ACM 模板

Hugh Locke

November 21, 2019

Contents

1 一切的开始

1.1 头文件与预处理

```
#include <map>
#include <set>
#include <ctime>
#include <cmath>
#include <queue>
#include <stack>
#include <vector>
#include <bitset>
#include <string>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <sstream>
#include <iostream>
#include <algorithm>
#include <functional>
using namespace std;
#define For(i, x, y) for(int i=x;i<=y;i++)
#define _For(i, x, y) for(int i=x;i>=y;i--)
#define Mem(f, x) memset(f,x,sizeof(f))
#define Sca(x) scanf("%d", &x)
#define Sca2(x,y) scanf("%d%d",&x,&y)
#define Sca3(x,y,z) scanf("%d%d%d",&x,&y,&z)
#define Scl(x) scanf("%lld",&x)
#define Pri(x) printf("%d\n", x)
#define Prl(x) printf("%lld\n",x)
#define CLR(u) for(int i=0;i<=N;i++)u[i].clear()
#define LL long long
#define ULL unsigned long long
#define mp make_pair
#define PII pair<int,int>
#define PIL pair<int,long long>
#define PLL pair<long long,long long>
#define pb push_back
#define fi first
#define se second
typedef vector<int> VI;
const double eps = 1e-9;
const int maxn = 110;
const int INF = 0x3f3f3f3f;
const int mod = 1e9 + 7;
```

1.2 cin 关同步

```
//关闭 cin 对 scanf 的同步，将不能使用 scanf
//会提速但还是不如 scanf 快
ios::sync_with_stdio(false)
```

1.3 读入挂

```
int read(){
    int x = 0,f = 1;
    char c = getchar();
    while (c<'0' || c>'9'){
        if (c == '-') f = -1;
        c = getchar();
    }
    while (c >= '0'&&c <= '9'){
        x = x * 10 + c - '0';
        c = getchar();
    }
    return x*f;
}
```

2 图论

2.1 图论的基础

2.1.1 链式前向星

```
struct Edge{
    int to,next,dis;
}edge[maxm]; //maxm 为边的数量，无向图数目需乘二
int head[maxn],tot;
void init(){
    for(int i = 1; i <= N ; i ++) head[i] = -1;
    tot = 0;
}
void add(int u,int v,int w){
    edge[tot].to = v;
    edge[tot].next = head[u];
    edge[tot].dis = w;
    head[u] = tot++;
}
```

2.2 最短路

2.2.1 堆优化 Dijkstra

```
int dis[1005],vis[1005];
struct node{
    int pos,val;
    node(int pos,int val):pos(pos),val(val){}
    friend bool operator < (node a,node b){
        return a.val > b.val;
    }
};
int Dijkstra(int s,int t){
    for(int i = 1; i <= N ; i ++) dis[i] = INF;
    dis[s] = 0;
    priority_queue<node>Q;
    Q.push(node(s,0));
    while(!Q.empty()){
```

```
        node u = Q.top(); Q.pop();
        if(u.val > dis[u.pos]) continue;
        for(int i = head[u.pos]; ~i; i = edge[i].next){
            int v = edge[i].to;
            if(dis[v] > edge[i].dis + u.val){
                dis[v] = edge[i].dis + u.val;
                Q.push(node(v,dis[v]));
            }
        }
    }
    return dis[t];
}
```

2.2.2 SPFA

```
//spfa 判负环，spfa 时间复杂度玄学，必要时手写队列以及打上快速读入，spfa 容易被卡，非必要时用 Dijkstra
bool vis[maxn];
int dis[maxn],cnt[maxn];
bool spfa(int s){
    for(int i = 1; i <= N ; i ++){
        vis[i] = 0; cnt[i] = 0; dis[i] = INF;
    }
    queue<int>Q;
    Q.push(s);
    vis[s] = 1; dis[s] = 0;
    while(!Q.empty()){
        int u = Q.front(); Q.pop();
        vis[u] = 0;
        if(cnt[u] >= N) return true;
        for(int i = head[u]; ~i ; i = edge[i].next){
            int v = edge[i].to;
            if(dis[v] > dis[u] + edge[i].dis){
                dis[v] = dis[u] + edge[i].dis;
                if(!vis[v]){
                    Q.push(v);
                    vis[v] = 1;
                    cnt[v]++;
                    if(cnt[v] >= N) return true;
                }
            }
        }
    }
    return false;
}
```

2.2.3 Floyd

```
//用于求任意两点间最短距离
//传递闭包：所有两两之间的连通关系
for(int k = 1; k <= N ; k ++){
    for(int i = 1; i <= N ; i ++){
        for(int j = 1; j <= N; j ++){
            MAP[i][j] = min(MAP[i][j],MAP[i][k] + MAP[k][j]);
```

```
        }
    }
}
```

2.3 分层图

```
/*
    分层图思想，其实就是在原图上建立很多的辅助结点
    然后用现有的算法（例如最短路）就可以简单的解决问题
    例如一个点买入东西在另一个点卖出且只能买入卖出一次的模型，就可以建立三个图，
    每个图之间的边权都是 0 表示相互之间没有交易，
    图 1 和图 2 相同点之间边权为-val[i] 表示买入
    图 2 和图 3 相同点之间边权为 val[i] 表示卖出
    最后图 3 上的结点的最长路就是经过一次买入卖出之后在这个节点上能赚到的最多钱
    分层图没有模板，放在这里就是提供一个做题时候的思路
*/
```

2.4 差分约束

```
/* 差分约束系统：如果一个系统由 n 个变量和 m 个约束条件组成，其中每个约束条件形如 xj - xi<=
↪  bk ( i , j  [1, n], k  [1, m])，则称其为差分约束系统。
例如如下的约束条件：
X1 - X2 <= 0 X1 - X5 <= -1
X2 - X5 <= 1 X3 - X1 <= 5
X4 - X1 <= 4 X4 - X3 <= -1
X5 - X3 <= -3 X5 - X4 <= -3
全都是两个未知数的差小于等于某个常数（大于等于也可以，因为左右乘以-1 就可以化成小于等于）。这样
↪  的不等式组就称作差分约束系统。
差分约束系统求解过程：
1. 新建一个图，N 个变量看作 N 个顶点，M 个约束条件作为 M 条边。每个顶点 Vi 分别对于一个未知
↪  量，每个有向边对应两个未知量的不等式。
2. 为了保证图的连通性，在图中新加一个节点 Vs，图中每个节点 Vi 都能从 Vs 可达，建立边 w(Vs,
↪  Vi) = 0。
3. 对于每个差分约束 Xj - Xi <= Bk(这里是小于等于号)，则建立边 w(Xi, Xj) = Bk。
4. 初始化 Dist[] = INF, Dist[Vs] = 0.
5. 求解以 Vs 为源点的单源最短路径，推荐用 SPFA，因为一般可能存在负值。
如果图中存在负权回路，则该差分约束系统不存在可行解。
Vs 到某点如果不存在最短路径，即最短路为 INF，则对于该点表示的变量可以取任意值，都能满足差分约
↪  束的要求，如果存在最短路径，则得到该变量的最大值。
上述过程最终得到的解为满足差分约束系统各项的最大值。
注意点：
1. 如果要求最大值想办法把每个不等式变为标准 x - y <= k 的形式，然后建立一条从 y 到 x 权值为
↪  k 的边，变得时候注意 x - y < k => x - y <= k-1。
2. 如果要求最小值的话，变为 x - y >= k 的标准形式，然后建立一条从 y 到 x 权值为 k 的边，求出
↪  最长路径即可。
3. 如果权值为正，用 Dijkstra, SPFA, BellmanFord 都可以，如果为负不能用 Dijkstra，并且需要判
↪  断是否有负环，有的话就不存在。*/
#include<iostream>
#include<algorithm>
#include<cstdio>
#include<cstring>
#include<queue>
#define INF 0x7fffffff
using namespace std;
```

```cpp
const int MAXN = 1100;
const int MAXM = 30030;

struct EdgeNode
{
    int to;
    int w;
    int next;
}Edges[MAXM];

int Head[MAXN],Dist[MAXN],vis[MAXN],outque[MAXN],id;

void AddEdges(int u,int v,int w)
{
    Edges[id].to = v;
    Edges[id].w = w;
    Edges[id].next = Head[u];
    Head[u] = id++;
}
void SPFA(int s,int N)
{
    int ans = 0;
    memset(vis,0,sizeof(vis));
    memset(outque,0,sizeof(outque));
    for(int i = 1; i <= N; ++i) Dist[i] = INF;
    Dist[s] = 0;
    vis[s] = 1;
    queue<int> Q;
    Q.push(s);
    while( !Q.empty() )
    {
        int u = Q.front();
        Q.pop();
        vis[u] = 0;
        outque[u]++;
        if(outque[u] > N+1) //如果出队次数大于 N, 则说明出现负环
        {
            ans = -1;
            break;
        }
        for(int i = Head[u]; i != -1; i = Edges[i].next){
            int temp = Dist[u] + Edges[i].w;
            if(temp < Dist[Edges[i].to]){
                Dist[Edges[i].to] = temp;
                if( !vis[Edges[i].to]){
                    vis[Edges[i].to] = 1;
                    Q.push(Edges[i].to);
                }
            }
        }
    }

    if(ans == -1)   //出现负权回路, 不存在可行解
```

```
        printf("-1\n");
    else if(Dist[N] == INF) //可取任意值，都满足差分约束系统
        printf("-2\n");
    else
        printf("%d\n",Dist[N]);  //求使得源点 s 到 终点 t 的最大的值
}

int main(){
    int N,ML,MD,u,v,w;
    while(~scanf("%d%d%d", &N, &ML, &MD)){
        memset(Head,-1,sizeof(Head));
        id = 0;
        for(int i = 0; i < ML; ++i){
            scanf("%d%d%d",&u,&v,&w);
            AddEdges(u,v,w);//建边 u - v <= w
        }
        for(int i = 0; i < MD; ++i){
            scanf("%d%d%d",&u,&v,&w);
            AddEdges(v,u,-w);//建边 v - u <= w
        }
//这里不加也可以
//        for(int i = 1; i < N; ++i)
//            AddEdges(i+1,i,0);
        SPFA(1,N);  //求使得源点 s 到 终点 t 的最大的值
    }
    return 0;
}
```

2.5 欧拉路径

```
//欧拉回路：一笔画且回到起点
//欧拉路径（欧拉通路）：不要求回到起点
/*
无向图
G 存在欧拉通路的充要条件是：G 为连通图，并且 G 仅有两个奇度结点（度数为奇数的顶点）或者无奇度
↪ 结点。
推论
（1） 当 G 是仅有两个奇度结点的连通图时，G 的欧拉通路必以此两个结点为端点；
（2）当 G 是无奇度结点的连通图时，G 必有欧拉回路
（3）G 为欧拉图（存在欧拉回路）的充分必要条件是 G 为无奇度结点的连通图
*/
/*
（有向图） 定理
有向图 D 存在欧拉通路的充要条件是：D 为有向图，D 的基图连通，并且所有顶点的出度与入度相等；或
↪ 者 除两个顶点外，其余顶点的出度与入度都相等，而这两个顶点中一个顶点的出度与入度之差为 1，
↪ 另一个顶点的出度与入度之差为-1.
推论
（1）当 D 除出、入度之差为 1，-1 的两个顶点之外，其余顶点的出度与入度相等时，D 的有向欧拉通路
↪ 必以出、入度之差为 1 的顶点作为始点，以出、入度之差为-1 的顶点作为终点。
（2）当 D 的所有顶点的出、入度都相等时，D 中存在有向欧拉回路。
（3）有向图 D 为有向欧拉图的充要条件是 D 的基图为连通图，并且所有顶点的出、入度都相等。
*/
//模板：无向图找欧拉路径，字典序最小
//如果不要求字典序，可以采用邻接表
```

```
const int maxn = 510;
int N,M,K;
int MAP[maxn][maxn];
int ind[maxn];
vector<int>ans;
void dfs(int t){
    for(int i = 0 ; i < maxn; i ++){
        if(MAP[t][i]){
            MAP[t][i]--; MAP[i][t]--;
            dfs(i);
        }
    }
    ans.pb(t);
}
int main(){
    Sca(M); init();
    int root;
    for(int i = 1; i <= M ; i ++){
        int u,v; Sca2(u,v);
        MAP[u][v]++; MAP[v][u]++;
        ind[u]++; ind[v]++;
        root = v;
    }
    for(int i = 0 ; i < maxn; i ++){
        if(ind[i] & 1){
            root = i;
            break;
        }
    }
    dfs(root);
    for(int i = ans.size() - 1; i >= 0; i --){
        printf("%d\n",ans[i]);
    }
    return 0;
}
```

2.6 kruskal 重构树

```
/*kruskal 重构树
这个东西主要来处理最小生成树的最大边权问题
当然也可以处理最大生成树的最小边权问题  核心思想跟 krsskal 差不多
我们重构树的过程是这样的
将所有边按边权从小到大排序
每次最小的一条边，如果条边相连的两个点在同一个集合中，那么就跳过，否则就将这两个点的祖先都连到
↪ 一个虚点上去，让这个虚点的点权等于这条边的边权
这样的话这课被重构的树就有一些奇妙的性质
原本最小生成树上的点在重构树里都是叶节点
从任何一个点往根上引一条路径，这条路径经过的点的点权单调不降（最大生成树单调不升）
任意两点之间路径的最大边权就是他们的 LCA 的点权 */
//模板：给一张图，每次询问两点间路径最大值的最小值是多少
//做法 1. 最小生成树上 LCA 求树链最大值
//作法 2(如下代码):kruskal 重构树上直接求 LCA 点权
const int maxn = 3e5 + 10;
int N,M;
```

```
int tree[maxn];
int find(int t){
        if(tree[t] == t) return t;
        return tree[t] = find(tree[t]);
}
struct Edge{
        int to,next;
}edge[maxn * 2];
int head[maxn],tot;
void init(){
        for(int i = 0 ; i <= N ; i ++) head[i] = -1,tree[i] = i;
        tot = 0;
}
void add(int u,int v){
        edge[tot].to = v;
        edge[tot].next = head[u];
        head[u] = tot++;
}
struct E{
        int u,v,w;
}e[maxn];
bool cmp(E a,E b){
        return a.w < b.w;
}
int w[maxn];
const int SP = 20;
int pa[maxn][SP],dep[maxn];
void dfs(int u,int la){
        pa[u][0] = la; dep[u] = dep[la] + 1;
        for(int i = 1; i < SP; i ++){
                pa[u][i] = pa[pa[u][i - 1]][i - 1];
        }
        for(int i = head[u]; ~i; i = edge[i].next){
                int v = edge[i].to;
                if(v == la) continue;
                dfs(v,u);
        }
}
int lca(int u,int v){
        if(dep[u] < dep[v]) swap(u,v);
        int t = dep[u] - dep[v];
        for(int i = 0 ; i < SP; i ++) if(t & (1 << i)) u = pa[u][i];
        for(int i = SP - 1; i >= 0; i --){
                int uu = pa[u][i],vv = pa[v][i];
                if(uu != vv){
                        u = uu;
                        v = vv;
                }
        }
        return u == v?u:pa[u][0];
}
int main(){
        scanf("%d%d",&N,&M); init();
```

```
        for(int i = 1; i <= M ;  i++){
                scanf("%d%d%d",&e[i].u,&e[i].v,&e[i].w);
        }
        sort(e + 1,e + 1 + M,cmp);
        for(int i = 1; i <= M ; i ++){
                int u = find(e[i].u),v = find(e[i].v);
                if(u == v) continue;
                N++; tree[N] = N; w[N] = e[i].w; head[N] = -1;
                tree[u] = N; tree[v] = N;
                add(N,u); add(N,v);
        }
        for(int i = 1; i <= N ; i ++){
                if(tree[i] == i) dfs(i,-1);
        }
        int Q; scanf("%d",&Q);
        while(Q--){
                int u,v; scanf("%d%d",&u,&v);
                if(find(u) != find(v)) puts("impossible");
                else printf("%d\n",w[lca(u,v)]);
        }
        return 0;
}
```

2.7 Tarjan

2.7.1 割点

```
//在无向图中，如果删除一个点这个图就不联通，那么这个点就叫割点
//模板：Tarjan 求割点，N 个点 M 条边找有多少割点并输出
int Low[maxn],dfn[maxn],Stack[maxn];
int Index,top;
bool Instack[maxn];
bool cut[maxn]; //记录是否为割点
int add_block[maxn];   //删除这个点之后增加的联通块
void Tarjan(int u,int la){
    int v;
    Low[u] = dfn[u] = ++Index;
    Stack[++top] = u;
    Instack[u] = true;
    int son = 0;
    for(int i = head[u]; ~i; i = edge[i].next){
        int v = edge[i].to;
        if(v == la) continue;
        if(!dfn[v]){
            son++;
            Tarjan(v,u);
            if(Low[u] > Low[v]) Low[u] = Low[v];
            if(u != la && Low[v] >= dfn[u]){
                cut[u] = 1;
                add_block[u]++;
            }
        }else if(Low[u] > dfn[v]) Low[u] = dfn[v];
    }
    if(u == la && son > 1) cut[u] = 1;
```

```
    if(u == la) add_block[u] = son - 1;
    Instack[u] = false;
    top--;
}
int solve(){
    for(int i = 0 ; i <= N + 1; i ++)cut[i] = add_block[i] = dfn[i] = Instack[i] = 0;
    Index = top = 0;
    int ans = 0;
    for(int i = 1; i <= N ; i ++) if(!dfn[i]) Tarjan(i,i);
    for(int i = 1; i <= N ; i ++) ans += cut[i];
    return ans;
}
int main(){
    N = read(); M = read(); init();
    for(int i = 1; i <= M ; i ++){
        int u = read(), v = read();
        add(u,v); add(v,u);
    }
    Pri(solve());
    for(int i = 1; i <= N ; i ++) if(cut[i]) printf("%d ",i);
    return 0;
}
```

2.7.2 缩点

```
//Tarjan 缩点
/* 1. 有向图缩点
将有向图中的所有强连通分量缩成一个点
缩完点之后原图就变成了一个 DAG 图（有向无环图）
模板例题
    给定一个 n 个点 m 条边有向图，每个点有一个权值，求一条路径，使路径经过的点权值之和最大。你只需要求出这个权值和。
允许多次经过一条边或者一个点，但是，重复经过的点，权值只计算一次。
解法：对整个图缩完点之后跑拓扑排序 DP 一下即可
*/
const int maxn = 1e4 + 10;
const int maxm = 1e5 + 10;
int val[maxn];
int Low[maxn],dfn[maxn],Stack[maxn],Belong[maxn],num[maxn];
int Index,top,scc;
bool Instack[maxn];
void Tarjan(int u){
    int v;
    Low[u] = dfn[u] = ++Index;
    Stack[top++] = u;
    Instack[u] = true;
    for(int i = head[u]; ~i; i = edge[i].next){
        v = edge[i].to;
        if(!dfn[v]){
            Tarjan(v);
            if(Low[u] > Low[v]) Low[u] = Low[v];
        }else if(Instack[v] && Low[u] > dfn[v]) Low[u] = dfn[v];
    }
    if(Low[u] == dfn[u]){
```

```
        scc++;
        do{
            v = Stack[--top];
            Instack[v] = false;
            Belong[v] = scc;
            num[scc]++;
        }while(v != u);
    }
}
int VAL[maxn],ind[maxn];
int dp[maxn];
vector<int>P[maxn];
void solve(){
    for(int i = 1; i <= N ; i ++) dfn[i] = Instack[i] = num[i] = 0;
    Index = scc = top = 0;
    for(int i = 1; i <= N ; i ++) if(!dfn[i]) Tarjan(i);   //将原图缩点为 DAG 图
    for(int i = 1; i <= N ; i ++){                          //建 DAG 图
        VAL[Belong[i]] += val[i];
        for(int j = head[i]; ~j; j = edge[j].next){
            int v = edge[j].to;
            if(Belong[v] != Belong[i]){
                P[Belong[i]].pb(Belong[v]);
                ind[Belong[v]]++;
            }
        }
    }
    queue<int>Q;
    for(int i = 1; i <= scc;i ++){
        if(!ind[i]) Q.push(i);
        dp[i] = VAL[i];
    }
    int ans = 0;
    while(!Q.empty()){            //拓扑排序
        int u = Q.front(); Q.pop();
        ans = max(ans,dp[u]);
        for(int i = 0 ; i < P[u].size(); i ++){
            int v = P[u][i];
            ind[v]--;
            dp[v] = max(dp[v],dp[u] + VAL[v]);
            if(!ind[v]) Q.push(v);
        }
    }
    Pri(ans);

}
int main(){
    N = read(); M = read(); init();
    for(int i = 1; i <= N ; i ++) val[i] = read();
    for(int i = 1; i <= M ; i ++){
        int u = read(),v = read();
        add(u,v);
    }
    solve();
```

```
    return 0;
}
//2. 无向图缩点
//将所有含有无向图环的点缩点，最终形成一棵树
//模板：找一条路使得路上的点权最大，同一个点只能计入一次贡献，要求不能走回头路
//做法：如果成环那么整个环都可以走且可以走回来环的路上，那么缩点形成一棵树之后树形 dp
const int maxn = 4e5 + 10;
int N,M,K;
struct Edge{
    int to,next;
}edge[maxn * 2];
int head[maxn],tot;
void init(){
    for(int i = 0 ; i <= N ; i ++) head[i] = -1;
    tot = 0;
}
void add(int u,int v){
    edge[tot].to = v;
    edge[tot].next = head[u];
    head[u] = tot++;
}
LL w[maxn];
int belong[maxn],cnt;
int dfn[maxn],low[maxn],num;
LL size[maxn],weight[maxn];//size 表示缩点后的点内点的数量，weight 表示缩点后的点的点权和
int Stack[maxn],top;
bool Instack[maxn];
void tarjan(int u,int la){
    dfn[u] = low[u] = ++num;
    Stack[++top] = u;
    Instack[u] = true;
    int v;
    for(int i = head[u]; ~i; i = edge[i].next){
        v = edge[i].to;
        if(v == la) continue;
        if(!dfn[v]){
            tarjan(v,u);
            low[u] = min(low[u],low[v]);
        }else{
            low[u] = min(low[u],dfn[v]);
        }
    }
    if(low[u] == dfn[u]){
        v = Stack[top];
        ++cnt;
        while(v != u){
            belong[v] = cnt;
            weight[cnt] += w[v];
            size[cnt]++;
            Instack[v] = false;
            v = Stack[--top];
        }
        --top;
```

```
            belong[u] = cnt;
            weight[cnt] += w[u];
            size[cnt]++;
            Instack[u] = false;
        }
    }
    vector<int>P[maxn];
    bool vis[maxn],ret[maxn];
    LL dp[maxn],dp2[maxn];
    void dfs(int u,int la){
        dp2[u] = weight[u];
        for(int i = 0 ; i < P[u].size(); i ++){
            int v = P[u][i];
            if(v != la && !vis[v]){
                vis[v] = 1; dfs(v,u);
                if(ret[v]) ret[u] = 1;
                dp[u] += dp[v];
                dp2[u] += dp[v];
            }
        }
        if(size[u] > 1) ret[u] = 1;
        if(ret[u]) dp[u] += weight[u];
        LL t = dp2[u];
        for(int i = 0 ; i < P[u].size(); i ++){
            int v = P[u][i];
            if(v == la) continue;
            dp2[u] = max(dp2[u],dp2[v] + t - dp[v]);
        }
    }
    int main(){
        Sca2(N,M); init();
        for(int i = 1; i <= N ; i ++) Scl(w[i]);
        for(int i = 1; i <= M ; i ++){
            int u,v; Sca2(u,v); add(u,v); add(v,u);
        }
        for(int i = 1; i <= N ; i ++) if(!dfn[i]) tarjan(i,i);
        for(int u = 1; u <= N ; u ++){
            for(int i = head[u]; ~i; i = edge[i].next){
                int v = edge[i].to;
                if(belong[u] == belong[v]) continue;
                P[belong[u]].push_back(belong[v]);
            }
        }
        int S = read(); vis[belong[S]] = 1;
        dfs(belong[S],-1);
        Prl(dp2[belong[S]]);
        return 0;
    }
```

2.7.3 点双

```
//点的双联通分量
//一个割点可能被多个点双包含, 一个割边不会被多个边双包含
const int maxn = 5e5 + 10;
```

```cpp
const int INF = 0x3f3f3f3f;
const int mod = 998244353;
int N,M,K;
struct Edge{
    int to,next;
    bool vis;
}edge[maxn * 2];
int head[maxn],tot;
void init(){
    for(int i = 0 ; i <= N ; i ++) head[i] = -1;
    tot = 0;
}
void add(int u,int v){
    edge[tot].to = v;
    edge[tot].vis = 0;
    edge[tot].next = head[u];
    head[u] = tot++;
}
int Low[maxn],dfn[maxn],Stack[maxn];
bool Instack[maxn],cut[maxn];
int Index,top;
int dcn; //点双的个数
vector<int>dcc[maxn];//每个点双含有的点
void Tarjan(int u,int la){
    Low[u] = dfn[u] = ++Index;
    Stack[++top] = u;
    for(int i = head[u]; ~i; i = edge[i].next){
        int v = edge[i].to;
        if(v == la) continue;
        if(!dfn[v]){
            Tarjan(v,u);
            if(Low[u] > Low[v]) Low[u] = Low[v];
            if(Low[v] >= dfn[u]){
                cut[u] = 1;
                dcn++;
                dcc[dcn].clear();
                dcc[dcn].pb(u);
                while(1){
                    int w = Stack[top--];
                    dcc[dcn].pb(w);
                    if(w == v) break;
                }
            }
        }else if(Low[u] > dfn[v]) Low[u] = dfn[v];
    }
}
LL q_p(LL a,LL b){
    LL ans = 1;
    while(b){
        if(b & 1) ans = ans * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
```

```
    return ans;
}
int main(){
    Sca2(N,M); init();
    for(int i = 1; i <= M ; i ++){
        int u,v; Sca2(u,v);
        add(u,v); add(v,u);
    }
    dcn = Index = top = 0;
    for(int i = 1; i <= N; i ++){
        Low[i] = dfn[i] = Stack[i] = Instack[i] = cut[i] = 0;
    }
    LL sum = 1;
    int cnt = 0;
    for(int i = 1; i <= N ; i ++){
        if(!dfn[i]) Tarjan(i,-1);
    }
    for(int i = 1; i <= dcn; i ++){
        int v = dcc[i].size();
        if(v == 2) cnt++;
        else if(v > 2) sum = sum * (q_p(2,v) - 1 + mod) % mod;
    }
    sum = sum * q_p(2,cnt) % mod;
    Prl(sum);

    return 0;
}
```

2.8 2-SAT

2.8.1 缩点法

```
//2-SAT
/*
2—SAT: 给出 n 个布尔变量以及一堆约束，让你寻找其中的可行解
约束的方式形如 1. 必须 (不) 选 a 2.a 与 b 必须选一个 (不能都选) 3.a,b 选择情况相同 (相反)
↪  等等最多由两个变量建立起的约束。
对于这一类的问题，考虑用图论的方式解决，将一个点 i 拆点为 i 与 i + N, i 表示这个值为 1, i + N
↪  表示为 0
a -> b 的边表示如果 a 则 b
构图方法: 以下 a 表示 a 这个点，a' 表示 a + N 这个点
1. 必选 a: a'->a      2. 必不选 a:a->a'
2.a,b 中选择一个:a'->b,b'->a     3.a,b 不都选:a->b',b->a'
4.a,b 选择情况相同 a->b,b->a,a'->b',b'->a'
5.a,b 选择情况相反 a->b',b->a',a'->b,b'->a
诸如此类。
*/
/*      Tarjan 缩点法
如果题目不要求字典序最小，只要求输出任意一组，我们可以考虑 Tarjan 缩点的做法
注意到同一个强连通分量里面的点必定互相满足，也就是说如果任意存在一个 i 使得 i 和 i' 在一个强
↪  连通分量内，
则说明题目无解，因为不可能一个点即为 false 又为 true。
其次，缩点之后会形成一个 DAG 图，如果拓扑序排在前面的满足，那么拓扑序排在后面的也需要满足。
```

```
为了防止出现形如 i 为 i' 的前驱，选了 i 之后会满足 i' 成立这样的错误情况出现，对于每个点都选取拓扑序较后面的那一个
也就是如果 i -> i'，则选 i 为 false 一定不会错，但这样的操作无法满足字典序最小，如果要求字典序，需要采用染色法
我们不必特意去跑拓扑，因为 Tarjan 标记出的强连通分量编号满足逆拓扑序，编号较小的点即为拓扑序在后面的点。
*/
const int maxn = 2e6 + 10;
const int INF = 0x3f3f3f3f;
const int mod = 1e9 + 7;
int N,M,K;
struct Edge{
    int to,next;
}edge[maxn << 1];
int head[maxn],tot;
void init(){
    for(int i = 0; i <= (N << 1); i ++) head[i] = -1;
    tot = 0;
}
void add(int u,int v){
    edge[tot].to = v;
    edge[tot].next = head[u];
    head[u] = tot++;
}
int Low[maxn],dfn[maxn],Stack[maxn],Belong[maxn],num[maxn];
int Index,top,scc;
bool Instack[maxn];
void Tarjan(int u){
    int v;
    Low[u] = dfn[u] = ++Index;
    Stack[top++] = u;
    Instack[u] = true;
    for(int i = head[u]; ~i ; i = edge[i].next){
        v = edge[i].to;
        if(!dfn[v]){
            Tarjan(v);
            if(Low[u] > Low[v]) Low[u] = Low[v];
        }else if(Instack[v] && Low[u] > dfn[v]) Low[u] = dfn[v];
    }
    if(Low[u] == dfn[u]){
        scc++;
        do{
            v = Stack[--top];
            Instack[v] = false;
            Belong[v] = scc;
            num[scc]++;
        }while(v != u);
    }
}
int main(){
    Sca2(N,M); init();
    for(int i = 1; i <= M ; i ++){
        int x,a,y,b;
```

```
        scanf("%d%d%d%d",&x,&a,&y,&b);
        add(x + a * N,y + (b ^ 1) * N);
        add(y + b * N,x + (a ^ 1) * N);
    }
    for(int i = 1; i <= (N << 1); i ++) if(!dfn[i]) Tarjan(i);
    for(int i = 1; i <= N; i ++){
        if(Belong[i] == Belong[i + N]){
            puts("IMPOSSIBLE");
            return 0;
        }
    }
    puts("POSSIBLE");
    for(int i = 1; i <= N ; i ++){
        if(Belong[i] < Belong[i + N]) printf("1 ");
        else printf("0 ");
    }
    return 0;
}
```

2.8.2 染色法

```
//染色法
/* 当题目对字典序有要求的时候，就只能采用比较朴素的方法
在原来的图上，很显然相邻的点不能同时取到，那就是寻找一个字典序最小的染色方法
从小到大枚举点，如果能给这个点涂色就涂上，否则给 i' 涂色，如果这也不行，则没有可行方案
*/
//模板：1，2//3，4//5，6 为各自组的点，为了使得 i' = i ^ 1，将所有编号--，变为 0,1//2,3//4,5
//给出的约束是 u,v 两点不能同时涂色
const int maxn = 20010;
const int INF = 0x3f3f3f3f;
const int mod = 1e9 + 7;
int N,M,K;
struct Edge{
    int to,next;
}edge[maxn << 2];
int head[maxn],tot;
void init(){
    for(int i = 0 ; i <= (N << 1) ; i ++) head[i] = -1;
    tot = 0;
}
void add(int u,int v){
    edge[tot].to = v;
    edge[tot].next = head[u];
    head[u] = tot++;
}
int color[maxn],top;
int Stack[maxn];
bool dfs(int t){
    if(color[t]) return true;
    if(color[t ^ 1]) return false;
    color[t] = 1;
    Stack[++top] = t;
    for(int i = head[t]; ~i; i = edge[i].next){
        int v = edge[i].to;
```

```
        if(!dfs(v)) return false;
    }
    return true;
}
bool solve(){
    for(int i = 0 ; i <= (N << 1); i ++) color[i] = 0;
    for(int i = 0; i < (N << 1) ; i += 2){
        if(!color[i] && !color[i + 1]){
            top = 0;
            if(dfs(i)) continue;
            for(int j = 1; j <= top; j ++) color[Stack[j]] = 0;
            if(!dfs(i + 1)) return false;
        }
    }
    return true;
}
int main(){
    while(~Sca2(N,M)){
        init();
        for(int i = 1; i <= M ; i ++){
            int u,v; Sca2(u,v);
            u--; v--;
            add(u,v ^ 1); add(v,u ^ 1);
        }
        if(solve()){
            for(int i = 0; i < (N << 1); i ++){
                if(color[i]) Pri(1 + i);
            }
        }else{
            puts("NIE");
        }
    }
    return 0;
}
```

2.9 斯坦纳树

```
/*  斯坦纳树
在一张图上寻找一颗边权和最小的生成树将指定的点集合连起来。
最小生成树事实上是斯坦纳树的一种特殊形式
一般给的点集不会过大 (n <= 16 左右)
求解方式是状压 dp,dp[i][1 << k] 表示在 i 点并且已经连起来了 state 状态的情况下的最小值
更新的状态用两种
1. 相同点的两个子状态合并 dp[i][sta] = min(dp[i][sta1] + dp[i][sta2])
2. 点与点之间的松弛 dp[i][sta] = min(dp[j][sta] + dis[i][j])
做法
(1) 所以我们从小到大枚举状态
(2) 先转移第一种：在相同状态下先利用较小的状态更新过来
(2) 更新完了之后利用 SPFA 松弛，注意 SPFA 不用考虑进阶到下一状态，只要保证当前状态最优就可以
↪  了
 */
 //luogoP4294 游览计划
 /*
 题意：给出一张网格图，点权为正代表联通这个点的代价，点权为 0 代表需要被联通的点集
```

```
 求最小联通所有点集中的点的代价以及输出需要被联通的点，输出'x'表示
  */
const int maxn = 12;
const int INF = 0x3f3f3f3f;
int N,M,K;
int MAP[maxn][maxn],id[maxn][maxn];
int dp[maxn][maxn][(1 << 11) + 10];
char ans[maxn][maxn];
struct node{
    int x,y,state,val;
    node(){}
    node(int x,int y,int state,int val = 0):x(x),y(y),state(state),val(val){}
    friend bool operator < (node a,node b){
        return a.val > b.val;
    }
};
node pre[maxn][maxn][(1 << 11) + 10];
const int a[4][2] = {0,1,1,0,0,-1,-1,0};
int cnt = 0;
bool check(int x,int y){
    return 0 <= x && x < N && 0 <= y && y < M;
}
void dfs(int x,int y,int state){
    if(x == N) return;
    if(MAP[x][y]) ans[x][y] = 'o';
    else ans[x][y] = 'x';
    node u = pre[x][y][state];
    dfs(u.x,u.y,u.state);
    if(u.x == x && u.y == y) dfs(u.x,u.y,state - u.state); //说明是两个子集合并更新过来的
}
queue<PII>Q;
bool vis[maxn][maxn];
int limit;
void SPFA(int state){   //不要去更新下一层，只要把同一 state 的更新掉就可以了
    while(!Q.empty()){
        PII u = Q.front(); Q.pop();
        vis[u.fi][u.se] = 0;
        for(int i = 0 ; i < 4; i ++){
            PII h = u;
            h.fi += a[i][0]; h.se += a[i][1];
            if(!check(h.fi,h.se)) continue;
            if(dp[h.fi][h.se][state] > dp[u.fi][u.se][state] + MAP[h.fi][h.se]){
                dp[h.fi][h.se][state] = dp[u.fi][u.se][state] + MAP[h.fi][h.se];
                pre[h.fi][h.se][state] = node(u.fi,u.se,state);
                if(!vis[h.fi][h.se]){
                    Q.push(h);
                    vis[h.fi][h.se] = 1;
                }
            }
        }
    }
}
void solve(){
```

```
    for(int i = 0 ; i < N ; i ++){
        for(int j = 0 ; j < M ; j ++){
            if(!MAP[i][j]){
                Pri(dp[i][j][limit]);
                dfs(i,j,limit);
                return;
            }
        }
    }
}
int main(){
    Sca2(N,M); cnt = 0;
    for(int i = 0; i < N ; i ++){
        for(int j = 0; j < M; j ++){
            Sca(MAP[i][j]); ans[i][j] = '_';
            if(!MAP[i][j]) id[i][j] = cnt++;
        }
    }
    limit = (1 << cnt) - 1; //1111111
    //初始化
    for(int i = 0; i < N ; i ++){
        for(int j = 0; j < M ; j ++){
            for(int k = 0 ; k <= limit; k ++) dp[i][j][k] = INF;
            dp[i][j][0] = 0; pre[i][j][0] = node(N,M,-1);
            if(!MAP[i][j]){
                dp[i][j][1 << id[i][j]] = 0;
                pre[i][j][1 << id[i][j]] = node(N,M,-1);
            }
        }
    }
    //状态转移
    for(int state = 0 ; state <= limit; state ++){ //从小到大枚举状态
        for(int i = 0 ; i < N ; i ++){
            for(int j = 0 ; j < M ; j ++){
                for(int sta = state;sta; sta = (sta - 1) & state){    //枚举 state 的所有
                ↪ 子集
                    if(dp[i][j][sta] + dp[i][j][state - sta] - MAP[i][j] <
                    ↪ dp[i][j][state]){
                        dp[i][j][state] = dp[i][j][sta] + dp[i][j][state - sta] -
                        ↪ MAP[i][j]; //子集间的更新
                        pre[i][j][state] = node(i,j,sta); //因为要输出方案，所以记录前驱
                    }
                }
                if(dp[i][j][state] < INF){
                    Q.push(mp(i,j));
                    vis[i][j] = 1;
                }
            }
        }
        SPFA(state); //当前状态全部从子状态更新上来之后 SPFA 松弛到最优
    }
    solve();
    for(int i = 0 ; i < N ; i ++){
```

```
        for(int j = 0 ; j < M; j ++){
            printf("%c",ans[i][j]);
        }
        puts("");
    }
    return 0;
}
```

2.10 二分图匹配

```
//二分图匹配 匈牙利算法
//模板：左边 N 个点匹配右边 M 个点，总共 K 条边的最大匹配数
const int maxn = 1010;
int linker[maxn];
int vis[maxn];
bool dfs(int u){
    for(int i = head[u]; ~i ; i = edge[i].next){
        int v = edge[i].to;
        if(!vis[v]){
            vis[v] = true;
            if(linker[v] == -1 || dfs(linker[v])){
                linker[v] = u;
                return true;
            }
        }
    }
    return false;
}
int hungary(){
    int ans = 0;
    for(int i = 1; i <= M ; i ++) linker[i] = -1;    //注意是右边点的 linker
    for(int i = 1; i <= N ; i ++){
        for(int j = 1; j <= M ; j ++) vis[j] = 0;
        if(dfs(i)) ans++;
    }
    return ans;
}
int main(){
    Sca2(N,M); K = read(); init();
    for(int i = 1; i <= K; i ++){
        int u,v; Sca2(u,v);
        if(u > N || v > M) continue;
        add(u,v);
    }
    Pri(hungary());
    return 0;
}
```

2.11 KM 算法

```
/*KM 算法
二分图最大权匹配，时间复杂度 O(n^3)
1. 最大权匹配不一定是最大匹配
2. 若要求最小权匹配，就边权取反然后求最大权，将答案取反
```

```
3. 对于顶标 lx[i],ly[i] 的理解
lx 初始为 i 点连出去的最大的边权，ly 初始为 0
可行顶标：在任意时刻，对于任意边 (u,v,w) 都满足 lx[u] + ly[v] >= w
同时，满足 lx[u] + ly[v] >= MAP[u][v] 约束下的最小顶标和 ( lx + ly) 就是最大权匹配
满足 lx[u] + ly[v] <= MAP[u][v] 约束下的最大顶标和就是最小权匹配
*/
//模板：求最小权匹配
const int maxn = 210;
const LL INF = 1e18;
LL MAP[maxn][maxn];
LL vx[maxn],vy[maxn],lx[maxn],ly[maxn],slack[maxn];
int pre[maxn],Left[maxn],Right[maxn],NL,NR,Nm,N;
void match(int &u){
    for(;u;swap(u,Right[pre[u]])) Left[u] = pre[u];
}
void bfs(int u){
    static int q[maxn],front,rear;
    front = 0; vx[q[rear = 1] = u] = true;
    while(1){
        while(front < rear){
            int u = q[++front];
            for(int v = 1; v <= Nm; v ++){
                int tmp;
                if(vy[v] || (tmp = lx[u] + ly[v] - MAP[u][v]) > slack[v]) continue;
                pre[v] = u;
                if(!tmp){
                    if(!Left[v]) return match(v);
                    vy[v] = vx[q[++rear] = Left[v]] = true;
                }else slack[v] = tmp;
            }
        }
        LL a = INF;
        for(int i = 1; i <= Nm ; i ++)
            if(!vy[i] && a > slack[i]) a = slack[u = i];
        for(int i = 1; i <= Nm ; i ++){
            if(vx[i]) lx[i] -= a;
            if(vy[i]) ly[i] += a;
            else slack[i] -= a;
        }
        if(!Left[u]) return match(u);
        vy[u] = vx[q[++rear] = Left[u]] = true;
    }
}
void solve(){
    for(int i = 1; i <= Nm; i ++){
        for(int j = 1; j <= Nm; j++){
            slack[j] = INF;
            vx[j] = vy[j] = false;
        }
        bfs(i);
    }
}
LL KM(int nl,int nr){
```

```
    NL = nl; NR = nr;
    Nm = max(NL,NR);
    for(int i = 1; i <= Nm; i ++){
        lx[i] = -INF; ly[i] = 0;
        pre[i] = Left[i] = Right[i] = 0;
        for(int j = 1; j <= Nm; j ++){
            lx[i] = max(lx[i],MAP[i][j]);
        }
    }
    solve();
    LL ans = 0;
    for(int i = 1; i <= Nm; i ++) ans += lx[i] + ly[i];
    return ans;
}
void output(){ //输出左边和右边匹配的点
    for(int i = 1; i <= NL; i ++){
        printf("%d ",(MAP[i][Right[i]]?Right[i]:0));
    }
    puts("");
}
int main(){
    int T; scanf("%d",&T); int CASE = 1;
    while(T--){
        scanf("%d",&N);
        for(int i = 1; i <= N ; i ++){
            for(int j = 1; j <= N ; j ++){
                scanf("%lld",&MAP[i][j]);
                MAP[i][j] = -MAP[i][j];
            }
        }
        printf("Case #%d: %lld\n",CASE++,-KM(N,N));
    }
    return 0;
}
```

2.12 最大流

2.12.1 Dinic

```
//最大流 dinic 算法 N 个点 M 条边求 S 到 T 的最大流
//BFS 将残余网络分层, DFS 在 dep[v] = dep[u] + 1 的限制下找增广路
//注意：如果 dinic 写 T 了, 要关注 dfs 里有没有用 cur 数组优化
const int maxn = 10010;
const int maxm = 100000;
const int INF = 0x3f3f3f3f;
int N,M,K;
struct Dinic{
    struct Edge{
        int from,to,next,cap,flow;
        Edge(){}
        Edge(int from,int to,int next,int cap,int
        ↪ flow):from(from),to(to),next(next),cap(cap),flow(flow){}
    }edge[maxm * 2];      //记得 * 2
    int n,s,t,head[maxn],tot;
```

```
int dep[maxn],cur[maxn];
void init(int n,int s,int t){
    this->n = n; this->s = s; this->t = t; //记得三个 this 都要
    tot = 0;
    for(int i = 0 ; i <= n ; i ++) head[i] = -1;
}
inline void AddEdge(int s,int t,int w){
    edge[tot] = Edge(s,t,head[s],w,0);
    head[s] = tot++;
    edge[tot] = Edge(t,s,head[t],0,0);
    head[t] = tot++;
}
inline bool BFS(){
    for(int i = 0 ; i <= n ; i ++) dep[i] = -1;
    dep[s] = 1;
    queue<int>Q; Q.push(s);
    while(!Q.empty()){
        int u = Q.front(); Q.pop();
        for(int i = head[u]; ~i ; i = edge[i].next){
            int v = edge[i].to;
            if(~dep[v] || edge[i].flow >= edge[i].cap) continue;
            dep[v] = dep[u] + 1;
            Q.push(v);
        }
     }
     return ~dep[t];
}
inline int DFS(const int& u,int a){
    if(u == t || !a) return a;
    int flow = 0;
    for(int &i = cur[u]; ~i ; i = edge[i].next){
        int v = edge[i].to;
        if(dep[v] != dep[u] + 1) continue;
        int f = DFS(v,min(a,edge[i].cap - edge[i].flow));
        if(!f) continue;
        edge[i ^ 1].flow -= f;
        edge[i].flow += f;
        a -= f;
        flow += f;
    }
    return flow;
}
inline int maxflow(){
    return maxflow(s,t);
}
inline int maxflow(int s,int t){
    int flow = 0;
    while(BFS()){
        for(int i = 0; i <= n ; i ++) cur[i] = head[i];
        flow += DFS(s,INF);
    }
    return flow;
}
```

```
};
int main(){
    int S,T;
    Sca2(N,M); Sca2(S,T);
    g.init(N,S,T);
    while(M--){
        int u,v,w; Sca3(u,v,w);
        g.AddEdge(u,v,w);
    }
    Pri(g.maxflow());
    return 0;
}
```

