 1;
    ans[1][0] = 0, ans[1][1] = 1;
    if(k == 1) dp[1][0] = 2, dp[1][1] = 0, ans[1][0] = 1, ans[1][1] = 0;
    rep(i,2,b)
        rep(j,0,k-1){
            dp[i][j] = (dp[i-1][j]+dp[i-1][(j-poww[i]+k)%k])%mod;
            ans[i][j] = ((ans[i-1][j]+ans[i-1][(j-poww[i]+k)%k])%mod+dp[i-1][(j-poww[i
    ]+k)%k])%mod;
        }
    printf("%d\n",ans[b][0]);
    return 0;
}
```

3.7 斜率优化 dp

3.7.1 斜率优化例题

题意:

n 个需要被处理的任务，机器启动时间为 s。每个任务都有时间 Ti 和花费 Ci，计算方法为完成这批任务所需的时间是各个任务需要时间的总和。注意，同一批任务将在同一时刻完成，新的一批任务开始时，机器需要重新启动。确定一个分组方案，使得总费用最小。【每批任务包含相邻的若干任务】

思路:

1 <= n <= 3*1e5，0 <= S,Ci <= 512，-512 <= Ti <= 512。

首先我们需要列出 dp 方程，dp[i] 表示完成前 i 个任务的最小费用。此处由于涉及到 s，因此我们需要用到一个叫做“费用提前计算”的思想。因为 s 对于后续的每个任务都有影响，因此我们应当将 s 对后续任务的影响提前计算。

假设 dp[i] 由 dp[j] 更新而来，则 dp[i] = dp[j] + sumT[i]*(sumC[i]-sumC[j]) + s*(sumC[N]-sumC[j]); 将转移方程拆开，则可以得到 dp[j] - s*sumC[j] = sumT[i]*sumC[j]+dp[i]-s*sumC[N]-sumT[i]*sumC[i]，由此我们可以发现令横坐标 x = sumC[j]，纵坐标 y = dp[j]-s*sumC[j]，则对于每一个 j，都可以在平面中确定一个 (x,y) 坐标，而直线的斜率也是固定的，为 sumT[i]。

因此在平面中的这么多点中，我们需要维护一个下凸壳，更新 dp[i] 的时候，只需要在下凸壳中，二分 x 点与 x+1 点的斜率是否小于 sumT[i]，找到一个点，该点左边的斜率小于 k，右边斜率大于 k，则该点即为该下凸壳中的最优点。

再谈一下如何维护下凸壳，用一个数组，末尾为 qt，则判断 qt-1 与 x 的斜率是否小于等于 qt-1 与 qt 的斜率，如果小于等于，则需要弹出 qt，然后不断继续往下更新。注意此处如果是等于也需要弹出。

本题在维护凸壳时，可能会爆 long long，需要转成 long double。

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <algorithm>
#define rep(i,a,b) for(int i = a; i <= b; i++)
using namespace std;
const int N = 1e6+100;
typedef long long ll;

int n,qt;
ll s,T[N],C[N],sumT[N],sumC[N],dp[N],q[N],xx[N],yy[N];

void calc(int x)    //在凸壳中二分最优点，斜率优化
{
    int l = 1, r = qt-1, ans = q[l];
    ll k = sumT[x];
    while(l <= r)
    {
        int mid = (l+r)>>1;
        if((long double)(yy[q[mid+1]]-yy[q[mid]]) <= (long double)(xx[q[mid+1]]-xx[q[mid]])*(long double)k)
            ans = q[mid+1], l = mid+1;
        else r = mid-1;
    }
    dp[x] = yy[ans]-k*xx[ans]+s*sumC[n]+sumT[x]*sumC[x];
}

void solve()
{
    qt = 1, q[1] = 0;
    xx[0] = yy[0] = 0;
    rep(i,0,n) xx[i] = 0, yy[i] = 0;
    rep(i,1,n)
    {
        calc(i);
        xx[i] = sumC[i], yy[i] = dp[i]-s*sumC[i];
        while(qt >= 2){
            //重点在于 >=, > 会wa
            if((long double)(yy[q[qt]]-yy[q[qt-1]])*(long double)(xx[i]-xx[q[qt-1]]) >= (long double)(yy[i]-yy[q[qt-1]])*(long double)(xx[q[qt]]-xx[q[qt-1]]))
                qt--;
            else break;
```

```
        }
        qt++; q[qt] = i;
    }
    printf("%lld\n",dp[n]);
}

int main()
{
    while(~scanf("%d%lld",&n,&s))
    {
        sumT[0] = sumC[0] = 0;
        rep(i,1,n){
            scanf("%lld%lld",&T[i],&C[i]);
            sumT[i] = sumT[i-1]+T[i];
            sumC[i] = sumC[i-1]+C[i];
        }
        solve();
    }
    return 0;
}
```

3.7.2 动态维护下凸壳

题意:

给定一颗树，单向边，给出每个点的价值。然后任选树中一个点 i 进入，从 j 点出来，获得的价值为 val[i]*1+val[i+1]*2+...+val[j]*(j-i+1)，也可以选择不进入，节点价值有正有负，求最多可以获得多少价值。

思路:

这题可以简化成在一个序列上,找出一段 [x,y],求 SUM(val[i]*(i-x+1)) 的最大值,即 1*val[1]+2*val[2]+3*val[3]+... 的问题。

我们可以列一下 dp 方程，dp[i] 表示以 i 为右端点的区间和最大值，假设 dp[i] 的最优解是 [j+1, i] 这一段，则 dp[i] = f[i]-f[j]-(sum[i]-sum[j])*j，sum[i] 表示从根节点到 i 路径上所有点的权值之和，f[i] 表示从根节点到 i 路径上所有点按照 1*val[1]+2*val[2]+3*val[3] 的方式得到的累加和。

将 dp 方程拆成斜率优化的形式，可以得到 f[j]-j*sum[j] = -sum[i]*j+f[i]-dp[i]，而 dp[i] 就是答案。因此令横坐标为 j，纵坐标为 f[j]-j*sum[j]，在图中标出这些点，维护一个下凸壳，然后在下凸壳中二分左边斜率最接近-sum[i] 的点即为最优点。因此问题变成了如何维护下凸壳，正常的维护方法是用单调栈，添加一个点之后就将其他的点一一弹出,直到该点到达合理位置。但由于现在是在树上维护凸壳,因此下凸壳的序列会不断变化,因此我们需要进行动态维护。

可以发现每次往下凸壳中加入一个点，这个点最终都会到达原凸壳中的某一个位置，因此每次添加一个点，只是 O(1) 修改，然后我们在修改完之后再改回来，然后再维护原来的凸壳长度，即可实现动态维护。如何求该点最终到达的位置，只需要进行二分，找到点 x 与 x+1 之间斜率最接近 x 与新点之间斜率的点 x，x+1 处即为新点最终的更新位置。因此本题就是两个二分 +dfs 结束。

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <algorithm>
#define rep(i,a,b) for(int i = a; i <= b; i++)
using namespace std;
typedef long long ll;
const int N = 1e5+1000;

int n,q[N],fa[N],head[N],tot,qh,qt;
ll val[N],ans,x0[N],y0[N];
struct Edge{
```

```
13      int to,next;
14  }e[N];
15
16  void init()
17  {
18      tot = 1;
19      rep(i,0,n) head[i] = 0;
20  }
21
22  void add(int x,int y)
23  {
24      e[++tot].to = y, e[tot].next = head[x], head[x] = tot;
25  }
26
27  bool jud(int a,int b,int c)
28  {
29      return ((y0[a]-y0[b])*(x0[c]-x0[b]) > (y0[c]-y0[b])*(x0[a]-x0[b]));
30  }
31
32  int calc1(int l,int r,int x)    //维护凸壳
33  {
34      int tp = r+1; r = r-1;
35      while(l <= r)
36      {
37          int mid = (l+r)>>1;
38          if(jud(q[mid+1],q[mid],x)) tp = mid+1, r = mid-1;
39          else l = mid+1;
40      }
41      return tp;
42  }
43
44  void calc2(int l,int r,ll k,ll f)   //斜率优化dp,找到第一个右边斜率比k大的点
45  {
46      int tp = q[l]; r--;
47      while(l <= r)
48      {
49          int mid = (l+r)>>1;
50          if(y0[q[mid+1]]-y0[q[mid]] < k*(x0[q[mid+1]]-x0[q[mid]])) tp = q[mid+1], l =
        mid+1;
51          else r = mid-1;
52      }
53      ans = max(ans,-y0[tp]+k*x0[tp]+f);
54  }
55
56  void dfs(int x,int dep,ll sum,ll f) //从1号点往下dfs
57  {
58      x0[x] = dep, y0[x] = f-dep*sum;
59      calc2(qh,qt,-sum,f);
60      int oldh = qh, oldt = qt;
61      int pos = calc1(qh,qt,x), old = q[pos]; q[pos] = x, qt = pos;
62      for(int i = head[x]; i; i = e[i].next)
63      {
64          int y = e[i].to;
65          dfs(y,dep+1,sum+val[y],f+(dep+1)*val[y]);
66      }
67      qh = oldh, qt = oldt, q[pos] = old;
68  }
69
70  int main()
```

```
{
    int T; scanf("%d",&T);
    while(T--)
    {
        init();
        ans = 0;
        scanf("%d",&n);
        rep(i,1,n) scanf("%lld",&val[i]);
        rep(i,2,n) scanf("%d",&fa[i]), add(fa[i],i);
        qh = qt = 1; q[1] = 0;
        dfs(1,1,val[1],val[1]);
        printf("%lld\n",ans);
    }
    return 0;
}
```

4 数据结构

4.1 单调栈

给出一个序列，长度为 n。定义区间价值为区间和 * 区间最小值，求出这个序列中的最大区间价值。($n \leq 10^5, 0 \leq ai \leq 10^6$)

回忆一下单调栈和单调队列，单调栈——对于序列中每个点，求出序列中左/右边第一个比它大/小的点，单调队列——对于序列中每个点，求出序列中距离该点 K 步范围内的最小/大值。

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <stack>
#include <algorithm>
#define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
#define rep(i,a,b) for(int i = a; i <= b; i++)
#define LOG1(x1,x2) cout << x1 << ": " << x2 << endl;
#define LOG2(x1,x2,y1,y2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << endl;
#define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << " , " << z1 << ": " << z2 << endl;
typedef long long ll;
typedef double db;
const int N = 1e5+100;
const int M = 1e5+100;
const db EPS = 1e-9;
using namespace std;

int n,a[N],tt1[N],tt2[N];
ll sum[N];
stack<int> st;

int main()
{
    scanf("%d",&n);
    rep(i,1,n) scanf("%d",&a[i]);
    rep(i,1,n) tt1[i] = 0, tt2[i] = n+1;
    st.push(1);
    rep(i,2,n){
        while(st.size() && a[i] < a[st.top()]){
            int x = st.top();
            st.pop();
            tt2[x] = i;
            // LOG1("x",x);
        }
        if(st.size()) tt1[i] = st.top();
        st.push(i);
    }
    rep(i,1,n) sum[i] = sum[i-1]+(ll)a[i];
    ll ans = 0;
    int l = 1,r = 1;
    rep(i,1,n){
        if(ans < a[i]*(sum[tt2[i]-1]-sum[tt1[i]])){
            ans = a[i]*(sum[tt2[i]-1]-sum[tt1[i]]);
            l = tt1[i]+1;
            r = tt2[i]-1;
        }
    }
```

```
48      printf("%lld\n",ans);
49      printf("%d %d\n",l,r);
50      return 0;
51  }
```

4.2 单调队列

题意:

给定一串 01 串，m 次询问，每次询问给你一个数 k。k 为对于这个 01 串所能进行的最多次操作，每次操作可以将该串中任意一个位置的数移到任意一个其他的位置。

每次询问之后，输出在这个操作数之内，所能达到的最长的连续 0 的长度。

```
1   #include <cstdio>
2   #include <iostream>
3   #include <algorithm>
4   #include <cstring>
5   #define rep(i,a,b) for(int i = a;i <= b;i++)
6   using namespace std;
7   const int N = 1e6+1000;
8
9   char s[N];
10  int m,n,a[N],sum[N],q[N],cnt;
11
12  void solve(int k)
13  {
14      int ans = 0;
15      q[1] = 0;
16      int l = 1, r = 1;
17      rep(i,1,n)
18      {
19          while(l <= r && a[q[r]] >= a[i]) r--;   //维护单调队列，先对右端点进行更新
20          q[++r] = i;
21          while(l <= r && sum[i]-sum[q[l]] > k) l++;  //再对左端点进行更新
22          ans = max(ans,a[q[r]]-a[q[l]]+k);
23      }
24      printf("%d\n",min(ans,cnt));
25  }
26
27  int main()
28  {
29      while(~scanf("%s",s))
30      {
31          cnt = 0;
32          scanf("%d",&m);
33          int len = strlen(s);
34          n = len;
35          sum[0] = 0;a[0]=0;
36          rep(i,1,len)
37          {
38              if(s[i-1] == '0'){
39                  sum[i] = sum[i-1];
40                  cnt++;
41              }
42              else sum[i] = sum[i-1]+1;
43              a[i] = i-2*sum[i];
```

```
44
45          }
46          rep(i,1,m)
47          {
48              int x;
49              scanf("%d",&x);
50              solve(x);
51          }
52      }
53      return 0;
54  }
```

4.3 并查集

4.3.1 普通并查集

```
1   #include <cstdio>
2   #include <iostream>
3   using namespace std;
4   const int maxn=10000+10;
5
6   int n,m,p[maxn];
7
8   int find(int x) {
9       return p[x]==x?x:p[x]=find(p[x]);
10  }
11
12  void merge(int x,int y){
13      int r1=find(x),r2=find(y);
14      if(r1!=r2){
15          p[r1]=r2;
16      }
17  }
18
19  int main()
20  {
21      cin>>n>>m;
22      for(int i=1;i<=n;i++){
23          p[i]=i;
24      }
25      for(int i=1;i<=m;i++){
26          int x,y;
27          cin>>x>>y;
28          p[x]=y;
29      }
30      return 0;
31  }
```

4.3.2 带权并查集

题意：N 个人在玩剪刀石头布的游戏，其中有一个人是裁判，其余人随机被分到了三个阵营，即剪刀、石头、布。现在有 M 轮游戏，$x<,>,=y$ 表示 x 与 y 之间的关系，其中裁判可以自由变换阵营。问是否可以根据这 M 轮游戏，判断出谁是裁判，输出裁判是谁以及最早在哪一轮可以找到裁判。如果无法判断，则输出 $Can\ not\ determine$，如果游戏的情况不符合题意，则输出 $Impossible$ 。($N \leq 500, M \leq 2000$)

思路：一开始做这题的时候，的确有些懵，只能想到如何发现有人变换了阵营，但是不知道如何找到这个人，并且确定这种情况是唯一的。

所以我们可以发现直接考虑整个问题会非常困难，再加上此题数据范围很小，直接做的话复杂度肯定很少，太对不起这个数据范围了。因此我们考虑 N^2 做法，即枚举每个人为裁判。枚举 x 为裁判时，如果 x 恰好为裁判，则并查集合并时不会发生矛盾。我们统计有多少个人为裁判时，并查集合并会发生矛盾，记人数为 cnt，并且统计每个人为裁判时，发生矛盾的轮数 ri 。

现在我们来考虑输出答案的所有情况。
1. 如果 $cnt = n - 1$，则可以唯一确定裁判，并且最早发现该裁判的轮数为 $max(ri)$。
2. 如果 $cnt = n$，则输出 $impossible$。
3. 其余情况则为 $Can\ not\ determine$。

至于带权并查集的合并类似于食物链问题，是个模 3 剩余系中的加减问题。

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <algorithm>
#define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
#define rep(i,a,b) for(int i = a; i <= b; i++)
#define LOG1(x1,x2) cout << x1 << ": " << x2 << endl;
#define LOG2(x1,x2,y1,y2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << endl
    ;
typedef long long ll;
typedef double db;
const db EPS = 1e-9;
using namespace std;
const int N = 1000;
const int M  = 2000+100;
int n,m,fa[N],d[N],vis[N],error[N],cnt;

int find(int x){
    if(x == fa[x]) return x;
    int root = find(fa[x]);
    d[x] = (d[x]+d[fa[x]]+3)%3;
    return fa[x] = root;
}

struct Query{
    int x,y,cc;
}q[M];

int main()
{
    while(~scanf("%d%d",&n,&m))
    {
        memset(vis,0,sizeof vis);
        memset(error,0,sizeof error);
        cnt = 0;
        rep(i,1,m){
            int x,y; char cc;
            scanf("%d%c%d",&x,&cc,&y);
            q[i].x = x, q[i].y = y;
            if(cc == '=') q[i].cc = 0;
            else if(cc == '<') q[i].cc = 1;
            else q[i].cc = 2;
        }
        rep(i,0,n-1){
            rep(k,0,n) fa[k] = k, d[k] = 0;
            rep(j,1,m){
```

```
46                  int x = q[j].x, y = q[j].y, cc = q[j].cc;
47                  if(x == i || y == i) continue;
48                  int fx = find(x), fy = find(y);
49                  if(fx != fy){
50                      fa[fx] = fy;
51                      d[fx] = (d[y]-d[x]+cc+3)%3;
52                  }
53                  else{
54                      int ttp = (d[x]-d[y]+3)%3;
55                      if(ttp != cc){
56                          cnt++;      // 产生矛盾个数
57                          error[i] = j;
58                          vis[i] = 1;
59                          break;
60                      }
61                  }
62              }
63          }
64          // impossible
65          // not determine
66          // determine
67          if(cnt == (n-1)) { //determine
68              int ans = 0, hm;
69              rep(i,0,n-1){
70                  ans = max(ans,error[i]);
71                  if(!vis[i]) hm = i;
72              }
73              printf("Player %d can be determined to be the judge after %d lines\n",hm,
    ans);
74          }
75          else if(cnt == n){ // impossible
76              printf("Impossible\n");
77          }
78          else{  // not determine
79              printf("Can not determine\n");
80          }
81      }
82      return 0;
83  }
```

4.3.3 带权并查集与背包

题意：给出 $p1$ 个天使，$p2$ 个魔鬼，一共有 n 个问题。每个问题的格式为 x y (yes or no) 表示问 x y 是否为天使，如果 x 为天使，则会说真话，如果 x 为魔鬼，则会说假话。问根据这 n 个问题，是否可以确定哪些人为天使，如果可以确定，按编号大小输出天使编号。($n \leq 1000$， $p1, p2 \leq 300$)

思路：遇到这样的 yes or no 问题，显然我们需要对于可能出现的情况进行模拟。
1. 假如 x 为天使，y 为天使，则回答 yes 。
2. 假如 x 为天使，y 为魔鬼，则回答 no 。
3. 假如 x 为魔鬼，y 为天使，则回答 no 。
4. 假如 x 为魔鬼，y 为魔鬼，则回答 yes 。

可以发现，如果回答是 yes，则 x 与 y 属于同一类。如果回答 no，则 x 与 y 类别相反。因此一个模 2 剩余系的带权并查集合并就可以维护所有人之间的关系。

因此不难发现，处理完 n 个问题之后，我们会拥有若干个并查集，每个并查集中都会分为两批人，两批人类别不同。

因此问题变成了，从这若干个并查集中随机选一部分，使得最后选中的人数恰好为 $p1$，并且这种选择方式唯一，则我们可以确定哪些人为天使。因此本题就变成了一个类似背包的问题，$dp[i][j]$ 表示前 i 个并查集选取人数为 j 一共有多少种选择方案，$pre[i][j] = x$ 表示 $dp[i][j]$ 由 $dp[i-1][x]$ 更新而来。到此，本题即可顺利解决。

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <map>
#include <algorithm>
#define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
#define rep(i,a,b) for(int i = a; i <= b; i++)
#define LOG1(x1,x2) cout << x1 << ": " << x2 << endl;
#define LOG2(x1,x2,y1,y2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << endl
    ;
typedef long long ll;
typedef double db;
const db EPS = 1e-9;
using namespace std;
const int N = 700;

int n,p1,p2,fa[N],d[N],base[N][2],tot,dp[N][600],pre[N][600],vis[N];
map<int,int> mp;

int find(int x)
{
    if(x == fa[x]) return x;
    int root = find(fa[x]);
    d[x] = (d[x]+d[fa[x]]+2)%2;
    return fa[x] = root;
}

void solve()
{
    memset(base,0,sizeof base);
    memset(vis,0,sizeof vis);
    mp.clear();
    tot = 0;
    rep(i,1,p1+p2){
        fa[i] = find(i);
        if(mp.find(fa[i]) == mp.end()) mp[fa[i]] = ++tot;
        int pos = mp[fa[i]];
        base[pos][d[i]]++;  //0: 相同   1: 不同
    }
    memset(dp,0,sizeof dp);
    memset(pre,0,sizeof pre);
    dp[1][base[1][1]]++;
    dp[1][base[1][0]]++;
    rep(i,2,tot){
        int minn = min(base[i][0],base[i][1]);
        rep(j,minn,p1){
            if(j >= base[i][0] && dp[i-1][j-base[i][0]] != 0)
                dp[i][j] += dp[i-1][j-base[i][0]], pre[i][j] = j-base[i][0];
            if(j >= base[i][1] && dp[i-1][j-base[i][1]] != 0)
                dp[i][j] += dp[i-1][j-base[i][1]], pre[i][j] = j-base[i][1];
        }
    }
    if(dp[tot][p1] != 1) printf("no\n");
    else{
        int xx = tot, yy = p1;
```

```
55          while(xx > 1){
56              if(pre[xx][yy] == yy-base[xx][0]) vis[xx] = 0, yy -= base[xx][0];
57              else if(pre[xx][yy] == yy-base[xx][1]) vis[xx] = 1, yy -= base[xx][1];
58              xx--;
59          }
60          if(xx == 1){
61              if(base[1][0] == yy) vis[1] = 0;
62              else vis[1] = 1;
63          }
64          rep(i,1,p1+p2){
65              int pos = mp[fa[i]];
66              int dd = d[i];
67              if(vis[pos] == 1 && dd == 1) printf("%d\n",i);
68              else if(vis[pos] == 0 && dd == 0) printf("%d\n",i);
69          }
70          printf("end\n");
71      }
72  }
73
74  int main()
75  {
76      while(~scanf("%d%d%d",&n,&p1,&p2))
77      {
78          if((n+p1+p2) == 0) break;
79          rep(i,0,p1+p2) fa[i] = i, d[i] = 0;
80          rep(i,1,n){
81              int x,y; char op[10];
82              scanf("%d%d",&x,&y);
83              scanf("%s",op);
84              int fx = find(x), fy = find(y);
85              // LOG2("x",x,"y",y);
86              // LOG2("fx",fx,"fy",fy);
87              if(fx != fy){
88                  fa[fx] = fy;
89                  if(op[0] == 'n') d[fx] = (2+1+d[y]-d[x])%2;
90                  else d[fx] = (d[y]-d[x]+2)%2;
91              }
92          }
93          solve();
94      }
95      return 0;
96  }
```

4.3.4 按秩合并并查集

题意:
n 个点，m 个操作，操作共两类。 $1\ u\ v$ 表示在图中加一条边连接 $u\ v$， $2\ u\ v$ 表示查询 u 与 v 最早是在哪一次加边操作后连通，不连通输 -1。$(1 \leq n, m \leq 10^5)$

思路:
维护连通性，最直观的想法就是用并查集来维护连通性。但是如何通过并查集来查看两点最早什么时候连通呢?

首先可以知道并查集维护的其实是一个森林，假如我们不破坏树的结构，即不进行路径压缩，则并查集每次加边，则将边权定义为操作编号，那么两点树上边权最大值就是两点最早连通的加边操作。

因此问题变成如何维护树的结构进行并查集合并，方法就是按秩合并，将小的树合并到大的树上，这样可以保证每个节点最多被合并 $log(n)$ 次，因此每个节点的高度最多为 $log(n)$。因此对于每次查询，我们可以将两个点到根节点的路

径直接取出，然后查询第一次遇到的位置，输出达到这个位置之前的边权最大值即可。

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <algorithm>
#define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
#define rep(i,a,b) for(int i = a; i <= b; i++)
#define LOG1(x1,x2) cout << x1 << ": " << x2 << endl;
#define LOG2(x1,x2,y1,y2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << endl;
#define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << " , " << z1 << ": " << z2 << endl;
typedef long long ll;
typedef double db;
const int N = 1e5+100;
const int M = 1e5+100;
const db EPS = 1e-9;
using namespace std;

int f[N],n,m,siz[N],d[N],flag[N],vis[N];

int find(int x){
    if(x == f[x]) return x;
    else return find(f[x]);
}

int solve(int u,int v){
    vis[u] = 1; flag[u] = 0; int w = 0;
    while(f[u] != u){
        w = max(w,d[u]);
        vis[f[u]] = 1; flag[f[u]] = w;
        u = f[u];
    }
    vis[u] = 1, flag[u] = w;
    w = 0;
    if(vis[v] == 1) return flag[v];
    while(f[v] != v){
        w = max(w,d[v]);
        if(vis[f[v]] == 1) return max(w,flag[f[v]]);
        v = f[v];
    }
    if(vis[v] == 1) return max(flag[v],w);
}

void clear(int u){
    while(f[u] != u) vis[u] = 0, flag[u] = 0, u = f[u];
    vis[u] = 0, flag[u] = 0;
}

int main()
{
    int _; scanf("%d",&_);
    while(_--){
        scanf("%d%d",&n,&m);
        rep(i,0,n) f[i] = i, d[i] = 0, siz[i] = 1;
        rep(i,1,m){
            int op,u,v; scanf("%d%d%d",&op,&u,&v);
            int xu = find(u), xv = find(v);
```

```
56              if(op == 1 && xu != xv){
57                  if(siz[xu] < siz[xv]) f[xu] = xv, d[xu] = i, siz[xv] += siz[xu];
58                  else f[xv] = xu, d[xv] = i, siz[xu] += siz[xv];
59              }
60              else if(op == 2){
61                  if(xu != xv) printf("-1\n");
62                  else{
63                      printf("%d\n",solve(u,v));
64                      clear(u);
65                  }
66              }
67          }
68      }
69      return 0;
70  }
```

4.4 Hash

题意:
给出 n 个串，m 组询问。每组询问均为一个字符串，询问在初始 n 个串中是否存在一个串与该询问串恰好只有一个位置不相同。输出 YES or NO。

Hash 思路:
首先先讲讲 Hash 的算法，利用 bkdr 算法将每个字符串 hash 成一个数值，hash 函数如下:

hash: abccac
hash[1] = 0
$\text{hash}[6] = 0 * 131^5 + 1 * 131^4 + 2 * 131^3 + 2 * 131^2 + 0 * 131 + 2$

此处 hash[6] 即是这个字符串的 hash 值，131 为 seed，即 hash 种子，然后还要取一个模数，即 mod。由此可以发现 hash 值即是字符串中每一个位置的贡献，所以本题要求恰好只有一个位置不相同，即可以枚举不相同位置，减去原有贡献，加上新贡献即可。然后本题就可以解决，但是由于此题的数据卡的很 e xin，所以 seed 和 mod 取的不好的话，会被卡成 zz。

这里补充一下常见的 seed 和 mod，seed 一般取质数，3、5、7、13、131、13131 这些均可，mod 一般也取质数，1e9+7，1e11+7，1e13+7，1e18+7 均可，也可以直接将数据类型取为 unsigned long long，即可对 2^64-1 直接取模。

有一个注意点，在计算过程中，seed*mod 不能超过数据类型的最大值，否则相当于在计算过程中又模上了一个不是 mod 的数，会导致结果错误。

由于本题只有三个字符，所以可以将 seed 定为 3，mod 定一个很大的数，类似于三进制。

```
1   #include <cstdio>
2   #include <iostream>
3   #include <cstring>
4   #include <map>
5   #include <algorithm>
6   #define rep(i,a,b) for(int i = a; i <= b; i++)
7   using namespace std;
8   typedef long long ull;
9   const int N = 6*1e5+1000;
10  const ull mod = 1e11+7;
11  const ull ttp1 = 3;
12  const ull ttp2 = 5;
13  int n,m;
14  ull base[5] = {0,1,2,131313,123};
15  ull seed1[N],seed2[N];
```

```
16 char s[N];
17 struct Node{
18     ull a,b;
19 }gn;
20 map<Node,int> mp;
21 
22 bool operator == (Node x, Node y)
23 {
24     if(x.a == y.a && x.b == y.b) return true;
25     else return false;
26 }
27 
28 bool operator < (Node x, Node y)
29 {
30     return x.a < y.a;
31 }
32 
33 int solve(ull hash1, ull hash2)
34 {
35     int len = strlen(s);
36     // printf("s:%s,len:%d\n",s,len);
37     rep(j,0,len-1)
38     {
39         rep(k,0,2)
40         {
41             if(s[j]-'a' == k) continue;
42             ull tmp1 = hash1;
43             tmp1 = (tmp1+(-base[s[j]-'a']+base[k])*seed1[len-1-j]%mod)%mod;
44             if(tmp1 < 0) tmp1 += mod;
45 
46             ull tmp2 = hash2;
47             tmp2 = (tmp2+(-base[s[j]-'a']+base[k])*seed2[len-1-j]%mod)%mod;
48             if(tmp2 < 0) tmp2 += mod;
49 
50             gn.a = tmp1, gn.b = tmp2;
51             // printf("k:%d,tmp:%llu\n",k,tmp);
52             if(mp[gn] == 1) return 1;
53         }
54     }
55     return 0;
56 }
57 
58 int main()
59 {
60     // printf("mod:%lld\n",mod);
61     mp.clear();
62     seed1[0] = 1;
63     seed1[1] = ttp1;
64     seed2[0] = 1;
65     seed2[1] = ttp2;
66     int _ = 6*1e5+100;
67     rep(i,2,_)
68     {
69         seed1[i] = (seed1[i-1]*ttp1)%mod;
70         if(seed1[i] < 0) seed1[i] += mod;
71 
72         seed2[i] = (seed2[i-1]*ttp2)%mod;
73         if(seed2[i] < 0) seed2[i] += mod;
74     }
```

```
    scanf("%d%d",&n,&m);
    rep(i,1,n)
    {
        scanf("%s",s);
        int len = strlen(s);
        ull hash1 = 0;
        ull hash2 = 0;
        rep(j,0,len-1)
        {
            hash1 = (hash1*ttp1%mod+base[s[j]-'a'])%mod;
            if(hash1 < 0) hash1+=mod;

            hash2 = (hash2*ttp2%mod+base[s[j]-'a'])%mod;
            if(hash2 < 0) hash2+=mod;
        }
        // printf("s:%s,hash:%llu\n",s,hash);
        gn.a = hash1, gn.b = hash2;
        mp[gn] = 1;
    }
    rep(i,1,m)
    {
        scanf("%s",s);
        int len = strlen(s);
        ull hash1 = 0;
        ull hash2 = 0;
        rep(j,0,len-1)
        {
            hash1 = (hash1*ttp1%mod+base[s[j]-'a'])%mod;
            if(hash1 < 0) hash1+=mod;

            hash2 = (hash2*ttp2%mod+base[s[j]-'a'])%mod;
            if(hash2 < 0) hash2+=mod;
        }
        if(solve(hash1,hash2))
            printf("YES\n");
        else printf("NO\n");
    }
    return 0;
}

//mod: 1e9+7, seed: 257
//mod: 1e18+3, seed: 3、5
//mod: 1e11+7, seed: 3、5、1313、...
//如果mod*seed会越界的话，那么结果就会错误，因为计算过程中出现了两个seed，多了自动溢出的那个seed
```

4.5 字典树

```
#include <cstdio>
#include <iostream>
#include <cstring>
using namespace std;
const int charset = 26;
const int max_node = 100000+10;

struct Trie
{
    int tot,root,child[max_node][charset];  //root:根节点  tot:节点编号
    //child[i][j]=k，表示编号为i的节点的第j个孩子是编号为k的节点
```

```
12      bool flag[max_node];   //是否以某一个字符为结束
13      Trie()
14      {
15  //      memset(child,0,sizeof(child));
16          memset(flag,0,sizeof(flag));
17          root = tot = 1;  //根节点编号为1
18      }
19      void mem()  //将字典树初始化
20      {
21          memset(child,0,sizeof(child));
22          memset(flag,0,sizeof(flag));
23          root = tot = 1;  //根节点编号为1
24      }
25      void insert(const char *str)
26      {
27          int cur = root;
28          for(int i = 0;str[i];i++)
29          {
30              int x = str[i]-'a';
31              if(child[cur][x] == 0)
32                  child[cur][x] = ++tot;
33              cur = child[cur][x];
34          }
35      //  flag[cur] = true; 记录单词以该点结束
36      }
37      bool query(const char *str)
38      {
39          int cur = root;
40          for(int i = 0;str[i];i++)
41          {
42              int x = str[i]-'a';
43              if(child[cur][x] == 0) return false;
44              cur = child[cur][x];
45          }
46          return true;
47          //查询单词时应该 return flag[cur];
48      }
49  }tre;
```

4.6 树状数组

```
1   int c[N];
2
3   inline int lowbit(int x) { return x&(-x); }
4
5   inline void update(int x, int c){
6       for(int i = x; i <= k; i += lowbit(i)) t[i] += c;
7   }
8
9   inline int ask(int x){
10      int tp = 0;
11      for(int i = x; i; i -= lowbit(i)) tp += t[i];
12      return tp;
13  }
```

4.7 线段树

4.7.1 动态开点与区间修改 lazy

