

## Contents

1	Basic	1
1.1	.vimrc	1
1.2	IncreaseStackSize	1
1.3	Default Code	1
2	Data Structure	2
2.1	Bigint	2
2.2	Leftist Heap	2
2.3	extc_balance_tree	3
2.4	Treap	3
3	Graph	3
3.1	Tarjan	3
3.2	Strongly Connected Components:Kosaraju's Algorithm	3
3.3	DMST_with_sol	4
4	Flow	5
4.1	ISAP	5
4.2	Bipartite Matching (Augmenting Path)	5
4.3	SW-Mincut	5
4.4	Maximum Simple Graph Matching	6
4.5	2-Commodity Flow	6
5	Math	7
5.1	ax+by=gcd	7
5.2	Chinese Remainder	7
5.3	Miller Rabin	7
5.4	Mod	8
5.5	Primes	8
6	Geometry	8
6.1	Point operators	8
6.2	Minimum Covering Circle	8
6.3	Intersection of two circles	9
6.4	Intersection of two lines	9
6.5	Half line Intersection	9
7	Stringology	10
7.1	Suffix Array	10
7.2	Suffix Array (SAIS TWT514)	10
7.3	Aho-Corasick Algorithm	11
7.4	Z value	11
7.5	Z value (palindrome ver.)	11
7.6	Suffix Automaton	12
8	Problems	12
8.1	Qtree IV	12
8.2	Find the maximum tangent (x,y is increasing)	13
9	+lironwood's code	13
9.1	KDTreeAndNearestPoint	13
9.2	MinkowskiSum	14
9.3	MinimumMeanCycle	15
9.4	PolynomialGenerator	15
9.5	SwGeneralGraphMaxMatching	15
9.6	stoer-wagner-nm	16

## 1.2 IncreaseStackSize

```

1 //stack resize
1 asm( "mov %0,%esp\n" ::"g"(mem+10000000) );
1 //change esp to rsp if 64-bit system
1
2 //stack resize (linux)
2 #include <sys/resource.h>
2 void increase_stack_size() {
2     const rlim_t ks = 64*1024*1024;
2     struct rlimit rl;
2     int res=getrlimit(RLIMIT_STACK, &rl);
2     if(res==0){
2         if(rl.rlim_cur<ks){
2             rl.rlim_cur=ks;
2             res=setrlimit(RLIMIT_STACK, &rl);
2         }
2     }
2 }

```

## 1.3 Default Code

```

7 #include<bits/stdc++.h>
7 #include<cmath>
7 #include<cstdio>
7 #include<cstring>
7 #include<cstdlib>
7 #include<iostream>
7 #include<algorithm>
7 #include<vector>
7 using namespace std;
7 #define _SZ(n) memset((n),0,sizeof(n))
7 #define _SMO(n) memset((n),-1,sizeof(n))
7 #define _MC(n,m) memcpy((n),(m),sizeof(n))
7 #define _F first
7 #define _S second
7 #define _MP make_pair
7 #define _PB push_back
7 #define FOR(x,y) for(__typeof(y.begin())x=y.begin();x!=
7 y.end();x++)
7 #define IOS ios_base::sync_with_stdio(0)
7 // Let's Fight!
7
14 int main()
14 {
14     return 0;
14 }

```

## 1 Basic

### 1.1 .vimrc

```

colo torte
syn on
se cin ai ar sm nu ru is
se mouse=a bs=2 ww+=<, >, [, ] so=6 ts=4 sw=4 ttm=100
se makeprg=g++\ -Wall\ -Wshadow\ -O2\ -std=c++0x\ -o\
%<\ %
au BufNewFile *.cpp 0r ~/default.cpp

map <F7> <ESC>:wa<CR>:make!<CR>
imap <F7> <ESC>:wa<CR>:make!<CR>
map <C-F7> <ESC>:tabe %<.in<CR>
map <F8> :cope <CR>
map <S-F8> :ccl <CR>
map <F9> :!./%< <CR>
map <C-F9> :!./%< < %<.in <CR>