2.12.2 最小割

```
/*            最小割
最小割边：为了使原点（记为 S）和汇点（记为 T）不连通，最少要割几条边
最小割 = 最大流，求最大流即可

最小割点
删去最少的点使原图不联通
假设原来的点编号为 i，总共有 n 个点，那么我们就把每个点拆成两个点，编号分别为 i 和 i+n。
其中点 i 负责连接原图中连入这个点的边，点 i+n 负责连原图中连出这个点的边。
i 和 i + n 之间有一条容量为 1 的边，其余边都为 INF
删点的过程相当于删去 i 到 i + n 之间这条容量为 1 的边，使得其余点都不能通过这个点到其他的点
如果有不能删去的点，就在把 i 到 i + n 之间的容量变为 INF
*/
//模板：求起点到终点的最小割点
const int maxn = 1010;
const int maxm = 3010;
const int INF = 0x3f3f3f3f;
int N,M,K;
struct Dinic{
    struct Edge{
        int to,next,cap,flow;
        Edge(){}
        Edge(int to,int next,int cap,int flow):to(to),next(next),cap(cap),flow(flow){}
    }edge[maxm * 2];
    int head[maxn],tot;
    int pre[maxn],s,t,n,dis[maxn];
    void init(int n){
        this->n = n;
        for(int i = 0 ; i <= n ; i ++) head[i] = -1;
        tot = 0;
    }
    void add(int u,int v,int w){
        edge[tot] = Edge(v,head[u],w,0);
        head[u] = tot++;
        edge[tot] = Edge(u,head[v],0,0);
        head[v] = tot++;
    }
    bool BFS(){
        for(int i = 0 ; i <= n ; i ++) dis[i] = -1;
        dis[s] = 0;
```

```
        queue<int>Q; Q.push(s);
        while(!Q.empty()){
            int u = Q.front(); Q.pop();
            for(int i = head[u]; ~i ; i = edge[i].next){
                int v = edge[i].to;
                if(~dis[v] || edge[i].cap <= edge[i].flow) continue;
                dis[v] = dis[u] + 1;
                Q.push(v);
            }
        }
        return ~dis[t];
    }
    int DFS(int u,int a){
        if(u == t || !a) return a;
        int flow = 0;
        for(int& i = pre[u]; ~i ; i = edge[i].next){
            int v = edge[i].to;
            if(dis[v] == dis[u] + 1){
                int f = DFS(v,min(a,edge[i].cap - edge[i].flow));
                if(!f) continue;
                flow += f;
                a -= f;
                edge[i].flow += f;
                edge[i ^ 1].flow -= f;
            }
        }
        return flow;
    }
    int maxflow(int s,int t){
        this->s = s; this->t = t;
        int flow = 0;
        while(BFS()){
            for(int i = 0 ; i <= n; i ++) pre[i] = head[i];
            flow += DFS(s,INF);
        }
        return flow;
    }
}g;
int main(){
    Sca2(N,M); g.init(N + N);
    int S,T; Sca2(S,T);
    for(int i = 1; i <= N ; i ++) g.add(i,i + N,1);
    g.add(S,S + N,INF);
    g.add(T,T + N,INF);
    for(int i = 1; i <= M; i ++){
        int u,v; Sca2(u,v);
        g.add(u + N,v,INF);
        g.add(v + N,u,INF);
    }
    Pri(g.maxflow(S,T + N));
    return 0;
}
```

2.12.3 最大权闭合子图

```
/*      最大权闭合子图
对于一个图而言，如果一个子图中所有点都满足这个点的后继点都在子图内，这就是一个闭合子图
满足点权和最大的闭合子图就是最大权闭合子图
例如有一些实验，完成一个实验需要有一些器材，实验会赚钱器材会花钱，
寻找一个最赚钱的买器材做实验的方案，答案就是最大权闭合子图的总和
做实验赚钱的点权为正，指向那些需要的器材，器材点权为负
最大权闭合子图的解法：
建立源点 S 连接到所有正点权上，容量为点权值
建立汇点 T 被所有负点权连接，容量大小为点权值取绝对值
原本的点之间的边照样建，容量为 INF
然后答案就是所有正权值的和 - S 到 T 的最大流

理解：假设一开始就把所有的实验经费都收入囊中，然后我们需要放弃一些来满足调节。
要么选择不做这个实验，要么选择购买这些器材，使得满足条件的最小花费就是这个图的最小割

方案：在跑完最大流之后的残余网络中，依然和源点 S 相连的所有点就是必须选择的器材和必须做的实验
*/
//例题洛谷 P 2762：跑一个图中的最大权闭合子图并输出方案
//1 - N 为正权点，N + 1 - N + M 为负权点
const int maxn = 1010;
const int maxm = 10010;
const int INF = 0x3f3f3f3f;
int N,M,K;
int vis[maxn];
struct Dinic{
    struct Edge{
        int to,next,cap,flow;
        Edge(){}
        Edge(int to,int next,int cap,int flow):to(to),next(next),cap(cap),flow(flow){}
    }edge[maxn * 2];
    int head[maxn],dis[maxn],tot,pre[maxn];
    int n,s,t;
    void init(int n,int s,int t){
        this->n = n; this->s = s; this->t = t;
        for(int i = 0 ; i <= n ; i ++) head[i] = -1;
        tot = 0;
    }
    void add(int u,int v,int w){
        edge[tot] = Edge(v,head[u],w,0);
        head[u] = tot++;
        edge[tot] = Edge(u,head[v],0,0);
        head[v] = tot++;
    }
    bool BFS(){
        for(int i = 0 ; i <= n ; i ++) dis[i] = -1;
        queue<int>Q;
        dis[s] = 1; Q.push(s);
        while(!Q.empty()){
            int u = Q.front(); Q.pop();
            for(int i = head[u]; ~i; i = edge[i].next){
                int v = edge[i].to;
                if(~dis[v] || edge[i].cap <= edge[i].flow) continue;
```

```
                dis[v] = dis[u] + 1; Q.push(v);
            }
        }
        return ~dis[t];
    }
    int dfs(int u,int a){
        if(u == t || !a) return a;
        int flow = 0;
        for(int& i = pre[u]; ~i; i = edge[i].next){
            int v = edge[i].to;
            if(dis[v] != dis[u] + 1) continue;
            int f = dfs(v,min(edge[i].cap - edge[i].flow,a));
            if(!f) continue;
            edge[i].flow += f;
            edge[i ^ 1].flow -= f;
            flow += f;
            a -= f;
        }
        return flow;
    }
    int maxflow(int s,int t){
        int flow = 0;
        while(BFS()){
            for(int i = 0 ; i <= n ; i ++) pre[i] = head[i];
            flow += dfs(s,INF);
        }
        return flow;
    }
    int maxflow(){return maxflow(s,t);}
    void show(int t){
        vis[t] = 1;
        for(int i = head[t]; ~i ; i = edge[i].next){
            int v = edge[i].to;
            if(vis[v] || edge[i].cap <= edge[i].flow) continue;
            show(v);
        }
    }
}g;
int main(){
    cin >> N >> M; getchar();
    int s = N + M + 1,t = N + M + 2;
    g.init(N + M + 2,s,t);
    int sum = 0;
    for(int i = 1; i <= N ; i ++){
        int x; Sca(x);
        sum += x;
        g.add(s,i,x);
        while(1){
            char c;
            scanf("%d%c",&x,&c);
            g.add(i,x + N,INF);
            if(c == '\n' || c == '\r') break;
        }
```

```
    }
    for(int i = 1; i <= M ; i ++){
        int x; Sca(x);
        g.add(i + N,t,x);
    }
    int p = g.maxflow();
    g.show(s);                    //找一下那些点被选中了
    for(int i = 1; i <= N ; i++){
        if(vis[i]) printf("%d ",i);            //输出与 S 相连的正权点
    }
    puts("");
    for(int i = 1 + N; i <= M + N ; i ++){      //输出与 T 相连的负权点
        if(vis[i]) printf("%d ",i - N);
    }
    puts("");
    Pri(sum - p);
    return 0;
}
```

2.13 最小覆盖

2.13.1 DAG 最小点路径覆盖

```
/*
    有向无环图的最小点路径覆盖
    路径不相交，每个点属于一条路径（路径长度可为 0），求最少路径
    一开始最多有 n 条路径，对于每两个点的合并就会使路径减一
    所以将点 i 拆为 i 和 i + n 两个点，i 代表入点，i + n 代表出点
    S 与入点连边，出点与 T 连边，容量均为 1
    对于每条边 u,v，将 u 与 v + N 连边，容量均为 1
    S 到 T 的最大流就是最多的可合并点，n - maxflow 就是最小路径覆盖
    如果要求方案的话，如果 u - v + N 这条边上有流量，代表这两个点被合并了
*/
//洛谷 P2764 求 N 点 M 路的最小路径覆盖数量和方案
const int maxn = 410;
const int maxm = 20010;
int N,M,K;
struct Dinic{
    struct Edge{
        int to,next,cap,flow;
        Edge(){}
        Edge(int to,int next,int cap,int flow):to(to),next(next),cap(cap),flow(flow){}
    }edge[maxm * 2];
    int head[maxn],tot,n,s,t;
    int dis[maxn],pre[maxn];
    int nxt[maxn],vis[maxn];
    void init(int n,int s,int t){
        this->n = n; this->s = s; this->t = t;
        for(int i = 0 ; i <= n ; i ++) head[i] = -1;
        tot = 0;
    }
    void add(int u,int v,int w){
        edge[tot] = Edge(v,head[u],w,0);
        head[u] = tot++;
```

```
    edge[tot] = Edge(u,head[v],0,0);
    head[v] = tot++;
}
bool BFS(){
    for(int i = 0 ; i <= n ; i ++) dis[i] = -1;
    dis[s] = 0;
    queue<int>Q;
    Q.push(s);
    while(!Q.empty()){
        int u = Q.front(); Q.pop();
        for(int i = head[u]; ~i ; i = edge[i].next){
            int v = edge[i].to;
            if(~dis[v] || edge[i].cap <= edge[i].flow) continue;
            dis[v] = dis[u] + 1;
            Q.push(v);
        }
    }
    return ~dis[t];
}
int dfs(int u,int a){
    if(u == t || !a) return a;
    int flow = 0;
    for(int& i = pre[u]; ~i ; i = edge[i].next){
        int v = edge[i].to;
        if(dis[v] != dis[u] + 1) continue;
        int f = dfs(v,min(a,edge[i].cap - edge[i].flow));
        if(!f) continue;
        edge[i].flow += f;
        edge[i ^ 1].flow -= f;
        a -= f;
        flow += f;
    }
    return flow;
}
int maxflow(int s,int t){
    int flow = 0;
    while(BFS()){
        for(int i = 0 ; i <= n ; i ++) pre[i] = head[i];
        flow += dfs(s,INF);
    }
    return flow;
}
int maxflow(){
    return maxflow(s,t);
}
void show(){
    for(int i = 1; i <= N ; i ++){
        for(int j = head[i]; ~j; j = edge[j].next){
            int v = edge[j].to;
            if(!edge[j].flow) continue;
            if(1 + N <=v && v <= N + M){
                nxt[i] = v - N;
                vis[v - N] = 1;
```

```
                    break;
                }
            }
        }
        for(int i = 1; i <= N; i ++){
            if(!vis[i]){
                int t = i;
                while(t){
                    printf("%d ",t);
                    t = nxt[t];
                }
                puts("");
            }
        }
    }
}g;
int main(){
    Sca2(N,M);
    int S = 2 * N + 1,T = 2 * N + 2;
    g.init(2 * N + 2,S,T);
    for(int i = 1; i <= N ; i ++){
        g.add(S,i,1);
        g.add(i + N,T,1);
    }
    for(int i = 1; i <= M ; i ++){
        int u,v; Sca2(u,v);
        g.add(u,v + N,1);
    }
    int ans = N - g.maxflow();
    g.show();
    Pri(ans);
    return 0;
}
```

2.14 费用流

2.14.1 SPFA 费用流

```
//最小费用最大流
/*
    不断寻找费用最小的增广路，贪心的使得每一条找到的增广路都是当前残余网络下费用最小的路径
    最终寻找到流量最大前提下费用最小的路
*/
//模板 N 点 M 边求 S 到 T 的最小费用最大流
const int maxn = 5010;
const int maxm = 50010;
const int INF = 0x3f3f3f3f;
int N,M,K;
struct Mcmf{
    struct Edge{
        int to,next,cap,flow,cost;
        Edge(){}
        Edge(int to,int next,int cap,int flow,int
        ↪  cost):to(to),next(next),cap(cap),flow(flow),cost(cost){}
```

```
}edge[maxm * 2];
int n,head[maxn],tot;
int pre[maxn],dis[maxn],vis[maxn];
void init(int n){
    this->n = n;
    tot = 0;
    for(int i = 0 ; i <= n ; i ++) head[i] = -1;
}
void add(int u,int v,int cap,int cost){
    edge[tot] = Edge(v,head[u],cap,0,cost);
    head[u] = tot++;
    edge[tot] = Edge(u,head[v],0,0,-cost);
    head[v] = tot++;
}
bool spfa(int s,int t){
    for(int i = 0 ; i <= n ; i ++){
        vis[i] = 0;
        dis[i] = INF;
        pre[i] = -1;
    }
    dis[s] = 0;
    queue<int>Q; Q.push(s);
    while(!Q.empty()){
        int u = Q.front(); Q.pop();
        vis[u] = 0;
        for(int i = head[u]; ~i; i = edge[i].next){
            int v = edge[i].to;
            if(edge[i].cap - edge[i].flow > 0 && dis[v] > dis[u] + edge[i].cost){
                dis[v] = dis[u] + edge[i].cost;
                pre[v] = i;
                if(!vis[v]){
                    vis[v] = 1;
                    Q.push(v);
                }
            }
        }
    }
    return ~pre[t];
}
int mcmf(int s,int t,int &cost){
    int flow = 0;
    cost = 0;
    while(spfa(s,t)){
        int Min = INF;
        for(int i = pre[t]; ~i ; i = pre[edge[i ^ 1].to]){
            Min = min(Min,edge[i].cap - edge[i].flow);
        }
        flow += Min;
        for(int i = pre[t]; ~i ; i = pre[edge[i ^ 1].to]){
            edge[i].flow += Min;
            edge[i ^ 1].flow -= Min;
            cost += Min * edge[i].cost;
        }
```

```
        }
        return flow;
    }
}g;
int main(){
    int S,T;
    scanf("%d%d%d%d",&N,&M,&S,&T);
      g.init(N);
    for(int i = 1; i <= M ; i ++){
        int u,v,cap,cost;
        scanf("%d%d%d%d",&u,&v,&cap,&cost);
        g.add(u,v,cap,cost);
    }
    int maxflow,cost;
    maxflow = g.mcmf(S,T,cost);
    printf("%d %d",maxflow,cost);
    return 0;
}
```

2.15 有上下界的网络流

2.15.1 无源汇上下界可行流

```
//无源汇上下界可行流
/*
模型：一个网络，求出一个流，使得每条边的流量必须 >=Li 且 <=Hi，每个点必须满足总流入量 = 总流
↪ 出量（流量守恒）(这个流的特点是循环往复，无始无终).
求解方法：1. 以每条边的最低容量形成一个初始流（不一定平衡）
2. 求出初始流中每个点进入流量和输出流量的差
3. 开始建图（附加流）：建立一个源点和汇点，原图上所有边建立一条容量为 Hi - Li 的边，a[i] 表示
↪ 初始流中输入流量-输出流量，如果 a[i] > 0,
则将 i 向汇点连容量为 a[i] 的边，如果 a[i] < 0 则源点向 i 连容量为 a[i] 的边
4. 可以证明所有 a[i] 之和相等，因此在原图上跑最大流，如果最大流与正数 a[i] 之和相等（即满流）
↪ 则说明有解，解就是
附加流上的流量加上初始流的流量
 */
const int maxn = 210;
const int maxm = 11010;
const int INF = 0x3f3f3f3f;
const int mod = 1e9 + 7;
int N,M,K;
struct E{
    int u,v,l,r;
}e[maxm];
int F[maxm];
int a[maxn];
struct dinic{
    struct Edge{
        int to,next,flow,cap,id;
        Edge(){}
        Edge(int to,int next,int flow, int cap,int id):
            to(to),next(next),flow(flow),cap(cap),id(id){}
    }edge[maxm * 2];
    int head[maxn],tot;
```

```
int s,n,t,cur[maxn],dep[maxn];
void init(int n,int s,int t){
    this->n = n; this->s = s; this->t = t;
    for(int i = 0 ; i <= n ; i ++) head[i] = -1;
    tot = 0;
}
void add(int u,int v,int w,int id){
    edge[tot] = Edge(v,head[u],0,w,id);
    head[u] = tot++;
    edge[tot] = Edge(u,head[v],0,0,id);
    head[v] = tot++;
}
inline bool bfs(){
    for(int i = 0 ; i <= n ; i ++) dep[i] = -1;
    dep[s] = 1;
    queue<int>Q; Q.push(s);
    while(!Q.empty()){
        int u = Q.front(); Q.pop();
        for(int i = head[u]; ~i ; i = edge[i].next){
            int v = edge[i].to;
            if(~dep[v] || edge[i].flow >= edge[i].cap) continue;
            dep[v] = dep[u] + 1;
            Q.push(v);
        }
    }
    return ~dep[t];
}
inline int dfs(const int &u,int a){
    if(u == t || !a) return a;
    int flow = 0;
    for(int &i = cur[u]; ~i ; i = edge[i].next){
        int v = edge[i].to;
        if(dep[v] != dep[u] + 1) continue;
        int f = dfs(v,min(a,edge[i].cap - edge[i].flow));
        flow += f;
        a -= f;
        edge[i].flow += f;
        edge[i ^ 1].flow -= f;
    }
    return flow;
}
int maxflow(){
    return maxflow(s,t);
}
int maxflow(int s,int t){
    int flow = 0;
    while(bfs()){
        for(int i = 0 ; i <= n ; i ++) cur[i] = head[i];
        flow += dfs(s,INF);
    }
    return flow;
}
void solve(){
```

```
        for(int i = 0 ; i <= n ; i ++){
            for(int j = head[i]; ~j; j = edge[j].next){
                if(edge[j].flow > 0) F[edge[j].id] += edge[j].flow;
            }
        }
    }
}g;
int main(){
    Sca2(N,M);
    int S = N + 1,T = N + 2;
    g.init(N + 2,S,T);
    for(int i = 1; i <= M ; i ++){
        scanf("%d%d%d%d",&e[i].u,&e[i].v,&e[i].l,&e[i].r);
        a[e[i].u] += e[i].l; a[e[i].v] -= e[i].l;     //a[i] 表示初始流在这个点入流量和出流
        ↪ 量的差
        g.add(e[i].u,e[i].v,e[i].r - e[i].l,i);       //在附加流上建边
    }
    int sum = 0;
    for(int i = 1; i <= N ; i ++){
        if(a[i] > 0){
            g.add(i,T,a[i],0);
            sum += a[i];              //表示汇出的总流量
        }
        if(a[i] < 0) g.add(S,i,-a[i],0);
    }
    if(g.maxflow() != sum) puts("NO");   //不满流则无解
    else{
        g.solve();
        puts("YES");
        for(int i = 1; i <= M; i ++){
            Pri(e[i].l + F[i]);
        }
    }
    return 0;
}
```

2.15.2 有源汇上下界网络流

```
/*
1. 有源汇上下界可行流
首先有源汇与无源汇的区别在于：有源汇可行流中，源点和汇点不一定流量平衡，但是源点多余的流量 =
↪ 汇点的多余流量。
解决有源汇问题时我们通常将其转化为无源汇的问题，具体方法就是连一条 (T,S,∞,0) 的边。这样的话，
↪ 汇点多余的输出就可以通过这条边进入源点。
然后我们建立超级源 SS 与超级汇 TT，之后判断是否有可行流就用前面的方法
2. 有源汇上下界最大流
因为 S 与 T 之间可能存在富余的边，导致没有得到最大流。
我们将超级源和超级汇以及与他们相连的边拆掉，然后在残余网络上在跑一次最大流。得到新的最大流与
↪ w 的和就是答案。
3. 有源汇上下界最小流
法 1：与最大流类似。我们先得到了 w 之后我们尽量减少 S 和 T 之间的流量。于是我们拆了超级源和超
↪ 级汇之后“反着”跑最大流，也就是拿 T 当源点，S 当汇点跑最大流。可以这么理解：反边增加流量
↪ 相当于正边减少流量。答案就是 w-最大流。
```

```
法 2：当然也可以不用改变图，直接反着跑最大流，答案就是 INF-最大流。这里 INF 就是我们
↪ (T,S,∞,0) 的上界。
 */
 //有源汇上下界最大流
const int maxn = 410;
const int maxm = 2e5 + 10;
const int INF = 0x3f3f3f3f;
int N,M,K,S,T;
int a[maxn],F[maxm];
struct E{
    int u,v,l,r;
}e[maxm];
struct dinic{
    struct Edge{
        int to,next,flow,cap,id;
        Edge(){}
        Edge(int to,int next,int flow, int cap,int id):
            to(to),next(next),flow(flow),cap(cap),id(id){}
    }edge[maxm * 2];
    int head[maxn],tot;
    int s,n,t,cur[maxn],dep[maxn];
    void init(int n){
        this->n = n;
        for(int i = 0 ; i <= n ; i ++) head[i] = -1;
        tot = 0;
    }
    void add(int u,int v,int w,int id){
        edge[tot] = Edge(v,head[u],0,w,id);
        head[u] = tot++;
        edge[tot] = Edge(u,head[v],0,0,id);
        head[v] = tot++;
    }
    inline bool bfs(){
        for(int i = 0 ; i <= n ; i ++) dep[i] = -1;
        dep[s] = 1;
        queue<int>Q; Q.push(s);
        while(!Q.empty()){
            int u = Q.front(); Q.pop();
            for(int i = head[u]; ~i ; i = edge[i].next){
                int v = edge[i].to;
                if(~dep[v] || edge[i].flow >= edge[i].cap) continue;
                dep[v] = dep[u] + 1;
                Q.push(v);
            }
        }
        return ~dep[t];
    }
    inline int dfs(const int &u,int a){
        if(u == t || !a) return a;
        int flow = 0;
        for(int &i = cur[u]; ~i ; i = edge[i].next){
            int v = edge[i].to;
            if(dep[v] != dep[u] + 1) continue;
```

```
            int f = dfs(v,min(a,edge[i].cap - edge[i].flow));
            flow += f;
            a -= f;
            edge[i].flow += f;
            edge[i ^ 1].flow -= f;
            if(!a) return flow;
        }
        return flow;
    }
    int maxflow(int s,int t){
        this->s = s; this->t = t;
        int flow = 0;
        while(bfs()){
            for(int i = 0 ; i <= n ; i ++) cur[i] = head[i];
            flow += dfs(s,INF);
        }
        return flow;
    }
    void solve(){
        for(int i = 0 ; i <= n ; i ++){
            for(int j = head[i]; ~j; j = edge[j].next){
                if(edge[j].flow > 0) F[edge[j].id] += edge[j].flow;
            }
        }
    }
    void del(int S,int T,int SS,int TT){
        for(int i = head[T]; ~i; i = edge[i].next){
            if(edge[i].cap == INF){
                edge[i].cap = edge[i].flow = 0;
                edge[i ^ 1].cap = edge[i ^ 1].flow = 0;
            }
        }
        for(int i = head[SS]; ~i; i = edge[i].next){
            edge[i].cap = edge[i].flow = 0;
            edge[i ^ 1].cap = edge[i ^ 1].flow = 0;
        }
        for(int i = head[TT]; ~i; i = edge[i].next){
            edge[i].cap = edge[i].flow = 0;
            edge[i ^ 1].cap = edge[i ^ 1].flow = 0;
        }
    }
}g;
int main(){
    scanf("%d%d%d%d",&N,&M,&S,&T);
    int SS = N + 1,TT = N + 2;
    g.init(N + 2);
    for(int i = 1; i <= M ; i ++){
        scanf("%d%d%d%d",&e[i].u,&e[i].v,&e[i].l,&e[i].r);
        g.add(e[i].u,e[i].v,e[i].r - e[i].l,i);
        a[e[i].u] += e[i].l; a[e[i].v] -= e[i].l;
    }
    g.add(T,S,INF,0);
    int sum = 0;
```

```
    for(int i = 1; i <= N; i ++){
        if(a[i] > 0){
            g.add(i,TT,a[i],M + 1);
            sum += a[i];
        }
        else if(a[i] < 0) g.add(SS,i,-a[i],M + 1);
    }
    if(sum != g.maxflow(SS,TT)){
        puts("please go home to sleep");
        return 0;
    }
    //方法 1. 删除加入的边之后再残余网络上跑最大流
    // g.solve();
    // g.del(S,T,SS,TT);   //删除超级源点汇点以及加的 T 到 S 的边
    // int ans = g.maxflow(S,T) + F[0];
    //方法 2. 求解可行流并判断可行后，原封不动地进行一次最大流，这个最大流就是答案
    int ans = g.maxflow(S,T);
    Pri(ans);
    return 0;
}
//有源汇上下界最小流
const int maxn = 50010;
const int maxm = 2e5 + 10;
const int INF = 0x3f3f3f3f;
const int mod = 1e9 + 7;
int N,M,K,S,T;
int a[maxn],F[maxm];
struct E{
    int u,v,l,r;
}e[maxm];
struct dinic{
    struct Edge{
        int to,next,flow,cap,id;
        Edge(){}
        Edge(int to,int next,int flow, int cap,int id):
            to(to),next(next),flow(flow),cap(cap),id(id){}
    }edge[maxm * 2];
    int head[maxn],tot;
    int s,n,t,cur[maxn],dep[maxn];
    void init(int n){
        this->n = n;
        for(int i = 0 ; i <= n ; i ++) head[i] = -1;
        tot = 0;
    }
    void add(int u,int v,int w,int id){
        edge[tot] = Edge(v,head[u],0,w,id);
        head[u] = tot++;
        edge[tot] = Edge(u,head[v],0,0,id);
        head[v] = tot++;
    }
    inline bool bfs(){
        for(int i = 0 ; i <= n ; i ++) dep[i] = -1;
        dep[s] = 1;
```

```
    queue<int>Q; Q.push(s);
    while(!Q.empty()){
        int u = Q.front(); Q.pop();
        for(int i = head[u]; ~i ; i = edge[i].next){
            int v = edge[i].to;
            if(~dep[v] || edge[i].flow >= edge[i].cap) continue;
            dep[v] = dep[u] + 1;
            Q.push(v);
        }
    }
    return ~dep[t];
}
inline int dfs(const int &u,int a){
    if(u == t || !a) return a;
    int flow = 0;
    for(int &i = cur[u]; ~i ; i = edge[i].next){
        int v = edge[i].to;
        if(dep[v] != dep[u] + 1) continue;
        int f = dfs(v,min(a,edge[i].cap - edge[i].flow));
        flow += f;
        a -= f;
        edge[i].flow += f;
        edge[i ^ 1].flow -= f;
        if(!a) return flow;
    }
    return flow;
}
int maxflow(int s,int t){
    this->s = s; this->t = t;
    int flow = 0;
    while(bfs()){
        for(int i = 0 ; i <= n ; i ++) cur[i] = head[i];
        flow += dfs(s,INF);
    }
    return flow;
}
void solve(){
    for(int i = 0 ; i <= n ; i ++){
        for(int j = head[i]; ~j; j = edge[j].next){
            if(edge[j].flow > 0) F[edge[j].id] += edge[j].flow;
        }
    }
}
void del(int S,int T,int SS,int TT){
    for(int i = head[T]; ~i; i = edge[i].next){
        if(edge[i].cap == INF){
            edge[i].cap = edge[i].flow = 0;
            edge[i ^ 1].cap = edge[i ^ 1].flow = 0;
        }
    }
    for(int i = head[SS]; ~i; i = edge[i].next){
        edge[i].cap = edge[i].flow = 0;
        edge[i ^ 1].cap = edge[i ^ 1].flow = 0;
```

```
        }
        for(int i = head[TT]; ~i; i = edge[i].next){
            edge[i].cap = edge[i].flow = 0;
            edge[i ^ 1].cap = edge[i ^ 1].flow = 0;
        }
    }
}g;
int main(){
    scanf("%d%d%d%d",&N,&M,&S,&T);
    int SS = N + 1,TT = N + 2;
    g.init(N + 2);
    for(int i = 1; i <= M ; i ++){
        e[i].u = read(),e[i].v = read();e[i].l = read();e[i].r = read();
        g.add(e[i].u,e[i].v,e[i].r - e[i].l,i);
        a[e[i].u] += e[i].l; a[e[i].v] -= e[i].l;
    }
    g.add(T,S,INF,0);
    int sum = 0;
    for(int i = 1; i <= N; i ++){
        if(a[i] > 0){
            g.add(i,TT,a[i],M + 1);
            sum += a[i];
        }
        else if(a[i] < 0) g.add(SS,i,-a[i],M + 1);
    }
    if(sum != g.maxflow(SS,TT)){
        puts("please go home to sleep"); //不可行情况
        return 0;
    }
    g.solve();
    /*g.del(S,T,SS,TT);   //删除超级源点汇点以及加的 T 到 S 的边
    int ans = F[0] - g.maxflow(T,S);*/
    int ans = INF - g.maxflow(T,S);  //可将本行替换为上面的注释内容
    Pri(ans);
    return 0;
}
```

2.16 树的重心

```
//树的重心：删除该点之后，使得形成的多棵树中节点数最大值最小。
//模板：求树的所有重心从小到达以此输出
const int maxn = 5e4 + 10;
int N,M,K;
struct Edge{
    int to,next;
}edge[maxn * 2];
int head[maxn],tot;
void init(){
    for(int i = 0 ; i <= N ; i ++) head[i] = -1;
    tot = 0;
}
void add(int u,int v){
    edge[tot].to = v;
    edge[tot].next = head[u];
```

```
    head[u] = tot++;
}
int size[maxn],ans,weight[maxn];
void dfs(int t,int la){
    size[t] = 1;
    int heavy = 0;
    for(int i = head[t]; ~i; i = edge[i].next){
        int y = edge[i].to;
        if(y == la) continue;
        dfs(y,t);
        size[t] += size[y];
        heavy = max(heavy,size[y]);
    }
    heavy = max(heavy,N - size[t]);
    weight[t] = heavy;
    ans = min(ans,heavy);
}
int main(){
    Sca(N); init(); ans = INF;
    for(int i = 1; i <= N - 1; i ++){
        int u,v; Sca2(u,v);
        add(u,v); add(v,u);
    }
    int root = 1;
    dfs(root,-1);
    for(int i = 1; i <= N ; i ++){
        if(ans == weight[i]){
            printf("%d ",i);
        }
    }
    return 0;
}
```

2.17 树的直径

2.17.1 双 dfs 法

```
//树的直径：树上最长长度的链被称作树的直径，求树直径长度的方法有双 dfs 法和树 dp 法
//一个有用的结论：从树上任意一个点出发到达的最远的点一定是这棵树的直径的一个端点（距离较长的
↪ 那一个）。
//双 dfs: 任选一个点寻找出距离他最远的点 a，然后以 a 为起点寻找出距离他最远的点 b,ab 距离为
↪ 树的直径
const int maxn = 1e5 + 10;
int N,M,K;
struct Edge{
    int to,next,dis;
}edge[maxn * 2];
int head[maxn],tot;
int dis[maxn];
void init(){
    for(int i = 0 ; i <= N ; i ++) head[i] = -1;
    tot = 0;
}
void add(int u,int v,int w){
```

```
    edge[tot].to = v;
    edge[tot].next = head[u];
    edge[tot].dis = w;
    head[u] = tot++;
}
void dfs(int t,int la){
    for(int i = head[t]; ~i; i = edge[i].next){
        int v = edge[i].to;
        if(v == la) continue;
        dis[v] = dis[t] + edge[i].dis;
        dfs(v,t);
    }
}
int main(){
    int T;
    scanf("%d",&T);
    int CASE = 1;
    while(T--){
        Sca(N); init();
        for(int i = 1; i <= N - 1; i ++){
            int u,v,w; Sca3(u,v,w);
            add(u,v,w); add(v,u,w);
        }
        int root = 1;
        dis[root] = 0; dfs(root,-1);
        for(int i = 1; i <= N ; i ++) if(dis[root] < dis[i]) root = i;
        dis[root] = 0; dfs(root,-1);
        for(int i = 1; i <= N ; i ++) if(dis[root] < dis[i]) root = i;
        printf("Case %d: ",CASE++);
        Pri(dis[root]);
    }
    return 0;
}
```

2.17.2 树形 dp 法

```
//树形 dp 法：记录当前节点最远端的节点和次远端的节点，两遍 dfs 更新即可
//注：如果是图上直径，可采用两边 BFS 的方法
const int maxn = 1e5 + 10;
int N,M,K;
struct Edge{
    int to,next,dis;
}edge[maxn * 2];
int head[maxn],tot;
int dp[maxn],dp2[maxn];
void init(){
    for(int i = 0 ; i <= N ; i ++){
        head[i] = -1;
        dp[i] = dp2[i] = 0;
    }
    tot = 0;
}
void add(int u,int v,int w){
    edge[tot].to = v;
```

```
    edge[tot].next = head[u];
    edge[tot].dis = w;
    head[u] = tot++;
}
void change(int t,int w){
    if(dp[t] < w){
        dp2[t] = dp[t];
        dp[t] = w;
    }else if(dp2[t] < w){
        dp2[t] = w;
    }
}
void dfs(int t,int la){
    for(int i = head[t]; ~i; i = edge[i].next){
        int v = edge[i].to,w = edge[i].dis;
        if(v == la) continue;
        dfs(v,t);
        change(t,dp[v] + w);
    }
}
void dfs2(int t,int la){
    for(int i = head[t];~i ; i = edge[i].next){
        int v = edge[i].to,w = edge[i].dis;
        if(v == la) continue;
        if(dp[t] == dp[v] + w) change(v,dp2[t] + w);
        else change(v,dp[t] + w);
        dfs2(v,t);
    }
}
int main(){
    int T;
    scanf("%d",&T);
    int CASE = 1;
    while(T--){
        Sca(N); init();
        for(int i = 1; i <= N - 1; i ++){
            int u,v,w; Sca3(u,v,w); u++; v++;
            add(u,v,w); add(v,u,w);
        }
        int root = 1;
        dfs(root,-1);
        dfs2(root,-1);
        int ans = 0;
        for(int i = 1; i <= N ; i++) ans = max(ans,dp[i] + dp2[i]);
        printf("Case %d: %d\n",CASE++,ans);
    }
    return 0;
}
```

2.18 树上 k 半径覆盖

```
/* 树上 k 半径覆盖问题
一个点覆盖距离他最长为 k 的所有点，求一棵树上最少几个点可以全覆盖
贪心做法，从叶子结点向上遍历，每次遇到不得不放点的位置就在他往上 k 个位置放点。
```

```
*/
//例题 洛谷 P2279 一个点覆盖最长距离为 2 的所有点，求做小覆盖点数
const int maxn =2010;
int N,M,K;
int fa[maxn],deep[maxn],a[maxn],o[maxn]; //a 数组用来排序，o 数组记录和最近的已经放的点还有
↪ 多少距离
bool cmp(int a,int b){
    return deep[a] > deep[b];
}
int main(){
    Sca(N);  o[1] = INF;
    fa[1] = N + 1;
    fa[N + 1] = N + 2;
    o[N + 1] = o[N + 2] = INF;
    for(int i = 1; i <= N; i ++) a[i] = i;
    for(int i = 2; i <= N ; i ++){
        Sca(fa[i]);
        deep[i] = deep[fa[i]] + 1;
        o[i] = INF;
    }
    sort(a + 1,a + 1 + N,cmp);
    int ans = 0;
    for(int i = 1; i <= N ; i ++){
        int u = a[i];
        int v = fa[u],w = fa[fa[u]];
        o[u] = min(o[u],min(o[v] + 1,o[w] + 2));
        if(o[u] > 2){
            o[w] = 0;
            o[fa[w]] = min(o[fa[u]],1);
            o[fa[fa[w]]] = min(o[fa[fa[w]]],2);
            ans++;
        }
    }
    Pri(ans);
    return 0;
}
```

2.19 dfs 序

```
//括号序列
/* 首先 dfs 整棵树一遍，进入一个节点的时候加上一个左括号，然后是节点编号，
当这个节点的所有子树遍历完后再添上一个右括号，这就是括号序列
假如一棵树的括号序列是 (1(2(3))(4(5)(6)(7(8))))
我们要求 3 到 8 的距离，截取两点间的括号序列为 3))(4(5)(6)(7(8
把编号和匹配的括号删掉 ))(((
剩下了 5 个左右括号，而这就是 3 到 8 的距离。这就是括号序列的性质。*/
/* 模板
操作 1. 将一个结点黑白变换，操作 2. 询问所有黑点之间的最长距离
解法：相当于求出括号序列，线段树维护两个黑点之间的最大距离 sum*/
const int maxn = 1e5 + 10;
const int INF = 0x3f3f3f3f;
int N,M,K;
struct Edge{
    int to,next;
```

```
}edge[maxn * 2];
int head[maxn],tot;
void init(){
    for(int i = 0 ; i <= N ; i ++) head[i] = -1;
    tot = 0;
}
void add(int u,int v){
    edge[tot].to = v;
    edge[tot].next = head[u];
    head[u] = tot++;
}
//括号序列
int id[maxn << 2],cnt,pos[maxn];
void dfs(int u,int la){
    id[++cnt] = -1; //左括号
    id[++cnt] = u; pos[u] = cnt;
    for(int i = head[u]; ~i ; i = edge[i].next){
        int v = edge[i].to;
        if(v == la) continue;
        dfs(v,u);
    }
    id[++cnt] = -2; // 右括号
}
//设左子树左右括号数量为 l1,r1, 右子树为 l2,r2
//sum = l1 + abs(r1 - l2) + r2 = max(l1 + r1 + (r2 - l2),l1 - r1 + (r2 + l2))
//所以维护后缀的 max(l + r),max(l - r), 前缀的 max(r - l),max(r + l)
struct Tree{
    int l,r;
    int a,b,sum;
    int l1,l2; //max(l + r),max(r - l)
    int r1,r2; //max(l - r),max(l + r)
}tree[maxn << 4];
//l1 表示当前区间前缀的一段中左括号和右括号的和最大是多少, 其他同理
bool col[maxn];
//因为是黑点的最远距离, 除了黑点之外的坐标都不维护 pre,erp,sum, 直接当作-INF
void update(int t){
    tree[t].a = tree[t].b = 0;
    tree[t].sum = tree[t].l1 = tree[t].l2 = tree[t].r1 = tree[t].r2 = -INF;
    int v = tree[t].l;
    if(id[v] == -1) tree[t].b = 1;
    else if(id[v] == -2) tree[t].a = 1;
    else if(!col[id[v]]){
        tree[t].l1 = tree[t].l2 = tree[t].r1 = tree[t].r2 = 0;
    }
}
void Pushup(int t){
    tree[t].b = tree[rc].b;
    tree[t].a = tree[lc].a;
    if(tree[lc].b > tree[rc].a) tree[t].b += tree[lc].b - tree[rc].a;
    else tree[t].a += tree[rc].a - tree[lc].b;
    tree[t].l1 = max(tree[lc].l1,max(tree[rc].l1 + tree[lc].a - tree[lc].b,tree[rc].l2 +
    ↪  tree[lc].a + tree[lc].b));
    tree[t].l2 = max(tree[lc].l2,tree[rc].l2 + tree[lc].b - tree[lc].a);
```

```
    tree[t].r1 = max(tree[rc].r1,tree[lc].r1 + tree[rc].a - tree[rc].b);
    tree[t].r2 = max(tree[rc].r2,max(tree[lc].r2 + tree[rc].b - tree[rc].a,tree[lc].r1 +
    ↪  tree[rc].b + tree[rc].a));
    tree[t].sum = max(tree[lc].r1 + tree[rc].l1,tree[lc].r2 + tree[rc].l2);
    tree[t].sum = max(max(tree[lc].sum,tree[rc].sum),tree[t].sum);
}
void Build(int t,int l,int r){
    tree[t].l = l; tree[t].r = r;
    if(l == r){
        update(t);
        return;
    }
    int m = l + r >> 1;
    Build(t << 1,l,m); Build(t << 1 | 1,m + 1,r);
    Pushup(t);
}
void update(int t,int p){
    if(tree[t].l == tree[t].r){
        update(t);
        return;
    }
    int m = tree[t].l + tree[t].r >> 1;
    if(p <= m) update(t << 1,p);
    else update(t << 1 | 1,p);
    Pushup(t);
}
int main(){
    Sca(N); init();
    for(int i = 1; i <= N - 1; i ++){
        int u = read(),v = read();
        add(u,v); add(v,u);
    }
    dfs(1,-1);
    Build(1,1,cnt);
    Sca(M); int num = N;
    while(M--){
        char op[3]; scanf("%s",op);
        if(op[0] == 'G'){
            if(num == 1) puts("0");
            else if(!num) puts("-1");
            else Pri(tree[1].sum);
        }else{
            int v = read();
            if(col[v]) num--;
            else num++;
            col[v] ^= 1; update(1,pos[v]);
        }
    }
    return 0;
}
```

2.20 括号序列

```
//括号序列
/* 首先 dfs 整棵树一遍，进入一个节点的时候加上一个左括号，然后是节点编号，
当这个节点的所有子树遍历完后再添上一个右括号，这就是括号序列
假如一棵树的括号序列是 (1(2(3))(4(5)(6)(7(8))))
我们要求 3 到 8 的距离，截取两点间的括号序列为 3))(4(5)(6)(7(8
把编号和匹配的括号删掉 ))(((
剩下了 5 个左右括号，而这就是 3 到 8 的距离。这就是括号序列的性质。*/
/* 模板
操作 1. 将一个结点黑白变换，操作 2. 询问所有黑点之间的最长距离
解法：相当于求出括号序列，线段树维护两个黑点之间的最大距离 sum*/
const int maxn = 1e5 + 10;
const int INF = 0x3f3f3f3f;
int N,M,K;
struct Edge{
    int to,next;
}edge[maxn * 2];
int head[maxn],tot;
void init(){
    for(int i = 0 ; i <= N ; i ++) head[i] = -1;
    tot = 0;
}
void add(int u,int v){
    edge[tot].to = v;
    edge[tot].next = head[u];
    head[u] = tot++;
}
//括号序列
int id[maxn << 2],cnt,pos[maxn];
void dfs(int u,int la){
    id[++cnt] = -1; //左括号
    id[++cnt] = u; pos[u] = cnt;
    for(int i = head[u]; ~i ; i = edge[i].next){
        int v = edge[i].to;
        if(v == la) continue;
        dfs(v,u);
    }
    id[++cnt] = -2; // 右括号
}
//设左子树左右括号数量为 l1,r1, 右子树为 l2,r2
//sum = l1 + abs(r1 - l2) + r2 = max(l1 + r1 + (r2 - l2),l1 - r1 + (r2 + l2))
//所以维护后缀的 max(l + r),max(l - r), 前缀的 max(r - l),max(r + l)
struct Tree{
    int l,r;
    int a,b,sum;
    int l1,l2; //max(l + r),max(r - l)
    int r1,r2; //max(l - r),max(l + r)
}tree[maxn << 4];
//l1 表示当前区间前缀的一段中左括号和右括号的和最大是多少，其他同理
bool col[maxn];
//因为是黑点的最远距离，除了黑点之外的坐标都不维护 pre,erp,sum, 直接当作-INF
void update(int t){
    tree[t].a = tree[t].b = 0;
```

```
    tree[t].sum = tree[t].l1 = tree[t].l2 = tree[t].r1 = tree[t].r2 = -INF;
    int v = tree[t].l;
    if(id[v] == -1) tree[t].b = 1;
    else if(id[v] == -2) tree[t].a = 1;
    else if(!col[id[v]]){
        tree[t].l1 = tree[t].l2 = tree[t].r1 = tree[t].r2 = 0;
    }
}
void Pushup(int t){
    tree[t].b = tree[rc].b;
    tree[t].a = tree[lc].a;
    if(tree[lc].b > tree[rc].a) tree[t].b += tree[lc].b - tree[rc].a;
    else tree[t].a += tree[rc].a - tree[lc].b;
    tree[t].l1 = max(tree[lc].l1,max(tree[rc].l1 + tree[lc].a - tree[lc].b,tree[rc].l2 +
    ↪  tree[lc].a + tree[lc].b));
    tree[t].l2 = max(tree[lc].l2,tree[rc].l2 + tree[lc].b - tree[lc].a);
    tree[t].r1 = max(tree[rc].r1,tree[lc].r1 + tree[rc].a - tree[rc].b);
    tree[t].r2 = max(tree[rc].r2,max(tree[lc].r2 + tree[rc].b - tree[rc].a,tree[lc].r1 +
    ↪  tree[rc].b + tree[rc].a));
    tree[t].sum = max(tree[lc].r1 + tree[rc].l1,tree[lc].r2 + tree[rc].l2);
    tree[t].sum = max(max(tree[lc].sum,tree[rc].sum),tree[t].sum);
}
void Build(int t,int l,int r){
    tree[t].l = l; tree[t].r = r;
    if(l == r){
        update(t);
        return;
    }
    int m = l + r >> 1;
    Build(t << 1,l,m); Build(t << 1 | 1,m + 1,r);
    Pushup(t);
}
void update(int t,int p){
    if(tree[t].l == tree[t].r){
        update(t);
        return;
    }
    int m = tree[t].l + tree[t].r >> 1;
    if(p <= m) update(t << 1,p);
    else update(t << 1 | 1,p);
    Pushup(t);
}
int main(){
    Sca(N); init();
    for(int i = 1; i <= N - 1; i ++){
        int u = read(),v = read();
        add(u,v); add(v,u);
    }
    dfs(1,-1);
    Build(1,1,cnt);
    Sca(M); int num = N;
    while(M--){
        char op[3]; scanf("%s",op);
```

```
        if(op[0] == 'G'){
            if(num == 1) puts("0");
            else if(!num) puts("-1");
            else Pri(tree[1].sum);
        }else{
            int v = read();
            if(col[v]) num--;
            else num++;
            col[v] ^= 1; update(1,pos[v]);
        }
    }
    return 0;
}
```

