九野的模版 3.15.10@NIT

acm International Collegiate Programming Contest

Baidu IBM | event sponsor

共你乾杯再举箸 突然间相看莞尔 盘中透著那味儿
大概今生有些事 是提早都不可以 明白其妙处
就像你当日痛心她回绝一番美意
怎发现你从情劫亦能学懂开解与宽恕
也像我很纠结的公事 此际回头看 原来并没有事
真想不到当初我们也讨厌吃苦瓜
今天竟吃得出那睿智愈来愈记挂
开始时捱一些苦 栽种绝处的花
幸得艰辛的引路甜蜜不致太寡
青春的快餐只要求快不理哪一家
哪有玩味的空档来欣赏细致淡雅
到大悟大彻将虎咽的升华 等消化学沏茶
至共你觉得苦也不太差

图论 1
Lca 倍增法 1
强连通 1
边双连通 2
点双连通 3
2-sat 4
二分匹配 5
全局最小割 5
网络流 6
网络流 Dinic 6
网络流 Isap 7
费用流 Spfa 9
费用流 Zkw 9
KM 算法 11
最小生成树计数 12
最小树形图 13
哈密顿回路 14
欧拉通路 15
二维平面最小曼哈顿生成树 17
莫队算法 18
数据结构 20
树状数组 20
树状数组改段求段 20
RMQ 21
树链剖分 21
Treap 树 23
Splay 树 24
Link cut tree 27
树的点分治 28
字符串 30
Hash 30
KMP 30
Manacher 31
字典树 31
AC 自动机 32
后缀数组 33
数论 34
自适应辛普森公式 35
高斯消元 浮点数解 35
高斯消元 求整数解 36
欧拉函数 36
Exgcd 求逆元 37
扩展 gcd+中国剩余定理 37
高精度模版 37
素数 38
随机测大素数 40

矩阵快速幂 40
计算几何 40
不共线凸包 40
共线凸包 42
其他 46
三维凸包 46
输入输出挂 47
Java 读写挂 47
优先队列小的数先出来 48
Java 大数用法示例 48

图论

Lca 倍增法

```
#include"cstdio"
#include"iostream"
#include"queue"
#include"algorithm"
#include"set"
#include"queue"
#include"cmath"
#include"string.h"
#include"vector"
using namespace std;
#define N 10050
struct Edge{
    int from, to, dis, nex;
}edge[5*N];
int head[N],edgenum,dis[N],fa[N][20],dep[N];  //fa[i][x] 是 i 的第 2^x 个父亲（如果超过范围就是根）
void add(int u,int v,int w){
    Edge E={u,v,w,head[u]};
    edge[edgenum] = E;
    head[u]=edgenum++;
}
void bfs(int root){
    queue<int> q;
    fa[root][0]=root;dep[root]=0;dis[root]=0;
    q.push(root);
    while(!q.empty()){
        int u=q.front();q.pop();
        for(int i=1;i<20;i++)fa[u][i]=fa[fa[u][i-1]][i-1];
        for(int i=head[u]; ~i;i=edge[i].nex){
            int v=edge[i].to;if(v==fa[u][0])continue;
            dep[v]=dep[u]+1;dis[v]=dis[u]+edge[i].dis;fa[v][0]=u;
            q.push(v);
        }
    }
}
int Lca(int x,int y){
    if(dep[x]<dep[y])swap(x,y);
    for(int i=0;i<20;i++)if((dep[x]-dep[y])&(1<<i))x=fa[x][i];
    if(x==y)return x;
    for(int i=19;i>=0;i--)if(fa[x][i]!=fa[y][i])x=fa[x][i],y=fa[y][i];
    return fa[x][0];
}
void init(){memset(head, -1, sizeof head); edgenum = 0;}
```

强连通

```
#define N 30100
//N 为最大点数
#define M 150100
//M 为最大边数
int n, m;//n m 为点数和边数

struct Edge{
    int from, to, nex;
    bool sign;//是否为桥
}edge[M<<1];
int head[N], edgenum;
void add(int u, int v){//边的起点和终点
    Edge E={u, v, head[u], false};
    edge[edgenum] = E;
    head[u] = edgenum++;
}

int DFN[N], Low[N], Stack[N], top, Time; //Low[u]是点集{u 点及以 u 点为根的子树} 中(所有反向弧)能指向的(离根最近的祖先 v) 的 DFN[v]值(即 v 点时间戳)
int taj;//连通分支标号，从 1 开始
int Belong[N];//Belong[i] 表示 i 点属于的连通分支
bool Instack[N];
vector<int> bcc[N]; //标号从 1 开始
```

```
void tarjan(int u ,int fa){
    DFN[u] = Low[u] = ++ Time ;
    Stack[top ++ ] = u ;
    Instack[u] = 1 ;

    for (int i = head[u] ; ~i ; i = edge[i].nex ){
        int v = edge[i].to ;
        if(DFN[v] == -1)
        {
            tarjan(v , u) ;
            Low[u] = min(Low[u] ,Low[v]) ;
            if(DFN[u] < Low[v])
            {
                edge[i].sign = 1;//为割桥
            }
        }
        else if(Instack[v]) Low[u] =
min(Low[u] ,DFN[v]) ;
    }
    if(Low[u] == DFN[u]){
        int now;
        taj ++ ; bcc[taj].clear();
        do{
            now = Stack[-- top] ;
            Instack[now] = 0 ;
            Belong [now] = taj ;
            bcc[taj].push_back(now);
        }while(now != u) ;
    }
}

void tarjan_init(int all){
    memset(DFN, -1, sizeof(DFN));
    memset(Instack, 0, sizeof(Instack));
    top = Time = taj = 0;
    for(int i=1;i<=all;i++)if(DFN[i]==-1 )tarjan(i,
i); //注意开始点标！！！
}
vector<int>G[N];
int du[N];
void suodian(){
    memset(du, 0, sizeof(du));
    for(int i = 1; i <= taj; i++)G[i].clear();
    for(int i = 0; i < edgenum; i++){
        int u = Belong[edge[i].from], v =
Belong[edge[i].to];
        if(u!=v)G[u].push_back(v), du[v]++;
    }
}
void init(){memset(head, -1, sizeof(head));
edgenum=0;}
```

边双连通

缩点 求桥模版：

调用方法：

```
init();
solve(int l, int r){}; [l, r] 是点标
suodian();
#include <stdio.h>
#include <iostream>
#include <algorithm>
#include <string.h>
#include <queue>
#include <vector>
using namespace std;
#define N 100005
#define M 200300
#define inf 10000000
struct Edge{
    int from,to,next;
    bool cut;
}edge[2*M];
int head[N],edgenum;
int Low[N],DFN[N],Stack[N];//Belong 数组的值是 1~block
int Index,top;
int Belong[N],block;//新图的连通块标号（1~block）
bool Instack[N];
int bridge; //割桥数量

void addedge(int u,int v){
    Edge E={u,v,head[u],0}; edge[edgenum]=E; head[u]
= edgenum++;
    Edge E2={v,u,head[v],0};edge[edgenum]=E2;head[v]
= edgenum++;
}
void Tarjan(int u,int pre){
    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;
        // 如果重边有效的话下面这句改成: if(v == pre && pre_num == 0){pre_num++;continue;} pre_num 在 for 上面定义 int pre_num=0;
        if( v == pre )continue;
        if( !DFN[v] ){
            Tarjan(v,u);
            Low[u]=min(Low[u],Low[v]);
            if(Low[v] > DFN[u]){
                bridge++;
                edge[i].cut = true;
                edge[i^1].cut = true;
            }
        }
        else if(Instack[v])Low[u] = min(Low[u],DFN[v]);
    }
    if(Low[u] == DFN[u]){
        block++;
        do{
            v = Stack[--top];
            Instack[v] = false;
            Belong[v] = block;
        }while( v != u );
    }
}
void work(int l, int r){
    memset(DFN,0,sizeof(DFN));
    memset(Instack,false,sizeof(Instack));
    Index = top = block = bridge = 0;
    for(int i = l; i <= r; i++)if(!DFN[i])Tarjan(i,i);
}
vector<int>G[N];//点标从 1-block
void suodian(){
    for(int i = 1; i <= block; i++)G[i].clear();
    for(int i = 0; i < edgenum; i+=2){
        int u = Belong[edge[i].from], v = Belong[edge[i].to];
        if(u==v)continue;
        G[u].push_back(v), G[v].push_back(u);
    }
}
void init(){edgenum = 0;
memset(head,-1,sizeof(head));}
```

点双连通

一个割点属于他相邻的所有双连通分量

```
#include<iostream>
#include<string>
#include<algorithm>
#include<cstdlib>
#include<cstdio>
#include<set>
#include<map>
#include<vector>
#include<cstring>
#include<stack>
#include<cmath>
#include<queue>
using namespace std;
#define M 200004
#define N 10004
//1 init()
//2 加边
//3 find_bcc(1,n);若点标从 1 开始
//4 得到每个点所属的连通分支 Belong[] 是否为割点 iscut[], 新图 G[N]; 新图点标从 1-block
struct Edge{
    int from,to,nex;
}edge[M*2];
int head[N],edgenum;
void add(int u, int v){
    Edge E ={u,v,head[u]}; edge[edgenum] = E ;
head[u]=edgenum++;
    Edge E2={v,u,head[v]}; edge[edgenum] = E2;
head[v]=edgenum++;
}
int DFN[N],dfs_clock,block;
int Belong[N];
bool iscut[N];
// 割顶的 bccno 无意义
stack<Edge> S;
vector<int>G[N];
int Tarjan(int u,int fa){
    int lowu, child=0;
    lowu = DFN[u] = ++dfs_clock;
    for(int i=head[u]; ~i; i = edge[i].nex)
    {
        int v = edge[i].to;
        Edge e;
        e.from=u,e.to=v;
```

```
        if(!DFN[v])
        {
            S.push(e);
            child++;
            int lowv=Tarjan(v,u);
            lowu=min(lowu,lowv);
            if(lowv>=DFN[u])
            {
                iscut[u]=1;
                block++;
                G[block].clear();
                while(1)
                {
                    Edge x=S.top();S.pop();
                    if(Belong[x.from]!=block)
                    {
                        G[block].push_back(x.from);

                        Belong[x.from]=block;
                    }
                    if(Belong[x.to]!=block)
                    {
                        G[block].push_back(x.to);
                        Belong[x.to]=block;
                    }
                    if(x.from==u&&x.to==v)break;
                }
            }
        }
        else if(DFN[v]<DFN[u]&&v!=fa)
        {
            S.push(e);
            lowu=min(lowu,DFN[v]);
        }
    }
    if(fa<0&&child==1)iscut[u]=0;
    return lowu;
}
void find_bcc(int l, int r){
    memset(DFN,0,sizeof(DFN));
    memset(iscut,0,sizeof(iscut));
    memset(Belong,0,sizeof(Belong));
    dfs_clock=block=0;
    for(int i=l;i<=r;i++) if(!DFN[i])Tarjan(i,-1);
}
void init(){ memset(head, -1, sizeof head); edgenum =
0;}
```

2-sat

求最小字典序解

对于一个变量 xi 的真假，我们可以认为是存在 2 个物体，一个物体为 xi=0 ，一个物体为 xi=1，（显然 若我们两个物体只能取其中一个，若都不取表示 xi 还是个未知变量，若都取是不科学的=>无解）

我们用 mark[i *2] = true or false 表示 xi = 0 这个物品取或不取 如我们取了 xi=0 这个物品则 mark[i*2] = 1
用 mark[i*2+1] = true or false 表示 xi = 1 这个物品取或不取，如我们取了 xi=1 这个物品则 mark[i*2+1] = 1
if(mark[i*2] == 0 && mark[i*2+1] == 0){ （说明没有设置 xi 这个变量，则我们任意假设 xi 的值）
if(mark[i*2] && mark[i*2+1]) {说明无解}

```
#define N 1005*2
#define M 40000+5
//注意 n 是拆点后的大小 即 n <<= 1 N 为点数(注意要翻倍) M 为边数 i&1=0 为 i 真 i&1=1 为 i 假
struct Edge{
    int to, nex;
}edge[M];
//注意 N M 要修改
int head[N], edgenum;
void addedge(int u, int v){
    Edge E = {v, head[u]};
    edge[edgenum] = E;
    head[u] = edgenum ++;
}

bool mark[N];
int Stack[N], top;
void init(){
    memset(head, -1, sizeof(head)); edgenum = 0;
    memset(mark, 0, sizeof(mark));
}

bool dfs(int x){
    if(mark[x^1])return false;//一定是拆点的点先判断
    if(mark[x])return true;

    mark[x] = true;
    Stack[top++] = x;
```

```
    for(int i = head[x]; i != -1; i = edge[i].nex)
        if(!dfs(edge[i].to)) return false;

    return true;
}

bool solve(int n){
    for(int i = 0; i < n; i+=2)
        if(!mark[i] && !mark[i^1])
        {
            top = 0;
            if(!dfs(i))
            {
                while( top ) mark[ Stack[--top] ] = false;
                if(!dfs(i^1)) return false;
            }
        }
    return true;}
```

二分匹配

最小边覆盖 = N - 二分图最大匹配
最小顶点覆盖数 = 最大匹配数
最大独立集=2 个点集点数-最大匹配数
最小路径覆盖 = 顶点数 - 最大匹配数
最大团个数=x+y 顶点数 - 补图最大匹配数 二分匹配

```
int lef[N], pn;//lef[v]表示 Y 集的点 v 当前连接的点 , pn 为 x 点集的点数
bool T[N];     //T[u] 表示 Y 集 u 是否已连接 X 集
vector<int>G[N]; //匹配边  G[X 集].push_back(Y 集)  注意 G 初始化

bool match(int x){ // x 和 Y 集 匹配 返回 x 点是否匹配成功
    for(int i=0; i<G[x].size(); i++)
    {
        int v = G[x][i];
        if(!T[v])
        {
            T[v] = true;
            if(lef[v] == -1 || match( lef[v] ))
//match(lef[v]) : 原本连接 v 的 X 集点 lef[v] 能不能和别人连，如果能 则 v 这个点就空出来和 x 连
            {
                lef[v] = x;
                return true;
            }
        }
    }
    return false;
}

int solve(){
    int ans = 0;
    memset(lef, -1, sizeof(lef));
    for(int i = 1; i<= pn; i++)//X 集匹配，X 集点标号从 1-pn 匹配边是 G[左点].size()
    {
        memset(T, 0, sizeof(T));
        if( match( i ) ) ans++;
    }
    return ans;
}
```

全局最小割

```
//点标[0,n-1] 开始时先 init 复杂度 n^3
//对于边(u,v,flow): g[u][v]+=flow, g[v][u]+=flow;
typedef long long ll;
const int N = 305;
const ll inf = 1e18;
ll g[N][N], w[N];
int a[N], v[N], na[N];
ll mincut(int n) {
    int i, j, pv, zj;
    ll best = inf;
    for(i = 0; i < n; i ++) v[i] = i;

    while(n > 1) {
        for(a[v[0]] = 1, i = 1; i < n; i ++) {
            a[v[i]] = 0;
            na[i-1] = i;
            w[i] = g[v[0]][v[i]];
        }
        for(pv = v[0], i = 1; i < n; i ++) {
            for(zj = -1, j = 1; j < n; j ++)
                if(!a[v[j]] && (zj < 0 || w[j] > w[zj])) zj = j;

            a[v[zj]] = 1;
            if(i == n-1) {
```

```
                if(best > w[zj]) best = w[zj];
                for(i = 0; i < n; i ++) {
                    g[v[i]][pv] = g[pv][v[i]] += g[
v[zj]][v[i]];
                }
                v[zj] = v[--n];
                break;
            }
            pv = v[zj];
            for(j = 1; j < n; j ++) if(!a[v[j]])
                w[j] += g[v[zj]][v[j]];
        }
    }
    return best;
}
void init(int n){
    for(int i = 0; i < n; i ++)
        for(int j = 0; j < n; j ++)
            g[i][j] = g[j][i] = 0;
}
```