```
#include <cstdio>
#include <algorithm>
#define rep(i,a,b) for(int i = a; i <= b; i++)
const int N = 1e5+100;
using namespace std;

int ls[2*N],rs[2*N],maxn[2*N],lazy[2*N],rt,sz,a[N],n; //rt: root, sz: 当前节点编号

void push_down(int now){
    if(ls[now] == 0) ls[now] = ++sz;
    if(rs[now] == 0) rs[now] = ++sz;
    maxn[ls[now]] += lazy[now], maxn[rs[now]] += lazy[now];
    lazy[ls[now]] += lazy[now], lazy[rs[now]] += lazy[now];
    lazy[now] = 0;
}
void update(int& now, int l, int r, int lx, int rx, int c){ //区间修改
    if(!now) now = ++sz;
    if(lx <= l && rx >= r){
        maxn[now] += c; lazy[now] += c;
        return;
    }
    if(lazy[now] != 0) push_down(now);
    int mid = (l+r)>>1;
    if(lx <= mid) update(ls[now],l,mid,lx,rx,c);
    if(rx > mid) update(rs[now],mid+1,r,lx,rx,c);
    maxn[now] = max(maxn[ls[now]],maxn[rs[now]]);
}
int query(int& now, int l, int r){  //查询最右边第一个值大于0的点
    if(!now) now = ++sz;
    if(maxn[now] <= 0) return -1;
    if(l == r) return a[l];
    if(lazy[now] != 0) push_down(now);
    int mid = (l+r)>>1;
    if(maxn[rs[now]] > 0) return query(rs[now],mid+1,r);
    else return query(ls[now],l,mid);
}

int main()
{
    scanf("%d",&n);
    rep(i,1,n){
        int pos,op; scanf("%d%d",&pos,&op);
        if(op == 1){ //push
            int xx; scanf("%d",&xx); a[pos] = xx;
            update(rt,1,n,1,pos,1);
        }
        else update(rt,1,n,1,pos,-1); //pop
        printf("%d\n",query(rt,1,n));
    }
    return 0;
}
```

4.7.2 区间递减序列和

题意：与楼房重建题意类似，但是求的是递减序列，而且询问的是区间 $[l,r]$ 的递减序列和。

思路：只需将维护内容的 $maxn$ 改为 $minn$ 即可，然后解决一下区间查询的问题。继续使用刚才的 $calc(now,tp)$ 函数，计算节点 now 在最小值为 tp 下的贡献。依然是左右区间二分，如果左区间最小值小于 tp，则直接计算右区间贡献，然后递归左区间。如果左区间最小值大于 tp，则直接递归右区间。

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <algorithm>
#define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
#define rep(i,a,b) for(int i = a; i <= b; i++)
#define LOG1(x1,x2) cout << x1 << ": " << x2 << endl;
#define LOG2(x1,x2,y1,y2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << endl;
#define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << " , " << z1 << ": " << z2 << endl;
typedef long long ll;
typedef double db;
const int N = 1e5+100;
const int M = 3*1e5+100;
const ll inf = 1e13;
const db EPS = 1e-9;
using namespace std;

int n,q,rt,sz,ls[M],rs[M];
ll a[N],sum[M],minn[M],ans;

ll calc(int &now,int l,int r,ll tp){
    if(!now) now = ++sz;
    if(l == r) return (minn[now]<tp?minn[now]:0ll);
    int mid = (l+r)>>1;
    if(minn[ls[now]] < tp) return (sum[now]-sum[ls[now]]+calc(ls[now],l,mid,tp));
    else return calc(rs[now],mid+1,r,tp);
}

void update(int &now,int l,int r,int pos,ll w){
    if(!now) now = ++sz;
    if(l == r){
        sum[now] = minn[now] = w;
        return;
    }
    int mid = (l+r)>>1;
    if(pos <= mid) update(ls[now],l,mid,pos,w);
    else update(rs[now],mid+1,r,pos,w);
    minn[now] = min(minn[rs[now]],minn[ls[now]]);
    sum[now] = sum[ls[now]]+calc(rs[now],mid+1,r,minn[ls[now]]);
}

ll query(int &now,int l,int r,int pos1,int pos2,ll w){
    if(!now) now = ++sz;
    if(pos1 <= l && pos2 >= r){
        ans += calc(now,l,r,w);
        return minn[now];
    }
    int mid = (l+r)>>1;
    if(pos1 <= mid) w = min(w,query(ls[now],l,mid,pos1,pos2,w));
    if(pos2 > mid) w = min(w,query(rs[now],mid+1,r,pos1,pos2,w));
    return w;
```

```
}

int main()
{
    int _; scanf("%d",&_);
    while(_--){
        rt = sz = 0;
        memset(ls,0,sizeof ls);
        memset(rs,0,sizeof rs);
        scanf("%d%d",&n,&q);
        rep(i,1,n){
            scanf("%lld",&a[i]);
            update(rt,1,n,i,a[i]);
        }
        rep(i,1,q){
            int l,r,p,c; scanf("%d%d%d%d",&l,&r,&p,&c);
            ans = 0, query(rt,1,n,l,r,inf);
            printf("%lld\n",ans);
            if(p != 0 || c != 0) update(rt,1,n,p,c);
        }
    }
    return 0;
}
```

4.8 扫描线

4.8.1 面积并

只要矩形一条线一条线扫描的时候，右边的线没有被读进来，则右边的线一直在扫描面积

举个简单的例子，本题中的扫描线只是一个消除矩阵面积重复计算部分的一个方法

```
#include <iostream>
#include <cstdio>
#include <algorithm>
using namespace std;
const int SIZE = 300+10;
struct Line{
    double x,y1,y2;
    int flag;
}line[SIZE];

bool cmp(Line a,Line b)
{
    return a.x<b.x;
}
struct tree{
    int l,r;
    double ml,mr;
    int s;   //s始终 >= 0
    double len;
}t[SIZE*4];
int n;
double y[SIZE];
void build(int p,int l,int r)
{
    t[p].l = l; t[p].r = r; t[p].ml = y[l]; t[p].mr = y[r];
    t[p].s = 0;
```

```
27      t[p].len = 0;
28      if(t[p].l+1 == t[p].r) return;
29      int mid = (l+r)>>1;
30      build(p*2,l,mid);
31      build(p*2+1,mid,r);
32  }
33
34  void callen(int p)
35  {
36      if(t[p].s > 0)
37          t[p].len = t[p].mr-t[p].ml;
38      else if(t[p].l == (t[p].r-1)) //所以s == 0的点，最终长度会被赋成0
39          t[p].len = 0;
40      else
41          t[p].len = t[p*2].len+t[p*2+1].len;
42  }
43
44  void change(int p,Line tmp)
45  {
46      if(t[p].ml == tmp.y1 && t[p].mr == tmp.y2)
47      {
48          t[p].s += tmp.flag;
49          callen(p);
50          return;
51      }
52      if(tmp.y2 <= t[p*2].mr) change(p*2,tmp);
53      else if(tmp.y1 >= t[p*2+1].ml) change(p*2+1,tmp);
54      else
55      {
56          Line tp = tmp;
57          tp.y2 = t[p*2].mr;
58          change(p*2,tp);
59          tp = tmp;
60          tp.y1 = t[p*2+1].ml;
61          change(p*2+1,tp);
62      }
63      callen(p);
64  }
65
66  int main()
67  {
68      int cnt = 1;
69      while(scanf("%d",&n) && n!=0)
70      {
71          int num = 1;
72          double x1,x2,y1,y2;
73          for(int i = 0;i < n;i++)
74          {
75              scanf("%lf%lf%lf%lf",&x1,&y1,&x2,&y2);
76              line[num].x = x1; line[num].y1 = y1; line[num].y2 = y2;
77              line[num].flag = 1;
78              //保存的是线，flag == 1 表示该线在左边
79              y[num++] = y1; //将所有的y都读入数组中，进行离散化
80              line[num].x = x2; line[num].y1 = y1; line[num].y2 = y2;
81              line[num].flag = -1;
82              //flag == -1 表示该线在右边
83              y[num++] = y2;
84          }
85          sort(line+1,line+num,cmp); //按照横坐标进行排序
```

```
86          //对纵坐标进行离散化
87          sort(y+1,y+num);
88          int cm = unique(y+1,y+num)-y-1;
89          //在y轴上建立线段树
90          build(1,1,cm);
91          change(1,line[1]);
92          double ans = 0;
93          for(int i = 2;i < num;i++)
94          {
95              ans += t[1].len*(line[i].x-line[i-1].x);
96              change(1,line[i]);
97          }
98          printf("Test case #%d\n",cnt++);
99          printf("Total explored area: %.2f\n\n",ans);
100     }
101     return 0;
102 }
```

4.8.2 面积交

只要矩形一条线一条线扫描的时候，右边的线没有被读进来，则右边的线一直在扫描面积

举个简单的例子，本题中的扫描线只是一个消除矩阵面积重复计算部分的一个方法

```
1  #include <iostream>
2  #include <cstdio>
3  #include <algorithm>
4  using namespace std;
5  const int SIZE = 300+10;
6  struct Line{
7      double x,y1,y2;
8      int flag;
9  }line[SIZE];
10
11 bool cmp(Line a,Line b)
12 {
13     return a.x<b.x;
14 }
15 struct tree{
16     int l,r;
17     double ml,mr;
18     int s;   //s始终 >= 0
19     double len;
20 }t[SIZE*4];
21 int n;
22 double y[SIZE];
23 void build(int p,int l,int r)
24 {
25     t[p].l = l; t[p].r = r; t[p].ml = y[l]; t[p].mr = y[r];
26     t[p].s = 0;
27     t[p].len = 0;
28     if(t[p].l+1 == t[p].r) return;
29     int mid = (l+r)>>1;
30     build(p*2,l,mid);
31     build(p*2+1,mid,r);
32 }
33
```

```
34 void callen(int p)
35 {
36     if(t[p].s > 0)
37         t[p].len = t[p].mr-t[p].ml;
38     else if(t[p].l == (t[p].r-1)) //所以s == 0的点，最终长度会被赋成0
39         t[p].len = 0;
40     else
41         t[p].len = t[p*2].len+t[p*2+1].len;
42 }
43
44 void change(int p,Line tmp)
45 {
46     if(t[p].ml == tmp.y1 && t[p].mr == tmp.y2)
47     {
48         t[p].s += tmp.flag;
49         callen(p);
50         return;
51     }
52     if(tmp.y2 <= t[p*2].mr) change(p*2,tmp);
53     else if(tmp.y1 >= t[p*2+1].ml) change(p*2+1,tmp);
54     else
55     {
56         Line tp = tmp;
57         tp.y2 = t[p*2].mr;
58         change(p*2,tp);
59         tp = tmp;
60         tp.y1 = t[p*2+1].ml;
61         change(p*2+1,tp);
62     }
63     callen(p);
64 }
65
66 int main()
67 {
68     int cnt = 1;
69     while(scanf("%d",&n) && n!=0)
70     {
71         int num = 1;
72         double x1,x2,y1,y2;
73         for(int i = 0;i < n;i++)
74         {
75             scanf("%lf%lf%lf%lf",&x1,&y1,&x2,&y2);
76             line[num].x = x1; line[num].y1 = y1; line[num].y2 = y2;
77             line[num].flag = 1;
78             //保存的是线，flag == 1 表示该线在左边
79             y[num++] = y1; //将所有的y都读入数组中，进行离散化
80             line[num].x = x2; line[num].y1 = y1; line[num].y2 = y2;
81             line[num].flag = -1;
82             //flag == -1 表示该线在右边
83             y[num++] = y2;
84         }
85         sort(line+1,line+num,cmp); //按照横坐标进行排序
86         //对纵坐标进行离散化
87         sort(y+1,y+num);
88         int cm = unique(y+1,y+num)-y-1;
89         //在y轴上建立线段树
90         build(1,1,cm);
91         change(1,line[1]);
92         double ans = 0;
```

```
93          for(int i = 2;i < num;i++)
94          {
95              ans += t[1].len*(line[i].x-line[i-1].x);
96              change(1,line[i]);
97          }
98          printf("Test case #%d\n",cnt++);
99          printf("Total explored area: %.2f\n\n",ans);
100     }
101     return 0;
102 }
```

4.8.3 周长并

题意:
给一堆矩阵，求出所有矩阵拼起来，求出矩阵并起来的总周长。

思路:
其实与面积并差不多，就是求 ans 的时候，ans += abs(last-t[1].len)，每次插入一根线段，对答案的贡献值为使得 t[1].len 增加或减少的长度。

```
1  #include <cstdio>
2  #include <iostream>
3  #include <cstring>
4  #include <algorithm>
5  #include <cmath>
6  #define rep(i,a,b) for(int i = a; i <= b; i++)
7  using namespace std;
8  const int N = 10000+100;
9
10 int n;
11 struct Line{
12     int x,y1,y2;
13     int flag;
14 }s1[N],s2[N];
15
16 bool cmp(Line a,Line b)
17 {
18     return a.x < b.x;
19 }
20
21 struct Tree{
22     int l,r,s;
23     int ml,mr,len;
24 }t[N*4];
25
26 int y[4][N],ans;
27 int num1,num2;
28
29 void build(int p,int l,int r,int idx)
30 {
31     t[p].l = l, t[p].r = r, t[p].s = 0, t[p].ml = y[idx][l], t[p].mr = y[idx][r];
32     if(l == (r-1)) return;
33     int mid = (l+r)>>1;
34     build(p*2,l,mid,idx);
35     build(p*2+1,mid,r,idx);
36 }
37
```

```
38 void calc(int p)
39 {
40     if(t[p].s >= 1) t[p].len = t[p].mr-t[p].ml;
41     else if(t[p].l == (t[p].r-1)) t[p].len = 0;
42     else{
43         t[p].len = t[p*2].len+t[p*2+1].len;
44     }
45 }
46
47 void update(int p, Line tp)
48 {
49     if(tp.y1 <= t[p].ml && t[p].mr <= tp.y2)
50     {
51         t[p].s += tp.flag;
52         calc(p);
53         return;
54     }
55     if(t[p*2].mr >= tp.y2) update(p*2,tp);
56     else if(t[p*2+1].ml <= tp.y1) update(p*2+1,tp);
57     else{
58         update(p*2,tp);
59         update(p*2+1,tp);
60     }
61     calc(p);
62 }
63
64 int main()
65 {
66     while(~scanf("%d",&n))
67     {
68         num1 = 0, num2 = 0, ans = 0;
69         rep(i,1,n)
70         {
71             int x1,y1,x2,y2;
72             scanf("%d%d%d%d",&x1,&y1,&x2,&y2);
73             s1[++num1].x = x1, s1[num1].y1 = y1, s1[num1].y2 = y2, y[1][num1] = y1, s1[
    num1].flag = 1;
74             s1[++num1].x = x2, s1[num1].y1 = y1, s1[num1].y2 = y2, y[1][num1] = y2, s1[
    num1].flag = -1;
75
76             s2[++num2].x = y1, s2[num2].y1 = x1, s2[num2].y2 = x2, y[2][num2] = x1, s2[
    num2].flag = 1;
77             s2[++num2].x = y2, s2[num2].y1 = x1, s2[num2].y2 = x2, y[2][num2] = x2, s2[
    num2].flag = -1;
78         }
79         sort(s1+1,s1+1+num1,cmp);
80         sort(s2+1,s2+1+num2,cmp);
81         sort(y[1]+1,y[1]+1+num1);
82         sort(y[2]+1,y[2]+1+num2);
83
84         int scr1 = unique(y[1]+1,y[1]+1+num1)-y[1]-1;
85         int scr2 = unique(y[2]+1,y[2]+1+num2)-y[2]-1;
86
87         build(1,1,scr1,1);
88         rep(i,1,num1)
89         {
90             int last = t[1].len;
91             update(1,s1[i]);
92             ans += abs(last-t[1].len);
```

```
93          }
94
95          build(1,1,scr2,2);
96          rep(i,1,num2)
97          {
98              int last = t[1].len;
99              update(1,s2[i]);
100             ans += abs(last-t[1].len);
101         }
102
103         printf("%d\n",ans);
104     }
105     return 0;
106 }
```

4.8.4 包星星问题

题意：
给出一大堆星星的坐标，给出每个星星的亮度。然后给出一个矩形，要求用这个矩形包住的星星的亮度最大。注意：如果星星在矩形边界上，则不计算这个星星的亮度。

思路：
我们来思考一下，一个矩形的位置是不是由这个矩形右上角这个点所决定的，所以我们可以把考虑矩形的位置改为考虑右上角这个点所在的位置。

然后我们可以发现，对于一颗星星，(x,y)，只要右上角这个点在（x+0.1，y+0.1）~（x+w-0.1，y+h-0.1）这个范围内，即可包住这颗星星。此处取 0.1 的原因是星星不能在矩形边界上。

因此一个星星就可以确定一个矩形，那么本题就变成了给出一大堆矩形，每个矩形都有一个权值，问其中哪一个区域矩形值之和最大。

因此我们可以将每个矩形的左右边界抽离出来，然后就变成了区间覆盖问题。

询问在线段树维护下的这根扫描线上亮度最大的值是多少，所以线段树上只需要维护一个最大值，再加上一个 lazy 标记，然后边插入边，边更新 ans，就可以通过此题。

```
1  #include <cstdio>
2  #include <iostream>
3  #include <cstring>
4  #include <algorithm>
5  #define rep(i,a,b) for(int i = a; i <= b; i++)
6  using namespace std;
7  const int N = 100000;
8
9  struct Line{
10     double x,y1,y2;
11     int flag;
12 }line[N];
13
14 bool cmp(Line a,Line b)
15 {
16     return a.x < b.x;
17 }
18
19 int n,w,h,num,ans;
20 double y[N];
21
```

```
22 struct Tree{
23     int l,r,lazy;
24     double ml,mr;
25     int maxn;
26 }t[N*4];
27 
28 void build(int p,int l,int r)
29 {
30     t[p].l = l, t[p].r = r, t[p].ml = y[l], t[p].mr = y[r], t[p].maxn = 0, t[p].lazy =
       0;
31     if(l == r) return;
32     int mid = (l+r)>>1;
33     build(p*2,l,mid);
34     build(p*2+1,mid+1,r);
35 }
36 
37 void pushup(int p)
38 {
39     if(t[p].lazy != 0)
40     {
41         t[p*2].lazy += t[p].lazy;
42         t[p*2+1].lazy += t[p].lazy;
43         t[p*2].maxn += t[p].lazy;
44         t[p*2+1].maxn += t[p].lazy;
45         t[p].lazy = 0;
46     }
47 }
48 
49 void change(int p, Line a)
50 {
51 //  cout << a.y1 << " " << a.y2 << endl;
52     if(a.y1 <= t[p].ml && t[p].mr <= a.y2)
53     {
54     //  t[p].s += a.flag;
55         t[p].lazy += a.flag;
56         t[p].maxn += a.flag;
57         return;
58     }
59     pushup(p);
60     if(t[p*2].mr >= a.y2) change(p*2,a);
61     else if(t[p*2+1].ml <= a.y1) change(p*2+1,a);
62     else{
63         change(p*2,a);
64         change(p*2+1,a);
65     }
66     t[p].maxn = max(t[p*2].maxn,t[p*2+1].maxn);
67 }
68 
69 int main()
70 {
71     while(~scanf("%d%d%d",&n,&w,&h))
72     {
73         ans = 0, num = 0;
74         rep(i,1,n)
75         {
76             double x1,y1,z1;
77             scanf("%lf%lf%lf",&x1,&y1,&z1);
78             line[++num].x = x1+0.1, line[num].y1 = y1+0.1, line[num].y2 = y1+h-0.1, y[
       num] = y1+0.1, line[num].flag = z1;
```

```
            line[++num].x = x1+w-0.1, line[num].y1 = y1+0.1, line[num].y2 = y1+h-0.1, y
    [num] = y1+h-0.1, line[num].flag = -z1;
        }
        sort(line+1,line+1+num,cmp);
        sort(y+1,y+1+num);
        int scr = unique(y+1,y+1+num)-y-1;
        build(1,1,scr);

        rep(i,1,num)
        {
            change(1,line[i]);
            ans = max(ans,t[1].maxn);
        }
        printf("%d\n",ans);
    }
    return 0;
}
```

4.8.5 覆盖奇数次的面积

题意:
给出 n 个矩形，求被覆盖区域为奇数次的总面积。

思路:
扫描线有很多种写法，可以打 lazy 更新到底，也可以不打 lazy，只是单纯对目标边进行更新，然后再区间合并上去。

本题问的是被覆盖区域为奇数次的总面积。因此线段树每个节点记录被覆盖的次数，被覆盖奇数次的长度，被覆盖偶数次的长度。每次加入一条边，只对被覆盖的那个最大的区间，覆盖次数 +1，对于该区间下面的区间不再进行更新，之后也不会更新。

因此假如第一次加入的线段是 (1,4)，第二次加入的线段是 (1,2)，因此第一次 (1,4) 区间覆盖次数变为 1，第二次 (1,2) 区间覆盖次数变为 1。然后区间合并的时候，len1 表示覆盖奇数次的长度，len2 表示覆盖偶数次的长度，(1,2) 区间 len1 = 1，len2 = 0. (1,4) 区间覆盖次数为奇数，因此 len1 = (1,2) 与 (3,4) 区间被覆盖偶数次的长度，因为偶 + 奇 = 奇，而 len2 = (1,2) 与 (3,4) 区间被覆盖奇数次的长度。因此可以发现，虽然在加入直线的时候，没有将更新次数一次性更新到底，但是在区间合并的时候，会将之前覆盖的长度一并算入。

由于每次插入直线的时候，最后都会合并到整根扫描线上，因此询问整根扫描线的奇偶长度得到的答案是正确的。但是如果询问某一个区间被覆盖奇数次的长度则会得到错误答案，因为这个区间被覆盖的次数还取决于这个区间之上的区间。

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <algorithm>
#define rep(i,a,b) for(int i = a; i <= b; i++)
using namespace std;
const int N = 2*1e5+1000;
typedef long long ll;

struct Line{
    ll x,y1,y2;
    int flag;
}line[N];
int n, num;
ll y[N], ans;
struct Tree{
    int l,r,s;
```

```
18      ll ml,mr,len1,len2; //记录覆盖偶数次的长度 和 覆盖奇数次的长度
19  }t[N*4];
20
21  bool cmp(Line a,Line b)
22  {
23      return a.x < b.x;
24  }
25
26  void build(int p,int l,int r)
27  {
28      t[p].l = l, t[p].r = r, t[p].ml = y[l], t[p].mr = y[r], t[p].s = 0, t[p].len1 = y[r
        ]-y[l], t[p].len2 = 0;
29  //  cout << t[p].ml << " " << t[p].mr << endl;
30      if(l == (r-1)) return;
31      int mid = (l+r)>>1;
32      build(p*2,l,mid);
33      build(p*2+1,mid,r);
34  }
35
36  void calc(int p)
37  {
38      //len1: 覆盖偶数次, len2: 覆盖奇数次
39      if(t[p].s % 2){ //覆盖奇数次
40          if(t[p].r == (t[p].l+1)) t[p].len2 = t[p].mr-t[p].ml, t[p].len1 = 0;
41          else{
42              t[p].len1 = t[p*2].len2+t[p*2+1].len2;
43              t[p].len2 = t[p*2].len1+t[p*2+1].len1;
44              // t[p].len2 = t[p].mr-t[p].ml-t[p*2].len2-t[p*2+1].len2;
45          }
46      }
47      else if(t[p].s % 2 == 0){   //覆盖偶数次
48          if(t[p].r == (t[p].l+1)) t[p].len1 = t[p].mr-t[p].ml, t[p].len2 = 0;
49          else{
50              // t[p].len1 = t[p].mr-t[p].ml-t[p*2].len2-t[p*2+1].len2;
51              t[p].len1 = t[p*2].len1+t[p*2+1].len1;
52              t[p].len2 = t[p*2].len2+t[p*2+1].len2;
53          }
54      }
55  }
56
57  void change(int p, Line tp)
58  {
59      if(tp.y1 <= t[p].ml && tp.y2 >= t[p].mr)
60      {
61          t[p].s += tp.flag;
62          calc(p);
63          return;
64      }
65      if(t[p*2].mr >= tp.y2) change(p*2,tp);
66      else if(t[p*2+1].ml <= tp.y1) change(p*2+1,tp);
67      else{
68          change(p*2,tp);
69          change(p*2+1,tp);
70      }
71      calc(p);
72  }
73
74  int main()
75  {
```

```
76      ans = 0, num = 0;
77      scanf("%d",&n);
78      rep(i,1,n){
79          ll x1,x2,y1,y2;
80          scanf("%lld%lld%lld%lld",&x1,&y1,&x2,&y2);
81          line[++num].x = x1, line[num].y1 = y1, line[num].y2 = y2, line[num].flag = 1;
82          y[num] = y1;
83          line[++num].x = x2, line[num].y1 = y1, line[num].y2 = y2, line[num].flag = -1;

84          y[num] = y2;
85      }
86      sort(line+1,line+1+num,cmp);
87      sort(y+1,y+1+num);
88      int sc = unique(y+1,y+num+1)-y-1;
89      build(1,1,sc);
90      change(1,line[1]);
91      rep(i,2,num)
92      {
93          // printf("%f\n",t[1].len2);
94          ans += t[1].len2*(line[i].x-line[i-1].x);
95          change(1,line[i]);
96      }
97      cout << ans << endl;
98      return 0;
99  }
```

4.9 主席树

4.9.1 静态主席树

新建多个权值线段树副本，记录只考虑 1-i 个数时，每个数出现在各个区间的个数是多少，类似于建多棵权值线段树然后第 i 棵线段树，参考第 i-1 棵线段树，优化空间空间为 4*n*log(n)

```
1   #include <cstdio>
2   #include <iostream>
3   #include <cstring>
4   #include <algorithm>
5   #define rep(i,a,b) for(int i = a; i <= b; i++)
6   using namespace std;
7   const int N = 1e5+100;
8
9   struct Tree{
10      int lc,rc; //左右节点 t数组 编号
11      int l,r; //节点的左右端点
12      int sum;
13  }t[N*20];
14  int n,m,a[N],num,b[N],root[N],tot;
15
16  int build(int l,int r)
17  {
18      int p = ++tot; // 新建一个节点，编号为p，代表当前区间[l,r]
19      t[p].l = l, t[p].r = r, t[p].sum = 0;
20      if(l == r) return p;
21      int mid = (l+r)>>1;
22      t[p].lc = build(l,mid);
23      t[p].rc = build(mid+1,r);
24      return p;
25  }
26
```

```
27  int insert(int now,int pos,int k)
28  {
29      int p = ++tot;
30      t[p] = t[now]; //建立副本
31      if(t[p].l == t[p].r){
32          t[p].sum += k; //在副本上修改
33          return p;
34      }
35      int mid = (t[p].l+t[p].r)>>1;
36      if(pos <= mid) t[p].lc = insert(t[p].lc,pos,k); //保留右儿子部分，把左儿子更新
37      else t[p].rc = insert(t[p].rc,pos,k);
38      t[p].sum = t[t[p].lc].sum + t[t[p].rc].sum;
39      return p;
40  }
41
42  int ask(int lp,int rp,int k) //lp和rp所代表的区间是相同的，他们只不过是在不同状态下的副本
43  {
44      if(t[lp].l == t[lp].r) return t[lp].l; //找到答案
45      int cnt = t[t[rp].lc].sum-t[t[lp].lc].sum; // 值在[l,mid]中的数有多少个
46      if(cnt >= k) return ask(t[lp].lc,t[rp].lc,k);
47      else return ask(t[lp].rc,t[rp].rc,k-cnt);
48  }
49
50  int main()
51  {
52      num = tot = 0;
53      scanf("%d%d",&n,&m);
54      rep(i,1,n){
55          scanf("%d",&a[i]);
56          b[++num] = a[i];
57      }
58      sort(b+1,b+1+num); //离散化
59      num = unique(b+1,b+1+num)-b-1;
60      root[0] = build(1,num); //root[0]这颗树是一棵空树，关于离散化后的值域建树
61      rep(i,1,n)
62      {
63          int x = lower_bound(b+1,b+1+num,a[i])-b; //离散化后的值
64          root[i] = insert(root[i-1],x,1); //值为x的数增加1个
65      }
66      rep(i,1,m)
67      {
68          int x,y,z;
69          scanf("%d%d%d",&x,&y,&z);
70          int ans = ask(root[x-1],root[y],z);
71          printf("%d\n",b[ans]); //从离散化后的值变回原值
72      }
73      return 0;
74      //root[i]:表示只考虑1-i这些数时候建树的情况，这颗树树根的编号
75  }
```

4.9.2 区间中不同数的个数

题意：
长度为 n 的序列，q 次询问，每次给出 l、r，返回序列 $[l,r]$ 中不同数的个数。$(1 \le n \le 3*10^4, 1 \le q \le 2*10^5)$

思路：
与之前主席树的权值线段树思路不同，此题的思路是建立 n 颗线段树，第 i 颗线段树存储区间 $[1,i]$ 的信息。其中每个节点维护 sum，表示节点对应区间中数的个数，因此每棵线段树中只保留每个数最后出现的位置。

举个例子，序列为 5 5 5 5 5，则第 1 颗线段树中只有第一个位置 sum 为 1，然后第二颗线段树从第一颗线段树继承过来，由于 5 这个数字之前出现过，因此在第二颗线段树中令第一个位置的 sum 为 0，令第二个位置的 sum 为 1，来保存每个数字最后出现的位置。

因此查询区间 $[l,r]$ 时，就在第 r 颗线段树中查询区间 $[l,n]$ 中数的个数即可。

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <algorithm>
#define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
#define rep(i,a,b) for(int i = a; i <= b; i++)
#define LOG1(x1,x2) cout << x1 << ": " << x2 << endl;
#define LOG2(x1,x2,y1,y2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << endl;
#define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << " , " << z1 << ": " << z2 << endl;
typedef long long ll;
typedef double db;
const int N = 3*1e4+100;
const int M = 1e6+100;
const db EPS = 1e-9;
using namespace std;

int n,a[N],q;
int ls[40*N],rs[40*N],sum[40*N],root[N],sz;
int pos[M]; //记录每个数的位置

void build(int &now,int l,int r){
    if(!now) now = ++sz;
    if(l == r){
        sum[now] = 0; return;
    }
    build(ls[now],l,(l+r)>>1);
    build(rs[now],((l+r)>>1)+1,r);
    sum[now] = sum[ls[now]]+sum[rs[now]];
}

void update(int x,int &now,int l,int r,int p1,int ct){
    if(!now) now = ++sz;
    if(l == r){
        sum[now] = sum[x]+ct; return;
    }
    int mid = (l+r)>>1;
    if(p1 <= mid){
        if(!rs[now]) rs[now] = rs[x]; //继承前一个备份的节点
        if(ls[now] <= now) ls[now] = 0; //因为现在要修改ls[now], 因此要重新开空间
        //如果ls[now] <= now, 表示这个节点是之前继承的, 因此要赋为0, 重新开空间
        update(ls[x],ls[now],l,mid,p1,ct);
    }
    else{
        if(!ls[now]) ls[now] = ls[x];
        if(rs[now] <= now) rs[now] = 0;
        update(rs[x],rs[now],mid+1,r,p1,ct);
    }
    sum[now] = sum[ls[now]]+sum[rs[now]];
}

```

```
51 int ask(int &now,int l,int r,int p1){
52     if(!now) now = ++sz;
53     if(l >= p1) return sum[now];
54     else if(r < p1) return 0;
55     int mid = (l+r)>>1, ans = 0;
56     if(mid >= p1) ans += ask(ls[now],l,mid,p1)+sum[rs[now]];
57     else ans += ask(rs[now],mid+1,r,p1);
58     return ans;
59 }
60
61 int main()
62 {
63     scanf("%d",&n);
64     rep(i,1,n) scanf("%d",&a[i]);
65     build(root[0],1,n);
66     rep(i,1,n){
67         if(pos[a[i]]){
68             update(root[i-1],root[i],1,n,pos[a[i]],-1);
69             update(root[i-1],root[i],1,n,i,1);
70             pos[a[i]] = i;
71         }
72         else update(root[i-1],root[i],1,n,i,1), pos[a[i]] = i;
73     }
74     scanf("%d",&q);
75     rep(i,1,q){
76         int xx,yy; scanf("%d%d",&xx,&yy);
77         printf("%d\n",ask(root[yy],1,n,xx));
78     }
79     return 0;
80 }
```

4.9.3 单点修改主席树

题意：

给定一个区间，求这个区间第 k 小的数，支持单点修改。

思路：

动态主席树裸题。

我们先来回顾一下静态主席树的做法，对于数组中每一个位置都维护一棵权值线段树，该权值线段树保存的是区间 [1,x] 的信息。因此我想要求区间 [l,r] 之间第 k 大的时候，只需要将 root[r]-root[l-1] 就是维护区间 [l,r] 信息的权值线段树，因此就可以快速直接求出这个区间中第 k 大的元素是多少。

现在我们来看看单点修改的操作。

如果我现在要将 a[pos] 修改为 x，那么最暴力的做法就是对于 root[pos] ~ root[n] 中的每一颗权值线段树都进行修改，即将 a[pos] 这个点的值减 1，将 x 这个点的值 +1。

最暴力的做法显然是无法通过此题的，因此我们可以想到有没有一种 logn 的方法，可以只修改 logn 个节点，就可以对于每一个线段树记录修改信息，于是我们想到了树状数组。

我们来回忆一下树状数组，每一个节点记录区间 [x-lowbit(x)+1, x] 的所有信息，因此当需要求 [1,x] 内维护的信息的时候，只需要从节点 x 出发，每次进行 x-=lowbit(x) 的操作，即可求出 [1,x] 内维护的所有信息。