2.21 LCA

```
//倍增 lca
//注：如果动态建树的话，dfs 函数非必要，建树的同时维护稀疏表即可
const int SP = 20; //(1 << SP) 为这棵树最多的深度
int pa[maxn][SP],dep[maxn];
void init(){
    Mem(head, -1);
    tot = 0;
}
void dfs(int u, int fa){
    pa[u][0] = fa; dep[u] = dep[fa] + 1;
    for(int i = 1; i < SP; i ++) pa[u][i] = pa[pa[u][i - 1]][i - 1];
    for (int i = head[u]; ~i; i = edge[i].next){
        int v = edge[i].to;
        if (v == fa) continue;
        dfs(v, u);
    }
}
int lca(int u, int v){
    if (dep[u] < dep[v]) swap(u, v);
    int t = dep[u] - dep[v];
    for(int i = 0 ; i < SP; i ++) if (t & (1 << i)) u = pa[u][i];
    for(int i = SP - 1; i >= 0 ; i --){
        int uu = pa[u][i], vv = pa[v][i];
        if (uu != vv){
            u = uu;
            v = vv;
        }
    }
    return u == v ? u : pa[u][0];
}
```

2.22 点分治

```
/*    树上点分治
    处理树链端点之间的关系，例如树上是否存在距离 k 的点对，或者树链点权相加小于等于 K 的个数
 ↪  之类
    核心思想是
    (1) 找到树的重心
```

```
    (2) 求这个重心的子树之间互相是否形成满足题意的条件（子树与子树之间，操作的是经过根节点的那条链）
    (3) 遍历所有子树，将子树当成独立的树之后每棵树重复 (1)
*/
//例题：洛谷 P2634 寻找一棵树上有多少点之间的距离为 3 的的倍数（T），以及不是 3 的倍数 (F) 求 T / (F + T) 的最简分数
int root,max_part,SUM;
int size[maxn];
void dfs_root(int u,int la){
    size[u] = 1;int heavy = 0;
    for(int i = head[u]; ~i ; i = edge[i].next){
        int v = edge[i].to;
        if(v == la || vis[v]) continue;
        dfs_root(v,u);
        heavy = max(heavy,size[v]);
        size[u] += size[v];
    }
    if(max_part > max(heavy,SUM - heavy)){
        max_part = max(heavy,SUM - heavy);
        root = u;
    }
}
int judge[5],dis[5];
void dfs_dis(int t,int la,int d){
    dis[d]++;
    for(int i = head[t]; ~i; i = edge[i].next){
        int v = edge[i].to;
        if(vis[v] || v == la) continue;
        dfs_dis(v,t,(d + edge[i].dis) % 3);
    }
}
void work(int t){
    judge[0]++;
    for(int i = head[t]; ~i; i = edge[i].next){
        int v = edge[i].to;
        if(vis[v]) continue;
        dfs_dis(v,t,edge[i].dis);
        for(int k = 0 ; k < 3; k ++){
            for(int j = 0 ; j < 3; j ++){
                if((k + j) % 3 == 0){
                    T += dis[k] * judge[j];
                }else{
                    F += dis[k] * judge[j];
                }
            }
        }
        for(int k = 0 ; k < 3; k ++){
            judge[k] += dis[k];
            dis[k] = 0;
        }
    }
    for(int i = 0 ; i < 3; i ++) judge[i] = 0;
}
```

```
void divide(int t){
    max_part = INF,root = t;
    dfs_root(t,-1);
    vis[root] = 1;
    work(root);
    for(int i = head[root]; ~i ; i = edge[i].next){
        int v = edge[i].to;
        if(vis[v]) continue;
        SUM = size[v];
        divide(v);
    }
}
int gcd(int a,int b){
    return !b?a:gcd(b,a % b);
}
int main(){
    Sca(N); init();
    for(int i = 1; i <= N - 1; i ++){
        int u,v,w; Sca3(u,v,w);
        w %= 3;
        add(u,v,w); add(v,u,w);
    }
    SUM = N;divide(1);
    T <<= 1; F <<= 1;
    T += N;//cout << T << "  " << F + T << endl;
    LL GCD = gcd(T,T + F);
    printf("%lld/%lld",T / GCD,(T + F) / GCD);
    return 0;
}
```

2.23 动态点分治

```
//动态点分治（点分树）
/* 点分树：当我们可以形如点分治一样的统计答案，即每次确定一个重心，然后计算他们子树之间的贡献
 ↪ 和得出答案的时候
我们可以将每个区域的重心作为其所有子树的重心的父亲，构成一颗新的树，显然这棵树的深度不会超过
 ↪ logn
每次对于单点（边）更新的时候，只要对其所有的父亲更新，就只需要更新 log 个点，这样的数据结构就
 ↪ 是点分树
*/
//模板:BZOJ1095
/*
对于本题来说，最终的答案是在每个点作为链上一个点的时候，找每个点出发的最长链和次长链的和的最大
 ↪ 值
所以用一个堆 A 维护每个点在点分树中子树下所有的点到这个点父亲的距离
再用一个堆 B 维护每个点所有儿子点的堆 A 的最大值，即每条链的最长的长度
最后用一个堆 C 维护每个点的最长值 + 次长值的大小
tips:
1. 树上两两之间的点的距离可以 rmq + ST 表预处理之后 O(1) 查询，注意转化成序列不是 dfs 序，具
 ↪ 体看代码
2. 做一个供删除的优先队列，可以用两个优先队列 A,B，一个正常用，删除操作就是把元素加入 B，当
 ↪ AB 顶部相同的时候一起弹出
*/
const int maxn = 2e5 + 10;
```

```
int N,M,K;
//带删除的优先队列
struct heap{
    priority_queue<int>A,B;
    void push(int x){A.push(x);}
    void del(int x){B.push(x);}
    void init(){
        while(!B.empty() && A.top() == B.top()){
            A.pop(); B.pop();
        }
    }
    int top(){
        init();
        if(A.empty()) return 0;
        return A.top();
    }
    int size(){
        return A.size() - B.size();
    }
    int Maxdis(){
        if(size() < 2) return 0;
        int x = top(); A.pop();
        int y = top(); A.push(x);
        return x + y;
    }
};
struct Edge{
    int to,next;
}edge[maxn << 1];
int head[maxn],tot;
void init(){
    for(int i = 0 ; i <= N ; i ++) head[i] = -1;
    tot = 0;
}
void add(int u,int v){
    edge[tot].to = v;
    edge[tot].next = head[u];
    head[u] = tot++;
}
//st 表预处理树上距离
int dep[maxn];
int id[maxn],pos[maxn],cnt;
void dfsinit(int u,int la){
    id[++cnt] = dep[u];
    pos[u] = cnt;
    for(int i = head[u]; ~i ; i = edge[i].next){
        int v = edge[i].to;
        if(v == la) continue;
        dep[v] = dep[u] + 1;
        dfsinit(v,u);
        id[++cnt] = dep[u]; //注意这个点是 u 而不是 v, 这不是一个 dfs 序
    }
}
```

```
const int SP = 20;
int MIN[maxn][SP],mm[maxn];
void initRMQ(int n,int b[]){
    for(int i = 1; i <= n ; i ++){
        for(int j = 0 ; j < SP; j ++){
            MIN[i][j] = INF;
        }
    }
    mm[0] = -1;
    for(int i = 1; i <= n; i ++){
        mm[i] = ((i & (i - 1)) == 0)?mm[i - 1] + 1:mm[i - 1];
        MIN[i][0] = b[i];
    }
    for(int j = 1; j <= mm[n]; j ++){
        for(int i = 1; i + (1 << j) - 1 <= n; i ++){
            MIN[i][j] = min(MIN[i][j - 1],MIN[i + (1 << (j - 1))][j - 1]);
        }
    }
}
int rmq(int x,int y){
    if(x > y) swap(x,y);
    int k = mm[y - x + 1];
    return min(MIN[x][k],MIN[y - (1 << k) + 1][k]);
}
int getdis(int x,int y){ //查询 x 到 y 的树上距离
    return dep[x] + dep[y] - 2 * rmq(pos[x],pos[y]);
}
struct dtnode{
    int fa;
    heap Q,P;
    int Maxdis(){return Q.top();}
    int Maxline(){return P.Maxdis();}
}dt[maxn];
heap fans;
int root,max_part,SUM;
int size[maxn],vis[maxn];
int dis[maxn];
void dfs_root(int u,int la){
    size[u] = 1; int heavy = 0;
    for(int i = head[u]; ~i ; i = edge[i].next){
        int v = edge[i].to;
        if(v == la || vis[v]) continue;
        dfs_root(v,u);
        heavy = max(heavy,size[v]);
        size[u] += size[v];
    }
    if(max_part > max(heavy,SUM - heavy)){
        max_part = max(heavy,SUM - heavy);
        root = u;
    }
}
void dfs(int u,int la){
    dis[u] = getdis(u,dt[root].fa);
```

```
    dt[root].Q.push(dis[u]);
    for(int i = head[u]; ~i ; i = edge[i].next){
        int v = edge[i].to;
        if(v == la || vis[v]) continue;
        dfs(v,u);
    }
}
int divide(int t,int la){
    max_part = INF;
    root = t;
    dfs_root(t,-1);
    int now = root;
    dt[root].fa = la; dt[root].P.push(0);
    if(~la) dfs(root,-1);
    vis[root] = 1;
    for(int i = head[now]; ~i ; i = edge[i].next){
        int v = edge[i].to;
        if(vis[v]) continue;
        SUM = size[v];
        v = divide(v,now);
        dt[now].P.push(dt[v].Maxdis());
    }
    if(dt[now].P.size() >= 2)fans.push(dt[now].Maxline());
    return now;
}
int use[maxn];
int main(){
    Sca(N); init();
    for(int i = 1; i < N ; i ++){
        int u = read(),v = read();
        add(u,v); add(v,u);
    }
    dfsinit(1,-1); initRMQ(cnt,id);
    SUM = N; divide(1,-1);
    int num = N;
    Sca(M);
    while(M--){
        char op[3]; scanf("%s",op);
        if(op[0] == 'G'){
            if(num == 1) puts("0");
            else if(!num) puts("-1");
            else{
                Pri(fans.top());
            }
        }else{
            int v = read(); int tmp = v;
            if(use[v]) num++;
            else num--;
            if(dt[v].P.size() >= 2) fans.del(dt[v].Maxline());
            if(use[v]) dt[v].P.push(0);
            else dt[v].P.del(0);
            if(dt[v].P.size() >= 2) fans.push(dt[v].Maxline());
            while(~dt[tmp].fa){
```

```
                int u = dt[tmp].fa;
                if(dt[u].P.size() >= 2) fans.del(dt[u].Maxline());
                if(dt[tmp].Q.size()) dt[u].P.del(dt[tmp].Maxdis());
                int d = getdis(v,u);
                if(use[v]) dt[tmp].Q.push(d);
                else dt[tmp].Q.del(d);
                if(dt[tmp].Q.size()) dt[u].P.push(dt[tmp].Maxdis());
                if(dt[u].P.size() >= 2) fans.push(dt[u].Maxline());
                tmp = u;
            }
            use[v] ^= 1;
        }
    }
    return 0;
}
```

2.24 树上差分

```
/*    树上差分
    用来解决一系列将树链上的点权或者边权加上一个数，最后询问每个点的点权（边的边权）的操作
    如果是点权，对于 u 到 v 这条树链上的点权全部加上 w,
    就进行 val[u] += w; val[v] += w; val[lca(u,v)] -= w; val[fa[lca(u,v)]] -= w;
    最后求子树和的操作，每个数的子树和就是我们要的点权
    如果是边权，就将边都转为两端深度较深的那个点，u 到 v 的边权加上 w，就是
    val[u] += w; val[v] += w; val[lca(u,v)] -= 2 * w;
    最后同样求一个子树和的操作即可。
*/
//模板 点权的差分

int val[maxn];
void dfs2(int u,int la){
    for(int i = head[u]; ~i; i = edge[i].next){
        int v = edge[i].to;
        if(v == la) continue;
        dfs2(v,u);
        val[u] += val[v];
    }
}
int main(){
    Sca(N); init();
    for(int i = 1; i <= N ; i ++) Sca(a[i]);
    for(int i = 1; i <= N - 1; i ++){
        int u,v; Sca2(u,v);
        add(u,v); add(v,u);
    }
    dfs(1,0);   //是 lca 里的 dfs
    for(int i = 1; i < N ; i ++){
        int u = a[i],v = a[i + 1];
        val[u]++;val[v]++;
        int l = lca(u,v);
        val[l]--; val[fa[l][0]]--;
    }
    dfs2(1,0);
    for(int i = 1; i <= N ; i ++) Pri(val[i]);
```

```
    return 0;
}
```

2.25 树链剖分

```
//树链剖分，处理将整棵树化成一个序列方便区间操作，用线段树或者其他数据结构统一处理
//注意：如果树剖写 T 了，要关注序列有没有按照重链来排，有可能是重链那部分写错了
const int maxn = 3e4 + 10;
const int INF = 0x3f3f3f3f;
const int mod = 1e9 + 7;
int N,M,K;
struct Edge{
    int to,next;
}edge[maxn * 2];
int head[maxn],tot;
void init(){
    for(int i = 1;i <= N ; i ++) head[i] = -1;
    tot = 0;
}
void add(int u,int v){
    edge[tot].to = v;
    edge[tot].next = head[u];
    head[u] = tot++;
}

//求结点信息
int nw[maxn];
int dep[maxn],top[maxn],fa[maxn],key[maxn],pos[maxn],size[maxn],son[maxn];
int To_num[maxn];
void dfs1(int t,int la){
    size[t] = 1; son[t] = t;
    int heavy = 0;
    for(int i = head[t]; ~i ; i = edge[i].next){
        int v = edge[i].to;
        if(v == la) continue;
        dep[v] = dep[t] + 1;
        fa[v] = t;
        dfs1(v,t);
        if(size[v] > heavy){
            heavy = size[v];
            son[t] = v;
        }
        size[t] += size[v];
    }
}
int cnt;
void dfs2(int t,int la){
    top[t] = la;
    pos[t] = ++cnt;
    To_num[cnt] = t;
    nw[cnt] = key[t];
    if(son[t] == t) return;
    dfs2(son[t],la);
    for(int i = head[t]; ~i ; i = edge[i].next){
```

```
        int v = edge[i].to;
        if((fa[t] == v) || (v == son[t])) continue;
        dfs2(v,v);
    }
}
//线段树
struct Tree{
    int l,r;
    int sum,MAX;
}tree[maxn << 2];
void Pushup(int t){
    tree[t].sum = tree[t << 1].sum + tree[t << 1 | 1].sum;
    tree[t].MAX = max(tree[t << 1].MAX,tree[t << 1 | 1].MAX);
}
void Build(int t,int l,int r){
    tree[t].l = l; tree[t].r = r;
    if(l == r){
        tree[t].sum = tree[t].MAX = nw[l];
        return;
    }
    int m = (l + r) >> 1;
    Build(t << 1,l,m); Build(t << 1 | 1,m + 1,r);
    Pushup(t);
}
void update(int t,int p,int x){
    if(tree[t].l == tree[t].r){
        tree[t].MAX = tree[t].sum = x;
        return;
    }
    int m = (tree[t].l + tree[t].r) >> 1;
    if(p <= m) update(t << 1,p,x);
    else update(t << 1 | 1,p,x);
    Pushup(t);
}
int query(int t,int l,int r,int p){
    if(l <= tree[t].l && tree[t].r <= r){
        if(p) return tree[t].sum;
        else return tree[t].MAX;
    }
    int m = (tree[t].l + tree[t].r ) >> 1;
    if(r <= m) return query(t << 1,l,r,p);
    else if(l > m) return query(t << 1 | 1,l,r,p);
    else{
        if(p) return query(t << 1,l,m,p) + query(t << 1 | 1,m + 1,r,p);
        else return max(query(t << 1,l,m,p),query(t << 1 | 1,m + 1,r,p));
    }
}

//树链剖分求链上信息
int query(int u,int v,int p){
    int ans = 0;
    if(!p) ans = -INF;
    while(top[u] != top[v]){
```

```
        if(dep[top[u]] < dep[top[v]]) swap(u,v);
        if(p) ans += query(1,pos[top[u]],pos[u],1);
        else ans = max(ans,query(1,pos[top[u]],pos[u],0));
        u = fa[top[u]];
    }
    if(dep[u] > dep[v]) swap(u,v);
    if(p) ans += query(1,pos[u],pos[v],p);
    else ans = max(ans,query(1,pos[u],pos[v],p));
    return ans;
}
int main(){
    Sca(N); init();
    for(int i = 1; i <= N - 1; i ++){
        int u,v; Sca2(u,v);
        add(u,v); add(v,u);
    }
    for(int i = 1; i <= N ; i ++) Sca(key[i]);
    int root = 1;
    dfs1(root,-1);
    cnt = 0;
    dfs2(root,root);
    Build(1,1,N);
    Sca(M);
    while(M--){
        char op[10]; int u,v;
        scanf("%s%d%d",op,&u,&v);
        if(op[0] == 'C'){
            update(1,pos[u],v);     //把结点 u 权值改为 t
        }else if(op[1] == 'M'){
            Pri(query(u,v,0));      //查询链上结点最大值
        }else if(op[1] == 'S'){
            Pri(query(u,v,1));       //查询树链上的权值和
        }
    }
    return 0;
}
```

2.26 图论小贴士

!!!!!!!!! 1.SPFA 不得已不要用，虽然在有负权边（不只是负环）的时候无法使用 Dijkstra, 但是如果是一个 DAG 图，可以考虑拓扑排序求最短路

2. 如果一道长得很像最短路题，他的最优状态数组形如 dp[i][j] 表示 i 个点 j 个状态，可以考虑把他拆为 i * j 个点然后重新建图跑
3. 先看眼是有向图还是无向图，无向图数组开两倍。
4. 如果题目中没有声明无自环和重边，需要注意
5. 有些遍历的题要考虑环，否则可能死循环，可以使用缩点
6. 如果题目中边权小于等于零，要考虑负环、零环的情况

3 数据结构

3.1 链表

```
//手写链表，可用于优化某些需要反复删除的数组。
```

```
int pre[maxn],nxt[maxn];
void del(int x){          //删除 x 结点
    nxt[pre[x]] = nxt[x];
    pre[nxt[x]] = pre[x];
}
void init(){              //初始化
    nxt[N] = 0;
    for(int i = 1; i <= N ; i ++){
        nxt[i - 1] = i; pre[i] = i - 1;
    }
}
```

3.2 并查集

3.2.1 可撤销并查集

```
//可撤销并查集，可以将并查集的操作倒回到 n 步之前
//用一个栈存储进行的操作，倒回的时候依次退栈即可。
//不可以使用路径压缩，只能用按秩合并（size 小的连到 size 大的上去）优化
int Stack[maxn],size[maxn];
int fa[maxn],top;
int now;
void init(){
    for(int i = 1; i <= N ; i ++){
        fa[i] = -1;size[i] = 0;
    }
    top = 0;
}
int find(int x){
    while(fa[x] != -1) x = fa[x];
    return x;
}
bool Union(int x,int y){
    x = find(x); y = find(y);
    if(x == y) return false;
    if(size[x] > size[y]) swap(x,y);
    Stack[top++] = x;
    fa[x] = y;
    size[y] += size[x] + 1;
    now--;
    return true;
}
void rewind(int t){
    while(top > t){
        int x = Stack[--top];
        size[fa[x]] -= size[x] + 1;
        fa[x] = -1;
        now++;
    }
}
```

3.2.2 可持久化并查集

```
/* 用可持久化数组维护并查集的 fa 数组，模板为 BZOJ3674
强制在线，每个值 ^ lastans
```

```
n 个集合 m 个操作
操作：
1 a b 合并 a,b 所在集合
2 k 回到第 k 次操作之后的状态（查询算作操作）
3 a b 询问 a,b 是否属于同一集合，是则输出 1 否则输出 0
*/
const int maxn = 2e5 + 10;
int N,M,K;
int T[maxn],tot;
struct Tree{
    int lt,rt;
    int fa,size;
}tree[maxn * 52];
int fa[maxn],Size[maxn];
void newnode(int &t){
    t = ++tot;
    tree[t].lt = tree[t].rt = tree[t].fa = 0;
}
void Build(int &t,int l,int r){
    newnode(t);
    if(l == r){
        tree[t].fa = l;
        tree[t].size = 0;
        return;
    }
    int m = l + r >> 1;
    Build(tree[t].lt,l,m); Build(tree[t].rt,m + 1,r);
}
int query(int t,int l,int r,int x){
    if(l == r){
        Size[x] = tree[t].size;
        return tree[t].fa;
    }
    int m = l + r >> 1;
    if(x <= m) return query(tree[t].lt,l,m,x);
    else return query(tree[t].rt,m + 1,r,x);
}
void update(int &t,int pre,int l,int r,int x){
    newnode(t);
    tree[t] = tree[pre];
    if(l == r){
        tree[t].fa = fa[x];
        tree[t].size = Size[x];
        return;
    }
    int m = l + r >> 1;
    if(x <= m) update(tree[t].lt,tree[pre].lt,l,m,x);
    else update(tree[t].rt,tree[pre].rt,m + 1,r,x);
}
int find(int id,int x){
    int f = query(T[id],1,N,x);
    if(f == x) return x;
    return find(id,f);
```

```
}
int main(){
    //freopen("C.in","r",stdin);
    Sca2(N,M);
    Build(T[0],1,N);
    int ans = 0;
    for(int i = 1; i <= M; i ++){
        int op = read();
        if(op == 1){
            int a = read() ^ ans,b = read() ^ ans;
            a = find(i - 1,a); b = find(i - 1,b);
            if(Size[a] > Size[b]) swap(a,b);
            fa[a] = b;
            update(T[i],T[i - 1],1,N,a);
            if(Size[a] == Size[b]){
                Size[b]++; fa[b] = b;
                int x;
                update(x,T[i],1,N,b);
                T[i] = x;
            }
        }else if(op == 2){
            int k = read() ^ ans;
            T[i] = T[k];
        }else if(op == 3){
            T[i] = T[i - 1];
            int a = read() ^ ans,b = read() ^ ans;
            a = find(i,a); b = find(i,b);
            ans = (a == b);
            if(a == b) puts("1");
            else puts("0");
        }

    }
    return 0;
}
```

3.2.3 种类并查集

```
//带权并查集 （又称种类并查集）
//除了 tree 用来记录是否在同一个集合，利用 group 数组来记录当前节点和父亲结点的关系
int find(int x){
    if (x == tree[x]) return x;
    int fx = find(tree[x]);
    group[x] = (group[tree[x]] + group[x] + 2) % 2; //孙子和爷爷的关系 = 孙子和父亲的关系
    ↪ + 父亲和爷爷的关系
    tree[x] = fx;
    return fx;
}
void Union(int a, int b){
    int fa = find(a),fb = find(b);
    if (fa != fb){
        group[fa] = (-group[a] + 1 + group[b] + 2) % 2; // 父节点和 x 的关系，x 和 y 的关
        ↪ 系，y 和父节点的关系
        tree[fa] = fb;
```

```
    }
}
```

3.3 启发式合并

```
/* dsu on tree 启发式合并
nlogn 时间处理一类静态无修的子树信息查询问题
通常暴力是每个结点下的子树 dfs 一遍，时间复杂度 n²
像树剖一样处理出每个子树下的重子树，从上往下 dfs 的时候重子树的信息就可以选择保留
具体步骤：
1. 递归所有 u 的轻点，目的在于计算出他们的答案，不保留贡献
2. 递归 u 的重点，除了计算答案之外还保留贡献
3. 再次递归 u 的轻点，目的在于计算贡献
*/
//模板：每个节点有个颜色，一个子树中颜色节点最多的颜色占领整个子树，问所有子树被占领的颜色编号和（由于节点数有并列，所以是颜色编号和）
const int maxn = 1e5 + 10;
int N,M,K;
LL color[maxn];
struct Edge{
    int to,next;
}edge[maxn * 2];
int head[maxn],tot;
void init(){
    for(int i = 1; i <= N ; i ++) head[i] = -1;
    tot = 0;
}
void add(int u,int v){
    edge[tot].to = v;
    edge[tot].next = head[u];
    head[u] = tot++;
}
int cnt[maxn];
int fa[maxn],son[maxn],size[maxn];
void dfs1(int t,int la){
    fa[t] = la; son[t] = 0;
    size[t] = 1;
    int heavy = 0;
    for(int i = head[t]; ~i ; i = edge[i].next){
        int v = edge[i].to;
        if(v == la) continue;
        dfs1(v,t);
        if(size[v] > heavy){
            son[t] = v;
            heavy = size[v];
        }
        size[t] += size[v];
    }
}
LL Max,ans[maxn],sum;
void dfs2(int t,int isson,int keep){
    if(keep){
        for(int i = head[t]; ~i ; i = edge[i].next){
            int v = edge[i].to;
```

```
            if(v == son[t] || v == fa[t]) continue;
            dfs2(v,0,1);
        }
    }
    if(son[t]) dfs2(son[t],1,keep);
    for(int i = head[t]; ~i ; i = edge[i].next){
        int v = edge[i].to;
        if(v == son[t] || v == fa[t]) continue;
        dfs2(v,0,0);
    }
    cnt[color[t]]++;
    if(cnt[color[t]] > Max) sum = color[t],Max = cnt[color[t]];
    else if(cnt[color[t]] == Max) sum += color[t];
    if(keep) ans[t] = sum;
    if(keep && !isson){
         for(int i = 1; i <= N; i ++) cnt[i] = 0;
         Max = sum = 0;
    }
}
int main(){
    Sca(N); init();
    for(int i = 1; i <= N ; i ++) Scl(color[i]);
    for(int i = 1; i < N ; i ++){
        int u,v; Sca2(u,v);
        add(u,v); add(v,u);
    }
    dfs1(1,0); dfs2(1,1,1);
    for(int i = 1; i <= N ; i ++){
        printf("%lld ",ans[i]);
    }
    return 0;
}
```

3.4 ST 表

```
/*  ST 表
    静态处理区间最大最小值
    不可求和，不可动态
*/
//模板：洛谷 P2880 求区间最大值和最小值的差
const int maxn = 50010;
const int SP = 20;
int MAX[maxn][SP];
int MIN[maxn][SP];
int mm[maxn];
void initRMQ(int n,int b[]){          //预处理 b 数组，长度为 n
    for(int i = 1; i <= n ; i ++){
        for(int j = 0; j < SP; j ++){
            MIN[i][j] = INF;
        }
    }
    mm[0] = -1;
    for(int i = 1; i <= n ; i ++){
        mm[i] = ((i & (i - 1)) == 0)?mm[i - 1] + 1:mm[i - 1];
```

```
        MIN[i][0] = MAX[i][0] = b[i];
    }
    for(int j = 1; j <= mm[n]; j ++){
        for(int i = 1; i + (1 << j) - 1 <= N ; i ++){
            MAX[i][j] = max(MAX[i][j - 1],MAX[i + (1 << (j - 1))][j - 1]);
            MIN[i][j] = min(MIN[i][j - 1],MIN[i + (1 << (j - 1))][j - 1]);
        }
    }
}
int rmq(int x,int y){
    int k = mm[y - x + 1];
    return max(MAX[x][k],MAX[y - (1 << k) + 1][k]) - min(MIN[x][k],MIN[y - (1 << k) +
    ↪  1][k]);
}
//倘若我们对模板稍加修改，便可以得到一个求最值下标的 ST 表
//模板：求区间最大值的下标
const int SP = 20;
int Max[maxn][SP];
int mm[maxn];
void initRMQ(int n,LL b[]){
    mm[0] = -1;
    for(int i = 1; i <= n; i ++){
        mm[i] = ((i & (i - 1)) == 0)?mm[i - 1] + 1:mm[i - 1];
        Max[i][0] = i;
    }
    for(int j = 1; j <= mm[n]; j ++){
        for(int i = 1; i + (1 << j) - 1 <= n ; i ++){
            int x = Max[i][j - 1],y = Max[i + (1 << (j - 1))][j - 1];
            Max[i][j] = b[x] > b[y]?x:y;
        }
    }
}
int rmq(int x,int y,LL b[]){
    int k = mm[y - x + 1];
    return b[Max[x][k]] > b[Max[y - (1 << k) + 1][k]]?Max[x][k]:Max[y - (1 << k) + 1][k];
}
```

3.5 树状数组

```
//树状数组 注意 0 是无效下标
int tree[maxn];
void add(int x,int v){   //单点修改
    for(;x < N; x += x & -x) tree[x] += v;
}
LL sum(int x){           //查询前缀和
    LL ans = 0;
    for(;x > 0;x -= x & -x) ans += tree[x];
    return ans;
}
int kth(int k){           //查询从小到大数第 K 小，数组的下标表示数的大小，第二维表示数的个
↪  数
    int ans = 0;
    LL cnt = 0;
    for(int i = SP - 1; i >= 0 ; i --){
```

```
        ans += (1 << i);
        if(ans >= N || cnt + tree[ans] >= k){
            ans -= 1 << i;
        }else cnt += tree[ans];
    }
    return ans + 1;
}
//模板:POJ2985 初始 N 个为 1 的集合，操作 1. 合并两个集合，操作 2. 查询第 K 大的集合的大小
const int maxn = 2e5 + 10;
const int SP = 20;
int N,M,K;
int fa[maxn],tree[maxn],sz[maxn];
void init(){
    for(int i = 0 ; i <= N ; i ++) fa[i] = i,sz[i] = 1;
}
int find(int x){
    if(fa[x] == x) return x;
    return fa[x] = find(fa[x]);
}
void add(int x,int v){
    for(;x < N; x += x & -x) tree[x] += v;
}
LL sum(int x){
    LL ans = 0;
    for(;x > 0;x -= x & -x) ans += tree[x];
    return ans;
}
int kth(int k){
    int ans = 0;
    LL cnt = 0;
    for(int i = SP - 1; i >= 0 ; i --){
        ans += (1 << i);
        if(ans >= N || cnt + tree[ans] >= k){
            ans -= 1 << i;
        }else cnt += tree[ans];
    }
    return ans + 1;
}
int main(){
    Sca2(N,M); init();
    add(1,N); int num = N;
    for(int i = 1; i <= M ; i ++){
        int op = read();
        if(op == 0){
            int u,v; Sca2(u,v);
            u = find(u),v = find(v);
            if(u == v) continue;
            add(sz[u],-1); add(sz[v],-1);
            fa[u] = v;
            sz[v] += sz[u];
            add(sz[v],1); num--;
        }else{
            int k; Sca(k);
```

```
            k = num - k + 1;
            Pri(kth(k));
        }
    }
    return 0;
}
//二维树状数组
//模板 hdoj2642: 单点修改, 区间查询
const int maxn = 1010;
int N,M,K;
LL tree[maxn][maxn];
bool MAP[maxn][maxn];
void add(int x,int y,int v){
    for(int tx = x;tx < maxn; tx += tx & -tx){
        for(int ty = y;ty < maxn; ty += ty & -ty) tree[tx][ty] += v;
    }
}
LL getsum(int x,int y){
    LL ans = 0;
    for(int tx = x;tx > 0; tx -= tx & -tx){
        for(int ty = y; ty > 0; ty -= ty & -ty) ans += tree[tx][ty];
    }
    return ans;
}
LL getsum(int x1,int y1,int x2,int y2){
    return getsum(x2,y2) + getsum(x1 - 1,y1 - 1) - getsum(x2,y1 - 1) - getsum(x1 - 1,y2);
}
int main(){
    while(~Sca(M)){
        Mem(tree,0);
        while(M--){
            char op[2]; scanf("%s",op);
            if(op[0] == 'B'){
                int x = read(),y = read();
                x++;y++;
                if(!MAP[x][y]){
                    MAP[x][y] = 1;
                    add(x,y,1);
                }
            }else if(op[0] == 'D'){
                int x = read(),y = read();
                x++;y++;
                if(MAP[x][y]){
                    add(x,y,-1);
                    MAP[x][y] = 0;
                }
            }else{
                int x1 = read(),x2 = read(),y1 = read(),y2 = read();
                x1++;x2++;y1++;y2++;  //消除 x = 0,y = 0 的影响,0 是无效点
                if(x1 > x2) swap(x1,x2);
                if(y1 > y2) swap(y1,y2);
                LL t = getsum(x1,y1,x2,y2);
                Prl(t);
```

```
            }
        }
    }
    return 0;
}
```

3.6 二维平面

3.6.1 二维前缀和

```
//二维前缀和
//模板 hdu6541 给出 1e6 个全 1 矩阵，然后查询 1e6 个矩阵是否为全 1 矩阵
//做法：先用前缀和求出每个点是否是 1，然后转化为 01 矩阵之后再求一个前缀和
const int maxn = 1e7 + 10;
int N,M,K;
LL MAP[maxn];
int id(int i,int j){
    if(i <= 0 || j <= 0 || i > N || j > M) return 0;
    return (i - 1) * M + j;
}
void add(int i,int j,int x){
    int v = id(i,j);
    if(!v) return;
    MAP[v] += x;
}
int main(){
    while(~Sca2(N,M)){
        int Q = read();
        for(int i = 0 ; i <= N * M; i ++) MAP[i] = 0;
        while(Q--){
            int x1 = read(),y1 = read(),x2 = read(),y2 = read();
            if(x1 > x2) swap(x1,x2);
            if(y1 > y2) swap(y1,y2);
            add(x1,y1,1);
            add(x1,y2 + 1,-1);
            add(x2 + 1,y1,-1);
            add(x2 + 1,y2 + 1,1);
        }
        for(int i = 1; i <= N; i ++){
            for(int j = 1; j <= M ; j ++){
                MAP[id(i,j)] += MAP[id(i - 1,j)] + MAP[id(i,j - 1)] - MAP[id(i - 1,j -
                ↪ 1)];
            }
        }
        for(int i = 1; i <= N ; i ++){
            for(int j = 1; j <= M ; j ++){
                MAP[id(i,j)] = (MAP[id(i,j)] > 0);
            }
        }
        for(int i = 1; i <= N; i ++){
            for(int j = 1; j <= M ; j ++){
                MAP[id(i,j)] += MAP[id(i - 1,j)] + MAP[id(i,j - 1)] - MAP[id(i - 1,j -
                ↪ 1)];
            }
```

```
        }
        Q = read();
        while(Q--){
            int x1 = read(),y1 = read(),x2 = read(),y2 = read();
            if(x1 > x2) swap(x1,x2);
            if(y1 > y2) swap(y1,y2);
            if(1ll * (x2 - x1 + 1) * (y2 - y1 + 1) == MAP[id(x2,y2)] + MAP[id(x1 - 1,y1 -
            ↪ 1)] - MAP[id(x2,y1 - 1)] - MAP[id(x1 - 1,y2)])
                puts("YES");
            else
                puts("NO");
        }
    }
    return 0;
}
```

3.6.2 二维 ST 表

```
//二维 ST 表 n²log(n)² 预处理,O(1) 查询
//注意空间复杂度不要 MLE,SP 和 maxn 能小尽量小
const int maxn = 255;
const int SP = 8;
int N,M,K;
int Max[maxn][maxn][SP][SP],Min[maxn][maxn][SP][SP];
int a[maxn][maxn],fac[20];
inline int max4(int a,int b,int c,int d){
    return max(max(a,b),max(c,d));
}
inline int min4(int a,int b,int c,int d){
    return min(min(a,b),min(c,d));
}
void init(){
    for(int i = 0 ; i < SP; i ++) fac[i] = (1 << i);
    for(int i = 1; i <= N ; i ++){
        for(int j = 1; j <= M ; j ++){
            Min[i][j][0][0] = a[i][j];
            Max[i][j][0][0] = a[i][j];
        }
    }
    int k = (int)(log((double)N) / log(2.0));
    for(int x = 0; x <= k; x++){
        for(int y = 0;y <= k; y++){
            if(x == 0 && y == 0) continue;
            for(int i = 1; i + fac[x] - 1 <= N; i ++){
                for(int j = 1; j + fac[y] - 1 <= M; j ++){
                    if(x == 0){
                        Min[i][j][x][y] = min(Min[i][j][x][y - 1],Min[i][j + (1 << (y -
                        ↪ 1))][x][y - 1]);
                        Max[i][j][x][y] = max(Max[i][j][x][y - 1],Max[i][j + (1 << (y -
                        ↪ 1))][x][y - 1]);
                    }else{
                        Min[i][j][x][y] = min(Min[i][j][x - 1][y],Min[i + (1 << (x -
                        ↪ 1))][j][x - 1][y]);
```

```
                        Max[i][j][x][y] = max(Max[i][j][x - 1][y],Max[i + (1 << (x -
                        ↪ 1))][j][x - 1][y]);
                    }
                }
            }
        }
    }
}
inline int getmax(int x1,int y1,int x2,int y2){
    int k1 = x2 - x1 + 1,k2 = y2 - y1 + 1;
    k1 = (int)(log((double)k1)/log(2.0));
    k2 = (int)(log((double)k2)/log(2.0));
    return max4(Max[x1][y1][k1][k2],Max[x1][y2 - fac[k2] + 1][k1][k2],Max[x2 - fac[k1] +
    ↪ 1][y1][k1][k2],
    Max[x2 - fac[k1] + 1][y2 - fac[k2] + 1][k1][k2]);
}
inline int getmin(int x1,int y1,int x2,int y2){
    int k1 = x2 - x1 + 1,k2 = y2 - y1 + 1;
    k1 = (int)(log((double)k1)/log(2.0));
    k2 = (int)(log((double)k2)/log(2.0));
    return min4(Min[x1][y1][k1][k2],Min[x1][y2 - fac[k2] + 1][k1][k2],Min[x2 - fac[k1] +
    ↪ 1][y1][k1][k2],
    Min[x2 - fac[k1] + 1][y2 - fac[k2] + 1][k1][k2]);
}
int ans[maxn][maxn];
int main(){
    int B;
    Sca3(N,B,K); M = N;
    for(int i = 1; i <= N ; i ++){
        for(int j = 1; j <= N ; j ++){
            Sca(a[i][j]);
        }
    }
    init();
    for(int i = 1; i <= N ; i ++){
        for(int j = 1; j <= M ; j ++){
            int ii = i + B - 1,jj = j + B - 1;
            ii = min(ii,N); jj = min(jj,M);
            ans[i][j] = getmax(i,j,ii,jj) - getmin(i,j,ii,jj);
        }
    }
    while(K--){
        int x = read(),y = read();
        Pri(ans[x][y]);
    }
    return 0;
}
```

3.7 莫队算法

3.7.1 无修莫队

```
//无修莫队 时间复杂度 n√n
//小 z 的袜子 查询区间内任选两个颜色相同的概率，用分数表示
```

```
const int maxn = 50010;
const int maxm = 50010;
int N,M,K,unit;
struct Query{
    int L,R,id;
    int l,r;
}node[maxm];
struct Ans{
    LL up,down;
    void reduce(){
        if(!up || !down){
            up = 0;
            down = 1;
            return;
        }
        LL g = __gcd(up,down);
        up /= g; down /= g;
    }
}ans[maxm];
int a[maxn];
bool cmp(Query a,Query b){
    if(a.l != b.l) return a.l < b.l;
    return a.r < b.r;
}
int num[maxn];
LL up,down,sum;
void add(int t){
    down += sum; sum++;
    up += num[t]; num[t]++;
}
void del(int t){
    sum--; down -= sum;
    num[t]--; up -= num[t];
}
void solve(){
    int L = 1,R = 0;
    sum = up = down = 0;
    for(int i = 1; i <= M ; i ++){
        while(R < node[i].R) add(a[++R]);
        while(R > node[i].R) del(a[R--]);
        while(L < node[i].L) del(a[L++]);
        while(L > node[i].L) add(a[--L]);
        ans[node[i].id].up = up;
        ans[node[i].id].down = down;
    }
}
int main(){
    Sca2(N,M); unit = (int)sqrt(N);
    for(int i = 1; i <= N ; i ++) Sca(a[i]);
    for(int i = 1; i <= M ; i ++){
        Sca2(node[i].L,node[i].R); node[i].id = i;
        node[i].l = node[i].L / unit;
        node[i].r = node[i].R / unit;
```

```
    }
    sort(node + 1,node + 1 + M,cmp);
    solve();
    for(int i = 1; i <= M ; i ++){
        ans[i].reduce();
        printf("%lld/%lld\n",ans[i].up,ans[i].down);
    }
    return 0;
}
```

3.7.2 回滚莫队

```
/* 例题：无修改求区间 L-R 出现次数 * 权值最大的点
问题：普通的莫队无法删除，因为删除了最大值之后无法直接找到次大值
解决方法：按照左端点所在的块排序，相同的块按照右端点排序
对于左右端点相同块的询问直接暴力 √n
对于左端点处于同一块的询问一起处理，因为右端点递增，所以右端点只有删除惭怍
对于左端点，每次将左端点初始化到所在块的最右端，先向右扩充之后记录当前答案 tmp，然后向左扩充到
↪ 当前询问的左端点，找到当前询问的答案之后回溯到最右端，答案也返回 tmp
虽然依然有删除操作，但是因为我们之前记录过 tmp，避免了真正删除完寻找答案的过程
时间复杂度 n√n
*/
//BZOJ 4241
const int maxn = 1e5 + 10;
int N,M,K,Q,unit;
LL a[maxn],Hash[maxn];
int belong[maxn]; //每个点所在块的编号
struct Query{
    int l,r,id;
}query[maxn];
bool cmp(Query a,Query b){
    if(belong[a.l] == belong[b.l]) return a.r < b.r;
    return belong[a.l] < belong[b.l];
}
LL vis[maxn],Max;
LL ans[maxn];
void add(int p){
    vis[a[p]]++;
    if(vis[a[p]] * Hash[a[p]]> Max){
        Max = vis[a[p]] * Hash[a[p]];
    }
}
void del(int p){
    vis[a[p]]--;
}
LL use[maxn];
LL cul(int l,int r){
    LL sum = 0;
    for(int i = l; i <= r; i ++){
        use[a[i]]++;
        sum = max(sum,use[a[i]] * Hash[a[i]]);
    }
    for(int i = l; i <= r; i ++) use[a[i]]--;
    return sum;
```

```
}
void solve(){
    int up = N / unit;
    int j = 1;
    for(int i = 0; i <= up; i ++){
        Max = 0;
        LL L = (i + 1) * unit,R = L - 1,la,normal = L;
        for(;j <= Q && belong[query[j].l] == i; j ++){  //对于所有左端点在一个块里的单独处
        ↪ 理
            if(belong[query[j].r] == i){  //如果处于同一个块 就暴力
                ans[query[j].id] = cul(query[j].l,query[j].r);
                continue;
            }
            while(R < query[j].r) add(++R);   //先扩充右边的
            la = Max;                         //记录下当前答案
            while(L > query[j].l) add(--L);   //再扩充左边的
            ans[query[j].id] = Max;           //真正的答案
            while(L < normal) del(L++);        //回溯，删除之前扩充的左边的
            Max = la;                         //回溯答案
        }
        while(L <= R) del(L++);
    }
}
int main(){
    Sca2(N,Q); unit = (int)sqrt(N);
    for(int i = 1; i <= N ; i ++) Hash[i] = a[i] = read();
    sort(Hash + 1,Hash + 1 + N);
    int cnt = unique(Hash + 1,Hash + 1 + N) - Hash - 1;
    for(int i = 1; i <= N ; i ++) a[i] = lower_bound(Hash + 1,Hash + 1 + cnt,a[i]) -
    ↪ Hash;
    for(int i = 1; i <= N ; i ++) belong[i] = i / unit;
    for(int i = 1; i <= Q; i ++){
        query[i].l = read(); query[i].r = read();
        query[i].id = i;
    }
    sort(query + 1,query + 1 + Q,cmp);
    solve();
    for(int i = 1; i <= Q; i ++) Prl(ans[i]);
    return 0;
}
```