网络流

无源汇上下界网络流：构图如下：

1、首先对于与每条边(u,v,L,H)，u->v 连一条容量为 H-L 的边

2、创建一个源 S，和汇 T

3、对于任意一个结点，如果 u 出边下界和 OutL > 入边下界和 InL，则 u->T 一条 OutL - InL 的边。

否则连 S->u 一条 InL-OutL 的边。

4、求 s-t 最大流，若与 S 关联的边满容量，则有解。则残余网络中 u->v 的流量+其原来图的下界构成一个可行流

有源汇（S T）上下界最小流

像无源无汇上下界可行流那样先建好图，记图的超级源点为 SS，超级汇点为 TT。

先从 SS 到 TT 跑一遍最大流，然后加边 T->S 容量为无穷大，然后再从 SS 到 TT 跑一遍最大流，若与 SS 关联的边满容量，则有解。其中最小流为最后加进去的 n→1 的边的流量，找边的流量跟无源无汇上下界可行流一样，否则无解。

有源汇（S T）上下界最大流

像无源无汇上下界可行流那样先建好图，记图的超级源点为 SS，超级汇点为 TT。

然后 T 到 S 连一条边，容量为无穷大。

从 SS->TT 跑一遍最大流 判可行性

最后从源点 S 到汇点 T 跑一遍最大流就是答案，

每条边容量的取法和无源无汇上下界可行流一样。

最大权闭合图建图：

建图：把点权为 b(b>0)的连到源点，边权为 b

点权为 a(a<0)的连到汇点，边权为 a

然后建出原图，原图中所有边权为 inf

最后答案为 ： 正点权和 - 最大流

正点权->负点权的边是约束条件(取了这个 正点权必须取哪些点)

正点权割边(源点->正点权)表示该点不选

负点权割边(负点权->汇点)表示该点选

网络流 Dinic

特别注意：无向图并不是加 2 条边，而是 add(u,v,cap,cap);

有向边则是 add(u,v,cap,0);

Dinic:

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <iostream>
using namespace std;
//点标 [0,n]
const int N  = 200010;
const int M  = 500010;
const int INF  = ~0u >> 2;
template<class T>
struct Max_Flow {
    int n;
    int Q[N], sign;
    int head[N], level[N], cur[N], pre[N];
    int nxt[M], pnt[M], E;
    T cap[M];
    void Init(int n) {
        this->n  = n + 1;
        E  = 0;
        std::fill(head, head  + this->n, -1);
    }
    //有向 rw 就= 0
    void add(int from, int to, T c, T rw) {
        pnt[E] = to;
        cap[E] = c;
        nxt[E] = head[from];
        head[from] = E++;

        pnt[E] = from;
        cap[E] = rw;
```

```
        nxt[E] = head[to];
        head[to] = E++;
    }
    bool Bfs(int s, int t) {
        sign  = t;
        std::fill(level, level  + n, -1);
        int *front  = Q, *tail  = Q;
        *tail++ = t; level[t] = 0;
        while (front < tail && level[s] == -1) {
            int u  = *front++;
            for (int e  = head[u]; e  != -1; e  =
nxt[e]) {
                if (cap[e  ^ 1] > 0 && level[pnt[e]]
< 0) {
                    level[pnt[e]] = level[u] + 1;
                    *tail ++ = pnt[e];
                }
            }
        }
        return level[s] != -1;
    }
    void Push(int t, T &flow) {
        T mi  = INF;
        int p  = pre[t];
        for (int p  = pre[t]; p  != -1; p  = pre[pnt[p
^ 1]]) {
            mi  = std::min(mi, cap[p]);
        }
        for (int p  = pre[t]; p  != -1; p  = pre[pnt[p
^ 1]]) {
            cap[p] -= mi;
            if (!cap[p]) {
                sign  = pnt[p  ^ 1];
            }
            cap[p  ^ 1] += mi;
        }
        flow  += mi;
    }
    void Dfs(int u, int t, T &flow) {
        if (u  == t) {
            Push(t, flow);
            return ;
        }
        for (int &e  = cur[u]; e  != -1; e  = nxt[e])
{
            if (cap[e] > 0 && level[u] - 1 ==
level[pnt[e]]) {
                pre[pnt[e]] = e;
                Dfs(pnt[e], t, flow);
                if (level[sign] > level[u]) {
                    return ;
                }
                sign  = t;
            }
        }
    }
    T Dinic(int s, int t) {
        pre[s] = -1;
        T flow  = 0;
        while (Bfs(s, t)) {
            std::copy(head, head  + n, cur);
            Dfs(s, t, flow);
        }
        return flow;
    }
};
Max_Flow <int>F;
```

网络流 Isap

```
#include<stdio.h>
#include<string.h>
#include<iostream>
#include<algorithm>
#include<vector>
using namespace std;
#define ll int
const int MAXN = 100010;//点数的最大值
const int MAXM = 400010;//边数的最大值
const int INF = 0x3f3f3f3f;
struct Edge
{
    int to,next,cap,flow;
}edge[MAXM];//注意是 MAXM
int tol;
int head[MAXN];
int gap[MAXN],dep[MAXN],cur[MAXN];
void add(int u,int v,int w,int rw = 0)
{
    edge[tol].to = v; edge[tol].cap = w; edge[tol].flow
= 0;
    edge[tol].next = head[u]; head[u] = tol++;
    edge[tol].to = u; edge[tol].cap = rw;
```

```
edge[tol].flow = 0;
    edge[tol].next = head[v]; head[v] = tol++;
}
int Q[MAXN];
void BFS(int start,int end)
{
    memset(dep,-1,sizeof(dep));
    memset(gap,0,sizeof(gap));
    gap[0] = 1;
    int front = 0, rear = 0;
    dep[end] = 0;
    Q[rear++] = end;
    while(front != rear)
    {
        int u = Q[front++];
        for(int i = head[u]; i != -1; i = edge[i].next)
        {
            int v = edge[i].to;
            if(dep[v] != -1)continue;
            Q[rear++] = v;
            dep[v] = dep[u] + 1;
            gap[dep[v]]++;
        }
    }
}
int S[MAXN];
int sap(int start,int end,int N)
{
    BFS(start,end);
    memcpy(cur,head,sizeof(head));
    int top = 0;
    int u = start;
    int ans = 0;
    while(dep[start] < N)
    {
        if(u == end)
        {
            int Min = INF;
            int inser;
            for(int i = 0;i < top;i++)
                if(Min > edge[S[i]].cap -
edge[S[i]].flow)
                {
                    Min = edge[S[i]].cap -
edge[S[i]].flow;
                    inser = i;
                }
            for(int i = 0;i < top;i++)
            {
                edge[S[i]].flow += Min;
                edge[S[i]^1].flow -= Min;
            }
            ans += Min;
            top = inser;
            u = edge[S[top]^1].to;
            continue;
        }
        bool flag = false;
        int v;
        for(int i = cur[u]; i != -1; i = edge[i].next)
        {
            v = edge[i].to;
            if(edge[i].cap - edge[i].flow && dep[v]+1
== dep[u])
            {
                flag = true;
                cur[u] = i;
                break;
            }
        }
        if(flag)
        {
            S[top++] = cur[u];
            u = v;
            continue;
        }
        int Min = N;
        for(int i = head[u]; i != -1; i = edge[i].next)
            if(edge[i].cap - edge[i].flow &&
dep[edge[i].to] < Min)
            {
                Min = dep[edge[i].to];
                cur[u] = i;
            }
        gap[dep[u]]--;
        if(!gap[dep[u]])return ans;
        dep[u] = Min + 1;
        gap[dep[u]]++;
        if(u != start)u = edge[S[--top]^1].to;
    }
    return ans;
}
```

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

费用流 Spfa

浪的上下界费用流做法
把有下界的边(u,v,lowBound,upBound)
拆成两条
一条是(u,v)容量为 lowBound,费用为-inf
然后另一条容量为 upBound-lowBound ,费用正常
费用为原费用。。答案预先加上 lowBound*费用
然后这个图跑一遍
答案对 inf 取个模

```
#include<iostream>
#include<stdio.h>
#include<string.h>
#include<queue>
#include<math.h>
using namespace std;
#define ll int
#define inf 0x3f3f3f3f
#define Inf 0x3FFFFFFFFFFFFFFFLL
#define N 605*605*2
#define M 605*605*4
struct Edge {
    ll to, cap, cost, nex;
    Edge(){}
    Edge(ll to,ll cap,ll cost,ll
next):to(to),cap(cap),cost(cost),nex(next){}
} edge[M<<1];
ll head[N], edgenum;
ll D[N], A[N], P[N];
bool inq[N];
void add(ll from,ll to,ll cap,ll cost) {
    edge[edgenum] = Edge(to,cap,cost,head[from]);
    head[from] = edgenum++;
    edge[edgenum] = Edge(from,0,-cost,head[to]);
    head[to] = edgenum++;
}
bool spfa(ll s, ll t, ll &flow, ll &cost) {
    for(ll i = 0; i <= t; i++) D[i] = inf;
    memset(inq, 0, sizeof inq);
    queue<ll>q;
    q.push(s);
    D[s] = 0; A[s] = inf;
    while(!q.empty()) {
        ll u = q.front(); q.pop();
        inq[u] = 0;
        for(ll i = head[u]; ~i; i = edge[i].nex)
        {
            Edge &e = edge[i];
            if(e.cap && D[e.to] > D[u] + e.cost)
            {
                D[e.to] = D[u] + e.cost;
                P[e.to] = i;
                A[e.to] = min(A[u], e.cap);
                if(!inq[e.to])
                {inq[e.to]=1; q.push(e.to);}
            }
        }
    }
     //若费用为 inf 则中止费用流
    if(D[t] == inf) return false;
    cost += D[t] * A[t];
    flow += A[t];
    ll u = t;
    while(u != s) {
        edge[ P[u] ].cap -= A[t];
        edge[P[u]^1].cap += A[t];
        u = edge[P[u]^1].to;
    }
    return true;
}
ll Mincost(ll s,ll t){
    ll flow = 0, cost = 0;
    while(spfa(s, t, flow, cost));
    return cost;
}
void init(){memset(head,-1,sizeof head); edgenum = 0;}
```

费用流 Zkw

```
#include <stdio.h>
#include <string.h>
#include <iostream>
#include <math.h>
#include <queue>
#include <set>
#include <algorithm>
using namespace std;
#define N 300005
#define M 3000055
#define inf 0x3f3f3f3f
//点标[0,n] 图中不能有负环
struct MaxFlow
```

```
{
    int size, n;
    int st, en, maxflow, mincost;
    bool vis[N];
    int net[N], pre[N], cur[N], dis[N];
    queue <int> Q;
    struct EDGE  {
        int v, cap, cost, next;
        EDGE(){}
        EDGE(int a, int b, int c, int d)
        {
            v = a, cap = b, cost = c, next = d;
        }
    }E[M<<1];
    void init(int _n)//传入点的个数[1-_n]
    {
        n = _n, size = 0;
        memset(net, -1, sizeof(net));
    }
    void add(int u, int v, int cap, int cost)
    {
        E[size] = EDGE(v, cap, cost, net[u]);
        net[u] = size++;
        E[size] = EDGE(u, 0, -cost, net[v]);
        net[v] = size++;
    }
    bool adjust()
    {
        int v, min = inf;
        for(int i = 0; i <= n; i++)
        {
            if(!vis[i])
                continue;
            for(int j = net[i]; v = E[j].v, j != -1;
j = E[j].next)
                if(E[j].cap)
                    if(!vis[v] &&
dis[v]-dis[i]+E[j].cost < min)
                        min = dis[v] - dis[i] +
E[j].cost;
        }
        if(min == inf)
            return false;
        for(int i = 0; i <= n; i++)
            if(vis[i])
                cur[i] = net[i], vis[i] = false, dis[i]
+= min;
        return true;
    }
    int augment(int i, int flow)
    {
        if(i == en)
        {
            mincost += dis[st] * flow;
            maxflow += flow;
            return flow;
        }
        vis[i] = true;
        for(int j = cur[i], v; v = E[j].v, j != -1; j
= E[j].next)
        {
            if(!E[j].cap)
                continue;
            if(vis[v] || dis[v]+E[j].cost != dis[i])
                continue;
            int delta = augment(v, std::min(flow,
E[j].cap));
            if(delta)
            {
                E[j].cap -= delta;
                E[j^1].cap += delta;
                cur[i] = j;
                return delta;
            }
        }
        return 0;
    }
    void spfa()
    {
        int u, v;
        for(int i = 0; i <= n; i++)
            vis[i] = false, dis[i] = inf;
        dis[st] = 0;
        Q.push(st);
        vis[st] = true;
        while(!Q.empty())
        {
            u = Q.front(), Q.pop();
            vis[u] = false;
            for(int i = net[u]; v = E[i].v, i != -1;
i = E[i].next)
            {
```

```
                if(!E[i].cap || dis[v] <= dis[u] +
E[i].cost)
                    continue;
                dis[v] = dis[u] + E[i].cost;
                if(!vis[v])
                {
                    vis[v] = true;
                    Q.push(v);
                }
            }
        }
        for(int i = 0; i <= n; i++)
            dis[i] = dis[en] - dis[i];
    }
    int zkw(int s, int t)
    {
        st = s, en = t;
        spfa();
        mincost = maxflow = 0;
        for(int i = 0; i <= n; i++)
            vis[i] = false, cur[i] = net[i];
        do
        {
            while(augment(st, inf))
                memset(vis, false, sizeof(vis));
        }while(adjust());
        return mincost;
    }
}zkw;
```

KM 算法

图必须是完美匹配

```
#include <stdio.h>
#include <string.h>
#define M 310
#define inf 0x3f3f3f3f

int n,nx,ny;
int link[M],lx[M],ly[M],slack[M];    //lx,ly 为顶标，
nx,ny 分别为 x 点集 y 点集的个数
int visx[M],visy[M],w[M][M];

int DFS(int x)
{
    visx[x] = 1;
    for (int y = 1;y <= ny;y ++)
    {
        if (visy[y])
            continue;
        int t = lx[x] + ly[y] - w[x][y];
        if (t == 0)      //
        {
            visy[y] = 1;
            if (link[y] == -1||DFS(link[y]))
            {
                link[y] = x;
                return 1;
            }
        }
        else if (slack[y] > t)  //不在相等子图中 slack 取
最小的
            slack[y] = t;
    }
    return 0;
}
int KM()
{
    int i,j;
    memset (link,-1,sizeof(link));
    memset (ly,0,sizeof(ly));
    for (i = 1;i <= nx;i ++)           //lx 初始化为与
它关联边中最大的
        for (j = 1,lx[i] = -inf;j <= ny;j ++)
            if (w[i][j] > lx[i])
                lx[i] = w[i][j];

    for (int x = 1;x <= nx;x ++)
    {
        for (i = 1;i <= ny;i ++)
            slack[i] = inf;
        while (1)
        {
            memset (visx,0,sizeof(visx));
            memset (visy,0,sizeof(visy));
            if (DFS(x))     //若成功（找到了增广轨），则
该点增广完成，进入下一个点的增广
                break;  //若失败（没有找到增广轨），则需
要改变一些点的标号，使得图中可行边的数量增加。
                        //方法为：将所有在增广轨中（就是
在增广过程中遍历到）的 X 方点的标号全部减去一个常数 d，
                        //所有在增广轨中的 Y 方点的标号全
部加上一个常数 d
            int d = inf;
```

```
            for (i = 1;i <= ny;i ++)
                if (!visy[i]&&d > slack[i])
                    d = slack[i];
            for (i = 1;i <= nx;i ++)
                if (visx[i])
                    lx[i] -= d;
            for (i = 1;i <= ny;i ++)  //修改顶标后，要把所有不在交错树中的 Y 顶点的 slack 值都减去 d
                if (visy[i])
                    ly[i] += d;
                else
                    slack[i] -= d;
        }
    }
    int res = 0;
    for (i = 1;i <= ny;i ++)
        if (link[i] > -1)
            res += w[link[i]][i];
    return res;
}
int main ()
{
    int i,j;
    while (scanf ("%d",&n)!=EOF)
    {
        nx = ny = n;
      //  memset (w,0,sizeof(w));
        for (i = 1;i <= n;i ++)
            for (j = 1;j <= n;j ++)
                scanf ("%d",&w[i][j]);
        int ans = KM();
        printf ("%d\n",ans);
    }
    return 0;
}
```

最小生成树计数

Init->add(u,v)->cal_MST(点数,mod) 点标[1,n]