每次对 x 节点进行修改的时候，只需要不断进行 x+=lowbit(x) 的操作，就可以访问到所有存储 x 节点信息的节点，因此实现了 logn 的查询。

因此本题也维护一个树状数组。

节点 x 维护的是区间 [x-lowbit(x)+1, x] 的权值线段树，因此当我需要访问 [l, r] 区间信息的时候，只需要将 root[r]-root[l-1]+getsum(r)-getsum(l-1) 这里面维护的便是区间 [l, r] 的权值线段树。

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <algorithm>
#define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
#define rep(i,a,b) for(int i = a; i <= b; i++)
#define LOG1(x1,x2) cout << x1 << ": " << x2 << endl;
#define LOG2(x1,x2,y1,y2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << endl;
#define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << " , " << z1 << ": " << z2 << endl;
typedef long long ll;
typedef double db;
const int N = 6*1e4+100;
const int M = 32*N+100;
const db EPS = 1e-9;
using namespace std;

int rt[N],root[N],ls[M],rs[M],sz,n,m,tot,h1,h2,L[N],R[N],sum[M],a[N],b[N];
struct Node{
    int op,xx,yy,k; //保存修改信息
}t[N];

int find(ll x) {return (lower_bound(b+1,b+1+tot,x)-b);}
int lowbit(int x) {return x&(-x);}

void update(int pre,int &now,int l,int r,int pos,int c){
    if(!now) now = ++sz;
    sum[now] = sum[pre]+c;
    if(l == r) return;
    int mid = (l+r)>>1;
    if(pos <= mid){rs[now] = rs[pre]; update(ls[pre],ls[now],l,mid,pos,c);}
    else{ls[now] = ls[pre]; update(rs[pre],rs[now],mid+1,r,pos,c);}
}

int query(int pre,int now,int l,int r,int k){
    int sum1 = 0, sum2 = 0;
    if(l == r) return l;
    rep(i,1,h1) sum1 += sum[ls[L[i]]]; //左边树状数组节点累加值
    rep(i,1,h2) sum2 += sum[ls[R[i]]]; //右边树状数组节点累加值
    int temp = sum[ls[now]]-sum[ls[pre]]+sum2-sum1;
    int mid = (l+r)>>1;
    if(temp >= k){
        rep(i,1,h1) L[i] = ls[L[i]];
        rep(i,1,h2) R[i] = ls[R[i]];
        return query(ls[pre],ls[now],l,mid,k);
    }
    else{
        rep(i,1,h1) L[i] = rs[L[i]];
        rep(i,1,h2) R[i] = rs[R[i]];
        return query(rs[pre],rs[now],mid+1,r,k-temp);
    }
}

void init(){
```

```
54      scanf("%d%d",&n,&m);
55      rep(i,1,n){
56          scanf("%d",&a[i]); b[++tot] = a[i];
57      }
58      rep(i,1,m){
59          char op[10]; scanf("%s",op);
60          if(op[0] == 'Q'){
61              t[i].op = 1; scanf("%d%d%d",&t[i].xx,&t[i].yy,&t[i].k);
62          }
63          else{
64              t[i].op = 2; scanf("%d%d",&t[i].xx,&t[i].yy);
65              b[++tot] = t[i].yy;
66          }
67      }
68      sort(b+1,b+1+tot);
69      tot = unique(b+1,b+1+tot)-b-1;
70      rep(i,1,n)
71          update(rt[i-1],rt[i],1,tot,find(a[i]),1);
72  }
73
74  void solve(){
75      rep(i,1,m){
76          if(t[i].op == 1){ //查询
77              h1 = h2 = 0;
78              for(int j = t[i].xx-1; j; j -= lowbit(j)) L[++h1] = root[j]; //记录树状数组要
        计算的节点
79              for(int j = t[i].yy; j; j -= lowbit(j)) R[++h2] = root[j];
80              int pos = query(rt[t[i].xx-1],rt[t[i].yy],1,tot,t[i].k);
81              printf("%d\n",b[pos]);
82          }
83          else{ //修改
84              int pos1 = find(a[t[i].xx]), pos2 = find(t[i].yy);
85              for(int j = t[i].xx; j <= n; j += lowbit(j)) update(root[j],root[j],1,tot,
        pos1,-1);
86              for(int j = t[i].xx; j <= n; j += lowbit(j)) update(root[j],root[j],1,tot,
        pos2,1);
87              a[t[i].xx] = t[i].yy;
88          }
89      }
90  }
91
92  int main()
93  {
94      int _; scanf("%d",&_);
95      while(_--){
96          init();
97          solve();
98          rep(i,0,sz) ls[i] = rs[i] = sum[i] = 0;
99          rep(i,0,n) root[i] = rt[i] = 0;
100         sz = 0, tot = 0;
101     }
102     return 0;
103 }
```

4.10 启发式合并

题意：
给定一棵 n 个点的树，以及一张 k 个点的图。树中每一个节点都控制图中的一条边，问：对于树中每一个节点，将其子

树 (包括自己) 中所有节点控制的边加到图中，图中连通块个数即为这个节点的答案。

思路：
树上启发式合并裸题，但是比赛的时候没有写过启发式合并以为这样写会 T... 就没有尝试...

先具体讲一下这题的解法，再分析一下复杂度。'用并查集维护图中的连通关系，对于当前节点，最后求其重儿子子树所形成的并查集，并将重儿子的并查集直接复用在求父节点上。对于非重儿子节点再对其子树中的边进行添加。

非常暴力的做法... 但是为什么能够快速通过呢... 我们先考虑一下序列上的启发式合并，由于每次选择将小的合并进大的，所以每个元素的大小每次至少 $*2$，因此最多被合并 log 次就可以达到 n 的大小，所以复杂度是 $nlogn$。

而对于树上的启发式合并，每次将小的子树合并进大的子树中，因此子树的大小每次至少 $*2$，因此最多 $logn$ 次子树的大小就可以达到 n，因此复杂度也是 $nlogn$。

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <vector>
#include <algorithm>
#define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
#define rep(i,a,b) for(int i = a; i <= b; i++)
#define LOG1(x1,x2) cout << x1 << ": " << x2 << endl;
#define LOG2(x1,x2,y1,y2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << endl
    ;
#define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2
    << " , " << z1 << ": " << z2 << endl;
typedef long long ll;
typedef double db;
const int N = 1e5+100;
const int M = 1e5+100;
const db EPS = 1e-9;
using namespace std;

int sz[N],n,k,l[N],r[N],ans[N],f[N],num;
vector<int> p[N];

int find(int x){
    return x == f[x] ? x : (f[x] = find(f[x]));
}

void clear(int x){
    f[l[x]] = l[x], f[r[x]] = r[x];
    for(int u : p[x]) clear(u);
}

void unite(int x,int y){
    int fx = find(x), fy = find(y);
    if(fx != fy) f[fx] = fy, num--;
}

void solve(int x){
    unite(l[x],r[x]);
    for(int u : p[x]) solve(u);
}

void dfs(int x){
    int y = -1;
    for(int u : p[x])
        if(y == -1 || sz[y] < sz[u]) y = u;
```

```
    for(int u : p[x]){
        if(u == y) continue;
        num = k, dfs(u), clear(u);
    }
    num = k;
    if(y != -1) dfs(y);
    for(int u : p[x]){
        if(u != y) solve(u);
    }
    unite(l[x],r[x]);
    ans[x] = num;
}

int main()
{
    scanf("%d%d",&n,&k);
    rep(i,2,n){
        int pp; scanf("%d",&pp);
        p[pp].push_back(i);
    }
    rep(i,1,n) scanf("%d%d",&l[i],&r[i]);
    for(int i = n; i >= 1; i--){
        for(int y : p[i]) sz[i] += sz[y];
        sz[i]++;
    }
    rep(i,0,k) f[i] = i;
    dfs(1);
    rep(i,1,n) printf("%d\n",ans[i]);
    return 0;
}
```

4.11 Splay

4.11.1 普通 Splay

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <algorithm>
#define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
#define rep(i,a,b) for(int i = a; i <= b; i++)
#define LOG1(x1,x2) cout << x1 << ": " << x2 << endl;
#define LOG2(x1,x2,y1,y2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << endl
    ;
#define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2
    << " , " << z1 << ": " << z2 << endl;
typedef long long ll;
typedef double db;
const int N = 1e6+100;
const int M = 1e5+100;
const db EPS = 1e-9;
using namespace std;

int f[N],cnt[N],ch[N][2],siz[N],key[N],sz,rt;
//f[x]-x的父节点, ch[x][0]表示x的左儿子, ch[x][1]表示x的右儿子, key[x]表示x的关键点,
//cnt[x]表示x节点关键字出现的次数(权值), siz[x]表示包括x的这个子树的大小, sz为整棵树的大小, rt为整
    棵树的根
//空间大小为所有插入点个数, 点被删除之后空间不能复用

```

```
22 void init(){
23     memset(f,0,sizeof f); memset(cnt,0,sizeof cnt);
24     memset(ch,0,sizeof ch); memset(siz,0,sizeof siz);
25     memset(key,0,sizeof key); sz = rt = 0;
26 
27 }
28 void clear(int x) { f[x] = cnt[x] = ch[x][0] = ch[x][1] = siz[x] = key[x] = 0; }
29 bool get(int x) { return ch[f[x]][1] == x;}
30 void push_up(int x) { siz[x] = siz[ch[x][0]]+siz[ch[x][1]]+cnt[x]; } //更新节点信息，此处
       为siz
31 void rotate(int x){
32     int old = f[x], oldf = f[old], which = get(x);
33     ch[old][which] = ch[x][which^1]; f[ch[old][which]] = old; //把儿子过继给爸爸，同时处理
       父子两个方向上的信息
34     ch[x][which^1] = old; f[old] = x; //我给我爸爸当爹，我爸爸管我叫爸爸
35     f[x] = oldf; //我的爷爷成了我的爸爸
36     if(oldf) ch[oldf][ch[oldf][1]==old] = x;
37     push_up(old); push_up(x); //分别维护信息
38 }
39 void splay(int x){ //将x旋为根
40     for(int fa; (fa = f[x]); rotate(x))
41         if(f[fa]) rotate((get(x) == get(fa))?fa:x); //如果祖父三代连成一线，则旋转父亲，否则
       旋转自己
42     rt = x;
43 }
44 
45 void insert(int x){ //x为权值
46     if(rt == 0) {
47         key[++sz] = x, rt = sz;
48         cnt[sz] = siz[sz] = 1; f[sz] = ch[sz][0] = ch[sz][1] = 0;
49         return;
50     } //空树
51     int now = rt, fa = 0;
52     while(1){
53         if(x == key[now]){ //这个数出现过
54             cnt[now]++, push_up(now), push_up(fa), splay(now); return;
55         }
56         fa = now, now = ch[now][key[now]<x];
57         if(now == 0){
58             ++sz; siz[sz] = cnt[sz] = 1;
59             ch[sz][0] = ch[sz][1] = 0;
60             ch[fa][x>key[fa]] = sz; //根据加入点顺序重新编号
61             f[sz] = fa, key[sz] = x, push_up(fa), splay(sz); return;
62         }
63     }
64 }
65 
66 int rnk(int x){ //查询x的排名
67     int now = rt, ans = 0;
68     while(now){
69         if(x < key[now]) now = ch[now][0];
70         else{
71             ans += siz[ch[now][0]];
72             if(x == key[now]) { splay(now); return (ans+1);} //x在树中节点的位置
73             ans += cnt[now], now = ch[now][1]; //到达右孩子处
74         }
75     }
76     return -1; //找不到
77 }
```

```
78  int kth(int x){ //查询排名为x的数
79      int now = rt;
80      while(1){
81          if(ch[now][0] && x <= siz[ch[now][0]]) now = ch[now][0];
82          else{
83              int tmp = siz[ch[now][0]]+cnt[now];
84              if(x <= tmp) return key[now];
85              x -= tmp, now = ch[now][1];
86          }
87      }
88  }
89  int pre(){ //进行splay后, x已经在根节点了, 因此只用找左子树最大节点即可, 返回节点编号
90      int now = ch[rt][0];
91      while(ch[now][1]) now = ch[now][1];
92      return now;
93  }
94  int next(){ //找右子树最小
95      int now = ch[rt][1];
96      while(ch[now][0]) now = ch[now][0];
97      return now;
98  }
99
100 void del(int x){
101     rnk(x); //把x对应节点转到了根
102     if(cnt[rt] > 1) {cnt[rt]--; push_up(rt); return;} //有多个相同的数
103     if(!ch[rt][0] && !ch[rt][1]) {clear(rt); rt = 0; return;}
104     if(!ch[rt][0] || !ch[rt][1]){ //只有一个儿子
105         int oldrt = rt; rt = ch[rt][1]+ch[rt][0]; f[rt] = 0; clear(oldrt); return;
106     }
107     int oldrt = rt, leftbig = pre();
108     splay(leftbig);
109     ch[rt][1] = ch[oldrt][1];
110     f[ch[oldrt][1]] = rt;
111     clear(oldrt);
112     push_up(rt);
113 }
114
115 int main()
116 {
117     int m; scanf("%d",&m);
118     rep(i,1,m){
119         // LOG1("i",i);
120         int op,x; scanf("%d%d",&op,&x);
121         if(op == 1) insert(x);
122         else if(op == 2) del(x);
123         else if(op == 3) printf("%d\n",rnk(x));
124         else if(op == 4) printf("%d\n",kth(x));
125         else if(op == 5){
126             insert(x);
127             printf("%d\n",key[pre()]);
128             del(x);
129         }
130         else if(op == 6){
131             insert(x);
132             printf("%d\n",key[next()]);
133             del(x);
134         }
135     }
136     return 0;
```

```
}
```

4.11.2 区间 Splay

```
#include<iostream>
#include<cstdio>
using namespace std;
struct{int l,r,size,dat,bit,add,rev;}a[200010];
int L[200010],R[200010],n,m,tot,root,i,x,y,z;
char str[10];
//size-子树大小, dat-节点真实值, bit-子树中最小值, add-子树区间加标记, rev-子树翻转标记

void spreadadd(int x,int y)
{
    if(!x||!y) return;
    a[x].add+=y; a[x].dat+=y; a[x].bit+=y;
}
//add是lazy标记

void spreadrev(int x)
{
    if(!a[x].rev) return;
    swap(a[x].l,a[x].r);
    if(a[x].l) a[a[x].l].rev^=1; //翻转标记下传
    if(a[x].r) a[a[x].r].rev^=1;
}

void spread(int x)
{
    spreadadd(a[x].l,a[x].add); //标记下传
    spreadadd(a[x].r,a[x].add);
    spreadrev(x); //rev表示是否要翻转
    a[x].add=a[x].rev=0;
}

inline void update(int x) //更换左右儿子后, 更新其siz及bit
{
    a[x].size=a[a[x].l].size+a[a[x].r].size+1;
    a[x].bit=min(a[x].dat,min(a[a[x].l].bit,a[a[x].r].bit));
}

void turnleft(int &x) //x左旋, 即x的右儿子左旋到x位置
{
    int y=a[x].r; a[x].r=a[y].l; a[y].l=x;
    update(x); update(y);   x=y;
}

void turnright(int &x) //x右旋, 即x的左儿子右旋到x位置
{
    int y=a[x].l; a[x].l=a[y].r; a[y].r=x;
    update(x); update(y);   x=y;
    //先更新x, 因为x是儿子
}

void splay(int &x,int y) //将x子树中第y小的数转为根
{
    if(!x) return;
    L[0]=R[0]=0; //L存储了y转到x之后, 左子树中的各个节点
```

```
55      //R存储y转到x之后，右子树中的各个节点
56      while(1)
57      {
58          //lazy、旋转标记下传
59          spread(x),spread(a[x].l),spread(a[x].r);
60          int temp=a[a[x].l].size+1; //<=x的数的个数
61          //完成了目标
62          if(y==temp||(y<temp&&!a[x].l)||(y>temp&&!a[x].r)) break;
63          if(y<temp) //让y的左儿子旋转上来
64          {
65              if(a[a[x].l].l&&y<=a[a[a[x].l].l].size) {turnright(x); if(!a[x].l) break;}
66              R[++R[0]]=x; x=a[x].l;
67          }
68          else //让y的右儿子旋转上来
69          {
70              y-=temp;
71              int temp=a[a[a[x].r].l].size+1;
72              if(a[a[x].r].r&&y>temp) {y-=temp; turnleft(x); if(!a[x].r) break;}
73              L[++L[0]]=x; x=a[x].r;
74          }
75      }
76      L[++L[0]]=a[x].l; R[++R[0]]=a[x].r; //此处的x已经成为y所对应的节点
77      //将左右子树一一连接
78      for(int i=L[0]-1;i>0;i--) {a[L[i]].r=L[i+1]; update(L[i]);}
79      for(int i=R[0]-1;i>0;i--) {a[R[i]].l=R[i+1]; update(R[i]);}
80      a[x].l=L[1]; a[x].r=R[1]; update(x);
81  }
82
83  void ADD(int x,int y,int z)
84  {
85      splay(root,y+1); //把后继转到树根
86      splay(a[root].l,x-1); //把前驱转到树根左儿子
87      spreadadd(a[a[root].l].r,z); //区间+，打lazy标记
88  }
89
90  void INSERT(int x,int y) //将y插入到a[x]之后
91  {
92      splay(root,x); //令x为根
93      a[++tot].dat=y; a[tot].r=a[root].r; a[root].r=tot; //令y为x的右儿子，x原右儿子为y右儿子
94      update(tot); update(root);
95  }
96
97  void DELETE(int x) //删除数组中第x个数
98  {
99      splay(root,x); //将数组中第x个数旋转到根
100     splay(a[root].r,1); //将子树中最小的数转到根，则根节点的右儿子没有左儿子
101     a[a[root].r].l=a[root].l; root=a[root].r;
102     update(root);
103 }
104
105 void REVERSE(int x,int y)
106 {
107     splay(root,y+1);
108     splay(a[root].l,x-1);
109     a[a[a[root].l].r].rev^=1; //子树中所有点都左右翻转
110     //此处的操作与add处不同，原因在于区间反转不会导致该区间维护的min发生变化，因此可以这样操作
111     //但是正常打标记的操作应该是与ADD函数一致
112 }
113
```

```
114 void REVOLVE(int x,int y,int z) //将[x,y-len]与[y-len+1,y]互换位置, len为z%(y-x+1)
115 {
116     z%=y-x+1;
117     if(!z)return;
118     int mid=y-z; //找到右区间的第一个数
119     splay(root,mid); //将mid转到根
120     splay(a[root].l,x-1); //根左儿子的右儿子对应区间[x,y-len] (不包括根左儿子的右儿子)
121     splay(a[root].r,y-a[a[root].l].size); //根右儿子的左儿子对应区间[y-len+2,y] (不包括根右儿
        子的左儿子)
122     z=a[root].l;
123     a[root].l=a[z].r;
124     a[z].r=a[a[root].r].l;
125     a[a[root].r].l=0;
126     update(z); update(a[root].r); update(root);
127     splay(root,1); //空出根的左儿子
128     a[root].l=z;
129     update(root);
130 }
131
132 void MIN(int x,int y)
133 {
134     splay(root,y+1); //将根节点中第y+1小的数转为根
135     splay(a[root].l,x-1); //将根节点左儿子子树中第x-1小的数转为根节点左儿子
136     printf("%d\n",a[a[a[root].l].r].bit);
137 }
138
139 // splay(root,n+2);
140 // splay(a[root].l,0);
141 // Print(a[a[root].l].r);
142 void Print(int x){ //打印区间[1,n]的数, 输出之前要先把这个区间的数旋转到一个子树中
143     spread(x); //标记下传
144     if(a[x].l) Print(a[x].l);
145     printf("%d ",a[x].dat); //中序遍历
146     if(a[x].r) Print(a[x].r);
147 }
148
149 int main()
150 {
151     cin>>n;
152     a[1].size=1; a[n+2].l=n+1; a[n+2].size=n+2;
153     a[1].dat=a[n+2].dat=a[1].bit=a[0].bit=0x3fffffff;
154     //所有数右移一格, 2表示1, n+1表示n
155     for(i=2;i<=n+1;i++) //一开始构建出一根链
156     {
157         scanf("%d",&a[i].dat); //dat为真实值
158         a[i].l=i-1; a[i].size=i; //l表示左儿子, siz表示子树大小
159         a[i].bit=min(a[i-1].bit,a[i].dat);  //bit表示最小值
160     }
161     a[n+2].bit=a[n+1].bit;
162     root=tot=n+2; //一共n+2个节点, 加了两个前驱和后续
163     cin>>m;
164     for(i=0;i<m;i++)
165     {
166         scanf("%s",&str);
167         //ADD x y z 区间[x,y]每个数+z
168         if(str[0]=='A') {scanf("%d%d%d",&x,&y,&z); ADD(++x,++y,z);}
169         //INSERT x y 在序列第x个数之后插入y
170         if(str[0]=='I') {scanf("%d%d",&x,&y); INSERT(++x,y);}
171         //DELETE x 删除序列第x个数
```

```
172         if(str[0]=='D') {scanf("%d",&x); DELETE(++x);}
173         //MIN x y 查询区间[x,y]的最小值
174         if(str[0]=='M') {scanf("%d%d",&x,&y);   MIN(++x,++y);}
175         //REVERSE x y 将区间[x,y]中的数翻转，如原序列1 2 3 4 5，REVERSE 3 5，得到1 2 5 4 3
176         if(str[0]=='R'&&str[3]=='E') {scanf("%d%d",&x,&y);  REVERSE(++x,++y);}
177         //REVOLVE x y z 将区间[x,y]旋转z次，即将最后一个数移到第一个位置，如原序列1 2 3 4 5，
        REVOLVE 2 4 2，得到1 3 4 2 5
178         //区间左右旋转可以互相转换
179         if(str[0]=='R'&&str[3]=='O') {scanf("%d%d%d",&x,&y,&z); REVOLVE(++x,++y,z);}

180     }
181     return 0;
182 }
```

4.11.3 Splay (三个 lazy 标记)

Replace a b c, 将 [a,b] 中所有括号改为 c Swap a b, 将 [a,b] 中所有字符翻转 Invert a b, 将 [a,b] 中所有'(' 变为')', ')' 变为'(' Query a b, 询问 [a,b] 之间的字符至少要改变多少位才能变成合法的括号序列

```
1  #include <iostream>
2  #include <cstdio>
3  #define rep(i,a,b) for(int i = a; i <= b; i++)
4  const int N = 1e5+100;
5  using namespace std;
6  struct Node{int l,r,size,dat,add,rev,res,l1,l2,r1,r2,sum;}a[N];
7  int L[N],R[N],n,m,tot,root;
8  char str[10];
9  //size-子树大小, dat-节点真实值, add-子树区间赋值标记, rev-子树翻转标记, res-子树取反标记
10 //l1、l2-最大、小前缀和, r1、r2-最大、小后缀和
11 void spreadadd(int x,int y)
12 {
13     if(!x||!y) return;
14     a[x].add=y; a[x].dat=y;
15     a[x].rev = 0, a[x].res = 0;
16     if(y < 0) a[x].sum = -a[x].size;
17     else a[x].sum = a[x].size;
18     a[x].r1 = a[x].l1 = (y==1?a[x].sum:0);
19     a[x].r2 = a[x].l2 = (y==1?0:a[x].sum);
20 }
21 //add是lazy标记
22
23 void spreadrev(int x) //翻转
24 {
25     if(!x) return;
26     swap(a[x].l1,a[x].r1);
27     swap(a[x].l2,a[x].r2);
28     swap(a[x].l,a[x].r);
29     a[x].rev ^= 1;
30 }
31
32 void spreadres(int x) //取反
33 {
34     if(!x) return;
35     swap(a[x].l1,a[x].l2);
36     swap(a[x].r1,a[x].r2);
37     a[x].sum = -a[x].sum; a[x].dat = -a[x].dat;
38     a[x].l1 = -a[x].l1, a[x].l2 = -a[x].l2;
39     a[x].r1 = -a[x].r1, a[x].r2 = -a[x].r2;
```

```
40      a[x].res ^= 1;
41  }
42
43  void spread(int x)
44  {
45      if(a[x].rev){
46          spreadrev(a[x].l);
47          spreadrev(a[x].r);
48      }
49      if(a[x].add){
50          spreadadd(a[x].l,a[x].add); //标记下传
51          spreadadd(a[x].r,a[x].add);
52      }
53      if(a[x].res){
54          spreadres(a[x].l);
55          spreadres(a[x].r);
56      }
57      a[x].add=a[x].rev=a[x].res=0;
58  }
59
60  inline void update(int x)
61  {
62      int L = a[x].l, R = a[x].r;
63      a[x].size=a[L].size+a[R].size+1;
64      a[x].sum = a[L].sum+a[R].sum+a[x].dat;
65      a[x].l1 = max(a[L].l1,a[L].sum+a[R].l1+a[x].dat);
66      a[x].l2 = min(a[L].l2,a[L].sum+a[R].l2+a[x].dat);
67      a[x].r1 = max(a[R].r1,a[R].sum+a[L].r1+a[x].dat);
68      a[x].r2 = min(a[R].r2,a[R].sum+a[L].r2+a[x].dat);
69  }
70
71  void turnleft(int &x) //x左旋，即x的右儿子左旋到x位置
72  {
73      int y=a[x].r; a[x].r=a[y].l; a[y].l=x;
74      update(x); update(y);   x=y;
75  }
76
77  void turnright(int &x) //x右旋，即x的左儿子右旋到x位置
78  {
79      int y=a[x].l; a[x].l=a[y].r; a[y].r=x;
80      update(x); update(y);   x=y;
81      //先更新x，因为x是儿子
82  }
83
84  void splay(int &x,int y) //将x子树中第y小的数转为根
85  {
86      if(!x) return;
87      L[0]=R[0]=0; //L存储了y转到x之后，左子树中的各个节点
88      //R存储y转到x之后，右子树中的各个节点
89      while(1)
90      {
91          //lazy、旋转标记下传
92          spread(x),spread(a[x].l),spread(a[x].r);
93          int temp=a[a[x].l].size+1; //<=x的数的个数
94          //完成了目标
95          if(y==temp||(y<temp&&!a[x].l)||(y>temp&&!a[x].r)) break;
96          if(y<temp) //让y的左儿子旋转上来
97          {
98              if(a[a[x].l].l&&y<=a[a[a[x].l].l].size) {turnright(x); if(!a[x].l) break;}
```

```
99              R[++R[0]]=x; x=a[x].l;
100         }
101         else //让y的右儿子旋转上来
102         {
103             y-=temp;
104             int temp=a[a[a[x].r].l].size+1;
105             if(a[a[x].r].r&&y>temp) {y-=temp; turnleft(x); if(!a[x].r) break;}
106             L[++L[0]]=x; x=a[x].r;
107         }
108     }
109     L[++L[0]]=a[x].l; R[++R[0]]=a[x].r; //此处的x已经成为y所对应的节点
110     //将左右子树一一连接
111     for(int i=L[0]-1;i>0;i--) {a[L[i]].r=L[i+1]; update(L[i]);}
112     for(int i=R[0]-1;i>0;i--) {a[R[i]].l=R[i+1]; update(R[i]);}
113     a[x].l=L[1]; a[x].r=R[1]; update(x);
114 }
115
116 void ADD(int x,int y,int z)
117 {
118     splay(root,y+1); //把后继转到树根
119     splay(a[root].l,x-1); //把前驱转到树根左儿子
120     spreadadd(a[a[root].l].r,z); //区间+, 打lazy标记
121 }
122
123 void SWAP(int x,int y)
124 {
125     splay(root,y+1);
126     splay(a[root].l,x-1);
127     spreadrev(a[a[root].l].r);
128 }
129
130 void Invert(int x,int y)
131 {
132     splay(root,y+1);
133     splay(a[root].l,x-1);
134     spreadres(a[a[root].l].r);
135 }
136
137 void Query(int x,int y)
138 {
139     splay(root,y+1);
140     splay(a[root].l,x-1);
141     int x1 = a[a[a[root].l].r].l2;
142     int x2 = a[a[a[root].l].r].r1;
143     x1 = -x1;
144     int ans = ((x1+1)/2)+((x2+1)/2); //左端最小前缀和/2上取整+右端最大前缀和上取整
145     //abs(左端最小前缀和)=x,表示左端至少有x个'('未匹配
146     //abs(右端最大前缀和)=x,表示右端至少有x个')'未匹配
147     printf("%d\n",ans);
148 }
149
150 char s[N];
151
152 int main()
153 {
154     cin>>n>>m;
155     a[1].size=1; a[n+2].l=n+1; a[n+2].size=n+2;
156     a[1].dat = a[n+2].dat; update(1);
157     scanf("%s",s+1);
```

```
158     rep(i,2,n+1) a[i].dat = (s[i-1]=='('?1:-1);
159     rep(i,2,n+2)
160     {
161         a[i].l=i-1; a[i].size=i; //l表示左儿子，siz表示子树大小
162         update(i);
163     }
164     root=tot=n+2; //一共n+2个节点，加了两个前驱和后续
165     rep(i,1,m)
166     {
167         int x,y,z; char op[5];
168         scanf("%s",str);
169         if(str[0]=='R'){
170             scanf("%d%d",&x,&y); scanf("%s",op);
171             if(op[0] == '(') z = 1;
172             else z = -1;
173             ADD(++x,++y,z);
174         }
175         else if(str[0]=='S'){
176             scanf("%d%d",&x,&y);
177             SWAP(++x,++y);
178         }
179         else if(str[0] == 'I'){
180             scanf("%d%d",&x,&y);
181             Invert(++x,++y);
182         }
183         else if(str[0] == 'Q'){
184             scanf("%d%d",&x,&y);
185             Query(++x,++y);
186         }
187     }
188     return 0;
189 }
```

4.12 树链剖分

4.12.1 点剖模板

一颗 N 个节点的树，每个节点上都有初始权值。现在有四种操作：操作 1 ——1 x y z，表示将 x 到 y 节点最短路径上所有节点的值加 z 操作 2 ——2 x y，表示求 x 到 y 节点最短路径上所有节点值之和操作 3 ——3 x z，表示将以 x 为根节点的子树内所有节点值加 z 操作 4 ——4 x，表示求以 x 为根节点的子树内所有节点值之和

```
1  #include <cstdio>
2  #include <iostream>
3  #include <cstring>
4  #include <algorithm>
5  #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
6  #define rep(i,a,b) for(int i = a; i <= b; i++)
7  #define LOG1(x1,x2) cout << x1 << ": " << x2 << endl;
8  #define LOG2(x1,x2,y1,y2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << endl
       ;
9  typedef long long ll;
10 typedef double db;
11 #define int long long
12 const db EPS = 1e-9;
13 const int N = 1e5+100;
14 using namespace std;
15
16 struct Edge { int next,to;} e[2*N];
17 struct Node { int l,r,ls,rs,sum,lazy;} t[2*N];
```

```
18 int n,m,root,rt,mod,val[N],head[N],tot,fa[N],d[N],son[N],size[N],top[N],id[N],rk[N];
19 //top[x]: x节点所在链的顶端节点, id[x]: 节点dfs序, rk[x]: dfs序对应的节点
20 //val[x]: 每个点初始权值, fa[x]: 每个点父节点, d[x]: 节点深度, size[x]: 节点子树大小
21 //rt: 线段树根节点编号
22 void init(){
23     memset(head,0,sizeof head);
24     tot = 1, size[0] = 0;
25 }
26
27 void add(int x, int y){
28     e[++tot].next = head[x], e[tot].to = y, head[x] = tot;
29 }
30
31 void dfs1(int x){   //求出每个点的子树大小、深度、重儿子
32     size[x] = 1, d[x] = d[fa[x]]+1, son[x] = 0;
33     for(int v,i = head[x]; i; i = e[i].next)
34         if((v = e[i].to)!=fa[x]){
35             fa[v] = x, dfs1(v), size[x] += size[v];
36             if(size[son[x]] < size[v])
37                 son[x] = v;
38         }
39 }
40
41 void dfs2(int x, int tp){   //求出每个节点的dfs序, dfs序对应的节点, 以及每个点所在链的顶端节点
42     top[x] = tp, id[x] = ++tot, rk[tot] = x;
43     if(son[x]) dfs2(son[x],tp);
44     for(int v,i = head[x]; i; i = e[i].next)
45         if((v = e[i].to)!=fa[x] && v!=son[x]) dfs2(v,v);
46 }
47
48 inline void pushup(int x){ //基础的线段树向上区间合并
49     t[x].sum = (t[t[x].ls].sum+t[t[x].rs].sum)%mod;  //此题需要将sum和对mod取模
50 }
51
52 void build(int l, int r, int x){ //基础建树, 动态开点
53     t[x].l = l, t[x].r = r, t[x].lazy = 0;
54     if(l == r){
55         t[x].sum = val[rk[l]]; return;
56     }
57     int mid = (l+r)>>1;
58     t[x].ls = ++tot, t[x].rs = ++tot;
59     build(l,mid,t[x].ls), build(mid+1,r,t[x].rs), pushup(x);
60 }
61
62 inline int len(int x) { return t[x].r-t[x].l+1; }
63
64 inline void pushdown(int x){ //基础的线段树标记下放
65     if(t[x].lazy && t[x].l != t[x].r){
66         int ls = t[x].ls, rs = t[x].rs, lz = t[x].lazy;
67         (t[ls].lazy+=lz) %= mod, (t[rs].lazy+=lz) %= mod;
68         (t[ls].sum+=lz*len(ls)) %= mod, (t[rs].sum+=lz*len(rs)) %= mod;
69         t[x].lazy = 0;
70     }
71 }
72
73 void update(int l, int r, int x, int c){ //基础的线段树更新
74     if(t[x].l >= l && t[x].r <= r){
75         (t[x].lazy += c) %= mod, (t[x].sum += len(x)*c) %= mod;
76         return;
```