3.7.3 带修莫队

```
//带修莫队 时间复杂度三次根号内 (n ^ 4 * t)
//模板：带修询问区间颜色种类数
const int maxn = 150010;
int N,M,K,cnt,t,unit;
int a[maxn];
struct Update{
    int pre,aft;  //修改前和修改后
    int pos;    //修改的位置
    Update(int pos = 0,int pre = 0,int aft = 0):pos(pos),pre(pre),aft(aft){}
}update[maxn];
struct Query{
```

```
    int l,r,t;
    int L,R;
    int upnum; //修改过的次数
    Query(){}
    Query(int t,int l,int r,int upnum):t(t),l(l),r(r),upnum(upnum){}
}query[maxn];
bool cmp(Query a,Query b){
    if(a.L != b.L) return a.L < b.L;
    if(a.R != b.R) return a.R < b.R;
    return a.t < b.t;
}
int num[1000010],sum,ans[maxn];
void add(int t){
    num[t]++;
    sum += (num[t] == 1);
}
void del(int t){
    num[t]--;
    sum -= (num[t] == 0);
}
void solve(){
    int l = 1, r = 0,t = 0;
    sum = 0;
    for(int i = 1; i <= M ; i ++){
        while(query[i].l < l) add(a[--l]);
        while(query[i].l > l) del(a[l++]);
        while(query[i].r > r) add(a[++r]);
        while(query[i].r < r) del(a[r--]);
        while(query[i].upnum < t){
            int p = update[t].pos;
            if(l <= p && p <= r) del(a[p]);
            a[p] = update[t--].pre;
            if(l <= p && p <= r) add(a[p]);
        }
        while(query[i].upnum > t){
            int p = update[++t].pos;
            if(l <= p && p <= r) del(a[p]);
            a[p] = update[t].aft;
            if(l <= p && p <= r) add(a[p]);
        }
        ans[query[i].t] = sum;
    }
}
int main(){
    Sca2(N,M); cnt = t = 0;
    for(int i = 1; i <= N ; i ++) Sca(a[i]);
    for(int i = 1; i <= M ; i ++){
        char op[2]; scanf("%s",op);
        if(op[0] == 'R'){ //修改
            int p = read(),c = read();//p 点改为 c
            update[++t] = Update(p,a[p],c); a[p] = c;
        }else{
            cnt++; int l = read(),r = read();
```

```
            query[cnt] = Query(cnt,l,r,t);
        }
    }
    unit = ceil(exp((log(N) + log(t))/3));
    for(int i = t; i >= 1; i --) a[update[i].pos] = update[i].pre;  //初始化 a 数组
    for(int i = 1; i <= cnt ; i ++){
        query[i].L = query[i].l/unit;
        query[i].R = query[i].R/unit;
    }
    sort(query + 1,query + 1 + cnt,cmp);
    solve();
    for(int i = 1; i <= cnt ; i ++) Pri(ans[i]);
    return 0;
}
```

3.7.4 树上莫队

```
//树上莫队
/* 查询树链的关系
求出树的欧拉序，将树链关系转化为序列关系
对于不同的子树上的两点而言，序列上前一个的 ed 到后一个的 st 包含的所有只出现一次的结点就是树
↪ 链上的结点。
但是 lca 不包含在内，因此要特判 lca
*/
//查询树链上不同颜色的种数
const int maxn = 4e5 + 10;
const int maxm = 4e5 + 10;
const int SP = 20;
int fa[maxn][SP],dep[maxn];
int N,M,K,unit;
struct Edge{
    int to,next;
}edge[maxn * 2];
int head[maxn],tot;
void init(){
    for(int i = 0; i <= N ; i ++) head[i] = -1;
    tot = 0;
}
void add(int u,int v){
    edge[tot].to = v;
    edge[tot].next = head[u];
    head[u] = tot++;
}
int idx[maxn],cnt;
PII pos[maxn];
int ans[maxn],b[maxn],a[maxn];
void dfs(int u,int la){
    fa[u][0] = la;
    dep[u] = dep[la] + 1;
    for(int i = 1; i < SP; i ++) fa[u][i] = fa[fa[u][i - 1]][i - 1];
    idx[++cnt] = u; a[cnt] = b[u];
    pos[u].fi = cnt;
    for(int i = head[u]; ~i ; i = edge[i].next){
        int v = edge[i].to;
```

```
        if(v == la) continue;
        dfs(v,u);
    }
    idx[++cnt] = u; a[cnt] = b[u];
    pos[u].se = cnt;
}
int lca(int u,int v){
    if(dep[u] < dep[v]) swap(u,v);
    int t = dep[u] - dep[v];
    for(int i = 0 ; i < SP; i ++) if(t & (1 << i)) u = fa[u][i];
    for(int i = SP - 1; i >= 0 ; i --){
        int uu = fa[u][i],vv = fa[v][i];
        if(uu != vv){
            u = uu;
            v = vv;
        }
    }
    return u == v? u : fa[u][0];
}
struct Query{
    int l,r,t;
    int L,R;
    int lc;  //特判 lca
}node[maxm];
bool cmp(Query a,Query b){
    if(a.l != b.l) return a.l < b.l;
    return a.r < b.r;
}
int num[maxm],sum,vis[maxn];
void add(int p){
    num[p]++;
    sum += (num[p] == 1);
}
void del(int p){
    num[p]--;
    sum -= (num[p] == 0);
}
void work(int t){
    if(vis[t]) del(b[t]);
    else add(b[t]);
    vis[t] ^= 1;
}
void solve(){
    int L = 1,R = 0;
    sum = 0;
    for(int i = 1; i <= M ; i ++){
        while(R < node[i].R) work(idx[++R]);
        while(R > node[i].R) work(idx[R--]);
        while(L < node[i].L) work(idx[L++]);
        while(L > node[i].L) work(idx[--L]);
        if(node[i].lc) work(node[i].lc);
        ans[node[i].t] = sum ;
        if(node[i].lc) work(node[i].lc);
```

```
    }
}
int Hash[maxn];
int main(){
    Sca2(N,M); init(); cnt = 0;
    for(int i = 1; i <= N ; i ++) Hash[i] = b[i] = read();
    sort(Hash + 1,Hash + 1 + N);
    int c = unique(Hash + 1,Hash + 1 + N) - Hash;
    for(int i = 1; i <= N ; i ++) b[i] = lower_bound(Hash + 1,Hash + 1 + c,b[i]) - Hash;
    for(int i = 1; i <= N - 1 ; i ++){
        int u,v; Sca2(u,v);
        add(u,v); add(v,u);
    }
    dfs(1,-1); unit = (int)sqrt(cnt);
    for(int i = 1; i <= M ; i ++){
        int u = read(),v = read();
        if(pos[u].fi > pos[v].fi) swap(u,v);
        node[i].lc = lca(u,v);
        node[i].L = pos[u].se; node[i].R = pos[v].fi;
        if(node[i].lc == u){    //如果 u 是 v 的祖先，就不用特判 lca
            node[i].L = pos[u].fi;
            node[i].lc = 0;
        }
        node[i].l = node[i].L / unit; node[i].r = node[i].R / unit;
        node[i].t = i;
    }
    sort(node + 1,node + 1 + M,cmp);
    solve();
    for(int i = 1; i <= M ; i ++) Pri(ans[i]);
    return 0;
}
```

3.8 珂朵莉树

```
/* 珂朵莉树
1. 名称来源：从 CF896C 发明的算法，题干主角是动漫人物珂朵莉
2. 解决一类问题：(1) 含有区间赋值操作 (2) 数据随机 (重要)
3. 具体思路:
(1) 用 set 存储每个区间的 (l,r) 端点和信息
(2)split: 当出现新的在 l,r 之间的端点 pos 的时候将区间分为 (l,pos- 1)(pos,r)
(3)assign: 为了降低节点数量，区间赋值的时候把整个区间的结点都去掉，换一个 Node(l,r,v) 进去
 */
 //模板 CF896C, 区间加，区间赋值，区间 K 小，区间每个数 K 次方和
const int maxn = 1e5 + 10;
const int mod = 1e9 + 7;
int N,M,K;
LL n,m,seed,vmax,ret;
LL rnd(){
    ret = seed;
    seed = (seed * 7 + 13) % mod;
    return ret;
}
LL quick_power(LL a,LL b,LL Mod){
    LL ans = 1;
```

```
    a %= Mod;
    while(b){
        if(b & 1) ans = (ans * a) % Mod;
        b >>= 1;
        a = a * a % Mod;
    }
    return ans;
}
#define IT set<node>::iterator
struct node{
    int l,r;
    mutable LL v; //表示可变的变量, 突破 const 函数的限制
    node(int l,int r = -1,LL v = 0):l(l),r(r),v(v) {}
    friend bool operator < (node a,node b){
        return a.l < b.l;
    }
};
set<node>s;
//分割出左端点包含 l 的区间
IT split(int pos){
    IT it = s.lower_bound(node(pos));
    if(it != s.end() && it->l == pos) return it;
    --it;
    int L = it->l,R = it->r;
    LL V = it->v;
    s.erase(it);
    s.insert(node(L,pos - 1,V));
    return s.insert(node(pos,R,V)).first;
}
//推平 l-r 区间
void assign(int l,int r,LL val){
    IT itr = split(r + 1),itl = split(l); //一定要先 split(r + 1)
    s.erase(itl,itr);
    s.insert(node(l,r,val));
}
void add(int l,int r,LL val){
    IT itr = split(r + 1),itl = split(l);
    for(;itl != itr; ++itl) itl->v += val;
}
LL ranks(int l,int r,int k){ //第 k 小
    vector<PLI> vp;
    IT itr = split(r + 1),itl = split(l);
    vp.clear();
    for(;itl != itr; ++itl) vp.pb(mp(itl->v,itl->r - itl->l + 1));
    sort(vp.begin(),vp.end());
    for(int i = 0 ; i < vp.size(); i ++){
        k -= vp[i].se;
        if(k <= 0) return vp[i].fi;
    }
}

LL sum(int l,int r,LL k,LL Mod){
    IT itr = split(r + 1),itl = split(l);
```

```
    LL ans = 0;
    for(;itl != itr; itl++) ans = (ans + (LL)(itl->r - itl->l + 1) *
    ↪ quick_power(itl->v,k,Mod)) % Mod;
    return ans;
}
LL a[maxn];
int main(){
    scanf("%lld%lld%lld%lld",&n,&m,&seed,&vmax);
    for(int i = 1; i <= n ; i ++){
        a[i] = (rnd() % vmax) + 1;
        s.insert(node(i,i,a[i]));
    }
    s.insert(node(n + 1,n + 1,0)); //防止查询到空
    for(int i = 1; i <= m ; i ++){
        int op = (rnd() % 4) + 1;
        int l = (rnd() % n) + 1,r = (rnd() % n) + 1;
        if(l > r) swap(l,r);
        int x,y;
        if(op == 3) x = (rnd() % (r - l + 1)) + 1;
        else x = (rnd() % vmax) + 1;
        if(op == 4) y = (rnd() % vmax) + 1;
        //以上随机 op,x,y,l,r
        if(op == 1) add(l,r,x); //区间加
        else if(op == 2) assign(l,r,x); //推平
        else if(op == 3) Prl(ranks(l,r,x)); //k 小
        else if(op == 4) Prl(sum(l,r,x,y)); //区间 x 的次方和模 y
    }
    return 0;
}
```

3.9 动态开点线段树

```
/* 动态线段树
如果线段树要操作的范围在 1 - INF（一个空间存不下的大小）
如果离线可以考虑离散化，如果强制在线就要考虑一下动态线段树了
动态线段树不是一口气把全部结点都开完，而是动态的只开要开的结点
*/
//模板，hdu1199 n 个操作为把 a - b 范围涂成黑色或者白色，查询为最长的连续白色区间，同样长输出
↪ 最左边
const int maxn = 2010;
const int INF = 0x3f3f3f3f;
const int mod = 1e9 + 7;

int N,M;
struct Tree{
    LL sum;    //最大连续
    LL lsum;   //左连续
    LL rsum;   //右连续
    int lt,rt,lazy;
    void init(){
        lsum = rsum = sum = lt = rt = 0;
        lazy = -1;
    }
}tree[maxn * 60];
```

```
int tot;
void check(int &t){
    if(t) return;
    t = ++tot;
    tree[t].init();
}
void add(int &t,LL L,LL R,int v){
    if(v){
        tree[t].sum = tree[t].lsum = tree[t].rsum = R - L + 1;
    }else{
        tree[t].sum = tree[t].lsum = tree[t].rsum = 0;
    }
    tree[t].lazy = v;
}
void Pushdown(int& t,LL L,LL R){
    if(tree[t].lazy == -1) return;
    int &lt = tree[t].lt; int &rt = tree[t].rt;
    LL M = (L + R) >> 1;
    check(lt); check(rt);
    add(lt,L,M,tree[t].lazy);
    add(rt,M + 1,R,tree[t].lazy);
    tree[t].lazy = -1;
}
void Pushup(int t,LL L,LL R){
    int &ls = tree[t].lt; int &rs = tree[t].rt;
    LL M = (L + R) >> 1;
    check(ls); check(rs);
    tree[t].sum = max(tree[ls].sum,tree[rs].sum);
    tree[t].sum = max(tree[t].sum,tree[ls].rsum + tree[rs].lsum);
    tree[t].lsum = tree[ls].lsum;
    if(tree[ls].lsum == M - L + 1){
        tree[t].lsum = tree[ls].lsum + tree[rs].lsum;
    }
    tree[t].rsum = tree[rs].rsum;
    if(tree[rs].rsum == R - M){
        tree[t].rsum = tree[rs].rsum + tree[ls].rsum;
    }
}
//如果是单点修改的话，一个小优化是一路直接修改下去而不是从底部往上 Pushup
void update(int &t,int q1,int q2,LL L,LL R,int v){
    check(t);
    if(q1 <= L && R <= q2){
        add(t,L,R,v);
        return;
    }
    Pushdown(t,L,R);
    LL m = (L + R) >> 1;
    if(q1 <= m) update(tree[t].lt,q1,q2,L,m,v);
    if(q2 > m) update(tree[t].rt,q1,q2,m + 1,R,v);
    Pushup(t,L,R);
}
LL Left,Right;
//查询有时可用的一个优化是如果这个点不存在，不需要开辟空间继续查询，之间 return 0 即可
```

```
void query(int t,LL L,LL R){
    if(L == R){
        Left = L;
        Right = R;
        return;
    }
    check(tree[t].lt); check(tree[t].rt);
    int ls = tree[t].lt; int rs = tree[t].rt;
    LL M = (L + R) >> 1;
    if(tree[ls].sum == tree[t].sum) query(ls,L,M);
    else if(tree[rs].sum == tree[t].sum) query(rs,M + 1,R);
    else{
        Left = M - tree[ls].rsum + 1;
        Right = M + tree[rs].lsum;
        return;
    }

}
int main()
{
    while(~Sca(N)){
        LL L = 1; LL R = 2147483647;
        tot = 0;
        int root = 0;
        update(root,L,R,L,R,0);
        For(i,1,N){
            LL l,r;
            char op[3];
            scanf("%lld%lld%s",&l,&r,&op);
            if(op[0] == 'w'){
                update(root,l,r,L,R,1);
            }else{
                update(root,l,r,L,R,0);
            }
        }
        if(!tree[root].sum){
            puts("Oh, my god");
            continue;
        }
        query(root,L,R);
        printf("%lld %lld\n",Left,Right);
    }
    return 0;
}
```

3.10 李超树

```
/* 李超线段树
用来维护二维平面上很多条线段（直线）在 x = x0 上的最值问题
定义：
1. 永久化标记：即线段树标记不删除，每个结点维护的也不一定是最优的信息，需要查询的时候一路统计
↪ 标记达到最优
2. 优势线段：李超树每个节点含有一个优势线段，意为完全覆盖当前区间且在当前区间 mid 处相比于其
↪ 他该位置线段最大（小）的线段，李超树的 id 记录的即为当前的优势线段
```

```
3. 核心思想：假设维护最大值，每次插入线段时，如果斜率较大的线段为优势线段，则斜率较小的线段只有在左子树才有机会比优势线段大
如果斜率较小的线段为优势线段，则较大的线段只有在右子树才能翻盘
修改 (log(n))²(但是常数不大)，查询 log(n),
*/
/* 模板
操作 1. 增加一条斜率为 P，起始点为 S 的直线
操作 2. 询问 k 点上所有直线的最大值
*/
const int maxn = 5e4 + 10;
int N,M,K;
struct Tree{
    int l,r;
    double S,P;  //P 为斜率,S 为初始值
}tree[maxn << 2];
void Build(int t,int l,int r){
    tree[t].l = l; tree[t].r = r;
    tree[t].S = tree[t].P = 0;
    if(l == r) return;
    int m = l + r >> 1;
    Build(t << 1,l,m); Build(t << 1 | 1,m + 1,r);
}
void update(int t,double S,double P){
    int mid = tree[t].l + tree[t].r >> 1;
    if(P > tree[t].P) swap(tree[t].P,P), swap(tree[t].S,S);
    if(S + P * mid < tree[t].S + tree[t].P * mid){ //斜率大的更大就用小的更新左边
        if(tree[t].l == tree[t].r) return;
        update(t << 1,S,P);
    }else{                                          //否则用大的更新右边
        swap(tree[t].P,P); swap(tree[t].S,S);
        if(tree[t].l == tree[t].r) return;
        update(t << 1 | 1,S,P);
    }
}
double ans;
void query(int t,int k){
    int mid = tree[t].l + tree[t].r >> 1;
    ans = max(ans,k * tree[t].P + tree[t].S);
    if(tree[t].l == tree[t].r) return ;
    if(k <= mid) query(t << 1,k);
    else query(t << 1 | 1,k);
}
char op[10];
int main(){
    Sca(N);
    Build(1,1,50000);
    for(int i = 1; i <= N ; i ++){
        scanf("%s",op);
        if(op[0] == 'Q'){
            ans = 0;
            query(1,read());
            Pri((int)(ans / 100));
        }else{
```

```
            double S,P; scanf("%lf%lf",&S,&P);
            S -= P;
            update(1,S,P);
        }
    }
    return 0;
}
//模板 2
//1. 加入 x1,y1,x2,y2 为端点的线段
//2. 查询 k 点最大值
//注意 x1,x2 相同的时候需要特判
const int maxn = 4e4 + 10;
const int maxm = 1e5 + 10;
int N,M,K;
double S[maxm],P[maxm];
struct Tree{
    int l,r;
    int id;
}tree[maxn << 2];
void Build(int t,int l,int r){
    tree[t].l = l; tree[t].r = r;
    tree[t].id = 0;
    if(l == r) return;
    int m = l + r >> 1;
    Build(t << 1,l,m);
    Build(t << 1 | 1,m + 1,r);
}
bool dcmp(double a){
    return fabs(a) > eps;
}
void update(int t,int l,int r,int id){
    int m = (tree[t].l + tree[t].r) >> 1;
    if(l <= tree[t].l && tree[t].r <= r){
        int &a = id;
        int &b = tree[t].id;
        if(P[a] > P[b]) swap(a,b);
        if(m * P[a] + S[a] > m * P[b] + S[b] || (!dcmp(m * P[a] + S[a] - m * P[b] + S[b])
        ↪ && a < b)){
            swap(a,b);
            if(tree[t].l == tree[t].r) return;
            update(t << 1 | 1,l,r,id);
        }else{
            if(tree[t].l == tree[t].r) return;
            update(t << 1,l,r,id);
        }
        return;
    }
    if(r <= m) update(t << 1,l,r,id);
    else if(l > m) update(t << 1 | 1,l,r,id);
    else{
        update(t << 1,l,m,id);
        update(t << 1 | 1,m + 1,r,id);
    }
```

```
}
int Id;
void query(int t,int k){
    int a = tree[t].id;
    double Ans = P[Id] * k + S[Id];
    if(Ans < P[a] * k + S[a] || (!dcmp(Ans - P[a] * k - S[a]) && Id > a)){
        Id = a;
    }
    if(tree[t].l == tree[t].r) return;
    int m = (tree[t].l + tree[t].r) >> 1;
    if(k <= m) query(t << 1,k);
    else query(t << 1 | 1,k);
}
int main(){
    Sca(N); Id = 0; int tot = 0;
    Build(1,1,40000);
    for(int i = 1; i <= N ; i ++){
        int op = read();
        if(!op){
            int k = (read() + Id - 1) % 39989 + 1;
            Id = 0; query(1,k);
            Pri(Id);
        }else{
            int x0 = (read() + Id - 1) % 39989 + 1;
            int y0 = (read() + Id - 1) % 1000000000 + 1;
            int x1 = (read() + Id - 1) % 39989 + 1;
            int y1 = (read() + Id - 1) % 1000000000 + 1;
            tot++;
            if(x0 == x1){
                P[tot] = 0;
                S[tot] = max(y1,y0);
            }else{
                P[tot] = 1.0 * (y1 - y0) / (x1 - x0);
                S[tot] = y1 - x1 * P[tot];
            }
            update(1,min(x0,x1),max(x0,x1),tot);
        }
    }
    return 0;
}
//模板 3
/*
题意：树链上每个点添加形如 a * dis + b 的值的一个点，每次查询树链上最小点
树链剖分 + 李超树
关于李超树的区间查询：
用一个 Min 维护区间最小值,Min 的组成来源为两个子树的 Min 的较小值和本结点优势线段在两个端点
↪ 的较小值
如果查询包含整个区间则直接返回 Min，否则来源为正常区间最小值查找 + 本结点优势线段在查询两端的
↪ 较小值
*/
const int maxn = 1e5 + 10;
const LL INF = 123456789123456789;
int N,M,K;
```

```
struct Edge{
    int to,next;
    LL dis;
}edge[maxn * 2];
int head[maxn],tot;
void init(){
    for(int i = 0 ; i <= N ; i ++) head[i] = -1;
    tot = 0;
}
void add(int u,int v,LL w){
    edge[tot].to = v;
    edge[tot].next = head[u];
    edge[tot].dis = w;
    head[u] = tot++;
}
int dep[maxn],top[maxn],fa[maxn],pos[maxn];
int sz[maxn],son[maxn],to[maxn];
LL dis[maxn];
void dfs1(int t,int la){
    sz[t] = 1; son[t] = t;
    int heavy = 0;
    for(int i = head[t]; ~i ; i = edge[i].next){
        int v = edge[i].to;
        if(v == la) continue;
        dis[v] = dis[t] + edge[i].dis;
        dep[v] = dep[t] + 1;
        fa[v] = t;
        dfs1(v,t);
        if(sz[v] > heavy){
            heavy = sz[v];
            son[t] = v;
        }
        sz[t] += sz[v];
    }
}
int cnt;
void dfs2(int t,int la){
    top[t] = la;
    pos[t] = ++cnt; to[cnt] = t;
    if(son[t] == t) return;
    dfs2(son[t],la);
    for(int i = head[t]; ~i ; i = edge[i].next){
        int v = edge[i].to;
        if((fa[t] == v) || (v == son[t])) continue;
        dfs2(v,v);
    }
}
LL S[maxn * 2],P[maxn * 2];
int ttt;
struct Tree{
    int l,r,id;
    LL Min;
}tree[maxn << 2];
```

```
void Pushup(int t){
    if(tree[t].l == tree[t].r) return;
    tree[t].Min = min(tree[t].Min,min(tree[t << 1].Min,tree[t << 1 | 1].Min));
}
void Build(int t,int l,int r){
    tree[t].l = l; tree[t].r = r;
    tree[t].id = 0; tree[t].Min = INF;
    if(l == r) return;
    int m = l + r >> 1;
    Build(t << 1,l,m); Build(t << 1 | 1,m + 1,r);
}
void update(int t,int l,int r,int id){  //区间查询
    int m = tree[t].l + tree[t].r >> 1;
    if(l <= tree[t].l && tree[t].r <= r){
        int &a = id; int &b = tree[t].id;
        if(P[a] > P[b]) swap(a,b);
        if(dis[to[m]] * P[a] + S[a] < dis[to[m]] * P[b] + S[b]){
            swap(a,b);
            if(tree[t].l != tree[t].r) update(t << 1,l,r,id);
        }else{
            if(tree[t].l != tree[t].r) update(t << 1 | 1,l,r,id);
        }
        Pushup(t);
        tree[t].Min = min(tree[t].Min,dis[to[tree[t].l]] * P[b] + S[b]);
        tree[t].Min = min(tree[t].Min,dis[to[tree[t].r]] * P[b] + S[b]);
        return;
    }
    if(r <= m) update(t << 1,l,r,id);
    else if(l > m) update(t << 1 | 1,l,r,id);
    else{
        update(t << 1,l,m,id);
        update(t << 1 | 1,m + 1,r,id);
    }
    Pushup(t);
}
LL query(int t,int l,int r){
    if(l <= tree[t].l && tree[t].r <= r) return tree[t].Min;
    int m = (tree[t].l + tree[t].r) >> 1;
    LL ans = INF;
    ans = min(ans,dis[to[l]] * P[tree[t].id] + S[tree[t].id]);
    ans = min(ans,dis[to[r]] * P[tree[t].id] + S[tree[t].id]);
    if(r <= m) return min(ans,query(t << 1,l,r));
    else if(l > m) return min(ans,query(t << 1 | 1,l,r));
    else{
        return min(ans,min(query(t << 1,l,m),query(t << 1 | 1,m + 1,r)));
    }
}
int lca(int x,int y){
    while(top[x] != top[y]) dep[top[x]] > dep[top[y]]?x = fa[top[x]]:y = fa[top[y]];
    return dep[x] < dep[y]?x:y;
}
void update(int u,int v,int id){
    while(top[u] != top[v]){
```

```
        if(dep[top[u]] < dep[top[v]]) swap(u,v);
        update(1,pos[top[u]],pos[u],id);
        u = fa[top[u]];
    }
    if(dep[u] > dep[v]) swap(u,v);
    update(1,pos[u],pos[v],id);
}
LL query(int u,int v){
    LL ans = INF;
    while(top[u] != top[v]){
        if(dep[top[u]] < dep[top[v]]) swap(u,v);
        ans = min(ans,query(1,pos[top[u]],pos[u]));
        u = fa[top[u]];
    }
    if(dep[u] > dep[v]) swap(u,v);
    ans = min(ans,query(1,pos[u],pos[v]));
    return ans;
}
int main(){
    Sca2(N,M); init();
    P[0] = 0; S[0] = INF;
    for(int i = 1; i <= N - 1; i ++){
        int u = read(),v = read();
        LL w = read();
        add(u,v,w); add(v,u,w);
    }
    int root = 1;
    dfs1(root,-1); cnt = 0;
    dfs2(root,root);
    Build(1,1,N); ttt = 0;
    while(M--){
        int op = read();
        if(op == 1){
            int s = read(),t = read();
            LL a = read(),b = read();
            int x = lca(s,t);
            ttt++; P[ttt] = -a; S[ttt] = a * dis[s] + b;
            update(s,x,ttt);
            ttt++; P[ttt] = a; S[ttt] = a * (dis[s] - 2 * dis[x]) + b;
            update(x,t,ttt);
        }else{
            int s = read(),t = read();
            Prl(query(s,t));
        }
    }
    return 0;
}
```

3.11 左偏树

```
//左偏树（可并堆）
//用来合并两个堆（优先队列）的数据结构
/*
模板 op1: 合并两个下标所在的堆
```

```
op:2 输出下标所在堆的最小值并 pop, key 值相同先 pop 下标小的那个
做法：维护每个小根堆
*/
const int maxn = 1e5 +10;

int N,M;
struct node{
    int l,r,dis,key;
}tree[maxn];
int fa[maxn];
int find(int x){
    if(fa[x] == x) return x;
    return fa[x] = find(fa[x]);    //路径压缩
}
int merge(int a,int b){
    if(!a) return b;
    if(!b) return a;
    if(tree[a].key >tree[b].key) swap(a,b);
    if((tree[a].key == tree[b].key) && (a > b)) swap(a,b);
    tree[a].r = merge(tree[a].r,b);
    fa[tree[a].r] = a;
    if(tree[tree[a].l].dis < tree[tree[a].r].dis)
        swap(tree[a].l,tree[a].r);
    if(tree[a].r) tree[a].dis = tree[tree[a].r].dis + 1;
    else tree[a].dis = 0;
    return a;
}
bool vis[maxn];
int pop(int a){
    int l = tree[a].l;
    int r = tree[a].r;
    fa[l] = l; fa[r] = r;
    vis[a] = 1;
    return fa[a]= merge(l,r);  //少了这部不能路径压缩
}
int a[maxn];

int main(){
    scanf("%d%d",&N,&M);
    for(int i = 1; i <= N ; i ++){
        scanf("%d",&tree[i].key); fa[i] = i;
        tree[i].l = tree[i].r = tree[i].dis = 0;
    }
    for(int i = 1; i <= M ; i ++){
        int op; scanf("%d",&op);
        if(op == 1){
            int x,y; scanf("%d%d",&x,&y);
            if(vis[x] || vis[y]) continue;
            x = find(x); y = find(y);
            if(x == y) continue;
            merge(x,y);
        }else{
            int x; scanf("%d",x);
```

```
            if(vis[x]){
                puts("-1");continue;
            }
            x = find(x);
            printf("%d\n",tree[x].key);
            pop(x);
        }
    }
    return 0;
}
```

3.12 Splay

```
//Splay 伸展树
/*
再二叉搜索树的基础上增加了伸展 (splay) 功能，降低了查询和插入的时间复杂度。
本质上是通过将每一次查询的节点都进行伸展使其靠近根节点
 */
 //luogu 普通平衡树
const int maxn = 1e5 + 10;
const int INF = 0x3f3f3f3f;
const int mod = 1e9 + 7;
int N,M,K;
struct Splay{
    #define root e[0].ch[1]
    struct node{
        int ch[2];
        int sum,num;
        int v,fa;
    }e[maxn];
    int n,points;
    void update(int x){
        e[x].sum = e[e[x].ch[0]].sum + e[e[x].ch[1]].sum + e[x].num;
    }
    int id(int x){
        return x == e[e[x].fa].ch[0]?0:1;
    }
    void connect(int x,int y,int p){
        e[x].fa = y;
        e[y].ch[p] = x;
    }
    int find(int v){
        int now = root;
        while(1){
            if(e[now].v == v){
                splay(now,root);
                return now;
            }
            int next = v < e[now].v?0:1;
            if(!e[now].ch[next]) return 0;
            now = e[now].ch[next];
        }
        return 0;
    }
```

```
void rotate(int x){
    int y = e[x].fa;
    int z = e[y].fa;
    int ix = id(x),iy = id(y);
    connect(e[x].ch[ix ^ 1],y,ix);
    connect(y,x,ix ^ 1);
    connect(x,z,iy);
    update(y); update(x);
}
void splay(int u,int v){
    v = e[v].fa;
    while(e[u].fa != v){
        int fu = e[u].fa;
        if(e[fu].fa == v) rotate(u);
        else if(id(u) == id(fu)){
            rotate(fu);
            rotate(u);
        }else{
            rotate(u);
            rotate(u);
        }
    }
}
int crenode(int v,int father){
    n++;
    e[n].ch[0] = e[n].ch[1] = 0;
    e[n].fa = father;
    e[n].num = e[n].sum = 1;
    e[n].v = v;
    return n;
}
void destroy(int x){
    e[x].v = e[x].fa = e[x].num = e[x].sum = e[x].v = 0;
    if (x == n) n--;
}
int insert(int v){
    points++;
    if(points == 1){
        n = 0;
        root = 1;
        crenode(v,0);
        return 1;
    }else{
        int now = root;
        while(1){
            e[now].sum++;
            if(v == e[now].v){
                e[now].num++;
                return now;
            }
            int next = v < e[now].v?0:1;
            if(!e[now].ch[next]){
                crenode(v,now);
```

```
                e[now].ch[next] = n;
                return n;
            }
            now = e[now].ch[next];
        }
    }
}
void push(int v){  //添加元素
    int add = insert(v);
    splay(add,root);
}
void pop(int x){
    int pos = find(x);  //找到节点并作为根
    if(!pos) return;
    points--;
    if(e[pos].num > 1){
        e[pos].num--;
        e[pos].sum--;   //他已经是根节点了，不需要更新祖先的 sum 了
        return;
    }
    if(!e[pos].ch[0]){        //没有左孩子就直接删除，右孩子作为根
        root = e[pos].ch[1];
        e[root].fa = 0;
    }else{          //有左孩子： 将左子树最大的点转到根，将右子树连到它上
        int lef = e[pos].ch[0];
        while(e[lef].ch[1]) lef = e[lef].ch[1];
        splay(lef,e[pos].ch[0]);
        int rig = e[pos].ch[1];
        connect(rig,lef,1); connect(lef,0,1);
        update(lef);
    }
    destroy(pos);
}
int atrank(int x){
    if(x > points) return -INF;
    int now = root;
    while(1){
        int mid = e[now].sum - e[e[now].ch[1]].sum;
        if(x > mid){
            x -= mid;
            now = e[now].ch[1];
        }else if(x <= e[e[now].ch[0]].sum){
            now = e[now].ch[0];
        }else break;
    }
    splay(now,root);
    return e[now].v;
}
int rank(int x){
    int now = find(x);
    if(!now) return 0;
    return e[e[now].ch[0]].sum + 1;
}
```

```
    int upper(int v){
        int now = root;
        int ans = INF;
        while(now){
            if(e[now].v > v && e[now].v < ans) ans = e[now].v;
            if(v < e[now].v) now = e[now].ch[0];
            else now = e[now].ch[1];
        }
        return ans;
    }
    int lower(int v){
        int now = root;
        int ans = -INF;
        while(now){
            if(e[now].v < v && e[now].v > ans) ans = e[now].v;
            if(v > e[now].v) now = e[now].ch[1];
            else now = e[now].ch[0];
        }
        return ans;
    }
    #undef root
}F;
int main(){
    Sca(N);
    while(N--){
        int x,op; Sca2(op,x);
        if(op == 1) F.push(x);     //插入一个元素
        else if(op == 2) F.pop(x);   //删除一个元素
        else if(op == 3) Pri(F.rank(x));   //查询 x 数的排名（排名定义为比当前数小的数的个
        ↪  数 +1+1。若有多个相同的数，因输出最小的排名）
        else if(op == 4) Pri(F.atrank(x)); //查询排名为 x 的数
        else if(op == 5) Pri(F.lower(x));  //求 xx 的前驱（前驱定义为小于 xx，且最大的数）
        else Pri(F.upper(x)); //求 xx 的后继（后继定义为大于 xx，且最小的数）
    }
    return 0;
}

//Splay 合并咋整鸭
//启发式合并，将小的合并到大的上去，用中序遍历的方式暴力插入，时间复杂度 ln 级别
/* 题意
永无乡包含 n 座岛，编号从 1 到 n，
每座岛都有自己的独一无二的重要度，按照重要度可以将这 n 座岛排名，
名次用 1 到 n 来表示。某些岛之间由巨大的桥连接，通过桥可以从一个岛到达另一个岛。
如果从岛 a 出发经过若干座（含 0 座）桥可以到达岛 b，则称岛 a 和岛 b 是连通的。
B x y 表示在岛 x 与岛 y 之间修建一座新桥。
Q x k 表示询问当前与岛 x 连通的所有岛中第 k 重要的是哪座岛，
即所有与岛 x 连通的岛中重要度排名第 k 小的岛是哪座，请你输出那个岛的编号。
 */
 const int maxn = 5e5 + 10;
const int maxm = 3e5 + 10;
const int INF = 0x3f3f3f3f;
const int mod = 1e9 + 7;
int N,M,K;
```

```
struct node{
    int ch[2];
    int sum,v,fa,id;
}e[maxn];
int n,points[maxn];
int root[maxn],tree[maxn];
int crenode(int v,int id,int fa){
    n++;
    e[n].ch[0] = e[n].ch[1] = 0;
    e[n].fa = fa;
    e[n].sum = 1;
    e[n].v = v; e[n].id = id;
    return n;
}
int id(int x){
    return x == e[e[x].fa].ch[0]?0:1;
}
void connect(int u,int v,int p){
    e[u].fa = v; e[v].ch[p] = u;
}
void update(int x){
    e[x].sum = e[e[x].ch[0]].sum + e[e[x].ch[1]].sum + 1;
}
int insert(int t,int num){
    points[num]++;
    int now = root[num];
    int v = e[t].v;
    while(1){
        e[now].sum++;
        int nxt = v < e[now].v?0:1;
        if(!e[now].ch[nxt]){
            connect(t,now,nxt);
            e[t].ch[0] = e[t].ch[1] = 0;
            e[t].sum = 1;
            return t;
        }
        now = e[now].ch[nxt];
    }
}

void rotate(int x){
    int y = e[x].fa;
    int z = e[y].fa;
    int ix = id(x),iy = id(y);
    connect(e[x].ch[ix ^ 1],y,ix);
    connect(y,x,ix ^ 1);
    connect(x,z,iy);
    update(y); update(x);
}
void splay(int u,int v){
    v = e[v].fa;
    while(e[u].fa != v){
        int fu = e[u].fa;
```

```
        if(e[fu].fa == v) rotate(u);
        else if(id(u) == id(fu)){
            rotate(fu);
            rotate(u);
        }else{
            rotate(u);
            rotate(u);
        }
    }
    if(v <= N) root[v] = u;
}

void push(int t,int num){
    int x = insert(t,num);
    splay(x,root[num]);
}
int find(int p){
    return p == tree[p]?p:tree[p] = find(tree[p]);
}
void dfs(int t,int num){
    int l = e[t].ch[0],r = e[t].ch[1];
    if(l) dfs(l,num);
    push(t,num);        //中序遍历合并是 log 复杂度
    if(r) dfs(r,num);
}
void Union(int u,int v){
    int x = find(u),y = find(v);
    if(x == y) return;
    if(points[x] > points[y]) swap(x,y);
    tree[x] = y;
    dfs(root[x],y);
}
int Kth(int id,int k){
    if(points[id] < k) return -1;
    int now = root[id];
    while(1){
        int mid = e[now].sum - e[e[now].ch[1]].sum;
        if(k > mid){
            k -= mid;
            now = e[now].ch[1];
        }else if(k <= e[e[now].ch[0]].sum){
            now = e[now].ch[0];
        }else break;
    }
    splay(now,root[id]);
    return e[now].id;
}
int main(){
    Sca2(N,M);
    n = N;      //1 ~ N 表示每颗平衡树的 0 点
    for(int i = 1; i <= N ; i ++){
        tree[i] = i;
        int v; Sca(v);
```

```
        points[i] = 1;
        root[i] = crenode(v,i,i);
    }
    for(int i = 1; i <= M; i ++){
        int u,v; Sca2(u,v);
        Union(v,u);
    }
    int Q; Sca(Q);
    while(Q--){
        char op[2]; int x,y;
        scanf("%s%d%d",op,&x,&y);
        if(op[0] == 'B') Union(x,y);
        else Pri(Kth(find(x),y));
    }
    return 0;
}
```

3.13 LCT

```
//link cut tree (LCT)
/* lct 的性质
    1.lct 是将一棵树分为若干的 Splay
    每一个 Splay 维护的是一条从上到下按在原树中深度严格递增的路径，且中序遍历 Splay 得到的每
↪ 个点的深度序列严格递增。
    2. 每个节点包含且仅包含于一个 Splay 中
    3. 边分为实边和虚边，实边包含在 Splay 中，而虚边总是由一棵 Splay 指向另一个节点（指向该
↪ Splay 中中序遍历最靠前的点在原树中的父亲）。因为性质 2，当某点在原树中有多个儿子时，只能向
↪ 其中一个儿子拉一条实链（只认一个儿子），而其它儿子是不能在这个 Splay 中的。那么为了保持树
↪ 的形状，我们要让到其它儿子的边变为虚边，由对应儿子所属的 Splay 的根节点的父亲指向该点，而
↪ 从该点并不能直接访问该儿子（认父不认子）。
 */
const int maxn = 3e5 + 10;
int N,M,K;
#define lc e[x].ch[0]
#define rc e[x].ch[1]
#define fc e[x].fa
struct node{
    int fa,ch[2];
    int v,s;   //v 存储当前节点信息，s 存储子树的异或和
    bool rev;
}e[maxn];
int Stack[maxn];
//1 轻边重边主要看父亲认不认儿子（儿子一定认父亲）
void pushup(int x){
    e[x].s = e[lc].s ^ e[rc].s ^ e[x].v;
}
bool nroot(int x){   //判断节点是否为一个 Splay 的根 , 返回 1 代表不是根
    return e[fc].ch[0] == x || e[fc].ch[1] == x; //如果是根，他的父亲和他之间是轻边（父亲不
    ↪ 认儿子）
}
void rev(int x){     //翻转子树
    swap(lc,rc); e[x].rev ^= 1;
}
void pushdown(int x){   //向下更新
```

```
    if(e[x].rev){
        if(lc) rev(lc);
        if(rc) rev(rc);
        e[x].rev = 0;
    }
}
void rotate(int x){    //在 x 在自己的 splay 里向上旋转
    int y = e[x].fa; int z = e[y].fa;
    int ix = e[y].ch[1] == x; int iy = e[z].ch[1] == y;
    int w = e[x].ch[ix ^ 1];
    if(nroot(y)) e[z].ch[iy] = x;   //不判断会错
    if(w) e[w].fa = y;
    e[x].ch[ix ^ 1] = y; e[y].ch[ix] = w;
    e[y].fa = x; e[x].fa = z;
    pushup(y);
}
void splay(int x){ //只传了一个参数的原因是所有目标都是该 Splay 的根
    int y = x,top = 0;
    Stack[++top] = y;
    while(nroot(y)) Stack[++top] = y = e[y].fa;
    while(top) pushdown(Stack[top--]);  //先下方整条路径上的信息
    while(nroot(x)){
        y = e[x].fa; int z = e[y].fa;
        if(nroot(y)) rotate((e[y].ch[0] == x) ^ (e[z].ch[0] == y)?x:y);  //直连就转父亲,
        ↪ 否则转自己
        rotate(x);
    }
    pushup(x);
}
void access(int x){  //联通 x 到根，使他们在同一个 Splay
    for(int y = 0;x;x = e[x].fa){
        splay(x);
        e[x].ch[1] = y;
        pushup(x);
        y = x;
    }
}
//换原树的根
void makeroot(int x){
    access(x); splay(x); rev(x);
}
//找根，即 x 所属的 splay 中深度最浅的那个点
//findroot(x)==findroot(y) 表明 x,y 在同一棵树中
int findroot(int x){
    access(x); splay(x);
    while(lc) pushdown(x),x = lc;
    splay(x);
    return x;
}
//拉出 x-y 的路径成为一个 Splay (以 y 作为该 Splay 的根)
void split(int x,int y){
    makeroot(x);
    access(y); splay(y);
```

```
}
//连一条 x-y 的边（使 x 的父亲指向 y，连一条轻边）
void link(int x,int y){
    makeroot(x);
    if(findroot(y) != x) e[x].fa = y;
}
//将 x-y 的边断开
void cut(int x,int y){
    makeroot(x);
    if(findroot(y) == x && e[y].fa == x && !e[y].ch[0]){
        e[y].fa = e[x].ch[1] = 0;
        pushup(x);
    }
}
/* 如果保证断的边都存在可以这么写
void cut(int x,int y){
        split(y,x);
        e[y].fa = e[x].ch[1] = 0;
        pushup(x);
}
 */
int main(){
    Sca2(N,M);
    for(int i = 1; i <= N; i ++) Sca(e[i].v);
    while(M--){
        int op,x,y; Sca3(op,x,y);
        switch(op){
        case 0:split(x,y); Pri(e[y].s); break;
        case 1:link(x,y); break;
        case 2:cut(x,y); break;
        case 3:splay(x); e[x].v = y; break;
        }
    }
    return 0;
}

//LCT 的边转点
/*
树链剖分等算法的边转点一般是将边转化为深度交深的点
当用 LCT 维护类似于树链上最大的边的时候，可以采用 (u,v) 这条边转化为 (u,id),(id,v) 的方式，
↪ 其中
id 为 u,v 之间的边，即将边看作是一个点
模板：luoguP4172 水管局长
给一个 n 个点 m 条边的图，支持以下操作：
1. 询问 x 到 y 所有路径中，路径上最大边权的最小值
2. 删去一条边
做法：离线之后变为加边，用 LCT 维护一个最小生成树
*/

const double eps = 1e-9;
const int maxn = 2e5 + 10;
const int maxm = 1500000;
const int maxq = 1e5 + 10;
```

```
const int INF = 0x3f3f3f3f;
const int mod = 1e9 + 7;
int N,M,K,Q;
struct Edge{
        int u,v,t;
}edge[maxm + maxn];
struct Q{
        int k,x,y;
}q[maxm];
bool vis[maxm + maxn];
map<PII,int>ide;
int fa[maxm];
void init(){for(int i = 0 ; i <= N ; i ++) fa[i] = i;}
int find(int x){return x == fa[x]?x:fa[x] = find(fa[x]);}
void Union(int a,int b){
        a = find(a); b = find(b);
        fa[a] = b;
}
#define lc e[x].ch[0]
#define rc e[x].ch[1]
#define fc e[x].fa
struct node{
        int ch[2],fa;
        int iv,imax;
        bool rev;
}e[maxn + maxm];
void pushup(int x){
        int a = e[x].iv,b = e[lc].imax,c = e[rc].imax;
        if(edge[a].t < edge[b].t) a = b;
        if(edge[a].t < edge[c].t) a = c;
        e[x].imax = a;
}
bool nroot(int x){
        return e[fc].ch[0] == x || e[fc].ch[1] == x;
}
void rev(int x){
        swap(lc,rc); e[x].rev ^= 1;
}
void pushdown(int x){
        if(e[x].rev){
                if(lc) rev(lc);
                if(rc) rev(rc);
                e[x].rev = 0;
        }
}
void rotate(int x){
        int y = e[x].fa;int z = e[y].fa;
        int ix = e[y].ch[1] == x;
        int iy = e[z].ch[1] == y;
        int w = e[x].ch[ix ^ 1];
        if(nroot(y)) e[z].ch[iy] = x;
        if(w) e[w].fa = y;
        e[x].ch[ix ^ 1] = y; e[y].ch[ix] = w;
```

```
        e[y].fa = x;  e[x].fa = z;
        pushup(y);
}
int Stack[maxn + maxm];
void splay(int x){
        int y = x,top = 0;
        Stack[++top] = y;
        while(nroot(y)) Stack[++top] = y = e[y].fa;
        while(top) pushdown(Stack[top--]);
        while(nroot(x)){
                y = e[x].fa; int z = e[y].fa;
                if(nroot(y)) rotate((e[y].ch[0] == x) ^ (e[z].ch[0] == y)?x:y);
                rotate(x);
        }
        pushup(x);
}

void access(int x){
        for(int y = 0;x;x = e[x].fa){
                splay(x);
                e[x].ch[1] = y;
                pushup(x);
                y = x;
        }
}
void makeroot(int x){
        access(x); splay(x); rev(x);
}
int findroot(int x){
        access(x); splay(x);
        while(lc) pushdown(x),x = lc;
        splay(x);
        return x;
}
void split(int x,int y){
        makeroot(x);
        access(y); splay(y);
}
void link(int x,int y){
        makeroot(x);
        if(findroot(y) != x) e[x].fa = y;
}
void cut(int x,int y){
        split(y,x);
        e[y].fa = e[x].ch[1] = 0;
        pushup(x);
}
void add(int u,int v,int id){
        if(find(u) != find(v)){
                Union(u,v);
                link(u,id); link(id,v);
                return;
        }
```

```
        split(u,v);
        int tmp = e[v].imax;
        if(edge[tmp].t > edge[id].t){
                cut(edge[tmp].u,tmp);
                cut(tmp,edge[tmp].v);
                link(u,id); link(id,v);
        }
}
int ans[maxm];
bool cmp(Edge a,Edge b){
        return a.t < b.t;
}
int main(){
        N = read(),M = read(),Q = read();init();
        edge[0].t = -1;
        for(int i = 1; i <= N; i ++) e[i].imax = e[i].iv = 0;
        for(int i = 1 + N; i <= M + N; i ++){
                edge[i].u = read(); edge[i].v = read(); edge[i].t = read();
                if(edge[i].u > edge[i].v) swap(edge[i].u,edge[i].v);
        }
        sort(edge + 1 + N,edge + 1 + M + N,cmp);
        for(int i = 1 + N; i <= M + N; i ++){
                e[i].iv = e[i].imax = i;
                ide[mp(edge[i].u,edge[i].v)] = i;
        }
        for(int i = 1; i <= Q ; i ++){
                q[i].k = read(); q[i].x = read(); q[i].y = read();
                if(q[i].x > q[i].y) swap(q[i].x,q[i].y);
                if(q[i].k == 2) vis[ide[mp(q[i].x,q[i].y)]] = 1;
        }
        int cnt = 0;
        for(int i = 1 + N; i <= M + N && cnt < N - 1; i ++){
                if(vis[i]) continue;
                int u = edge[i].u,v = edge[i].v;
                if(find(u) == find(v)) continue;
                cnt++;
                Union(u,v); link(u,i); link(i,v);
        }
        int tot = 0;
        for(int i = Q; i >= 1; i --){
                if(q[i].k == 1){
                        split(q[i].x,q[i].y);
                        ans[++tot] = edge[e[q[i].y].imax].t;
                }else{
                        add(q[i].x,q[i].y,ide[mp(q[i].x,q[i].y)]);
                }
        }
        for(int i = tot; i >= 1; i --) Pri(ans[i]);
        return 0;
}
```