```

## 2 Data Structure

### 2.1 Bigint

```
const int bL = 1000;
const int bM = 10000;

struct Bigint{
    int v[bL],l;
    Bigint(){ memset(v, 0, sizeof(v));l=0; }

    void n(){
        for(;l;l--) if(v[l-1]) return;
    }

    Bigint(long long a){
        for(l=0;a;v[l++]=a%bM,a/=bM);
    }
    Bigint(char *a){
        l=0;
        int t=0,i=strlen(a),q=1;
        while(i){
            t+=(a[--i]-'0')*q;
            if((q*=10)>=bM) {
                v[l++]=t; t=0; q=1;
            }
        }
        if(t) v[l++]=t;
    }

    void prt() {
        if(l==0){ putchar('0');return; }
        printf("%d",v[l-1]);
        for(int i=l-2;i>=0;i--) printf("%.4d",v[i]);
    }

    int cp3(const Bigint &b)const {
        if(l!=b.l) return l>b.l?-1:-1;
        for(int i=l-1;i>=0;i--)
            if(v[i]!=b.v[i])
                return v[i]>b.v[i]?1:-1;
        return 0;
    }

    bool operator < (const Bigint &b)const{ return cp3(b)==-1; }
    bool operator == (const Bigint &b)const{ return cp3(b)==0; }
    bool operator > (const Bigint &b)const{ return cp3(b)==1; }

    Bigint operator + (const Bigint &b) {
        Bigint r;
        r.l=max(l,b.l);
        for(int i=0;i<r.l;i++) {
            r.v[i]=v[i]+b.v[i];
            if(r.v[i]>=bM) {
                r.v[i+1]+=r.v[i]/bM;
                r.v[i]%=bM;
            }
        }
        if(r.v[r.l]) r.l++;
        return r;
    }

    Bigint operator - (const Bigint &b) {
        Bigint r;
        r.l=l;
        for(int i=0;i<l;i++) {
            r.v[i]=v[i];
            if(i<b.l) r.v[i]-=b.v[i];
            if(r.v[i]<0) {
                r.v[i]+=bM;
                r.v[i+1]--;
            }
        }
        r.n();
        return r;
    }
}
```

```
Bigint operator * (const Bigint &b) {
    Bigint r;
    r.l=l+b.l;
    for(int i=0;i<l;i++) {
        for(int j=0;j<b.l;j++) {
            r.v[i+j]+=v[i]*b.v[j];
            if(r.v[i+j]>=bM) {
                r.v[i+j+1]+=r.v[i+j]/bM;
                r.v[i+j]%=bM;
            }
        }
    }
    r.n();
    return r;
}

Bigint operator / (const Bigint &b) {
    Bigint r;
    r.l=max(1,l-b.l+1);
    for(int i=r.l-1;i>=0;i--) {
        int d=0,u=bM-1,m;
        while(d<u) {
            m=(d+u)>>1;
            r.v[i]=m;
            if((r*b)>(*this)) u=m-1;
            else d=m;
        }
        r.v[i]=d;
    }
    r.n();
    return r;
}

Bigint operator % (const Bigint &b) {
    return (*this)-(*this)/b*b;
}
};
```

### 2.2 Leftist Heap

```
const int MAXSIZE = 10000;

class Node{
public:
    int num,lc,rc;
    Node () : num(0), lc(-1), rc(-1) {}
    Node (int _v) : num(_v), lc(-1), rc(-1) {}
}tree[MAXSIZE];

int merge(int x, int y){
    if (x == -1) return y;
    if (y == -1) return x;
    if (tree[x].num < tree[y].num)
        swap(x, y);
    tree[x].rc = merge(tree[x].rc, y);
    swap(tree[x].lc, tree[x].rc);
    return x;
}

/* Usage
merge: root = merge(x, y)
delmin: root = merge(root.lc, root.rc)
*/
```

## 2.3 extc\_balance\_tree

```
#include <bits/extc++.h>
using namespace __gnu_pbds;
typedef tree<int, null_type, less<int>, rb_tree_tag,
    tree_order_statistics_node_update> set_t;

int main()
{
    // Insert some entries into s.
    set_t s;
    s.insert(12);
    s.insert(505);

    // The order of the keys should be: 12, 505.
    assert(*s.find_by_order(0) == 12);
    assert(*s.find_by_order(3) == 505);

    // The order of the keys should be: 12, 505.
    assert(s.order_of_key(12) == 0);
    assert(s.order_of_key(505) == 1);

    // Erase an entry.
    s.erase(12);

    // The order of the keys should be: 505.
    assert(*s.find_by_order(0) == 505);

    // The order of the keys should be: 505.
    assert(s.order_of_key(505) == 0);
}
```