```
77      }
78      pushdown(x);
79      int mid = (t[x].l+t[x].r)>>1;
80      if(mid >= l) update(l,r,t[x].ls,c);
81      if(mid < r) update(l,r,t[x].rs,c);
82      pushup(x);
83  }
84
85  int query(int l, int r, int x){  //基础的线段树查询
86      if(t[x].l >= l && t[x].r <= r) return t[x].sum;
87      pushdown(x);
88      int mid = (t[x].l+t[x].r)>>1, tp = 0;
89      if(mid >= l) tp += query(l,r,t[x].ls);
90      if(mid < r) tp += query(l,r,t[x].rs);
91      return tp%mod;
92  }
93
94  inline int sum(int x, int y){   //将区间分为多条链，对于每条链直接查询
95      int ret = 0;
96      while(top[x] != top[y]){ //让x与y到达同一条链
97          if(d[top[x]] < d[top[y]]) swap(x,y); //找到更深的点
98          (ret += query(id[top[x]],id[x],rt)) %= mod;
99          x = fa[top[x]];
100     }
101     if(id[x] > id[y]) swap(x,y);
102     return (ret+query(id[x],id[y],rt))%mod;
103 }
104
105 inline void updates(int x, int y, int c){   //区间加z，将区间分为多条链
106     while(top[x] != top[y]){
107         if(d[top[x]] < d[top[y]]) swap(x,y);
108         update(id[top[x]],id[x],rt,c);  //对于每条链直接修改
109         x = fa[top[x]];
110     }
111     if(id[x] > id[y]) swap(x,y);
112     update(id[x],id[y],rt,c);
113 }
114
115 signed main()
116 {
117     scanf("%lld%lld%lld%lld",&n,&m,&root,&mod);
118     rep(i,1,n) scanf("%lld",&val[i]);
119     init();
120     rep(i,1,n-1){
121         int x,y; scanf("%lld%lld",&x,&y);
122         add(x,y), add(y,x);
123     }
124     tot = 0, dfs1(root), dfs2(root, root);
125     tot = 0, build(1,n,rt = ++tot);
126     rep(i,1,m){
127         int op,x,y,k; scanf("%lld",&op);
128         if(op == 1){
129             scanf("%lld%lld%lld",&x,&y,&k);
130             updates(x,y,k);
131         }
132         else if(op == 2){
133             scanf("%lld%lld",&x,&y);
134             printf("%lld\n",sum(x,y));
135         }
```

```
136         else if(op == 3){
137             scanf("%lld%lld",&x,&y);
138             update(id[x],id[x]+size[x]-1,rt,y);
139         }
140         else{
141             scanf("%lld",&x);
142             printf("%lld\n",query(id[x],id[x]+size[x]-1,rt));
143         }
144     }
145     return 0;
146 }
```

4.12.2 树上路径颜色段数量

题意:
一颗 n 个节点的树，两个操作:
将 $a->b$ 路径上的点都染成颜色 c
查询 $a->b$ 路径上的颜色段数量

思路:
很明显是一个树剖问题，树剖的基础实现就不多说了，我们来考虑一下线段树需要维护什么。

首先求的是路径上不同颜色段数量，因此肯定需要维护一个 cnt，表示路径上不同颜色段的数量。然后我们来看如何实现区间合并，主要观察的就是左区间的右端点和右区间的左端点是否一样，如果一样，合并的 cnt 需要减 1。但是如何去快速查询左右节点颜色呢，因此可以想到再维护区间左右节点颜色即可完成本题。

树剖成多个树链时需要注意链条交界处的颜色是否一致。

```
1  #include <cstdio>
2  #include <iostream>
3  #include <cstring>
4  #include <algorithm>
5  #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
6  #define rep(i,a,b) for(int i = a; i <= b; i++)
7  #define LOG1(x1,x2) cout << x1 << ": " << x2 << endl;
8  #define LOG2(x1,x2,y1,y2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << endl
       ;
9  typedef long long ll;
10 typedef double db;
11 #define int long long
12 const db EPS = 1e-9;
13 const int N = 1e5+100;
14 using namespace std;
15
16 struct Edge { int next,to;} e[2*N];
17 struct Node { int l,r,ls,rs,cnt,lazy,lc,rc;} t[2*N]; //cnt: 不同颜色个数, lc: 左端颜色, rc
       : 右端颜色
18 int n,m,root,rt,mod,val[N],head[N],tot,fa[N],d[N],son[N],size[N],top[N],id[N],rk[N];
19 //top[x]: x节点所在链的顶端节点, id[x]: 节点dfs序, rk[x]: dfs序对应的节点
20 //val[x]: 每个点初始权值, fa[x]: 每个点父节点, d[x]: 节点深度, size[x]: 节点子树大小
21 //rt: 线段树根节点编号
22 void init(){
23     memset(head,0,sizeof head);
24     tot = 1, size[0] = 0;
25 }
26
27 void add(int x, int y){
```

```
28      e[++tot].next = head[x], e[tot].to = y, head[x] = tot;
29  }
30
31  void dfs1(int x){   //求出每个点的子树大小、深度、重儿子
32      size[x] = 1, d[x] = d[fa[x]]+1, son[x] = 0;
33      for(int v,i = head[x]; i; i = e[i].next)
34          if((v = e[i].to)!=fa[x]){
35              fa[v] = x, dfs1(v), size[x] += size[v];
36              if(size[son[x]] < size[v])
37                  son[x] = v;
38          }
39  }
40
41  void dfs2(int x, int tp){   //求出每个节点的dfs序, dfs序对应的节点, 以及每个点所在链的顶端节点
42      top[x] = tp, id[x] = ++tot, rk[tot] = x;
43      if(son[x]) dfs2(son[x],tp);
44      for(int v,i = head[x]; i; i = e[i].next)
45          if((v = e[i].to)!=fa[x] && v!=son[x]) dfs2(v,v);
46  }
47
48  inline void pushup(int x){ //基础的线段树向上区间合并
49      t[x].lc = t[t[x].ls].lc, t[x].rc = t[t[x].rs].rc, t[x].cnt = t[t[x].ls].cnt + t[t[x
        ].rs].cnt;
50      if(t[t[x].ls].rc == t[t[x].rs].lc) t[x].cnt--;
51  }
52
53  void build(int l, int r, int x){ //基础建树, 动态开点
54      t[x].l = l, t[x].r = r, t[x].lazy = -1;
55      if(l == r){
56          t[x].lc = t[x].rc = val[rk[l]], t[x].cnt = 1; return;
57      }
58      int mid = (l+r)>>1;
59      t[x].ls = ++tot, t[x].rs = ++tot;
60      build(l,mid,t[x].ls), build(mid+1,r,t[x].rs), pushup(x);
61  }
62
63  inline void pushdown(int x){ //基础的线段树标记下放
64      if(t[x].lazy!=-1 && t[x].l != t[x].r){
65          int ls = t[x].ls, rs = t[x].rs, lz = t[x].lazy;
66          t[ls].lazy = lz, t[rs].lazy = lz;
67          t[ls].lc = t[ls].rc = lz, t[rs].lc = t[rs].rc = lz;
68          t[ls].cnt = t[rs].cnt = 1;
69          t[x].lazy = -1;
70      }
71  }
72
73  void update(int l, int r, int x, int c){ //基础的线段树更新
74      if(t[x].l >= l && t[x].r <= r){
75          t[x].lazy = c, t[x].cnt = 1, t[x].lc = t[x].rc = c;
76          return;
77      }
78      pushdown(x);
79      int mid = (t[x].l+t[x].r)>>1;
80      if(mid >= l) update(l,r,t[x].ls,c);
81      if(mid < r) update(l,r,t[x].rs,c);
82      pushup(x);
83  }
84
85  int query(int l, int r, int x){  //基础的线段树查询
```

```
    // LOG2("l",l,"r",r);
    // LOG2("t[x].l",t[x].l,"t[x].r",t[x].r);
    if(t[x].l >= l && t[x].r <= r) return t[x].cnt;
    pushdown(x);
    int mid = (t[x].l+t[x].r)>>1, tp = 0;
    if(mid >= l && mid >= r) tp += query(l,r,t[x].ls);
    else if(mid < l && mid < r) tp += query(l,r,t[x].rs);
    else if(mid >= l && mid < r){
        tp += query(l,r,t[x].ls);
        tp += query(l,r,t[x].rs);
        if(t[t[x].ls].rc == t[t[x].rs].lc) tp--;
    }
    return tp;
}

int query_color(int pos, int x){  //查询单点颜色
    if(t[x].l == pos) return t[x].lc;
    if(t[x].r == pos) return t[x].rc;
    if(pos >= t[x].l && pos <= t[x].r && t[x].cnt == 1) return t[x].lc;
    int mid = (t[x].l+t[x].r)>>1;
    if(pos <= mid) return query_color(pos,t[x].ls);
    else return query_color(pos,t[x].rs);
}

inline int sum(int x, int y){   //将区间分为多条链，对于每条链直接查询
    int ret = 0;
    while(top[x] != top[y]){ //让x与y到达同一条链
        if(d[top[x]] < d[top[y]]) swap(x,y); //找到更深的点
        ret += query(id[top[x]],id[x],rt);
        // LOG1("ret",ret);
        if(query_color(id[top[x]],rt) == query_color(id[fa[top[x]]],rt)) ret--;
        x = fa[top[x]];
    }
    if(id[x] > id[y]) swap(x,y);
    return (ret+query(id[x],id[y],rt));
}

inline void updates(int x, int y, int c){   //区间加z，将区间分为多条链
    while(top[x] != top[y]){
        if(d[top[x]] < d[top[y]]) swap(x,y);
        update(id[top[x]],id[x],rt,c);  //对于每条链直接修改
        x = fa[top[x]];
    }
    if(id[x] > id[y]) swap(x,y);
    update(id[x],id[y],rt,c);
}

signed main()
{
    scanf("%lld%lld",&n,&m);
    rep(i,1,n) scanf("%lld",&val[i]);
    init();
    rep(i,1,n-1){
        int x,y; scanf("%lld%lld",&x,&y);
        add(x,y), add(y,x);
    }
    scanf("%lld",&m);
    root = 1;
    tot = 0, dfs1(root), dfs2(root, root);
```

```
    tot = 0, build(1,n,rt = ++tot);
    rep(i,1,m){
        char op[10]; scanf("%s",op);
        int x,y,z;
        if(op[0] == 'Q'){
            scanf("%lld%lld",&x,&y);
            printf("%lld\n",sum(x,y));
        }
        else{
            scanf("%lld%lld%lld",&x,&y,&z);
            updates(x,y,z);
        }
    }
    return 0;
}
```

4.12.3 动态开点树剖

题意:
现在有 n 个城市，构成了一颗树。每个城市都有自己信仰的宗教，以及城市评级。现在一共有四种操作:
某个城市改信 c 教
某个城市的评级调整为 w
$x->y$ 路径上所有与 x 信仰相同的城市的评级之和
$x->y$ 路径上所有与 x 信仰相同的城市的评级最大值

$(N, Q \leq 10^5\ C \leq 10^5)$

思路:
需要维护的操作只是单点修改和区间最值与最大值，维护的操作都不难。主要困难的地方在于最值和最大值都只在信仰相同的城市之间统计，因此我们需要对每个信仰都建一颗线段树。

但是由于空间的限制，对每个宗教都建一颗完整的线段树是不可能的，因此我们需要动态开点的操作。采用动态开点的原因是本题初始最多只有 10^5 个点，操作最多也只有 10^5 次，因此有效的点最多只有 $2 * 10^5$ 个，所以我们只需要维护这些有效点即可。

我们再来仔细讲一下动态开点的原理。如下图所示，现在只有点 A 是有效点，因此我们只需给 $root->A$ 路径上的点分配空间，不需要给其他的点分配空间，因此就达到了简化空间的目的。所以动态开一个点的空间费用最多是 $logn$，我们只需要给没颗树记一个根节点，以及每个节点对应的左右儿子编号即可。因此我们也不需要之前建树的 $build$ 函数了，用 $update$ 函数动态插入每个点即可。

回到这道题来，解决的思路就很简单了。给每个宗教建一颗线段树，维护每颗线段树的根节点编号，然后对于每个线段树进行查询和修改即可，一个点从信仰 a 变为信仰 b，只需在 a 线段树中将这个点赋为 0，然后在 b 线段树将这个点再赋值即可。

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <algorithm>
#define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
#define rep(i,a,b) for(int i = a; i <= b; i++)
#define LOG1(x1,x2) cout << x1 << ": " << x2 << endl;
#define LOG2(x1,x2,y1,y2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << endl
    ;
typedef long long ll;
typedef double db;
const db EPS = 1e-9;
using namespace std;
```

```
13 const int N = 2*1e5+1000;
14
15 int ls[20*N], rs[20*N], val[20*N], maxn[20*N], sz, rt[N];
16 int n,m,head[N],tot,son[N],size[N],d[N],id[N],rk[N],fa[N],top[N],ans1,ans2;
17 int re[N],lev[N];
18 struct Edge{
19     int to,next;
20 }e[N];
21
22 void init() { tot = 1, memset(head,0,sizeof head); }
23 void add(int x, int y) { e[++tot].to = y, e[tot].next = head[x], head[x] = tot; }
24 void dfs1(int x){   //求出每个点的子树大小、深度、重儿子
25     size[x] = 1, d[x] = d[fa[x]]+1, son[x] = 0;
26     for(int v,i = head[x]; i; i = e[i].next)
27         if((v = e[i].to)!=fa[x]){
28             fa[v] = x, dfs1(v), size[x] += size[v];
29             if(size[son[x]] < size[v])
30                 son[x] = v;
31         }
32 }
33 void dfs2(int x, int tp){   //求出每个节点的dfs序, dfs序对应的节点, 以及每个点所在链的顶端节点
34     top[x] = tp, id[x] = ++tot, rk[tot] = x;
35     if(son[x]) dfs2(son[x],tp);
36     for(int v,i = head[x]; i; i = e[i].next)
37         if((v = e[i].to)!=fa[x] && v!=son[x]) dfs2(v,v);
38 }
39 void update(int& now, int l, int r, int x, int c){  //单点修改
40     if(!now) now = ++sz;
41     if(l == r){
42         val[now] = c, maxn[now] = c;
43         return;
44     }
45     int mid = (l+r)>>1;
46     if(x <= mid) update(ls[now],l,mid,x,c);
47     if(x > mid) update(rs[now],mid+1,r,x,c);
48     val[now] = val[ls[now]]+val[rs[now]];
49     maxn[now] = max(maxn[ls[now]],maxn[rs[now]]);
50 }
51 void query(int now, int l, int r, int xx, int yy){
52     if(l >= xx && r <= yy){
53         ans1 += val[now], ans2 = max(ans2,maxn[now]);
54         return;
55     }
56     int mid = (l+r)>>1;
57     if(xx <= mid) query(ls[now],l,mid,xx,yy);
58     if(yy > mid) query(rs[now],mid+1,r,xx,yy);
59 }
60 inline void updates(int x, int y){  //区间加z, 将区间分为多条链
61     int tp = re[x];
62     while(top[x] != top[y]){
63         if(d[top[x]] < d[top[y]]) swap(x,y);
64         query(rt[tp],1,n,id[top[x]],id[x]); //对于每条链直接修改
65         x = fa[top[x]];
66     }
67     if(id[x] > id[y]) swap(x,y);
68     query(rt[tp],1,n,id[x],id[y]);
69 }
70 int main()
71 {
```

```
72      scanf("%d%d",&n,&m);
73      rep(i,1,n) scanf("%d%d",&lev[i],&re[i]);
74      init();
75      rep(i,1,n-1){
76          int xx,yy; scanf("%d%d",&xx,&yy);
77          add(xx,yy), add(yy,xx);
78      }
79      tot = sz = 0, dfs1(1), dfs2(1,1);
80      rep(i,1,n) update(rt[re[i]],1,n,id[i],lev[i]);
81      rep(i,1,m){
82          char s[10]; int x,y;
83          scanf("%s",s);
84          if(s[1] == 'C'){
85              scanf("%d%d",&x,&y);
86              update(rt[re[x]],1,n,id[x],0);
87              update(rt[y],1,n,id[x],lev[x]);
88              re[x] = y;
89          }
90          else if(s[1] == 'W'){
91              scanf("%d%d",&x,&y);
92              update(rt[re[x]],1,n,id[x],y);
93              lev[x] = y;
94          }
95          else{
96              scanf("%d%d",&x,&y);
97              ans1 = ans2 = 0;
98              updates(x,y);
99              if(s[1] == 'S') printf("%d\n",ans1);
100             else printf("%d\n",ans2);
101         }
102     }
103     return 0;
104 }
```

4.12.4 树剖换根

题意:
n 个点的树，每个顶点都有一个值，需要进行三种操作:
将树根修改为 x
将 $x->y$ 路径上所有点的值修改为 v
询问在当前树根状态下，以节点 x 为根的子树中的最小值
需要注意，树根修改了之后，节点 x 对应的子树也就变化了。

思路:
我们来分类讨论，枚举一下情况。我们需要求节点 x 在当前树根下的子树中的最小值。
如果当前树根就是 x，那很明显答案就是整棵树的节点最小值。
如果当前树根在 x 的子树中，比如 $A1$，那么我们可以发现答案就是去除 $A1$ 在 x 中的这一段子树部分，剩余部分中找最小值即可。
如果当前树根在 x 子树之外，比如 $A2$，那么答案就是在 x 原来对应的子树中找一个最小值即可。

所以这题就变成了对于 x 与当前树根求一个 LCA 的问题，然后只有在第二种情况中才需要进行特殊处理，即在 x 的子树找到对应 $A1$ 的那一个儿子即可。当然在树剖中，利用 top 数组就可以完成 LCA 能够完成的功能，见如下代码。

```
1  #include <cstdio>
2  #include <iostream>
3  #include <cstring>
```

```
#include <algorithm>
#define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
#define rep(i,a,b) for(int i = a; i <= b; i++)
#define LOG1(x1,x2) cout << x1 << ": " << x2 << endl;
#define LOG2(x1,x2,y1,y2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << endl;
typedef long long ll;
typedef double db;
#define int long long
const db EPS = 1e-9;
const int N = 1e5+100;
using namespace std;

struct Edge { int next,to;} e[2*N];
struct Node { int l,r,ls,rs,lazy,minn;} t[2*N];
int n,m,root,rt,mod,val[N],head[N],tot,fa[N],d[N],son[N],size[N],top[N],id[N],rk[N];
//top[x]: x节点所在链的顶端节点, id[x]: 节点dfs序, rk[x]: dfs序对应的节点
//val[x]: 每个点初始权值, fa[x]: 每个点父节点, d[x]: 节点深度, size[x]: 节点子树大小
//rt: 线段树根节点编号
void init() { memset(head,0,sizeof head); tot = 1, size[0] = 0; }
void add(int x, int y) { e[++tot].next = head[x], e[tot].to = y, head[x] = tot; }
void dfs1(int x){   //求出每个点的子树大小、深度、重儿子
    size[x] = 1, d[x] = d[fa[x]]+1, son[x] = 0;
    for(int v,i = head[x]; i; i = e[i].next)
        if((v = e[i].to)!=fa[x]){
            fa[v] = x, dfs1(v), size[x] += size[v];
            if(size[son[x]] < size[v])
                son[x] = v;
        }
}
void dfs2(int x, int tp){   //求出每个节点的dfs序, dfs序对应的节点, 以及每个点所在链的顶端节点
    top[x] = tp, id[x] = ++tot, rk[tot] = x;
    if(son[x]) dfs2(son[x],tp);
    for(int v,i = head[x]; i; i = e[i].next)
        if((v = e[i].to)!=fa[x] && v!=son[x]) dfs2(v,v);
}
inline void pushup(int x) { t[x].minn = min(t[t[x].ls].minn,t[t[x].rs].minn); }
void build(int l, int r, int x){ //基础建树, 动态开点
    t[x].l = l, t[x].r = r, t[x].lazy = 0;
    if(l == r){
        t[x].minn = val[rk[l]]; return;
    }
    int mid = (l+r)>>1;
    t[x].ls = ++tot, t[x].rs = ++tot;
    build(l,mid,t[x].ls), build(mid+1,r,t[x].rs), pushup(x);
}
inline int len(int x) { return t[x].r-t[x].l+1; }
inline void pushdown(int x){ //基础的线段树标记下放
    if(t[x].lazy && t[x].l != t[x].r){
        int ls = t[x].ls, rs = t[x].rs, lz = t[x].lazy;
        (t[ls].lazy=lz), (t[rs].lazy=lz);
        t[ls].minn = lz, t[rs].minn = lz;
        t[x].lazy = 0;
    }
}
void update(int l, int r, int x, int c){ //基础的线段树更新
    if(t[x].l >= l && t[x].r <= r){
        t[x].lazy = c, t[x].minn = c;
        return;
```

```
    }
    pushdown(x);
    int mid = (t[x].l+t[x].r)>>1;
    if(mid >= l) update(l,r,t[x].ls,c);
    if(mid < r) update(l,r,t[x].rs,c);
    pushup(x);
}
int query(int l, int r, int x){  //基础的线段树查询
    if(t[x].l >= l && t[x].r <= r) return t[x].minn;
    pushdown(x);
    int mid = (t[x].l+t[x].r)>>1, tp = 1e15;
    if(mid >= l) tp = min(tp,query(l,r,t[x].ls));
    if(mid < r) tp = min(tp,query(l,r,t[x].rs));
    return tp;
}
inline void updates(int x, int y, int c){   //区间加z, 将区间分为多条链
    while(top[x] != top[y]){
        if(d[top[x]] < d[top[y]]) swap(x,y);
        update(id[top[x]],id[x],rt,c);  //对于每条链直接修改
        x = fa[top[x]];
    }
    if(id[x] > id[y]) swap(x,y);
    update(id[x],id[y],rt,c);
}
int find(int x, int y){ // top数组求LCA
    int base = x;
    while(top[x] != top[y]){
        if(fa[top[y]] == base) return top[y]; //返回子树中存在当前树根的子儿子
        if(d[top[x]] < d[top[y]]) y = fa[top[y]];
        else x = fa[top[x]];
    }
    if(d[x] > d[y]) x = y;
    if(x == base) return son[x]; //返回子树中存在当前树根的子儿子
    else return 0; //当前树根在x子树之外
}
signed main()
{
    scanf("%lld%lld",&n,&m);
    init();
    rep(i,1,n-1){
        int x,y; scanf("%lld%lld",&x,&y);
        add(x,y), add(y,x);
    }
    rep(i,1,n) scanf("%lld",&val[i]);
    scanf("%lld",&root);
    tot = 0, dfs1(root), dfs2(root, root);
    tot = 0, build(1,n,rt = ++tot);
    rep(i,1,m){
        int op,x,y,z;
        scanf("%lld",&op);
        if(op == 1){
            scanf("%lld",&x);
            root = x;
        }
        else if(op == 2){
            scanf("%lld%lld%lld",&x,&y,&z);
            updates(x,y,z);
        }
        else{
```

```
121             scanf("%lld",&x); int thp;
122             if(root == x){
123                 printf("%lld\n",query(1,n,rt));
124             }
125             else if((thp = find(x,root))){
126                 printf("%lld\n",min(query(1,id[thp]-1,rt),query(id[thp]+size[thp],n,rt)
    ));
127             }
128             else printf("%lld\n",query(id[x],id[x]+size[x]-1,rt));
129         }
130     }
131     return 0;
132 }
```

4.12.5 边剖 + 主席树

题意:
给定一棵 n 个节点的树，每条边都有一个权值，m 次查询，每次询问树上两点路径上边权小于 k 的边有多少条? $(1 \leq n, m \leq 10^5)$

思路:
比较裸的题目，可以离线操作，然后将询问值从小到大进行排序，然后每次单点修改，将比当前询问小的边加入树中。

此处需要注意是边权树剖，因此将每条边的权值压到深度更深的节点上，然后树剖路径查询时，最后两点处于同一条链时，将深度最浅的点改为该点的儿子即可。

比赛时没有考虑到将询问值排序，然后依次修改边权，因此采用了树上主席树的做法。先将所有的边都插入树中，然后直接查询区间中小于 k 的节点有多少个即可。

```
1  #include <cstdio>
2  #include <iostream>
3  #include <cstring>
4  #include <algorithm>
5  #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
6  #define rep(i,a,b) for(int i = a; i <= b; i++)
7  #define LOG1(x1,x2) cout << x1 << ": " << x2 << endl;
8  #define LOG2(x1,x2,y1,y2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << endl
       ;
9  #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2
       << " , " << z1 << ": " << z2 << endl;
10 typedef long long ll;
11 typedef double db;
12 const db EPS = 1e-9;
13 const int N = 1e5+100;
14 using namespace std;
15
16 struct Edge { int next,to,w;} e[2*N];
17 struct Node { int l,r,ls,rs,sum;} t[20*N];
18 int n,m,rt,mod,val[N],head[N],tot,fa[N],d[N],son[N],size[N],top[N],id[N],rk[N],root[N],
       rot;
19 //top[x]: x节点所在链的顶端节点, id[x]: 节点dfs序, rk[x]: dfs序对应的节点
20 //val[x]: 每个点初始权值, fa[x]: 每个点父节点, d[x]: 节点深度, size[x]: 节点子树大小
21 //rt: 线段树根节点编号
22 void init(){
23     memset(head,0,sizeof head);
24     tot = 1, size[0] = 0;
25 }
```

```
26
27  void add(int x, int y, int z){
28      e[++tot].next = head[x], e[tot].to = y, head[x] = tot, e[tot].w = z;
29  }
30
31  void dfs1(int x){   //求出每个点的子树大小、深度、重儿子
32      size[x] = 1, d[x] = d[fa[x]]+1, son[x] = 0;
33      for(int v,i = head[x]; i; i = e[i].next)
34          if((v = e[i].to)!=fa[x]){
35              fa[v] = x, dfs1(v), size[x] += size[v];
36              if(size[son[x]] < size[v])
37                  son[x] = v;
38          }
39  }
40
41  void dfs2(int x, int tp){   //求出每个节点的dfs序, dfs序对应的节点, 以及每个点所在链的顶端节点
42      top[x] = tp, id[x] = ++tot, rk[tot] = x;
43      if(son[x]) dfs2(son[x],tp);
44      for(int v,i = head[x]; i; i = e[i].next)
45          if((v = e[i].to)!=fa[x] && v!=son[x]) dfs2(v,v);
46  }
47
48  inline void pushup(int x){ //基础的线段树向上区间合并
49      t[x].sum = t[t[x].ls].sum+t[t[x].rs].sum;
50  }
51
52  int build(int l,int r) //主席树建树部分
53  {
54      int p = ++tot; // 新建一个节点, 编号为p, 代表当前区间[l,r]
55      t[p].l = l, t[p].r = r, t[p].sum = 0;
56      if(l == r) return p;
57      int mid = (l+r)>>1;
58      t[p].ls = build(l,mid);
59      t[p].rs = build(mid+1,r);
60      return p;
61  }
62
63  inline int len(int x) { return t[x].r-t[x].l+1; }
64
65  int ask(int lp,int rp,int k){ //主席树查询[lp,rp]比k小的有多少个
66      if(t[lp].l == t[lp].r) return (t[rp].sum-t[lp].sum);
67      int mid = (t[lp].l+t[lp].r)>>1;
68      int tp = 0;
69      if(mid < k){
70          tp += (t[t[rp].ls].sum-t[t[lp].ls].sum);
71          tp += ask(t[lp].rs,t[rp].rs,k);
72      }
73      else tp += ask(t[lp].ls,t[rp].ls,k);
74      return tp;
75  }
76
77  inline int solve(int x,int y,int k){
78      int ret = 0;
79      while(top[x] != top[y]){ //让x与y到达同一条链
80          if(d[top[x]] < d[top[y]]) swap(x,y); //找到更深的点
81          int xx = id[top[x]]-1, yy = id[x];
82          int hp = ask(root[xx],root[yy],k);
83          ret += hp;
84          x = fa[top[x]];
```

```
85      }
86      if(x == y) return ret;
87      if(id[x] > id[y]) swap(x,y);
88      int xx = id[son[x]]-1, yy = id[y]; //边权树链剖分，最后两点同链之后，取最高点的儿子来计算
89      //-1是因为主席树要获取-1时候的副本进行sum减运算
90      int hp = ask(root[xx],root[yy],k);
91      return (ret+hp);
92  }
93
94  int insert(int now,int pos,int k) //主席树插入一个新的值
95  {
96      int p = ++tot;
97      t[p] = t[now]; //建立副本
98      if(t[p].l == t[p].r){
99          t[p].sum += k; //在副本上修改
100         return p;
101     }
102     int mid = (t[p].l+t[p].r)>>1;
103     if(pos <= mid) t[p].ls = insert(t[p].ls,pos,k); //保留右儿子部分，把左儿子更新
104     else t[p].rs = insert(t[p].rs,pos,k);
105     t[p].sum = t[t[p].ls].sum + t[t[p].rs].sum;
106     return p;
107 }
108
109 int ed[2*N][3];
110 int b[2*N],tb;
111
112 signed main()
113 {
114     scanf("%d%d",&n,&m);
115     init();
116     rep(i,1,n-1){
117         int x,y,z; scanf("%d %d %d",&x,&y,&z);
118         add(x,y,z), add(y,x,z);
119         ed[i][0] = x, ed[i][1] = y, ed[i][2] = z;
120     }
121     rot = 1;
122     tot = 0, dfs1(rot), dfs2(rot, rot);
123     rep(i,1,n-1){
124         if(d[ed[i][0]] > d[ed[i][1]]) swap(ed[i][0],ed[i][1]);
125         val[ed[i][1]] = ed[i][2];
126         b[++tb] = ed[i][2];
127     }
128     sort(b+1,b+1+tb);
129     tb = unique(b+1,b+1+tb)-b-1;
130     tot = 0;
131     root[0] = build(1,tb);
132     rep(i,1,n){
133         int x = lower_bound(b+1,b+1+tb,val[rk[i]])-b;
134         root[i] = insert(root[i-1],x,1);
135     }
136     rep(i,1,m){
137         int u,v,k; scanf("%d%d%d",&u,&v,&k);
138         int pos = upper_bound(b+1,b+1+tb,k)-b;
139         pos--;
140         if(pos == 0) printf("0\n");
141         else printf("%d\n",solve(u,v,pos));
142     }
143     return 0;
```

```
}
```

4.13 CDQ

4.13.1 陌上开花 (三维偏序)

描述:
有 n 朵花, 每朵花有三个属性: 花形 (s)、颜色 (c)、气味 (m), 用三个整数表示。
现在要对每朵花评级, 一朵花的级别是它拥有的美丽能超过的花的数量。
定义一朵花 A 比另一朵花 B 要美丽, 当且仅 Sa>=Sb,Ca>=Cb,Ma>=Mb。
显然, 两朵花可能有同样的属性。需要统计出评出每个等级的花的数量。

输入:
第一行为 N,K (1 <= N <= 100,000, 1 <= K <= 200,000), 分别表示花的数量和最大属性值。
以下 N 行, 每行三个整数 si, ci, mi (1 <= si, ci, mi <= K), 表示第 i 朵花的属性

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <algorithm>
#define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
#define rep(i,a,b) for(int i = a; i <= b; i++)
#define LOG1(x1,x2) cout << x1 << ": " << x2 << endl;
#define LOG2(x1,x2,y1,y2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << endl
    ;
typedef long long ll;
typedef double db;
const int N = 2*1e5+100;
const int M = 1e5+100;
const db EPS = 1e-9;
using namespace std;

int n,k,ans[N],tot,t[N];
struct Node{
    int a,b,c,s,ans;
    bool operator == (Node xx) const {
        return ((a == xx.a) && (b == xx.b) && (c == xx.c));
    }
    bool operator < (Node xx) const {
        if(b != xx.b) return b < xx.b;
        else return c < xx.c;
    }
}p[N];

bool cmp(Node x,Node y){
    if(x.a != y.a) return x.a < y.a;
    else if(x.b != y.b) return x.b < y.b;
    else return x.c < y.c;
}

inline int lowbit(int x) { return x&(-x); }

inline void update(int x, int c){
    for(int i = x; i <= k; i += lowbit(i)) t[i] += c;
}

inline int ask(int x){
    int tp = 0;
```

```
42      for(int i = x; i; i -= lowbit(i)) tp += t[i];
43      return tp;
44  }
45
46  void CDQ(int l, int r)
47  {
48      if(l == r) return;
49      int mid = (l+r)>>1;
50      CDQ(l,mid); CDQ(mid+1,r);
51      sort(p+l,p+mid+1); sort(p+mid+1,p+r+1); //将第二维排序
52      int i = l, j = mid+1;
53      while(j <= r){
54          while(i <= mid && p[i].b <= p[j].b){
55              update(p[i].c,p[i].s);
56              i++;
57          }
58          p[j].ans += ask(p[j].c);
59          j++;
60      }
61      rep(kk,l,i-1) update(p[kk].c,-p[kk].s); //消除之前的影响，此处是 i，不是 mid
62  }
63
64  int main()
65  {
66      scanf("%d%d",&n,&k);
67      rep(i,1,n) scanf("%d%d%d",&p[i].a,&p[i].b,&p[i].c);
68      //离散化
69      sort(p+1,p+1+n,cmp);
70      int cnt = 0;
71      rep(i,1,n){
72          cnt++;
73          if(p[i]==p[i+1]) continue;
74          p[++tot] = p[i], p[tot].s = cnt, cnt = 0, p[tot].ans = 0;
75      }
76      CDQ(1,tot);
77      rep(i,1,tot)
78          ans[p[i].ans+p[i].s-1] += p[i].s;
79      rep(i,0,n-1) printf("%d\n",ans[i]);
80      return 0;
81  }
```

4.13.2 动态逆序对

描述：
对于序列 A，它的逆序对数定义为满足 i<j，且 Ai>Aj 的数对 (i,j) 的个数。给 1 到 n 的一个排列，按照某种顺序依次删除 m 个元素，你的任务是在每次删除一个元素之前统计整个序列的逆序对数

输入：
输入第一行包含两个整数 n 和 m，即初始元素的个数和删除的元素个数。
以下 n 行每行包含一个 1 到 n 之间的正整数，即初始排列。
以下 m 行每行一个正整数，依次为每次删除的元素。
N<=100000 M<=50000

输出：
输出包含 m 行，依次为删除每个元素之前，逆序对的个数。

```
1   #include <cstdio>
```

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>
#define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
#define rep(i,a,b) for(int i = a; i <= b; i++)
#define LOG1(x1,x2) cout << x1 << ": " << x2 << endl;
#define LOG2(x1,x2,y1,y2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << endl;
#define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << " , " << z1 << ": " << z2 << endl;
typedef long long ll;
typedef double db;
const int N = 1e5+100;
const int M = 5*1e4+100;
const db EPS = 1e-9;
using namespace std;

int n,m,tot,a[N],vis[N],b[N],mp[N];
ll t[N],ans[N];
struct Node{
    int x,y,f; //插在x处, val = y, 贡献对象为f
    bool operator < (Node xx) const {
        return x < xx.x;
    }
}p[N];

inline int lowbit(int x) { return x&(-x); }
inline void update(int x, int cc) { while(x <= n) t[x] += cc, x += lowbit(x); }
inline ll ask(int x) { ll tp = 0; while(x) tp += t[x], x -= lowbit(x); return tp; }

bool cmp(Node xx, Node yy){
    return xx.x > yy.x;
}

void solve(int l, int r){
    if(l == r) return;
    int mid = (l+r)>>1;
    solve(l,mid); solve(mid+1,r);
    sort(p+l,p+mid+1); sort(p+mid+1,p+r+1);
    int i = l, j = mid+1;
    while(j <= r){
        while(p[i].x <= p[j].x && i <= mid){
            update(p[i].y,1); i++;
        }
        int tp = ask(n)-ask(p[j].y);
        ans[p[j].f] += tp; j++;
    }
    rep(kk,l,i-1) update(p[kk].y,-1);

    sort(p+l,p+mid+1,cmp); sort(p+mid+1,p+r+1,cmp);
    i = l, j = mid+1;
    while(j <= r){
        while(p[i].x >= p[j].x && i <= mid){
            update(p[i].y,1); i++;
        }
        int tp = ask(p[j].y);
        ans[p[j].f] += tp; j++;
    }
    rep(kk,l,i-1) update(p[kk].y,-1);
```

```
}

int main()
{
    scanf("%d%d",&n,&m);
    rep(i,1,n){
        scanf("%d",&a[i]);
        mp[a[i]] = i;
    }
    rep(i,1,m){
        scanf("%d",&b[i]);
        vis[mp[b[i]]] = 1;
    }
    rep(i,1,n)
        if(!vis[i]) p[++tot] = {i,a[i],m};
    for(int i = m; i; i--){
        p[++tot] = {mp[b[i]],b[i],i};
    }
    solve(1,tot);
    for(int i = m-1; i; i--) ans[i] += ans[i+1];
    rep(i,1,m) printf("%lld\n",ans[i]);
    return 0;
}
```

4.13.3 矩阵前缀和

描述:
维护一个 W*W 的矩阵, 初始值均为 S. 每次操作可以增加某格子的权值, 或询问某子矩阵的总权值. 修改操作数 M<=160000, 询问数 Q<=10000,W<=2000000.