```
typedef long long ll;
const int N = 105;    //点的个数
const int M = 1005;  //边的个数
//点标从 1-n MOD 是 long long
struct node {
    int set[N];
    void init(int n) {
        for (int i = 0; i <= n; i++) set[i] = i;
    }
    int find(int x) {
        return x == set[x] ? x : set[x] = find(set[x]);
    }
    int Union(int x, int y) {
        int xx = find(x);
        int yy = find(y);
        if (xx == yy) return -1;
        set[xx] = yy;
        return 1;
    }
}a, b, c;

struct Node {
    int u, v, dis;
}edge[M];
int edgenum;
void add(int u, int v, int d){
    Node E = { u, v, d };
    edge[++edgenum] = E;
}

bool visit[N];
vector<int> g[N];
ll p[N][N], deg[N][N];
int cmp(Node a, Node b) {
    return a.dis < b.dis;
}
ll DET(ll a[][N], int n, ll MOD)
{
    int i, j, k;
    ll temp = 1, t;
    for (i = 0; i < n; i++) for (j = 0; j < n; j++) a[i][j] %= MOD;
    for (i = 1; i < n; i++)
    {
        for (j = i + 1; j < n; j++) while (a[j][i])

        {
            t = a[i][i] / a[j][i];
            for (k = i; k < n; k++)
            {
                a[i][k] -= a[j][k] * t;
                a[i][k] %= MOD;
            }
```

```
            for (k = i; k < n; k++)
                swap(a[i][k], a[j][k]);

            temp = -temp;
        }
        temp = temp*a[i][i] % MOD;
    }
    return (temp + MOD) % MOD;
}

ll cal_MST_count(int n, ll MOD) {
    sort(edge + 1, edge + edgenum + 1, cmp);
    int pre = edge[1].dis;
    ll ans = 1;
    a.init(n);
    b.init(n);
    memset(visit, 0, sizeof(visit));
    memset(deg, 0, sizeof(deg));
    for (int i = 0; i <= n; i++) g[i].clear();
    for (int t = 1; t <= edgenum + 1; t++)
    {
        if (edge[t].dis != pre || t == edgenum + 1)

        {
            for (int i = 1, k; i <= n; i++) if (vis
it[i])
            {
                k = b.find(i);
                g[k].push_back(i);
                visit[i] = 0;
            }
            for (int i = 1; i <= n; i++)
            if (g[i].size())
            {
                memset(p, 0, sizeof(p));
                for (int j = 0; j < g[i].size(); j+
+)
                for (int k = j + 1, x, y; k < g[i].
size(); k++)
                {
                    x = g[i][j];
                    y = g[i][k];
                    p[j][k] = p[k][j] = -deg[x][y];

                    p[j][j] += deg[x][y];
                    p[k][k] += deg[x][y];
                }
                ans = ans*DET(p, g[i].size(), MOD)
% MOD;
                for (int j = 0; j < g[i].size(); j+
+) a.set[g[i][j]] = i;
            }
            memset(deg, 0, sizeof(deg));
            for (int i = 1; i <= n; i++)
            {
                b.set[i] = a.find(i);
                g[i].clear();
            }
            if (t == edgenum + 1) break;
            pre = edge[t].dis;
        }
        int x = a.find(edge[t].u);
        int y = a.find(edge[t].v);
        if (x == y) continue;
        visit[x] = visit[y] = 1;
        b.Union(x, y);
        deg[x][y]++;
        deg[y][x]++;
    }
    if (!edgenum) return 0;
    for (int i = 2; i <= n; i++)
    if (b.find(i) != b.find(1))
        return 0;
    return ans;
}
void init(){ edgenum = 0; }
```

最小树形图

复杂度 O(NM)

点下标[0,n-1] 边下标[0,m-1]

有向边表示：u->v 花费为 cost

返回最小树形图的边权和，-1 表示不存在最小树形图

```
#include <stdio.h>
#include <string.h>
#include <iostream>
#include <algorithm>
#include <math.h>
using namespace std;
const int INF = 100000000;
const int MAXN = 1010; //点数
const int MAXM = 1010000;//边数
#define ll int
struct Edge{
```

```
    int u,v;
    ll cost;
}edge[MAXM];
int pre[MAXN],id[MAXN],visit[MAXN],edgenum;
void addedge(int u, int v, ll cost){
    Edge E = {u, v, cost}; edge[edgenum++] = E;
}
ll in[MAXN];
ll zhuliu(int root,int n,int m,Edge edge[])//树根(注意是有向树，树根不能任意) 点数 边数 edge

{
    int u,v;
    ll res=0;
    while(1)
    {
        for(int i = 0;i < n;i++)
            in[i] = INF;
        for(int i = 0;i < m;i++)
            if(edge[i].u != edge[i].v && edge[i].cost < in[edge[i].v])
            {
                pre[edge[i].v] = edge[i].u;
                in[edge[i].v] = edge[i].cost;
            }
            for(int i = 0;i < n;i++)
                if(i != root && in[i] == INF)
                    return -1;//不存在最小树形图
            int tn = 0;
            memset(id,-1,sizeof(id));
            memset(visit,-1,sizeof(visit));
            in[root] = 0;
            for(int i = 0;i < n;i++)
            {
                res += in[i];
                v = i;
                while( visit[v] != i && id[v] == -1 && v != root)
                {
                    visit[v] = i;
                    v = pre[v];
                }
                if( v != root && id[v] == -1 )
                {
                    for(int u = pre[v]; u != v ;u = pre[u])
                        id[u] = tn;
                    id[v] = tn++;
                }
            }
            if(tn == 0)break;//没有有向环
            for(int i = 0;i < n;i++)
                if(id[i] == -1)
                    id[i] = tn++;
            for(int i = 0;i < m;)
            {
                v = edge[i].v;
                edge[i].u = id[edge[i].u];
                edge[i].v = id[edge[i].v];
                if(edge[i].u != edge[i].v)
                    edge[i++].cost -= in[v];
                else
                    swap(edge[i],edge[--m]);
            }
            n = tn;
            root = id[root];
    }
    return res; //-1 为不存在最小树形图
}
void init(){
    edgenum = 0;
}
```

哈密顿回路

一个无向图，若每个点连到其他的一半或一半以上的点，则这个图一定存在哈密顿回路

（就是 5 个点，则每个点至少有 3 条边）

```
#include <cstdio>
#include <cstring>
#include <iostream>
using namespace std;

const int N = 155;

int n, m;
bool mp[N][N];

int S, T, top, Stack[N];
bool vis[N];
void _reverse(int l,int r) {
    while (l<r)
        swap(Stack[l++],Stack[r--]);
}
```

```
void expand() {
    while(1) {
        bool flag = 0;
        for (int i=1; i<=n && false == flag; i++)
            if (!vis[i] && mp[T][i])
            {
                Stack[top++]=i;
                T=i;
                flag = vis[i] = 1;
            }
        if (!flag) return;
    }
}
void hamiltun(int Start){
    memset(vis, 0, sizeof vis);

    S = Start;
    for(T=2; T<=n; T++) //任意找两个相邻的节点 S 和 T
        if (mp[S][T]) break;
    top = 0;
    Stack[top++]=S;
    Stack[top++]=T;
    vis[S] = vis[T] = true;
    while (1)
    {
        expand(); //在它们基础上扩展出一条尽量长的没有重复节点的路径:步骤
        _reverse(0,top-1);
        swap(S,T);
        expand(); //在它们基础上扩展出一条尽量长的没有重复节点的路
        int mid=0;
        if (!mp[S][T]) //若 S 与 T 不相邻,可以构造出一个回路使新的 S 和 T 相
        {
            //设路径 S→T 上有 k+2 个节点,依次为 S,v1,v2…… vk 和 T.
            //可以证明存在节点 vi,i∈[1,k),满足 vi 与 T 相邻,且 vi+1 与 S 相
            for (int i=1; i<top-2; i++)
                if (mp[Stack[i]][T] && mp[Stack[i+1]][S])
                {
                    mid=i+1; break;
                }
            //把原路径变成 S→vi→T→vi+1→S,即形成了一个回路
            _reverse(mid,top-1);
            T=Stack[top-1];
        }
        if (top==n) break;
        //现在我们有了一个没有重复节点的回路.如果它的长度为 N,则汉密尔顿回路就找到了
        //否则,由于整个图是连通的,所以在该回路上,一定存在一点与回路以外的点相邻
        //那么从该点处把回路断开,就变回了一条路径,再按照步骤 1 的方法尽量扩展路径
        for (int i = 1, j; i <= n; i++)
            if (!vis[i])
            {
                for (j=1; j<top-1; j++)
                    if (mp[Stack[j]][i]) break;
                if (mp[Stack[j]][i])
                {
                    T=i; mid=j;
                    break;
                }
            }
        S=Stack[mid-1];
        _reverse(0,mid-1);
        _reverse(mid,top-1);
        Stack[top++]=T;
        vis[T]=true;
    }
}
int main() {
    while (cin>>n>>m) {
        memset(mp, 0, sizeof mp);
        for (int i = 1, u, v; i <= m; i++) {
            scanf("%d %d",&u, &v);
            mp[u][v] = mp[v][u] = 1;
        }
        hamiltun(1);
        for (int i = 0; i < top; i++)
        printf("%d%c", Stack[i], i==top-1?'\n':' ');
    }
    return 0;
}
```

欧拉通路

```
public class Main {
    class Node implements Comparable<Node>{
```

```
        int v, vis;
        String str;
        Node(){}
        Node(int v, int vis, String str){
  this.v = v;
            this.vis = vis;
            this.str = str;
        }
        public int compareTo(Node o) {
            return this.str.compareTo(o.str);
        }
    }
    int n;
    ArrayList<Node>[] G = new ArrayList[Maxn];
    Stack<String> stack = new Stack();
    int[] vis = new int[Maxn], f = new int[Maxn], I
n = new int[Maxn], Out = new int[Maxn];
    int find(int x){return (x==f[x])?x:(f[x]=find(f
[x]));}
    void Union(int x, int y){
        int fx = find(x), fy = find(y);
        if(fx == fy)return ;
        f[fx] = fy;
    }
    void find_path(int u){
        for (int i = 0; i<G[u].size(); i++)
        {
            int v = G[u].get(i).v;
            if (0 == G[u].get(i).vis)
            {
                G[u].get(i).vis = 1;
                find_path(v);
                stack.push(G[u].get(i).str);
            }
        }
    }
    void print(){
        out.print(stack.pop());
        while(!stack.isEmpty())
            out.print("."+stack.pop());
        out.println();
    }
    int id;//id 为起点编号
    boolean euler_formula(){
        int cnt_big = 0, cnt_less = 0; id = -1;
        for (int i = 0; i<26; i++)
            if (vis[i] == 1)
            {
                if (In[i] == Out[i])continue;
                if (In[i] - Out[i] == 1)cnt_big++;

                else if (In[i] - Out[i] == -1){cnt_
less++; id = i;}
                else return false;
            }

        if (false == (cnt_big == 1 && cnt_less == 1)
 && false == (cnt_big == 0 && cnt_less == 0))return
 false;
        int cnt = 0; //图的联通块个数必须为 1
        for(int i = 0; i < Maxn; i++)
            if(vis[i] == 1 && f[i] == i)cnt++;
        return cnt == 1;
    }
    void add(int u, int v, String str){
        Out[u] ++; In[v] ++;
        vis[u] = vis[v] = 1;
        Node E = new Node(v, 0, str);
        G[u].add(E);
        Union(u, v);
    }
    void init(){
        for(int i = 0; i < Maxn; i++){
            f[i] = i;
            vis[i] = In[i] = Out[i] = 0;
            G[i].clear();
        }
        stack.clear();
    }
    void input(){
        init();
        n = cin.nextInt();
        for(int i = 1; i <= n; i++)
        {
            String s = "";
            while(s.length()==0 || s.charAt(0)<'a'|
|s.charAt(0)>'z')s = cin.next();
            int u = s.charAt(0)-'a', v = s.charAt(s.
length()-1)-'a';
            add(u, v, s);
        }
        for(int i = 0; i < Maxn; i++) Collections.s
```

```
ort(G[i]); //排序保证字典序最小
    }
    void work() {
        for(int i = 0; i < Maxn; i++) G[i] = new Ar
rayList();
        int T = cin.nextInt();
        while(T-->0){
            input();
            if(euler_formula()){
                if(id == -1)
                    for(int i = 0; i < 26 && id ==
-1; i++)
                        if(vis[i] == 1)
                            id = i;//若每个点出度=入
度则起点任取
                find_path(id);
                print();
            }
            else out.println("***");
        }
    }

    Main() {
        cin = new Scanner(System.in);
        out = new PrintWriter(System.out);
    }

    public static void main(String[] args) {
        Main e = new Main();
        e.work();
        out.close();
    }

    public Scanner cin;
    public static PrintWriter out;
    static int Maxn = 26;
}
```

二维平面最小曼哈顿生成树

```
const int N = 1e5 + 10;
typedef long long ll;
class MST{
    struct Edge{
        int from, to, dis;
        Edge(int _from = 0, int _to = 0, int _dis =
0) :from(_from), to(_to), dis(_dis){}
        bool operator < (const Edge &x) const{return
dis < x.dis;}
    }edge[N << 3];
    int f[N], tot;
    int find(int x){ return x == f[x] ? x : f[x] =
find(f[x]); }
    bool Union(int x, int y){
        x = find(x); y = find(y);
        if (x == y)return false;
        if (x > y)swap(x, y);
        f[x] = y;
        return true;
    }
public:
    void init(int n){
        for (int i = 0; i <= n; i++)f[i] = i;
        tot = 0;
    }
    void add(int u, int v, int dis){
        edge[tot++] = Edge(u, v, dis);
    }
    ll work(){//计算最小生成树，返回花费
        sort(edge, edge + tot);
        ll cost = 0;
        for (int i = 0; i < tot; i++)
            if (Union(edge[i].from, edge[i].to))
                cost += edge[i].dis;
        return cost;
    }
}mst;
struct Point{//二维平面的点
    int x, y, id;
    bool operator < (const Point&a) const{
        return x == a.x ? y < a.y : x < a.x;
    }
}p[N];
class BIT{//树状数组
    int c[N], id[N], maxn;
    int lowbit(int x){ return x&-x; }
public:
    void init(int n){
        maxn = n + 10;
        fill(c, c + maxn + 1, inf);
        fill(id, id + maxn + 1, -1);
```

```
    }
    void updata(int x, int val, int _id){
        while (x){
            if (val < c[x]){ c[x] = val; id[x] = _id; }
            x -= lowbit(x);
        }
    }
    int query(int x){
        int val = inf, _id = -1;
        while (x <= maxn){
            if (val > c[x]){ val = c[x]; _id = id[x]; }
            x += lowbit(x);
        }
        return _id;
    }
}tree;
inline bool cmp(int *x, int *y){ return *x < *y; }
class Manhattan_MST{//复杂度 O(max(N*1.5,Nlog(N)))
    int A[N], B[N];
public:
    ll work(int l, int r){
        mst.init(r);
        for (int dir = 1; dir <= 4; dir++){
            if (dir%2==0)for (int i = l; i <= r; i++)swap(p[i].x, p[i].y);
            else if (dir == 3)for (int i = l; i <= r; i++)p[i].y = -p[i].y;
            sort(p + l, p + r + 1);
            for (int i = l; i <= r; i++) A[i] = B[i] = p[i].y - p[i].x; //离散化
            sort(B + 1, B + N + 1);
            int sz = unique(B + 1, B + N + 1) - B - 1;
            //初始化反树状数组
            tree.init(sz);
            for (int i = r; i >= l; i--)
            {
                int pos = lower_bound(B + 1, B + sz + 1, A[i]) - B;
                int id = tree.query(pos);
                if (id != -1)
                    mst.add(p[i].id,      p[id].id, abs(p[i].x - p[id].x) + abs(p[i].y - p[id].y));
                tree.updata(pos,  p[i].x  +  p[i].y, i);
            }
        }for (int i = l; i <= r; i++)p[i].y = -p[i].y;
        return mst.work();
    }
}m_mst;

int n;
int main(){
    int Cas = 1;
    while (cin >> n, n){
        for (int i = 1; i <= n; i++)rd(p[i].x), rd(p[i].y), p[i].id = i;
        printf("Case %d: Total Weight = ", Cas++);
        cout << m_mst.work(1, n) << endl;
    }
    return 0;
}
```

莫队算法

我需要做什么：

完成两个函数：增加一个区间: add(int l, int r){}，删除一个区间: del(int l, inr){}

注意：

1、如果当前区间是空的，即 l > r, 那么新增加的点需要特殊处理。

2、add(int l, int r), del 的函数复杂度必须为 O(r-l)

3、莫队模板的复杂度：O(query*log(query))