## 2.4 Treap

```
class Node{
public:
    int pri,num,cnt,lc,rc;
    Node () : pri(-1), num(0), cnt(0), lc(0), rc(0) {}
    Node (int _num){
        pri = (rand()<<15) + rand();
        num = _num;
        cnt = 1;
        lc = rc = 0;
    }
}tree[MX];

int nMem;

int get_rand(){
    return (rand()<<15) + rand();
}

int get_node(){
    tree[nMem] = Node();
    if (nMem >= MX) while(1);
    return nMem++;
}

void upd_node(int rt){
    if (!rt) return ;
    int lc=tree[rt].lc;
    int rc=tree[rt].rc;
    tree[rt].cnt = tree[lc].cnt + tree[rc].cnt + 1;
}

int merge(int a, int b){
    if (!a) return b;
    if (!b) return a;
    int res=0;
    if (tree[a].pri > tree[b].pri){
        res = a; //get_node();
        tree[res] = tree[a];
        tree[res].rc = merge(tree[res].rc,b);
    } else {
        res = b; //get_node();
        tree[res] = tree[b];
        tree[res].lc = merge(a,tree[res].lc);
    }
    upd_node(res);
    return res;
}
```

```
pair<int,int> split(int a, int k){
    if (k == 0) return MP(0,a);
    if (k == tree[a].cnt) return MP(a,0);
    int lc=tree[a].lc, rc=tree[a].rc;
    pair<int,int> res;
    int np=a; //get_node();
    //tree[np] = tree[a];
    if (tree[lc].cnt >= k){
        res = split(lc,k);
        tree[np].lc = res._S;
        res._S = np;
    } else {
        res = split(rc,k-tree[lc].cnt-1);
        tree[np].rc = res._F;
        res._F = np;
    }
    upd_node(res._F);
    upd_node(res._S);
    return res;
}
```

## 3 Graph

### 3.1 Tarjan

```
const int MAXV = 101000;

int V, E;
vector<int> el[MAXV];
int dfn[MAXV], low[MAXV], did;
bool ins[MAXV];
stack<int> st;
int scc[MAXV], scn;

void tarjan(int u){
    cout << u << endl;
    dfn[u] = low[u] = ++did;
    st.push(u); ins[u] = true;

    for(int i=0; i<(int)el[u].size(); i++){
        int v = el[u][i];
        if(!dfn[v]){
            tarjan(v);
            low[u] = min(low[u], low[v]);
        } else if(ins[v]){
            low[u] = min(low[u], dfn[v]);
        }
    }

    if(dfn[u] == low[u]){
        int v;
        do{
            v = st.top();
            st.pop();
            scc[v] = scn;
            ins[v] = false;
        }while(v != u);
        scn ++;
    }
}

void calcscc(){
    did = scn = 0;
    for(int i=0; i<V; i++){
        if(!dfn[i]) tarjan(i);
    }
}
```

### 3.2 Strongly Connected Components: Kosaraju's Algorithm

```
class Scc{
public:
    int n,vst[MAXN];
    int nScc,bln[MAXN];
    vector<int> E[MAXN], rE[MAXN], vc;
```

Compo-

```

void init(int _n){
    n = _n;
    for (int i=0; i<MAXN; i++){
        E[i].clear();
        rE[i].clear();
    }
}
void add_edge(int u, int v){
    E[u]._PB(v);
    rE[v]._PB(u);
}
void DFS(int u){
    vst[u]=1;
    FOR(it, E[u]){
        if (!vst[*it])
            DFS(*it);
    }
    vc._PB(u);
}
void rDFS(int u){
    vst[u] = 1;
    bln[u] = nScc;
    FOR(it, rE[u]){
        if (!vst[*it])
            rDFS(*it);
    }
}
void solve(){
    nScc=0;
    vc.clear();
    _SZ(vst);
    for (int i=0; i<n; i++){
        if (!vst[i])
            DFS(i);
    }
    reverse(vc.begin(),vc.end());
    _SZ(vst);
    FOR(it,vc){
        if (!vst[*it]){
            rDFS(*it);
            nScc++;
        }
    }
}
};

```

### 3.3 DMST\_with\_sol

```

const int INF = 1029384756;

struct edge_t{
    int u,v,w;
    set< pair<int,int> > add,sub;
    edge_t(){
        u = -1;
        v = -1;
        w = 0;
    }
    edge_t(int _u, int _v, int _w){
        u = _u;
        v = _v;
        w = _w;
        add.insert(_MP(_u,_v));
    }
    edge_t& operator += (const edge_t& obj) {
        w += obj.w;
        FOR (it, obj.add) {
            if (!sub.count(*it)) add.insert(*it);
            else sub.erase(*it);
        }
        FOR (it, obj.sub) {
            if (!add.count(*it)) sub.insert(*it);
            else add.erase(*it);
        }
        return *this;
    }
}

edge_t& operator -= (const edge_t& obj) {
    w -= obj.w;
    FOR (it, obj.sub) {