输入:
第一行两个整数,S,W; 其中 S 为矩阵初始值;W 为矩阵大小

接下来每行为一下三种输入之一 (不包含引号):

"1 x y a", 你需要把 (x,y)(第 x 行第 y 列) 的格子权值增加 a

"2 x1 y1 x2 y2", 你需要求出以左下角为 (x1,y1), 右上角为 (x2,y2) 的矩阵内所有格子的权值和, 并输出

"3", 表示输入结束

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <algorithm>
#define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
#define rep(i,a,b) for(int i = a; i <= b; i++)
#define LOG1(x1,x2) cout << x1 << ": " << x2 << endl;
#define LOG2(x1,x2,y1,y2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << endl
    ;
#define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2
    << " , " << z1 << ": " << z2 << endl;
typedef long long ll;
typedef double db;
const int N = 160000+4*1e4+100;
const int M = 1e4+100;
const db EPS = 1e-9;
```

```
15 using namespace std;
16
17 int T,n,ans[M],tot,num,t[2000010]; //tot-操作个数  num-查询个数
18 struct Node{
19     int x,y,op,val,f; //x坐标 y坐标 op:1-修改 op:2-查询 val:答案贡献(1/-1) f:对应答案编号
20     bool operator < (Node xx) const {
21         if(x != xx.x) return x < xx.x;
22         else return op < xx.op;
23     }
24 }p[N];
25
26 inline int lowbit(int x) { return x&(-x); }
27 inline void update(int x, int val) { while(x <= n) t[x] += val, x += lowbit(x); }
28 inline int ask(int x){
29     int tp = 0; while(x >= 1) tp += t[x], x -= lowbit(x);
30     return tp;
31 }
32
33 void solve(int l, int r){
34     if(l == r) return;
35     int mid = (l+r)>>1;
36     solve(l,mid); solve(mid+1,r);
37     sort(p+l,p+mid+1); sort(p+mid+1,p+r+1);
38     int i = l, j = mid+1;
39     while(j <= r){
40         if(p[j].op == 1){
41             j++; continue;
42         }
43         while(p[i].x <= p[j].x && i <= mid){
44             if(p[i].op == 1) update(p[i].y,p[i].val);
45             i++;
46         }
47         ans[p[j].f] += p[j].val*ask(p[j].y);
48         j++;
49     }
50     rep(kk,l,i-1){
51         if(p[kk].op == 1) update(p[kk].y,-p[kk].val);
52     }
53 }
54
55 int main()
56 {
57     scanf("%d%d",&T,&n);
58     int op,xx,yy,aa,zz;
59     while(scanf("%d",&op)){
60         if(op == 3) break;
61         if(op == 1){
62             scanf("%d%d%d",&xx,&yy,&aa);
63             p[++tot] = {xx,yy,1,aa,0};
64         }
65         else{
66             scanf("%d%d%d%d",&xx,&yy,&aa,&zz);
67             ++num;
68             p[++tot] = {aa,zz,2,1,num};
69             p[++tot] = {xx-1,yy-1,2,1,num};
70             p[++tot] = {xx-1,zz,2,-1,num};
71             p[++tot] = {aa,yy-1,2,-1,num};
72         }
73     }
```

```
74      solve(1,tot);
75      rep(i,1,num) printf("%d\n",ans[i]);
76      return 0;
77  }
78
79  /*
80  0 4
81  1 2 3 3
82  2 1 1 3 3
83  1 2 2 2
84  2 2 2 3 4
85  3
86  */
```

4.14 莫队

4.14.1 普通莫队

描述:
N 只袜子从 1 到 N 编号, 每个袜子都有一个数值表示颜色, 然后从编号 L 到 R 有多大的概率抽到两只颜色相同的袜子。

输出:
包含 M 行，对于每个询问在一行中输出分数 A/B 表示从该询问的区间 [L,R] 中随机抽出两只袜子颜色相同的概率。若该概率为 0 则输出 0/1，否则输出的 A/B 必须为最简分数。

时间复杂度: $O(n^{\frac{1}{2}})$

```
1   #include <cstdio>
2   #include <iostream>
3   #include <cstring>
4   #include <cmath>
5   #include <algorithm>
6   #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
7   #define rep(i,a,b) for(int i = a; i <= b; i++)
8   #define LOG1(x1,x2) cout << x1 << ": " << x2 << endl;
9   #define LOG2(x1,x2,y1,y2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << endl
        ;
10  #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2
        << " , " << z1 << ": " << z2 << endl;
11  typedef long long ll;
12  typedef double db;
13  const int N = 2*1e5+100;
14  const int M = 1e5+100;
15  const db EPS = 1e-9;
16  using namespace std;
17
18  int a[N],pos[N],n,m,L,R;
19  ll ans[N][2],flag[N],Ans;
20  struct Node{
21      int l,r,id;
22      bool operator < (Node xx) const{
23          if(pos[l] == pos[xx.l]) return r < xx.r;
24          else return pos[l] < pos[xx.l];
25      }
26  }Q[N];
27
28  ll gcd(ll a,ll b)
29  {
```

```
30      return b == 0 ? a:gcd(b,a%b);
31  }
32
33  void add(int x){
34      Ans += flag[a[x]];
35      flag[a[x]]++;
36  }
37
38  void del(int x){
39      flag[a[x]]--;
40      Ans -= flag[a[x]];
41  }
42
43  int main()
44  {
45      L = 1, R = 0;
46      scanf("%d%d",&n,&m);
47      int sz = sqrt(n);
48      rep(i,1,n){
49          scanf("%d",&a[i]);
50          pos[i] = i/sz;
51      }
52      rep(i,1,m){
53          scanf("%d%d",&Q[i].l,&Q[i].r);
54          Q[i].id = i;
55      }
56      sort(Q+1,Q+1+m);
57      rep(i,1,m){
58          while(L<Q[i].l){
59              del(L);
60              L++;
61          }
62          while(L>Q[i].l){
63              L--;
64              add(L);
65          }
66          while(R<Q[i].r){
67              R++;
68              add(R);
69          }
70          while(R>Q[i].r){
71              del(R);
72              R--;
73          }
74          ll len = Q[i].r-Q[i].l+1;
75          ll tp = len*(len-1ll)/(ll)2;
76          ll g = gcd(Ans,tp);
77          ans[Q[i].id][0] = Ans/g;
78          ans[Q[i].id][1] = tp/g;
79      }
80      rep(i,1,m) printf("%lld/%lld\n",ans[i][0],ans[i][1]);
81      return 0;
82  }
```

4.14.2 带修改莫队

区间求 mex, 支持修改
一开始弄了个树状数组然后二分, 然后被 T 了。于是想起 SG 函数求 mex, 直接暴力从 0 开始枚举, 因此改为暴力枚举,

然后通过。
时间复杂度：$O(n^{frac53})$

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <cmath>
#include <algorithm>
#define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
#define rep(i,a,b) for(int i = a; i <= b; i++)
#define LOG1(x1,x2) cout << x1 << ": " << x2 << endl;
#define LOG2(x1,x2,y1,y2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << endl;
#define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << " , " << z1 << ": " << z2 << endl;
typedef long long ll;
typedef double db;
const int N = 2*1e5+100;
const int M = 1e5+100;
const db EPS = 1e-9;
using namespace std;

int n,qq,a[N],b[N],tot,Qnum,Cnum,pos[N],ans[N],L,R,T,flag[N],vis[N];
struct Query{
    int l,r,id,t;
    bool operator < (Query xx) const {
        if(pos[l] != pos[xx.l]) return pos[l] < pos[xx.l];
        else if(pos[r] != pos[xx.r]) return pos[r] < pos[xx.r];
        else return t < xx.t;
    }
}q[M];
struct Change{
    int pos,val;
}C[M];

int find(int x){
    return lower_bound(b+1,b+1+tot,x)-b;
}

void add(int x){
    if(flag[a[x]]!=0) vis[flag[a[x]]]--;
    flag[a[x]]++; vis[flag[a[x]]]++;
}

void del(int x){
    vis[flag[a[x]]]--; flag[a[x]]--;
    if(flag[a[x]] != 0) vis[flag[a[x]]]++;
}

void Work(int x,int i){
    if(C[x].pos >= q[i].l && C[x].pos <= q[i].r){
        vis[flag[a[C[x].pos]]]--; flag[a[C[x].pos]]--;
        if(flag[a[C[x].pos]] != 0) vis[flag[a[C[x].pos]]]++;
        if(flag[C[x].val] != 0) vis[flag[C[x].val]]--;
        flag[C[x].val]++; vis[flag[C[x].val]]++;
    }
    swap(a[C[x].pos],C[x].val);
}

```

```
int solve(){
    rep(i,0,n)
        if(!vis[i]) return i;
}

int main()
{
    scanf("%d%d",&n,&qq);
    rep(i,1,n){
        scanf("%d",&a[i]);
        b[++tot] = a[i];
    }
    rep(i,1,qq){
        int op,l,r; scanf("%d%d%d",&op,&l,&r);
        if(op == 1) Qnum++, q[Qnum] = {l,r,Qnum,Cnum};
        else C[++Cnum] = {l,r}, b[++tot] = r;
    }
    sort(b+1,b+1+tot);
    tot = unique(b+1,b+1+tot)-b-1;
    int sz = pow(n,0.66666666666666);
    rep(i,1,n) pos[i] = i/sz;
    sort(q+1,q+1+Qnum);
    L = 1, R = 0, T = 0;
    vis[0] = 1;
    rep(i,1,n) a[i] = find(a[i]);
    rep(i,1,Cnum) C[i].val = find(C[i].val);
    rep(i,1,Qnum){
        while(L < q[i].l) del(L++);
        while(L > q[i].l) add(--L);
        while(R < q[i].r) add(++R);
        while(R > q[i].r) del(R--);
        while(T < q[i].t) Work(++T,i);
        while(T > q[i].t) Work(T--,i);
        ans[q[i].id] = solve();
    }
    rep(i,1,Qnum) printf("%d\n",ans[i]);
    return 0;
}
```

4.14.3 树上带修改莫队

题意:
树上每个点都有一个权值, 表示这个节点所属类别, 第 i 次遇到种类节点 x 获得愉悦度 W[i]*C[x], 询问树上 [l,r] 路径上的愉悦值并且支持树上单点修改

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <cmath>
#include <algorithm>
#define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
#define rep(i,a,b) for(int i = a; i <= b; i++)
#define LOG1(x1,x2) cout << x1 << ": " << x2 << endl;
#define LOG2(x1,x2,y1,y2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << endl
    ;
#define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2
    << " , " << z1 << ": " << z2 << endl;
```

```
11 typedef long long ll;
12 typedef double db;
13 const int N = 1e5+100;
14 const int M = 1e6+100;
15 const db EPS = 1e-9;
16 using namespace std;
17
18 int n,m,k,V[N],W[N],head[N],tot,C[N],qnum,cnum,f[N][25],t,d[N],Euler[2*N],ncnt,fir[N],
       las[N],pos[2*N],L,R,T,flag[N],vis[N];
19 //pos-分块位置、fir-欧拉序第一次、las-欧拉序第二次、Euler-欧拉序数组、ncnt-欧拉序数组长度
20 //vis-这个树上节点出现了几次, flag-这个糖果种类
21 ll ans[N],now;
22 struct Edge{
23     int to,next;
24 }e[2*N];
25 struct Query{
26     int l,r,id,lca,t; //l、r、id-查询顺序、lca-两点lca、t-之前有几次修改
27     bool operator < (Query xx) const {
28         if(pos[l] != pos[xx.l]) return pos[l] < pos[xx.l];
29         else if(pos[r] != pos[xx.r]) return pos[r] < pos[xx.r];
30         else return t < xx.t;
31     }
32 }q[N];
33 struct Change{
34     int pos, val;
35 }ch[N];
36
37 void add(int x,int y){
38     e[++tot].to = y, e[tot].next = head[x], head[x] = tot;
39 }
40
41 void dfs(int u,int fa)
42 {
43     Euler[++ncnt] = u; fir[u] = ncnt;
44     d[u]=d[fa]+1; f[u][0]=fa;
45     for(int i=1;(1<<i)<=d[u];i++)
46         f[u][i]=f[f[u][i-1]][i-1];
47     for(int i=head[u]; i; i=e[i].next){
48         int v=e[i].to;
49         if(v!=fa) dfs(v,u);
50     }
51     Euler[++ncnt] = u; las[u] = ncnt;
52 }
53
54 int LCA(int x,int y)
55 {
56     if(d[x] > d[y]) swap(x,y);
57     for(int i = t; i >= 0; i--)
58         if(d[f[y][i]] >= d[x]) y = f[y][i];  //往上追溯, 直至y和x位于同一深度
59     if(x == y) return x;  //如果已经找到了, 就返回x
60     for(int i = t; i >= 0; i--)
61         if(f[x][i] != f[y][i]) x = f[x][i], y = f[y][i];  //x和y同时往上走, 一直到x和y恰好为
    lca的子节点
62     return f[x][0];  //x和y共同的根节点就是lca
63 }
64
65 void Add(int pos){
66     flag[C[pos]]++;
67     now += (ll)W[flag[C[pos]]]*(ll)V[C[pos]];
```

```
}

void Del(int pos){
    now -= (ll)W[flag[C[pos]]]*(ll)V[C[pos]];
    flag[C[pos]]--;
}

void add_del(int pos){ //增加和减少取决于这个点被遍历了几次
    vis[pos] ? Del(pos) : Add(pos);
    vis[pos] ^= 1;
}

void work(int x){
    if(vis[ch[x].pos]){ //修改点为有效点
        add_del(ch[x].pos); //减掉
        swap(C[ch[x].pos], ch[x].val);
        add_del(ch[x].pos); //加上
    }
    else swap(C[ch[x].pos], ch[x].val);
}

int main()
{
    scanf("%d%d%d",&n,&m,&k);
    rep(i,1,m) scanf("%d",&V[i]);
    rep(i,1,n) scanf("%d",&W[i]);
    rep(i,1,n-1){
        int xx,yy; scanf("%d%d",&xx,&yy);
        add(xx,yy); add(yy,xx);
    }
    rep(i,1,n) scanf("%d",&C[i]);
    t = (int)(log(n)/log(2))+1;
    dfs(1,0);
    int sz = pow(ncnt,2.0/3.0);
    for(int i = 0; i <= ncnt; i++) pos[i] = i/sz;
    rep(i,1,k){
        int op,x,y; scanf("%d%d%d",&op,&x,&y);
        if(op){
            int lca = LCA(x,y);
            q[++qnum].t = cnum; q[qnum].id = qnum;
            if(fir[x] > fir[y]) swap(x,y);
            if(x == lca) q[qnum].l = fir[x], q[qnum].r = fir[y], q[qnum].lca = 0;
            else q[qnum].l = las[x], q[qnum].r = fir[y], q[qnum].lca = lca;
        }
        else ch[++cnum] = {x,y};
    }
    sort(q+1,q+1+qnum);
    L = 1, R = 0, T = 0;
    rep(i,1,qnum){
        while(L < q[i].l){
            add_del(Euler[L]); L++;
        }
        while(L > q[i].l){
            L--; add_del(Euler[L]);
        }
        while(R < q[i].r){
            R++; add_del(Euler[R]);
        }
        while(R > q[i].r){
```

```
127             add_del(Euler[R]); R--;
128         }
129         while(T < q[i].t){
130             ++T; work(T);
131         }
132         while(T > q[i].t){
133             work(T); --T;
134         }
135         if(q[i].lca) add_del(q[i].lca); //lca不在欧拉序列区间中
136         ans[q[i].id] = now;
137         if(q[i].lca) add_del(q[i].lca); //恢复这个区间的状态
138     }
139     rep(i,1,qnum) printf("%lld\n",ans[i]);
140     return 0;
141 }
```

4.15 虚树

题意：给定一棵树, 敌人在 1 号节点, 每次询问给出 k 个节点, 每条边都有边权, 问删掉树中边使得 1 号节点不能到达 k 个节点中任意一个节点需要的最小代价

输入：第一行一个整数 n，代表岛屿数量。
接下来 n-1 行，每行三个整数 u,v,w，代表 u 号岛屿和 v 号岛屿由一条代价为 c 的桥梁直接相连，保证 1<=u,v<=n 且 1<=c<=100000。
第 n+1 行，一个整数 m，代表敌方机器能使用的次数。
接下来 m 行，每行一个整数 ki，代表第 i 次后，有 ki 个岛屿资源丰富，接下来 k 个整数 h1,h2,…hk，表示资源丰富岛屿的编号。每次输出最小代价

思路：
首先考虑最暴力的 dp, 设 $f[x]$ 表示处理完以 x 为根的子树的最小花费

转移有两种情况
1. 断开自己与父亲的联系，代价为从根到该节点的最小值
2. 将子树内的节点全都处理掉的代价

```
1  #include <cstdio>
2  #include <iostream>
3  #include <cstring>
4  #include <queue>
5  #include <cmath>
6  #include <algorithm>
7  #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
8  #define rep(i,a,b) for(int i = a; i <= b; i++)
9  #define LOG1(x1,x2) cout << x1 << ": " << x2 << endl;
10 #define LOG2(x1,x2,y1,y2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << endl
       ;
11 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2
       << " , " << z1 << ": " << z2 << endl;
12 typedef long long ll;
13 typedef double db;
14 const int N = 250000+100;
15 const int M = 500000+100;
16 const db EPS = 1e-9;
17 using namespace std;
18
19 int n,tot1,tot2,head1[N],head2[N],m,dfn[N],d[N],f[N][20],t,idx,idq[N],top,stk[N],vis[N
       ];
```

```
20 //idq —— 存取有效节点, top —— stk栈顶, vis —— 表示这个节点是否是有效节点
21 ll me[N];
22 struct Node{
23     int to,next; ll w;
24 }e1[M],e2[M];
25
26 void add1(int x,int y,ll cc){
27     e1[++tot1].to = y, e1[tot1].next = head1[x], e1[tot1].w = cc, head1[x] = tot1;
28 }
29 void add2(int x,int y,ll cc){
30     e2[++tot2].to = y, e2[tot2].next = head2[x], e2[tot2].w = cc, head2[x] = tot2;
31 }
32 bool cmp(int a,int b){return dfn[a] < dfn[b];}
33
34 void bfs() //LCA部分
35 {
36     queue<int> q;
37     while(q.size()) q.pop();
38     q.push(1); d[1] = 1;    //把1当做树根
39     while(q.size())
40     {
41         int x = q.front(); q.pop();
42         for(int i = head1[x]; i ;i = e1[i].next){
43             int y = e1[i].to;
44             if(d[y]) continue;
45             d[y] = d[x]+1;
46             f[y][0] = x;  //y走2^0步到达x
47             for(int j = 1; j <= t;j++)
48                 f[y][j] = f[f[y][j-1]][j-1];
49             q.push(y);
50         }
51     }
52 }
53
54 int LCA(int x,int y) //LCA部分
55 {
56     if(d[x] > d[y]) swap(x,y);
57     for(int i = t; i >= 0; i--)
58         if(d[f[y][i]] >= d[x]) y = f[y][i];  //往上追溯, 直至y和x位于同一深度
59     if(x == y) return x;  //如果已经找到了, 就返回x
60     for(int i = t; i >= 0; i--)
61         if(f[x][i] != f[y][i]) x = f[x][i], y = f[y][i];  //x和y同时往上走, 一直到x和y恰好为
    lca的子节点
62     return f[x][0];  //x和y共同的根节点就是lca
63 }
64
65 void dfs(int x,int fa){
66     dfn[x] = ++idx;
67     for(int i = head1[x]; i; i = e1[i].next){
68         int y = e1[i].to;
69         if(y == fa) continue;
70         if(x == 1) me[y] = e1[i].w;
71         else me[y] = min(me[x],e1[i].w);
72         dfs(y,x);
73     }
74 }
75
76 void insert(int u){ //构成虚树
77     if(top == 1) {stk[++top] = u; return;}
```

```
78      int lca = LCA(u,stk[top]); //与栈顶比较
79      if(lca == stk[top]) {stk[++top] = u; return;}
80      while(top > 1 && dfn[lca] <= dfn[stk[top-1]]){
81          add2(stk[top-1],stk[top],0); --top;
82      }
83      if(lca != stk[top]){
84          add2(lca,stk[top],0); stk[top] = lca;
85      }
86      stk[++top] = u;
87  }
88
89  ll DP(int x){ //使当前节点与子树中非有效节点不连通的最小代价
90      ll cost = 0;
91      for(int i = head2[x]; i; i = e2[i].next){
92          int y = e2[i].to;
93          cost += min(me[y],DP(y));
94      }
95      head2[x] = 0;
96      if(vis[x]){
97          vis[x] = 0;
98          return me[x];
99      }
100     else return cost;
101 }
102
103 int main()
104 {
105     scanf("%d",&n);
106     t = (int)(log(n)/log(2))+1;
107     rep(i,1,n-1){
108         int xx,yy; ll cc;
109         scanf("%d%d%lld",&xx,&yy,&cc);
110         add1(xx,yy,cc); add1(yy,xx,cc);
111     }
112     bfs(); dfs(1,-1);
113     scanf("%d",&m);
114     rep(i,1,m){
115         tot2 = 1;
116         int sz; scanf("%d",&sz);
117         rep(j,1,sz){
118             scanf("%d",&idq[j]); vis[idq[j]] = 1;
119         }
120         sort(1+idq,1+idq+sz,cmp);
121         top = 0; stk[++top] = 1; //加入树根
122         rep(j,1,sz) insert(idq[j]); //构建虚树
123         while(top > 1){
124             add2(stk[top-1],stk[top],0); top--;
125         }
126         // LOG1("i",i);
127         cout << DP(1) << endl;
128     }
129     return 0;
130 }
```

4.16 十字链表

题意：给出一个 $n*m$ 的矩阵，q 次操作，每次操作在矩阵中指定两个不重叠且不接触的小矩阵，将两个矩阵中的对应元素互换。在 q 次操作之后，输出最后的结果矩阵。$(2 \le n,m \le 10^3, 1 \le q \le 10^4)$

思路：此题是子矩阵交换，我们可以先考虑一维状态下的子序列交换，区间 $[a,b]$ 与 $[c,d]$ 交换，如下图所示。每个节点存储右指针，只需将节点 [a-1] 与 [c-1] 以及 [b] 与 [d] 交换右指针即可。此处需注意，如果两个区间重叠或相互接触，则会发生错误，可以自行手动模拟一下。

知道了一维情况下的操作，我们可以考虑二维情况下如何操作。首先每个节点需要维护向右与向下的指针，然后我们来考虑下图情况，两个子矩形相互交换元素。我们可以发现决定这个矩形具体所在位置只是 A1、A3 区域的向下指针与 A2、A4 区域的向右指针。

因此我们只需将 A1、A3 与 B1、B3 的向下指针进行交换，A2、A4 与 B2、B4 的向右指针进行交换即可完成题目要求。注意如果两个矩形有可能相交或接触，则此算法不可行。

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <algorithm>
#define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
#define rep(i,a,b) for(int i = a; i <= b; i++)
#define LOG1(x1,x2) cout << x1 << ": " << x2 << endl;
#define LOG2(x1,x2,y1,y2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << endl
    ;
#define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2
    << " , " << z1 << ": " << z2 << endl;
typedef long long ll;
typedef double db;
const int N = 2*1e6+100;
const int M = 1e5+100;
const db EPS = 1e-9;
using namespace std;

int n,m,q;
struct Node{
    int r,d,v;
}t[N];

int Pos(int x,int y){
    x++, y++;
    return (x-1)*(m+1)+y;
}

int main()
{
    scanf("%d%d%d",&n,&m,&q);
    rep(i,1,n)
        rep(j,1,m) scanf("%d",&t[Pos(i,j)].v);
    //一共n+1行，m+1列，从(0,0)开始编号
    rep(i,0,n)
        rep(j,0,m){
            t[Pos(i,j)].r = Pos(i,j+1);
            t[Pos(i,j)].d = Pos(i+1,j);
        }
    rep(i,1,q){
        int a,b,c,d,h,w;
        scanf("%d%d%d%d%d%d",&a,&b,&c,&d,&h,&w);
        int x = 1,y = 1,u,v;
        rep(j,1,a-1) x = t[x].d;
        rep(j,1,b-1) x = t[x].r;
        rep(j,1,c-1) y = t[y].d;
        rep(j,1,d-1) y = t[y].r;
```

```
46          u = x, v = y; //先往下再往右，左下周长部分
47          rep(j,1,h){
48              u = t[u].d, v = t[v].d;
49              swap(t[u].r,t[v].r);
50          }
51          rep(j,1,w){
52              u = t[u].r, v = t[v].r;
53              swap(t[u].d,t[v].d);
54          }
55          u = x, v = y; //先往右再往下，右上周长部分
56          rep(j,1,w){
57              u = t[u].r, v = t[v].r;
58              swap(t[u].d,t[v].d);
59          }
60          rep(j,1,h){
61              u = t[u].d, v = t[v].d;
62              swap(t[u].r,t[v].r);
63          }
64      }
65      int x = 1;
66      x = t[x].d, x = t[x].r;
67      rep(i,1,n){
68          int y = x;
69          rep(j,1,m){
70              printf("%d",t[y].v);
71              y = t[y].r;
72              if(j == m) printf("\n");
73              else printf(" ");
74          }
75          x = t[x].d;
76      }
77      return 0;
78  }
```

5 图论

5.1 最短路

5.1.1 Dijkstra

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <string>
#include <queue>
#include <algorithm>
#define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
#define rep(i,a,b) for(int i = a; i <= b; i++)
#define LOG1(x1,x2) cout << x1 << ": " << x2 << endl;
#define LOG2(x1,x2,y1,y2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << endl;
#define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << " , " << z1 << ": " << z2 << endl;
typedef long long ll;
typedef double db;
const int N = 5*1e6+100;
const int M = 5*1e6+100;
const ll inf = 5*1e16;
const db EPS = 1e-9;
using namespace std;

struct Edge{
    int to,next;
    ll w;
}e[M];
int head[N],tot,n,vis[N];
ll c[N],dis[N];

void add(int x,int y,ll w){
    e[++tot].to = y, e[tot].next = head[x], e[tot].w = w, head[x] = tot;
}

priority_queue< pair<ll,int> > q;

ll dijkstra(int s)
{
    while(q.size()) q.pop();
    memset(vis,0,sizeof vis);
    rep(i,0,2*n+2) dis[i] = inf;
    dis[s] = 0;
    q.push(make_pair(0ll,s));
    while(q.size())
    {
        int x = q.top().second;
        q.pop();
        if(vis[x]) continue;
        vis[x] = 1; //只有从队列中弹出时才是更新好了这个点，此时才可以更新vis，不能在加入的时候就更新vis
        for(int i = head[x]; i; i = e[i].next)
        {
            int y = e[i].to;
            ll z = e[i].w;
            if(dis[y] > (ll)(dis[x] + z)){
```

```
                dis[y] = dis[x]+z;
                q.push(make_pair(-dis[y],y));
            }
        }
    }
    if(dis[n*2+2] == inf) return -1;
    return dis[2*n+2];
}

int main()
{
    __; cin >> n; tot = 1;
    rep(i,1,n) cin >> c[i];
    string b1,b2;
    int r1,r2;
    rep(i,1,n){
        string s1,s2;
        cin >> s1; s2 = s1;
        reverse(s2.begin(),s2.end());
        if(i == 1){
            b1 = s1, b2 = s2;
            r1 = i, r2 = i+n;
            add(2*n+1,r1,0);
            add(2*n+1,r2,c[1]);
        }
        else{
            if(b1 <= s1) add(r1,i,0ll);
            if(b1 <= s2) add(r1,i+n,c[i]);
            if(b2 <= s1) add(r2,i,0ll);
            if(b2 <= s2) add(r2,i+n,c[i]);
            b1 = s1, b2 = s2;
            r1 = i, r2 = i+n;
        }
    }
    add(n,2*n+2,0);
    add(2*n,2*n+2,0);
    ll x1 = dijkstra(2*n+1);
    cout << x1 << endl;
    return 0;
}
```

5.1.2 Dijkstra 记录路径

```
#include <cstdio>
#include <iostream>
#include <stack>
#include <cstring>
using namespace std;
const int N = 100+10;
const int inf = 0x3fffffff;

int e[N][N],dis[N],path[N],vis[N];
int n,m;

void dijkstra(int x)
{
    //初始化dis数组
    for(int i = 1;i <= n;i++)
```

```
16          dis[i] = e[x][i];
17
18      memset(vis,0,sizeof(vis));
19      vis[x] = 1;
20      for(int i = 1;i <= n-1;i++)
21      {
22          int minn = inf;
23          int u;
24          for(int j = 1;j <= n;j++)
25          {
26              if(!vis[j] && dis[j] < minn)
27              {
28                  minn = dis[j];
29                  u = j;
30              }
31          }
32          vis[u] = 1;
33          for(int j = 1;j <= n;j++)
34          {
35              if(e[u][j] < inf)
36              {
37                  if(e[u][j]+dis[u] < dis[j])
38                  {
39                      dis[j] = e[u][j]+dis[u];
40                      path[j] = u;    //记录该点是被哪一个点松弛的
41                  }
42              }
43          }
44      }
45  }
46
47  void print(int x,int y)   //x为起点，y为终点
48  {
49      stack<int> q;
50      int tmp = y;
51      while(path[y] != -1)
52      {
53          q.push(y);
54          y = path[y];
55      }
56      q.push(y);
57
58      //打印路径
59      printf("%d=>%d, length:%d, path: %d ",x,tmp,dis[tmp],x);
60      while(!q.empty())       //先进后出,获得正序
61      {
62          printf("%d ",q.top());//打印堆的头节点
63          q.pop();            //将堆的头节点弹出
64      }
65      printf("\n");
66  }
67
68  int main()
69  {
70      cin>>n>>m;
71      memset(path,-1,sizeof(path));
72      //把图初始化
73      for(int i = 1;i <= n;i++)
74      {
```