```
const int N = 1e5 + 10;
vector<int>G[N];
int l[N], r[N];
class MST {
    struct Edge {
        int from, to, dis;
        Edge(int _from = 0, int _to = 0, int _dis = 0) :from(_from), to(_to), dis(_dis) {}
        bool operator < (const Edge &x) const { return dis < x.dis; }
    }edge[N << 3];
    int f[N], tot;
    int find(int x) { return x == f[x] ? x : f[x] = find(f[x]); }
    bool Union(int x, int y) {
        x = find(x); y = find(y);
        if (x == y)return false;
        if (x > y)swap(x, y);
        f[x] = y;
        return true;
    }
```

```
public:
    void init(int n) {
        for (int i = 0; i <= n; i++)f[i] = i;
        tot = 0;
    }
    void add(int u, int v, int dis) {
        edge[tot++] = Edge(u, v, dis);
    }
    ll work() {//计算最小生成树，返回花费
        sort(edge, edge + tot);
        ll cost = 0;
        for (int i = 0; i < tot; i++)
            if (Union(edge[i].from, edge[i].to)) {
                cost += edge[i].dis;
                G[edge[i].from].push_back(edge[i].to);
                G[edge[i].to].push_back(edge[i].from);
            }
        return cost;
    }
}mst;
struct Point {//二维平面的点
    int x, y, id;
    bool operator < (const Point&a) const {
        return x == a.x ? y < a.y : x < a.x;
    }
}p[N];
bool cmp_id(const Point&a, const Point&b) {
    return a.id < b.id;
}
class BIT {//树状数组
    int c[N], id[N], maxn;
    int lowbit(int x) { return x&-x; }
public:
    void init(int n) {
        maxn = n + 10;
        fill(c, c + maxn + 1, inf);
        fill(id, id + maxn + 1, -1);
    }
    void updata(int x, int val, int _id) {
        while (x) {
            if (val < c[x]) { c[x] = val; id[x] = _id; }
            x -= lowbit(x);
        }
    }
    int query(int x) {
        int val = inf, _id = -1;
        while (x <= maxn) {
            if (val > c[x]) { val = c[x]; _id = id[x]; }
            x += lowbit(x);
        }
        return _id;
    }
}tree;
inline bool cmp(int *x, int *y) { return *x < *y; }
class Manhattan_MST {
    int A[N], B[N];
public:
    ll work(int l, int r) {
        for (int i = l; i <= r; i++)G[i].clear();
        mst.init(r);
        for (int dir = 1; dir <= 4; dir++) {
            if (dir % 2 == 0)for (int i = l; i <= r; i++)swap(p[i].x, p[i].y);
            else if (dir == 3)for (int i = l; i <= r; i++)p[i].y = -p[i].y;
            sort(p + l, p + r + 1);
            for (int i = l; i <= r; i++) A[i] = B[i] = p[i].y - p[i].x; //离散化
            sort(B + l, B + r + 1);
            int sz = unique(B + l, B + r + 1) - B;
            //初始化反树状数组
            tree.init(sz);
            for (int i = r; i >= l; i--)
            {
                int pos = lower_bound(B + l, B + sz, A[i]) - B;
                int id = tree.query(pos);
                if (id != -1)
                    mst.add(p[i].id, p[id].id, abs(p[i].x - p[id].x) + abs(p[i].y - p[id].y));
                tree.updata(pos, p[i].x + p[i].y, i);
            }
        }
        for (int i = l; i <= r; i++)p[i].y = -p[i].y;
        return mst.work();
    }
}m_mst;

int n, m, a[N];
ll ans[N], now;
void add(int x, int y) {
    for (int i = x; i <= y; i++)
```

```
    {

    }
}
void del(int x, int y) {
    for (int i = x; i <= y; i++)
    {

    }
}
void dfs(int u, int fa) {
    if (fa != u)
        add(l[u], r[u]);
    else
    {
        if (l[u] < l[fa]) add(l[u], l[fa] - 1);
        if (r[u] > r[fa]) add(r[fa] + 1, r[u]);
        if (l[u] > l[fa]) del(l[fa], l[u] - 1);
        if (r[u] < r[fa]) del(r[u] + 1, r[fa]);
    }
    ans[u] = now;
    for (int v : G[u]) if (v != fa)dfs(v, u);
    if (fa != u)
    {
        if (l[u] > l[fa]) add(l[fa], l[u] - 1);
        if (r[u] < r[fa]) add(r[u] + 1, r[fa]);
        if (l[u] < l[fa]) del(l[u], l[fa] - 1);
        if (r[u] > r[fa]) del(r[fa] + 1, r[u]);
    }
}
int main() {
    rd(n);
    for (int i = 1; i <= n; i++)rd(a[i]);
    rd(m);
    for (int i = 1; i <= m; i++) {
        rd(p[i].x); rd(p[i].y); p[i].id = i;
        l[i] = p[i].x; r[i] = p[i].y;
    }
    m_mst.work(1, m);
    dfs(1, 1);
    for (int i = 1; i <= m; i++)pt(ans[i]), puts("");
    return 0;
}
```

数据结构

树状数组

```
inline int Lowbit(int x){return x&(-x);}
void change(int i, int x)//i 点增量为 x
{
    while(i <= maxn)
    {
        c[i] += x;
        i += Lowbit(i);
    }
}
int sum(int x){//区间求和 [1,x]
    int ans = 0;
    for(int i = x; i >= 1; i -= Lowbit(i))
        ans += c[i];
    return ans;
}
```

树状数组改段求段

```
const int N = 4e5 + 100;
template<class T>
struct Tree{
    T c[2][N];
    int maxn;
    void init(int x){
        maxn = x+10; memset(c, 0, sizeof c);
    }
    inline int lowbit(int x){ return x&-x; }
    T sum(T *b, int x){
        T ans = 0;
        if (x == 0)ans = b[0];
        while (x)ans += b[x], x -= lowbit(x);
        return ans;
    }
    void change(T *b, int x, T value){
        if (x == 0)b[x] += value, x++;
        while (x <= maxn)b[x] += value, x += lowbit(x);
    }
    T get_pre(int r){
        return sum(c[0], r) * r + sum(c[1], r);
    }
    void add(int l, int r, T value){//区间加权
        change(c[0], l, value);
        change(c[0], r + 1, -value);
        change(c[1], l, value * (-l + 1));
        change(c[1], r + 1, value * r);
```

```
    }
    T get(int l, int r){//区间求和
        return get_pre(r) - get_pre(l - 1);
    }
};
Tree<ll> tree;
```

RMQ

```
const int MAXN = 100100;
int n, query;
int A[MAXN];
int FMin[MAXN][20], FMax[MAXN][20];
void Init() {
    int i, j;
    for (i = 1;i <= n;i++)
        FMin[i][0] = FMax[i][0] = A[i];
    for (i = 1;(1 << i) <= n;i++) {
    for (j = 1;j + (1 << i) - 1 <= n;j++) {
            FMin[j][i] = min(FMin[j][i - 1], FMin[j +
(1 << (i - 1))][i - 1]);
            FMax[j][i] = max(FMax[j][i - 1], FMax[j +
(1 << (i - 1))][i - 1]);
        }
    }
}

int Query(int l, int r) {
    int k = (int)(log(double(r - l + 1)) /
log((double)2));
    return max(FMax[l][k], FMax[r - (1 << k) + 1][k]);
}
int main() {
    int i, a, b;
    scanf("%d %d", &n, &query);
    for (i = 1;i <= n;i++) scanf("%d", &A[i]);
    Init();
    while (query--) {
        scanf("%d %d", &a, &b);
        printf("%d\n", Query(a, b));
    }
    return 0;
}
```

树链剖分

```
#include<stdio.h>
#include<string.h>
#include<iostream>
#include<stdlib.h>
#define N 60003
#define L(x) (x<<1)
#define R(x) (x<<1|1)
#define Mid(x,y) ((x+y)>>1)
#define inf 10000000
using namespace std;

struct Edge{
    int to, nex;
}edge[N*2];
int head[N], edgenum;
void add(int u, int v){
    Edge E={v,head[u]};  edge[edgenum] = E;
    head[u] = edgenum++;
}

int n, Query, a[N];

int fa[N];//父节点
int dep[N];//深度
int son[N];//重儿子
int size[N];//子树节点数
int p[N]; //p[v]表示 v 在线段树中的位置
int fp[N]; //与 p 数组相反
int top[N]; // top[v] 表示 v 所在的重链的顶端节点 (这样 v
与 top[v] 的重路径可以直接在线段树上操作)

int dfs(int u, int Father, int deep){
    fa[u] = Father;      size[u] = 1; dep[u] = deep;
    for(int i = head[u]; ~i; i = edge[i].nex){
        int v = edge[i].to;
        if(v == Father)continue;
        size[u] += dfs(v, u, deep+1);
        if(son[u] == -1 || size[v] >
size[son[u]])son[u] = v;
    }
    return size[u];
}

int tree_id;
//tree_id 表示线段树中所有的边（给每条边编号）
void Have_p(int u, int Father){
    top[u] = Father;
```

```
    p[u] = tree_id++; fp[p[u]] = u;
    if(son[u] == -1)return ;
    Have_p(son[u], top[u]); //注意这里
    for(int i = head[u]; ~i; i = edge[i].nex){
        int v = edge[i].to;
        if(v != son[u] && v != fa[u])Have_p(v, v);
 //注意这里
    }
}

struct node{
    int l, r;
    int Max, Sum;
}tree[N*8];
void push_up(int id){
    tree[id].Max = max(tree[L(id)].Max,
tree[R(id)].Max);
    tree[id].Sum = tree[L(id)].Sum + tree[R(id)].Sum;
}
void build(int l, int r, int id){
    tree[id].l = l, tree[id].r = r;
    if(l == r)
    {
        tree[id].Sum = tree[id].Max = a[fp[l]];
//注意这里
        return ;
    }
    int mid = Mid(l, r);
    build(l, mid, L(id));
    build(mid+1, r, R(id));
    push_up(id);
}

void updata(int pos, int val, int id){
    if(tree[id].l == tree[id].r)
    {
        tree[id].Max = tree[id].Sum = val;
        return ;
    }
    int mid = Mid(tree[id].l, tree[id].r);
    if(pos <= mid)updata(pos, val, L(id));
    else updata(pos, val, R(id));
    push_up(id);
}
int querySum(int l, int r, int id){
    if(l == tree[id].l && tree[id].r == r)
        return tree[id].Sum;
    int mid = Mid(tree[id].l, tree[id].r);
    if(r<=mid) return querySum(l, r, L(id));
    else if(mid<l)return querySum(l, r, R(id));
    return querySum(l, mid, L(id)) + querySum(mid+1,
r, R(id));
}
int queryMax(int l, int r, int id){
    if(l == tree[id].l && tree[id].r == r)
        return tree[id].Max;
    int mid = Mid(tree[id].l, tree[id].r);
    if(r<=mid) return queryMax(l, r, L(id));
    else if(mid<l)return queryMax(l, r, R(id));
    return max(queryMax(l, mid, L(id)),
queryMax(mid+1, r, R(id)));
}
int findSum(int u, int v){
    int f1 = top[u], f2 = top[v];
    int tmp = 0;
    while(f1 != f2)
//相等就说明两点在同一重路径上 || 相同点
    {
        if(dep[f1]<dep[f2]) swap(f1, f2), swap(u, v);
        tmp += querySum(p[f1], p[u], 1);
//每次只爬其中一条重链!
        u = fa[f1];
        f1 = top[u];
    }
    if(dep[u] > dep[v])swap(u, v);
    return tmp + querySum(p[u], p[v], 1);
}
int findMax(int u, int v){
    int f1 = top[u], f2 = top[v];
    int tmp = -inf;
    while(f1 != f2)
    {
        if(dep[f1]<dep[f2]) swap(f1, f2), swap(u, v);
        tmp = max(tmp, queryMax(p[f1], p[u], 1));
        u = fa[f1];
        f1 = top[u];
    }
    if(dep[u] > dep[v])swap(u, v);
    return max(tmp, queryMax(p[u], p[v], 1));
}

void init(){
```

```
    memset(head, -1, sizeof(head));
    memset(son, -1, sizeof(son));  //注意这里 son 初始化
    edgenum = 0;
    tree_id = 1; //这个的所有可用编号就是线段树的区间 注意这里是从 1 开始的
}
char op[10];
int main(){
    int u, v, i;
    while(~scanf("%d",&n)){
        init();
        for(i = 0; i < n-1; i++)
{scanf("%d %d",&u,&v);    add(u,v), add(v,u);  }
        for(i = 1; i <= n; i++)scanf("%d",&a[i]);
        dfs(1, 1, 0);
        Have_p(1, 1);
        build(1, n, 1);
        scanf("%d",&Query);
        while(Query--){
            scanf("%s%d%d",op,&u,&v);
            if(op[0] == 'C')
                updata(p[u],v,1);
            else if(op[1] == 'M')
                printf("%d\n", findMax(u, v));
            else
                printf("%d\n", findSum(u, v));
        }
    }
    return 0;
}
```

Treap 树

```
#include <stdio.h>
#include <iostream>
#include <algorithm>
#include <math.h>
#include <vector>
#include <set>
#include <map>
#include <queue>
using namespace std;
#define L(id) tree[id].ch[0]
#define R(id) tree[id].ch[1]
#define Size(id) tree[id].size
#define Father(id) tree[id].fa
#define Val(id) tree[id].val
#define ll int
ll Mid(ll x,ll y){return (x+y)>>1;}
#define N 30100

ll a[N], n;
int ch[N][2],val[N],counts[N],r[N],size[N],tot,root;
int Newnode(int &rt,int v)
{
    rt=++tot;
    val[rt]=v;
    ch[rt][0]=ch[rt][1]=0;
    counts[rt]=size[rt]=1;
    r[rt]=rand();
    return rt;
}
inline void PushUp(int rt)
{
    size[rt]=size[ch[rt][0]]+size[ch[rt][1]]+counts[rt];
}
void Rotate(int &x,int kind)
{
    int y=ch[x][kind^1];
    ch[x][kind^1]=ch[y][kind];
    ch[y][kind]=x;
    PushUp(x);PushUp(y);
    x=y;
}
int Insert(int &rt,int v)
{
    if(rt==0)
        return Newnode(rt,v);
    int ans;
    if(v==val[rt]) counts[rt]++, ans = rt;
    else
    {
        int kind=(v>val[rt]);
        ans = Insert(ch[rt][kind],v);
        if(r[ch[rt][kind]]<r[rt])
            Rotate(rt,kind^1);
    }
    PushUp(rt);
    return ans;
}
```

```
int select(int rt,int k)
{
    if(size[ch[rt][0]]>=k) return
select(ch[rt][0],k);
    if(size[ch[rt][0]]+counts[rt]>=k) return
val[rt];
    return
select(ch[rt][1],k-size[ch[rt][0]]-counts[rt]);
}
void remove(int &rt,int v)
{
    if(val[rt]==v)
    {
        if(counts[rt]>1)
            counts[rt]--;
        else if(!ch[rt][0]&&!ch[rt][1])
        {rt=0;return ;}
        else
        {
            int kind=r[ch[rt][0]]<r[ch[rt][1]];
            Rotate(rt,kind);
            remove(rt,v);
        }
    }
    else remove(ch[rt][v>val[rt]],v);
    PushUp(rt);
}
void Init()
{
    ch[0][0]=ch[0][1]=0;
    size[0]=counts[0]=val[0]=0;
    tot=root=0;
    r[0]=(1LL<<31)-1;
    Newnode(root,2000000001);
}
int q[N];
char s[2];
int main(){
    int que, i, j, k, l, r;
    while(~scanf("%d %d",&n,&que)){
        Init();
        while(n--){
            scanf("%s",s);
    if(s[0]=='I')scanf("%d",&l),Insert(root,-l);
            else printf("%d\n",-select(root,que));
        }
    }
    return 0;
}
```

Splay 树