```

```

        if (!sub.count(*it)) add.insert(*it);
        else sub.erase(*it);
    }
    FOR (it, obj.add) {
        if (!add.count(*it)) sub.insert(*it);
        else add.erase(*it);
    }
    return *this;
}
}eg[MXN*MXN],prv[MXN],EDGE_INF(-1,-1,INF);
int N,M;
int cycid,incycle[MXN],contracted[MXN];
vector<int> E[MXN];

edge_t dmst(int rt){
    edge_t cost;
    for (int i=0; i<N; i++){
        contracted[i] = 0;
        incycle[i] = 0;
        prv[i] = EDGE_INF;
    }
    cycid = 0;
    int u,v;
    while (true){
        for (v=0; v<N; v++){
            if (v != rt && !contracted[v] && prv[v].w
                == INF)
                break;
        }
        if (v >= N) break; // end
        for (int i=0; i<M; i++){
            if (eg[i].v == v && eg[i].w < prv[v].w){
                prv[v] = eg[i];
            }
        }
        if (prv[v].w == INF){ // not connected
            return EDGE_INF;
        }
        cost += prv[v];
        for (u=prv[v].u; u!=v && u!=-1; u=prv[u].u);
        if (u == -1) continue;
        incycle[v] = ++cycid;
        for (u=prv[v].u; u!=v; u=prv[u].u){
            contracted[u] = 1;
            incycle[u] = cycid;
        }
        for (int i=0; i<M; i++){
            if (incycle[eg[i].u] != cycid && incycle[eg
                [i].v] == cycid){
                eg[i] -= prv[eg[i].v];
            }
        }
        for (int i=0; i<M; i++){
            if (incycle[eg[i].u] == cycid) eg[i].u = v;
            if (incycle[eg[i].v] == cycid) eg[i].v = v;
            if (eg[i].u == eg[i].v) eg[i--] = eg[--M];
        }
        for (int i=0; i<N; i++){
            if (contracted[i]) continue;
            if (prv[i].u>=0 && incycle[prv[i].u] ==
                cycid)
                prv[i].u = v;
        }
        prv[v] = EDGE_INF;
    }
    return cost;
}

void solve(){
    edge_t cost = dmst(0);
    FOR(it,cost.add){ // find a solution
        E[it->_F]._PB(it->_S);
        prv[it->_S] = edge_t(it->_F,it->_S,0);
    }
}

```

## 4 Flow

### 4.1 ISAP

```

class Isap{
public:
    class Edge{
    public:
        int v,f,re;
        Edge (){ v=f=re=-1; }
        Edge (int _v, int _f, int _r){
            v = _v;
            f = _f;
            re = _r;
        }
    };
    int n,s,t,h[N],gap[N];
    vector<Edge> E[N];
    void init(int _n, int _s, int _t){
        n = _n;
        s = _s;
        t = _t;
        for (int i=0; i<N; i++){
            E[i].clear();
        }
    }
    void add_edge(int u, int v, int f){
        E[u]._PB(Edge(v,f,E[v].size()));
        E[v]._PB(Edge(u,f,E[u].size()-1));
    }
    int DFS(int u, int nf, int res=0){
        if (u == t) return nf;
        FOR(it,E[u]){
            if (h[u]==h[it->v]+1 && it->f>0){
                int tf = DFS(it->v,min(nf,it->f));
                res += tf;
                nf -= tf;
                it->f -= tf;
                E[it->v][it->re].f += tf;
                if (nf == 0) return res;
            }
        }
        if (nf){
            if (--gap[h[u]] == 0) h[s]=n;
            gap[++h[u]]++;
        }
        return res;
    }
    int flow(int res=0){
        _SZ(h);
        _SZ(gap);
        gap[0] = n;
        while (h[s] < n)
            res += DFS(s,2147483647);
        return res;
    }
}flow;

```

### 4.2 Bipartite Matching (Augmenting Path)

```

bool DFS(int u){
    FOR(it,E[u]){
        if (!vst[*it]){
            vst[*it]=1;
            if (match[*it] == -1 || DFS(match[*it])){
                match[*it] = u;
                match[u] = *it;
                return true;
            }
        }
    }
    return false;
}
int DoMatch(int res=0){
    MSET(match,-1);
    for (int i=1; i<=m; i++){
        if (match[i] == -1){
            memset(vst,0,sizeof(vst));
            DFS(i);
        }
    }
}

```

```

    }
}
for (int i=1; i<=m; i++){
    if (match[i] != -1) res++;
}
return res;
}