```
        for(int j = 1;j <= n;j++)
        {
            if(i == j) e[i][j] = 0;
            else e[i][j] = inf;
        }
    }
    //读入边
    for(int i = 1;i <= m;i++)
    {
        int t1,t2,t3;
        cin>>t1>>t2>>t3;
        e[t1][t2] = t3;
    }
    dijkstra(1);
    cout<<"n:"<<n<<endl;
    print(1,n);
    return 0;
}
```

5.1.3 分层图

题意:
给一张有向图，给一个 k，可以将图中 k 条边的边权变为 0，求从 1->n 的最短路径

解法:
由于 k<=10，由此考虑分层图做法
给每个点建 10 个分身，dis[i][j] 表示从城市 1 到达城市 i，一共改了 j 条边的最短路径
其余操作和正常最短路求法一致，更新的时候有两种更新方法。假设队列弹出的节点是 dis[i][j]，y 为 i 可达到的点
则用 dis[i][j] 去更新 dis[y][j] 和 dis[y][j+1]，相当于扩点操作，其余操作和 dij 模板一致
如此操作即可求出答案

```
#include <cstdio>
#include <iostream>
#include <algorithm>
#include <cstring>
#include <queue>
#define rep(i,a,b) for(int i = a;i <= b;i++)
using namespace std;
const int N = 1e5+100;
const int M = 5*1e5+10;
typedef long long ll;
const ll inf = 1e15;

struct Edge{
    int to,next,w;
}e[M];
int n,m,k,head[N],tot;
ll dis[N][15];
int vis[N][15];

void init()
{
    tot = 1;
    rep(i,0,n) head[i] = 0;
}

void add(int x,int y,int z)
```

```
{
    e[++tot].to = y, e[tot].next = head[x], head[x] = tot, e[tot].w = z;
}

struct Point{
    ll ans;
    int u,cnt;
}thp;

bool operator < (Point a,Point b)
{
    return a.ans > b.ans;
}

priority_queue<Point> q;

void dijstra(int s)
{
    while(q.size()) q.pop();
    thp.ans = 0, thp.u = s, thp.cnt = 0;
    q.push(thp);
    rep(i,1,n)
        rep(j,0,k) vis[i][j] = 0;
    rep(i,1,n)
        rep(j,0,k) dis[i][j] = inf;
    dis[s][0] = 0;
    while(q.size())
    {
        thp = q.top();
        q.pop();
        int u = thp.u;
        int x = thp.cnt;
        if(vis[u][x]) continue;
        vis[u][x] = 1;
        for(int i = head[u]; i ; i = e[i].next)
        {
            int y = e[i].to;
            if(!vis[y][x]){
                if(dis[y][x] > dis[u][x]+e[i].w){
                    dis[y][x] = dis[u][x] + e[i].w;
                    thp.u = y, thp.ans = dis[y][x], thp.cnt = x;
                    q.push(thp);
                }
            }
            if(!vis[y][x+1] && (x < k)){
                if(dis[y][x+1] > dis[u][x]){
                    dis[y][x+1] = dis[u][x];
                    thp.u = y, thp.ans = dis[y][x+1], thp.cnt = x+1;
                    q.push(thp);
                }
            }
        }
    }
}

int main()
{
    int T;
    scanf("%d",&T);
```

```
    while(T--)
    {
        init();
        scanf("%d%d%d",&n,&m,&k);
        rep(i,1,m)
        {
            int x,y,z;
            scanf("%d%d%d",&x,&y,&z);
            add(x,y,z);
        }
        dijstra(1);
        printf("%lld\n",dis[n][k]);
    }
    return 0;
}
```

5.1.4 spfa

```
//最短路 —— spfa
#include <cstdio>
#include <iostream>
#include <queue>
#include <algorithm>
#include <cstring>
using namespace std;

const int maxn = 1000+10;
const int maxm = 4000+10;
const int inf = 0x3fffffff;

struct edge{
    int u,v,w;
}e[maxm];
int book[maxn],first[maxn],next[maxm];
//book用来标记       first[i]指以节点i为出发点的边的编号
int dis[maxn];
int n,m;
queue<int>q;         //用以保存节点的队列

//x为要求从哪一个点出发
void spfa(int x)
{
    memset(book,0,sizeof(book));
    for(int i = 1;i <= n;i++) dis[i] = inf;   //距离赋成无穷大
    while(!q.empty()) q.pop();
    q.push(x);
    book[x] = 1;
    dis[x] = 0;
    while(!q.empty())
    {
        int k = first[q.front()];
        while(k!=-1)   //遍历图
        {
            if(dis[e[k].v] > dis[e[k].u]+e[k].w)   //用队列中的边去松弛两端的点
            {
                dis[e[k].v] = dis[e[k].u]+e[k].w;  //松弛成功
                if(book[e[k].v] == 0)   //把松弛成功的点加入队列
                {
```

```
                    q.push(e[k].v);
                    book[e[k].v] = 1;    //标记这个点在队列中了
                }
            }
            k = next[k];  //遍历这个点的下一条边
        }
        book[q.front()] = 0;    //将这个点出队列
        q.pop();
    }
}

int main()
{
    cin>>m>>n;
    memset(first,-1,sizeof(first));
    for(int i = 1;i <= m;i++)
    {
        scanf("%d%d%d",&e[i].u,&e[i].v,&e[i].w);
        //双向边的保存
        e[m+i].u = e[i].v;
        e[m+i].v = e[i].u;
        e[m+i].w = e[i].w;
        next[i] = first[e[i].u];   //表示e[i].u这个点当前这条边的上一条边
        first[e[i].u] = i;         //表示e[i].u这个点现在的边
        next[m+i] = first[e[i+m].u];
        first[e[i+m].u] = i+m;
    }
    spfa(1);
    cout<<dis[n]<<endl;
    return 0;
}
```

5.1.5 spfa 判断负环

```
//spfa判断负环
#include <cstdio>
#include <iostream>
#include <queue>
#include <cstring>
using namespace std;
const int maxn = 105;
const int inf = 0x3fffffff;

bool p[maxn][maxn];
bool f[maxn][maxn];
int n, pow[maxn], book[maxn], cnt[maxn], dis[maxn];

void Floyd() //判连通
{
    for(int k = 1;k <= n;k++)
        for(int i = 1;i <= n;i++)
            for(int j = 1;j <= n;j++)
                f[i][j] = f[i][j] || (f[i][k] && f[k][j]);
}

bool spfa(int x)
{
    memset(book,0,sizeof(book));
```

```
25      memset(cnt,0,sizeof(cnt));
26      memset(dis,0,sizeof(dis));
27      queue<int>q;
28      q.push(x);
29      book[x] = 1;
30      dis[x] = 100;
31      cnt[x]++;
32      while(!q.empty())
33      {
34          x = q.front();
35          q.pop();
36          if(cnt[x] > n) continue;   //如果一个点进入队列次数>n，就不能用这个点去松弛，否则会TLE
37          for(int i = 1;i <= n;i++)
38          {
39              if(p[x][i] && dis[i] < pow[x]+dis[x] && book[i] == 0 && (pow[x]+dis[x]) >
    0)
40              {
41                  q.push(i);
42                  dis[i] = pow[x]+dis[x];
43                  cnt[i]++;
44                  book[i] = 1;
45                  if(cnt[i] >= n) dis[i] = inf;   //判断负环
46              }
47          }
48          book[x] = 0;
49      }
50      return dis[n] > 0;
51  }
52
53  int main()
54  {
55      while(~scanf("%d",&n) && n!=-1)
56      {
57          memset(p,0,sizeof(p));
58          memset(f,0,sizeof(f));
59          for(int i = 1;i <= n;i++)
60          {
61              int k;
62              scanf("%d%d",&pow[i],&k);
63              for(int j = 1;j <= k;j++)
64              {
65                  int tmp;
66                  scanf("%d",&tmp);
67                  p[i][tmp] = true;
68                  f[i][tmp] = true;
69              }
70          }
71          Floyd();
72          if(spfa(1) && f[1][n])
73              printf("winnable\n");
74          else
75              printf("hopeless\n");
76      }
77      return 0;
78  }
```

5.2 最小生成树

```cpp
//最小生成树 ——Kruscal
#include <cstdio>
#include <iostream>
#include <algorithm>
#include <cstring>
using namespace std;

const int maxn = 1000+10;
const int maxm = 4000+10;

struct edge{
    int u,v,w;
}e[maxn];
int n,m,sum;   //sum为总共的路程
int p[maxn]; //记录每个点的祖先

bool cmp(edge e1,edge e2)
{
    return e1.w < e2.w;
}

int find(int x)
{
    return p[x]==x?x:p[x]=find(p[x]);
}

bool merge(int x,int y)
{
    int t1 = find(x);
    int t2 = find(y);
    if(t1!=t2)
    {
        p[t1] = t2;
        return true;
    }
    return false;
}

int main()
{
    scanf("%d%d",&n,&m);
    for(int i = 1;i <= m;i++)
    {
        scanf("%d%d%d",&e[i].u,&e[i].v,&e[i].w);
    }
    sort(e+1,e+m+1,cmp);
    for(int i = 1;i <= n;i++) p[i] = i;

    //Kruscal算法核心部分
    int count = 0;  //最小生成树已经有几条边了
    sum = 0;
    int maxx = -1;
    int minn = 10010;
    for(int i = 1;i <= m;i++)
    {
        //判断一条边的两个顶点是否已经连通，即判断是否已在同一个集合中
        if(merge(e[i].u,e[i].v))
        {
            count++;
```

```
            sum = sum+e[i].w;
            maxx = max(maxx,e[i].w);
            minn = min(minn,e[i].w);
        }
        if(count == n-1) break;
    }
    cout<<sum<<endl;
    cout<<maxx-minn<<endl;

    return 0;
}
```

5.3 最近公共祖先

5.3.1 LCA (dfs 版本)

```
#include <cstdio>
#include <iostream>
#include <algorithm>
#include <cstring>
#include <queue>
#include <cmath>
#define LOG1(x1,x2) cout << x1 << ": " << x2 << endl;
#define LOG2(x1,x2,y1,y2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << endl
    ;
#define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2
    << " , " << z1 << ": " << z2 << endl;
#define rep(i,a,b) for(int i = a;i <= b;i++)
using namespace std;
const int N = 5*1e5+100;

int f[N][25],d[N],dis[N];  //f[x][k]:表示点x向根节点走2^k步到达的点
//d[x]:表示点x在图中的深度  dis[x]:表示根节点到点x的距离
struct Edge{
    int to,next;
}e[2*N];    //树的边为n-1，此处是因为无向边，所以正反各一条
int head[N],tot,t,n,m,s; //n个点，m次询问

void init()
{
    tot = 1;
    memset(head,0,sizeof head);
    memset(d,0,sizeof d);
}

void add(int x,int y)
{
    e[++tot].to = y; e[tot].next = head[x]; head[x] = tot;
}

void dfs(int u,int fa)
{
    d[u]=d[fa]+1; f[u][0]=fa;
    for(int i=1;(1<<i)<=d[u];i++)
        f[u][i]=f[f[u][i-1]][i-1];
    for(int i=head[u]; i; i=e[i].next){
        int v=e[i].to;
        if(v!=fa) dfs(v,u);
    }
```

```
}

int lca(int x,int y)
{
    if(d[x] > d[y]) swap(x,y);
    for(int i = t; i >= 0; i--)
        if(d[f[y][i]] >= d[x]) y = f[y][i];  //往上追溯，直至y和x位于同一深度
    if(x == y) return x;  //如果已经找到了，就返回x
    for(int i = t; i >= 0; i--)
        if(f[x][i] != f[y][i]) x = f[x][i], y = f[y][i];  //x和y同时往上走，一直到x和y恰好为lca的子节点
    return f[x][0];  //x和y共同的根节点就是lca
}

int main()
{
    scanf("%d%d%d",&n,&m,&s);
    t = (int)(log(n)/log(2))+1;
    init();
    rep(i,1,n-1){
        int x,y;
        scanf("%d%d",&x,&y);
        add(x,y), add(y,x);
    }
    dfs(s,0);
    for(int i = 1;i <= m;i++)
    {
        int x,y;
        scanf("%d%d",&x,&y);
        printf("%d\n",lca(x,y));
    }
    return 0;
}
```

5.3.2 LCA (bfs 版本)

```
#include <cstdio>
#include <iostream>
#include <algorithm>
#include <cstring>
#include <queue>
#include <cmath>
#define rep(i,a,b) for(int i = a;i <= b;i++)
using namespace std;
const int N = 50010;

int f[N][20],d[N],dis[N];  //f[x][k]:表示点x向根节点走2^k步到达的点
//d[x]:表示点x在图中的深度  dis[x]:表示根节点到点x的距离
struct Edge{
    int to,next,w;
}e[2*N];        //树的边为n-1，此处是因为无向边，所以正反各一条
int head[N],tot,t,n,m; //n个点，m次询问

void init()
{
    tot = 1;
    memset(head,0,sizeof head);
    memset(d,0,sizeof d);
```

```
}

void add(int x,int y,int z)
{
    e[++tot].to = y; e[tot].next = head[x]; e[tot].w = z; head[x] = tot;
}

void bfs()
{
    queue<int> q;
    while(q.size()) q.pop();
    q.push(1); d[1] = 1; dis[1] = 0;    //把1当做树根
    while(q.size())
    {
        int x = q.front(); q.pop();
        for(int i = head[x]; i ;i = e[i].next){
            int y = e[i].to;
            if(d[y]) continue;
            d[y] = d[x]+1;
            dis[y] = dis[x]+e[i].w; //dist[y]:从1到y的距离
            f[y][0] = x;  //y走2^0步到达x
            for(int j = 1; j <= t;j++)
                f[y][j] = f[f[y][j-1]][j-1];
            q.push(y);
        }
    }
}

/*
    这里有一个非常容易错的点，因为每一个点的父节点初始化为0，所以如果你的点从0-n编号的话，一定要注意
    bfs的时候将0作为根节点
    否则如果用1作为根节点的话，你会发现1的父节点是0，那么程序就开始出错了!

    所以更好的方法是以后给点编号的时候，统一用1-n编号，不要用0编号，否则极易出错

    从1-n编号的话，可以从任意起点拉起一棵树，但是要注意题目中n的最小值
*/

int lca(int x,int y)
{
    if(d[x] > d[y]) swap(x,y);
    for(int i = t; i >= 0; i--)
        if(d[f[y][i]] >= d[x]) y = f[y][i];  //往上追溯，直至y和x位于同一深度
    if(x == y) return x;  //如果已经找到了，就返回x
    for(int i = t; i >= 0; i--)
        if(f[x][i] != f[y][i]) x = f[x][i], y = f[y][i];  //x和y同时往上走，一直到x和y恰好为
    lca的子节点
    return f[x][0];  //x和y共同的根节点就是lca
}

int main()
{
    int T;
    scanf("%d",&T);
    while(T--)
    {
        scanf("%d%d",&n,&m);
        t = (int)(log(n)/log(2))+1;
        init();
```

```
        rep(i,1,n-1){
            int x,y,z;
            scanf("%d%d%d",&x,&y,&z);
            add(x,y,z), add(y,x,z);
        }
        bfs();
        for(int i = 1;i <= m;i++)
        {
            int x,y;
            scanf("%d%d",&x,&y);
            printf("%d\n",dis[x]+dis[y]-2*dis[lca(x,y)]);
        }
    }
    return 0;
}
```

5.4 拓扑排序

题意:
一个 n*n 的网格图，每行每列各涂一次，每行每列均有 26 种涂法，输出涂色方案。

思路:
可以发现本题有个性质，最后涂的行或列，一定只有一个元素。
因此开一个结构体记录每行每列，各个颜色的涂色情况，以及该行或列一共有几种不同的颜色。
然后类似于跑拓扑排序，将只有一种颜色的行列加入队列，再依次撤销。
将新的颜色数变为 1 的行列加入队列，即可完成本题。

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <queue>
#include <algorithm>
#define rep(i,a,b) for(int i = a; i <= b; i++)
#define LOG1(x) cout << "x: " << x << endl;
#define LOG2(x,y) cout << "x: " << x << ", y: " << y << endl;
typedef long long ll;
typedef double db;
const db EPS = 1e-9;
using namespace std;
const int N = 3000+100;

int n;
char mp[N][N];
struct Node{
    int total;
    int num[30];
}t[2*N];
int ans[2*N][4],tot;
int vis[2*N];

void solve()
{
    tot = 0;
    queue<int> q;
    while(q.size()) q.pop();
    rep(i,1,n*2) {
        if(t[i].total == 1) q.push(i);
```

```
31          else if(t[i].total == 0) ans[++tot][1] = i, ans[tot][2] = 0, vis[i] = 1;
32      }
33      while(q.size()){
34          int x = q.front();
35          q.pop();
36          if(vis[x]) continue;
37          t[x].total = 0;
38          ans[++tot][1] = x;
39          vis[x] = 1;
40          rep(i,0,27)
41              if(t[x].num[i] >= 1) ans[tot][2] = i;
42          int pp = ans[tot][2];
43          if(x <= n){
44              rep(i,n+1,n*2){
45                  if(mp[x][i-n]-'a' != pp) continue;
46                  t[i].num[pp]--;
47                  if(t[i].num[pp] == 0){
48                      t[i].total--;
49                      if(t[i].total == 1) q.push(i);
50                  }
51              }
52          }
53          else{
54              rep(i,1,n){
55                  if(mp[i][x-n]-'a' != pp) continue;
56                  t[i].num[pp]--;
57                  if(t[i].num[pp] == 0){
58                      t[i].total--;
59                      if(t[i].total == 1) q.push(i);
60                  }
61              }
62          }
63      }
64      for(int i = 2*n; i >= 1; i--)
65      {
66          int x = ans[i][1], y = ans[i][2];
67          if(x <= n) printf("h %d ",x);
68          else printf("v %d ",x-n);
69          printf("%c\n",'a'+y);
70      }
71  }
72
73  int main()
74  {
75      scanf("%d",&n);
76      rep(i,1,n) scanf("%s",mp[i]+1);
77      rep(i,1,n)
78          rep(j,1,n){
79              if(mp[i][j] == '?') continue;
80              int x = mp[i][j]-'a';
81              t[i].num[x]++;
82              if(t[i].num[x] == 1) t[i].total++;
83              t[j+n].num[x]++;
84              if(t[j+n].num[x] == 1) t[j+n].total++;
85          }
86      solve();
87      return 0;
88  }
```

5.5 欧拉路

题意：
给出 m 条边，求出一条欧拉路，起点任意，终点任意，每条边只经过一次。要求给出的欧拉路字典序最小。($1 \leq m \leq 1024, 1 \leq n \leq 500$)

思路：
先总结一下 '有向图、无向图求欧拉路与欧拉回路的性质 '。
无向图：有且仅有两个点度数为奇数则有欧拉路，所有点度数均为偶数则有欧拉回路。
有向图：所有点入度 = 出度则有欧拉回路。有且仅有两个点入度不等于出度，且起点出度比入度大 1 ，终点入度比出度大 1 则有欧拉路。

算法——Hierholzer (解决无向图、有向图、欧拉路、欧拉回路问题)

选一个点 x 为起点，存在边 $<x,y>$，则删去边 $<x,y>$，若为无向图还需删除 $<y,x>$。若无边可走，则将 x 加入结果栈。最后输出结构栈即可。

回到此题，要求找到字典序最小的欧拉路。因此选择一个编号最小的奇数点进行递归，递归过程中优先走字典序更小的点，可以用 $multiset$ 维护。

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <set>
#include <stack>
#include <algorithm>
#define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
#define rep(i,a,b) for(int i = a; i <= b; i++)
#define LOG1(x1,x2) cout << x1 << ": " << x2 << endl;
#define LOG2(x1,x2,y1,y2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << endl
    ;
#define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2
    << " , " << z1 << ": " << z2 << endl;
typedef long long ll;
typedef double db;
const int N = 500+100;
const int M = 1e5+100;
const db EPS = 1e-9;
using namespace std;

multiset<int> st[N];
stack<int> stk;
int m,deg[N];

void dfs(int x){
    while(st[x].size()){
        int y = (*st[x].begin());
        st[x].erase(st[x].begin());
        st[y].erase(st[y].find(x));
        dfs(y);
    }
    stk.push(x);
}

int main()
{
    rep(i,0,500) st[i].clear();
    while(stk.size()) stk.pop();
```

```
    scanf("%d",&m);
    rep(i,1,m){
        int xx,yy; scanf("%d%d",&xx,&yy);
        st[xx].insert(yy);
        st[yy].insert(xx);
        deg[xx]++, deg[yy]++;
    }
    int x = -1;
    rep(i,0,500) if(deg[i]%2){x = i; break;}
    if(x == -1) x = 1;
    dfs(x);
    while(stk.size()){
        printf("%d\n",stk.top());
        stk.pop();
    }
    return 0;
}
```

5.6 双连通分量

5.6.1 边双缩点以及求割边

求边双连通分量、割边（e-DCC）
将每个 e-DCC 看成一个节点，进行缩点
求边双连通分量——即该分量中不存在割边

边双连通图的条件——当且仅当任意一条边都包含在至少一个简单环中
求边双连通分量的原理：
先求出无向图中所有的桥，把桥都删除后，无向图会分成若干个连通块，每一个连通块就是一个“边双连通分量”

```
#include <cstdio>
#include <iostream>
#include <algorithm>
#include <cstring>
#define rep(i,a,b) for(int i = a;i <= b;i++)
using namespace std;
const int N = 1e5+10, M = 5*1e5+10;

struct Edge{
    int to,next;
}e[2*M],ec[2*M]; //双向边，ec为缩点之后的图【森林/树】
int head[N],dfn[N],low[N],headc[N];  //dfn:记录该点的dfs序      low:记录该点所能达到的最小dfn值
//hc:缩点之后记录每一个点所连接的第一条边的边序
bool bridge[2*M];   //记录该边是否为割边
int tot,n,m,num,dcc,tc; //tot记录边序，num记录dfs序，dcc记录有多少个边双连通分量
//tc:缩点之后记录边序
int c[N];  //c[x]表示节点x所属的“边双连通分量”的编号

void init()
{
    num  = 0; tot = 1; dcc = 0;
    rep(i,0,n) head[i] = dfn[i] = low[i] = c[i] = 0;
    rep(i,0,2*m+2) bridge[i] = 0;
}

void initc()
{
```

```
28      tc = 1; //缩点之后记录边序
29      // memset(headc,0,sizeof headc); //缩点之后记录每一个点所连接的第一条边的边序
30      rep(i,0,dcc) headc[i] = 0;
31  }
32
33  void add(int x,int y)
34  {
35      e[++tot].to = y; e[tot].next = head[x]; head[x] = tot;
36  }
37
38  void addc(int x,int y)  //对缩点之后的图进行加边操作
39  {
40      ec[++tc].to = y; ec[tc].next = headc[x]; headc[x] = tc;
41  }
42
43  void tarjan(int x,int in_edge)
44  {
45      dfn[x] = low[x] = ++num;
46      for(int i = head[x]; i ; i = e[i].next)
47      {
48          int y = e[i].to;
49          if(!dfn[y]){
50              tarjan(y,i);  //传入边(x,y)的序号
51              low[x] = min(low[x],low[y]);
52              if(low[y] > dfn[x])  //该边连接(x,y)，y点无法连接x点上面的点，因此该边是割边
53                  bridge[i] = bridge[i^1] = true;
54          }
55          else if(i != (in_edge^1)) //y点所连接的边 不能是(x,y)边的反向边
56              low[x] = min(low[x],dfn[y]);
57      }
58  }
59
60  void dfs(int x) //用于将图划分为多个边双连通分量
61  {
62      c[x] = dcc;
63      for(int i = head[x]; i ; i = e[i].next)
64      {
65          int y = e[i].to;
66          if(c[y] || bridge[i]) continue;  //如果点y已经属于别的强连通分量，或者边i是割边，则
        continue
67          dfs(y);
68      }
69  }
70
71  int main()
72  {
73      while(~scanf("%d%d",&n,&m))
74      {
75          init();
76          rep(i,1,m){
77              int x,y;
78              scanf("%d%d",&x,&y);
79              add(x,y); add(y,x);
80          }
81          rep(i,1,n)
82              if(!dfn[i]) tarjan(i,0); //将边序号传进去
83          rep(i,1,n)
84              if(!c[i]){ //如果i点未被标记过
85                  ++dcc;
```

```
86                  dfs(i);
87              }
88          initc(); //对缩点之后的图进行初始化 这里很重要! 不要忘记! ! !
89          for(int i = 2; i <= tot; i += 2){ //遍历所有边 2-tot
90              int x = e[i^1].to, y = e[i].to; //记录该边连接的两个端点
91              if(c[x] == c[y]) continue; //如果连接的两个点属于同一连通分量，则continue
92              //没必要用mp记录,不会重复，可以用ct记录有多少条边
93              addc(c[x],c[y]);  //用连通的分量的编号来代表整个连通分量，以此来进行缩点
94              addc(c[y],c[x]);
95          }
96          printf("缩点之后的森林，点数%d，边数%d(可能有重边)\n",dcc,tc/2);
97      /*  printf("There are %d e-DCCs.\n",dcc);
98          rep(i,1,n){
99              printf("%d belongs to DCC %d.\n",i,c[i]); //输出每个点属于的e-DCC
100         }*/
101     /*  for(int i = 2; i <= tot; i+=2)
102             if(bridge[i])
103                 printf("%d %d\n",e[i^1].to,e[i].to);  //该割边：(x,y)*/
104     }
105     return 0;
106 }
```

5.6.2 边双缩点 LCA

题意：
给出一个无向图，n 个点，m 条边，可能有重边与自环，也可能不连通。q 组询问，每组询问给出 3 个点，u、v、w，问是否存在两条路径不存在公共边，并且一条路径是 $v \to u$，另一条路径是 $w \to u$，存在输出 Yes，否则输出 No. $(1 \le n \le 10^5, 0 \le m \le 2 * 10^5, 1 \le q \le 10^5)$

思路：
既然是无向图上求两条互不相交的路径，比较直接的想法就是先求出边双连通分量进行缩点，然后在树上进行考虑。

求出边双连通分量之后，假如 u、v、w 三点在同一个双连通分量中，则答案必定为 Yes。若 v、w 在同一个双连通分量中，而 u 在另一个双连通分量中，则答案必定为 No。若 v 或 w 和 u 在同一个双连通分量中，另一个点不在其中，则答案也为 Yes。考虑完了一个和两个双连通分量的情况之后，我们来考虑三个的情况。

假如 u、v、w 分属于三个不同的双连通分量中，则需要进行分类。Yes 的情况只有两种，第一种情况是 v 和 w 都在 u 子树中，即 $lca(v,w) = u$ 即可，如图 (1)。第二种情况是 v、w 中有一个在 u 的子树中，另一个则不在。对于这种情况，我们先求出 $lca(v,w) = y$，再求出 $x_1 = lca(u,w), x_2 = lca(u,v)$，则 x_1 与 x_2 中一定有一个为 y，另一个为 u，才能输出 Yes，否则输出 No. 到此，这题分类讨论就结束了。
但是这一题还需要注意一些细节，因为图可能不连通，因此需要预先判断 u、v、w 三个点是否连通，如果不连通，直接输出 No. 还有一个细节，因为图可能是个森林，因此需要对每一个树进行 lca 处理。

```
1  #include <cstdio>
2  #include <iostream>
3  #include <cstring>
4  #include <cmath>
5  #include <queue>
6  #include <algorithm>
7  #define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
8  #define rep(i,a,b) for(int i = a; i <= b; i++)
9  #define LOG1(x1,x2) cout << x1 << ": " << x2 << endl;
10 #define LOG2(x1,x2,y1,y2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << endl
       ;
11 #define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2
       << " , " << z1 << ": " << z2 << endl;
12 typedef long long ll;
```

```
13 typedef double db;
14 const int N = 2*1e5+100;
15 const int M = 5*1e5+100;
16 const db EPS = 1e-9;
17 using namespace std;
18
19 struct Edge{
20     int to,next;
21 }e[M],ec[M];
22 int n,m,q,head[N],dfn[N],low[N],headc[N],dis[N],d[N],f[N][20],t;
23 bool bridge[M];
24 int tot,num,dcc,tc,DD[N];
25 int c[N];
26
27 void init(){
28     tot = tc = 1;
29     num = dcc = 0;
30     rep(i,0,n) head[i] = headc[i] = 0;
31     rep(i,0,2*m) bridge[i] = 0;
32     rep(i,0,n) c[i] = dfn[i] = low[i] = 0;
33     rep(i,0,n) dis[i] = d[i] = DD[i] = 0;
34 }
35
36 void add(int x,int y){
37     e[++tot].to = y; e[tot].next = head[x]; head[x] = tot;
38 }
39
40 void addc(int x,int y)
41 {
42     ec[++tc].to = y; ec[tc].next = headc[x]; headc[x] = tc;
43 }
44
45 void tarjan(int x,int in_edge)
46 {
47     dfn[x] = low[x] = ++num;
48     for(int i = head[x]; i ; i = e[i].next)
49     {
50         int y = e[i].to;
51         if(!dfn[y]){
52             tarjan(y,i);  //传入边(x,y)的序号
53             low[x] = min(low[x],low[y]);
54             if(low[y] > low[x])  //该边连接(x,y)，y点无法连接x点上面的点，因此该边是割边
55                 bridge[i] = bridge[i^1] = true;
56         }
57         else if(i != (in_edge^1)) //y点所连接的边 不能是(x,y)边的反向边
58             low[x] = min(low[x],dfn[y]);
59     }
60 }
61
62 void dfsD(int x,int hp){
63     DD[x] = hp;
64     for(int i = head[x]; i; i = e[i].next){
65         int y = e[i].to;
66         if(DD[y] == 0) dfsD(y,hp);
67     }
68 }
69
70 void dfs(int x) //用于将图划分为多个边双连通分量
71 {
```

```
72      c[x] = dcc;
73      for(int i = head[x]; i ; i = e[i].next)
74      {
75          int y = e[i].to;
76          if(c[y] || bridge[i]) continue;  //如果点y已经属于别的强连通分量，或者边i是割边，则
        continue
77          dfs(y);
78      }
79  }
80
81  void bfs(int s)
82  {
83      queue<int> q;
84      while(q.size()) q.pop();
85      q.push(s); d[s] = 1; dis[s] = 0;    //把1当做树根
86      while(q.size())
87      {
88          int x = q.front(); q.pop();
89          for(int i = headc[x]; i ;i = ec[i].next){
90              int y = ec[i].to;
91              if(d[y]) continue;
92              d[y] = d[x]+1;
93              dis[y] = dis[x]+1;  //dist[y]:从1到y的距离
94              f[y][0] = x;  //y走2^0步到达x
95              for(int j = 1; j <= t;j++)
96                  f[y][j] = f[f[y][j-1]][j-1];
97              q.push(y);
98          }
99      }
100 }
101
102 int lca(int x,int y)
103 {
104     if(d[x] > d[y]) swap(x,y);
105     for(int i = t; i >= 0; i--)
106         if(d[f[y][i]] >= d[x]) y = f[y][i];  //往上追溯，直至y和x位于同一深度
107     if(x == y) return x;  //如果已经找到了，就返回x
108     for(int i = t; i >= 0; i--)
109         if(f[x][i] != f[y][i]) x = f[x][i], y = f[y][i];  //x和y同时往上走，一直到x和y恰好为
        lca的子节点
110     return f[x][0];  //x和y共同的根节点就是lca
111 }
112
113
114 int main()
115 {
116     int _; scanf("%d",&_);
117     while(_--){
118         scanf("%d%d%d",&n,&m,&q);
119         init();
120         rep(i,1,m){
121             int xx,yy; scanf("%d%d",&xx,&yy);
122             add(xx,yy); add(yy,xx);
123         }
124         int ctt = 0;
125         rep(i,1,n)
126             if(!DD[i]) dfsD(i,++ctt);
127         rep(i,1,n)
128             if(!dfn[i]) tarjan(i,0); //将边序号传进去
```