3.14 主席树

3.14.1 静态区间第 k 小

```
//静态区间第 K 小 nlogn
const int maxn = 2e5 + 10;
int N,M,K;
int Hash[maxn],a[maxn];
int T[maxn];
struct Tree{
    int lt,rt,sum;
    void init(){
        lt = rt = sum = 0;
    }
}tree[maxn * 60];
int tot;
void newnode(int &t){
    t = ++tot;
    tree[t].init();
}
void Build(int &t,int l,int r){
    newnode(t);
    if(l == r) return;
    int m = l + r >> 1;
    Build(tree[t].lt,l,m); Build(tree[t].rt,m + 1,r);
}
void update(int &t,int pre,int l,int r,int p){
    newnode(t);
    tree[t] = tree[pre];
    tree[t].sum++;
    if(l == r) return;
    int m = l + r >> 1;
    if(p <= m) update(tree[t].lt,tree[pre].lt,l,m,p);
    else update(tree[t].rt,tree[pre].rt,m + 1,r,p);
}
int query(int L,int R,int l,int r,int k){
    if(l >= r) return l;
    int num = tree[tree[R].lt].sum - tree[tree[L].lt].sum;
    int m = l + r >> 1;
    if(num >= k) return query(tree[L].lt,tree[R].lt,l,m,k);
    else return query(tree[L].rt,tree[R].rt,m + 1,r,k - num);
}
int main(){
    Sca2(N,M);
    for(int i = 1; i <= N ; i ++) Hash[i] = a[i] = read();
    sort(Hash + 1,Hash + 1 + N);
    int cnt = unique(Hash + 1,Hash + 1 + N) - Hash - 1;
    Build(T[0],1,cnt);
    for(int i = 1; i <= N ; i ++){
        int p = lower_bound(Hash + 1,Hash + 1 + cnt,a[i]) - Hash;
        update(T[i],T[i - 1],1,cnt,p);
    }
    for(int i = 1; i <= M ; i ++){
        int l = read(),r = read(),k = read();
```

```
        Pri(Hash[query(T[l - 1],T[r],1,cnt,k)]);
    }
    return 0;
}
```

3.15 cdq 分治

3.15.1 三维偏序问题

```
//cdq 分治：陌上花开
//定义 node 的等级为 xyz 全不大于他的 node 的种数，求每个等级的 node 有多少种
//注意需要先将 xyz 完全相同的 node 合并才行
const double eps = 1e-9;
const int maxn = 1e5 + 10;
const int maxm = 2e5 + 10;
int N,M,K;
struct Node{
    int x,y,z,ans,cnt;
    Node(){}
    Node(int x,int y,int z,int ans,int cnt):x(x),y(y),z(z),ans(ans),cnt(cnt){}
    bool operator == (Node a){
        return x == a.x && y == a.y && z == a.z;
    }
}node[maxn],tmp[maxn];
int ans[maxn],sum[maxn];
int tree[maxm];
void add(int x,int y){
    for(;x <= M; x += x & -x) tree[x] += y;
}
int getsum(int x){
    int ans = 0;
    for(;x > 0; x -= x & -x) ans += tree[x];
    return ans;
}
bool cmp(Node a,Node b){
    if(a.x != b.x) return a.x < b.x;
    if(a.y != b.y) return a.y < b.y;
    return a.z < b.z;
}
void cdq(int l,int r){
    if(l >= r) return;
    int m = l + r >> 1;
    cdq(l,m); cdq(m + 1,r);
    int cnt1 = l,cnt2 = m + 1;
    for(int i = l; i <= r; i ++){
        if((cnt2 > r) || ((node[cnt1].y <= node[cnt2].y) && cnt1 <= m)){
            add(node[cnt1].z,node[cnt1].cnt);
            tmp[i] = node[cnt1++];
        }else{
            node[cnt2].ans += getsum(node[cnt2].z);
            tmp[i] = node[cnt2++];
        }
    }
    for(int i = l; i <= m; i ++) add(node[i].z,-node[i].cnt);
```

```
    for(int i = l; i <= r; i ++) node[i] = tmp[i];
}
int main(){
    Sca2(N,M); int cnt = 1;
    for(int i = 1; i <= N ; i ++) node[i] = Node(read(),read(),read(),0,1);
    sort(node + 1,node + 1 + N,cmp);
    for(int i = 2; i <= N ; i ++){
        if(node[i - 1] == node[i]) node[cnt].cnt++;
        else node[++cnt] = node[i];
    }
    cdq(1,cnt);
    for(int i = 1; i <= cnt ; i ++) sum[node[i].ans + node[i].cnt - 1] += node[i].cnt;
    for(int i = 0; i < N ; i ++) Pri(sum[i]);
    return 0;
}
```

3.16 kd 树

```
/* K-DTree 是用来解决 K 维空间中数点问题强有力的数据结构，可以在 (NlogN)-(N√N) 的时间复杂度内完成查询和修改。
建树类似于二叉平衡树，只是排序的依据每个维度轮流，第一层取所有节点 0 维度的中位数点作为结点，比他小的放左边，比他大的放右边
第二层就按照 1 维度来排序....
每个节点除了表示一个当前节点外，还存储子树信息，查询的时候如果有子树显然不符合条件，就 skip 掉 (类似于剪枝)
*/
//模板 1HDU4347. 给出 N 个 K 维度点，查询的时候给个点和 M，询问 N 个点中前 M 靠近给定点的点
const int maxn = 50010;
const int maxk = 7;
int N,M,K;
int idx;
struct point{
    int x[maxk];
    bool operator < (const point &u) const{
        return x[idx] < u.x[idx];  //每层排序的维度都不一样，用全局变量 idx 标识
    }
}p[maxn];
typedef pair<double,point>tp;
priority_queue<tp>nq;    //维护离查询点最近的点,fi 表示距离,se 表示点坐标
int pow2(int x){return x * x;}
struct kdTree{
    point pt[maxn << 2]; //结点表示的点坐标
    int son[maxn << 2];  //区间长度
    void build(int t,int l,int r,int dep = 0){
        if(l > r) return;
        son[t] = r - l;
        son[t << 1] = son[t << 1 | 1] = -1;
        idx = dep % K;
        int mid = l + r >> 1;
        nth_element(p + l,p + mid,p + r + 1);  //当前 idx 维度排名为中位数的点作为这个结点表示的点
        pt[t] = p[mid];
        build(t << 1,l,mid - 1,dep + 1);
        build(t << 1 | 1,mid + 1,r,dep + 1);
```

```
    }
    void query(int t,point q,int m,int dep = 0){
        if(son[t] == -1) return;
        tp nd(0,pt[t]);
        for(int i = 0 ; i < K; i ++) nd.first += pow2(nd.se.x[i] - q.x[i]);
        int dim = dep % K,x = t << 1,y = t << 1 | 1,flag = 0;
        if(q.x[dim] >= pt[t].x[dim]) swap(x,y);   //如果待查寻的点这个维度比他大就查询右子
        ↪ 树
        if(~son[x]) query(x,q,m,dep + 1);
        if(nq.size() < m) nq.push(nd),flag = 1;
        else{
            if(nd.first < nq.top().fi) nq.pop(),nq.push(nd);
            if(pow2(q.x[dim] - pt[t].x[dim]) < nq.top().fi) flag = 1; //如果这个维度的距
            ↪ 离已经比所有的都大了，就没有继续查的必要了
        }
        if(~son[y] && flag) query(y,q,m,dep + 1);
    }
}kd;
void print(point &t){
    for(int j = 0 ; j < K ; j ++) printf("%d%c",t.x[j],j == K - 1?'\n':' ');
}
int main(){
    while(~Sca2(N,K)){
        for(int i = 0; i < N ; i ++){
            for(int j = 0 ; j < K; j ++) Sca(p[i].x[j]);
        }
        kd.build(1,0,N - 1);
        int t = read();
        while(t--){
            point ask;
            for(int j = 0 ; j < K ; j ++) Sca(ask.x[j]);
            Sca(M); kd.query(1,ask,M);
            printf("the closest %d points are:\n",M);
            point pt[20];
            for(int j = 0;!nq.empty(); j ++) pt[j] = nq.top().second,nq.pop();
            for(int j = M - 1; j >= 0 ; j --) print(pt[j]);
        }
    }
    return 0;
}
//当题目需要在线的时候，就不能用上面的方法构树了
/*insert 函数动态加点，利用重构系数判断重构，节点中 Max,Min 表示该子树中该维度的最大（小）值
重构 kdtree 的参数选择，插入多查询少的情况，最优参数是接近 0.5 + x / (x + y) 的
x 是插入个数，y 是查询个数
插入少查询多的话，最优参数其实是更接近 0.7 + x / (x + y) 的， 查询再多也不建议参数低于 0.7
当然最优参数的话，有时间可以自己去测试调整
*/
//模板，操作 1 在坐标 (x,y) 中加权值 v，操作 2 查询子矩阵和
const int maxn = 1e5 + 10;
const int maxk = 2;
const int INF = 0x3f3f3f3f;
const int mod = 1e9 + 7;
int N,M,K;
```

```
int p[maxk];
int q[maxk][2];
struct Tree{
    int d[maxk];
    int l,r,sum,val,size;
    int Min[maxk],Max[maxk];
}tree[maxn];
int cnt = 0;
const double alpha = 0.75;  //重构系数，当左（右）子树权重超过一定比例的时候将子树重构
int idx,num,tmp[maxn],A;
inline void erase(int &x){ //删除子树中的所有点，把这些点加入 tmp
    if(!x) return;
    tmp[++num] = x;
    erase(tree[x].l); erase(tree[x].r);
    x = 0;
}
bool cmp(const int &a,const int &b){    //按照 idx 维度的大小排序
    return tree[a].d[idx] < tree[b].d[idx];
}
inline void update(int t){   //类似于 Pushup
    int l = tree[t].l,r = tree[t].r;
    tree[t].size = tree[l].size + tree[r].size + 1;
    tree[t].sum = tree[t].val + tree[l].sum + tree[r].sum;
    for(int i = 0 ; i < K; i ++){
        if(l){
            tree[t].Max[i] = max(tree[t].Max[i],tree[l].Max[i]);
            tree[t].Min[i] = min(tree[t].Min[i],tree[l].Min[i]);
        }
        if(r){
            tree[t].Max[i] = max(tree[t].Max[i],tree[r].Max[i]);
            tree[t].Min[i] = min(tree[t].Min[i],tree[r].Min[i]);
        }
    }
}
inline void build(int &t,int l,int r,int k){  //构造
    if(l > r) return;
    int mid = l + r >> 1; idx = k;
    nth_element(tmp + l,tmp + 1 + mid,tmp + r + 1,cmp);
    t = tmp[mid];
    tree[t].sum = tree[t].val;
    for(int i = 0; i < K; i ++) tree[t].Max[i] = tree[t].Min[i] = tree[t].d[i];
    build(tree[t].l,l,mid - 1,(k + 1) % K);   //mid 点就是 t 点，已经被加入了
    build(tree[t].r,mid + 1,r,(k + 1) % K);
    update(t);
}
inline void rebuild(int &t,int k){    //重构子树
    tmp[num = 1] = ++cnt;
    tree[cnt].size = 1;
    for(int i = 0; i < K ; i ++) tree[cnt].d[i] = p[i];
    tree[cnt].val = tree[cnt].sum = A;
    erase(t);
    build(t,1,num,k);
}
```

```
void insert(int& t,int k){
    if(!t){
        tree[t = ++cnt].size = 1;
        tree[t].val = tree[t].sum = A;
        for(int i = 0 ; i < K ; i ++){
            tree[t].Max[i] = tree[t].Min[i] = tree[t].d[i] = p[i];
        }
        return;
    }
    if(p[k] < tree[t].d[k]){
        if(tree[tree[t].l].size > tree[t].size * alpha) rebuild(t,k);
        else insert(tree[t].l,(k + 1) % K);
    }else{
        if(tree[tree[t].r].size > tree[t].size * alpha) rebuild(t,k);
        else insert(tree[t].r,(k + 1) % K);
    }
    update(t);
}
inline bool check_range(int t){  //检查这个区域是否被两个 q 完全包围
    if(!t) return 0;
    for(int i = 0 ; i < K ; i ++){
        if(q[i][0] > tree[t].Min[i] || q[i][1] < tree[t].Max[i]) return 0;
    }
    return 1;
}
inline bool check_point(int t){   //检查点是否在子矩阵内
    if(!t) return 0;
    for(int i = 0 ; i < K ; i ++){
        if(tree[t].d[i] < q[i][0] || tree[t].d[i] > q[i][1]) return 0;
    }
    return 1;
}
inline bool check(int t){ //检查是否两个区间是否有相交部分
    if(!t) return false;
    for(int i = 0 ; i < K ; i ++){
        if(q[i][1] < tree[t].Min[i] || q[i][0] > tree[t].Max[i]) return false;
    }
    return true;
}
int ans;
inline void query(int t){
    if(check_range(t)){
        ans += tree[t].sum;
        return;
    }
    if(check_point(t)) ans += tree[t].val;
    if(check(tree[t].l)) query(tree[t].l);
    if(check(tree[t].r)) query(tree[t].r);
}
int main(){
    N = read(); K = 2;   //默认为平面 (2 维)
    int root = 0;
    while(1){
```

```
        int op = read();
        if(op == 1){
            for(int i = 0 ; i < K ; i ++) p[i] = read() ^ ans;
            A = read() ^ ans;
            insert(root,0);
        }else if(op == 2){
            for(int i = 0; i <= 1; i ++){
                for(int j = 0 ; j < K ;j ++) q[j][i] = read() ^ ans;
            }
            ans = 0; query(root);
            Pri(ans);
        }else{
            break;
        }
    }
    return 0;
}
```

4 字符串

4.1 Hash

4.1.1 字符串 Hash

```
//字符串 Hash, 将字符串 s 变为一个数字
//预处理
p[0] = 1; // 131^0
for (int i = 1; i <= n; i++) {
    f[i] = f[i-1] * 131 + (s[i]-'a'+1); // hash of 1~i
    p[i] = p[i-1] * 131; // 131^i
}
//使用
f[r1] - f[l1-1] * p[r1-l1+1]  // hash of l1~r1
f[r2] - f[l2-1] * p[r2-l2+1]) // hash of l2~r2

/* 扩展
字符串 hash 其实和进制进位有一些相似, 我们想到了这一点就可以线性的时间正着求前缀字符串的 hash
↪ 值和前缀字符串的倒转的 hash 值

当我们求正序前缀的 hash 的时候, 每一次操作就相当于再字符串的最前面插入一个值, 所以我们要首先
↪ 处理出每一位的指数级 ans, 然后 f[i] = f[i - 1] + str[i] * ans;

当我们求逆序前缀的 hash 值的时候, 每一次操作就相当于在字符串的最后插入一个值, 整个字符串就要
↪ 左移一位然后把这个位塞进去 也就是 f [i]= f[i - 1] * tmp + str[i];
*/
//例题 判断一个字符串的每一个前缀是否相等
ULL ans = 1;
ULL f1[maxn],f2[maxn];
char str[maxn];
for(int i = 1 ; i <= l; i ++){
    if (str[i] >= '0' && str[i] <= '9')   str[i] = str[i] - '0';
    else if (str[i] >= 'a' && str[i] <= 'z') str[i] = str[i] - 'a' + 10;
    else str[i] = str[i] - 'A' + 36;
    f1[i] = f1[i - 1] * 131 + str[i];
```

```
    f2[i] = (f2[i - 1] + str[i] * ans);
    ans *= 131;
    if (f1[i] == f2[i]){
        puts("i is equal");
    }
}
```

4.1.2 图上 Hash

```
//图上 Hash
//当一个点和他连到的点组成的集合和另一个点和他连到的点组成的集合完全相同的话，那么他们的 Hash
↪ 值相同
ULL Hash[maxn],id[maxn];
for(int i = 1; i <= N ; i ++ ) Hash[i] = id[i] = id[i - 1] * 3;
for(int i = 1; i <= M ; i ++ ){
    int u,v; Sca2(u,v);
    add(u,v); add(v,u);
    Hash[u] += id[v]; Hash[v] += id[u];
}
```

4.2 KMP

4.2.1 KMP

```
//KMP 算法模板：输出 s2 在 s1 内出现的所有位置的起始点，再输出 next 数组
//next[i] 表示前 i 个字符前后最长匹配
char s1[maxn],s2[maxn];
int Next[maxn];
void kmp_pre(char x[],int m,int nxt[]){     //计算 next 数组
    int j = 0;
    nxt[1] = 0;
    for(int i = 2; i <= m; i ++){
        while(j && x[i] != x[j + 1]) j = nxt[j];
        if(x[j + 1] == x[i]) j ++;
        nxt[i] = j;
    }
}
void KMP(char x[],int m,char y[],int n){     //1 - n 开始计算
    kmp_pre(x,m,Next);
    int j = 0;
    for(int i = 1 ; i <= n; i ++){
        while(j > 0 && x[j + 1] != y[i]) j = Next[j];
        if(x[j + 1] == y[i]) j ++;
        if(j == m){
            Pri(i - m + 1);                 //如果是计数可以开个计数器然后 ++
            j = Next[j];
        }
    }
}
int main(){
    scanf("%s%s",s1 + 1,s2 + 1);
    int l1 = strlen(s1 + 1),l2 = strlen(s2 + 1);
    KMP(s2,l2,s1,l1);
    for(int i = 0 ; i < l2; i ++) printf("%d ",Next[i + 1]);
```

```
    return 0;
}
```

4.2.2 EXKMP

```
/* 拓展 KMP 算法
nxt[]:x 串的每一个后缀与整个串的最长公共前缀，即 x[i .. m - 1] 与 x[0...m - 1] 的最长公共前
↪ 缀
extend[]:y 的每一个后缀与 x 的整个串的最长公共前缀，即 y[i ... n - 1] 与 x[0 .. m - 1] 的最
↪ 长公共前缀
模板：求 str2 的后缀和 str1 的前缀的最长公共前缀
*/
const int maxn = 1e6 + 10;
int nxt[maxn],extend[maxn];
char str1[maxn],str2[maxn];
void pre_EKMP(char x[],int m,int next[]){
    next[0] = m;
    int j = 0;
    while(j + 1 < m && x[j] == x[j + 1]) j ++;
    next[1] = j;
    int k = 1;
    for(int i = 2; i < m ; i ++){
        int p = next[k] + k - 1;
        int L = next[i - k];
        if(i + L < p + 1) next[i] = L;
        else{
            j = max(0,p - i + 1);
            while(i + j < m && x[i + j] == x[j]) j ++;
            next[i] = j;
            k = i;
        }
    }
}
void EKMP(char x[],int m,char y[],int n,int next[],int extend[]){
    pre_EKMP(x,m,next);
    int j = 0;
    while(j < n && j < m && x[j] == y[j]) j++;
    extend[0] = j;
    int k = 0 ;
    for(int i = 1; i < n ; i++){
        int p = extend[k] + k - 1;
        int L = next[i - k];
        if(i + L < p + 1) extend[i] = L;
        else{
            j = max(0,p - i + 1);
            while(i + j < n && j < m && y[i + j] == x[j]) j ++;
            extend[i] = j;
            k = i;
        }
    }
}
int main(){
    while(~scanf("%s%s",str1,str2)){
        int l1 = strlen(str1),l2 = strlen(str2);
```

```
        EKMP(str1,l1,str2,l2,nxt,extend);
        int ans = 0;
        for(int i = 0 ; i < l2 ; i ++) if(extend[i] == l2 - i){
             ans = l2 - i;
             break;
        }
        if(ans) for(int i = 0; i < ans; i ++) putchar(str1[i]);
        if(ans) putchar(' ');
        Pri(ans);
    }
    return 0;
}
```

4.3 Shift-And

```
/*Shift-and 算法
b[i][j] 表示在字符 i 模式串中第 j 个位置出现
ans[i] 表示当前字符串的后缀与模式串的长度为 i 的前缀匹配
每次加入字符的时候；
1. 将 ans 左移一位 2.ans[0] 置 1 3.ans 与 b[str[i]] 进行与运算，str[i] 为当前加入的字符
就更新完成了
*/
//模板 hdu5972
/* 给你 N 位数，接下来有 N 行，第 i 行先输入 n，表示这个数的第 i 位上可以在接下来的 n 个数中
↪  挑选，然后 i 行再输 n 个数。
然后输入需要匹配的母串，让你输出母串中有多少个可行的 N 位子串并输出
 */
char str[maxn];
bitset<1010>b[12],ans;
int main(){
    while(~Sca(N)){
        for(int i = 0 ; i < 10; i ++) b[i].reset();
        for(int i = 0; i < N ; i ++){
            int x; x = read();
            while(x--) b[read()][i] = 1;
        }
        scanf("%s",str); ans.reset();
        LL sum = 0;
        for(int i = 0;str[i]; i ++){
            ans <<= 1; ans[0] = 1;
            ans &= b[str[i] - '0'];
            if(ans[N - 1]){
                char tmp = str[i + 1];
                str[i + 1] = '\0';  printf("%s\n",str + (i - N + 1));
                str[i + 1] = tmp;
            }
        }
    }
    return 0;
}
```

4.4 Manacher

```
//Manacher 算法，O(n) 求解最大回文子串
//Ma 数组为变换后的串，Mp 数组为变换后串的 i 位置为中心最长回文串
const int maxn = 11000010;
char str[maxn];
char Ma[maxn << 1];
int Mp[maxn << 1];
void Manacher(char s[],int len){
    int l = 0;
    Ma[l++] = '$';
    Ma[l++] = '#';
    for(int i = 0 ; i < len ; i ++){
        Ma[l++] = s[i];
        Ma[l++] = '#';
    }
    Ma[l] = 0;
    int mx = 0,id = 0;
    for(int i = 0 ; i < l ; i ++){
        Mp[i] = mx > i?min(Mp[2 * id - i],mx - i):1;
        while(Ma[i + Mp[i]] == Ma[i - Mp[i]]) Mp[i]++;
        if(i + Mp[i] > mx){
            mx = i + Mp[i];
            id = i;
        }
    }
}
int main(){
    scanf("%s",str);
    int len = strlen(str);
    Manacher(str,len);
    int ans = 0;
    for(int i = 0 ; i < 2 * len + 2; i ++) ans = max(ans,Mp[i] - 1);
    Pri(ans);
    return 0;
}
```

4.5 回文树

```
/* 回文树
一个节点表示一个回文串，每个节点之间表示的是本质不同的回文串
len[i] 表示 i 结点表示的回文串的长度
cnt[i] 表示 i 结点表示的回文串的个数，建树的时候 cnt 是不完全的，需要调用 count() 之后才是完
↪ 全的
fail[i] 表示结点 i 失配后跳转的结点，表示结点 i 表示的回文串的最长后缀回文串
next[i][c] 表示结点 i 表示的字符串在两边添加字符 c 变成的回文串的编号
num[i] 表示 i 表示的回文串的最右端点为回文串结尾的回文串个数
last 指向新添加一个字母后形成的最长回文串标识的结点
S[i] 表示第 i 次添加的字符（一开始设 S[0] = -1，也可以是之后不会出现的任意字符）
 */
const int maxn = 2e5 + 10;
const int maxc = 26;
int N,M,K;
struct Pal_T{
```

```
    int next[maxn][maxc];
    int fail[maxn],cnt[maxn],num[maxn],len[maxn],S[maxn];
    int last,n,tot;
    int newnode(int l){
        for(int i = 0 ; i < maxc; i ++) next[tot][i] = 0;
        cnt[tot] = num[tot] = 0;
        len[tot] = l;
        return tot++;
    }
    void init(){
        tot = 0;
        newnode(0); //偶数根节点
        newnode(-1); //奇数根节点
        last = n = 0;
        S[n] = -1; fail[0] = 1;
    }
    int getfail(int x){
        while(S[n - len[x] - 1] != S[n]) x = fail[x];
        return x;
    }
    void add(int c){
        c -= 'a';
        S[++n] = c;
        int cur = getfail(last);
        if(!next[cur][c]){
            int now = newnode(len[cur] + 2);
            fail[now] = next[getfail(fail[cur])][c];
            next[cur][c] = now;
            num[now] = num[fail[now]] + 1;
        }
        last = next[cur][c];
        cnt[last]++;
    }
    void count(){
        for(int i = tot - 1; i >= 0; i --) cnt[fail[i]] += cnt[i];
    }
}PT,PT2;
```

4.6 Trie 树

```
//字典树
//模板：洛谷 P2580 给定 N 个单词，查询 M 个单词在其中是否出现过以及是否第一次出现
//用空间换取时间的静态实现（跑得更快内存更大）
const int maxn = 500010;
int N,M,K;
int nxt[maxn][28];
int val[maxn],cnt;
char str[100];
void insert(char *s){
    int p = 1;
    for(int i = 0; str[i]; i ++){
        int id = str[i] - 'a';
        if(!nxt[p][id]) nxt[p][id] = ++cnt;
        p = nxt[p][id];
```

```
    }
    val[p] = 1;
}
void query(char *s){
    int p = 1;
    for(int i = 0; str[i]; i ++){
        int id = str[i] - 'a';
        if(!nxt[p][id]){
            puts("WRONG");
            return;
        }
        p = nxt[p][id];
    }
    if(val[p] == 0) puts("WRONG");
    else if(val[p] == 1){
        puts("OK"); val[p] = 2;
    }else{
        puts("REPEAT");
    }
}
int main(){
    Sca(N); cnt = 0;
    for(int i = 1; i <= N; i ++){
        scanf("%s",str);
        insert(str);
    }
    Sca(M);
    while(M--){
        scanf("%s",str);
        query(str);
    }
    return 0;
}


//用时间换取空间的动态实现（跑的更慢空间更小）
int N,M,K;
struct node{
    node* nxt[26];
    int val;
}*root;
char str[100];
node* newnode(){
    node *p = new node();
    p->val = 0;
    for(int i = 0 ; i < 26; i ++) p->nxt[i] = NULL;
    return p;
}
void insert(char *s){
    node* p = root;
    for(int i = 0; str[i]; i ++){
        int id = str[i] - 'a';
        if(p->nxt[id] == NULL) p->nxt[id] = newnode();
```

```
        p = p->nxt[id];
    }
    p->val = 1;
}
void query(char *s){
    node* p = root;
    for(int i = 0; str[i]; i ++){
        int id = str[i] - 'a';
        if(p->nxt[id] == NULL){
            puts("WRONG");
            return;
        }
        p = p->nxt[id];
    }
    if(p->val == 0) puts("WRONG");
    else if(p->val == 1){
        puts("OK"); p->val = 2;
    }else{
        puts("REPEAT");
    }
}
void deal(node *p){
    for(int i = 0 ; i < 26; i ++) if(p->nxt[i]) deal(p->nxt[i]);
    free(p);
}
```

4.7 AC 自动机

```
//AC 自动机，在文本串里寻找有多少个模式串出现过
const int maxn = 1e6 + 10;
const int INF = 0x3f3f3f3f;
int N,M,K;
struct AC{
    int next[maxn][26],tot,fail[maxn],end[maxn],root;
    int newnode(){
        for(int i = 0 ; i < 26; i ++) next[tot][i] = -1;
        end[tot++] = 0;
        return tot - 1;
    }
    void init(){
        tot = 0;
        root = newnode();
    }
    void insert(char *str){
        int p = root;
        for(int i = 0 ; str[i] ; i ++){
            int id = str[i] - 'a';
            if(next[p][id] == -1) next[p][id] = newnode();
            p = next[p][id];
        }
        end[p]++;
    }
    void Build(){
        queue<int>Q;
```

```
        fail[root] = root;
        for(int i = 0 ; i < 26; i ++){
            if(~next[root][i]){
                fail[next[root][i]] = root;
                Q.push(next[root][i]);
            }else{
                next[root][i] = root;
            }
        }
        while(!Q.empty()){
            int u = Q.front(); Q.pop();
            for(int i = 0 ; i < 26; i ++){
                if(~next[u][i]){
                    fail[next[u][i]] = next[fail[u]][i];
                    Q.push(next[u][i]);
                }else next[u][i] = next[fail[u]][i];
            }
        }
    }
    int query(char *str){
        int p = root,ans = 0;
        for(int i = 0; str[i]; i ++){
            int id = str[i] - 'a';
            p = next[p][id];
            int tmp = p;
            while(~end[tmp]){
                ans += end[tmp];
                end[tmp] = -1;
                tmp = fail[tmp];
            }
        }
        return ans;
    }
}ac;
char str[maxn];
int main(){
    Sca(N);
    ac.init();
    for(int i = 1; i <= N ; i ++){
        scanf("%s",str);
        ac.insert(str);
    }
    ac.Build();
    scanf("%s",str);
    Pri(ac.query(str));
    return 0;
}
```

4.8 序列自动机

```
//序列自动机
//nxt[i][j] 表示 i 后面第一个 j 出现的位置 (不包括 i)
//模板：给一个长串，查询 N 个字符串是否为长串的子序列
//做法：直接跳 next 即可
```

```
const int maxn = 1e5 + 10;
char str[maxn],s[maxn];
int nxt[maxn][26]; //表示 i 后面第一个 j 出现的位置 (不包括 i)
int main(){
    scanf("%s",str + 1);
    int l = strlen(str + 1);
    for(int i = 0 ; i < 26; i ++) nxt[l][i] = -1; //后面不存在则为-1
    for(int i = l ; i >= 1; i --){   //构造序列自动机
        for(int j = 0; j < 26; j ++) nxt[i - 1][j] = nxt[i][j];
        nxt[i - 1][str[i] - 'a'] = i;
    }
    scanf("%d",&N);
    while(N--){
        scanf("%s",s);
        int now = 0;
        for(int i = 0 ;s[i] && ~now; i ++) now = nxt[now][s[i] - 'a'];
        if(~now) puts("YES");
        else puts("NO");
    }
    return 0;
}
//求子序列个数 f[i] 表示以 i 起始的子序列个数
int dfs(int x) //main函数调用：dfs(0);
{
    if(f[x]) return f[x];
    for(int i=1;i<=a;i++)
        if(nxt[x][i]) f[x]+=dfs(nxt[x][i]);
    return ++f[x];
}
//求两串的公共子序列个数，两串都构造一下然后跑
LL dfs(LL x,LL y){
    if(f[x][y]) return f[x][y];
    for(LL i=1;i<=a;++i)
        if(nxt1[x][i]&&nxt2[y][i])
            f[x][y]+=Dfs(nxt1[x][i],nxt2[y][i]);
    return ++f[x][y];
}
//求字符串回文子序列的个数：正反都构造一遍然后跑
LL dfs(LL x,LL y){
    if(f[x][y]) return f[x][y];
    for(LL i=1;i<=a;++i)
        if(nxt1[x][i]&&nxt2[y][i]){
            if(nxt1[x][i]+nxt2[y][i]>n+1) continue;
            if(nxt1[x][i]+nxt2[y][i]<n+1) f[x][y]++;
            f[x][y]=(f[x][y]+Dfs(nxt1[x][i],nxt2[y][i]))%mod;
        }
    return ++f[x][y];
}
//求一个 A,B 的最长公共子序列 S，使得 C 是 S 的子序列
//dfs(int x,int y,int z)，表示一匹配到 C 的 z 位
//需要改变 C 的构造方法
for(LL i=1;i<=a;++i) nxt[n][i]=n;
for(LL i=0;i<n;++i){
```

```
    for(LL j=1;j<=a;++j) nxt[i][j]=i;
    nxt[i][c[i+1]]=i+1;
}
```

4.9 fail 树

```
//fail 树：将 AC 自动机上的所有 fail 指针反向，就形成了一颗 fail 树
//fail 树所有的父节点是子节点的在所有单词中的最长后缀
//所以同一个 AC 自动机中，串 x 包含被其他所有串包含的次数就是 x 尾结点在 fail 树上的子树大小
//若要求 x 被 y 包含的次数，则仅将 y 到根节点的所有结点 val 值 ++ ，求 x 的子树大小即可
//模板 ： 洛谷 P3966 求出每个单词在所有单词中出现的次数
const int maxm = 5e5 + 10;
const int INF = 0x3f3f3f3f;
const int mod = 1e9 + 7;
int N,M,K;
int head[maxm],tot;
struct Edge{
    int to,next;
}edge[maxm];
void init(int cnt){
    for(int i = 0 ; i <= cnt ; i ++) head[i] = -1;
    tot = 0;
}
void add(int u,int v){
    edge[tot].to = v;
    edge[tot].next = head[u];
    head[u] = tot++;
}
int nxt[maxm][26],fail[maxm],cnt[maxm];
int ttt,root;
int newnode(){
    for(int i = 0 ; i < 26; i ++) nxt[ttt][i] = -1;
    cnt[ttt] = 0;
    return ttt++;
}
void INIT(){
    ttt = 0;
    root = newnode();
}
int insert(char *str){
    int p = root;
    for(int i = 0; str[i]; i ++){
        int id = str[i] - 'a';
        if(nxt[p][id] == -1) nxt[p][id] = newnode();
        p = nxt[p][id];
        cnt[p]++;
    }
    return p;
}
void Build(){
    queue<int>Q;
    for(int i = 0; i < 26; i ++){
        if(nxt[root][i] == -1){
            nxt[root][i] = 0;
```

```
        }else{
            fail[nxt[root][i]] = root;
            Q.push(nxt[root][i]);
            add(root,nxt[root][i]);
        }
    }
    while(!Q.empty()){
        int u = Q.front(); Q.pop();
        for(int i = 0 ; i < 26; i ++){
            if(nxt[u][i] == -1){
                nxt[u][i] = nxt[fail[u]][i];
            }else{
                fail[nxt[u][i]] = nxt[fail[u]][i];
                add(nxt[fail[u]][i],nxt[u][i]);
                Q.push(nxt[u][i]);
            }
        }
    }
}
void dfs(int t,int la){
    for(int i = head[t]; ~i ; i = edge[i].next){
        int v = edge[i].to;
        if(v == la) continue;
        dfs(v,t);
        cnt[t] += cnt[v];
    }
}
int Index[maxn];
char str[1000010];
int main(){
    Sca(N); INIT();
    for(int i = 1; i <= N ; i ++){
        scanf("%s",str);
        Index[i] = insert(str);
    }
    init(ttt);
    Build();
    dfs(root,0);
    for(int i = 1; i <= N ; i ++){
        Pri(cnt[Index[i]]);
    }
    return 0;
}
```

4.10 后缀数组

```
/*  后缀数组
(1) 将字符串的所有后缀按照字典序从小到大排序
sa[i] 表示排名第 i 的后缀的下标 , rak[i] 表示下标为 i 的后缀的排名
(2)Height 数组
lcp(x,y)：字符串 x 与字符串 y 的最长公共前缀，在这里指 x 号后缀与与 y 号后缀的最长公共前缀
height[i]：lcp(sa[i],sa[i-1])，即排名为 i 的后缀与排名为 i-1 的后缀的最长公共前缀
H[i]：height[rak[i]]，即 i 号后缀与它前一名的后缀的最长公共前缀
应用：
```

```
1. 两个后缀的最大公共前缀：lcp(x,y)=min(heigh[x-y]), 用 rmq 维护, O(1) 查询
2. 可重叠最长重复子串：Height 数组里的最大值
3. 不可重叠最长重复子串 POJ1743：首先二分答案 x, 对 height 数组进行分组, 保证每一组的
↪ minheight 都 >=x
依次枚举每一组, 记录下最大和最小长度, 多 sa[mx]-sa[mi]>=x 那么可以更新答案
4. 本质不同的子串的数量：枚举每一个后缀, 第 i 个后缀对答案的贡献为 len-sa[i]+1-height[i]
*/
const int maxn = 1e6 + 10;
int N,M,K;
char str[maxn];
int sa[maxn],rak[maxn],tex[maxn],tp[maxn],Height[maxn];
void GetHeight() {
    int j, k = 0;
    for(int i = 1; i <= N; i++){
        if(k) k--;
        int j = sa[rak[i] - 1];
        while(str[i + k] == str[j + k]) k++;
        Height[rak[i]] = k;
    }
}
void Qsort(){
    for(int i = 0; i <= M ; i ++) tex[i] = 0;
    for(int i = 1; i <= N ; i ++) tex[rak[i]]++;
    for(int i = 1; i <= M ; i ++) tex[i] += tex[i - 1];
    for(int i = N; i >= 1 ; i --) sa[tex[rak[tp[i]]]--] = tp[i];
}
void SA(){
    for(int i = 1; i <= N ; i ++) rak[i] = str[i] - '0' + 1,tp[i] = i;
    Qsort();
    for(int w = 1,p = 0; p < N; w <<= 1,M = p){
        p = 0;
        for(int i = 1; i <= w; i ++) tp[++p] = N - w + i;
        for(int i = 1; i <= N ; i ++) if(sa[i] > w) tp[++p] = sa[i] - w;
        Qsort();
        swap(tp,rak);
        rak[sa[1]] = p = 1;
        for(int i = 2; i <= N ; i ++){
            rak[sa[i]] = (tp[sa[i - 1]] == tp[sa[i]] && tp[sa[i - 1] + w] == tp[sa[i] +
            ↪ w])?p:++p;
        }
    }
}
int main(){
    scanf("%s",str + 1);
    N = strlen(str + 1);
    M = 122;
    SA();
    for(int i = 1; i <= N ; i ++){
        printf("%d ",sa[i]);
    }
    return 0;
}
```