```

### 4.3 SW-Mincut

```

typedef long long LL;
typedef long double LD;
typedef std::pair<int,int> PII;

const int N=514;
const int INF=2147483647>>1;

int n, m, del[N], vst[N], wei[N], rd[N][N];

PII sw(){
    MSET(vst,0);
    MSET(wei,0);
    int p1=-1,p2=-1,mx,cur=0;
    while(1){
        mx=-1;
        REP(i,1,n){
            if (!del[i] && !vst[i] && mx<wei[i]){
                cur=i;
                mx=wei[i];
            }
        }
        if (mx==-1) break;
        vst[cur]=1;
        p1=p2;
        p2=cur;
        REP(i,1,n)
            if (!vst[i] && !del[i])
                wei[i]+=rd[cur][i];
    }
    return std::MP(p1,p2);
}

void input(){
    REP(i,1,n){
        del[i]=0;
        REP(j,1,n)
            rd[i][j] = 0;
    }
    REP(i,1,m){
        int u,v,c;
        scanf("%d%d%d",&u,&v,&c);
        ++u; ++v;
        rd[u][v]+=c;
        rd[v][u]+=c;
    }
}

void solve(){
    int ans=INF;
    PII tmp;
    REP(i,1,n-1){
        tmp=sw();
        int x=tmp.F;
        int y=tmp.S;
        if (wei[y] < ans) ans=wei[y];
        del[y]=1;
        REP(j,1,n){
            rd[j][x]+=rd[j][y];
            rd[x][j]+=rd[y][j];
        }
    }
    printf("%d\n", ans);
}

int main(){
    while (~scanf("%d%d",&n, &m)){
        input();
        solve();
    }
    return 0;
}

```

## 4.4 Maximum Simple Graph Matching

```

const int MAX = 300;

int V, E;
int el[MAX][MAX];
int mtp[MAX];
int djs[MAX];
int bk[MAX], pr[MAX], vt[MAX];
queue<int> qu;

int ffa(int a){
    return (djs[a] == -1) ? a : djs[a] = ffa(djs[a]);
}

void djo(int a, int b){
    int fa = ffa(a), fb = ffa(b);
    if (fa != fb) djs[fb] = fa;
}

int lca(int u, int v){
    static int ts = 0;
    ts++;
    while(1){
        if( u != -1 ){
            u = ffa(u);
            if(vt[u] == ts) return u;
            vt[u] = ts;
            if(pr[u] != -1) u = bk[pr[u]];
            else u = -1;
        }
        swap(u, v);
    }
    return u;
}

void flower(int u, int w){
    while(u != w){
        int v1 = pr[u], v2 = bk[v1];
        if(ffa(v2) != w) bk[v2] = v1;
        if(mtp[v1] == 1){
            qu.push(v1);
            mtp[v1] = 0;
        }
        if(mtp[v2] == 1){
            qu.push(v2);
            mtp[v2] = 0;
        }
        djo(v1, w);
        djo(v2, w);
        djo(u, w);
        u = v2;
    }
}

bool flow(int s){
    memset(mtp, -1, sizeof(mtp));
    while(qu.size()) qu.pop();
    qu.push(s);
    mtp[s] = 0; bk[s] = pr[s] = -1;

    while(qu.size() && pr[s] == -1){
        int u = qu.front(); qu.pop();
        for(int v=0; v<V; v++){
            if (el[u][v] == 0) continue;
            if (ffa(v) == ffa(u)) continue;

            if(pr[v] == -1){
                do{
                    int t = pr[u];
                    pr[v] = u; pr[u] = v;
                    v = t; u = t== -1? -1: bk[t];
                }while( v != -1 );
                break;
            }else if(mtp[v] == 0){
                int w = lca(u, v);
                if(ffa(w) != ffa(u)) bk[u] = v;
                if(ffa(w) != ffa(v)) bk[v] = u;
                flower(u, w);
                flower(v, w);
            }else if(mtp[v] != 1){

```

```

                bk[v] = u;
                mtp[v] = 1;
                mtp[pr[v]] = 0;
                qu.push(pr[v]);
            }
        }
    }
    return pr[s] != -1;
}

int match(){
    memset(pr, -1, sizeof(pr));
    int a = 0;
    for (int i=0; i<V; i++){
        if (pr[i] == -1){
            if(flow(i)) a++;
            else mtp[i] = i;
        }
    }
    return a;
}