```
#include<stdio.h>
#include<iostream>
#include<algorithm>
#include<string.h>

using namespace std;
inline int Mid(int a,int b){return (a+b)>>1;}
#define inf 100000000
#define N 510000
#define L(x) tree[x].ch[0]
#define R(x) tree[x].ch[1]
#define Siz(x) tree[x].siz
#define Father(x) tree[x].fa
#define Val(x) tree[x].val
#define Lsum(x) tree[x].lsum
#define Rsum(x) tree[x].rsum
#define Sum(x) tree[x].sum
#define Subsum(x) tree[x].subsum
#define Filp(x) tree[x].filp
#define Cha(x) tree[x].change
struct node{
    int ch[2], siz, fa;
    int val, lsum, rsum, sum, subsum, filp;
    bool change;
}tree[N];
int tot, root;
int num[N], hehe;//内存池
void Newnode(int &id, int val, int fa, int siz = 1){
    if(hehe)id = num[hehe--];
    else
    id = ++tot;
    L(id) = R(id) = 0;
    Father(id) = fa;
    Siz(id) = siz;
    Val(id) = Sum(id) = Subsum(id) = Lsum(id) = Rsum(id)
= val;
    Cha(id) = Filp(id) = 0;
}
void Change(int id, int v){
    if(id == 0)return;
```

```
    Cha(id) = 1;
    Val(id) = v;
    Sum(id) = v*Siz(id);
    Lsum(id) = Rsum(id) = Subsum(id) = max(v,Sum(id));
}
void Filp_id(int id){
    if(id == 0)return ;
    Filp(id) ^= 1;
    swap(Lsum(id), Rsum(id));
    swap(L(id), R(id));
}

void push_up(int id){
    Siz(id) = Siz(L(id)) + Siz(R(id)) +1;
    Sum(id) = Sum(L(id)) + Sum(R(id)) + Val(id);
    Lsum(id) = max(Lsum(L(id)), Sum(L(id)) + Val(id));
    Lsum(id) = max(Lsum(id), Sum(L(id)) + Val(id) +
Lsum(R(id)));

    Rsum(id) = max(Rsum(R(id)), Sum(R(id)) + Val(id));
    Rsum(id) = max(Rsum(id), Sum(R(id)) + Val(id) +
Rsum(L(id)));

    Subsum(id) = max(Val(id), max(Subsum(L(id)),
Subsum(R(id))));
    Subsum(id) = max(Subsum(id),
max(Lsum(R(id))+Val(id), Rsum(L(id))+Val(id)));

    Subsum(id) = max(Subsum(id), Lsum(R(id)) +
Rsum(L(id))+Val(id));
}
void push_down(int id){
    if(Filp(id)){
        Filp(id) = 0;
        Filp_id(L(id));
        Filp_id(R(id));
    }
    if(Cha(id)){
        Cha(id) = 0;
        Change(L(id), Val(id));
        Change(R(id), Val(id));
    }
}

void Rotate(int id, int kind){
    int y = Father(id);
    push_down(y); push_down(id); //here
    tree[y].ch[kind^1] = tree[id].ch[kind];
    Father(tree[id].ch[kind]) = y;
    if(Father(y))
        tree[Father(y)].ch[R(Father(y))==y] = id;
    Father(id) = Father(y);
    Father(y) = id;
    tree[id].ch[kind] = y;
    push_up(y);
}
void splay(int id, int goal){
    push_down(id);
    while(Father(id) != goal){
        int y = Father(id);
        if(Father(y) == goal)
            Rotate(id, L(y)==id);
        else
        {
            int kind = L(Father(y)) == y;
            if(tree[y].ch[kind] == id)
            {
                Rotate(id, kind^1);
                Rotate(id, kind);
            }
            else
            {
                Rotate(y, kind);
                Rotate(id,kind);
            }
        }
    }
    push_up(id);
    if(goal == 0)root = id;
}
int Get_kth(int kth, int sor){//找到在 sor 后面的第 k 个
数
    push_down(sor);
    int id = sor;
    while(Siz(L(id)) != kth){
        if(Siz(L(id)) > kth)
            id = L(id);
        else
        {
            kth -= (Siz(L(id))+1);
            id = R(id);
        }
```

```
        push_down(id);
    }
    return id;
}
void Get_sec(int l, int r){//把区间[l,r]转到 L(R(root))
    splay(Get_kth(l-1,root), 0);
    splay(Get_kth(r+1, root), root);
}
//-----------------------------------------
int a[N], top;
void build(int l, int r, int &id, int fa, int *a){
    if(l>r)return;
    int mid = Mid(l, r);
    Newnode(id, a[mid], fa);
    build(l, mid-1, L(id), id, a);
    build(mid+1, r, R(id), id, a);
    push_up(id);
}
//-----------------------------------------
void Insert(int kth){//把 Stack 加到 kth 下
    Get_sec(kth+1,kth);
    build(0, top-1, L(R(root)), R(root), a);
    push_up(R(root));
    push_up(root);
}
void Filp_sec(int l, int r){
    Get_sec(l, r);
    Filp_id(L(R(root)));
    push_up(R(root));
    push_up(root);
}
void Erase(int id){//内存回收
    if(!id)return;
    num[++hehe] = id;
    Erase(L(id));
    Erase(R(id));
}
void Del_sec(int l, int r){//删除这个区间
    Get_sec(l,r);
    Erase(L(R(root)));
    L(R(root)) = 0;
    push_up(R(root));
    push_up(root);
}
void Change_sec(int l, int r, int v){
    Get_sec(l,r);
    Change(L(R(root)), v);
    push_up(R(root));
    push_up(root);
}
int Max_sum(){
    return Subsum(root);
}
int Get_sum(int l, int r){
    Get_sec(l, r);
    return Sum(L(R(root)));
}
char s[20];
int n, m;

void init(){
    hehe = 0;
    tot = root = 0;
    L(0) = R(0) = Siz(0) = Father(0) = 0;
    Val(0) = Sum(0) =  0;
    Subsum(0) = Lsum(0) = Rsum(0) = -inf;
    Newnode(root, -inf, 0);
    Newnode(R(root), -inf, root);
    build(1, n, L(R(root)), R(root), a);
    push_up(R(root));    push_up(root);
}
int main(){
    int i, j, k;
    while(~scanf("%d %d",&n,&m)){
        for(i = 1; i <= n; i++)scanf("%d",&a[i]);
        init();
        while(m--)
        {
            scanf("%s",s);
            if(s[0] == 'I')
            {
                scanf("%d %d",&i,&top);
                for(j = 0; j < top;
j++)scanf("%d",&a[j]);
                Insert(i);
            }
            else if(s[0] == 'D')
            {
                scanf("%d %d",&i,&j);
                Del_sec(i, i+j-1);
            }
            else if(s[2] == 'K')
```

```
            {
                scanf("%d %d %d",&i,&j,&k);
                Change_sec(i,i+j-1,k);
            }
            else if(s[0] == 'R')
            {
                scanf("%d %d",&i,&j);
                Filp_sec(i,i+j-1);
            }
            else if(s[0] == 'G')
            {
                scanf("%d %d",&i,&j);
                printf("%d\n",Get_sum(i,i+j-1));
            }
            else
                printf("%d\n",Max_sum());
        }
    }
    return 0;
}
/*
9 8
2 -6 3 5 1 -5 -3 6 3
GET-SUM 5 4
MAX-SUM
INSERT 8 3 -5 7 2
DELETE 12 1
MAKE-SAME 3 3 2
REVERSE 3 6
GET-SUM 5 4
MAX-SUM
*/
```

Link cut tree

```
typedef long long ll;
typedef pair<int, int> pii;
const int N = 30005;
const int inf = 10000000;
struct Node *null;
struct Node {
    Node *fa, *ch[2];
    int size;
    int val, ma, sum, id;
    bool rev;
    inline void put() {
    }
    inline void clear(int _val, int _id) {
        fa = ch[0] = ch[1] = null;
        size = 1;
        rev = 0;
        id = _id;
        val = ma = sum = _val;
    }
    inline void push_up() {
        size = 1 + ch[0]->size + ch[1]->size;
        sum = ma = val;
        if (ch[0] != null) {
            sum += ch[0]->sum;
            ma = max(ma, ch[0]->ma);
        }
        if (ch[1] != null) {
            sum += ch[1]->sum;
            ma = max(ma, ch[1]->ma);
        }
    }
    inline void push_down() {
        if (rev) {
            ch[0]->flip();
            ch[1]->flip();
            rev = 0;
        }
    }
    inline void setc(Node *p, int d) {
        ch[d] = p;
        p->fa = this;
    }
    inline bool d() {
        return fa->ch[1] == this;
    }
    inline bool isroot() {
        return fa == null || fa->ch[0] != this &&
fa->ch[1] != this;
    }
    inline void flip() {
        if (this == null)return;
        swap(ch[0], ch[1]);
        rev ^= 1;
    }
    inline void go() {//从链头开始更新到 this
        if (!isroot())fa->go();
        push_down();
    }
```

```
    inline void rot() {
        Node *f = fa, *ff = fa->fa;
        int c = d(), cc = fa->d();
        f->setc(ch[!c], c);
        this->setc(f, !c);
        if (ff->ch[cc] == f)ff->setc(this, cc);
        else this->fa = ff;
        f->push_up();
    }
    inline Node*splay() {
        go();
        while (!isroot()) {
            if (!fa->isroot())
                d() == fa->d() ? fa->rot() : rot();
            rot();
        }
        push_up();
        return this;
    }
    inline Node* access() {//access 后 this 就是到根的一条 splay，并且 this 已经是这个 splay 的根了
        for (Node *p = this, *q = null; p != null; q = p, p = p->fa) {
            p->splay()->setc(q, 1);
            p->push_up();
        }
        return splay();
    }
    inline Node* find_root() {
        Node *x;
        for (x = access(); x->push_down(), x->ch[0] != null; x = x->ch[0]);
        return x;
    }
    void make_root() {
        access()->flip();
    }
    void cut() {//把这个点的子树脱离出去
        access();
        ch[0]->fa = null;
        ch[0] = null;
        push_up();
    }
    void cut(Node *x) {
        if (this == x || find_root() != x->find_root())return;
        else {
            x->make_root();
            cut();
        }
    }
    void link(Node *x) {
        if (find_root() == x->find_root())return;
        else {
            make_root(); fa = x;
        }
    }
};
Node pool[N], *tail;
Node *node[N];
void init(int n) {
    tail = pool;
    null = tail++;
    null->clear(0, 0);
    null->size = 0;
    for (int i = 1; i <= n; i++) {
        node[i] = tail++;
        node[i]->clear(0, i);
    }
}
void debug(Node *x) {
    if (x == null)return;
    x->put();
    debug(x->ch[0]);
    debug(x->ch[1]);
}

int main() {
    int n;
    init(n);
    return 0;
}
```

树的点分治

题意：
给定 n 个点的树， K 值
下面 n-1 条边
问 两点之间距离<= K 的点对有多少

采用点分治，无根树转有根树时 根为树的重心（可以把树高度降低，防止树退化成链）

思路：
对于一棵 以 u 为根的树
以下我们成(a,b)为合法点对（即 dist(a,b) <=K)
(a, b) 之间路径是唯一确定的。

将点对分 2 类：
1、两点间路径经过 u 点
2、两点间路径不经过 u 点 = 两点都在 u 的同一子树下 （也就是 a,b 有个公共祖先 x ， x 是 u 的儿子节点 ）
显然合法点对数 = 第一类+第二类

对于第一类：//计算树的重心复杂度为 O(n) ，排序为 O(nlogn)
经过 u 点 点对数 = 所有 dist(a,b) <=K 数 - 求和（u 的儿子节点 v ）（v 子树中的合法节点数）
对于所有的 合法节点数 ，我们可以用单调性求出（具体实现在函数 getans() 中）
公式在 work 函数中实现。
这样计算完后，u 节点就直接删去（用 vis 数组记录）
这样会得到 u 的儿子形成的森林

对于第二类：//递归实现，由于用树的重心优化高度，递归次数为 log n
ans += u 的所有儿子作为第一类点时的答案

总的复杂度为 O(n*logn * logn)
注意一点： 计算树的重心时，树的总节点数会改变（用 size 表示当前的节点数）

```
#define N 10100

struct Edge { int to, nex, dis; }edge[N << 1];
int head[N], edgenum;
void addedge(int u, int v, int dis) { Edge E = { v, head[u], dis };edge[edgenum] = E;head[u] = edgenum++; }

int n, K, Ans;
int num[N];//num[i]表示 以 i 为根的树 节点数
int dp[N];//树重心的定义：dp[i]表示 将 i 点删去后 最大联通块的点数

int Stack[N], top1, root;
bool vis[N];//vis[i]表示 i 点是否删去
int size; //** 表示当前 计算的树的节点数

void getroot(int u, int fa) {//找树的重心
    dp[u] = 0; num[u] = 1; //以 u 为根的子树节点数
    for (int i = head[u]; ~i; i = edge[i].nex) {
        int v = edge[i].to;
        if (v != fa && !vis[v]) {
            getroot(v, u);
            num[u] += num[v];
            dp[u] = Max(num[v], dp[u]);
        }
    }
    dp[u] = Max(dp[u], size - num[u]);
    if (dp[u] < dp[root]) root = u;
}

int dis[N];
inline bool cmp(int i, int j) { return dis[i] < dis[j]; }

void Find_dis(int u, int fa, int d) { //把该树所有点入栈 并算出每个点到根节点的距离
    Stack[++top1] = u;
    dis[u] = d;
    for (int i = head[u]; ~i; i = edge[i].nex) {
        int v = edge[i].to;     if (vis[v] || v == fa)continue;
        Find_dis(v, u, d + edge[i].dis);
    }
}

int getans(int l, int r) {//计算树中 距离<=k 的点对数量
    int j = r, ans = 0;
    for (int i = l; i <= r; i++) {
        while (dis[Stack[i]] + dis[Stack[j]] > K && j>i)j--;
        if (i == j)return ans;
        ans += j - i;
    }
    return ans;
}
```

//先找到重心 (这样转成有根树不会造成树退化的情况) 对于此树计算后，删去根节点(vis 数组记录，因为已经求出所有经过根节点的合法点对，则根节点无用了)
//若合法点对存在于子树中 则要递归计算子树的点对(用 f 数组判断点对是否在一个子树中) 将子树的重心来转为有根树进行操作

```
void work(int u, int fa) {//计算出 路径经过 u 点的合法点对
                                //路径经过 u 点的合法点对数 = 总点对数 - 不经过 u 点(即点对都在一棵子树上）
```

```
    dp[0] = N; size = num[u];//注意初始化
    root = 0;

    getroot(u, fa);//root 为 u 的联通块中的重心

    top1 = 0;  int top2 = 0;
    for (int i = head[root]; ~i; i = edge[i].nex) {
        int v = edge[i].to;     if (vis[v]) continue;

        top2 = top1;            //Stack[top2+1] 到
Stack[top1] 之间的点就是 v 子树所有的点
        Find_dis(v, root, edge[i].dis);
        sort(Stack + top2 + 1, Stack + top1 + 1, cmp);
        Ans -= getans(top2 + 1, top1);
    }
    //Stack[1] - Stack[top1]就是 root 子树所有的点
    Stack[++top1] = root;     dis[root] = 0;
    sort(Stack + 1, Stack + top1 + 1, cmp);
    Ans += getans(1, top1);

    vis[root] = 1;//去掉 root 点

    for (int i = head[root]; ~i; i = edge[i].nex)//
路径不经过 u 点的点对数
        if (!vis[edge[i].to]) work(edge[i].to,
root);
}
void init() {
    memset(head, -1, sizeof(head)); edgenum = 0;
    memset(vis, 0, sizeof(vis));    Ans = 0;
}
int main() {
    while (scanf("%d %d", &n, &K), n + K) {
        init();
        for (int i = 1; i < n; i++)
        {
            int u, v, d; scanf("%d %d %d", &u, &v, &d);
            addedge(u, v, d);   addedge(v, u, d);
        }
        num[1] = n;
        work(1, 0);
        printf("%d\n", Ans);
    }
    return 0;
}
```

字符串

Hash

```
const int N = 200105;
typedef long long ll;
const int MAGIC = 311, MOD = 1e9 + 7;
template <class T>
struct HASH {
    ll h[N], base[N];
    inline void init(T *s, int len) {
        h[0] = 0;
        for (int i = 1; i <= len; ++i) h[i] = (h[i -
1] * MAGIC % MOD + s[i - 1]) % MOD;
        base[0] = 1;
        for (int i = 1; i <= len; ++i) base[i] = (base[i
- 1] * MAGIC) % MOD;
    }
    inline long long get(int l, int r) {
        return (h[r] - h[l - 1] * base[r - l + 1] % MOD
+ MOD) % MOD;
    }
};
HASH <int>A, B;
```

KMP