4.11 后缀自动机

```
/* SAM 后缀自动机
时空复杂度 O(n)
1. 每个状态对应一个 endpos 的等价类，表示以该点结尾的子串的集合。
2.t0 到任意节点之间的转移连接起来就是字符串的子串，任意子串对应一条 t0 出发的路径
3. 通常一个结点表示的字符串是这个等价类集合中长度最长的字符串，集合中的所有其他字符串都是他的
↪ 后缀
性质 1. 考虑一个 endpos 等价类，将类中的所有子串按长度非递增的顺序排序。每个子串都不会比它前
↪ 一个子串长，与此同时每个子串也是它前一个子串的后缀。换句话说，对于同一等价类的任一两子串，
↪ 较短者为较长者的后缀，且该等价类中的子串长度恰好覆盖整个区间 [x,y] 。

后缀链接 (parent 树)
1. 结点指向结点表示的最短字符串的去掉第一个字符的字符串
例如结点最短字符串表示 abcabc，则指向结点的最长字符串 bcabc
2. 叶子节点都是主链上的节点，主链上的点不一定是叶子结点
3. 在主链上的点，最长的子串都是原串的前缀
4. 一个节点上的子串出现次数是一样的
5. 对应同一状态 v 的所有子串在文本串 T 中的出现次数相同
6.parents 上父亲上的子串出现次数，是儿子上的子串出现次数之和，如果父亲在主链上，就再加一
7. 点 i 上面表示子串的数量为 len[fa[i]]-len[i]。
8. 两节点的最长公共后缀是他们在 parent 树上的 LCA 点表示的最长字符串
可以证明后缀自动机上至多有 2 * n - 1 个结点和 3 * n - 4 条边
*/
//tag: 以下模板不自带初始化，son,fa,len,num 等数组均未初始化，多样例请特别注意
/* 模板 1：求字符串中出现次数不为 1 的子串中出现次数 * 长度最大的值
num[i] 为当前状态点出现次数，len[i] 为当前状态点最大字符串长度
根据 parent 树的性质 6，主链上的结点肯定只出现一次，所以 dfs 之后统计一下儿子的和就可以知道每
↪ 个结点出现的次数
采用桶排序模拟 dfs 的过程可以不必实质建出 parent 树 */
//tag: 如果需要动态维护 num，每次新加点之后对 parent 树的每个祖先都 +1，同时创建 nq 的时候需
↪ 要复制 q 的 num 值，下面有给出
int len[maxn << 1],fa[maxn << 1],son[maxn << 1][maxc];
LL num[maxn << 1];
int size,last;
void Init(){
    size = last = 1;
}
void insert(char c){
    int s = c - 'a';
    int p = last,np = ++size;last = np; num[np] = 1;
    //cout << np << endl;
    len[np] = len[p] + 1;
    for(;p && !son[p][s]; p = fa[p]) son[p][s] = np;
    if(!p) fa[np] = 1;
    else{
        int q = son[p][s];
        if(len[p] + 1 == len[q]) fa[np] = q;
        else{
            int nq = ++size; len[nq] = len[p] + 1;
            memcpy(son[nq],son[q],sizeof(son[q]));
            fa[nq] = fa[q]; fa[q] = fa[np] = nq;
            //num[nq] = num[q] if 需要动态维护 num
            for(;son[p][s] == q && p;p = fa[p]) son[p][s] = nq;
```

```
        }
    }
}
void insert(char *s){
    Init();
    for(int i = 0; s[i] ; i ++) insert(s[i]);
}
char str[maxn];
int A[maxn << 1],tmp[maxn << 1];
void Qsort(){
    //排出一个按照 len 从小到大的结点序列
    //本质上是模拟 dfs 的加,len 小的一定在 len 大的之前遍历
    for(int i = 1; i <= size; i ++) tmp[len[i]]++;
    for(int i = 1; i <= size; i ++) tmp[i] += tmp[i - 1];
    for(int i = 1; i <= size; i ++) A[tmp[len[i]]--] = i;
}
int main(){
    scanf("%s",str);
    insert(str);
    Qsort();
    for(int i = size; i >= 1; i --) num[fa[A[i]]] += num[A[i]];
    LL ans = 0;
    for(int i = 2; i <= size; i ++) if(num[i] != 1)ans = max(ans,len[i] * num[i]);
    Prl(ans);
    return 0;
}

/* 应用，输出长度为 1-n 的子串出现最多次数的子串的次数
做法：利用模板 1 求出每个节点的出现次数，然后每个节点 i 将 len[min]-len[max] 的答案更新
↪ num[i]
模板：在模板 1 的基础上添加如下代码 */
for(int i = 2; i <= size; i ++) Max[len[i]] = max(Max[len[i]],num[i]);
for(int i = N; i >= 1 ; i --) Max[i] = max(Max[i],Max[i + 1]);
for(int i = 1; i <= N ; i ++) Prl(Max[i]);

/* 应用：求 s,t 两串的最长公共子串 (On)
做法：对 S 建 SAM 然后 t 开始跑，跑到失配点 i 就跳转到 fa[i]*/
scanf("%s%s",s,t);
insert(s);
int ans = 0,tmp = 0;
int p = 1;
for(int i = 0;t[i]; i ++){
    int v = t[i] - 'a';
    while(p != 1 && !son[p][v]) p = fa[p],tmp = len[p];
    if(son[p][v]){
        p = son[p][v]; tmp++;
    }
    ans = max(ans,tmp);
}
Pri(ans);
/* 应用：多个字符串的最长公共子串
做法：选取一个字符串建 SAM，其它字符串如上跑一遍
每个状态对应每个字符串的答案为 s1,s2,s3..sn，则每个状态的最终答案为 min(s1,s2..sn)，取最大
↪ 即可
```

```
模板：dp[i] 表示当前结尾的字符匹配上之后的最长匹配长度，显然一个结点匹配上之后他的父亲也匹配
↪ 上
所以将 parent 树从叶子到跟再更行一遍 */
int dp[maxn << 1],Max[maxn << 1];
int A[maxn << 1],tmp[maxn << 1];
void ins(char* str){
    for(int i = 0 ; i <= size; i ++) dp[i] = 0;
    int l = 0,p = 1;
    for(int i = 0;str[i]; i ++){
        int v = str[i] - 'a';
        while(p != 1 && !son[p][v]) p = fa[p],l = len[p];
        if(son[p][v]){
            p = son[p][v]; l++;
            dp[p] = max(dp[p],l);
        }
    }
    for(int i = size; i >= 2; i --){
        dp[fa[A[i]]] = max(dp[fa[A[i]]],min(len[fa[A[i]]],dp[A[i]]));
    }
    for(int i = 1; i <= size; i ++) Max[i] = min(Max[i],dp[i]);
}
int main(){
    scanf("%s",s); int cnt = 1;
    while(~scanf("%s",t[cnt])) cnt++;
    cnt--;
    insert(s);
    for(int i = 1; i <= size; i ++) tmp[len[i]]++;
    for(int i = 1; i <= size; i ++) tmp[i] += tmp[i - 1];
    for(int i = 1; i <= size; i ++) A[tmp[len[i]]--] = i;
    for(int i = 1; i <= size; i ++) Max[i] = len[i];
    for(int i = 1; i <= cnt ; i ++) ins(t[i]);
    int ans = 0;
    for(int i = 2; i <= size; i ++) ans = max(ans,Max[i]);
    Pri(ans);
}
/* 最小表示法，长度为 N 的循环字符串中找到字典序最小的字符串
模板：将原串倍增一遍之后 SAM 直接寻找长度为 N 的字典序最小的路径
数值大小不定，采用 map 建图，用时间换空间
*/
const int maxn = 6e5 + 10;
int N,M,K;
int len[maxn << 1],fa[maxn << 1];
map<int,int>son[maxn << 1];
int size,last;
void Init(){
    size = last = 1;
}
inline void insert(int c){
    int p = last,np = ++size; last = np;
    len[np] = len[p] + 1;
    for(;p && !son[p].count(c); p = fa[p]) son[p][c] = np;
    if(!p) fa[np] = 1;
    else{
```

```
        int q = son[p][c];
        if(len[p] + 1 == len[q]) fa[np] = q;
        else{
            int nq = ++size; len[nq] = len[p] + 1;
            son[nq] = son[q];
            fa[nq] = fa[q]; fa[q] = fa[np] = nq;
            for(;son[p][c] == q && p; p = fa[p]) son[p][c] = nq;
        }
    }
}
int a[maxn];
int main(){
    Sca(N); Init();
    for(int i = 1; i <= N ; i ++) insert(a[i] = read());
    for(int i = 1; i <= N ; i ++) insert(a[i]);
    int t = 1;
    for(int i = 1; i <= N ; i ++){
        printf("%d ",son[t].begin()->first);
        t = son[t].begin()->second;
    }
    return 0;
}
/* 应用：检查字符串是否出现在文本中
做法：对文本建后缀自动机，能跑完模式串且不出现 NULL 结点就是出现在文本中了 */

/* 应用：不同子串个数
做法：相当于图上从原点起有多少不同的路径，因为这是一个 DAG 图，所以直接 DP 一下就可以了
dp[i] 表示从 i 点出发的路径数量
初始 dp[i] = 1，然后从终点向起点累加，即 dp[i] = 1 + Σdp[j](j 为 i 的儿子)
最终不同子串的个数就是 dp[1] - 1，因为需要减掉一个空串 */

/* 模板：求第 k 小子串，选项 0 为不同位置的相同子串算一个，1 为不同位置的相同子串算不同
做法：选项 0：如同上述求不同字串个数的做法求出每个节点出发的路径数量，然后就如权值线段树找 k
↪ 大一般找第 k 大的路径
选项 1：利用模板 1 求出不同结点表示的串的个数 num[i],dp[i] 的初始化从 1 变为 num[i]，即从 i
↪ 到 i 的路径变为了 num[i] 条，然后同上
*/
const int maxn = 5e5 + 10;
char str[maxn];
int tmp[maxn << 1],A[maxn << 1];
int sum[maxn << 1];
void dfs(int t,int k){  //找出 t 出发的第 k 大路径
    if(k <= num[t]) return;
    k -= num[t];   //减去当前节点表示的路径数
    for(int i = 0 ; i < 26; i ++){
        if(!son[t][i]) continue;
        int v = son[t][i];
        if(sum[v] >= k){
            printf("%c",i + 'a');
            dfs(v,k);
            return;
        }
        k -= sum[v];
```

```
    }
}
int main(){
    scanf("%s",str);
    insert(str);
    for(int i = 1; i <= size; i ++) tmp[len[i]]++;
    for(int i = 1; i <= size; i ++) tmp[i] += tmp[i - 1];
    for(int i = 1; i <= size; i ++) A[tmp[len[i]]--] = i;
    for(int i = size; i >= 1; i --) num[fa[A[i]]] += num[A[i]];
    int op = read(),k = read();
    for(int i = 1; i <= size; i ++) sum[i] = op?num[i]:num[i] = 1;
    sum[1] = num[1] = 0;
    for(int i = size; i >= 1; i --){
        for(int j = 0 ; j < 26; j ++){
            if(son[A[i]][j]) sum[A[i]] += sum[son[A[i]][j]];
        }
    }
    if(sum[1] < k) puts("-1");
    else dfs(1,k);
    return 0;
}
/* 求区间不同字串个数
做法：一个很关键的性质：SAM 每次插入字符后增加的不同字串的个数为 len[last] - len[fa[last]]
所以对于每个左端点跑一边 SAM，然后插入的时候记录答案，即可获得 dp[i][j] 表示区间不同字符串的
↪ 个数 */
int dp[maxn][maxn];
int main(){
    int T = read();
    while(T--){
        scanf("%s",str + 1);
        for(int i = 1; str[i]; i ++){
            for(int i = 0; i <= size; i ++){
                fa[i] = len[i] = 0;
                for(int j = 0 ; j < 26; j ++) son[i][j] = 0;
            }
            Init();
            for(int j = i ; str[j] ; j ++){
                insert(str[j]);
                dp[i][j] = dp[i][j - 1] + len[last] - len[fa[last]];
            }
        }
        Sca(M);
        while(M--){
            int l = read(),r = read();
            Pri(dp[l][r]);
        }
    }
    return 0;
}
```

5 动态规划

5.1 悬线法 dp

```
//悬线法 dp，用于处理矩阵相关的 dp，例如求解最大的满足某个限制的长方形（正方形）
//用 Left Right 两个数组记录这个位置最左端和最右端的满足题目限制的条件。
//用 up 数组记录 这个位置在满足了左右两端条件之后可向上扩充的最大长度
//然后使用 up * (Right - Left + 1) 的方法更新答案即可。

//模板：ZJOI2007 棋盘制作 寻找最大的满足 01 交错的长方形和正方形
int Left[maxn][maxn],Right[maxn][maxn],up[maxn][maxn];
int MAP[maxn][maxn];
int main()
{
    Sca2(N,M);
    for(int i = 1; i <= N ; i ++){
        for(int j = 1; j <= M ; j ++){
            Sca(MAP[i][j]);
            Left[i][j] = Right[i][j] = j;
            up[i][j] = 1;
        }
    }
    for(int i = 1; i <= N ; i ++){
        for(int j = 2; j <= M ; j ++){
            if(MAP[i][j] != MAP[i][j - 1]){
                Left[i][j] = Left[i][j - 1];
            }
        }
        for(int j = M - 1; j >= 1; j --){
            if(MAP[i][j] != MAP[i][j + 1]){
                Right[i][j] = Right[i][j + 1];
            }
        }
    }
    int ans1 = 0,ans2 = 0;
    for(int i = 1; i <= N ; i ++){
        for(int j = 1; j <= M ; j ++){
            if(i > 1 && MAP[i][j] != MAP[i - 1][j]){
                Left[i][j] = max(Left[i][j],Left[i - 1][j]);
                Right[i][j] = min(Right[i][j],Right[i - 1][j]);
                up[i][j] = up[i - 1][j] + 1;
            }
            int a = Right[i][j] - Left[i][j] + 1;
            int b = min(a,up[i][j]);
            ans1 = max(ans1,b * b);
            ans2 = max(ans2,a * up[i][j]);
        }
    }
    Pri(ans1);
    Pri(ans2);
    return 0;
}
```

5.2 斜率优化 dp

```
//斜率优化 dp
/* 方法 1.
将 dp 转移方程化成斜率单调递增 (减) 的 y = kx + b 形式
其中 b 是需要被转移的 dp[i],x 是只与 i 有关的单调递增的东西, y 是由只与 i 有关或只与 j 有关
↪ 的式子
方法 2(推荐).
假设在更新 i 的时候, 对于 i 存在 x > y 且 x 比 y 更优
即 dp[x] + fun(x,i) < dp[y] + fun(y,i) 假设求最小值
化为 (dp[x] - dp[y]) / (fun(x) - fun(y)) < fun(i)
则 fun(i) 为斜率, 单调队列维护即可。
*/
/* 关于最大 (小) 值
求最大值: 上凸包, 斜率单调递减
求最小值: 下凸包, 斜率单调递增
上下凸包在单调队列判断的时候有点区别
*/

/* 求 dp[i] 最小值, 斜率单调递增 (下凸包)
方法 1.
最终状态转移方程 dp[i]+d[i]*sum[j]=dp[j]+pre[j]+d[i]*sum[i-1]-pre[i-1]+c[i]
其中 b 为 dp[i],k 为 d[i](单调递增),x 为 sum[j],y 为 f[j] + pre[j]
其中 d[i]*sum[i-1]-pre[i-1]+c[i] 和 j 无关, 算 dp[i] 的时候可以直接算上
方法 2.
设存在 x,y 对于 i 使得 x > y 且 x 更优
dp[x]+pre[x]-d[i]*sum[x]<dp[y]+pre[y]-d[i]*sum[y]
->(dp[x]+pre[x]-dp[y]-pre[y]) / (sum[x]-sum[y]) < d[i]
*/
//模板是方法 1
const int maxn = 1e6 + 10;
const int INF = 0x3f3f3f3f;
const int mod = 1e9 + 7;
int N,M;
struct Node{
    LL d,w,C;
}node[maxn];
LL dp[maxn],sum[maxn],pre[maxn];
int Q[maxn];
inline double Y(int i){return dp[i] + pre[i];}
inline double X(int i){return sum[i];}
inline double K(int i,int j){return (Y(j) - Y(i)) / (X(j) - X(i));}
int main(){
    Sca(N);
    for(int i = 1; i <= N ; i ++ ){
        scanf("%lld%lld%lld",&node[i].d,&node[i].w,&node[i].C);
        sum[i] = sum[i - 1] + node[i].w;
        pre[i] = pre[i - 1] + node[i].w * node[i].d;
    }
    int tail = 0,head = 1;  //单调队列有没有初始数字要看题目决定
    for(int i = 1; i <= N ; i ++){
        dp[i] = node[i].d * sum[i - 1] - pre[i - 1] + node[i].C;
        while(tail > head && K(Q[head],Q[head + 1]) < node[i].d) head++; //这行基本一样
```

```
        int j = Q[head]; dp[i] = min(dp[j] + pre[j] + (sum[i - 1] - sum[j]) * node[i].d -
        ↪ pre[i - 1] + node[i].C,dp[i]);
        while(tail > head && K(Q[tail - 1],Q[tail]) > K(Q[tail],i)) tail--;  //这行基本一
        ↪ 样
        Q[++tail] = i;
    }
    Prl(dp[N]);
    return 0;
}

/* 依然是下凸包，模板是方法 2
dp[i] = dp[j] + w[j + 1] * h[i];
方法 2. 存在 x > y 且 x 优于 y
dp[x] + w[x + 1] * h[i] < dp[y] + w[y + 1] * h[i];
(dp[x] - dp[y]) / (w[y + 1] - w[x + 1]) < h[i]
不等式左边为 slope，右边为 K
 */
const int maxn = 5e4 + 10;
const int INF = 0x3f3f3f3f;
int N,M;
int Q[maxn];
struct node{
    LL w,h;
}a[maxn];
LL dp[maxn];
double slope(int i,int j){return (dp[i] - dp[j]) / (a[j + 1].w - a[i + 1].w);}
double K(int i){return a[i].h;}
bool cmp(node a,node b){
    if(a.w == b.w) return a.h > b.h;
    return a.w > b.w;
}
int main(){
    Sca(N);
    for(int i = 1; i <= N ; i ++) scanf("%lld%lld",&a[i].h,&a[i].w);
    sort(a + 1,a + 1 + N,cmp); int cnt = 1;
    for(int i = 2; i <= N ; i ++){
        if(a[i].h > a[cnt].h) a[++cnt] = a[i];
    }
    N = cnt;
    int tail = 1,head = 1;
    for(int i = 1; i <= N ; i ++){
        while(tail > head && slope(Q[head],Q[head + 1]) <= K(i)) head++;
        int j = Q[head]; dp[i] = dp[j] + a[j + 1].w * a[i].h;
        while(tail > head && slope(Q[tail - 1],Q[tail]) > slope(Q[tail],i)) tail--;
        Q[++tail] = i;
    }
    Prl(dp[N]);
    return 0;
}
/* 上凸包 (求最大值)
不等式为 dp[i] = dp[j] + a * (sum[i] - sum[j])² + b * (sum[i] - sum[j]) + c (a < 0)
方法 1.
dp[i] + 2 * a * sum[i] * sum[j] = dp[j] + a * sum[i]² + a * sum[j]² + b * sum[i] - b *
↪ sum[j] + c
```

```
斜率为单调递减的 2 * a * sum[i]
方法 2.
(dp[x] + a * sum[x]² - b * sum[x] - dp[y] - a * sum[y]² + b * sum[y]) / (sum[x] - sum[y])
↪  < 2 * a * sum[i]
不等式左边为 slope(x,y)
*/
//方法 1 模板
const int maxn = 1e6 + 10;
int N,M;
LL sum[maxn],x[maxn],dp[maxn];
LL a,b,c;
int Q[maxn];
inline double X(int i){return sum[i];}
inline double Y(int j){return dp[j] + a * sum[j] * sum[j] - b * sum[j];}
inline double K(int i,int j){return (Y(i) - Y(j)) / (X(i) - X(j));}
int main(){
    Sca(N); scanf("%lld%lld%lld",&a,&b,&c);
    for(int i = 1; i <= N ; i ++){
        Scl(x[i]); sum[i] = sum[i - 1] + x[i];
    }
    int head = 1,tail = 1;
    for(int i = 1; i <= N  ; i++){
        while(tail > head && K(Q[head],Q[head + 1]) >= 2 * a * sum[i]) head++;
        int j = Q[head]; dp[i] = Y(j) - 2 * a * sum[i] * sum[j] + a * sum[i] * sum[i] + b
        ↪  * sum[i] + c;
        while(tail > head && K(Q[tail - 1],Q[tail]) < K (Q[tail],i)) tail--;
        Q[++tail] = i;
    }
    Prl(dp[N]);
    return 0;
}
//方法 2 模板
const int maxn = 1e6 + 10;
int N,M;
LL sum[maxn],x[maxn],dp[maxn];
LL a,b,c;
int Q[maxn];
inline double K(int i){return 2 * a * sum[i];}
inline double slope(int i,int j){return (dp[i] + a * sum[i] * sum[i] - b * sum[i] - dp[j]
↪  - a * sum[j] * sum[j] + b * sum[j]) / (sum[i] - sum[j]);}
int main(){
    Sca(N); scanf("%lld%lld%lld",&a,&b,&c);
    for(int i = 1; i <= N ; i ++){
        Scl(x[i]); sum[i] = sum[i - 1] + x[i];
    }
    int head = 1,tail = 1;
    for(int i = 1; i <= N  ; i++){
        while(tail > head && slope(Q[head],Q[head + 1]) > K(i)) head++;
        int j = Q[head]; dp[i] = dp[j] + a * (sum[i] - sum[j]) * (sum[i] - sum[j]) + b *
        ↪  (sum[i] - sum[j]) + c;
        while(tail > head && slope(Q[tail - 1],Q[tail]) < slope(Q[tail],i)) tail--;
        Q[++tail] = i;
    }
```

```
    Prl(dp[N]);
    return 0;
}
```

5.3 数位 dp

```
//数位 dp, 最普遍的题目：求 A 到 B 中满足若干条件的数有多少个
//模板：条件是相邻数字的差至少大于 2
const int maxn = 12;
int N,M,K,cnt;
int str[maxn];
int dp[maxn][maxn];   //i 位置的 j 数
int dfs(int pos,int num,int zero,int limit){
        if(pos == 0) return (zero ^ 1);
        if(~dp[pos][num] && !zero && !limit) return dp[pos][num];
        int top = limit?str[pos - 1]:9;
        int ans = 0;
        for(int i = 0 ; i <= top; i ++){
                if(abs(i - num) <= 1 && !zero) continue;
                ans += dfs(pos - 1,i,zero && !i,limit && str[pos - 1] == i);
        }
        if(!zero && !limit) dp[pos][num] = ans;
        return ans;
}
int solve(int x){
        if(!x) return 0;
        Mem(dp,-1);
        cnt = 0;
        while(x){
                str[cnt++] = x % 10;
                x /= 10;
        }
        str[cnt] = 0;
        int ans = dfs(cnt,0,1,1);
        return ans;
}
int main(){
    int A = read(), B = read();
    Pri(solve(B) - solve(A - 1));
    return 0;
}
```

5.4 错排公式

```
//n 封信对应 n 个信封，求恰好全部装错了信封的方案数
//dp[i] = (i-1) * (dp[i-1] + dp[i-2])
int N,M,K;
LL dp[30];
int main(){
        dp[1] = 0,dp[2] = 1;
        for(int i = 3; i <= 20; i ++) dp[i] = (i - 1) * (dp[i - 1] + dp[i - 2]);
        while(~Sca(N)) Prl(dp[N]);
        return 0;
}
```

6 数论

6.1 gcd

```
/*
1.lcm(S/a, S/b) = S/gcd(a, b)
2.gcd(a,b) = gcd(a,a - b) => gcd(a1,a2,a3...an) = gcd(a1,a2 - a1,a3 - a2,a4 - a3...an -
↪  an - 1);
3.gcd(a,lcm(b,c)) = lcm(gcd(a,b),gcd(a,c))
4.lcm(a,gcd(b,c)) = gcd(lcm(a,b),lcm(a,c))
5. 在坐标里，将点 (0, 0) 和 (a, b) 连起来，通过整数坐标的点的数目（除了 (0, 0) 一点之外）就是
↪  gcd(a, b)。
6.gcd(a,b)=gcd(b, a mod b)
7.gcd(ab,m)=gcd(a,m) * gcd(b,m)
 */
int gcd(int a,int b) {
    return b==0?a:gcd(b,a%b);
}
```

6.2 exgcd

```
//求出 ax + by = c 的一组可行解
//有解的前提 gcd(a,b)|c , 否则无解
LL exgcd(LL a,LL b,LL &x,LL &y){
    if(a == 0 && b == 0) return -1;
    if(b == 0){
        x = 1;y = 0;
        return a;
    }
    LL d = exgcd(b,a % b,y,x);
    y -= a / b * x;
    return d;
}
```

6.3 逆元

```
//逆元    (a / b) % mod = a * inv(b) % mod
//拓展欧几里得定理求 inv(b)
int mod = 1e9 + 7;
int N;
LL exgcd(LL a,LL b,LL &x,LL &y){
    if(a == 0 && b == 0) return -1;
    if(b == 0){
        x = 1;y = 0;
        return a;
    }
    LL d = exgcd(b,a % b,y,x);
    y -= a / b * x;
    return d;
}
LL inv(LL a,LL n){
    LL x,y;
    LL d = exgcd(a,n,x,y);
    if(d == 1) return (x % n+ n) % n;
    else return -1;
```

```
}
int main(){
    scanf("%lld%lld",&N,&mod);
    Prl(inv(N,mod));
    return 0;
}
//当 mod 为质数时，可以利用费马小定理，b 的逆元为 n ^ (mod - 2) % mod
int mod = 1e9 + 7;
int N;
LL quick_power(LL a,LL b){
    LL ans = 1;
    while(b){
        if(b & 1) ans = ans * a % mod;
        b >>= 1;
        a = a * a % mod;
    }
    return ans;
}
LL inv(LL a,LL n){
    return quick_power(a,n - 2);
}
int main(){
    scanf("%lld%lld",&N,&mod);
    For(i,1,N) Prl(inv(i,mod));
    return 0;
}
//当我们需要求连续一段数在 mod p 下的逆元的时候，我们可以使用逆元的递推公式
//模板：求 1 到 N 的所有逆元
LL mod = 1e9 + 7;
LL N;
LL inv[maxn];
int main(){
    scanf("%lld%lld",&N,&mod);
    inv[1] = 1;
    for(int i = 2; i <= N; i ++) inv[i] = (mod - mod / i) * inv[mod % i] % mod;
    for(int i = 1; i <= N ; i ++) Prl(inv[i]);
    return 0;
}
```

6.4 卡特兰数

递归公式 1

$$f(n) = \sum_{i=0}^{n-1} f(i) * f(n-i-1)$$

递归公式 2

$$f(n) = \frac{f(n-1) * (4 * n - 2)}{n+1}$$

组合公式 1

$$f(n) = \frac{C_{2n}^{n}}{n+1}$$

组合公式 2，重要！重要！重要！

$$f(n) = C_{2n}^{n} - C_{2*n}^{n-1}$$

递推公式

$$f[n] = \sum_{i=0}^{n-1} f[i] * f[n-i-1]$$

```
/* 卡特兰数
1. 前 30 项
[1,1,2,5,14,42,132,429,1430,4862,16796,58786,
 208012,742900,2674440,9694845,35357670,129644790,
 477638700,1767263190,6564120420,24466267020,
 91482563640,343059613650,1289904147324,
 4861946401452,18367353072152,69533550916004,
 263747951750360,1002242216651368,3814986502092304]
2. 在比赛的时候很可能不能证明一个数列是卡特兰数，一般手算几项发现对上了卡特兰数就冲了
*/
//卡特兰数取模模板
LL Catalan[maxn],inv[maxn];
inline void Catalan_init(int n,LL mod){
    inv[1] = 1;
    for(int i = 2; i <= n + 1; i ++){
        inv[i] = (mod - mod / i) * inv[mod % i] % mod;
    }
    Catalan[0] = Catalan[1] = 1;
    for(int i = 2; i <= n ; i ++){
        Catalan[i] = Catalan[i - 1] * (4 * i - 2) % mod * inv[i + 1] % mod;
    }
}
int main(){
    Catalan_init(10000,mod);
    while(~Sca(N)) Prl(Catalan[N]);
    return 0;
}


//卡特兰数大数模板
#include<bits/stdc++.h>
using namespace std;
const int C_maxn = 100 + 10;///项数

int Catalan_Num[C_maxn][1000];///保存卡特兰大数、第二维为具体每个数位的值
int NumLen[C_maxn];///每个大数的数长度、输出的时候需倒序输出

void catalan() //求卡特兰数
{
    int i, j, len, carry, temp;
    Catalan_Num[1][0] = NumLen[1] = 1;
    len = 1;
    for(i = 2; i < 100; i++)
    {
        for(j = 0; j < len; j++) //乘法
        Catalan_Num[i][j] = Catalan_Num[i-1][j]*(4*(i-1)+2);
        carry = 0;
        for(j = 0; j < len; j++) //处理相乘结果
        {
            temp = Catalan_Num[i][j] + carry;
```

```
            Catalan_Num[i][j] = temp % 10;
            carry = temp / 10;
        }
        while(carry) //进位处理
        {
            Catalan_Num[i][len++] = carry % 10;
            carry /= 10;
        }
        carry = 0;
        for(j = len-1; j >= 0; j--) //除法
        {
            temp = carry*10 + Catalan_Num[i][j];
            Catalan_Num[i][j] = temp/(i+1);
            carry = temp%(i+1);
        }
        while(!Catalan_Num[i][len-1]) //高位零处理
        len --;
        NumLen[i] = len;
    }
}
int main(void)
{
    catalan();
    for(int i=1; i<=30; i++){
        for(int j=NumLen[i]-1; j>=0; j--){
            printf("%d", Catalan_Num[i][j]);
        }puts("");
    }
    return 0;
}
```

6.5 Miller-robin

```
//miller_robin 素数测试
LL q_p(LL a,LL b,LL mod){
    LL ans = 1;
    while(b){
        if(b & 1) ans = ans * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return ans;
}
LL Test[10] = {2,3,5,7,11,13,17};
bool Miller_robin(LL p){
    if(p == 1) return 0;
    LL t = p - 1,k = 0;
    while(!(t & 1)) k++,t >>= 1;
    for(int i = 0 ; i < 4; i ++){
        if(p == Test[i]) return true;
        LL a = q_p(Test[i],t,p),nxt = a;
        for(int j = 1; j <= k ; j ++){
            nxt = a * a % p;
            if(nxt == 1 && a != 1 && a != p - 1) return false;
```

```
            a = nxt;
        }
        if(a != 1) return false;
    }
    return true;
}
```

6.6 0/1 分数规划

如一个物品有成本 bi 和价值 ai 的时候, 我们需要一种方案使得取 K 的物品之后价值和成本的比例最高

$$R = \sum \frac{a_i * x_i}{b_i * x_i}$$

形如式子中选定每个 xi 的值为 0/1, 最终使得 R 最大
解: 二分出答案 t 按来代入式子取 check, 每次选取 ai - bi * t 最大的 K 个物品, 如果最终 ai - bi * t >= 0, 说明答案 t 可行

6.7 SG 函数

```
/* SG 函数
必败态：当前状态不管怎么选择，只要对手采取最优策略就败
必胜态：当前状态只要你采取最优策略就胜
1. 当某个局面导向的所有状态都是必胜态时候，当前状态必败
2. 当某个局面导向有一个状态是必败的时候，当前状态必胜
3.SG(x) = mex{SG(y1),SG(y2),SG(y3)...SG(yk)} 其中 y1..yk 是 x 导向的所有状态
当 SG(x) > 0 时说明当前状态必胜，SG(x) < 0 时说明当前状态必败
4. 多个有向图游戏的和 SG 函数值等于他包含的各个子游戏的异或和
SG(G) = SG(G1) ^ SG(G2) ^ SG(G3)..... ^ SG(Gm)
SG 可以通过递推或者记忆化搜索等方式求出
*/
//模板: 总共 n 堆石子，每个人只能抓 2 的幂次 (1,2,4,8...) 个石子，轮流抓取抓不到石头的输
const int maxn = 1010;
const int INF = 0x3f3f3f3f;
const int mod = 1e9 + 7;
int N,M,K;
int sg[maxn];
int step[maxn];  //step 存储可以走的步数，step[0] 存储有多少种走法
bool vis[maxn];
void getsg(){   //直接递推
    sg[0] = 0;
    for(int i = 1; i <= 1000; i ++){
        for(int j = 0 ; j <= i ; j ++) vis[j] = 0;
        for(int j = 1; j <= step[0]; j ++){
            if(step[j] > i) break; //step 需要从小到大排序
            vis[sg[i - step[j]]] = 1;
        }
        for(int j = 0; j <= i ; j ++){
            if(vis[j]) continue;
            sg[i] = j; break;
        }
    }
}

int main(){
```

```
    step[0] = 10; step[1] = 1;
    for(int i = 2; i <= 10 ; i ++) step[i] = step[i - 1] * 2;
    getsg();
    while(~Sca(N)){
        if(sg[N]) puts("Kiki");  //先手胜
        else puts("Cici");   //后手胜
    }
    return 0;
}
```

6.8 中国剩余定理

```
//n 个方程：x % m[i] = a[i] (0<=i<n)
LL china(int n, LL *a, LL *m){
    LL M = 1, ret = 0;
    for(int i = 0; i < n; i ++) M *= m[i];
    for(int i = 0; i < n; i ++){
        LL w = M / m[i];
        ret = (ret + w * inv(w, m[i]) * a[i]) % M;
    }
    return (ret + M) % M;
}
//m1,m2...mn 不保证两两互质的情况
#include<cstdio>
#include<algorithm>
using namespace std;
typedef long long LL;
typedef pair<LL, LL> PLL;
PLL linear(LL A[], LL B[], LL M[], int n) {//求解 A[i]x = B[i] (mod M[i]), 总共 n 个线性
↪  方程组
    LL x = 0, m = 1;
    for(int i = 0; i < n; i ++) {
        LL a = A[i] * m, b = B[i] - A[i]*x, d = gcd(M[i], a);
        if(b % d != 0)  return PLL(0, -1);//答案不存在，返回-1
        LL t = b/d * inv(a/d, M[i]/d)%(M[i]/d);
        x = x + m*t;
        m *= M[i]/d;
    }
    x = (x % m + m ) % m;
    return PLL(x, m);//返回的 x 就是答案，m 是最后的 lcm 值
}
```

6.9 bsgs

```
//baby step,giant step 算法
//给定整数 a,b,p, 其中 a,p 互质，求一个非负整数 x，使得 a ^ x  b (mod p)
LL bsgs(LL a,LL b,LL p){
    map<LL,LL> hash; hash.clear();
    b %= p;
    int t = (int)sqrt(p) + 1;
    for(int j = 0 ; j < t; j ++){
        int val = (LL)b * quick_power(a,j,p) % p;
        hash[val] = j;
    }
```

```
    a = quick_power(a,t,p);
    if(a == 0) return b == 0?1:-1;
    for(int i = 0 ; i <= t; i ++){
        int val = quick_power(a,i,p);
        int j = hash.find(val) == hash.end() ? -1:hash[val];
        if(j >= 0 && i * t - j >= 0) return i * t - j;
    }
    return -1;
}
```

6.10 组合数

```
//组合数 C(n,m)
//预处理 Comb[maxn][maxn] , 复杂度 n ^ 2
#include<cstdio>
const int N = 2000 + 5;
const int MOD = (int)1e9 + 7;
int comb[N][N];//comb[n][m] 就是 C(n,m)
void init(){
    for(int i = 0; i < N; i ++){
        comb[i][0] = comb[i][i] = 1;
        for(int j = 1; j < i; j ++){
            comb[i][j] = comb[i-1][j] + comb[i-1][j-1];
            comb[i][j] %= MOD;
        }
    }
}
int main(){
    init();
}
//O(n) 求 C(n,m)
#include<cstdio>
const int N = 200000 + 5;
const int MOD = (int)1e9 + 7;
int F[N], Finv[N], inv[N];//F 是阶乘, Finv 是逆元的阶乘
void init(){
    inv[1] = 1;
    for(int i = 2; i < N; i ++){
        inv[i] = (MOD - MOD / i) * 1ll * inv[MOD % i] % MOD;
    }
    F[0] = Finv[0] = 1;
    for(int i = 1; i < N; i ++){
        F[i] = F[i-1] * 1ll * i % MOD;
        Finv[i] = Finv[i-1] * 1ll * inv[i] % MOD;
    }
}
int comb(int n, int m){//comb(n, m) 就是 C(n, m)
    if(m < 0 || m > n) return 0;
    return F[n] * 1ll * Finv[n - m] % MOD * Finv[m] % MOD;
}
int main(){
    init();
}
//不预处理
```

```
LL fact(int n, LL p){//n 的阶乘求余 p
    LL ret = 1;
     for (int i = 1; i <= n ; i ++) ret = ret * i % p ;
    return ret ;
}
void ex_gcd(LL a, LL b, LL &x, LL &y, LL &d){
    if (!b) {d = a, x = 1, y = 0;}
    else{
        ex_gcd(b, a % b, y, x, d);
        y -= x * (a / b);
    }
}
LL inv(LL t, LL p){//如果不存在, 返回-1
    LL d, x, y;
    ex_gcd(t, p, x, y, d);
    return d == 1 ? (x % p + p) % p : -1;
}
LL comb(int n, int m, LL p){//C(n, m) % p
    if (m < 0 || m > n) return 0;
    return fact(n, p) * inv(fact(m, p), p) % p * inv(fact(n-m, p), p) % p;
}
```

6.11 卢卡斯定理

```
//卢卡斯定理, 计算 C(n,m) % p, 当 n, m 很大 (1e18) 且 p 比较小 (1e5) 的情况
//C(n, m) % p  =  C(n / p, m / p) * C(n%p, m%p) % p
LL Lucas(LL n, LL m, int p){
        return m ? Lucas(n/p, m/p, p) * comb(n%p, m%p, p) % p : 1;
}
```

```
//例题 hdu5446
/*
给你三个数 n, m, k
第二行是 k 个数, p1,p2,p3...pk
所有 p 的值不相同且 p 都是质数
求 C(n, m) % (p1*p2*p3*...*pk) 的值
范围：1 m n 1e18, 1 k 10, pi 1e5, 保证 p1*p2*p3*...*pk 1e18
我们知道题目要求 C(n, m) % (p1*p2*p3*...*pk) 的值
其实这个就是中国剩余定理最后算出结果后的最后一步求余
那 C(n, m) 相当于以前我们需要用中国剩余定理求的值
然而 C(n, m) 太大, 我们只好先算出
C(n, m) % p1 = r1
C(n, m) % p2 = r2
C(n, m) % p3 = r3
.
C(n, m) % pk = rk
用 Lucas, 这些 r1,r2,r3...rk 可以算出来
然后又是用中国剩余定理求答案
 */
const int maxn = 110;
const int maxp = 1e5 + 10;
const int INF = 0x3f3f3f3f;
```

```
LL N,M,K;
LL F[maxp],Finv[maxp],inv[maxp];
void init(LL MOD){
    inv[1] = 1;
    for(int i = 2; i < maxp; i ++){
        inv[i] = (MOD - MOD / i) * 1LL * inv[MOD % i] % MOD;
    }
    F[0] = Finv[0] = 1;
    for(int i = 1; i < maxp; i ++){
        F[i] = F[i - 1] * 1LL * i % MOD;
        Finv[i] = Finv[i - 1] * 1LL * inv[i] % MOD;
    }
}
LL quick_power(LL a,LL b,LL p){
    LL ans = 1;
    while(b){
        if(b & 1) ans = ans * a % p;
        b >>= 1;
        a = a * a % p;
    }
    return ans;
}
void ex_gcd(LL a, LL b, LL &x, LL &y, LL &d){
    if (!b) {d = a, x = 1, y = 0;}
    else{
        ex_gcd(b, a % b, y, x, d);
        y -= x * (a / b);
    }
}
LL Inv(LL t, LL p){//如果不存在，返回-1
    LL d, x, y;
    ex_gcd(t, p, x, y, d);
    return d == 1 ? (x % p + p) % p : -1;
}
LL fact(int n, LL p){//n 的阶乘求余 p
    LL ret = 1;
     for (int i = 1; i <= n ; i ++) ret = ret * i % p ;
    return ret ;
}
LL comb(int n,int m,LL p){
    if(m < 0 || m > n) return 0;
    return fact(n,p) * Inv(fact(m,p),p) % p * Inv(fact(n - m,p),p) % p;
}
LL Lucas(LL n,LL m,LL p){
    return m?Lucas(n / p,m / p,p) * comb(n % p,m % p,p) % p : 1;
}

LL mul(LL a,LL b,LL p){
    LL ans = 0;
    while(b){
        if(b & 1) ans = (ans + a) % p;
        a = (a + a) % p;
        b >>= 1;
```

```
    }
    return ans;
}
LL china(int n,LL *a,LL *m){
    LL M = 1,ret = 0;
    for(int i = 1; i <= n; i ++) M *= m[i];
    for(int i = 1; i <= n; i ++){
        LL w = M / m[i];
        ret = (ret + mul(w * Inv(w,m[i]),a[i],M)) % M;
    }
    return (ret + M) % M;
}
LL a[maxn],b[maxn];
int main(){
    int T; Sca(T);
    while(T--){
        scanf("%lld%lld%lld",&N,&M,&K);
        for(int i = 1; i <= K ; i ++){
            Scl(a[i]);
            b[i] = Lucas(N,M,a[i]);
        }
        Prl(china(K,b,a));
    }
    return 0;
}
```

6.12 康拓展开

```
//康拓展开，将一个长度为 n 的全排列 (1 - n) 映射为一个数字
//如果要求当前全排列是第几个排列，则需要把数字 + 1
int cantor(int *a, int n)
{
    int ans = 0;
    for (int i = 0; i < n; i++){
        int x = 0;
        int c = 1, m = 1; //c 记录后面的阶乘
        for (int j = i + 1; j < n; j++){
            if (a[j] < a[i]) x++;
            m *= c;
            c++;
        }
        ans += x * m;
    }
    return ans;
}

//康托展开逆运算，将一个数字映射回全排列，模板假设排列小于 10
static const int FAC[] = {1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880};    // 阶乘
void decantor(int x, int n){   //x 转化为 n 个数字的全排列
    vector<int> v; // 存放当前可选数
    vector<int> a; // 所求排列组合
    for (int i = 1; i <= n; i++) v.push_back(i);
    for (int i = n; i >= 1; i--){
        int r = x % FAC[i - 1];
```

```
        int t = x / FAC[i - 1];
        x = r;
        sort(v.begin(), v.end()); // 从小到大排序
        a.push_back(v[t]);        // 剩余数里第 t+1 个数为当前位
        v.erase(v.begin() + t);   // 移除选做当前位的数
    }
}
```

6.13 母函数

```
//母函数
/*
求用 1 分、2 分、3 分的邮票贴出不同数值的方案数：(每张邮票的数量是无限的)
那么
1 分：(1+x^1+x^2+x^3+x^4+.....)
2 分：(1+x^2+x^4+x^6+x^8+.....)
3 分：(1+x^3+x^6+x^9+x^12+.....)
然后这 3 个乘起来（让电脑去乘吧）
对于这种无限的，题目肯定会给你他询问的数值的范围，计算到最大的范围就可以了
 */
 #include<cstdio>
typedef long long LL;
const int maxn = 100 + 5;//假如题目只问到 100 为止
const int MAX = 3;//题目只有 1,2,3 这 3 种邮票
LL c1[maxn], c2[maxn];//c2 是临时合并的多项式，c1 是最终合并的多项式
int n;
void init(){
    c1[0] = 1;//一开始 0 的情况算一种
    for(int i = 1; i <= MAX; i ++){//把 1 分到 MAXN 的邮票合并，变成一个多项式
        for(int j = 0; j < maxn; j += i){//i 分的邮票，步长是 i
            for(int k = 0; j + k < maxn; k ++){//从 x^0 到 x^N 遍历一遍
                c2[j + k] += c1[k];//因为 j 的所有项系数为 1，所以 c1[k] 可以看成 c1[k]*1;
            }
        }
        for(int j = 0; j < maxn; j ++){//把 c2 的数据抄到 c1，清空 c2
            c1[j] = c2[j];
            c2[j] = 0;
        }
    }
}
int main(){
    init();
    while(scanf("%d", &n) != EOF){
        printf("%I64d\n", c1[n]);
    }
}
//样例 2，问一个数字 n 能够拆成多少种数字的和
/* 比如 n = 4
比如 n=4
4 = 4;4 = 3 + 1;
4 = 2 + 2;4 = 2 + 1 + 1;
4 = 1 + 1 + 1 + 1;
有 5 种，那么答案就是 5*/
//相当于把所有数当成邮票看能组成多少
```

```
const int maxn = 130;
const int MAX = 120;
int N,M,K;
LL c1[maxn],c2[maxn];
void init(){
    c1[0] = 1;
    for(int i = 1; i <= MAX; i ++){
        for(int j = 0 ; j < maxn; j += i){
            for(int k = 0 ; j + k < maxn; k ++){
                c2[j + k] += c1[k];
            }
        }
        for(int j = 0 ; j < maxn; j ++){
            c1[j] = c2[j];
            c2[j] = 0;
        }
    }
}
int main(){
    init();
    while(~Sca(N)) Prl(c1[N]);
    return 0;
}
```

6.14 高斯消元

```
//高斯消元, 时间复杂度 O(n ^ 3)
/*
高斯消元是求解线性方程组的一种做法。将线性方程组列为矩阵的形式
x1 + 2x2 - x3 = -6
2x1 + x2 - 3x3 = -9
-x1 - x2 + 2x3 = 7
可以写成
1 2 -1 -6
0 -3 -1 3
-1 -1 2 7
这么一个矩阵, 然后利用初等行变换化为最简求每个变量的值
 */
 //模板, 给出像上述增广矩阵求每个变量的值
int N,M,K;
double MAP[maxn][maxn];
double ans[maxn];
void gauss(){
    //化简系数矩阵为 01 矩阵
    for(int i = 1; i <= N; i ++){
        int r = i;
        for(int j = i + 1; j <= N ; j ++){
            if(fabs(MAP[r][i]) < fabs(MAP[j][i])) r = j;
        }
        if(fabs(MAP[r][i]) < eps){
            puts("No Solution"); //没有唯一解 (无穷多个解)
            //如果要求解的个数, 那么自由元 ++, 答案是 2 ^ 自由元
            return ;
        }
```

```
        swap(MAP[i],MAP[r]);
        double div = MAP[i][i];
        for(int j = i; j <= N + 1; j ++) MAP[i][j] /= div;
        for(int j = i + 1; j <= N ; j ++){
            div = MAP[j][i];
            for(int k = i ; k <= N + 1; k ++) MAP[j][k] -= MAP[i][k] * div;
        }
    }
    for(int i = N; i >= 1; i --){
        ans[i] = MAP[i][N + 1];
        for(int j = i + 1; j <= N ; j ++){
            ans[i] -= MAP[i][j] * ans[j];
        }
    }
    for(int i = 1; i <= N ; i ++) printf("%.2lf\n",ans[i]);
}
int main(){
    Sca(N);
    for(int i = 1; i <= N ; i ++){
        for(int j = 1; j <= N + 1; j ++){
            scanf("%lf",&MAP[i][j]);
        }
    }
    gauss();
    return 0;
}

/*
高斯消元也可求解一类开关问题
所有开关的的开闭状态从一个指定状态变为 src 另一个指定状态 dst 的方法
改变一个开关 j 会改变另一个开关 i 的状态 aij, 可列出方程
a1,1 ^ x1 ^ a1,2 ^ x2 .... ^ a1n ^ xn = src1 ^ dst1
a2,1 ^ x1 ^ a2,2 ^ x2 .....^ a2n ^ xn = src2 ^ dst2
a 为已知系数, src 和 dst 为已知系数, 直接高斯消元求解可行种数
种数就是 (1 << 自由元个数)
n 小的话可以用状态压缩, 大的话可以用 bitset
 */
 //POJ1830
//有 N 个相同的开关, 每个开关都与某些开关有着联系, 每当你打开或者关闭某个开关的时候, 其他的与
↪ 此开关相关联的开关也会相应地发生变化, 即这些相联系的开关的状态如果原来为开就变为关, 如果
↪ 为关就变为开。你的目标是经过若干次开关操作后使得最后 N 个开关达到一个特定的状态。对于任意
↪ 一个开关, 最多只能进行一次开关操作。你的任务是, 计算有多少种可以达到指定状态的方法。(不计
↪ 开关操作的顺序)
const int maxn = 33;
const int INF = 0x3f3f3f3f;
const int mod = 1e9 + 7;
int N,M,K;
bitset<maxn>MAP[maxn];
int gauss(){
    int cnt = 0;
    for(int i = 1; i <= N ; i ++){
        int r = i;
        for(int j = i + 1; j <= N && !MAP[r][i]; j ++){
```

```
            if(MAP[j][i]) r = j;
        }
        if(MAP[i].count() == 1 && MAP[i][N + 1] == 1) return -1;  //注意先判断无解再判断自由元
        if(!MAP[r][i]){
            cnt++;  //自由元 ++
            continue;
        }
        swap(MAP[i],MAP[r]);
        for(int j = i + 1; j <= N ; j ++){
            if(!MAP[j][i]) continue;
            for(int k = i ; k <= N + 1; k ++) MAP[j][k] = MAP[j][k] ^ MAP[i][k];
        }
    }
    return (1 << cnt);
}
int main(){
    int T; Sca(T);
    while(T--){
        Sca(N);
        for(int i = 0 ; i <= N + 1; i ++) MAP[i].reset();
        for(int i = 1; i <= N ; i ++) MAP[i][N + 1] = read();
        for(int i = 1; i <= N ; i ++){
            MAP[i][N + 1] = MAP[i][N + 1] ^ read();
            MAP[i][i] = 1;
        }
        int x,y;
        while(~Sca2(x,y) && x && y) MAP[y][x] = 1;
        int ans = gauss();
        if(~ans) Pri(ans);
        else puts("Oh,it's impossible~!!");
    }
    return 0;
}
```