```

## 4.5 2-Commodity Flow

```

const int MAXN = 64;
const int INF = 1029384756;

int N;
int s1, s2, t1, t2, d1, d2, S, T;
int edge[MAXN][MAXN];
int cap[MAXN][MAXN];

int h[MAXN], gap[MAXN];
bool vis[MAXN];

int isap(int v, int f)
{
    if(v == T) return f;

    if(vis[v]) return 0;
    vis[v] = true;

    for(int i=0; i<N+2; i++){
        if(cap[v][i] <= 0) continue;
        if(h[i] != h[v] - 1) continue;
        int res = isap(i, min(cap[v][i], f));
        if(res > 0)
        {
            cap[v][i] -= res;
            cap[i][v] += res;
            return res;
        }
    }

    gap[h[v]]--;
    if(gap[h[v]] <= 0) h[S] = N + 4;
    h[v]++;
    gap[h[v]]++;

    return 0;
}

int get_flow()
{
    for(int i=0; i<MAXN; i++){
        h[i] = gap[i] = 0;
    }
    gap[0] = N + 2;

    int flow = 0;

    while(h[S] <= N + 3)
    {
        for(int i=0; i<N+2; i++){
            vis[i] = false;

```

```

    }

    int df = isap(S, INF);
    flow += df;
}

return flow;
}

int main()
{
    ios_base::sync_with_stdio(0);

    int TT;
    cin>>TT;
    while(TT-->0)
    {
        cin>>N;
        cin>>s1>>t1>>d1>>s2>>t2>>d2;

        for(int i=0; i<MAXN; i++)
        {
            for(int j=0; j<MAXN; j++)
            {
                edge[i][j] = 0;
            }
        }

        for(int i=0; i<N; i++)
        {
            string s;
            cin>>s;
            for(int j=0; j<N; j++)
            {
                if(s[j] == 'X') edge[i][j] = 0;
                else if(s[j] == 'O') edge[i][j] = 1;
                else if(s[j] == 'N') edge[i][j] = INF;
            }
        }

        int ans = 0;

        S = N;
        T = N + 1;

        //first
        for(int i=0; i<MAXN; i++)
        {
            for(int j=0; j<MAXN; j++)
            {
                cap[i][j] = edge[i][j];
            }
        }

        cap[S][s1] = cap[t1][T] = d1;
        cap[S][s2] = cap[t2][T] = d2;

        ans = get_flow();

        //second
        for(int i=0; i<MAXN; i++)
        {
            for(int j=0; j<MAXN; j++)
            {
                cap[i][j] = edge[i][j];
            }
        }

        cap[S][s1] = cap[t1][T] = d1;
        cap[S][t2] = cap[s2][T] = d2;

        ans = min(ans, get_flow());

        cout<<(ans == d1 + d2 ? "Yes" : "No")<<endl;
    }

    return 0;
}

```

## 5 Math

### 5.1 $ax+by=\gcd$

```

typedef pair<int, int> pii;

pii gcd(int a, int b){
    if(b == 0) return make_pair(1, 0);
    else{
        int p = a / b;
        pii q = gcd(b, a % b);
        return make_pair(q.second, q.first - q.second * p);
    }
}

```

### 5.2 Chinese Remainder

```

int pfn; // number of distinct prime factors
int pf[MAXNUM]; // prime factor powers
int rem[MAXNUM]; // corresponding remainder
int pm[MAXNUM];
inline void generate_primes() {
    int i, j;
    pnum=1;
    prime[0]=2;
    for(i=3; i<MAXVAL; i+=2) {
        if(!nprime[i]) continue;
        prime[pnum++]=i;
        for(j=i*i; j<MAXVAL; j+=i) nprime[j]=1;
    }
}
inline int inverse(int x, int p) {
    int q, tmp, a=x, b=p;
    int a0=1, a1=0, b0=0, b1=1;
    while(b) {
        q=a/b; tmp=b; b=a-b*q; a=tmp;
        tmp=b0; b0=a0-b0*q; a0=tmp;
        tmp=b1; b1=a1-b1*q; a1=tmp;
    }
    return a0;
}
inline void decompose_mod() {
    int i, p, t=mod;
    pfn=0;
    for(i=0; i<pnum&&prime[i]<=t; i++) {
        p=prime[i];
        if(t%p==0) {
            pf[pfn]=1;
            while(t%p==0) {
                t/=p;
                pf[pfn]*=p;
            }
            pfn++;
        }
    }
    if(t>1) pf[pfn]=t;
}
inline int chinese_remainder() {
    int i, m, s=0;
    for(i=0; i<pfn; i++) {
        m=mod/pf[i];
        pm[i]=(long long)m*inverse(m, pf[i])%mod;
        s=(s+(long long)pm[i]*rem[i])%mod;
    }
    return s;
}

```

### 5.3 Miller Rabin

```

long long power(long long x, long long p, long long mod){
    long long s=1, m=x;
    while(p) {
        if(p&1) s=mult(s, m, mod);
        p>>=1;
        m=mult(m, m, mod);
    }
}