```
#include <stdio.h>
#include <string.h>
char T[10000],P[100];//从 0 开始存
int f[100];//记录 P 的自我匹配
void getFail(){
    int m=strlen(P);
    f[0]=f[1]=0;
    for(int i=1;i<m;i++){
        int j=f[i];
        while(j&&P[i]!=P[j])j=f[j];
        f[i+1]= P[i]==P[j] ? j+1 : 0;
    }
}

int find(){//返回第一个 P 在 T 中出现的位置
    int len1=strlen(T),len2=strlen(P);
    getFail();
    int j=0;
    for(int i=0;i<len1;i++)
```

```
    {
        while(j&&P[j]!=T[i])j=f[j];
        if(P[j]==T[i])j++;
//到这一步，j 就代表 T[i]已经匹配了前面 j 个 P 的字符串
        if(j==len2)return i - len2 + 1;
        }
    return -1; //表示 P 不存在于 T
}
```

Manacher

```
const int MAXN = 110010;
char Ma[MAXN * 2];
int Mp[MAXN * 2];
int Manacher(char s[]) {
    int l = 0, len = strlen(s);
    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 ans = 0;
    for (int i = 0; i<2 * len + 2; i++)
        ans = max(ans, Mp[i] - 1);
    return ans;
}
```

字典树

```
#include <stdio.h>
#include <iostream>
#include <string.h>
#include <algorithm>
#include <math.h>
using namespace std;

#define ll int
#define Word_Len 50500
#define Sigma_size 95
//Word_Len 是字典树的节点数 若都是小写字母 Sigma_size=26
sz 为当前节点数

struct Trie{
    ll ch[Word_Len][Sigma_size], sz;
    ll Have_word[Word_Len];
    ll val[Word_Len];
    ll pre[Word_Len];
    char he[Word_Len];
    ll Newnode()
    {
        memset(ch[sz], 0, sizeof(ch[sz]));
        val[sz]=Have_word[sz]=0; return sz++;
    }
    void init()
    { sz = 0; Newnode();}//初始化
    ll idx(char c){return c-32;}
    int insert(char *s){
        ll u = 0;
        for(ll i = 0; s[i]; i++){
            ll c = idx(s[i]);
            if(!ch[u][c])
            {
                ch[u][c] = Newnode();
                he[sz-1] = s[i];
                val[sz-1] = val[u]+1;
                pre[sz-1] = u;
            }
            u = ch[u][c];
        }
        Have_word[u]++;
        return u;
    }
    ll find_word(char *s){
        ll u = 0;
        for(ll i = 0; s[i]; i++){
            ll c = idx(s[i]);
            if(!ch[u][c])return 0;
            u = ch[u][c];
        }
        return Have_word[u];
    }
```

```
    void Have_name(char *s, ll now){
        ll len = val[now], cc = now;
        s[len--] = 0;
        while(cc)
        {
            s[len--] = he[cc];
            cc = pre[cc];
        }
    }
} ac;
```

AC 自动机

应用： 把多个模式串建成字典树，字典树中有多少个单词出现在母串中

注意 RE 的情况，maxnode=单词数*单词长度

```
const int maxnode = 250*1000+10000;
const int sigma_size = 26;

struct Trie{
    int ch[maxnode][sigma_size];
    int val[maxnode];     //该单词在模式串中出现的次数
    int last[maxnode];
    int f[maxnode];       //失配数组
    int num[maxnode];     //该单词出现在文本串的次数
    int pre[maxnode];     //该单词的前驱
    int len[maxnode];     //以该单词结尾的单词长度
    int Char[maxnode];    //该单词对应的字母
    int road[maxnode];    //路径压缩优化 针对计算模式串出现的种数
    int sz;
    int Newnode()
    {
        val[sz] = f[sz] = last[sz] = len[sz] = num[sz] = 0;
        memset(ch[sz], 0, sizeof ch[sz]);
        return sz++;
    }
    void init(){
        sz=0;
        Newnode();
    }
    int idx(char c){ return c-'A'; }
    int insert(char *s){
        int u = 0;
        for(int i = 0, c; s[i] ;i++){
            c = idx(s[i]);
            if(!ch[u][c])
                ch[u][c] = Newnode();
            pre[ch[u][c]] = u;
            Char[ch[u][c]] = s[i];
            len[ch[u][c]] = len[u]+1;
            road[ch[u][c]] = 1;
            u = ch[u][c];
        }
        val[u] = 1;
        num[u] = 0;
        return u;
    }
    void getFail(){
        queue<int> q;
        for(int i = 0; i<sigma_size; i++)
            if(ch[0][i]) q.push(ch[0][i]);
        int r, c, u, v;
        while(!q.empty()){
            r = q.front(); q.pop();
            for(c = 0; c<sigma_size; c++){
                u = ch[r][c];
                if(!u)continue;
                q.push(u);
                v = f[r];
                while(v && ch[v][c] == 0) v = f[v];
//沿失配边走上去 如果失配后有节点 且 其子节点 c 存在则结束循环
                f[u] = ch[v][c];
            }
        }
    }
    void find(char *T){
    //计算模式串出现的个数：（每种多次出现算多次）
        int j = 0;
        for(int i = 0, c, temp; T[i] ; i++){
            c = idx(T[i]);
            while(j && ch[j][c]==0) j = f[j];
            j = ch[j][c];

            temp = j;
            while(temp){
                num[temp]++;
                temp = f[temp];
            }
        }
    }
```

```
    void find_kind(char *T, int &ans){
    //计算种数， 重复出现的不再计算(若多个询问则要在此处加 for(i=0->sz)lu[i]=1;
        int j = 0, i, c, temp;
        for(i = 0; T[i]; i++){
            c = idx(T[i]);
            while(j && ch[j][c] == 0) j = f[j];
            j = ch[j][c];
            temp = j;
            while(temp && road[temp]){
                if(val[temp])
                {
                    ++ans;
                    val[temp] = 0;
                }
                road[temp] = 0;
                temp = f[temp];
            }
        }
    }
}ac;
```

后缀数组

题意：

给定 2 个字符串，求最长公共子串的长度

思路：

把两个字符串相连得到 S，则他们的公共子串就是部分 S 的后缀子串的前缀。

因为是相同的子串，所以 sa 必然是相邻的，因此扫一下 height，若 sa[i] 与 sa[i-1] 的后缀分别在分割符$前后，那就是两个字符串的后缀，求其最长公共前缀（即 height[i]）就是一个公共子串。

```
#include <stdio.h>
#include <string.h>
#include <iostream>
#include <algorithm>
#include <math.h>
#include <set>
using namespace std;
#define rank Rank
/*
* 后缀数组
* DC3 算法，复杂度 O(n)
* 所有的相关数组都要开三倍
*待排序数组长度为 n,放在 0~n-1 中，在最后面补一个 0
*da(str ,n+1,sa,rank,height, , );//注意是 n+1;
*例如：
*n = 8;
*num[] = { 1, 1, 2, 1, 1, 1, 1, 2, $ };注意 num 最后一位为 0，其他大于 0
*rank[] = { 4, 6, 8, 1, 2, 3, 5, 7, 0 };rank[0~n-1] 为有效值，rank[n]必定为 0 无效值
*sa[] = { 8, 3, 4, 5, 0, 6, 1, 7, 2 };sa[1~n]为有效值，sa[0]必定为 n 是无效值
*height[]= { 0, 0, 3, 2, 3, 1, 2, 0, 1 };height[2~n] 为有效值
*
*/
const int MAXN=301000;
int rank[MAXN],height[MAXN];
#define F(x) ((x)/3+((x)%3==1?0:tb))
#define G(x) ((x)<tb?(x)*3+1:((x)-tb)*3+2)
int wa[MAXN*3],wb[MAXN*3],wv[MAXN*3],wss[MAXN*3];
int c0(int *r,int a,int b)
{
    return r[a] == r[b] && r[a+1] == r[b+1] && r[a+2] == r[b+2];
}
int c12(int k,int *r,int a,int b)
{
    if(k == 2)
        return r[a] < r[b] || ( r[a] == r[b] && c12(1,r,a+1,b+1) );
    else return r[a] < r[b] || ( r[a] == r[b] && wv[a+1] < wv[b+1] );
}
void sort(int *r,int *a,int *b,int n,int m)
{
    int i;
    for(i = 0;i < n;i++)wv[i] = r[a[i]];
    for(i = 0;i < m;i++)wss[i] = 0;
    for(i = 0;i < n;i++)wss[wv[i]]++;
    for(i = 1;i < m;i++)wss[i] += wss[i-1];
    for(i = n-1;i >= 0;i--)
        b[--wss[wv[i]]] = a[i];
}
void dc3(int *r,int *sa,int n,int m)
{
    int i, j, *rn = r + n;
    int *san = sa + n, ta = 0, tb = (n+1)/3, tbc = 0,
```

```
p;
    r[n] = r[n+1] = 0;
    for(i = 0;i < n;i++)if(i %3 != 0)wa[tbc++] = i;
    sort(r + 2, wa, wb, tbc, m);
    sort(r + 1, wb, wa, tbc, m);
    sort(r, wa, wb, tbc, m);
    for(p = 1, rn[F(wb[0])] = 0, i = 1;i < tbc;i++)
        rn[F(wb[i])] = c0(r, wb[i-1], wb[i]) ? p - 1 :
p++;
    if(p < tbc)dc3(rn,san,tbc,p);
    else for(i = 0;i < tbc;i++)san[rn[i]] = i;
    for(i = 0;i < tbc;i++) if(san[i] < tb)wb[ta++] =
san[i] * 3;
    if(n % 3 == 1)wb[ta++] = n - 1;
    sort(r, wb, wa, ta, m);
    for(i = 0;i < tbc;i++)wv[wb[i] = G(san[i])] = i;
    for(i = 0, j = 0, p = 0;i < ta && j < tbc;p++)
        sa[p] = c12(wb[j] % 3, r, wa[i], wb[j]) ?
wa[i++] : wb[j++];
    for(;i < ta;p++)sa[p] = wa[i++];
    for(;j < tbc;p++)sa[p] = wb[j++];
}
//str 和 sa 也要三倍
void da(int str[],int sa[],int rank[],int height[],int
n,int m)
{
    for(int i = n;i < n*3;i++)
        str[i] = 0;
    dc3(str, sa, n+1, m);
    int i,j,k = 0;
    for(i = 0;i <= n;i++)rank[sa[i]] = i;
    for(i = 0;i < n; i++)
    {
        if(k) k--;
        j = sa[rank[i]-1];
        while(str[i+k] == str[j+k]) k++;
        height[rank[i]] = k;
    }
}
char str[MAXN];
int r[MAXN];
int sa[MAXN];
int main()
{
    gets(str);
    int len1 = strlen(str);
    str[len1] = '$';
    gets(str+len1+1);
    int len = strlen(str), n = len;
    for(int i = 0; i < n; i++)r[i] = str[i];
    da(r, sa, rank, height, n, 200);
    int ans = 0;
    for(int i = 2; i <= n; i++)
        if((sa[i] < len1 && sa[i-1] > len1) ||
(sa[i]>len1 && sa[i-1] < len1))
            ans = max(ans , height[i]);
    printf("%d\n", ans);

    return 0;
}
/*
yeshowmuchiloveyoumydearmotherreallyicannotbelievei
t
yeaphowmuchiloveyoumydearmother
abcd
stedste
aaaa
aaaa
aaaa
aaaaa
*/
```

数论

extend_gcd + 中国剩余定理

已知 a,b (a>=0,b>=0)

求一组解 (x,y) 使得 (x,y)满足

gcd(a,b) = ax+by

下面代码中 d = gcd(a,b)，顺便求出 gcd

可以扩展成求等式 ax+by = c,但 c 必须是 d 的倍数才有解,即 (c%gcd(a,b))==0

注意求出的 x,y 可能为 0 或负数

```
#include<stdio.h>
#include<string.h>
#include<iostream>
#include<algorithm>
#include<math.h>
#include<set>
#include<queue>
#include<vector>
#include<map>
```

```
using namespace std;
#define ll __int64
ll gcd(ll a, ll b) {
    return b == 0 ? a : gcd(b, a%b);
}
void extend_gcd (ll a , ll b , ll& d, ll &x , ll &y)
{
    if(!b){d = a; x = 1; y = 0;}
    else {extend_gcd(b, a%b, d, y, x); y-=x*(a/b);}
}
ll china(ll l, ll r, ll *m, ll *a){ //下标[l,r] 方程
    x%m=a;
    ll lcm = 1;
    for(ll i = l; i <= r; i++)
        lcm = lcm/gcd(lcm,m[i])*m[i];
    for(ll i = l+1; i <= r; i++) {
        ll A = m[l], B = m[i], d, x, y, c = a[i]-a[l];
        extend_gcd(A,B,d,x,y);
        if(c%d)return -1;
        ll mod = m[i]/d;
        ll K = ((x*c/d)%mod+mod)%mod;
        a[l] = m[l]*K + a[l];
        m[l] = m[l]*m[i]/d;
    }
    if(a[l]==0)return lcm;
    return a[l];
}
```

自适应辛普森公式

暴力求解一重定积分。

形如 ans = 积分从 a->b F(x) 的公式

黑匣子调用：

先修改函数 double F(double x){}

然后 ans = ars(a, b, eps);

```
double F(double x){
    return 1;
}
double simpson(double a, double b){
    double c = a + (b-a)/2.0;
    return (F(a) +4*F(c) + F(b)) * (b-a) / 6.0;
}
double asr(double a, double b, double eps, double A){
    double c = a + (b-a) / 2.0;
    double L = simpson(a, c), R = simpson(c, b);
    if(fabs(L+R-A) <= 15*eps) return L+R+(L+R-A)/15.0;
    return asr(a, c, eps/2.0, L) + asr(c, b, eps/2.0,
R);
}
double asr(double a, double b, double eps){
    return asr(a, b, eps, simpson(a,b));
}
```

高斯消元 浮点数解

```
struct BigInt
{
    const static int mod = 10000;
    const static int DLEN = 4;
    int a[600],len;
    BigInt()
    {
        memset(a,0,sizeof(a));
        len = 1;
    }
    BigInt(int v)  {
        memset(a,0,sizeof(a));
        len = 0;
        do   {
            a[len++] = v%mod;
            v /= mod;
        }while(v);
    }
    BigInt(const char s[])  {
        memset(a,0,sizeof(a));
        int L = strlen(s);
        len = L/DLEN;
        if(L%DLEN)len++;
        int index = 0;
        for(int i = L-1;i >= 0;i -= DLEN)
        {
            int t = 0;
            int k = i - DLEN + 1;
            if(k < 0)k = 0;
            for(int j = k;j <= i;j++)
                t = t*10 + s[j] - '0';
            a[index++] = t;
        }
    }
    BigInt operator +(const BigInt &b)const
    {
        BigInt res;
        res.len = max(len,b.len);
        for(int i = 0;i <= res.len;i++)
            res.a[i] = 0;
        for(int i = 0;i < res.len;i++)
```

```
        {
res.a[i] += ((i < len)?a[i]:0)+((i < b.len)?b.a[i]:0);
        res.a[i+1] += res.a[i]/mod;
        res.a[i] %= mod;
        }
        if(res.a[res.len] > 0)res.len++;
        return res;
    }
    BigInt operator *(const BigInt &b)const  {
        BigInt res;
        for(int i = 0; i < len;i++)   {
            int up = 0;
            for(int j = 0;j < b.len;j++)
            {
            int temp = a[i]*b.a[j] + res.a[i+j] + up;
                res.a[i+j] = temp%mod;
                up = temp/mod;
            }
            if(up != 0)
                res.a[i + b.len] = up;
        }
        res.len = len + b.len;
        while(res.a[res.len - 1] == 0 &&res.len > 1)
            res.len--;
        return res;
    }
    void output()  {
        printf("%d",a[len-1]);
        for(int i = len-2;i >=0 ;i--)
            printf("%04d",a[i]);
        printf("\n");
    }
};
```

高斯消元 求整数解