6.15 线性空间

```
/*
线性空间
定义
线性空间是一个关于一下两个运算封闭的向量集合：
1. 向量加法 a+b，其中 a,b 为向量
2. 标量乘法 k*a，其中 a 为向量，k 为常数
基础概念
1. 给定若干个向量 a1,a2,...,an，若向量 b 能够通过 a1,a2,...,an 经过向量加法和标量乘法得到，则称向量 b 能够通过 a1,a2,...,an 表出。
2. 对于向量 a1,a2,...,an 能够表出的所有向量所构成了一个线性空间，称 a1,a2,...,an 为这个线性空间的生成子集。
3. 在一个线性空间中选取若干个向量，若一个向量能够被其他向量表出，则称这些向量线性相关，反之，称这些向量线性无关。
4. 线性无关的生成子集成为线性空间的基底，简称基。
4 . 另外地，线性空间的极大线性无关子集也称为线性空间的基底，简称基。
5. 一个线性空间的所有基包含的向量个数都相等，这个个数称为线性空间的维数。
```

```
高斯消元和线性基
对于一个 n*m 的矩阵，我们可以把它看做 n 个长度为 m 的向量，即它的维数为 m。那么如果将这个矩阵看做系数矩阵进行高斯消元，则消元后得到的矩阵中所有非 0 行代表的向量线性无关。
因为初等行变换就是对每行的向量之间进行向量加法和标量乘法，所以高斯消元的操作并不改变 n 个向量所能表出的线性空间，则消元后的矩阵的所有非 0 行所代表的向量就是原线性空间的一个基。
综上所述，我们可以利用高斯消元算法对若干个向量求解其构成线性空间的一个基。
 */
//BZOJ4004
//每个向量有一个权值，求给定矩阵的秩和组成这些秩的向量的最小权值和
//这次不需要解线性方程组了，高斯消元的目的是化简矩阵
const int maxn = 510;
const double eps = 1e-5;  //这个精度可能要调，看 AC 情况
int N,M,K;

long double MAP[maxn][maxn]; //long double 提升精度
int c[maxn];
void gauss(){
    int num = 0,ans = 0;
    for(int i = 1; i <= min(N,M) ; i ++){
        int r = i;
        for(int j = i + 1; j <= N ; j ++){
            if(fabs(MAP[j][i]) < eps) continue;
            if(fabs(MAP[r][i]) < eps || c[r] > c[j]) r = j;
        } //每次选取代价最小的基底
        if(fabs(MAP[r][i]) < eps) continue;
        num++; ans += c[r];
        swap(MAP[r],MAP[i]); swap(c[r],c[i]);
        long double div = MAP[i][i];
        for(int j = i; j <= M; j ++) MAP[i][j] /= div;
        for(int j = i + 1; j <= N ; j ++){
            div = MAP[j][i];
            for(int k = i ; k <= M; k ++) MAP[j][k] -= MAP[i][k] * div;
        }
    }
    printf("%d %d",num,ans);
}
int main(){
    Sca2(N,M);
    for(int i = 1; i <= N ; i ++){
        for(int j = 1; j <= M ; j ++){
            cin >> MAP[i][j];
        }
    }
    for(int i = 1; i <= N; i ++) Sca(c[i]);
    gauss();
    return 0;
}
```

6.16 线性基

```
/*    线性基：处理许多数之间的异或问题
对于一个数集 V，它的线性基   是它的一个子集，满足   中所有数互相异或得到的集合等价于 V 中所有数互相异或得到的集合。也就是说，   可以看成是 V 的压缩。
线性基有一些性质：
```

```
1. 线性基的异或集合中不存在 0。也就是说， 是 V 中线性无关的极大子集。(这些概念以后再补吧。。)
2. 线性基中每个元素的异或方案唯一，也就是说，线性基中不同的异或组合异或出的数都是不一样的。这
↪ 个与性质 1 其实是等价的。
3. 线性基的二进制最高位互不相同。
4. 如果线性基是满的，它的异或集合为 [1,2n-1]。
5. 线性基中元素互相异或，异或集合不变。
*/
const int maxn = 110;
const int INF = 0x3f3f3f3f;
const int mod = 1e9 + 7;
int N,M,K;
LL a[maxn],p[100],d[100];
//建立线性基（插入操作）
inline void add(LL x){
    for(int i = 62; i >= 0; i --){
        if(!(x & (1LL << i))) continue; //注意 1LL
        if(!p[i]){p[i] = x;break;}      //注意 break
        x ^= p[i];                      //注意是 x 变动
    }
}
//查询异或最小值：线性基中 位数最低的数字
//查询异或最大值
inline LL getmax(){
    LL ans = 0;
    for(LL i = 62; i >= 0; i --){
        if(ans < (ans ^ p[i])) ans ^= p[i];
    }
    return ans;
}
//check 是否存在子集合可以异或为这个数
bool check(LL x){  // true 存在,false 不存在
    for(int i = 62; i >= 0 ; i --){
        if(!(x & (1LL << i))) continue;
        if(!p[i]) return false;
        x ^= p[i];
    }
    return x == 0;
}
//查询集合所有可异或数中 K 小的数
//先将线性基 rebuild, 然后可以 O(logn) 的查找
void rebuild(){
    for(LL i = 62; i >= 0 ; i --){
        for(LL j = i - 1; j >= 0; j --){
            if(p[i] & (1LL << j)) p[i] ^= p[j];
        }
    }
    for(int i = 0 ; i <= 62; i ++) if(p[i]) d[cnt++] = p[i];
    return ;
}
LL query(LL k){
    if(!k) return 0;
    if(k >= (1LL << cnt)) return -1;
    LL ans = 0;
```

```
    for(int i = 62; i >= 0; i --){
        if(k & (1LL << i)) ans ^= d[i];
    }
    return ans;
}
//main 函数：查找线性基异或数的 K 小
int main(){
    int T = read();
    int CASE = 1;
    while(T--){
        for(int i = 0 ; i <= 62; i ++) p[i] = 0;
        Sca(N);
        for(int i = 1; i <= N ; i ++){
            Scl(a[i]); add(a[i]);
        }
        cnt = 0; rebuild();
        Sca(Q);
        int flag = (cnt != N);
        for(int i = 1; i <= Q; i ++){
            LL k; scanf("%lld",&k);
            Prl(query(k - flag));
        }
    }
    return 0;
}

//例题：hdu6579
/* 给出一个序列.
操作 1 在序列尾增加一个数 操作 2. 求区间异或最大值
做法：维护每个前缀的线性基 (上三角形态)
对于每个线性基，讲出现为止靠右的数字尽可能放在高位，也就是说在插入新的数字的时候，要同时记录对
↪ 应位置上数字出现位置，
并且再找到可以插入的位置的时候，如果新数字比原来位置上的数字更靠右，就将该位置上原来的数字向地
↪ 位推。
在求最大值的时候，从高位向低位遍历，如果该位上数字出现在询问中左端点的右侧且可以使答案变大，就
↪ 异或进答案。
对于线性基的每一位，与它异或过的线性基更高位置上的数字肯定都出现在它优策
 */
const int maxn = 1e6 + 10;
int N,M,K;
struct xj{
    int a[32],pos[32];
    inline void init(){
        for(int i = 0; i <= 30; i ++) a[i] = pos[i] = 0;
    }
    inline void add(int x,int p){
        for(int i = 30; i >= 0 ; i --){
            if(!(x & (1 << i))) continue;
            if(!a[i]){
                a[i] = x; pos[i] = p;
                break;
            }else{
                if(p > pos[i])swap(a[i],x),swap(p,pos[i]);
```

```
                x ^= a[i];
            }
        }
    }
    inline int getmax(int l){
        int ans = 0;
        for(int i = 30; i >= 0; i --){
            if(pos[i] < l) continue;
            if(ans < (ans ^ a[i])) ans ^= a[i];
        }
        return ans;
    }
}pre[maxn];
int main(){
    int T; Sca(T);
    while(T--){
        Sca2(N,M);
        for(int i = 1; i <= N ; i ++){
            int x = read();
            pre[i] = pre[i - 1];
            pre[i].add(x,i);
        }
        int ans = 0;
        while(M--){
            int op = read();
            if(!op){
                int l = read() ^ ans,r = read() ^ ans;
                l = l % N + 1; r = r % N + 1;
                if(l > r) swap(l,r);
                ans = pre[r].getmax(l);
                Pri(ans);
            }else{
                int x = read() ^ ans;
                N++;
                pre[N] = pre[N - 1];
                pre[N].add(x,N);
            }
        }
    }
    return 0;
}
```

6.17 拉格朗日插值

6.17.1 拉格朗日插值

$$f(k) = \sum_{i=0}^{n} y_i \prod_{i!=j} \frac{k - x[j]}{x[i] - x[j]}$$

```
/* 拉格朗日插值
众所周知，n + 1 个 x 坐标不同的点可以确定唯一的最高为 n 次的多项式。在算法竞赛中，我们常常会
↪ 碰到一类题目，题目中直接或间接的给出了 n+1 个点，让我们求由这些点构成的多项式在某一位置的
↪ 取值
```

```
一个最显然的思路就是直接高斯消元求出多项式的系数，但是这样做复杂度巨大 (n^3) 且根据算法实现不同往往会存在精度问题
而拉格朗日插值法可以在 n^2 的复杂度内完美解决上述问题
假设该多项式为 f(x)，第 i 个点的坐标为 (xi,yi)，我们需要找到该多项式在 k 点的取值
根据拉格朗日公式可以直接计算
 */
 const int maxn = 2010;
const int INF = 0x3f3f3f3f;
const LL mod = 998244353;
int N,M,K;
LL x[maxn],y[maxn];
LL quick_power(LL a,LL b,LL Mod){
    LL ans = 1;
    while(b){
        if(b & 1) ans = ans * a % Mod;
        a = a * a % Mod;
        b >>= 1;
    }
    return ans;
}
LL inv(LL a,LL Mod){
    return quick_power(a,Mod - 2,Mod);
}
LL v;
int main(){
    Sca(N); Scl(v);
    for(int i = 1; i <= N ; i ++) scanf("%lld%lld",&x[i],&y[i]);
    LL ans = 0;
    for(int i = 1; i <= N ; i ++){
        LL sum = 1;
        for(int j = 1; j <= N ; j ++){
            if(i != j) sum = sum * (x[i] + mod - x[j]) % mod;
        }
        sum = inv(sum,mod);
        sum = sum * y[i] % mod;
        for(int j = 1; j <= N ; j ++){
            if(i != j) sum = sum * (v - x[j] + mod) % mod;
        }
        ans = (ans + sum) % mod;
    }
    Prl(ans);
    return 0;
}
```

6.17.2 在 x 取值连续时的做值

$$pre_i = \prod_{j=0}^{i} k - j$$

$$suf_i = \prod_{j=i}^{n} k - j$$

$$f(k) = \sum_{i=0}^{n} y_i \frac{pre_{i-1} * suf_{i+1}}{fac[i] * fac[n-i]}$$

6.17.3 重心拉格朗日插值法

$$g = \prod_{i=1}^{n} k - x[i]$$

$$t_i = \frac{y_i}{\prod_{j \neq i} x_i - x_j}$$

$$f(k) = g \sum_{i=0}^{n} \frac{t_i}{(k - x[i])}$$

```
/* 重心拉格朗日插值法
求原函数复杂度为 O(n^2)，单次求值复杂度 O(n^2)。
动态加点？
每次增加一个点，一般求法会重新计算一次达到 O(n2^) 的复杂度，而重心拉格朗日插值法只需 O(n) 计
↪ 算 wi 即可。
 */
 /* 模板：tyvj1858
f(i)=1^k+2^k+3^k+......+i^k
g(x)=f(1)+f(2)+f(3)+....+f(x)
求 (g(a)+g(a+d)+g(a+2d)+......+g(a+nd))mod p
解：可知 f(i) 是 k + 1 次多项式,g(x) 是 k + 2 次多项式,h(i) 是 k + 3 次多项式
  */
const int maxn = 210;
const LL mod = 1234567891;
LL K,a,n,d,p;
inline LL mul(LL a,LL b){return (a % mod * b % mod + mod) % mod;}
inline LL add(LL a,LL b){return ((a + b) % mod + mod) % mod;}
LL quick_power(LL a,LL b){
    LL ans = 1;
    while(b){
        if(b & 1) ans = mul(ans,a);
        b >>= 1;
        a = mul(a,a);
    }
    return ans;
}
LL inv(LL a){
    return quick_power(a,mod - 2);
}
LL get(LL *t,LL *x,LL *y,LL Lim,LL n){
    LL g = 1;
    for(int i = 1; i <= Lim; i ++) if(n == x[i]) return y[i];
    for(int i = 1; i <= Lim; i ++) g = mul(g,add(n,-x[i]));
    LL ans = 0;
    for(int i = 1; i <= Lim ; i ++) ans = add(ans,mul(t[i],inv(add(n,-x[i]))));
    ans = mul(ans,g);
    return ans;
}
LL f[maxn],g[maxn],h[maxn],t[maxn];
LL x[maxn];
int main(){
    int T; Sca(T);
    while(T--){
```

```
        scanf("%lld%lld%lld%lld",&K,&a,&n,&d);
        for(int i = 1; i <= 130; i ++) f[i] = add(f[i - 1],quick_power(i,K));
        for(int i = 1; i <= 130; i ++) g[i] = add(g[i - 1],f[i]);
        for(int i = 1; i <= K + 3; i ++) x[i] = i;
        for(int i = 1; i <= K + 3; i ++){
            t[i] = 1;
            for(int j = 1 ; j <= K + 3; j ++){
                if(i == j) continue;
                t[i] = mul(t[i],x[i] - x[j]);
            }
            t[i] = mul(g[i],inv(t[i]));
        }
        for(int i = 1; i <= K + 4; i ++) h[i] = add(h[i - 1],get(t,x,g,K + 3,add(a,mul(i
        ↪  - 1,d))));
        for(int i = 1; i <= K + 4; i ++) x[i] = add(a,mul(i - 1,d));
        for(int i = 1; i <= K + 4; i ++){
            t[i] = 1;
            for(int j = 1; j <= K + 4; j ++){
                if(i == j) continue;
                t[i] = mul(t[i],x[i] - x[j]);
            }
            t[i] = mul(h[i],inv(t[i]));
        }
        Prl(get(t,x,h,K + 4,add(a,mul(n,d))));
    }
    return 0;
}
```

6.17.4 自然数连续幂次和

```
//Σi ^ k
//1^k+2^k+3^k+...n^k
/* 可以被证明，结果是一个 n 为自变量的 k + 1 次多项式
因此我们将 n = (0,1,2,3....,k,k + 1) 代入算出函数值
将这 0 ~ k + 1 所有的点用拉格朗日插值插入
求解的时候就将 n 作为下标代入就可以了。
可以用上面在 x 取值连续时的做法优化，时间复杂度 k ^ 2
 */
 //luoguP4593 模板为函数 getsum
const int maxn = 70;
const int mod = 1e9 + 7;
LL N,M,K;
LL fac[maxn],ifac[maxn]; //阶乘，阶乘的逆元
LL mul(LL a,LL b){
    return a % mod * b % mod;
}
LL add(LL a,LL b){
    return ((a + b) % mod + mod) % mod;
}
LL quick_power(LL a,LL b){
    LL ans = 1;
    while(b){
        if(b & 1) ans = mul(ans,a);
        a = mul(a,a);
```

```
        b >>= 1;
    }
    return ans;
}
LL inv(LL x){
    return quick_power(x,mod - 2);
}
void init(){
    fac[0] = 1;
    for(int i = 1; i <= 60 ; i ++) fac[i] = mul(i,fac[i - 1]);
    ifac[60] = inv(fac[60]);
    for(int i = 59; i >= 0; i --) ifac[i] = mul(i + 1,ifac[i + 1]);
}
LL a[maxn],pre[maxn],suf[maxn],y[maxn];
LL getsum(LL n,LL k){ // 1 ^ k + 2 ^ k + ... + n ^ k
    LL Lim = k + 1,ans = 0; y[0] = 0;
    for(int i = 1; i <= Lim; i ++) y[i] = add(y[i - 1],quick_power(i,k));
    pre[0] = n; suf[Lim + 1] = 1;
    for(int i = 1; i <= Lim; i ++) pre[i] = mul(pre[i - 1],add(n,-i));
    for(int i = Lim; i >= 1; i --) suf[i] = mul(suf[i + 1],add(n,-i));
    for(int i = 1 ; i <= Lim; i ++){
        LL up = mul(y[i],mul(pre[i - 1],suf[i + 1]));
        LL down = mul(ifac[i],ifac[Lim - i]);
        if((Lim - i) & 1) down = mod - down;
        ans = add(ans,mul(down,up));
    }
    return ans;
}
int main(){
    int T; Sca(T); init();
    while(T--){
        scanf("%lld%lld",&N,&M); K = M + 1;
        for(int i = 1; i <= M; i ++) Scl(a[i]);
        a[++M] = ++N;
        sort(a + 1,a + 1 + M);
        LL ans = 0;
        for(int i = 1; i <= M ; i ++){
            for(int j = i ; j <= M ; j ++) ans = add(ans,add(getsum(a[j] -
            ↪  1,K),-getsum(a[j - 1],K)));
            for(int j = i + 1; j <= M ; j ++) a[j] = add(a[j],-a[i]);
            a[i] = 0;
        }
        Prl(ans);
    }
    return 0;
}
```

6.18 FFT

6.18.1 卷积

```
//FFT 中的卷积
/*
形如 A(i) = f(j)g(i - j)
```

```
将 f(j),g(i - j) 两个多项式相乘,fft 加速
第 i 项的系数就是 A(i)
*/
```

6.18.2 递归法

```
/* fft 快速傅里叶变换
用于加速多项式的乘法，形如
给定一个 n 次多项式 F(x)，和一个 m 次多项式 G(x)。
求出 F(x) 和 G(x) 的卷积的题目，常规做法是 O(n ^ 2) 将系数遍历相乘。
fft 先将两个多项式 nlogn 转化为点值表示法，然后 O(n) 相乘，最后 nlogn 转化为系数表示法
注：这里的卷积即两个多项式自然相乘
 */
 //递归法：常数巨大，容易爆栈，一般情况下用迭代法
const int maxn = 1e6 + 10;
const double PI = acos(-1.0);
int N,M,K;
struct complex{
    double x,y;
    complex(){}
    complex(double x,double y):x(x),y(y){}
    friend complex operator - (complex a,complex b){return complex(a.x - b.x,a.y - b.y);}
    friend complex operator + (complex a,complex b){return complex(a.x + b.x,a.y + b.y);}
    friend complex operator * (complex a,complex b){return complex(a.x * b.x - a.y *
     ↪  b.y,a.x * b.y + a.y * b.x);}
}a[maxn << 2],b[maxn << 2]; //最坏情况下是 4 倍
void FFT(int limit,complex *a,int type){  //limit 一定是 2 的幂次，这样保证每次可以对半分
    if(limit == 1) return;
    complex a1[limit >> 1],a2[limit >> 1]; //分出奇偶位置上的数
    for(int i = 0 ; i < limit; i += 2){
        a1[i >> 1] = a[i];
        a2[i >> 1] = a[i + 1];
    }
    FFT(limit >> 1,a1,type);
    FFT(limit >> 1,a2,type);
    complex Wn = complex(cos(2 * PI / limit),type * sin(2 * PI / limit)),w =
     ↪  complex(1,0);
    limit >>= 1;    //Wn 为单位元,w 为幂次
    for(int i = 0 ; i < limit; i ++, w = w * Wn){
        a[i] = a1[i] + w * a2[i];
        a[i + limit] = a1[i] - w * a2[i];
    }
}
int main(){
    Sca2(N,M);
    for(int i = 0; i <= N ; i ++){a[i].y = 0; scanf("%lf",&a[i].x);}
    for(int j = 0; j <= M ; j ++){b[j].y = 0; scanf("%lf",&b[j].x);}
    int limit = 1;
    while(limit <= N + M) limit <<= 1;//使用的是 0~limit - 1 这几个数
    FFT(limit,a,1); FFT(limit,b,1);   //系数转化为点值表示
    for(int i = 0; i < limit ; i ++) a[i] = a[i] * b[i];  //点值相乘只要 O(n)
    FFT(limit,a,-1);                  //点值转化回系数表示
    for(int i = 0 ; i <= N + M; i ++){
        printf("%d ",(int)(a[i].x / limit + 0.5));  //输出系数
```

```
    }
    return 0;
}
```

6.18.3 迭代法

```
//迭代法，发现原位置下标和目标位置下表是二进制下翻转的关系
//因此可以预处理好位置优化，不用递归实现，常数较小较安全
//FFT 小贴士:1. 一定要看清题目需求决定答案要不要/2, 例如只想求 a,b 组合出的情况，不除 2 的情
↪ 况下系数同时表示 a 和 b 组合以及 b 和 a 组合
//2. 如果是自身对自身 fft 一定要处理好相同位置相乘的情况
const int maxn = 1e6 + 10;
const double PI = acos(-1.0);
int N,M,K;
struct complex{
    double x,y;
    complex(){}
    complex(double x,double y):x(x),y(y){}
    friend complex operator - (complex a,complex b){return complex(a.x - b.x,a.y - b.y);}
    friend complex operator + (complex a,complex b){return complex(a.x + b.x,a.y + b.y);}
    friend complex operator * (complex a,complex b){return complex(a.x * b.x - a.y *
    ↪ b.y,a.x * b.y + a.y * b.x);}
}a[maxn << 2],b[maxn << 2];
int r[maxn << 2];
void FFT(int limit,complex *A,int *r,int type){  //limit 一定是 2 的幂次，这样保证每次可以
↪ 对半分
    for(int i = 0 ; i < limit; i ++) if(i < r[i]) swap(A[i],A[r[i]]);
    for(int mid = 1; mid < limit; mid <<= 1){  //待合并区间长度的一半
        complex Wn(cos(PI / mid),type * sin(PI / mid)); //单位根,mid 只有一半所以分子不用
        ↪ * 2
        int len = mid << 1;  //区间长度
        for(int j = 0; j < limit; j += len){ //j 表示更新到的位置
            complex w(1,0);
            for(int k = 0 ; k < mid; k ++, w = w * Wn){
                complex x = A[j + k],y = w * A[j + mid + k];
                A[j + k] = x + y;
                A[j + mid + k] = x - y;
            }
        }
    }
}
int main(){
    Sca2(N,M);
    for(int i = 0; i <= N ; i ++){a[i].y = 0; a[i].x = read();}
    for(int j = 0; j <= M ; j ++){b[j].y = 0; b[j].x = read();}
    int limit = 1,l = 0; while(limit <= N + M) limit <<= 1,l++;
    for(int i = 0 ; i < limit; i ++) r[i] = (r[i >> 1] >> 1) | ((i & 1) << (l - 1));
    FFT(limit,a,r,1); FFT(limit,b,r,1);
    for(int i = 0; i < limit ; i ++) a[i] = a[i] * b[i];
    FFT(limit,a,r,-1);
    for(int i = 0 ; i <= N + M; i ++){
        printf("%d ",(int)(a[i].x / limit + 0.5));
    }
    return 0;
```

```
}
```

6.18.4 字符串匹配

```
/* FFT 在字符串匹配中的应用
(1). 普通的单模式串匹配 (KMP 模板题)
模式串 A, 长度为 m, 文本串 B, 长度为 n
1. 定义 A(x),B(y) 的匹配函数 C(x,y) 为 (A(x) - B(y))^2, 若 C(x,y)=0, 则称 A 的第 x 个字符和
↪ B 的第 y 个字符匹配
2. 定义完全匹配函数 P(x) = C(i,x - m + i + 1) (0 <= i <= m - 1) 若 P(x)=0, 则称 B 以第 x
↪ 位结束的连续 m 位, 与 A 完全匹配
3. 将 A 反转为 S, 同时将 S, B 代入完全匹配函数 P(x), 得到
P(x) = [S(m - i - 1) - B(x - m + i + 1)]^2 (0 <= i <= m - 1)
展开化简, 得到
P(x) = S(m - i + 1)^2 + B(x - m + i + 1)^2 - 2S(m - i - 1)B(x - m + i + 1) (0 <= i <= m -
↪ 1)
第一项第二项是一个前缀和, 预处理一下就可以, 发现第三项的两个系数加起来正好是 x, 所以第三项可
↪ 以写为
-2S(i)B(j) (i + j = x) , 这就是一个基本卷积的形式了, 可以用 FFT 做
 */
 void FFT_Match(char *s1,char *s2,int m,int n){
    for(int i=0;i<m;i++) A[i].x=s1[i]-'a'+1;
    for(int i=0;i<n;i++) B[i].x=s2[i]-'a'+1;
    reverse(A,A+m);double T=0;
    for(int i=0;i<m;i++) T+=A[i].x*A[i].x;
    f[0]=B[0].x*B[0].x;
    for(int i=1;i<n;i++) f[i]=f[i-1]+B[i].x*B[i].x;
    FFT(A,len,1);FFT(B,len,1);
    for(int i=0;i<len;i++) g[i]=A[i]*B[i];
    FFT(g,len,-1);
    for(int x=m-1;x<n;x++){
        double P=T+f[x]-f[x-m]-2*g[x].r;
        if(fabs(P)<eps) printf("%d ",x-m+2);
    }
}
/*(2) 带有通配符的模式串匹配
这就不能用 KMP 了, 但是对于 fft 来说, 方法和上面相似
1. 定义 A(x),B(y) 的匹配函数 C(x,y) = (A(x) - B(y))^2 * A(x) * B(y); 将通配符 * 的值赋 0,
↪ 表示除了他俩相等, 还有出现通配符的情况都算匹配
2. 定义完全匹配书 P(x) = C(i,x - m + i + 1) (0 <= i <= m - 1) 若 P(x)=0, 则称 B 以第 x 位
↪ 结束的连续 m 位, 与 A 完全匹配
3. 将 A 反转为 S, 同时将 S,B 代入完全匹配函数 P(x), 得到
P(x) = [S(m - i - 1) - B(x - m + i + 1)]^2 * S(m - i - ) * B(x - m + i + 1)   (0 <= i <=
↪ m - 1)
展开化简, 得到
P(x) = S(m - i - 1)^3B(x - m + i + 1) + S(m - i - 1)B(x - m + i + 1)^3 - 2S(m - i -
↪ 1)^2B(x - m + i + 1)^2 (0 <= i <= m - 1)
因为 (m - i + 1) + (x - m + i + 1) = x, 所以
P(x) = S(i)^3B(j) + S(i)B(j)^3 - 2S(i)^2B(j)^2  (i + j = x)
三段 fft 可以解决
 */
 const int maxn = 3e5 + 10;
const double PI = acos(-1.0);
int N,M,K;
```

```
char str1[maxn],str2[maxn];
struct complex{
    double x,y;
    complex(double x = 0,double y = 0):x(x),y(y){}
    void init(){x = y = 0;}
    friend complex operator + (complex a,complex b){return complex(a.x + b.x,a.y + b.y);}
    friend complex operator - (complex a,complex b){return complex(a.x - b.x,a.y - b.y);}
    friend complex operator * (complex a,complex b){return complex(a.x * b.x - a.y *
    ↪  b.y,a.x * b.y + a.y * b.x);}
}a[maxn << 2],b[maxn << 2],ans[maxn << 2];
int ca[maxn],cb[maxn];
int r[maxn << 2];
vector<int>s;
void FFT(int limit,complex *A,int *r,int type){
    for(int i = 0 ; i < limit; i ++){
        if(i < r[i]) swap(A[i],A[r[i]]);
    }
    for(int mid = 1; mid < limit; mid <<= 1){
        int len = mid << 1;
        complex Wn(cos(PI/mid),type * sin(PI/mid));
        for(int j = 0 ; j < limit; j += len){
            complex w(1,0);
            for(int k = 0 ; k < mid; k ++,w = w * Wn){
                complex x = A[j + k],y = w * A[j + mid + k];
                A[j + k] = x + y;
                A[j + mid + k] = x - y;
            }
        }
    }
}
inline double cul(double x,int y){
    if(y == 1) return x;
    if(y == 2) return x * x;
    return x * x * x;
}
int main(){
    Sca2(N,M);
    scanf("%s%s",str1,str2); //长串 str2, 短串 str1
    for(int i = 0 ; i < N / 2; i ++) swap(str1[i],str1[N - i - 1]);
    int limit = 1,l = 0;
    while(limit <= N + M - 2) limit <<= 1,l++;
    for(int i = 0 ; i < limit; i ++) r[i] = (r[i >> 1] >> 1) | ((i & 1) << (l - 1));
    for(int i = 0 ; i < N ; i ++) if(str1[i] != '*') ca[i] = str1[i] - 'a' + 1;
    for(int i = 0 ; i < M ; i ++) if(str2[i] != '*') cb[i] = str2[i] - 'a' + 1;
    for(int k = 1; k <= 3; k ++){
        for(int i = 0 ; i < limit; i ++) a[i].init(),b[i].init();
        for(int i = 0 ; i < N ; i ++) a[i].x = cul(ca[i],k);
        for(int i = 0 ; i < M ; i ++) b[i].x = cul(cb[i],3 - k + 1);
        FFT(limit,a,r,1); FFT(limit,b,r,1);
        int t = 1; if(k == 2) t = -2;
        for(int i = 0 ; i < limit; i ++) ans[i] = ans[i] + t * a[i] * b[i];
    }
    FFT(limit,ans,r,-1);
```

```
    for(int i = N - 1; i < M; i ++){   //一般短串逆置的范围
        if(!(int)(ans[i].x / limit + 0.5)) s.push_back(i - N + 2);
    }
    Pri(s.size());
    for(int i = 0 ; i < s.size(); i ++) printf("%d ",s[i]);
    return 0;
}
```

6.18.5 NTT

```
//NTT(快速数论变换)
/*
干的事情和 FFT 差不多，优点：1. 可以取模，FFT 的浮点运算时不可以取模的
2. 全部整数运算避免了 fft 的浮点数误差
缺点:1. 多项式的系数必须是整数
2. 如果取模的数字不是 998244353,1004535809,469762049 这几个原根是 3 的数字，
需要中国剩余定理求，很麻烦
*/
//模数为 998244353
const int maxn = 3 * 1e6 + 10, P = 998244353, G = 3, Gi = 332748118;
int N, M, limit = 1, L, r[maxn];
LL a[maxn], b[maxn];
inline LL fastpow(LL a, LL k) {
    LL base = 1;
    while(k) {
        if(k & 1) base = (base * a ) % P;
        a = (a * a) % P;
        k >>= 1;
    }
    return base % P;
}
inline void NTT(LL *A, int type) {
    for(int i = 0; i < limit; i++)
        if(i < r[i]) swap(A[i], A[r[i]]);
    for(int mid = 1; mid < limit; mid <<= 1) {
        LL Wn = fastpow( type == 1 ? G : Gi , (P - 1) / (mid << 1));
        for(int j = 0; j < limit; j += (mid << 1)) {
            LL w = 1;
            for(int k = 0; k < mid; k++, w = (w * Wn) % P) {
                int x = A[j + k], y = w * A[j + k + mid] % P;
                A[j + k] = (x + y) % P,
                A[j + k + mid] = (x - y + P) % P;
            }
        }
    }
}
int main() {
    N = read(); M = read();
    for(int i = 0; i <= N; i++) a[i] = (read() + P) % P;
    for(int i = 0; i <= M; i++) b[i] = (read() + P) % P;
    while(limit <= N + M) limit <<= 1, L++;
    for(int i = 0; i < limit; i++) r[i] = (r[i >> 1] >> 1) | ((i & 1) << (L - 1));
    NTT(a, 1);NTT(b, 1);
    for(int i = 0; i < limit; i++) a[i] = (a[i] * b[i]) % P;
```

```
    NTT(a, -1);
    LL inv = fastpow(limit, P - 2);
    for(int i = 0; i <= N + M; i++)
        printf("%d ", (a[i] * inv) % P);
    return 0;
}
//任意模数 NTT
int mod;
namespace Math {
    inline int pw(int base, int p, const int mod) {
        static int res;
        for (res = 1; p; p >>= 1, base = static_cast<long long> (base) * base % mod) if
        ↪  (p & 1) res = static_cast<long long> (res) * base % mod;
        return res;
    }
    inline int inv(int x, const int mod) { return pw(x, mod - 2, mod); }
}

const int mod1 = 998244353, mod2 = 1004535809, mod3 = 469762049, G = 3;
const long long mod_1_2 = static_cast<long long> (mod1) * mod2;
const int inv_1 = Math::inv(mod1, mod2), inv_2 = Math::inv(mod_1_2 % mod3, mod3);
struct Int {
    int A, B, C;
    explicit inline Int() { }
    explicit inline Int(int __num) : A(__num), B(__num), C(__num) { }
    explicit inline Int(int __A, int __B, int __C) : A(__A), B(__B), C(__C) { }
    static inline Int reduce(const Int &x) {
        return Int(x.A + (x.A >> 31 & mod1), x.B + (x.B >> 31 & mod2), x.C + (x.C >> 31 &
        ↪  mod3));
    }
    inline friend Int operator + (const Int &lhs, const Int &rhs) {
        return reduce(Int(lhs.A + rhs.A - mod1, lhs.B + rhs.B - mod2, lhs.C + rhs.C -
        ↪  mod3));
    }
    inline friend Int operator - (const Int &lhs, const Int &rhs) {
        return reduce(Int(lhs.A - rhs.A, lhs.B - rhs.B, lhs.C - rhs.C));
    }
    inline friend Int operator * (const Int &lhs, const Int &rhs) {
        return Int(static_cast<long long> (lhs.A) * rhs.A % mod1, static_cast<long long>
        ↪  (lhs.B) * rhs.B % mod2, static_cast<long long> (lhs.C) * rhs.C % mod3);
    }
    inline int get() {
        long long x = static_cast<long long> (B - A + mod2) % mod2 * inv_1 % mod2 * mod1
        ↪  + A;
        return (static_cast<long long> (C - x % mod3 + mod3) % mod3 * inv_2 % mod3 *
        ↪  (mod_1_2 % mod) % mod + x) % mod;
    }
} ;

#define maxn 131072

namespace Poly {
#define N (maxn << 1)
```

```
    int lim, s, rev[N];
    Int Wn[N | 1];
    inline void init(int n) {
        s = -1, lim = 1; while (lim < n) lim <<= 1, ++s;
        for (register int i = 1; i < lim; ++i) rev[i] = rev[i >> 1] >> 1 | (i & 1) << s;
        const Int t(Math::pw(G, (mod1 - 1) / lim, mod1), Math::pw(G, (mod2 - 1) / lim,
        ↪ mod2), Math::pw(G, (mod3 - 1) / lim, mod3));
        *Wn = Int(1); for (register Int *i = Wn; i != Wn + lim; ++i) *(i + 1) = *i * t;
    }
    inline void NTT(Int *A, const int op = 1) {
        for (register int i = 1; i < lim; ++i) if (i < rev[i]) std::swap(A[i],
        ↪ A[rev[i]]);
        for (register int mid = 1; mid < lim; mid <<= 1) {
            const int t = lim / mid >> 1;
            for (register int i = 0; i < lim; i += mid << 1) {
                for (register int j = 0; j < mid; ++j) {
                    const Int W = op ? Wn[t * j] : Wn[lim - t * j];
                    const Int X = A[i + j], Y = A[i + j + mid] * W;
                    A[i + j] = X + Y, A[i + j + mid] = X - Y;
                }
            }
        }
        if (!op) {
            const Int ilim(Math::inv(lim, mod1), Math::inv(lim, mod2), Math::inv(lim,
            ↪ mod3));
            for (register Int *i = A; i != A + lim; ++i) *i = (*i) * ilim;
        }
    }
#undef N
}

int n, m;
Int A[maxn << 1], B[maxn << 1];
int main() {
    scanf("%d%d%d", &n, &m, &mod); ++n, ++m;
    for (int i = 0, x; i < n; ++i) scanf("%d", &x), A[i] = Int(x % mod);
    for (int i = 0, x; i < m; ++i) scanf("%d", &x), B[i] = Int(x % mod);
    Poly::init(n + m);
    Poly::NTT(A), Poly::NTT(B);
    for (int i = 0; i < Poly::lim; ++i) A[i] = A[i] * B[i];
    Poly::NTT(A, 0);
    for (int i = 0; i < n + m - 1; ++i) {
        printf("%d", A[i].get());
        putchar(i == n + m - 2 ? '\n' : ' ');
    }
    return 0;
}
```

6.19 FWT

```
/* FWT 快速沃尔什变换
给定 A,B 两个序列，可以通过 FWT 求出序列 C
其中 C(i) = A(j)B(k)  (j k = i)  可以为 or,and,xor
 */
```

```
const int maxn = 2e5 + 10;
const LL mod = 998244353;
int N,M,K;
LL inv2 = mod + 1 >> 1;
LL a1[maxn],b1[maxn],a2[maxn],b2[maxn],a3[maxn],b3[maxn];
inline LL add(LL a,LL b){return ((a + b) % mod + mod) % mod;}
inline LL mul(LL a,LL b){return (a % mod * b % mod + mod) % mod;}
void FWT_or(int limit,LL *a,int op){
    for(int i = 1; i < limit; i <<= 1){
        for(int p = i << 1,j = 0; j < limit; j += p){
            for(int k = 0; k < i ; k ++){
                if(op == 1) a[i + j + k] = add(a[j + k],a[i + j + k]);
                else a[i + j + k] = add(a[i + j + k],-a[j + k]);
            }
        }
    }
}
void FWT_and(int limit,LL *a,int op){
    for(int i = 1; i < limit; i <<= 1){
        for(int p = i << 1,j = 0; j < limit; j += p){
            for(int k = 0 ; k < i ; k ++){
                if(op == 1) a[j + k] = add(a[j + k],a[i + j + k]);
                else a[j + k] = add(a[j + k],-a[i + j + k]);
            }
        }
    }
}
void FWT_xor(int limit,LL *a,int op){
    for(int i = 1; i < limit; i <<= 1){
        for(int p = i << 1,j = 0;j < limit; j += p){
            for(int k = 0 ; k < i ; k ++){
                LL x = a[j + k],y = a[i + j + k];
                a[j + k] = add(x,y); a[i + j + k] = add(x,-y);
                if(op == -1) a[j + k] = mul(a[j + k],inv2),a[i + j + k] = mul(a[i + j +
                 ↪  k],inv2);
            }
        }
    }
}
int main(){
    Sca(N); N = (1 << N); N--;
    for(int i = 0; i <= N ; i ++) Scl(a1[i]),a2[i] = a3[i] = a1[i];
    for(int i = 0; i <= N ; i ++) Scl(b1[i]),b2[i] = b3[i] = b1[i];
    int limit = 1; while(limit <= N) limit <<= 1;
    FWT_or(limit,a1,1); FWT_or(limit,b1,1);
    for(int i = 0 ; i < limit; i ++) a1[i] = mul(a1[i],b1[i]);
    FWT_or(limit,a1,-1);

    FWT_and(limit,a2,1); FWT_and(limit,b2,1);
    for(int i = 0 ; i < limit; i ++) a2[i] = mul(a2[i],b2[i]);
    FWT_and(limit,a2,-1);

    FWT_xor(limit,a3,1); FWT_xor(limit,b3,1);
```

```
    for(int i = 0 ; i < limit; i ++) a3[i] = mul(a3[i],b3[i]);
    FWT_xor(limit,a3,-1);

    for(int i = 0; i <= N ; i ++) printf("%lld%c",a1[i],i == N?'\n':' ');
    for(int i = 0; i <= N ; i ++) printf("%lld%c",a2[i],i == N?'\n':' ');
    for(int i = 0; i <= N ; i ++) printf("%lld%c",a3[i],i == N?'\n':' ');
    return 0;
}
```

6.20 反演定理

6.20.1 二项式反演

```
//二项式反演
/*
    f(n) = ΣC(n,i) * g(i) (0 <= i <= n)
->  g(n) = Σ((-1) ^ (n - i)) * C(n,i) * f(i) (0 <= i <= n)
 */
//模板：n 封信对应 n 个信封，求恰好全部装错了信封的方案数
//设 g(i) 为恰好装错了 i 个信封的个数，则装的全部种类 f(n) = g(1) * C(n,1) + g(2) *
↪  C(n,2)..... + g(n) * C(n,n)
//代入二项式反演公式，其中 f(n) = n!
int N,M,K;
LL fac[maxn];
int comb[30][30];
void init(){
    fac[0] = 1;
    for(int i = 1; i <= 20; i ++) fac[i] = fac[i - 1] * i;
    for(int i = 0 ; i <= 20; i ++){
        comb[i][0] = comb[i][i] = 1;
        for(int j = 1; j < i ; j ++){
            comb[i][j] = comb[i - 1][j] + comb[i - 1][j - 1];
        }
    }
}
int main(){
    init();
    while(~Sca(N)){
        LL ans = 0;
        for(int i = 0 ; i <= N ; i ++){
            LL p = comb[N][i] * fac[i];
            if((N - i) & 1) p = -p;
            ans += p;
        }
        Prl(ans);
    }
    return 0;
}
```

6.20.2 莫比乌斯反演

```
//莫比乌斯反演
/*
f(n) = Σg(d) 其中 (d|n)
->g(n) =  (d)f(n / d) 其中 (d|n)
```

```
或者描述为
f(n) = g(d) 其中 (n/d)
->g(n) = (d/n) * f(d) 其中 (n/d)
*/
/*
(d) 为莫比乌斯函数，计算方式为
(1) = 1
x = p1 * p2 * p3 ……*pk（x 由 k 个不同的质数组成）则 (x) = (-1)^k
其他情况， (x) = 0
(d) 的常见性质：对于任意正整数 n 有
1. (d) (d/n) 在 n = 1 的时候为 1，在 n > 1 的时候为 0
2. (d)/d = (n)/n ，其中 (d/n), (n) 为欧拉函数
 */
//线性筛求莫比乌斯函数 O(n)
const int maxn = 1e6 + 5;
int mu[maxn], vis[maxn], prime[maxn];
int tot;//用来记录 prime 的个数
void init(){
    mu[1] = 1;
    for(int i = 2; i < maxn; i ++){
        if(!vis[i]){
            prime[tot ++] = i;
            mu[i] = -1;
        }
        for(int j = 0; j < tot && i * prime[j] < maxn; j ++){
            vis[i * prime[j]] = 1;
            if(i % prime[j]) mu[i * prime[j]] = -mu[i];
            else{
                mu[i * prime[j]] = 0;
                break;
            }
        }
    }
}
int main(){
    init();
}
```

6.21 数的位数公式

```
//求一个数 x 的位数的公式
//x 的位数 = log10(x) + 1
/*
    例题：洛谷 P2759，求 x ^ x 大于等于 n 位数的最小 x
    将 log(x ^ x) 化为 x * log(x)，二分查找即可
*/
LL N;
bool check(LL x){
    return x * log10(x) + 1 >= N;
}
LL solve(){
    LL l = 0,r = N;
    LL ans = 0;
    while(l <= r){
```

```
        LL m = (l + r) >> 1;
        if(check(m)){
            r = m - 1;
            ans = m;
        }else{
            l = m + 1;
        }
    }
    return ans;
}
int main(){
    Scl(N);
    Prl(solve());
    return 0;
}
```

6.22 辛普森积分

```
double simpson(double l, double r){
    double mid = (l+r)/2;
    return (f(l)+4*f(mid)+f(r))*(r-l)/6;
}


double jifen(double L, double R, double eps,double ST){
    //计算 [L,R] 上的定积分
    double mid = (L+R)/2;
    double SL = simpson(L,mid); //左半边
    double SR = simpson(mid,R); //右
    if(fabs(SL+SR-ST) <= 15*eps)
        return SL+SR+(SL+SR-ST)/15; //误差足够小，直接返回结果
    return jifen(L, mid, eps/2,SL) + jifen(mid, R, eps/2,SR); //继续分
```