```

```

    return s;
}
bool witness(long long a, long long n, long long u, int t)
{
    long long x = power(a, u, n);
    for(int i=0; i<t; i++) {
        long long nx = mult(x, x, n);
        if(nx==1 && x!=1 && x!=n-1) return 1;
        x=nx;
    }
    return x!=1;
}
bool miller_rabin(long long n, int s=100) {
    // iterate s times of witness on n
    // return 1 if prime, 0 otherwise
    if(n<2) return 0;
    if(!(n&1)) return n==2;
    long long u=n-1;
    int t=0;
    // n-1 = u*2^t
    while(u&1) {
        u>>=1;
        t++;
    }
    while(s-->0) {
        long long a=randll()%(n-1)+1;
        if(witness(a, n, u, t)) return 0;
    }
    return 1;
}

```

## 5.4 Mod

```

/// _fd(a,b) floor(a/b).
/// _rd(a,m) a-floor(a/m)*m.
/// _pv(a,m,r) largest x s.t x<=a && x%m == r.
/// _nx(a,m,r) smallest x s.t x>=a && x%m == r.
/// _ct(a,b,m,r) |A|, A = { x : a<=x<=b && x%m == r }.

int _fd(int a, int b){ return a<0?(-~a/b-1):a/b; }
int _rd(int a, int m){ return a-_fd(a,m)*m; }
int _pv(int a, int m, int r)
{
    r=(r%m+m)%m;
    return _fd(a-r,m)*m+r;
}
int _nt(int a, int m, int r)
{
    m=abs(m);
    r=(r%m+m)%m;
    return _fd(a-r-1,m)*m+r+m;
}
int _ct(int a, int b, int m, int r)
{
    m=abs(m);
    a=_nt(a,m,r);
    b=_pv(b,m,r);
    return (a>b)?0:((b-a+m)/m);
}

```

## 5.5 Primes

```

/*
* 12721
* 13331
* 14341
* 75577
* 123457
* 222557
* 556679
* 999983
* 1097774749
* 1076767633
* 100102021
* 999997771
* 1001010013
* 1000512343
* 987654361

```

```

* 999991231
* 999888733
* 98789101
* 987777733
* 999991921
* 1010101333
* 1010102101
*/

```

## 6 Geometry

### 6.1 Point operators

```

#include<bits/stdc++.h>
using namespace std;

#define _x first
#define _y second
typedef pair<double, double> pdd;

pdd operator + (const pdd p1, const pdd p2){
    return pdd(p1._x + p2._x, p1._y + p2._y);
}
pdd operator - (const pdd p1, const pdd p2){
    return pdd(p1._x - p2._x, p1._y - p2._y);
}

pdd operator * (const double c, const pdd p){
    return pdd(p._x * c, p._y * c);
}
pdd operator - (const pdd p){
    return (-1.0) * p;
}

double operator * (const pdd p1, const pdd p2){
    return p1._x * p2._x + p1._y * p2._y;
}
double operator % (const pdd p1, const pdd p2){
    return p1._x * p2._y - p2._x * p1._y;
}

```

### 6.2 Minimum Covering Circle

```

struct Point{
    typedef double T;
    T x, y;

    Point() : x(0), y(0) {}
    Point(T _x, T _y) : x(_x), y(_y) {}

    bool operator < (const Point &b) const{
        return tie(x,y) < tie(b.x,b.y);
    }
    bool operator == (const Point &b) const{
        return tie(x,y) == tie(b.x,b.y);
    }
    Point operator + (const Point &b) const{
        return Point(x+b.x, y+b.y);
    }
    Point operator - (const Point &b) const{
        return Point(x-b.x, y-b.y);
    }
    T operator * (const Point &b) const{
        return x*b.x + y*b.y;
    }
    T operator % (const Point &b) const{
        return x*b.y - y*b.x;
    }
    Point operator * (const T &b) const{
        return Point(x*b, y*b);
    }
    T len(){
        return sqrt(len2());
    }
    T len2(){
        return x*x + y*y;
    }
}