```
const int MAXN = 220;
double a[MAXN][MAXN], x[MAXN];//方程的左边的矩阵和等式右边的值，求解之后 x 存的就是结果
int equ, var;

int Gauss()
{
    int i, j, k, col, max_r;
    for (k = 0, col = 0;k<equ&&col<var;k++, col++)
    {
        max_r = k;
        for (i = k + 1;i<equ;i++)
            if
(fabs(a[i][col])>fabs(a[max_r][col]))
            max_r = i;
        if (fabs(a[max_r][col])<eps)return 0;
        if (k != max_r)
        {
            for (j = col;j<var;j++)
                swap(a[k][j], a[max_r][j]);
            swap(x[k], x[max_r]);
        }
        x[k] /= a[k][col];
        for  (j  =  col  +  1;j<var;j++)a[k][j]  /=
a[k][col];
        a[k][col] = 1;
        for (i = 0;i<equ;i++)
            if (i != k)
            {
                x[i] -= x[k] * a[i][k];
                for (j = col + 1;j<var;j++)a[i][j] -=
a[k][j] * a[i][col];
                a[i][col] = 0;
            }
    }
    return 1;
}
void init(int e, int v) {//方程数和未知数个数
    memset(a, 0, sizeof a);
    memset(x, 0, sizeof x);
    equ = e; var = v;
}
```

欧拉函数

A^x = A^(x % Phi(C) + Phi(C)) (mod C) （x>=Phi(C)）

```
#include<math.h>
#include<string.h>

#define ll int
#define N 1000000
//prime[0,primenum)
ll prime[1000000], primenum;
bool Isprime[N+10] = {0};
//<=Max_Prime 的素数
```

```
void PRIME(ll Max_Prime){
    primenum = 0;
    Isprime[0] = Isprime[1] = 0;
    Isprime[2] = 1;
    prime[primenum++] = 2;
    for(ll i = 3; i <= Max_Prime; i++)
        Isprime[i] = i&1;
    for (ll i = 3; i <= Max_Prime; i+=2){
        if(Isprime[i])
            prime[primenum++] = i;
        for (ll j = 0; j < primenum; j++){
            if(prime[j] * i > Max_Prime)break;
            Isprime[prime[j]*i] = 0;
            if(i%prime[j] == 0)break;
        }
    }
}
ll phi[N+10];
void PHI(ll Max_Phi){
    PRIME(Max_Phi);
    for(ll i=1;i<=Max_Phi;i++)phi[i]=i;
    for(ll i=2;i<=Max_Phi;i++)
        if(Isprime[i])
            for(ll j=i;j<=Max_Phi;j+=i)
                phi[j]=phi[j]/i*(i-1);
}
int main(){
    PHI(N-1);
}
```

Exgcd 求逆元

求 x 在模为 mod 时的逆元：
exgcd（x，mod，x，y）
求出后，第三个参数就是逆元。
mod 可以不为质数

```
template <class T>
T exgcd(T a, T b, T &x, T &y) {
    if (!b) {
        x = 1, y = 0;
        return a;
    }
    T t, ret;
    ret = exgcd(b, a%b, x, y);
    t = x, x = y, y = t - a / b*y;
    return ret;
}
```

扩展 gcd+中国剩余定理

```
ll gcd(ll a, ll b) {
    return b == 0 ? a : gcd(b, a%b);
}
void extend_gcd(ll a, ll b, ll& d, ll &x, ll &y) {
    if (!b) { d = a; x = 1; y = 0; }
    else { extend_gcd(b, a%b, d, y, x); y -= x*(a / b); }
}
ll china(ll l, ll r, ll *m, ll *a) { //下标[l,r] 方程 x%m=a;
    ll lcm = 1;
    for (ll i = l; i <= r; i++)lcm = lcm / gcd(lcm, m[i])*m[i];
    for (ll i = l + 1; i <= r; i++) {
        ll A = m[l], B = m[i], d, x, y, c = a[i] - a[l];
        extend_gcd(A, B, d, x, y);
        if (c%d)return -1;
        ll mod = m[i] / d;
        ll K = ((x*c / d) % mod + mod) % mod;
        a[l] = m[l] * K + a[l];
        m[l] = m[l] * m[i] / d;
    }
    if (a[l] == 0)return lcm;
    return a[l];
}
```

高精度模版

```
struct BigInt
{
    const static int mod = 10000;
    const static int DLEN = 4;
    int a[600],len;
    BigInt()
    {
        memset(a,0,sizeof(a));
        len = 1;
    }
    BigInt(int v)  {
        memset(a,0,sizeof(a));
        len = 0;
        do   {
            a[len++] = v%mod;
            v /= mod;
```

```
        }while(v);
    }
    BigInt(const char s[])  {
        memset(a,0,sizeof(a));
        int L = strlen(s);
        len = L/DLEN;
        if(L%DLEN)len++;
        int index = 0;
        for(int i = L-1;i >= 0;i -= DLEN)
        {
            int t = 0;
            int k = i - DLEN + 1;
            if(k < 0)k = 0;
            for(int j = k;j <= i;j++)
                t = t*10 + s[j] - '0';
            a[index++] = t;
        }
    }
    BigInt operator +(const BigInt &b)const
    {
        BigInt res;
        res.len = max(len,b.len);
        for(int i = 0;i <= res.len;i++)
            res.a[i] = 0;
        for(int i = 0;i < res.len;i++)
        {    res.a[i] += ((i < len)?a[i]:0)+((i <
b.len)?b.a[i]:0);
        res.a[i+1] += res.a[i]/mod;
        res.a[i] %= mod;
        }
        if(res.a[res.len] > 0)res.len++;
        return res;
    }
    BigInt operator *(const BigInt &b)const  {
        BigInt res;
        for(int i = 0; i < len;i++)   {
            int up = 0;
            for(int j = 0;j < b.len;j++)
            {
                int temp = a[i]*b.a[j] + res.a[i+j]
+ up;
                res.a[i+j] = temp%mod;
                up = temp/mod;
            }
            if(up != 0)
                res.a[i + b.len] = up;
        }
        res.len = len + b.len;
        while(res.a[res.len - 1] == 0 &&res.len > 1)
            res.len--;
        return res;
    }
    void output()  {
        printf("%d",a[len-1]);
        for(int i = len-2;i >=0 ;i--)
            printf("%04d",a[i]);
        printf("\n");
    }
};
```

素数

```
ll prime[N],primenum;//有 primenum 个素数 math.h
void PRIME(ll Max_Prime){
    primenum=0;
    prime[primenum++]=2;
    for(ll i=3;i<=Max_Prime;i+=2)
    for(ll j=0;j<primenum;j++)
        if(i%prime[j]==0)break;
        else if(prime[j]>sqrt((double)i) ||
j==primenum-1)
        {
            prime[primenum++]=i;
            break;
        }
}
```

素数线性筛

```
const int maxn = 100000003;
int p[6666666], tot;
bool vis[maxn];
void get_prime(){
    int i, j;
    for (i = 2; i < maxn; i++) {
        if (!vis[i]) p[++tot] = i;
        for (j = 1; j <= tot; j++){
            if (p[j] * i >= maxn)break;
            vis[p[j] * i] = 1;
            if (i % p[j] == 0)break;
        }
    }
}
```

2 ^ 63 次数的因式分解

把 n 的素因子都存在栈 P 中

```
ll mult_mod(ll a, ll b, ll c) {
```

```
    a  %= c;
    b  %= c;
    ll ret  = 0;
    while (b) {
        if (b & 1) ret  = (ret  + a) % c;
        a  = (a  + a) % c;
        b  >>= 1;
    }
    return ret;
}
ll pow_mod(ll x, ll n, ll mod) {
    if (n  == 1) return x % mod;
    x  %= mod;
    ll tmp  = x, ret  = 1;
    while (n  > 0){
        if (n & 1) ret  = mult_mod(ret, tmp, mod);
        tmp  = mult_mod(tmp, tmp, mod);
        n  >>= 1;
    }
    return ret;
}

bool check(ll a, ll n, ll x, ll t) {
    ll ret  = pow_mod(a, x, n);
    ll last  = ret;
    for (int i  = 1; i  <= t; i ++) {
        ret  = mult_mod(ret, ret, n);
        if (ret  == 1 && last  != 1 && last  != n -
1) return true;
        last  = ret;
    }
    if (ret  != 1) return true;
    return false;
}
bool Miller_Rabin(ll n) {
    if (n  < 2) return false;
    if (n == 2 || n == 3 || n == 5 || n == 7) return true;
    if (n % 2 == 0 || n % 3 == 0 || n % 5 == 0 || n %
7 == 0) return false;

    ll x  = n  - 1, t  = 0;
    while ((x & 1) == 0) {
        x  >>= 1;
        t ++;
    }
    for (int i  = 0; i < S; i ++) {
        ll a  = rand() % (n - 1) + 1;
        if (check(a, n, x, t)) return false;
    }
    return true;
}
ll gcd(ll a, ll b) {
    if (a < 0) return gcd(-a, b);
    if (b < 0) return gcd(a, -b);
    while (a > 0 && b > 0) {
        if (a > b) a  %= b;
        else b  %= a;
    }
    return a + b;
}
ll Pollard_rho(ll x, ll c) {
    ll i  = 1, k  = 2;
    ll x0  = ((rand() % x) + x) % x;
    ll y  = x0;
    while (true) {
        i ++;
        x0  = (mult_mod(x0, x0, x) + c) % x;
        ll d  = gcd(y - x0, x);
        if (d  != 1 && d  != x) return d;
        if (y  == x0) return x;
        if (i  == k) {
            y  = x0;
            k  += k;
        }
    }

}
ll P[N], tot;

void findfac(ll n) {
    if (Miller_Rabin(n)) {
        P[tot++] = n;
        return ;
    }
    ll p  = n;
    while (p  >= n){
        p  = Pollard_rho(p, rand() % (n - 1) + 1);
    }
    findfac(p);
    findfac(n / p);

}
```

```
void main(){
    tot = 0;
    findfac(n);
}
```

随机测大素数

```
typedef long long ll;
ll GCD(ll a, ll b) { return b ? GCD(b, a % b) : a; }

ll MultiMod(ll a, ll b, ll n) { // a * b % n
    ll res = 0; MillarRabin
    a %= n;
    while (b > 0) {
        if (b & 1) {
            res += a;
            if (res >= n) res -= n;
        }
        a <<= 1;
        if (a >= n) a -= n;
        b >>= 1;
    }
    return res;
}
ll QuickMod(ll a, ll b, ll n) { // a ^ b % n
    ll res = 1;
    a %= n;
    while (b > 0) {
        if (b & 1) res = MultiMod(res, a, n);
        a = MultiMod(a, a, n); b >>= 1;
    }
    return res;
}
bool MillarRabin(ll n) { // 判断是否素数
    if (n == 2 || n == 3 || n == 5 || n == 7 || n =
= 11) return true;
    if (n == 1 || !(n & 1) || !(n % 3) || !(n % 5)
|| !(n % 7) || !(n % 11)) return false;
    ll t = 0, m = n - 1, x, y;
    while (!(m & 1)) { m >>= 1; t++; }
    for (int i = 0; i < 10; i++) {
        ll a = rand() % (n - 2) + 2;
        x = QuickMod(a, m, n);
        for (ll j = 0; j < t; j++) {
            y = MultiMod(x, x, n);
            if (y == 1 && x != 1 && x != n - 1) ret
urn false;
            x = y;
        }
        if (y != 1) return false;
    }
    return true;
}
```

矩阵快速幂

```
struct Matrix {
    int r, c;
    int val[N][N];
    void clear() {
        memset(val, 0, sizeof(val));
    }
    void One() {
        for (int i  = 0; i  < r; ++i)
            val[i][i] = 1;
    }
};
Matrix multi(Matrix a, Matrix b) {
    Matrix re;
    re.r  = a.r, re.c  = b.c;
    re.clear();
    for (int i  = 0; i  < re.r; ++i)
        for (int j  = 0; j  < re.c; ++j)
        for (int k  = 0; k  < a.c; ++k)
        re.val[i][j] = (re.val[i][j] + (a.val[i][k] *
b.val[k][j]) % mod) % mod;
    return re;
}
Matrix Pow(Matrix a, int x) {
    Matrix re;
    re.clear();
    re.r  = re.c  = a.r;
    re.One();
    while (x) {
        if (x  & 1)
            re  = multi(re, a);
        a  = multi(a, a);
        x  >>= 1;
    }
    return re;
}
```

计算几何

不共线凸包

```
Codeforces 50C
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <iostream>
using namespace std;
#define INF 999999999.9
#define PI acos(-1.0)
#define ll long long
struct Point
{
    ll x, y, dis;
}pt[400005], stack[400005], p0;
ll top, tot;
//计算几何距离
ll Dis(ll x1, ll y1, ll x2, ll y2)
{
    return (x1 - x2)*(x1 - x2) + (y1 - y2)*(y1 - y2);
}
//极角比较， 返回-1: p0p1 在 p0p2 的右侧，返回 0:p0,p1,p2 共线
int Cmp_PolarAngel(struct Point p1, struct Point p2, struct Point pb)
{
    double delta = (p1.x - pb.x)*(p2.y - pb.y) - (p2.x - pb.x)*(p1.y - pb.y);
    if (delta<0.0) return 1;
    else if (delta == 0.0) return 0;
    else return -1;
}

// 判断向量 p2p3 是否对 p1p2 构成左旋
bool Is_LeftTurn(struct Point p3, struct Point p2, struct Point p1)
{
    int type = Cmp_PolarAngel(p3, p1, p2);
    if (type<0) return true;
    return false;
}

//先按极角排，再按距离由小到大排
int Cmp(const void*p1, const void*p2)
{
    struct Point*a1 = (struct Point*)p1;
    struct Point*a2 = (struct Point*)p2;
    int type = Cmp_PolarAngel(*a1, *a2, p0);
    if (type<0) return -1;
    else if (type == 0)
    {
        if (a1->dis<a2->dis) return -1;
        else if (a1->dis == a2->dis) return 0;
        else return 1;
    }
    else return 1;
}
//求凸包
ll step[4][2] = { 0, 1, 0, -1, 1, 0, -1, 0 };
void Solve(ll n)
{
    ll i, k;
    p0.x = p0.y = INF;
    ll x, y;
    for (i = 0; i<n; i++)
    {
        scanf("%I64d %I64d", &x, &y);
        for (ll j = 0; j < 4; j++)
        {
            Point P = { x + step[j][0], y + step[j][1] };
            pt[i * 4 + j] = P;

            if (pt[i * 4 + j].y < p0.y)
            {
                p0.y = pt[i * 4 + j].y;
                p0.x = pt[i * 4 + j].x;
                k = i * 4 + j;
            }
            else if (pt[i * 4 + j].y == p0.y)
            {
                if (pt[i * 4 + j].x<p0.x)
                {
                    p0.x = pt[i * 4 + j].x;
                    k = i * 4 + j;
                }
            }
        }
    }
    n *= 4;
    pt[k] = pt[0];
    pt[0] = p0;
    for (i = 1; i<n; i++)
```

```
        pt[i].dis = Dis(pt[i].x, pt[i].y, p0.x, p0.y);
    qsort(pt + 1, n - 1, sizeof(struct Point), Cmp);
    //去掉极角相同的点
    tot = 1;
    for (i = 2; i<n; i++)
    if (Cmp_PolarAngel(pt[i], pt[i - 1], p0))
        pt[tot++] = pt[i - 1];
    pt[tot++] = pt[n - 1];
    top = 1;
    stack[0] = pt[0];
    stack[1] = pt[1];
    for (i = 2; i<tot; i++)
    {
        while (top >= 1 && Is_LeftTurn(pt[i],
stack[top], stack[top - 1]) == false)
            top--;
        stack[++top] = pt[i];
    }
}
inline ll Abs(ll x){ return x>0 ? x : -x; }
ll len(Point a, Point b){
    return Abs(a.x - b.x) + Abs(a.y - b.y) - min(Abs(a.x
- b.x), Abs(a.y - b.y));
}
int main()
{
    ll n, x, y;
    while (cin >> n)
    {
        Solve(n);
        ll ans = 0;
        for (ll i = 0; i < top; i++)
            ans += len(stack[i], stack[i + 1]);
        ans += len(stack[top], stack[0]);
        cout << ans << endl;
    }
    return 0;
}
```

共线凸包

HDU 4946