```
129         rep(i,1,n)
130             if(!c[i]){ //如果i点未被标记过
131                 ++dcc;
132                 dfs(i);
133             }
134         rep(i,2,tot){
135             int x = e[i^1].to, y = e[i].to; //记录该边连接的两个端点
136             if(c[x] == c[y]) continue; //如果连接的两个点属于同一连通分量，则continue
137             addc(c[x],c[y]);  //用连通的分量的编号来代表整个连通分量，以此来进行缩点
138             addc(c[y],c[x]);
139         }
140         t = (int)(log(dcc+1)/log(2))+1;
141         rep(i,1,dcc)
142             if(d[i] == 0) bfs(i);
143         rep(i,1,q){
144             int u,v,w; scanf("%d%d%d",&u,&v,&w);
145             if(DD[u] == DD[v] && DD[u] == DD[w]){
146                 if(c[u] == c[v] && c[u] == c[w]) printf("Yes\n");
147                 else if(c[v] == c[w] && c[v] != c[u]) printf("No\n");
148                 else if(c[u] != c[v] && c[u] != c[w] && c[v] != c[w]){
149                     int y = lca(c[v],c[w]);
150                     if(y == c[u]) printf("Yes\n");
151                     else{
152                         int x1 = lca(c[v],c[u]);
153                         int x2 = lca(c[w],c[u]);
154                         if(x1 == y && x2 == c[u]) printf("Yes\n");
155                         else if(x2 == y && x1 == c[u]) printf("Yes\n");
156                         else printf("No\n");
157                     }
158                 }
159                 else if(c[v] == c[u] && c[w] != c[v]) printf("Yes\n");
160                 else if(c[w] == c[u] && c[w] != c[v]) printf("Yes\n");
161                 else printf("No\n");
162             }
163             else printf("No\n");
164         }
165     }
166     return 0;
167 }
```

5.6.3 点双缩点以及求割点

v-DCC 的缩点
由于一个割点可能属于多个 v-DCC，因此缩点之后，图中包含 p+t 个节点，p 个割点，t 个 v-DCC，将每个 v-DCC 和每个割点都作为新图中的节点
并将每个割点与包含它的所有 v-DCC 之间连边

此处需要注意：
缩点之后，所有的点双连通分量变成一个点，该点只与割点相连，不会与其他点双连通分量相连

无向图中求割点
x 是割点当且仅当搜索树上存在 x 的一个子节点 y，满足
dfn[x] <= low[y]
即 y 点无法访问 x 点之上的点

如果 x 不是根节点，则只需要一个点 y 即可

如果 x 是根节点，则需要两个点 y1、y2 存在，x 才能是割点，此时如果去掉 x，则 y1 与 y2 不连通

dfn: 记录该点的 dfs 序　low: 记录该点所能达到的最小 dfn 值

无向图中求点双连通分量
操作方法：
1. 当一个节点第一次被访问时，把该节点入栈。
2. 当割点判定法则中的条件 dfn[x] <= low[y] 成立时，无论 x 是否为根，都要：
(1) 从栈顶不断弹出节点，直至节点 y 被弹出。
(2) 刚才弹出的所有节点与节点 x 一起构成一个 v-DCC

割点：
若 x 为割点，则从图中删去节点 x 及所有与 x 相关联的边之后，G 分裂成了两个或两个以上不相连的子图

点双连通分量 (v-DCC) ——分量中无割点
注意: 点双连通分量与“删除割点后图中剩余的连通块”是不一样的概念
桥不属于任何 e-DCC，但是割点可能属于多个 v-DCC

无向连通图是点双连通图所需要满足的条件：【其中一个】
1. 图的顶点数不超过 2.
2. 图中任意两点都同时包含在至少一个简单环中。其中“简单环”指的是不自交的环，也就是我们通常画出的环

```cpp
#include <cstdio>
#include <iostream>
#include <algorithm>
#include <cstring>
#include <vector>
#define rep(i,a,b) for(int i = a;i <= b;i++)
using namespace std;
const int N = 1e5+10, M = 5*1e5+10;

struct Edge{
    int to,next;
}e[M*2],ec[M*2];  //双向边
int head[N],dfn[N],low[N],stack[N];  //dfn:记录该点的dfs序      low:记录该点所能达到的最小dfn值
//stack:手动模拟堆栈，先进先出
int n,m,tot,num,root,top,cnt; //top:用来记录stack的堆顶位置 cnt:记录一共有多少个点双连通分量
bool cut[N];  //记录该点是否为割点
vector<int> dcc[N]; //dcc[i]保存编号为i的v-DCC中的所有节点
int new_id[2*N];  //用于记录缩点之后的新图的各个点的编号
int headc[2*N],tc; //tc:记录新图的边的序号      headc:记录新图中每个点的第一条边

void init()
{
    tot = 1;  //记录第一条边
    num = 0;  //记录dfs序
    cnt = 0;  //记录点双连通分量个数
    top = 0;  //初始化堆栈
    memset(head,0,sizeof head);
    memset(dfn,0,sizeof dfn);
    memset(low,0,sizeof low);
    memset(cut,0,sizeof cut);
    rep(i,1,n){  //n个点
        dcc[i].clear();
    }
```

```
}

void initc()
{
    //cnt记录一共有多少个点双连通分量
    num = cnt;  //割点的编号从cnt+1开始，1-cnt的点为点双连通分量的编号
    tc = 1;
    memset(headc,0,sizeof headc);
}

void add(int x,int y)
{
    e[++tot].to = y; e[tot].next = head[x]; head[x] = tot;
}

void addc(int x,int y)
{
    ec[++tc].to = y; e[tot].next = headc[x]; headc[x] = tc;
}

void tarjan(int x)
{
    dfn[x] = low[x] = ++num;
    stack[++top] = x;   //当一个节点第一次被访问时，把该节点入栈
    if(x == root && head[x] == 0){  //孤立点-无边连接
        dcc[++cnt].push_back(x);    //孤立点自己就是一个点双连通分量
        return;
    }
    int flag = 0;   //记录存在几个子节点，使得low[y]>=dfn[x]，用于判断割点
    for(int i = head[x]; i ; i = e[i].next)
    {
        int y = e[i].to;
        if(!dfn[y]){
            tarjan(y);
            low[x] = min(low[x],low[y]);
            if(low[y] >= dfn[x]){
                flag++;
                if(x != root || flag > 1) cut[x] = true;    //x为割点
                cnt++;  //出现了一个新的点双连通分量
                int z;
                do{
                    z = stack[top--];   //x、y以及y之后的所有点构成一个点双连通分量
                    dcc[cnt].push_back(z);
                }while(z != y);
                dcc[cnt].push_back(x);
            }
        }
        else low[x] = min(low[x],dfn[y]);
    }
}

int main()
{
    while(~scanf("%d%d",&n,&m))
    {
        init();
        rep(i,1,m){
            int x,y;
            scanf("%d%d",&x,&y);
```

```
93              if(x == y) continue; //判断自环
94              add(x,y); add(y,x);
95          }
96          rep(i,1,n)
97              if(!dfn[i]) root = i, tarjan(i);  //因为原图可能不连通，因此需要对于每个点进行dfs
98          //以下是缩点操作
99          initc();
100         rep(i,1,n)
101             if(cut[i]) new_id[i] = ++num;   //cut:记录该点是否为割点，记录割点i在新图中的编号为num+1
102         //建新图，从每个v-DCC到它包含的所有割点连边
103         rep(i,1,cnt) //cnt表示点双连通分量个数
104             for(int j = 0; j < dcc[i].size(); j++)
105             {
106                 int x = dcc[i][j];
107                 if(cut[x]){ //如果x为割点，则将x与其所属的点强连通分量相连
108                     addc(i,new_id[x]);
109                     addc(new_id[x],i);
110                 }
111             //  else c[x] = i;  此处可有可无，此处的作用为c[x]表示x所在的强连通分量的编号
112             }
113         printf("缩点之后的森林，点数为%d，边数为%d\n",num,tc/2);    //num为新图中点数
114         printf("编号 1~%d 的为原图的v-DCC，编号 > %d 的为原图割点\n",cnt,cnt);  //cnt为点双连通分量个数
115         for(int i = 2;i < tc;i+=2)
116             printf("%d %d\n",ec[i^1].to,e[i].to);   //输出新图中的所有无向边
117
118     /*  rep(i,1,cnt){
119             printf("e-DCC #%d:",i);
120             for(int j = 0; j < dcc[i].size(); j++)
121                 printf(" %d",dcc[i][j]);  //将第i个点双连通分量中的点全部输出
122             puts("");
123         } */
124
125     /*  rep(i,1,n)
126             if(cut[i]) printf("%d ",i); //cut[i]标记i节点是否为割点
127         puts("are cut-vertexes"); */
128     }
129     return 0;
130 }
```

5.6.4 点双例题

题意：
n 个点，m 条边
如果把第 i 个点去掉，将有多少对点不能互通

```
1  #include <cstdio>
2  #include <iostream>
3  #include <algorithm>
4  #include <cstring>
5  #define rep(i,a,b) for(int i = a;i <= b;i++)
6  using namespace std;
7  const int N = 1e5+10, M = 5*1e5+10;
8  typedef long long ll;
9
10 struct Edge{
```

```
    int to,next;
}e[M*2];
int head[N],dfn[N],low[N],siz[N];   //dfn:记录该点的dfs序   low:记录该点所能达到的最小dfn值
ll ans[N];
bool cut[N];    //记录该点是否为割点
int n,m,tot,num,root;

void init()
{
    tot = 1;
    memset(head,0,sizeof head);
    memset(cut,0,sizeof cut);
    memset(low,0,sizeof low);
    memset(dfn,0,sizeof dfn);
    memset(ans,0,sizeof ans);
    memset(siz,0,sizeof siz);
}

void add(int x,int y)
{
    e[++tot].to = y; e[tot].next = head[x]; head[x] = tot;
}

void tarjan(int x)
{
    dfn[x] = low[x] = ++num;  //记录步长
    siz[x] = 1;  //记录该点之下有几个点
    int flag = 0,sum = 0;  //标记有几个子节点 low[y] >= dfn[x]
    for(int i = head[x]; i ; i = e[i].next)
    {
        int y = e[i].to;
        if(!dfn[y])
        {
            tarjan(y);  //每一次被dfs，都是因为该点未被访问过
            siz[x] += siz[y];
            low[x] = min(low[x],low[y]);
            if(low[y] >= dfn[x]){
                flag++; //根节点需要满足两次这个条件     非根节点只需满足一次这个条件 即为割点
                ans[x] += (ll)siz[y]*(n-siz[y]);  //flag++说明如果去掉x，则y与外界不再连通
                sum += siz[y];
                if(x != 1 || flag > 1) cut[x] = true;
            }
        }
        else{
            //siz[x] += siz[y]; //此处不能加这句话，因为y点之前已经被访问过了，说明就算去掉x，y仍然可以连通，因此不能算入
            low[x] = min(low[x],dfn[y]);  //因为该点之前已经被搜过了
        }
    }
    if(cut[x])
        ans[x] += (ll)(n-sum-1)*(sum+1)+(n-1);
    else
        ans[x] = 2*(n-1);
}

int main()
{
    while(~scanf("%d%d",&n,&m))
    {
```

```
        init();
        rep(i,1,m){
            int x,y;
            scanf("%d%d",&x,&y);
            if(x == y) continue;
            add(x,y); add(y,x);
        }
        tarjan(1);  //这里不需要从每一个点进行dfs，是因为题目保证所有城镇连通
        rep(i,1,n)
            printf("%lld\n",ans[i]);
    }
    return 0;
}
```

5.7 强连通分量

5.7.1 强连通分量及缩点

求有向图中的强连通分量
将 SCC 中的强连通分量进行缩点

求解算法：
1. 当节点 x 第一次被访问时，把 x 入栈，初始化 low[x] = dfn[x]
2. 扫描从 x 出发的每条边 (x,y)
(1) 若 y 没被访问过，则说明 (x,y) 是树枝边，递归访问 y，从 y 回溯之后，令 low[x] = min(low[x],low[y]).
(2) 若 y 被访问过并且 y 在栈中，即 y 被访问过，并且 y 不属于之前的强连通分量，则令 low[x] = min(low[x],dfn[y]).
3. 从 x 回溯之前，判断是否有 low[x] = dfn[x]。若成立，则不断从栈中弹出节点，直至 x 出栈.

```
#include <cstdio>
#include <iostream>
#include <algorithm>
#include <cstring>
#include <vector>
#define rep(i,a,b) for(int i = a;i <= b;i++)
using namespace std;
const int N = 1e5+10, M = 1e6+10;

struct Edge{
    int to,next;
}e[M],ec[M];
int head[N],dfn[N],low[N];  //dfn:dfs的序号 low:该点能够到达的最小的dfn值的点
int stack[N],vis[N],c[N];   //stack:维护通过点x到达的所有点  vis[x]:标记点x是否在堆栈中  c[x]:记录点x所在的强连通分量的编号
vector<int> scc[N]; //scc[i]:记录第i个强连通分量中的所有节点
int n,m,tot,num,top,cnt;    //tot记录边的序号，num记录dfs序，top记录stack堆栈的顶点，cnt记录图中强连通分量个数
int tc,headc[N];

void init()
{
    tot = 1; num = 0;
    top = 0; cnt = 0;
    memset(head,0,sizeof head);
    memset(c,0,sizeof c);
    memset(vis,0,sizeof vis);
    memset(dfn,0,sizeof dfn);
    memset(low,0,sizeof low);
    rep(i,1,n)
```

```
29          scc[i].clear();
30  }
31
32  void initc()
33  {
34      tc = 1;
35      memset(headc,0,sizeof headc);
36  }
37
38  void add(int x,int y)
39  {
40      e[++tot].to = y; e[tot].next = head[x]; head[x] = tot;
41  }
42
43  void addc(int x,int y)
44  {
45      ec[++tc].to = y; ec[tc].next = headc[x]; headc[x] = tc;
46  }
47
48  void tarjan(int x)
49  {
50      dfn[x] = low[x] = ++num;
51      stack[++top] = x, vis[x] = 1;   //标记点x在栈中
52      for(int i = head[x]; i ; i = e[i].next)
53      {
54          int y = e[i].to;
55          if(!dfn[y]){    //如果点y没有被访问过
56              tarjan(y);  //从y点一直递归下去
57              low[x] = min(low[x],low[y]);
58          }
59          else if(vis[y]) low[x] = min(low[x],dfn[y]);
60          //在求双连通分量时，此处为dfn[y]，即y点被访问过，则可用y的dfn值来更新x的low值
61          //但是在强连通分量中，图是有向图，因此vis[y]==0的点，要么没被访问过，要么已经出栈了，即已经属
        于另一块强连通分量中
62          //若y点已经属于另一块强连通分量，由于是有向图，因此y无法与x构成环，所以不能用y点dfn值来更新x
        的low值
63      }
64      if(dfn[x] == low[x])    //low[x] == dfn[x]成立，表明栈中从x到栈顶的所有节点构成一个强连通分
        量
65      {                       //这些节点无法与其他节点一起构成环
66          cnt++; int y;
67          do{
68              y = stack[top--],vis[y] = 0;  //节点出栈
69              c[y] = cnt, scc[cnt].push_back(y);  //将这些节点存入新的强连通分量中
70          }while(x != y);
71      }
72  }
73
74  int main()
75  {
76      while(~scanf("%d%d",&n,&m))
77      {
78          init();
79          rep(i,1,m){
80              int x,y;
81              scanf("%d%d",&x,&y);
82              add(x,y);
83          }
84          rep(i,1,n)
```

```
85              if(!dfn[i]) tarjan(i);
86          //将每个SCC缩成一个点，对于原图中的每条有向边
87          //如果c[x]!=c[y]，则在编号为c[x]和编号为c[y]的SCC之间连边
88          //最后结果将是一张有向无环图
89          initc();
90          rep(x,1,n){
91              for(int i = head[x]; i ;i = e[i].next){
92                  int y = e[i].to;
93                  if(c[x] == c[y]) continue;
94                  addc(c[x],c[y]);
95              }
96          }
97      }
98      return 0;
99  }
```

5.7.2 2-SAT

题意：
有一个小镇上只有一个牧师。这个小镇上有一个传说，在九月一日结婚的人会受到爱神的保佑，但是要牧师举办一个仪式。这个仪式要么在婚礼刚刚开始的时候举行，要么举行完婚礼正好结束。
现在已知有 n 场婚礼，告诉你每一场的开始和结束时间，以及举行仪式所需要的时间。问牧师能否参加所有的婚礼，如果能则输出一种方案。

```
1   #include <cstdio>
2   #include <iostream>
3   #include <algorithm>
4   #include <cstring>
5   #include <vector>
6   #define rep(i,a,b) for(int i = a;i <= b;i++)
7   using namespace std;
8   const int N = 1e5+10, M = 1e7;
9
10  struct Edge{
11      int to,next;
12  }e[M];
13  int head[N],dfn[N],low[N];  //dfn:dfs的序号 low:该点能够到达的最小的dfn值的点
14  int stack[N],vis[N],c[N];   //stack:维护通过点x到达的所有点  vis[x]:标记点x是否在堆栈中  c[x]:
        记录点x所在的强连通分量的编号
15  vector<int> scc[N]; //scc[i]:记录第i个强连通分量中的所有节点
16  int n,m,tot,num,top,cnt;    //tot记录边的序号，num记录dfs序，top记录stack堆栈的顶点，cnt记录图
        中强连通分量个数
17  int s[N],t[N],d[N];
18
19  void init()
20  {
21      tot = 1; num = 0;
22      top = 0; cnt = 0;
23      memset(head,0,sizeof head);
24      memset(c,0,sizeof c);
25      memset(vis,0,sizeof vis);
26      memset(dfn,0,sizeof dfn);
27      memset(low,0,sizeof low);
28      rep(i,1,n)
29          scc[i].clear();
30  }
31
```

```
32 void add(int x,int y)
33 {
34     e[++tot].to = y; e[tot].next = head[x]; head[x] = tot;
35 }
36
37 void tarjan(int x)
38 {
39     dfn[x] = low[x] = ++num;
40     stack[++top] = x, vis[x] = 1;   //标记点x在栈中
41     for(int i = head[x]; i ; i = e[i].next)
42     {
43         int y = e[i].to;
44         if(!dfn[y]){    //如果点y没有被访问过
45             tarjan(y);  //从y点一直递归下去
46             low[x] = min(low[x],low[y]);
47         }
48         else if(vis[y]) low[x] = min(low[x],dfn[y]);
49     }
50     if(dfn[x] == low[x])    //low[x] == dfn[x]成立，表明栈中从x到栈顶的所有节点构成一个强连通分
       量
51     {                       //这些节点无法与其他节点一起构成环
52         cnt++; int y;
53         do{
54             y = stack[top--],vis[y] = 0;  //节点出栈
55             c[y] = cnt, scc[cnt].push_back(y);  //将这些节点存入新的强连通分量中
56         }while(x != y);
57     }
58 }
59
60 int overlap(int a1,int b1,int a2,int b2)
61 {
62     if(a1>=a2&&a1<b2 || b1>a2&&b1<=b2 || a1<=a2&&b1>=b2) return 1;
63     return 0;
64 }
65
66 void solve()
67 {
68     rep(i,1,n){
69         if(c[i] == c[i+n]){
70             printf("NO\n");
71             return;
72         }
73     }
74     printf("YES\n");
75     int val[N];
76     memset(val,0,sizeof val);
77     rep(i,1,n){
78         if(c[i] < c[i+n])   //Tarjan之后的图就已经进行了拓扑排序
79             val[i] = 0;     //如果! a -> a，则a为真
80         else val[i] = 1;
81     }
82     rep(i,1,n){
83         if(val[i] == 0){
84             printf("%02d:%02d %02d:%02d\n",s[i]/60,s[i]%60,(s[i]+d[i])/60,(s[i]+d[i])
       %60);
85         }
86         else
87             printf("%02d:%02d %02d:%02d\n",(t[i]-d[i])/60,(t[i]-d[i])%60,t[i]/60,t[i
       ]%60);
```

```
    }
}

int main()
{
    while(~scanf("%d",&n))
    {
        init();
        rep(i,1,n){
            int x1,y1,x2,y2;
            scanf("%d:%d %d:%d %d",&x1,&y1,&x2,&y2,&d[i]);
            s[i] = x1*60+y1;
            t[i] = x2*60+y2;
        }
        //每个点都有两个状态，一个为真，一个为假，因此共2*n个状态
        rep(i,1,n){
            rep(j,i+1,n){
                if(overlap(s[i],s[i]+d[i],s[j],s[j]+d[j]))
                    add(i,j+n),add(j,i+n);
                if(overlap(s[i],s[i]+d[i],t[j]-d[j],t[j]))
                    add(i,j),add(j+n,i+n);
                if(overlap(t[i]-d[i],t[i],s[j],s[j]+d[j]))
                    add(i+n,j+n),add(j,i);
                if(overlap(t[i]-d[i],t[i],t[j]-d[j],t[j]))
                    add(i+n,j),add(j+n,i);
            }
        }
        rep(i,1,2*n)
            if(!dfn[i]) tarjan(i);
        solve();
    }
    return 0;
}
```

5.8 二分图匹配

```
#include <cstdio>
#include <iostream>
#include <cstring>
#define rep(i,a,b) for(int i = a;i <= b;i++)
using namespace std;
const int MAXN = 5010;  //点数最大值
const int MAXM = 50010; //边数最大值

struct Edge{
    int to,next; //next表示该边的上一条边的编号
}e[MAXM];
int head[MAXN],tot,uN;  //uN表示一共有多少个点
//head[u]表示以u为起点的边的编号
int girl[MAXN],used[MAXN];
//girl表示每个点的匹配情况
//used表示改点有没有被标记过

void init()
{
    tot = 0;
    memset(head,-1,sizeof head);
}
```

```

void addedge(int u,int v)
{
    tot++; //边的编号从1~n
    e[tot].to = v; e[tot].next = head[u]; //next表示与这条边共点的上一条边
    //如果next为-1，则表示该边无上一条边
    head[u] = tot;  //head[u]表示以u为起点的第一条边的编号
}

bool dfs(int u)
{
    for(int i = head[u]; i != -1;i = e[i].next){
        int v = e[i].to;
        if(!used[v])
        {
            used[v] = 1;
            if(!girl[v] || dfs(girl[v]))
            {
                girl[v] = u;
                return true;
            }
        }
    }
    return false;
}

int solve()
{
    int res = 0;
    memset(girl,0,sizeof girl);
    //点的编号 0 ~ uN-1
    for(int u = 0; u < uN;u++)
    {
        memset(used,0,sizeof used);
        if(dfs(u)) res++;
    }
    return res;
}

int main()
{
    //读入图
    printf("%d\n",solve());
    return 0;
}
```

5.9 第 K 短路

```
#include <cstdio>
#include <iostream>
#include <algorithm>
#include <cstring>
#include <queue>
#define rep(i,a,b) for(int i = a;i <= b;i++)
using namespace std;
const int M = 1e5+100;
const int N = 2000;
const int inf = 0x3f3f3f3f;
```

```
11
12  int n,m,s,t,k;
13  struct Edge{
14      int to,next,w;
15  }e[M],ef[M];
16  int head[N],headf[N],tot,totf;
17  int dis[N],book[N],dist[N];
18  struct Po{
19      int id; //到达哪一个点
20      int fw; //到该点已经走过的距离
21      int w; //到该点已经走过的距离+该点距离终点的距离
22  }tmp;
23
24  bool operator < (Po x,Po y) //优先队列只能重载 <
25  {
26      return x.w>y.w; //如果放在结构体内重载，需要加const
27  }
28  void init()
29  {
30      tot = 1; totf = 1;
31      memset(head,0,sizeof head);
32      memset(headf,0,sizeof headf);
33  }
34
35  void add(int x,int y,int w)
36  {
37      e[++tot].to = y; e[tot].next = head[x]; head[x] = tot; e[tot].w = w;
38  }
39
40  void addf(int x,int y,int w)
41  {
42      ef[++totf].to = y; ef[totf].next = headf[x]; headf[x] = totf; ef[totf].w = w;
43  }
44
45  void spfa(int x)
46  {
47      queue<int> q;
48      memset(book,0,sizeof book);
49      rep(i,1,n) dis[i] = inf;
50      while(!q.empty()) q.pop();
51      q.push(x);
52      book[x] = 1;
53      dis[x] = 0;
54      while(!q.empty())
55      {
56          int p = q.front();
57          for(int i = headf[p]; i ; i = ef[i].next)
58          {
59              if(dis[ef[i].to] > dis[p]+ef[i].w)
60              {
61                  dis[ef[i].to] = dis[p]+ef[i].w;
62                  if(book[ef[i].to] == 0)
63                  {
64                      q.push(ef[i].to);
65                      book[ef[i].to] = 1;
66                  }
67              }
68          }
69          book[p] = 0;
```

```
70          q.pop();
71      }
72  }
73
74  int bfs()
75  {
76      if(dis[s] == inf) return -1;    //如果不加这句话，就是wa，确保连通性
77      int val = 0;
78      priority_queue<Po> q;
79      while(!q.empty()) q.pop();
80      tmp.fw = 0; //走到id这个点目前所需要的距离
81      tmp.id = s;
82      tmp.w = tmp.fw+dis[tmp.id];
83      q.push(tmp);
84      if(t==s)k++;    //这里也是一个大坑点
85      while(!q.empty())
86      {
87          Po p = q.top();
88          q.pop();    //优先队列，必须在push之前就pop，不然队列顶的那个点会被调到底下去
89          int id = p.id;  //p所在的点
90          if(id == t) val++;
91          if(val == k) return p.fw;   //求第k短路
92          for(int i = head[id]; i ; i = e[i].next)    //加入新边
93          {
94              tmp.fw = p.fw+e[i].w;
95              tmp.id = e[i].to;
96              tmp.w = tmp.fw+dis[tmp.id]; //此处为估价函数
97              q.push(tmp);
98          }
99      }
100     if(val!=k){return -1;}
101 }
102
103 int main()
104 {
105     while(~scanf("%d%d",&n,&m))
106     {
107         init();
108         rep(i,1,m)
109         {
110             int x,y,z;
111             scanf("%d%d%d",&x,&y,&z);
112             add(x,y,z);
113             addf(y,x,z);
114         }
115         scanf("%d%d%d",&s,&t,&k);
116         spfa(t);
117         printf("%d\n",bfs());
118     }
119     return 0;
120 }
```

6 网络流

6.1 最大流

6.1.1 最小点覆盖

题意:
$N*N$ 的矩阵中有 K 个小行星，现在每行每列都有一个武器，可以消除此行或者此列中的所有小行星，问最少需要多少个武器可以将矩阵中所有小行星消除。

思路:
这是一个典型的最小点覆盖问题，可以用二分图匹配算法或者最大流算法进行解决。我们主要来讨论网络流的做法。

先来回顾一下最大流 $Dinic$ 的基础性质，先在残量网络上 BFS 求出所有节点的层次，构造了一个分层图。然后在分层图上 DFS 寻找增广路，在回溯时实时更新剩余容量。当在残量网络中 S 不能到达 T 时，算法结束。

因此最大流这个算法有个很重要的性质，即 == 最大流 = 最小割 = 最大匹配 = 最小点覆盖 ==，基本覆盖了大部分最大流应用的题目。

现在我们再回过头来看这题如何建图，这题就是用最少的点覆盖所有的边，典型的最小点覆盖题目。因此我们将其转化为最大匹配，即每个点只能选择一次，最多选择多少条边。因此对于一个在 (i,j) 位置的行星，将左边第 i 个点和右边第 j 个点相连，图中所有边流量均设为 1，跑最大流即可。

最后再来一个不严谨的”最大匹配 = 最小点覆盖”的证明。即当图中选的边已经达到了最大匹配之后，一定不存在一条未选的边，边的两个端点同时都没有被选。并且由于每个点只能被选择一次，因此每条边都代表只选了一个点，而这些点的集合覆盖了所有的边。不太严谨的证明，只是有利于记忆。详细证明可以网上继续查阅题解。

```
1  #include <cstdio>
2  #include <iostream>
3  #include <cstring>
4  #include <algorithm>
5  #include <queue>
6  #define rep(i,a,b) for(int i = a;i <= b;i++)
7  using namespace std;
8  const int inf = 1<<29,N = 1000+10,M = 300500;  //处理1e4-1e5规模的网络
9
10 struct Edge{
11     int to,next,v;
12 }e[M];
13 int n,m,s,t,k;  //顶点个数 边数 源点 汇点
14 int head[N],tot,dis[N],mp[N][N];
15 queue<int> q;
16
17 void init()   //千万别忘了初始化!
18 {
19     tot = 1; memset(head,0,sizeof head);  //点的编号是2~n，因为2^1 = 3, 3^1 = 2;  符合后续
       代码的操作
20 }
21
22 void add(int x,int y,int v)
23 {
24     e[++tot].to = y; e[tot].next = head[x]; e[tot].v = v; head[x] = tot;
25     e[++tot].to = x; e[tot].next = head[y]; e[tot].v = 0; head[y] = tot;  //反向边与正向
       边的流量之和为v
26 }
27
28 bool bfs()
29 {
```

```
30      memset(dis,0,sizeof dis);
31      while(!q.empty()) q.pop();
32      q.push(s); dis[s] = 1;
33      while(!q.empty())
34      {
35          int x = q.front(); q.pop();
36          for(int i = head[x];i;i = e[i].next)
37          {
38              if(e[i].v && !dis[e[i].to]){
39                  q.push(e[i].to);
40
41
42                  dis[e[i].to] = dis[x]+1;
43                  if(e[i].to == t) return 1;  //找到一条路就return
44              }
45          }
46      }
47      return 0;
48  }
49
50  int dinic(int x,int flow) //找增广路
51  {
52      if(x == t) return flow;
53      int rest = flow,k;  //rest为输入的流量
54      for(int i = head[x];i && rest; i = e[i].next)
55      {
56          if(e[i].v && dis[e[i].to] == dis[x]+1){
57              k = dinic(e[i].to,min(rest,e[i].v));
58              if(!k) dis[e[i].to] = 0;  //剪枝，去掉增广完毕的点
59              e[i].v -= k;
60              e[i^1].v += k;  //反向边加上flow，相当于我们可以反悔从这条路流过
61              rest -= k; //k为能够被送出去的流量
62          }
63      }
64      return flow-rest;  //总共被送出去了多少流量
65  }
66
67  int solve()
68  {
69      int flow = 0,maxflow = 0;
70      while(bfs())
71          while((flow = dinic(s,inf))) maxflow += flow;
72      return maxflow;
73  }
74
75  int main()
76  {
77      while(~scanf("%d%d",&n,&k))
78      {
79          init();
80          s = 1, t = 1+n+n+1;
81          rep(i,1,k){
82              int xx,yy; scanf("%d%d",&xx,&yy);
83              add(xx+1,yy+1+n,1);
84          }
85          rep(i,1,n) add(s,i+1,1), add(1+n+i,t,1);
86          printf("%d\n",solve());
87      }
88      return 0;
```

```
}
```

6.1.2 最大流 + 二分

题意:
给出了一个 $Y * X$ 的地图，在地图的四个边缘有门，用 $'D'$ 表示，$'X'$ 表示障碍物即不能走，$'.'$ 表示这个位置初始有一个人，现在地图中的所有人要逃出门外，每个 $'.'$ 这个点可以站好多人，但是出门的时候只能一个一个出，每个人移动一格的时间为 1，问最少需要多少时间，所有人可以撤出场地。如果不能撤出，输出 $impossible$ 。

思路:
因为是最少的时间，所以一开始想到的是最小割和最小费用最大流，然后发现均无法解决这个问题，因为难以解决每个时间每个门只能被经过一次这个问题。

后来发现我们可以 == 对每一个时刻的门分别建点 ==，然后使这个门的容量为 1，因此就可以满足每个时刻每个门只能出去一个人这个条件。

但是如果对门的所有时刻都建点的话，那么无疑跑出来的结果是不对的，即我们现在只能验证当前每一个时刻上限的情况下，所有人能不能出来。因此发现这个题的答案，即所需要的时间是单调的，因此可以进行二分答案。二分所有人出门的时间，然后用最大流建图来验证

当二分的时间为 x 时，则对每个门都建 x 个点，然后每个人到达这个门所需的时间如果为 y 的话，则将这个人与这个门所有 $>= y$ 的时刻连边。至此即可完成本题。

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <algorithm>
#include <map>
#include <queue>
#define rep(i,a,b) for(int i = a;i <= b;i++)
#define LOG1(x1,x2) cout << x1 << ": " << x2 << endl;
#define LOG2(x1,x2,y1,y2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << endl
    ;
using namespace std;
const int inf = 1<<29,N = 1e5+100,M = 5*1e6+100;  //处理1e4-1e5规模的网络

struct Edge{
    int to,next,v;
}e[M];
int n,m,s,t;  //顶点个数 边数 源点 汇点
int head[N],tot,dis[N];
queue<int> q;
char mp[20][20];
int numdoor,mandoor[200][200],numman,hp[200],vis[300];
int dir[4][2] = {{1,0},{-1,0},{0,1},{0,-1}};

void bbfs(int x)
{
    // printf("x:%d\n",x);
    queue<pair<int,int> > qp;
    while(qp.size()) qp.pop();
    memset(vis,0,sizeof vis);
    vis[x] = 1;
    qp.push(make_pair(x,0));
    while(qp.size()){
        int xx = qp.front().first, dist = qp.front().second;
        int yy = xx%m;
```

```
34          qp.pop();
35          xx = xx/m+1;
36          rep(i,0,3){
37              int xxx = xx+dir[i][0], yyy = yy+dir[i][1];
38              int tp = (xxx-1)*m+yyy;
39              if(xxx < 1 || xxx > n || yyy < 1 || yyy > m || mp[xxx][yyy] == 'X')
    continue;
40              if(vis[tp]) continue;
41              if(mp[xxx][yyy] == 'D'){
42                  // printf("oops!\n");
43                  // LOG2("x",x,"tp",tp);
44                  // LOG1("dist",dist);
45                  mandoor[hp[x]][hp[tp]] = dist+1;
46                  continue;
47              }
48              vis[tp] = 1;
49              qp.push(make_pair(tp,dist+1));
50          }
51      }
52  }
53
54  void init_solve()
55  {
56      memset(mandoor,0,sizeof mandoor);
57      numman = numdoor = 0;
58      memset(hp,0,sizeof hp);
59      rep(i,1,n)
60          rep(j,1,m)
61              if(mp[i][j] == '.') hp[(i-1)*m+j] = ++numman;
62              else if(mp[i][j] == 'D') hp[(i-1)*m+j] = ++numdoor;
63      rep(i,1,n)
64          rep(j,1,m)
65              if(mp[i][j] == '.'){
66                  // printf("i:%d,j:%d\n",i,j);
67                  bbfs((i-1)*m+j);
68              }
69  }
70
71  void init()   //千万别忘了初始化!
72  {
73      tot = 1; memset(head,0,sizeof head);  //点的编号是2~n, 因为2^1 = 3, 3^1 = 2;  符合后续
    代码的操作
74  }
75
76  void add(int x,int y,int v)
77  {
78      e[++tot].to = y; e[tot].next = head[x]; e[tot].v = v; head[x] = tot;
79      e[++tot].to = x; e[tot].next = head[y]; e[tot].v = 0; head[y] = tot;  //反向边与正向
    边的流量之和为v
80  }
81
82  bool bfs()
83  {
84      memset(dis,0,sizeof dis);
85      while(!q.empty()) q.pop();
86      q.push(s); dis[s] = 1;
87      while(!q.empty())
88      {
89          int x = q.front(); q.pop();
```

```
90          for(int i = head[x];i;i = e[i].next)
91          {
92              if(e[i].v && !dis[e[i].to]){
93                  q.push(e[i].to);
94                  dis[e[i].to] = dis[x]+1;
95                  if(e[i].to == t) return 1;  //找到一条路就return
96              }
97          }
98      }
99      return 0;
100 }
101
102 int dinic(int x,int flow) //找增广路
103 {
104     if(x == t) return flow;
105     int rest = flow,k;  //rest为输入的流量
106     for(int i = head[x];i && rest; i = e[i].next)
107     {
108         if(e[i].v && dis[e[i].to] == dis[x]+1){
109             k = dinic(e[i].to,min(rest,e[i].v));
110             if(!k) dis[e[i].to] = 0;  //剪枝，去掉增广完毕的点
111             e[i].v -= k;
112             e[i^1].v += k;  //反向边加上flow，相当于我们可以反悔从这条路流过
113             rest -= k; //k为能够被送出去的流量
114         }
115     }
116     return flow-rest;  //总共被送出去了多少流量
117 }
118
119
120 int solve()
121 {
122     int flow = 0,maxflow = 0;
123     while(bfs())
124         while(flow = dinic(s,inf)) maxflow += flow;
125     return maxflow;
126 }
127
128 void mainsolve()
129 {
130     int l = 1, r = 200, ans = -1;
131     while(l <= r){
132         int mid = (l+r)>>1;
133         init();
134         s = 1, t = 1+numman+mid*numdoor+1;
135         rep(i,1,numman) add(s,i+1,1);
136         rep(i,1,numdoor)
137             rep(j,1,mid) add(1+numman+(i-1)*mid+j,t,1);
138         rep(i,1,numman)
139             rep(j,1,numdoor)
140                 if(mandoor[i][j] <= mid && mandoor[i][j])
141                     rep(k,mandoor[i][j],mid) add(1+i,1+numman+(j-1)*mid+k,1);
142         if(solve() == numman) r = mid-1, ans = mid;
143         else l = mid+1;
144     }
145     if(ans == -1) printf("impossible\n");
146     else printf("%d\n",ans);
147 }
148
```

```
int main()
{
    int _; scanf("%d",&_);
    while(_--)
    {
        scanf("%d%d",&n,&m);
        rep(i,1,n) scanf("%s",mp[i]+1);
        init_solve();
        mainsolve();
    }
    return 0;
}
```