6.23 矩阵快速幂

```
//矩阵快速幂，以对一个 3 * 3 的矩阵求快速幂为例
struct Mat
{
    LL a[3][3];
    void init(){
        Mem(a, 0);
    }
};
Mat operator * (Mat a, Mat b)
{
    Mat ans; ans.init();
    for(int i = 0 ; i <= 2; i ++)
        for(int j = 0; j <= 2; j ++)
            for(int k = 0 ; k <= 2; k ++)
                ans.a[i][j] = (ans.a[i][j] + a.a[i][k] * b.a[k][j]) % mod;
    return ans;
}
```

6.24 欧拉函数

```
//欧拉函数是求小于等于 n 的数中与 n 互质的数的数目
//表示为 phi(i) 或者 (i)
//求单个欧拉函数 事件负责度 sqrt(n)
//欧拉函数
int phi(int x){
    int ans = x;
    for(int i = 2; i*i <= x; i++){
        if(x % i == 0){
            ans = ans / i * (i-1);
            while(x % i == 0) x /= i;
        }
    }
    if(x > 1) ans = ans / x * (x-1);
    return ans;
}
//求连续的欧拉函数
const int N = 1e6+10 ;
int phi[N], prime[N];
int tot;//tot 计数, 表示 prime[N] 中有多少质数
void Euler(){
    phi[1] = 1;
    for(int i = 2; i < N; i ++){
        if(!phi[i]){
            phi[i] = i-1;
            prime[tot ++] = i;
        }
        for(int j = 0; j < tot && 1ll*i*prime[j] < N; j ++){
            if(i % prime[j]) phi[i * prime[j]] = phi[i] * (prime[j]-1);
            else{
                phi[i * prime[j] ] = phi[i] * prime[j];
                break;
            }
        }
    }
}
int main(){
    Euler();
}
```

6.25 欧拉定理

6.25.1 ap 互质

$$a^b\%p = (a\%p)^{b\%\Phi(p)}\%p \tag{1}$$

6.25.2 扩展欧拉定理 ap 不互质

$$a^b\%p = (a\%p)^{\Phi(p)+b\%\Phi(p)}\%p(b >= \Phi(p)) \tag{2}$$

$$a^b\%p = (a\%p)^b\%p(b < \Phi(p)) \tag{3}$$

6.26 常用公式

自然数幂次和公式

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

$\sum_{i=1}^{n} i^2 = C_{n+1}^{2} + 2C_{n+1}^{3} = \frac{n(n+1)(2n+1)}{6}$

$\sum_{i=1}^{n} i^3 = C_{n+1}^{2} + 6C_{n+1}^{3} + 6C_{n+1}^{4} = \frac{n^2(n+1)^2}{4}$

$\sum_{i=1}^{n} i^4 = C_{n+1}^{2} + 14C_{n+1}^{3} + 36C_{n+1}^{4} + 24C_{n+1}^{5} = \frac{n(n+1)(6n^3+9n^2+n-1)}{30}$

$\sum_{i=1}^{n} i^5 = C_{n+1}^{2} + 30C_{n+1}^{3} + 150C_{n+1}^{4} + 240C_{n+1}^{5} + 120C_{n+1}^{6} = \frac{n^2(n+1)(2n^3+4n^2+n-1)}{12}$

$\sum_{i=1}^{n} i^p = \sum_{k=1}^{p} \left[\sum_{j=0}^{k-1} (-1)^j C_k^j (k-j)^{p+1}\right] C_{n+1}^{k+1}$

斯特林公式

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

皮克定理

2S = 2a + b - 2

S: 多边形面积,a: 多边形内部点数,b: 多边形边上点数

7 计算几何

7.1 计算几何

```
#include<cmath>
#include<queue>
#include<cstdio>
#include<cctype>
#include<vector>
#include<cstring>
#include<iostream>
#include<algorithm>
using namespace std;
typedef double db;
const db pi=acos(-1),eps=1e-10;
struct Point
{
    db x,y;
    Point(db x0=0,db y0=0) : x(x0) , y(y0) { }
    friend bool operator<(Point a,Point b)
    {
        return a.x!=b.x?a.x<b.x:a.y<b.y;
    }
    friend Point operator+(const Point &a,const Point &b)
    {
        return Point(a.x+b.x,a.y+b.y);
    }
    friend Point operator-(const Point &a,const Point &b)
    {
        return Point(a.x-b.x,a.y-b.y);
    }
    friend Point operator*(const Point &a,double b)
    {
        return Point(a.x*b,a.y*b);
    }
    friend Point operator/(const Point &a,double b)
    {
        return Point(a.x/b,a.y/b);
    }
}a,b,c,ans;
typedef Point Vector;
db dcmp(db x)//符号判断
{
    if(fabs(x)<eps)return 0;
    else return x<0?-1:1;
}
db Dot(const Point &a,const Point &b)//点积
{
    return a.x*b.x+a.y*b.y;
}
db Cross(const Point &a,const Point &b)//叉积
{
    return a.x*b.y-a.y*b.x;
}
```

```
db Length(const Point &a)//求向量的长度
{
    return sqrt(Dot(a,a));
}
db Angle(const Point &a,const Point &b)//求两个向量的夹角（余弦定理）
{
    return acos(Dot(a,b)/Length(a)/Length(b));
}
Point Rotate(const Point &a,double rad)//逆时针旋转 rad
{
    return Point(a.x*cos(rad)-a.y*sin(rad),a.x*sin(rad)+a.y*cos(rad));
}
Point Normal(Point &v)//求单位长度的法向量
{
    db L=Length(v);
    return Point(-v.y/L,v.x/L);
}
Point getLineIntersection(const Point &P,const Point &v,const Point &Q,const Point
↪ &w)//求两个线段交点
{
    Vector u=P-Q;
    db t=Cross(w,u)/Cross(v,w);
    return P+v*t;
}
bool SegmentProperIntersection(const Point &a1,const Point &b1,const Point &a2,const
↪ Point &b2)//两线段规范相交、即每条线段的端点分别在另一条一段的两侧
{
    db c1=Cross(b1-a1,a2-a1),c2=Cross(b1-a1,b2-a1);
    db c3=Cross(b2-a2,a1-a2),c4=Cross(b2-a2,b1-a2);
    return dcmp(c1)*dcmp(c2)<0 && dcmp(c3)*dcmp(c4)<0;
}
bool IsPointOnSegment(Point p,Point a1,Point a2)//点在线段上
{
    return dcmp(Cross(p-a1,p-a2))==0 && dcmp(Dot(p-a1,p-a2))<0;
}
double getDistToLine(const Point &P, const Point &A, const Point &B)//点到直线的有向距离
↪ （距离加绝对值）
{
    Vector AB = B-A, AP = P-A;
    return Cross(AB, AP) / Length(AB);
}
int ConvexHull(Point *p,int n,Point *ch)//构造逆时针凸包
{
    sort(p+1,p+n+1);//先按照横坐标再按照纵坐标排序
    int m=0;
    for(int i=1;i<=n;i++)
    {
        while(m>1 && Cross(ch[m]-ch[m-1],p[i]-ch[m-1])<=0)m--;
        ch[++m]=p[i];
    }
    int k=m;
    for(int i=n-1;i;i--)
    {
```

```
        while(m>k && Cross(ch[m]-ch[m-1],p[i]-ch[m-1])<=0)m--;
        ch[++m]=p[i];
    }
    if(n>1)m--;
    return m;
}
db PolygonArea(Point *p,int n)//求逆时针构成的多边形（可不凸）面积
{
    db ret=0;
    for(int i=2;i<n;i++)//第一个点是 p[1]，所以这样循环
        ret+=Cross(p[i]-p[1],p[i+1]-p[1]);
    return ret/2;
}
bool isPointInPolygon(Point p,Point *poly,int n)//点在凸多边形内的判定
{
    int wn=0;
    poly[n+1]=poly[1];
    for(int i=1;i<=n;i++)
    {
        if(IsPointOnSegment(p,poly[i],poly[i+1]))return -1;
        int k=dcmp(Cross(poly[i+1]-poly[i],p-poly[i]));
        int d1=dcmp(poly[i].y-p.y);
        int d2=dcmp(poly[i+1].y-p.y);
        if(k>0 && d1<=0 && d2>0)wn++;
        if(k<0 && d2<=0 && d1>0)wn--;
    }
    if(wn!=0)return 1;
    return 0;
}
db dis2(Point a,Point b)
{
    return (b.x-a.x)*(b.x-a.x)+(b.y-a.y)*(b.y-a.y);
}
db findDiameter(Point *ch,int m)//旋转卡壳求直径（Diatmeter：直径）
{
    if(m==2)    return dis2(ch[1],ch[2]);
    ch[m+1]=ch[1];
    db rec=0;
    int j=3;
    for(int i=1;i<=m;i++)
    {
        while(Cross(ch[i+1]-ch[i],ch[j]-ch[i])<Cross(ch[i+1]-ch[i],ch[j+1]-ch[i]))
        {
            j=j%m+1;
        }
        rec=max(rec,max(dis2(ch[j],ch[i]),dis2(ch[j],ch[i+1])));
    }
    return rec;
}
struct Line
{
    Point P;
    Vector v;
```

```
    db ang;
    Line() {}
    Line(Point P,Vector v):P(P),v(v){ang=atan2(v.y,v.x);}
    friend bool operator<(Line a,Line b)
    {
        return  a.ang<b.ang;
    }
};
Point GetIntersection(Line a,Line b)
{
    Vector u=a.P-b.P;
    db t=Cross(b.v,u)/Cross(a.v,b.v);
    return a.P+a.v*t;
}
bool OnLeft(Line L,Point p)
{
    return Cross(L.v,p-L.P)>=0;
}
int HalfplaneIntersection(Line *L,int n,Point *poly)//半平面交
{
    sort(L+1,L+n+1);
    int first,last;
    Point *p=new Point[n+10];
    Line *q=new Line[n+10];
    q[first=last=0]=L[1];
    for(int i=2;i<=n;i++)
    {
        while(first<last && !OnLeft(L[i],p[last-1]))last--;
        while(first<last && !OnLeft(L[i],p[first]))first++;
        q[++last]=L[i];
        if(fabs(Cross(q[last].v,q[last-1].v))<eps)
        {
            last--;
            if(OnLeft(q[last],L[i].P))q[last]=L[i];
        }
        if(first<last)p[last-1]=GetIntersection(q[last-1],q[last]);
    }
    while(first<last && !OnLeft(q[first],p[last-1]))last--;
    if(last-first<=1)return 0;
    p[last]=GetIntersection(q[last],q[first]);
    int m=0;
    for(int i=first;i<=last;i++)poly[++m]=p[i];
    return m;
}


//湘潭大学 G 题 HDU 6538
#include<iostream>
#include<cstring>
#include<cstdio>
#include<algorithm>
using namespace std;
```

```
struct Point
{
    long long x,y;
    Point(long long x=0,long long y=0):x(x),y(y){}
    Point operator - (const Point &b) const {return Point(x-b.x,y-b.y);}
    bool operator < (const Point &b) const {if(x==b.x)return y>b.y;return x<b.x;}
};

const int maxn=2005;

Point p[maxn];

struct Line
{
    int a,b;
    double k;
};


Line l[maxn*maxn+5];
int order[maxn],pos[maxn];
long long S;

bool cmp(Line a,Line b)
{
    return a.k<b.k;
}

long long Cross(Point a,Point b)
{
    return a.x*b.y-b.x*a.y;
}

long long S_2(int a,int b,int c)
{
    return abs(Cross(p[a]-p[b],p[a]-p[c]));
}


int main()
{
    int n,m;
    while(cin>>n)
    {
        for(int i=0;i<n;i++)
        {
            scanf("%lld %lld",&p[i].x,&p[i].y);
            order[i]=i;
            pos[i]=i;
        }
        m=0;
        sort(p,p+n);
        for(int i=0;i<n;i++)
```

```
        {
            for(int j=i+1;j<n;j++)
            {
                l[m].a=i;
                l[m].b=j;
                if(p[i].x==p[j].x)  l[m].k=-1e20;
                else    l[m].k=1.0*(p[j].y-p[i].y)/(p[j].x-p[i].x);
                m++;
            }
        }
        sort(l,l+m,cmp);
        long long MAX=-1,MIN=1e15;
        for(int i=0;i<m;i++)
        {
            int a=l[i].a,b=l[i].b;
            swap(pos[a],pos[b]);
            swap(order[pos[a]],order[pos[b]]);
            int t1=pos[a],t2=pos[b];

            if(t1>t2)   swap(t1,t2);
            if(t1!=0&&t2!=n-1)
            {
                MAX=max(MAX,S_2(a,b,order[0])+S_2(a,b,order[n-1]));
                MIN=min(MIN,S_2(a,b,order[t1-1])+S_2(a,b,order[t2+1]));
            }
        }
        printf("%lld %lld\n",MIN,MAX);
    }
    return 0;
}
```

7.2 极角排序

```
//极角排序
struct Point
{
        __int128 x,y;
        int Q;
        Point(__int128 x=0,__int128 y=0):x(x),y(y)
        {
                if(x>0&&y>=0)         Q=1;
                else if(x<=0&&y>0)          Q=2;
                else if(x<0&&y<=0)          Q=3;
                else if(x>=0&&y<0)          Q=4;
        }
        Point operator - (const Point &b) const {return Point(x-b.x,y-b.y); }
        bool operator < (const Point &b) const
        {
                if(Q==b.Q)          return x*b.y-b.x*y>0;
                return Q<b.Q;
        }
};
```

7.3 凸包

```
//凸包
struct Point{
    double x,y;
    Point(){}
    Point(double x,double y):x(x),y(y){}
    Point operator - (Point a){
        return Point(x - a.x,y - a.y);
    }
}point[maxn];
bool cmp(Point a,Point b){
    return a.x != b.x?a.x < b.x:a.y < b.y;
}
Point Stack[maxn];
double Cross(Point a,Point b){
    return a.x * b.y - a.y * b.x;
}
int ConvexHull(Point* p,int N,Point* S){
    int cnt = 0;
    for(int i = 1; i <= N ; i ++){
        while(cnt >= 2 && Cross(S[cnt] - S[cnt - 1],p[i] - S[cnt - 1]) <= 0) cnt--; // <
        ↪ 就包括凸包上的点, <= 不包括凸包边上的点
        S[++cnt] = p[i];
    }
    int k = cnt;
    for(int i = N; i >= 1; i --){
        while(cnt >= k + 2 && Cross(S[cnt] - S[cnt - 1],p[i] - S[cnt - 1]) <= 0) cnt--;
        S[++cnt] = p[i];
    }
    if(N > 1) cnt--;
    return cnt;
}
int main(){
    Sca(N);
    for(int i = 1; i <= N ; i ++) scanf("%lf%lf",&point[i].x,&point[i].y);
    sort(point + 1,point + 1 + N,cmp);
    printf("%d",ConvexHull(point,N,Stack));
    return 0;
}
```

8 其他

8.1 STL

8.1.1 multiset

```
//mulitiset
mulitiset<int>c;
c.size()     //返回当前的元素数量
c.empty ()   //判断大小是否为零, 等同于 0 == size(), 效率更高
count (elem) //返回元素值为 elem 的个数
find(elem)   //返回元素值为 elem 的第一个元素, 如果没有返回 end()
lower _bound(elem)  //返回元素值为 elem 的第一个可安插位置, 也就是元素值 >= elem 的第一个元
↪ 素位置
```

```
upper _bound (elem) //返回元素值为 elem 的最后一个可安插位置，也就是元素值 > elem 的第一个
↪ 元素位置
c.begin()  //返回一个随机存取迭代器，指向第一个元素
c.end()    //返回一个随机存取迭代器，指向最后一个元素的下一个位置
c.rbegin() //返回一个逆向迭代器，指向逆向迭代的第一个元素
c.rend()   //返回一个逆向迭代器，指向逆向迭代的最后一个元素的下一个位置
c.insert(elem) //插入一个 elem 副本，返回新元素位置，无论插入成功与否。
c.insert(pos, elem) //安插一个 elem 元素副本，返回新元素位置，pos 为收索起点，提升插入速度。
c.insert(beg,end) //将区间 [beg,end) 所有的元素安插到 c，无返回值。
c.erase(elem) //删除与 elem 相等的所有元素，返回被移除的元素个数。
c.erase(pos) //移除迭代器 pos 所指位置元素，无返回值。
c.erase(beg,end) //移除区间 [beg,end) 所有元素，无返回值。
c.clear() //移除所有元素，将容器清空
//注意事项:mulitiset 是在 set 的基础上允许重复元素，如果需要删除某个值 x 的其中一个元素，不
↪ 能通过 c.erase(3)，这样会删除所有值为 3 的元素
//而是 it = c.find(3); c.erase(it);
```

8.1.2 bitset

```
//bitset<maxn>a
/*
bitset 是一种专门用来储存二进制的数组，使用前要先调用函数库。
他的每一个元素只占 1 bit 空间，你可以将它当作 bool 类型的高精度。
他的优点很多，你可将他整体使用，也可单个访问
bitset 的原理大概是将很多数压成一个，从而节省空间和时间（暴力出奇迹）
一般来说 bitset 会让你的算法复杂度 /32
 */
//构造函数
bitset<4> bitset1;  //无参构造，长度为 4，默认每一位为 0
 bitset<8> bitset2(12);  //长度为 8，二进制保存，前面用 0 补充
string s = "100101";
bitset<10> bitset3(s);  //长度为 10，前面用 0 补充
char s2[] = "10101";
bitset<13> bitset4(s2);  //长度为 13，前面用 0 补充
cout << bitset1 << endl;  //0000
cout << bitset2 << endl;  //00001100
cout << bitset3 << endl;  //0000100101
cout << bitset4 << endl;  //0000000010101

bitset<2> bitset1(12);  //12 的二进制为 1100（长度为 4），但 bitset1 的 size=2，只取后面部分，
↪ 即 00
string s = "100101";
bitset<4> bitset2(s);  //s 的 size=6，而 bitset 的 size=4，只取前面部分，即 1001
char s2[] = "11101";
bitset<4> bitset3(s2);  //与 bitset2 同理，只取前面部分，即 1110
cout << bitset1 << endl;  //00
cout << bitset2 << endl;  //1001
cout << bitset3 << endl;  //1110

//可用操作符
bitset<4> foo (string("1001"));
bitset<4> bar (string("0011"));
cout << (foo^=bar) << endl;       // 1010 (foo 对 bar 按位异或后赋值给 foo)
cout << (foo&=bar) << endl;       // 0010 (按位与后赋值给 foo)
```

```
cout << (foo|=bar) << endl;        // 0011 (按位或后赋值给 foo)
cout << (foo<<=2) << endl;         // 1100 (左移2位，低位补0，有自身赋值)
cout << (foo>>=1) << endl;         // 0110 (右移1位，高位补0，有自身赋值)
cout << (~bar) << endl;            // 1100 (按位取反)
cout << (bar<<1) << endl;          // 0110 (左移，不赋值)
cout << (bar>>1) << endl;          // 0001 (右移，不赋值)
cout << (foo==bar) << endl;        // false (0110==0011 为 false)
cout << (foo!=bar) << endl;        // true  (0110!=0011 为 true)
cout << (foo&bar) << endl;         // 0010 (按位与，不赋值)
cout << (foo|bar) << endl;         // 0111 (按位或，不赋值)
cout << (foo^bar) << endl;         // 0101 (按位异或，不赋值)

//下标访问
bitset<4> foo ("1011");
cout << foo[0] << endl;  //1
cout << foo[1] << endl;  //1
cout << foo[2] << endl;  //0

//支持的函数
bitset<8> foo ("10011011");
cout << foo.count() << endl;  //5      (count 函数用来求 bitset 中 1 的位数，foo 中共有5个
↪  1
cout << foo.size() << endl;   //8      (size 函数用来求 bitset 的大小，一共有8位
cout << foo.test(0) << endl;  //true      (test 函数用来查下标处的元素是0还是1，并返回
↪  false 或 true，此处 foo[0] 为1，返回 true
cout << foo.test(2) << endl;  //false      (同理，foo[2] 为0，返回 false
cout << foo.any() << endl;    //true      (any 函数检查 bitset 中是否有1
cout << foo.none() << endl;   //false      (none 函数检查 bitset 中是否没有1
cout << foo.all() << endl;    //false      (all 函数检查 bitset 中是全部为1

//操作函数，相比于运用下标，这些会检查有没有越界
bitset<8> foo ("10011011");
cout << foo.flip(2) << endl;  //10011111      (flip 函数传参数时，用于将参数位取反，本行代
↪  码将 foo 下标2处"反转"，即0变1，1变0
cout << foo.flip() << endl;   //01100000      (flip 函数不指定参数时，将 bitset 每一位全部
↪  取反
cout << foo.set() << endl;    //11111111      (set 函数不指定参数时，将 bitset 的每一位
↪  全部置为1
cout << foo.set(3,0) << endl;  //11110111      (set 函数指定两位参数时，将第一参数位的元素
↪  置为第二参数的值，本行对 foo 的操作相当于 foo[3]=0
cout << foo.set(3) << endl;    //11111111      (set 函数只有一个参数时，将参数下标处置为1
cout << foo.reset(4) << endl;  //11101111      (reset 函数传一个参数时将参数下标处置为0
cout << foo.reset() << endl;   //00000000      (reset 函数不传参数时将 bitset 的每一位全部
↪  置为0

//转换类型
bitset<8> foo ("10011011");
string s = foo.to_string();  //将 bitset 转换成 string 类型
unsigned long a = foo.to_ulong();  //将 bitset 转换成 unsigned long 类型
unsigned long long b = foo.to_ullong();  //将 bitset 转换成 unsigned long long 类型
cout << s << endl;  //10011011
cout << a << endl;  //155
cout << b << endl;  //155
```

8.2 整数三分

```
//整数三分
int solve(int x)
{
    int r = N, l = x;
    while (l < r - 1){
        int m = (l + r) >> 1;
        int mm = (r + m) >> 1;
        if (check(m, x) > check(mm, x)){
            l = m;
        }
        else{
            r = mm;
        }
    }
    return min(check(l, x), check(r, x));
}
```

8.3 表达式树

```
//表达式树
//我觉得说不定有一天会用到他
typedef pair<int, int>PII;

const int MX = 3e5 + 5;

char S[MX];
int LT[MX], RT[MX], rear, n;
LL A[MX];
string op[MX];
string last[10];

int build(int L, int R) {
    int cur = L, u;
    LL tmp = 0;
    while(cur <= R && isdigit(S[cur])) tmp = tmp * 10 + S[cur] - '0', cur++;
    if(cur == R + 1) {
        u = rear++;
        op[u] = ".";
        A[u] = tmp;
        return u;
    }

    int p[10], cnt = 0;
    for(int i = 1; i <= 9; i++) p[i] = -1;
    while(cur <= R) {
        string oper = "";
        if(!isdigit(S[cur])) {
            oper += S[cur];
            if(cur + 1 <= R) {
                if(S[cur] == '>' && S[cur + 1] == '=') oper += S[++cur];
                if(S[cur] == '<' && S[cur + 1] == '=') oper += S[++cur];
                if(S[cur] == '=' && S[cur + 1] == '=') oper += S[++cur];
```

```
                if(S[cur] == '!' && S[cur + 1] == '=') oper += S[++cur];
                if(S[cur] == '&' && S[cur + 1] == '&') oper += S[++cur];
                if(S[cur] == '|' && S[cur + 1] == '|') oper += S[++cur];
            }
        }

        if(oper == "(") cnt++;
        else if(oper == ")") cnt--;
        else if(cnt);
        else if(oper == "*" || oper == "/" || oper == "%") last[1] = oper, p[1] = cur;
        else if(oper == "+" || oper == "-") last[2] = oper, p[2] = cur;
        else if(oper == ">" || oper == ">=" || oper == "<" || oper == "<=") last[3] =
        ↪  oper, p[3] = cur;
        else if(oper == "==" || oper == "!=") last[4] = oper, p[4] = cur;
        else if(oper == "&") last[5] = oper, p[5] = cur;
        else if(oper == "^") last[6] = oper, p[6] = cur;
        else if(oper == "|") last[7] = oper, p[7] = cur;
        else if(oper == "&&") last[8] = oper, p[8] = cur;
        else if(oper == "||") last[9] = oper, p[9] = cur;

        cur++;
    }
    int w = -1, wi;
    for(int i = 9; i >= 1; i--) {
        if(p[i] != -1) {
            w = p[i];
            wi = i;
            break;
        }
    }
    if(w == -1) u = build(L + 1, R - 1);
    else {
        u = rear++;
        op[u] = last[wi];
        LT[u] = build(L, w - op[u].length());
        RT[u] = build(w + 1, R);
    }
    return u;
}

LL solve(int u) {
    if(op[u] == ".") return A[u];
    LL l = solve(LT[u]), r = solve(RT[u]);
    if(op[u] == "*") return l * r;
    if(op[u] == "/") return l / r;
    if(op[u] == "%") return l % r;
    if(op[u] == "+") return l + r;
    if(op[u] == "-") return l - r;
    if(op[u] == ">") return l > r;
    if(op[u] == ">=") return l >= r;
    if(op[u] == "<") return l < r;
    if(op[u] == "<=") return l <= r;
    if(op[u] == "==") return l == r;
```

```
    if(op[u] == "!=") return l != r;
    if(op[u] == "&") return l & r;
    if(op[u] == "^") return l ^ r;
    if(op[u] == "|") return l | r;
    if(op[u] == "&&") return l && r;
    if(op[u] == "||") return l || r;
}

int main() {
    //FIN;
    while(~scanf("%s", S)) {
        n = strlen(S);
        if(S[0] == '0' && n == 1) break;

        rear = 0;
        build(0, n - 1);
        printf("%lld\n", solve(0));
    }
    return 0;
}
```

8.4 切比雪夫距离

```
/* 曼哈顿距离和切比雪夫距离
曼哈顿距离：两点横纵坐标差之和 即 dis = |x1 - x2| + |y1 - y2|;
切比雪夫距离：两点横纵坐标差的最大值 即 dis = max(|x1 - x2|,|y1 -y2|);
两距离可以相互转化
将一个点 (x,y) 的坐标变为 (x+y,x-y) 后，原坐标系中的曼哈顿距离 = 新坐标系中的切比雪夫距离
将一个点 (x,y) 的坐标变为 ((x+y)/2,(x-y)/2) 后，原坐标系中的切比雪夫距离 = 新坐标系中的曼哈
↪ 顿距离
用处：
1. 切比雪夫距离转曼哈顿距离
切比雪夫距离在计算的时候需要取 max，往往不是很好优化，对于一个点，计算其他点到该的距离的复杂度
↪ 为 O(n)
而曼哈顿距离只有求和以及取绝对值两种运算，我们把坐标排序后可以去掉绝对值的影响，进而用前缀和优
↪ 化，可以把复杂度降为 O(1)
2. 曼哈顿距离转切比雪夫距离
例如维护区间两点之间最大的曼哈顿距离，可以转为切比雪夫距离，分别维护区间 x,y 的最大（小）值，
↪ 求他们之间的差再取 x,y 中较大的值就是原坐标中的最大的曼哈顿距离
 */
//模板：给一堆点，求一个点使得他到其他所有点的切比雪夫距离最小
//做法：转曼哈顿距离之后排序维护前缀后缀 x,y 的和
const int maxn = 1e5 + 10;
const int INF = 0x3f3f3f3f;
const int mod = 1e9 + 7;
int N,M,K;
struct Node{
    LL x,y;
    int id;
}node[maxn];
bool cmp(Node a,Node b){return a.x < b.x;}
bool cmp2(Node a,Node b){return a.y < b.y;}
LL ans[maxn];
int main(){
```

```
    Sca(N);
    for(int i = 1; i <= N ; i ++){
        int x,y; Sca2(x,y);
        node[i].x = x + y; node[i].y = x - y;
        node[i].id = i;
    }
    sort(node + 1,node + 1 + N,cmp);
    LL pre = 0;
    for(int i = 1; i <= N ; i ++){
        ans[node[i].id] += (i - 1) * node[i].x - pre;
        pre += node[i].x;
    }
    pre = 0;
    for(int i = N ; i >= 1; i --){
        ans[node[i].id] += pre - (N - i) * node[i].x;
        pre += node[i].x;
    }
    sort(node + 1,node + 1 + N,cmp2);
    pre = 0;
    for(int i = 1; i <= N ; i ++){
        ans[node[i].id] += (i - 1) * node[i].y - pre;
        pre += node[i].y;
    }
    pre = 0;
    for(int i = N ; i >= 1; i --){
        ans[node[i].id] += pre - (N - i) * node[i].y;
        pre += node[i].y;
    }
    LL Min = 1e18;
    for(int i = 1; i <= N ; i ++) Min = min(Min,ans[i]);
    Min /= 2;
    Prl(Min);
    return 0;
}
```

8.5 离散化

```
//离散化，模板：离散 N 条线段
int Hash[maxn * 2];
int cnt = 0;
for(int i = 1; i <= N; i ++){
    line[i].fi = read(); line[i].se = read();
    Hash[++cnt] = line[i].fi; Hash[++cnt] = line[i].se;
}
sort(Hash + 1,Hash + 1 + cnt);
cnt = unique(Hash + 1,Hash + 1 + cnt) - Hash - 1;
for(int i = 1; i <= N ; i ++){
    line[i].fi = lower_bound(Hash + 1,Hash + 1 + cnt,line[i].fi) - Hash;
    line[i].se = lower_bound(Hash + 1,Hash + 1 + cnt,line[i].se) - Hash;
}
```

8.6 区间离散化

```
//区间离散化
/*
    涉及到区间修改的离散化是和普通离散化不同的
    假设修改 20 - 30, 查询 10 - 40
    常规的离散化变成 1, 2, 3, 4 之后线段树修改 2,3 两个点, 询问 1-4 的 sum
    但是修改的时候会分开修改 2 和 3, 这和我们想要的不一样
    因为离散化区间是不能直接相加的, [2,2] + [3,3] = [2,3] 是没毛病, [20,20] + [30,30] =
↪   [20,30] 就有问题了
    所以我们希望他的区间可以直接相加
    寻常的线段树是左右都闭, 我们需要把它转化为左闭右开的区间
    让他变为 [2,3) + [3,4) = [2,4)  => [20,30) + [30,31) = [20,31)
    所以说, 我们离散化的时候需要离散的是 l 和 r + 1 而不是寻常的 l,r
 */
LL read(){LL x = 0,f = 1;char c = getchar();while (c<'0' || c>'9'){if (c == '-') f = -1;c
↪   = getchar();}
while (c >= '0'&&c <= '9'){x = x * 10 + c - '0';c = getchar();}return x*f;}
const double eps = 1e-9;
const int maxn = 8e5 + 10;
LL N;
int M;
struct Query{
    int op;
    LL a,b,c;
}query[maxn];
LL Hash[maxn * 2];
struct Tree{
    int l,r;
    ULL sum,lazy;
}tree[maxn * 4];
void Build(int t,int l,int r){
    tree[t].l = l; tree[t].r = r;
    tree[t].sum = tree[t].lazy = 0;
    if(r - l <= 1) return;
    int m = l + r >> 1;
    Build(t << 1,l,m);
    Build(t << 1 | 1,m,r);
}
void Pushup(int t){
    tree[t].sum = tree[t << 1].sum + tree[t << 1 | 1].sum;
}
void solve(int t,ULL p){
    tree[t].lazy = p + tree[t].lazy;
    tree[t].sum = tree[t].sum + ((ULL)Hash[tree[t].r] - (ULL)Hash[tree[t].l]) * p;
}
void Pushdown(int t){
    if(tree[t].lazy){
        solve(t << 1,tree[t].lazy);
        solve(t << 1 | 1,tree[t].lazy);
        tree[t].lazy = 0;
    }
}
void update(int t,int l,int r,ULL p){
```

```
    if(l <= tree[t].l && tree[t].r <= r){
        solve(t,p);
        return;
    }
    Pushdown(t);
    int m = (tree[t].l + tree[t].r) >> 1;
    if(r <= m) update(t << 1,l,r,p);
    else if(l >= m) update(t << 1 | 1,l,r,p);
    else{
        update(t << 1,l,m,p);
        update(t << 1 | 1,m,r,p);
    }
    Pushup(t);
}
ULL _query(int t,int l,int r){
    if(l <= tree[t].l && tree[t].r <= r){
        return tree[t].sum;
    }
    Pushdown(t);
    int m = (tree[t].l + tree[t].r) >> 1;
    if(r <= m) return _query(t << 1,l,r);
    else if(l >= m) return _query(t << 1 | 1,l,r);
    else{
        return _query(t << 1,l,m) + _query(t << 1 | 1,m,r);
    }
}
int main(){
    cin >> N >> M;
    int cnt = 0;
    for(int i = 1; i <= M; i ++){
        query[i].op = read();
        query[i].a = read();
        query[i].b = read();
        if(query[i].op == 1) query[i].c = read();
        Hash[++cnt] = query[i].a;
       // Hash[++cnt] = query[i].b;
        Hash[++cnt] = query[i].b + 1;
    }
    sort(Hash + 1,Hash + 1 + cnt);
    cnt = unique(Hash + 1,Hash + 1 + cnt) - Hash - 1;
    Build(1,1,cnt);
    for(int i = 1; i <= M; i ++){
        query[i].a = lower_bound(Hash + 1,Hash + 1 + cnt,query[i].a) - Hash;
        query[i].b = lower_bound(Hash + 1,Hash + 1 + cnt,query[i].b + 1) - Hash;
        if(query[i].op == 1) update(1,query[i].a,query[i].b,query[i].c);
        else{
            ULL ans = _query(1,query[i].a,query[i].b);
            cout << ans << endl;
        }
    }
    return 0;
}
```

8.7 模拟退火

```
/* 模拟退火
1. 是一种随机算法,AC 不 AC 靠脸的算法，如果不过可能需要多交几发证明确实这个算法过不了
2. 当我们知道一个状态以及一个状态下的答案的时候，我们可以随机变换这个状态的变量并且像退火一
↪  样，一开始变很多 (温度高)，
然后变得少 (温度低) 直到冷却，每次都向更优秀的状态靠近，到最后就有可能随机到我们想要的状态
3.T 温度,delta 温度系数,t > eps(1e-18) 是冷却点，这些参数决定了模拟退火的次数和精度还有时间
4. 带 exp 的那个公式是模拟退火的固定公式，这样的概率被证明是最优的
5. 这个算法在计算几何里面比较常用，例如最小圆覆盖 可以三分套三分再套三分也可以模拟退火
 */
/* 模板 1: 桌子上一个点被 N 个悬挂着 w 重量的小球拉着，求平衡点 (重力势能最低点)
当我们知道了点的时候我们可以很容易得计算这个点的重力势能 (energe)，所以我们模拟退火他的坐标
↪  点
 */
const int maxn = 2010;
int N,M,K;
struct Node{
    double x,y,w;
    Node(double x = 0,double y = 0,double w = 0):x(x),y(y),w(w){}
}node[maxn];
double energe(double x,double y){ //计算 x,y 点的势能
    double sum = 0;
    for(int i = 1; i <= N ;  i++){
        double dx = x - node[i].x,dy = y - node[i].y;
        sum += sqrt(dx * dx + dy * dy) * node[i].w;
    }
    return sum;
}
double xans,yans; //answer
double ans = 1e18 + 7; //势能，需要寻找最小值，初始为 INF
const double delta = 0.998;  //4. 降温系数可以调整，需要十分接近 1

void s_a(){   //模拟退火
    double xx = xans,yy = yans; //初始位置
    double t = 3000;   //3. 温度需要足够高
    while(t > 1e-18){  //5.eps 要低一点
        double xtmp = xx + (rand() * 2 - RAND_MAX) * t;
        double ytmp = yy + (rand() * 2 - RAND_MAX) * t;
        //随机一个新坐标，变化幅度为 t
        //rand() * 2 - RAND_MAX 的范围是-RAND_MAX - RANDMAX - 1
        double new_ans = energe(xtmp,ytmp);  //计算当前状态下的值
        double DE = new_ans - ans;
        if(DE < 0){ //更优解
            xans = xx = xtmp; yans = yy = ytmp;
            ans = new_ans;
        }else if(exp(-DE/t) * RAND_MAX > rand()){  //更差解的话随机是否接受
            xx = xtmp,yy = ytmp;
        }
        t *= delta;
    }
}

void SA(){   //2. 多跑几遍模拟退火
```

```
    s_a(); s_a();
    s_a(); s_a();
    s_a(); s_a();
}
int main(){
    Sca(N); srand(time(NULL));
    for(int i = 1; i <= N ; i ++){
        scanf("%lf%lf%lf",&node[i].x,&node[i].y,&node[i].w);
        xans += node[i].x; yans += node[i].y;
    }
    xans /= N; yans /= N;  //1. 取平均值作为初始值
    ans = energe(xans,yans);
    SA();
    printf("%.3lf %.3lf",xans,yans);
    return 0;
}

/* luoguP4360
模板 2：伐木场选址，一条数轴上选择两个点使得这条其他点到这条数轴的花费最小，花费计算以及在
↪ fun(a,b) 中搞定
模拟退火部分：随便选两个点作为数轴上的初始答案，然后模拟退火随机让两个点往左右方向移动，最终
↪ 选定出状态下
fun(a,b) 最小的 ansa,ansb
 */

LL w[maxn],d[maxn],sum;
LL fun(int& a,int& b){
    if(a > b) swap(a,b);
    return d[a] * w[a] + (w[b] - w[a]) * d[b] + d[N + 1] * (w[N] - w[b]) - sum;
}
int ansa,ansb,ans;
const double delta = 0.998; //退火系数
void s_a(){
    double t = 30000;  //温度
    int a = ansa,b = ansb;
    while(t > 1e-14){
        int atmp = a + (rand() * 2 - RAND_MAX) * t; atmp = (atmp % N + N) % N; atmp++;
        int btmp = a + (rand() * 2 - RAND_MAX) * t; btmp = (btmp % N + N) % N; btmp++;
        ↪ //两个点向两边随机移动
        LL new_ans = fun(btmp,atmp);
        LL DE = new_ans - ans;
        if(DE < 0){
            ansa = a = atmp; ansb = b = btmp;
            ans = new_ans;
        }else if(exp(-DE / t) * RAND_MAX > rand()){
            a = atmp; b = btmp;
        }
        t *= delta;
    }
}
void SA(){
    s_a();s_a();s_a();
    s_a();s_a();s_a();
```

```
}
int main(){
    Sca(N); srand(time(NULL));
    for(int i = 1; i <= N; i ++){
        Scl(w[i]); Scl(d[i + 1]);
        sum += w[i] * d[i];
        w[i] += w[i - 1];
        d[i + 1] += d[i];

    }
    ansa = 1,ansb = N; ans = fun(ansa,ansb);
    SA();
    Prl(fun(ansa,ansb));
    return 0;
}
```

8.8 java 大数

```
//!!!! 提交 JAVA 的时候一定要去掉 package
//创建大数类
import java.math.BigDecimal;
import java.math.BigInteger;
import java.util.Scanner;

        Scanner cin=new Scanner(System.in);

        BigInteger num1=new BigInteger("12345");
        BigInteger num2=cin.nextBigInteger();

        BigDecimal num3=new BigDecimal("123.45");
        BigDecimal num4=cin.nextBigDecimal();
//整数
import java.math.BigInteger;

public class Main {
    public static void main(String[] args) {
        BigInteger num1=new BigInteger("12345");
        BigInteger num2=new BigInteger("45");
        //加法
        System.out.println(num1.add(num2));
        //减法
        System.out.println(num1.subtract(num2));
        //乘法
        System.out.println(num1.multiply(num2));
        //除法 (相除取整)
        System.out.println(num1.divide(num2));
        //取余
        System.out.println(num1.mod(num2));
        //最大公约数 GCD
        System.out.println(num1.gcd(num2));
        //取绝对值
        System.out.println(num1.abs());
        //取反
        System.out.println(num1.negate());
```

```
        //取最大值
        System.out.println(num1.max(num2));
        //取最小值
        System.out.println(num1.min(num2));
        //是否相等
        System.out.println(num1.equals(num2));
    }
}
//浮点数
import java.math.BigDecimal;

public class Main {
    public static void main(String[] args) {
        BigDecimal num1=new BigDecimal("123.45");
        BigDecimal num2=new BigDecimal("4.5");
        //加法
        System.out.println(num1.add(num2));
        //减法
        System.out.println(num1.subtract(num2));
        //乘法
        System.out.println(num1.multiply(num2));
        //除法（在 divide 的时候就设置好要精确的小数位数和舍入模式）
        System.out.println(num1.divide(num2,10,BigDecimal.ROUND_HALF_DOWN));
        //取绝对值
        System.out.println(num1.abs());
        //取反
        System.out.println(num1.negate());
        //取最大值
        System.out.println(num1.max(num2));
        //取最小值
        System.out.println(num1.min(num2));
        //是否相等
        System.out.println(num1.equals(num2));
        //判断大小 ( > 返回 1, < 返回-1)
        System.out.println(num2.compareTo(num1));
    }
}
//hdu1715 求第 N 项斐波那契数
import java.util.Scanner;
import java.math.BigInteger;
public class Main {
    static BigInteger a[] = new BigInteger[1001];   //开全局变量的数组
    public static void main(String[] args) {
        Scanner cin = new Scanner(System.in);
        a[1] = a[2] = new BigInteger("1");   //大数赋值
        for(int i = 3; i <= 1000 ; i ++) a[i] = a[i - 1].add(a[i - 2]); //大数加
        while(cin.hasNext()) {             //读到文件尾
            int N = cin.nextInt();
            for(int i = 1; i <= N ; i ++) {
                int p = cin.nextInt();
                System.out.println(a[p]);
            }
        }
```

```
    }
}
//POJ1001 高精度幂 浮点数幂次 + 去掉前导后导 0
import java.math.BigDecimal;
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner cin = new Scanner(System.in);
        while(cin.hasNext()) {
            BigDecimal a = cin.nextBigDecimal();
            int t = cin.nextInt();
            a = a.pow(t);
            String ans = a.stripTrailingZeros().toPlainString();  //去掉结尾的 0
            if(ans.startsWith("0")) ans = ans.substring(1);  //去掉开头的 0
            System.out.println(ans);
        }
    }
}
//HDU1042 N!
import java.math.BigInteger;
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner cin = new Scanner(System.in);
        while(cin.hasNext()) {
            int x = cin.nextInt();
            BigInteger a = BigInteger.ONE;
            for(int i = 1; i <= x; i ++) a = a.multiply(BigInteger.valueOf(i));
            System.out.println(a);
        }
    }
}
//hdu1753 浮点数相加 去结尾 0
import java.util.Scanner;
import java.math.BigDecimal;
public class Main {
    public static void main(String[] args) {
        Scanner cin = new Scanner(System.in);
        while(cin.hasNext()) {
            BigDecimal a = cin.nextBigDecimal(),b = cin.nextBigDecimal();
            a = a.add(b);
            String ans = a.stripTrailingZeros().toPlainString();
            System.out.println(ans);
        }
    }
}
//hdu1023 大数卡特兰数
import java.util.Scanner;
import java.math.BigInteger;
public class Main {
    static BigInteger a[] = new BigInteger[110];
    public static void main(String[] args) {
        Scanner cin = new Scanner(System.in);
```

```
        a[1] = BigInteger.ONE;
        for(int i = 2; i <= 100; i ++) {
            a[i] = a[i - 1].multiply(BigInteger.valueOf(4 * i -
             ↪  2)).divide(BigInteger.valueOf(i + 1));
        }
        while(cin.hasNext()) {
            int t = cin.nextInt();
            System.out.println(a[t]);
        }
    }
}
```

8.9 注意事项

!!!!!!!!!!!

1. 注意精度问题, 尤其是过程中的精度问题，例如 int * int 的值大于 LL，即使赋值给 LL 也会炸例如 (1 « 50) 是有问题的，需要 (1LL « 50)

2. 在交题前一定要重新审视一边整个代码，不要心急交题, 记得要测两个小样例和极端样例

3. 初始化请务必 0 N+1,memset 容易被卡, 手动初始化容易遗漏边界 0 等问题, 以及 init 记得在主函数里调用，不要调用在 N 的输入之前

4. 读入挂可以尝试但不要迷信, 包括 inline 和 register 这样的玄学优化，更加多的去寻找算法本身的问题。

5. 在遇到莫名其妙不知所云的错误的时候（例如一个值好好的莫名其妙就在中途改变了），很有可能是数组溢出之类的错误

6. 如果在想 dp 状态转移方程的时候始终觉得状态太多存不下，找规律又不怎么找得到的时候，要记得有一那么一类题，当你枚举出起始或者初始的有限个状态的时候，可以递推出后面所有的结果，所以只要枚举 + 验证即可，例如知道了石头剪刀布的结果（是谁赢）就可以推出过程的两个状态

7. 如果 DP 完了之后要求记忆路径的操作很麻烦或者很容易出错，可以考虑在 DP 的过程中就记录当前状态的前驱

8. 如果为了压行去大括号，请务必看清楚大括号里有几句话!!!

9. 如果感觉被卡常了且函数里含有形如 a.a/a.b<b.a/b.b 之类的 cmp 函数，不妨将结构体的答案 a/b 直接计算出来比较，每次比较的时候都要计算一下会很费时间

10. 结构体排序建议把所有成员的关键字都设置好使得这个排序是稳定的，否则可能会造成奇怪的问题 (至于为什么，暂时还不知道)

11. 涉及到区间修改的离散化请查阅上面的离散化板子，需要左闭右开的离散化而不能跟个愣头青一样离散化