```
#include <cstdio>
#include <vector>
#include <algorithm>
#include <set>
using namespace std;
#define INF 999999999.9
#define PI acos(-1.0)
#define ll int
const int MAX_N = 507;
const double eps = 1e-6;

struct Point {
    int x, y, v;
    int id;
    Point() {}
    Point(int _x, int _y, int _v, int _id) {
        x = x, y = _y, v = _v, id = _id;
    }
    bool operator < (const Point &rhs) const {
        if (x != rhs.x) return x < rhs.x;
        return y < rhs.y;
    }
};

int v[MAX_N];
bool vis[MAX_N];
int n, top;
int ans[MAX_N];
Point P[MAX_N], p1[MAX_N];

double cross(Point a, Point b, Point c) {
    return (a.x - c.x) * (b.y - c.y) - (b.x - c.x) *
(a.y - c.y);
}

bool cmp(Point a, Point b) {
    if (a.y == b.y) return a.x < b.x;
    return a.y < b.y;
}
void graham() {
    sort(p1, p1 + n, cmp);
    top = 1;
    for (int i = 0; i < 2; i++) v[i] = i;
    for (int i = 2; i < n; i++) {
        while (top > 0 && cross(p1[i], p1[v[top]],
p1[v[top - 1]]) > 0) top--;
        v[++top] = i;
    }
    int len = top;
    v[++top] = n - 2;
    for (int i = n - 3; i >= 0; i--) {
        while (top > len && cross(p1[i], p1[v[top]],
```

```
p1[v[top - 1]]) > 0) top--;
        v[++top] = i;
    }
}

void Clear() {
    memset(ans, 0, sizeof ans);
    memset(p1, 0, sizeof p1);
    memset(P, 0, sizeof P);
    memset(v, 0, sizeof v);
    memset(vis, 0, sizeof vis);
}
const int N = 505;

struct E {
    int x, y, v, id, ok;
}s[N];
vector<E> G;
bool cmp1(const E a, const E b) {
    if (a.v != b.v) return a.v > b.v;
    if (a.x != b.x) return a.x < b.x;
    if (a.y != b.y) return a.y < b.y;
    return a.id < b.id;
}
int nn;
void put(int ttop){
    for (int i = 1; i <= nn; i++) printf("%d", ans[i]);
    puts("");
}

int main() {
    int cas = 0;
    while (~scanf("%d", &nn), nn) {
        Clear();
        printf("Case #%d: ", ++cas);
        memset(ans, 0, sizeof ans);
        for (int i = 0; i < nn; i++) {
            scanf("%d%d%d", &s[i].x, &s[i].y,
&s[i].v);
            s[i].id = i + 1;
            s[i].ok = 1;
            for (int j = 0; j < i; j++)
            if (s[i].x == s[j].x && s[i].y == s[j].y
&& s[i].v == s[j].v)
            {
                s[i].ok = 0;
                break;
            }
        }
        sort(s, s + nn, cmp1);
        if (s[0].v == 0){ put(12); continue; }
        int ttop = 0;
        while (ttop < nn && s[ttop].v == s[0].v)
ttop++;
        //----------------------------
        G.clear();
        for (int i = 0; i < ttop; i++)if (s[i].ok)
G.push_back(s[i]);
        bool gongxian = true;
        int x = 0, y = 0;
        for (int i = 1; i < ttop; i++)
        {
            if (s[i].x == s[i - 1].x && s[i].y == s[i
- 1].y)continue;
            if (x == 0 && y == 0) {
                x = s[i].x - s[i - 1].x;
                y = s[i].y - s[i - 1].y;
            }
            else if (s[i].x - s[i - 1].x != x || s[i].y
- s[i - 1].y != y)
            {
                gongxian = false;
                break;
            }
        }
        if (G.size() <= 2 || gongxian) {
            for (int i = 0; i < G.size(); i++)
            {
                bool ok = true;
                for (int j = 0; ok && j < ttop; j++)
                if (G[i].id != s[j].id && G[i].x ==
s[j].x && G[i].y == s[j].y)
                    ok = false;
                ans[G[i].id] = ok;
            }
            put(ttop); continue;
        }
        for (int i = 0; i < G.size(); i++)
        {
            p1[i].x = G[i].x;
            p1[i].y = G[i].y;
            p1[i].id = G[i].id;
```

```
        }
        n = G.size();
        graham();
        for (int i = 0; i <= top; i++)
        {
            bool ok = true;
            for (int j = 0; ok && j < ttop; j++)
            {
                if (p1[v[i]].id != s[j].id &&
p1[v[i]].x == s[j].x && p1[v[i]].y == s[j].y)
                    ok = false;
            }
            ans[p1[v[i]].id] = ok;
        }
        put(ttop);
    }
    return 0;
}
```

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <iostream>
#include <queue>
#include <algorithm>
using namespace std;
#define PR 1e-8
#define N 510
struct TPoint{
    double x, y, z;
    TPoint(){}
    TPoint(double _x, double _y, double _z):x(_x),
y(_y), z(_z){}
    TPoint operator-(const TPoint p){return
TPoint(x-p.x, y-p.y, z-p.z);}
    TPoint operator*(const TPoint p){return
TPoint(y*p.z-z*p.y, z*p.x-x*p.z, x*p.y-y*p.x);}
    double operator^(const TPoint p){return
x*p.x+y*p.y+z*p.z;}
};
struct fac{
    int a, b, c;
    bool ok;
};
struct T3dhull{
    int n;
    TPoint ply[N];
    int trianglecnt;
    fac tri[N];
    int vis[N][N];
    double dist(TPoint a){return
sqrt(a.x*a.x+a.y*a.y+a.z*a.z);}
    double area(TPoint a, TPoint b, TPoint c)
    { return dist((b-a)*(c-a));}
    double volume(TPoint a, TPoint b, TPoint c, TPoint
d)
    { return (b-a)*(c-a)^(d-a);}
    double ptoplane(TPoint &p, fac &f)
    {
        TPoint m = ply[f.b] - ply[f.a], n =
ply[f.c]-ply[f.a], t = p-ply[f.a];
        return (m*n)^t;
    }
    void deal(int p, int a, int b){
        int f = vis[a][b];
        fac add;
        if(tri[f].ok)
        {
            if((ptoplane(ply[p], tri[f])) > PR)
                dfs(p, f);
            else
            {
                add.a = b, add.b = a, add.c = p, add.ok
= 1;
                vis[p][b] = vis[a][p] = vis[b][a] =
trianglecnt;
                tri[trianglecnt++] = add;
            }
        }
    }
    void dfs(int p, int cnt) {
        tri[cnt].ok = 0;
        deal(p, tri[cnt].b, tri[cnt].a);
        deal(p, tri[cnt].c, tri[cnt].b);
        deal(p, tri[cnt].a, tri[cnt].c);
    }
    bool same(int s, int e) {
        TPoint a = ply[tri[s].a], b = ply[tri[s].b], c
= ply[tri[s].c];
        return fabs(volume(a,b,c,ply[tri[e].a])) < PR
            && fabs(volume(a,b,c,ply[tri[e].b])) < PR
```

```
            && fabs(volume(a,b,c,ply[tri[e].c])) < PR;
    }
    void construct()
    {
        int i, j;
        trianglecnt = 0;
        if(n<4) return ;
        bool tmp = true;
        for(i = 1; i < n; i++)
        {
            if((dist(ply[0]-ply[i])) > PR)
            {
                swap(ply[1], ply[i]);
                tmp = false;
                break;
            }
        }
        if(tmp)return ;
        tmp = true;
        for(i = 2; i < n; i++)
        {

if((dist((ply[0]-ply[1])*(ply[1]-ply[i]))) > PR)
            {
                swap(ply[2], ply[i]);
                tmp = false;
                break;
            }
        }
        if(tmp) return ;
        tmp = true;
        for(i = 3; i < n; i++)
        {

if(fabs((ply[0]-ply[1])*(ply[1]-ply[2])^(ply[0]-ply
[i]))>PR)
            {
                swap(ply[3], ply[i]);
                tmp =false;
                break;
            }
        }
        if(tmp)return ;
        fac add;
        for(i = 0; i < 4; i++)
        {
            add.a = (i+1)%4, add.b = (i+2)%4, add.c =
(i+3)%4, add.ok = 1;
            if((ptoplane(ply[i], add))>0)
                swap(add.b, add.c);
            vis[add.a][add.b] = vis[add.b][add.c] =
vis[add.c][add.a] = trianglecnt;
            tri[trianglecnt++] = add;
        }
        for(i = 4; i < n; i++)
        {
            for(j = 0; j < trianglecnt; j++)
            {
                if(tri[j].ok && (ptoplane(ply[i],
tri[j])) > PR)
                {
                    dfs(i, j); break;
                }
            }
        }
        int cnt = trianglecnt;
        trianglecnt = 0;
        for(i = 0; i < cnt; i++)
        {
            if(tri[i].ok)
                tri[trianglecnt++] = tri[i];
        }
    }
    double area()
    {
        double ret = 0;
        for(int i = 0; i < trianglecnt; i++)
            ret += area(ply[tri[i].a], ply[tri[i].b],
ply[tri[i].c]);
        return ret/2.0;
    }
    double volume()
    {
        TPoint p(0,0,0);
        double ret = 0;
        for(int i = 0; i < trianglecnt; i++)
            ret += volume(p, ply[tri[i].a],
ply[tri[i].b], ply[tri[i].c]);
        return fabs(ret/6);
    }
}hull;
```

```
int main(){
    int Cas = 1;
    while(scanf("%d",&hull.n), hull.n){
        int i ;
        for(i = 0; i < hull.n; i++)
            scanf("%lf %lf %lf",&hull.ply[i].x,
&hull.ply[i].y, &hull.ply[i].z);
        hull.construct();
        printf("Case %d: %.2lf\n", Cas++,
hull.area());
    }
    return 0;
}
```

其他

三维凸包

求三维凸包的表面积和体积

```
#include<cstdio>
#include<iostream>
#include<algorithm>
#include<string.h>
#include<math.h>
using namespace std;
#define point Point
const double eps = 1e-8;
const double PI = acos(-1.0);
double ABS(double x){return x>0?x:-x;}
int sgn(double x){
    if(fabs(x) < eps)return 0;
    if(x < 0)return -1;
    else return 1;
}
struct Point
{
    double x,y;
    void put(){printf("(%.0lf,%.0lf)\n",x,y);}
    Point(){}
    Point(double _x,double _y){
    x = _x;y = _y;
    }
    Point operator -(const Point &b)const{
        return Point(x - b.x,y - b.y);
    }
    //叉积
    double operator ^(const Point &b)const{
        return x*b.y - y*b.x;
    }
    //点积
    double operator *(const Point &b)const{
        return x*b.x + y*b.y;
    }
    //绕原点旋转角度 B（弧度值），后 x,y 的变化
    void transXY(double B){
        double tx = x,ty = y;
        x = tx*cos(B) - ty*sin(B);
        y = tx*sin(B) + ty*cos(B);
    }
};
struct Line
{
    Point s,e;
    void put(){s.put();e.put();}
    Line(){}
    Line(Point _s,Point _e)
    {
    s = _s;e = _e;
    }
    //两直线相交求交点
    //第一个值为 0 表示直线重合，为 1 表示平行，为 0 表示相
交,为 2 是相交
    //只有第一个值为 2 时，交点才有意义
    pair<int,Point> operator &(const Line &b)const{
        Point res = s;
        if(sgn((s-e)^(b.s-b.e)) == 0)
        {
        if(sgn((s-b.e)^(b.s-b.e)) == 0)
        return make_pair(0,res);//重合
        else return make_pair(1,res);//平行
        }
        double t =
((s-b.s)^(b.s-b.e))/((s-e)^(b.s-b.e));
        res.x += (e.x-s.x)*t;
        res.y += (e.y-s.y)*t;
        return make_pair(2,res);
    }
};
double dist(Point a,Point b){return
sqrt((a-b)*(a-b));}
//*判断线段相交
bool inter(Line l1,Line l2){
    return
    max(l1.s.x,l1.e.x) >= min(l2.s.x,l2.e.x) &&
```

```
    max(l2.s.x,l2.e.x) >= min(l1.s.x,l1.e.x) &&
    max(l1.s.y,l1.e.y) >= min(l2.s.y,l2.e.y) &&
    max(l2.s.y,l2.e.y) >= min(l1.s.y,l1.e.y) &&
    sgn((l2.s-l1.e)^(l1.s-l1.e))*sgn((l2.e-l1.e)^(l
1.s-l1.e)) <= 0 &&
    sgn((l1.s-l2.e)^(l2.s-l2.e))*sgn((l1.e-l2.e)^(l
2.s-l2.e)) <= 0;
}
point symmetric_point(point p1, point l1, point
l2){ //p1 关于直线(l1,l2)的对称点
    point ret;
    if(ABS(l1.x-l2.x)<eps){
        ret.y = p1.y;
        ret.x = 2*l1.x - p1.x;
        return ret;
    }
    if(ABS(l1.y-l2.y)<eps) {
        ret.x = p1.x;
        ret.y = 2*l1.y - p1.y;
        return ret;
    }
    if (l1.x > l2.x - eps && l1.x < l2.x + eps)
    {
    ret.x = (2 * l1.x - p1.x);
    ret.y = p1.y;
    }
    else
    {
    double k = (l1.y - l2.y ) / (l1.x - l2.x);
    ret.x = (2*k*k*l1.x + 2*k*p1.y - 2*k*l1.y -
k*k*p1.x + p1.x) / (1 + k*k);
    ret.y = p1.y - (ret.x - p1.x ) / k;
    }
    return ret;
}
bool gongxian(Point a, Point b, Point c){
    return ABS((a.y-b.y)*(a.x-c.x) -
(a.y-c.y)*(a.x-b.x))<eps;
}
```

输入输出挂

```
template <class T>
inline bool rd(T &ret) {
    char c; int sgn;
    if(c=getchar(),c==EOF) return 0;
    while(c!='-'&&(c<'0'||c>'9')) c=getchar();
    sgn=(c=='-')?-1:1;
    ret=(c=='-')?0:(c-'0');
    while(c=getchar(),c>='0'&&c<='9')
ret=ret*10+(c-'0');
    ret*=sgn;
    return 1;
}
template <class T>
inline void pt(T x) {
    if (x <0) {
        putchar('-');
        x = -x;
    }
    if(x>9) pt(x/10);
    putchar(x%10+'0');
}
cin cout 缓冲
std::ios::sync_with_stdio(false);
```

Java 读写挂

```
void work() throws Exception{
    }

  public static void main(String[] args) throws Exc
eption{
    Main wo = new Main();
        in = new BufferedReader(new InputStreamRead
er(System.in));
        out = new PrintWriter(System.out);
  //    in = new BufferedReader(new InputStreamRead
er(new FileInputStream(new File("input.txt"))));
  //    out = new PrintWriter(new File("output.txt")
);
        wo.work();
        out.close();
    }
    DecimalFormat df=new DecimalFormat("0.0000");

    private String Next() throws Exception{
       while (str == null || !str.hasMoreElements())
          str = new StringTokenizer(in.readLine());
        return str.nextToken();
    }
    private int Int() throws Exception{
        return Integer.parseInt(Next());
    }
```

```
    private long Long() throws Exception{
        return Long.parseLong(Next());
    }
    StringTokenizer str;
    static BufferedReader in;
    static PrintWriter out;
```

优先队列小的数先出来

```
#include<functional>
#include<queue>
using namespace std;
priority_queue<int, vector<int>, greater<int> > q;
```

Java 大数用法示例

注意 OJ 要求 java 的类名

```
import java.math.*;
import java.util.*;
import java.io.*;

public class Main {
    BigInteger[] a = new BigInteger[3007];

    public void work() {
        int T;
        T = cin.nextInt();
        while (T-- > 0) {
            int n;
            n = cin.nextInt();
            for (int i = 0; i < n; ++i)
                a[i] = cin.nextBigInteger();

            int j = n - 1;
            BigInteger C = BigInteger.ONE, sum =
BigInteger.ZERO;
            for (int i = 0; i < n; ++i) {
                if (i % 2 == 0) {
                    BigInteger d1 = C.multiply(a[j]);
                    sum = sum.add(d1);
                } else {
                    BigInteger d2 = C.multiply(a[j]);
                    sum = sum.subtract(d2);
                }
                C = C.multiply(BigInteger.valueOf(n - 1
- i)).divide(BigInteger.valueOf(i + 1));
                --j;
            }
            System.out.println(sum);
        }
    }
//这下面的都不用改
    Main() {
        cin = new Scanner(System.in);
    }
    public static void main(String[] args) {
        Main e = new Main();
        e.work();
    }
    public Scanner cin;
}
```