6.1.3 最大流路径输出

题意:
给出 n 个起重机，每个起重机有两个属性，$W[i]$ 表示这个起重机的重量，$L[i]$ 表示这个起重机能够拉起的最大重量（可以拉重物也可以拉起重机）。现在有 m 栋楼以及 m 个重物，要求给出每栋楼起重机的分配方案，使得每栋楼最后留下的起重机可以拉起对应的重物。$(1 \leq n \leq 100, 1 \leq M \leq 100)$

思路:
比赛的时候以为所有的起重机都得用上... 然后就想到了费用流... 然后就是 wa wa wa... 其实这是一道比较简单的最大流问题，主要难点应该在路径输出。

我们先来考虑下如何建图。首先对于 n 个起重机进行拆点，拆点的目的是化边权为点权来限制流量。然后将源点 s 和所有 $W[i] = 0$ 的点的入点相连，流量为 1。每个拆开的点的入点和出点连一条流量为 1 的边。对于点 x 的出点与点 y 的入点，x 与 y 相连（流量为 1），当且仅当 $W[y] \leq L[x]$。

再将 m 个重物与汇点相连，流量为 1。最后对于每个重物，将所有 $L[i]$ 大于重物重量的点与重物相连，流量为 1。然后直接跑最大流即可。如果最大流为 m，则输出路径，如果小于 m，则输出 $impossible$。

最后考虑本题最关键的步骤，求出最大流之后如何输出路径。

对于 m 个点，通过反向边往回跑，仅当反向边流量不为 0，而正向边流量为 0 时才走这条边，跑到源点即停止，进行 m 遍 dfs 即可。

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <algorithm>
#include <queue>
#include <vector>
#define rep(i,a,b) for(int i = a;i <= b;i++)
#define LOG1(x1,x2) cout << x1 << ": " << x2 << endl;
#define LOG2(x1,x2,y1,y2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << endl
    ;
#define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2
    << " , " << z1 << ": " << z2 << endl;
using namespace std;
const int inf = 1<<29,N = 1000,M = 1e5;  //处理1e4-1e5规模的网络

struct Edge{
    int to,next,v,f;
}e[M];
int n,m,s,t;  //顶点个数 边数 源点 汇点
int head[N],tot,dis[N],L[N],W[N],T[N];
queue<int> q;
```

```
20 vector<int> ans[N];
21
22 void init()   //千万别忘了初始化!
23 {
24     tot = 1; memset(head,0,sizeof head);  //点的编号是2~n，因为2^1 = 3, 3^1 = 2;  符合后续
       代码的操作
25 }
26
27 void add(int x,int y,int v)
28 {
29     e[++tot].to = y; e[tot].next = head[x]; e[tot].v = v; head[x] = tot; e[tot].f = 1;
30     e[++tot].to = x; e[tot].next = head[y]; e[tot].v = 0; head[y] = tot; e[tot].f = -1;
         //反向边与正向边的流量之和为v
31 }
32
33 bool bfs()  //判断是否有增广路径
34 {
35     memset(dis,0,sizeof dis);
36     while(!q.empty()) q.pop();
37     q.push(s); dis[s] = 1;
38     while(!q.empty())
39     {
40         int x = q.front(); q.pop();
41         for(int i = head[x];i;i = e[i].next)
42         {
43             if(e[i].v && !dis[e[i].to]){
44                 q.push(e[i].to);
45                 dis[e[i].to] = dis[x]+1;
46                 if(e[i].to == t) return 1;  //找到一条路就return
47             }
48         }
49     }
50     return 0;
51 }
52
53 int dinic(int x,int flow) //dfs找增广路，找到一条路，就把增广的流量加到答案中
54 {
55     if(x == t) return flow;
56     int rest = flow,k;  //rest为输入的流量
57     for(int i = head[x];i && rest; i = e[i].next)
58     {
59         if(e[i].v && dis[e[i].to] == dis[x]+1){
60             k = dinic(e[i].to,min(rest,e[i].v));
61             if(!k) dis[e[i].to] = 0;  //剪枝，去掉增广完毕的点
62             e[i].v -= k;
63             e[i^1].v += k;  //反向边加上flow，相当于我们可以反悔从这条路流过
64             rest -= k; //k为能够被送出去的流量
65         }
66     }
67     return flow-rest;  //总共被送出去了多少流量
68 }
69
70 int solve()
71 {
72     int flow = 0,maxflow = 0;
73     while(bfs())
74         while((flow = dinic(s,inf))) maxflow += flow;   //while 循环内容 —— 判断是否还有
       增广路
75     return maxflow;
```

```
}

void dfs(int x,int flag){ //路径输出
    if(1 < x && x <= 1+n) ans[flag].push_back(x-1);
    if(x == s) return;
    for(int i = head[x]; i; i = e[i].next)
        if(e[i^1].v == 0 && e[i].to <= 1+2*n && e[i].f == -1) dfs(e[i].to,flag);
}

int main()
{
    scanf("%d",&n); tot = 1;
    rep(i,1,n) scanf("%d%d",&W[i],&L[i]);
    scanf("%d",&m);
    rep(i,1,m) scanf("%d",&T[i]);
    s = 1, t = 2+2*n+m;
    rep(i,1,m) add(1+2*n+i,t,1);
    rep(i,1,n)
        if(W[i] == 0) add(s,i+1,1);
    rep(i,1,n) add(i+1,1+n+i,1);
    rep(i,1,n)
        rep(j,1,n)
            if(i != j && L[i] >= W[j]) add(i+1+n,1+j,1);
    rep(i,1,m)
        rep(j,1,n)
            if(L[j] >= T[i]) add(1+n+j,1+2*n+i,1);
    int maxflow = solve();
    if(maxflow != m) printf("impossible\n");
    else{
        rep(i,1,m)
            dfs(1+2*n+i,i);
        rep(i,1,m){
            for(int j = ans[i].size()-1; j >= 0; j--){
                printf("%d",ans[i][j]);
                if(j == 0) printf("\n");
                else printf(" ");
            }
        }
    }
    return 0;
}
```

6.1.4 最大权闭合子图

有一个有向图，每一个点都有一个权值（可以为正或负或 0），选择一个权值和最大的子图，使得每个点的后继都在子图里面，这个子图就叫最大权闭合子图。

能选的子图有 Ø,4,3,4,2,4,1,2,3,4, 它们的权值分别为 0,-1,5,-6,4.
所以最大权闭合子图为 3,4，权值为 5.

· 解法
这个问题可以转化为最小割问题，用网络流解决。
从源点 s 向每个正权点连一条容量为权值的边，每个负权点向汇点 t 连一条容量为权值的绝对值的边，有向图原来的边容量全部为无限大。

求它的最小割，割掉后，与源点 s 连通的点构成最大权闭合子图，权值为（正权值之和-最小割）。

· 如何理解

割掉一条边的含义
由于原图的边都是无穷大，那么割边一定是与源点 s 或汇点 t 相连的。

割掉 s 与 i 的边，表示不选择 i 点作为子图的点；
割掉 i 与 t 的边，表示选择 i 点为子图的点。

如果 s 与 i 有边，表示 i 存在子图中；
如果 i 与 t 有边，表示 i 不存在于子图中。

· 合法性
只有 s 与 t 不连通时，才能得到闭合子图。

如果 s 与 t 连通，则存在点 i,j，使得 s 到 i 有边，i 到 j 连通，j 到 t 有边，所以 j 一定是 i 的后继，但选择了 i，没有选择 j，不是闭合子图。

如果 s 与 t 不连通，选择了正权点 i，一定选择了 i 后继中的所有负权点。设 j 是 i 的后继中的正权点，则割掉 s 到 j 的边是没有意义的，最小割不会割掉它，则 j 一点被选中，所以 i 的所有后继都被选中，符合闭合图的定义。

· 最优性
最小割 =(不选的正权之和 + 要选的负权绝对值之和)
最大权闭合子图 =（正权之和-不选的正权之和-要选的负权绝对值之和）= 正权值和-最小割
因为正权值和，是定值，而最小割保证值最小，所以最大权闭合子图一定最优。

```
return 0;
```

6.2 费用流

```
#include <cstdio>
#include <iostream>
#include <algorithm>
#include <cstring>
#include <queue>
#define rep(i,a,b) for(int i = a;i <= b;i++)
using namespace std;
const int N = 1100, M = 1e5+100;

struct Edge{
    int to,next,cap,cost; //cap为该边的容量，cost为该边的花费
}e[M];
int head[N],tot,s,t; //tot记录边数，s为源点，t为汇点
int dis[N],incf[N],pre[N],vis[N]; //dis为到每一个点的最短距离，incf为每一个点流入的流量
//pre为每一个点的最短距离是由哪一条边更新而来，vis标记该点有没有被访问过
int n,m,maxflow,ans;

void init()
{
    tot = 1; //从2开始“成对存储”，2和3是一对，4和5是一对
    memset(head,0,sizeof head);
}

void add(int x,int y,int z,int c) //z为容量，c为花费
{
    //正向边，初始容量为z，单位费用为c
    e[++tot].to = y; e[tot].next = head[x]; head[x] = tot; e[tot].cap = z; e[tot].cost
    = c;
    //反向边，初始容量为0，单位费用为-c，与正向边“成对存储”
    e[++tot].to = x; e[tot].next = head[y]; head[y] = tot; e[tot].cap = 0; e[tot].cost
    = -c;
```

```
}

bool spfa()
{
    //用最短路找增广路
    queue<int> q;
    //建议不要用memset赋值，太容易出错了，当dis设置为long long时
    //tmp还是一个int，就特别容易出错
    fill(dis,dis+N,1e8);
//  memset(dis,0x3f,sizeof dis); //memset是按字节赋值，一个字节8位，一个16进制位表示4个二进制位，因此0x3f为一个字节
    //此处dis赋值为inf，寻找最短路（最小费用流）
    //如果dis赋值为0xcf，则为寻找最长路（最大费用流）
    //0xcf为11001111，因此按位赋值之后为负数
    memset(vis,0,sizeof vis); //每个点都没被访问过
    q.push(s); dis[s] = 0; vis[s] = 1;
    incf[s] = 1<<30; //incf为每一个点流入的流量
    while(q.size())
    {
        int x = q.front(); vis[x] = 0; q.pop();
        for(int i = head[x]; i ; i = e[i].next) //通过与x相连的各边遍历与x相连的点
        {
            if(!e[i].cap) continue; //该边剩余流量为0，不在残量网络中，无法通过
            int y = e[i].to;
            if(dis[y] > dis[x] + e[i].cost)  //如果此处为 < ，则为最长路算法【最大费用流】
            {
                dis[y] = dis[x]+e[i].cost;
                incf[y] = min(incf[x],e[i].cap);
                pre[y] = i; //记录y点的最短路是由哪一条边更新而来的
                if(!vis[y]) vis[y] = 1, q.push(y);
            }
        }
    }
//  int tmp = 0x3f3f3f3f;
    if(dis[t] == 1e8) return false; //汇点不可达，已求出最大流
    return true;
}

void update()
{
    int x = t;
    while(x!=s)
    {
        int i = pre[x];
        e[i].cap -= incf[t];
        e[i^1].cap += incf[t];
        x = e[i^1].to; //相反边的终点，即为x的上一个点
    }
    maxflow += incf[t];
    ans += dis[t]*incf[t];  //dis[t]为这条最短路上的总花费，incf为这条最短路上的流
}

void solve()
{
    ans = 0; maxflow = 0;
    while(spfa()) update();
    printf("%d\n",ans);
}

```

```
88  int main()
89  {
90      while(~scanf("%d%d",&n,&m))
91      {
92          //建图过程
93          s = 1; t = 2+n;
94          init();
95          add(s,2,2,0);
96          add(n+1,t,2,0);
97          rep(i,1,m){
98              int a,b,c;
99              scanf("%d%d%d",&a,&b,&c);
100             add(a+1,b+1,1,c); add(b+1,a+1,1,c);
101         }
102         solve();
103     }
104     return 0;
105 }
106
107 /*
108 常见错误:
109     1.没有init();
110     2.spfa()函数部分dis是long long型的，用memset赋值，再用int型的tmp比较...
111     3.建议spfa()函数部分的dis用fill来进行赋值【不易出错】
112 */
```

7 计算几何

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <vector>
#include <cmath>
#include <algorithm>
#define rep(i,a,b) for(int i = a; i <= b; i++)
#define LOG1(x) cout << "x: " << x << endl;
#define LOG2(x,y) cout << "x: " << x << ", y: " << y << endl;
#define pi acos(-1.0)
#define cross(p1,p2,p3) ((p2.x-p1.x)*(p3.y-p1.y)-(p3.x-p1.x)*(p2.y-p1.y)) //向量(p1,p2)与(p1,p3)叉乘
#define crossOp(p1,p2,p3) sign(cross(p1,p2,p3)) //判断正负，顺时针为负，为0则代表三点共线
using namespace std;

//实数比较
typedef double db;
const db EPS = 1e-9;
inline int sign(db a) {return a < -EPS ? -1 : a > EPS; } //返回-1表示a < 0, 1表示a > 0, 0表示a = 0
inline int cmp(db a, db b) {return sign(a-b); } //返回-1表示a < b, 1表示a > b, 0表示 a==b

//点类
struct Point {
    db x,y;
    Point() {}
    Point(db _x, db _y) : x(_x), y(_y) {}
    Point operator+(Point p) { return {x+p.x, y+p.y}; }
    Point operator-(Point p) { return {x-p.x, y-p.y}; }
    Point operator*(db d) { return {x*d, y*d}; }
    Point operator/(db d) { return {x/d, y/d}; }
    Point rotleft() { return Point(-y,x); } //逆时针旋转90度
    Point rotright() { return Point(y,-x); } //顺时针旋转90度
    db dot(Point p) { return x*p.x+y*p.y; } //点积
    db det(Point p) { return x*p.y-y*p.x; } //叉积
    Point rot(db an) { return {x*cos(an)-y*sin(an),x*sin(an)+y*cos(an)}; }  //旋转
    db abs() { return sqrt(abs2()); }
    db abs2() { return x*x+y*y; }
    db disTo(Point p) { return (*this-p).abs(); }
    //此时在x负半轴上的点，排序结果是最小的。如果去掉sign(x)>=0，则排序结果是最大的
    int quad() const { return sign(y) == 1 || (sign(y) == 0 && sign(x) >= 0); } //判断该点是否在x轴上方或x轴上

    bool operator<(Point p) const {
        int c = cmp(x, p.x);
        if (c) return c == -1;  //先判断x大小
        return cmp(y, p.y) == -1;   //再判断y大小
    }
    bool operator==(Point p) const {
        return cmp(x, p.x) == 0 && cmp(y, p.y) == 0;
    }
    bool operator!=(Point p) const{
        return (cmp(x, p.x) || cmp(y,p.y));
    }
};

db area(vector<Point> ps){  //凸包面积
```

```
55      db ret = 0; rep(i,0,ps.size()-1) ret += ps[i].det(ps[(i+1)%ps.size()]);
56      return ret/2;
57  }
58
59  db perimeter(vector<Point> ps){ //凸包周长
60      db ret = 0; rep(i,0,ps.size()-1) ret += ps[i].disTo(ps[(i+1)%ps.size()]);
61      return ret;
62  }
63
64  db dot(Point A, Point B, Point C){  //三点点积
65      return (B-A).dot(C-A);
66  }
67
68  db rad(Point p1, Point p2){ //返回两个向量的夹角，范围为-2*pi~2*pi，顺时针转为负数，逆时针转为正
        数
69      return atan2l(p1.det(p2),p1.dot(p2));
70      //取绝对值就是旋转角度，否则会有正负区别，返回幅度制
71  }
72
73  /*
74  坐标系变换 —— 选中点集中的两点，两点构成的向量为x轴
75      Point p1 = {AA[2].x-AA[1].x,AA[2].y-AA[1].y}; //选定两点作为向量
76      rep(i,1,n){
77          Point p2 = {AA[i].x-AA[1].x,AA[i].y-AA[1].y};
78          db ang = rad(p1,p2);        //返回两个向量角度
79          db len = p2.abs();
80          Point tmp = {len*cos(ang),len*sin(ang)};    //得到新坐标系下的点坐标
81          mp[tmp] = 1;  //可以用map存变更坐标轴之后的点
82      }
83  */
84
85  //求凸包
86  vector<Point> convexHull(vector<Point> ps) {
87      int n = ps.size(); if(n <= 1) return ps;
88      sort(ps.begin(),ps.end());
89      vector<Point> qs(n*2); int k = 0;
90      for(int i = 0; i < n; qs[k++] = ps[i++])
91          while(k > 1 && crossOp(qs[k-2],qs[k-1],ps[i]) <= 0) --k;    //把 <= 改成 <，即可
        将凸包边上的点也包括在凸包中，不稳定凸包问题
92      for(int i = n-2, t = k; i >= 0; qs[k++] = ps[i--])
93          while(k > t && crossOp(qs[k-2],qs[k-1],ps[i]) <= 0) --k;
94      qs.resize(k-1);
95      return qs;
96  }
97
98  //最小矩形覆盖
99  db minRectangleCover(vector<Point> ps){
100     //凸包点集顺序按逆时针
101     int n = ps.size();
102     if(n < 3) return 0.0;
103     ps.push_back(ps[0]);
104     db ans = -1;
105     int r = 1, p = 1, q;
106     for(int i = 0; i < n; i++){
107         //求出离边 ps[i]-ps[i+1] 最远的点 r
108         while(sign(cross(ps[i],ps[i+1],ps[r+1])-cross(ps[i],ps[i+1],ps[r])) >= 0) //叉积
        最大即为到点r到ps[i+1]-ps[i]这条边的距离最大
109             r = (r+1)%n;
110         //卡出 ps[i]-ps[i+1] 方向上正向 n 最远的点 p
```

```
111         while(sign(dot(ps[i],ps[i+1],ps[p+1])-dot(ps[i],ps[i+1],ps[p])) >= 0)   //正向最
    远即为点积最大
112             p = (p+1)%n;
113         if(i == 0) q = p;
114         //卡出 ps[i]-ps[i+1]方向上负向最远的点 q
115         while(sign(dot(ps[i],ps[i+1],ps[q+1]) - dot(ps[i],ps[i+1],ps[q])) <= 0) //负向最
    大即为点积最小
116             q = (q+1)%n;
117         db d = ps[i].disTo(ps[i+1]);    //线段长度
118         d = d*d;
119         //叉积求出高，点积求出底边
120         db tmp = cross(ps[i],ps[i+1],ps[r]) *
121             (dot(ps[i],ps[i+1],ps[p])-dot(ps[i],ps[i+1],ps[q]))/d;
122         if(ans < 0 || ans > tmp) ans = tmp;
123     }
124     return ans;
125 }
126
127 bool cmp1(Point a, Point b){        //按照角度排序，第四象限-第三象限-第二象限-第一象限
128     if(a.quad() != b.quad()) return a.quad() < b.quad();
129     else return sign(a.det(b)) > 0;
130 }
131
132 bool cmp2(Point a, Point b){    //第二种极角排序方式，用角度直接排
133     return a.ang < b.ang;   //tp[j].ang = atan2(am[j].y-am[i].y,am[j].x-am[i].x)，返回幅
    度制，范围是-pi~pi
134 }
135
136 struct Line {
137     Point s,e;
138     Line() {}
139     Line(Point _s, Point _e) : s(_s), e(_e) {}
140     bool operator == (Line v) { return (s == v.s) && (e == v.e); }
141     // 根据一个点和倾斜角 angle 确定直线，0 <= angle < pi
142     Line(Point p, double angle) {
143         s = p;
144         if(sign(angle-pi/2) == 0) e = (s+Point(0,1));
145         else e = (s+Point(1,tan(angle)));
146     }
147     // ax+by+c = 0
148     Line(db a, db b, db c){
149         if(!sign(a)) s = Point(0,-c/b), e = Point(1,-c/b);
150         else if(!sign(b)) s = Point(-c/a,0), e = Point(-c/a,1);
151         else s = Point(0,-c/b), e = Point(1,(-c-a)/b);
152     }
153     void adjust() { if(e<s) swap(s,e); }
154     db length() { return s.disTo(e); }  //求线段长度
155     db angle() {    //返回直线倾斜角 0 <= angle < pi，弧度制
156         db k = atan2(e.y-s.y,e.x-s.x);
157         if(sign(k) < 0) k += pi;
158         if(sign(k-pi) == 0) k -= pi;
159         return k;
160     }
161     Point crosspoint(Line v){ //求两直线交点
162         db a1 = (v.e-v.s).det(s-v.s);
163         db a2 = (v.e-v.s).det(e-v.s);
164         return Point((s.x*a2-e.x*a1)/(a2-a1),(s.y*a2-e.y*a1)/(a2-a1));
165     }
166     // 点和直线关系，1在上方，2在下方，3在直线上
```

```
167     int relation(Point p) {
168         int c = sign((p-s).det(e-s));
169         if(c < 0) return 1;
170         else if(c > 0) return 2;
171         else return 3;
172     }
173     // 点在线段上的判断
174     bool pointonseg(Point p) { return sign((p-s).det(e-s)) == 0 && sign((p-s).dot(p-e))
        <= 0; }
175     // 两向量平行 (对应直线平行或重合)
176     bool parallel(Line v) { return sign((e-s).det(v.e-v.s)) == 0; }
177     // 线段比较，用于map存直线
178     bool operator<(Line l) const {
179         if(s != l.s) return s < l.s;
180         else return e < l.e;
181     }
182 };
183
184 struct circle{
185     Point p; //圆心
186     db r; //半径
187     circle() {}
188     circle(Point _p, db _r){
189         p = _p;
190         r = _r;
191     }
192     circle(Point a,Point b,Point c){
193         Line u = Line((a+b)/2,((a+b)/2)+((b-a).rotleft()));
194         Line v = Line((b+c)/2,((b+c)/2)+((c-b).rotleft()));
195         p = u.crosspoint(v);
196         r = p.disTo(a);
197     }
198 };
199
200 int main()
201 {
202     Point p1(1,3), p2(2,3), p3(1.5,1), p4(1.5,5);
203     Line l1(p1,p2); l1.adjust();
204     printf("%f\n",l1.angle());
205
206     return 0;
207 }
208
209
210 /*
211 例题1: [hihocoder 1879]
212     有n个点，求锐角三角形的个数/总面积 (2018 北京icpc )
213     n <= 2000，可能三点共线
214 解法1:
215     (n,3)-直角-钝角，O((n^2)logn)，因为锐角三角形会出现重复，因此计算直角和钝角，直角和钝角不会重复
216
217
218 例题2:
219     点旋转，用复数乘法来操作
220
221
222 例题3:
223     n个点，查看是否有三点共线
224     枚举一个点，将其它点进行极角排序，查看是否有三点共线 O((n^2)logn)
```

```
225     如果是整数点，还可以进行哈希
226     (x1,y1) (x2,y2)
227     d1 = gcd(x1,y1)
228     d2 = gcd(x2,y2)
229     hash (x1/d1, y1/d1), (x2/d2,y2/d2)
230 */
```

8 其他

8.1 三分

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <algorithm>
#define rep(i,a,b) for(int i = a; i <= b; i++)
using namespace std;
const int N = 20;
const double eps = 1e-7;
double coe[N];
int n;

double func(double x)
{
    double ans = 0;
    double base = 1;
    rep(i,0,n)
    {
        ans += base*coe[i];
        base *= x;
    }
    return ans;
}

void solve(double l, double r)
{
    while(l + eps < r)
    {
        double m1 = (2*l+r)/3, m2 = (2*r+l)/3;
        double r1 = func(m1), r2 = func(m2);
        if(r1 >= r2)
            r = m2;
        else
            l = m1;
    }
    printf("%.5f\n",r);
}

int main()
{
    double l,r;
    scanf("%d",&n);
    scanf("%lf%lf",&l,&r);
    for(int i = n; i >= 0; i--)
        scanf("%lf",&coe[i]);
    solve(l,r);
    return 0;
}
```

8.2 bitset 优化暴力

题意：
给出一个 $n*m$ 的矩阵，矩阵每个点为 1 或 0，每行每列均可翻转且只能翻转一次。问能否将矩形通过翻转变成一个从 $a_{1.1}...a_{1.m}$ $a_{2.1}...a_{n.m}$ 数值不下降的状态，如果可以给出每行每列的翻转状态，否则输出 NO。$(1 \leq n, m \leq 200)$

思路:
稍微考虑一下这个题目，就可以发现此题难点主要在于一个点是否翻转由该点所代表的行列同时决定，因此我们难以进行判断。

所以我们可以思考能否事先确定行或列的状态，这样判断起来就很方便了。然后就可以发现只要确定了矩阵的最终状态，再确定了第一行的状态，我们就可以确定每一列的状态。确定了每一列状态和矩形最终状态，就可以不断向下循环确定每一行的状态，如果一直都不发生冲突，则为 YES。

我们来考虑一下复杂度。首先枚举矩形的最终状态，即从 (n,m) 向 $(1,1)$ 不断填 1，因此一共有 $n*m$ 个最终状态。对于每个最终状态，我们枚举第一行是否翻转，然后确定每一列的状态，再对剩下的每一行进行判断。因此复杂度为 $O(n*m*n*m)$。

这样的复杂度肯定是会 T 的，因此考虑 $bitset$ 优化，对于每一行直接按位异或得到答案，可以将复杂度优化到 $O(n*m*n*m)/32$，因此可以通过此题。具体异或细节见代码。

```
#include <cstdio>
#include <cstdlib>
#include <iostream>
#include <cstring>
#include <bitset>
#include <algorithm>
#define __ ios::sync_with_stdio(0);cin.tie(0);cout.tie(0)
#define rep(i,a,b) for(int i = a; i <= b; i++)
#define LOG1(x1,x2) cout << x1 << ": " << x2 << endl;
#define LOG2(x1,x2,y1,y2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2 << endl
    ;
#define LOG3(x1,x2,y1,y2,z1,z2) cout << x1 << ": " << x2 << " , " << y1 << ": " << y2
    << " , " << z1 << ": " << z2 << endl;
typedef long long ll;
typedef double db;
const int N = 200+10;
const int M = 1e5+100;
const db EPS = 1e-9;
using namespace std;

bitset<N> row,col,norml[2];
bitset<N> mp[N],jud;
bitset<N> base[N];
int n,m;

void test()
{
    col = mp[0]^base[0]^norml[row[0]]; //枚举第一行状态, 确定每一列是否翻转
    rep(i,1,n-1){
        jud = base[i]^mp[i]^col; //判断这一行的row是否翻转
        if(jud.count() == m) row.set(i,1);
        else if(jud.count() == 0) row.set(i,0);
        else return;
    }
    printf("YES\n");
    rep(i,0,n-1) printf("%d",row[i]==1); printf("\n");
    rep(i,0,m-1) printf("%d",col[i]==1); printf("\n");
    exit(0);
}

int main()
{
```

```
    scanf("%d%d",&n,&m);
    rep(i,0,n-1)
        rep(j,0,m-1){
            int xx; scanf("%d",&xx);
            if(xx == 1) mp[i].set(j);
        }
    rep(i,0,m-1) norml[1].set(i);
    for(int i = n-1; i >= 0; i--)
        for(int j = m-1; j >= 0; j--){
            base[i].set(j);
            row.set(0,1);
            test();
            row.set(0,0);
            test();
        }
    printf("NO\n");
    return 0;
}
```

8.3 IDA*

```
#include <cstdio>
#include <iostream>
#include <algorithm>
#include <cmath>

int n, shell[21];

// swap [x1, x2] and [x2 + 1, x3]
void move (int x1, int x2, int x3)
{
     int tmp[21], i, j;
     for (i = x2 + 1, j = 0; i <= x3; i++, j++)
         tmp[j] = shell[i];
     for (i = x1; i <= x2; i++, j++)
         tmp[j] = shell[i];
     for (i = x1, j = 0; i <= x3; i++, j++)
         shell[i] = tmp[j];
     return;
}

int hfunc ()
{
    int i, ans = 0;
    for (i = 0; i < n - 1; i++)
        if (shell[i + 1] != shell[i] + 1) ans++;
    if (shell[n - 1] != n) ans++;
    return ans;
}

int maxdepth;
int dfs (int depth)
{
    int x1, x2, x3, h;
    for (x1 = 0; x1 <= n - 2; x1++) //枚举移动区间的左端点
    {
        for (x2 = x1; x2 <= n - 2; x2++)  //枚举移动区间的右端点
        {
```

```
38              for (x3 = x2 + 1; x3 <= n - 1; x3++)  //枚举插入点
39              {
40                  move(x1, x2, x3);   //进行移动
41                  h = hfunc();
42                  if (h == 0) return 1;
43                  else if (3 * depth + h <= 3 * maxdepth) //IDA*, 限制深度
44                  {
45                       if (dfs(depth + 1)) return 1;
46                  }
47                  move(x1, x1 - x2 + x3 - 1, x3); //如果不可行, 则返回原状
48              }
49          }
50      } return 0;
51  }
52
53  int main ()
54  {
55      int kase, i;
56      scanf("%d", &kase);
57      for (; kase > 0; kase--)
58      {
59          scanf("%d", &n);        //几本书
60          for (i = 0; i < n; i++)
61              scanf("%d", &shell[i]); //各个位置的序号
62          maxdepth = (int)ceil((double)hfunc() / 3);
63          if (maxdepth) while (maxdepth < 5 && dfs(1) == 0) maxdepth++;    //不断加大搜索深度
64          if (maxdepth == 5) printf("5 or more\n");
65          else printf("%d\n", maxdepth);
66      }
67      return 0;
68  }
69
70  /*
71      题意:
72  1-n本书, 可以选择任意一段连续的书, 再将这一段书插入到其他某个位置。
73  目标是将书按照1-n的顺序依次排列, 求最少需要多少次操作。
74  如果更换次数 >= 5, 则直接输出  “5 or more”
75
76      解法:
77  限制dfs的搜索深度, 并且设计一个估价函数。
78  可以发现, 将一段书插入到某一个位置, 最多改变3个数的后继值, 因此对于每一个状态, 记录该状态下i与i+1不匹配
        的个数
79  然后ceil(这个个数/3)的值就是该状态达到目标状态所至少需要的操作数
80
81  对于每个状态, 当前深度+目标操作数 < 5才继续进行深度搜索, 由此大大降低了搜索的复杂度
82  */
```