```



```

};

const int MAXN = 1000100;
struct Mcc{
    // return pair of center and r^2
    int n;
    Point p[MAXN], cen;
    double r2;

    void init(int _n, Point _p[]){
        n = _n;
        memcpy(p, _p, sizeof(Point)*n);
    }
    double sqr(double a){ return a*a; }
    Point center(Point p0, Point p1, Point p2) {
        Point a = p1-p0;
        Point b = p2-p0;
        double c1=a.len2()*0.5;
        double c2=b.len2()*0.5;
        double d = a % b;
        double x = p0.x + (c1 * b.y - c2 * a.y) / d;
        double y = p0.y + (a.x * c2 - b.x * c1) / d;
        return Point(x,y);
    }

    pair<Point,double> solve(){
        random_shuffle(p,p+n);
        r2=0;
        for (int i=0; i<n; i++){
            if ((cen-p[i]).len2() <= r2) continue;
            cen = p[i];
            r2 = 0;
            for (int j=0; j<i; j++){
                if ((cen-p[j]).len2() <= r2) continue;
                cen = Point((p[i].x+p[j].x)*0.5, (p[i].y+p[j].y)*0.5);
                r2 = (cen-p[j]).len2();
                for (int k=0; k<j; k++){
                    if ((cen-p[k]).len2() <= r2) continue;
                    cen = center(p[i],p[j],p[k]);
                    r2 = (cen-p[k]).len2();
                }
            }
        }
        return _MP(cen,r2);
    }
}mcc;

```

### 6.3 Intersection of two circles

Let  $O_1 = (x_1, y_1), O_2 = (x_2, y_2)$  be two centers of circles,  $r_1, r_2$  be the radius. If:

$$d = |O_1 - O_2| \quad u = \frac{1}{2}(O_1 + O_2) + \frac{(r_2^2 - r_1^2)}{2d^2}(O_1 - O_2)$$

$v = \frac{\sqrt{(r_1+r_2+d)(r_1-r_2+d)(r_1+r_2-d)(-r_1+r_2+d)}}{2d^2}(y_1 - y_2, -x_1 + x_2)$  then  $u + v, u - v$  are the two intersections of the circles, provided that  $d < r_1 + r_2$ .

### 6.4 Intersection of two lines

```

#include<bits/stdc++.h>

using namespace std;
const double EPS = 1e-9;

pdd interPnt(pdd p1, pdd p2, pdd q1, pdd q2){
    double f1 = (p2 - p1) % (q1 - p1);
    double f2 = (p2 - p1) % (p1 - q2);
    double f = (f1 + f2);

    if(fabs(f) < EPS) return pdd(nan(""), nan(""));

    return (f2 / f) * q1 + (f1 / f) * q2;
}

```

### 6.5 Half line Intersection

```

#include<bits/stdc++.h>

using namespace std;

#define _PB push_back
#define _MP make_pair
#define _x first
#define _y second

const int MXL = 5000;
const double EPS = 1e-8;

typedef pair<double, double> pdd;
typedef pair<pdd, pdd> Line;

pdd operator + (const pdd p1, const pdd p2){
    return pdd(p1._x + p2._x, p1._y + p2._y);
}

pdd operator - (const pdd p1, const pdd p2){
    return pdd(p1._x - p2._x, p1._y - p2._y);
}

pdd operator * (const double c, const pdd p){
    return pdd(p._x * c, p._y * c);
}

double operator % (const pdd p1, const pdd p2){
    return p1._x * p2._y - p2._x * p1._y;
}

vector<Line> lnlst;
double atn[MXL];

bool lncmp(int l1, int l2){
    return atn[l1] < atn[l2];
}

pdd interPnt(pdd p1, pdd p2, pdd q1, pdd q2){
    double f1 = (p2 - p1) % (q1 - p1);
    double f2 = (p2 - p1) % (p1 - q2);
    double f = (f1 + f2);

    if(fabs(f) < EPS) return pdd(nan(""), nan(""));

    return (f2 / f) * q1 + (f1 / f) * q2;
}

deque<Line> dq;

void halfLineInter(){
    int n = lnlst.size();
    vector<int> stlst;
    for(int i=0; i<n; i++){
        stlst._PB(i);
        pdd d = lnlst[i].second - lnlst[i].first;
        atn[i] = atan2(d._y, d._x);
    }
    sort(stlst.begin(), stlst.end(), lncmp);
    vector<Line> lst;

    for(int i=0; i<n; i++){
        if(i) {
            int j = i-1;
            Line li = lnlst[stlst[i]];
            Line lj = lnlst[stlst[j]];
            pdd di = li.second - li.first;
            pdd dj = lj.second - lj.first;
            if(fabs(di%dj) < EPS){
                if(di % (lj.second - li.second) < 0) {
                    lst.pop_back();
                }else continue;
            }
        }
        lst._PB(lnlst[stlst[i]]);
    }

    dq._PB(lst[0]);
    dq._PB(lst[1]);
    for(int i=2; i<n; i++){
        int dsz = dq.size();

```

```

Line l = lst[i];
while(dsz >= 2){
    Line l1 = dq[dsz-1];
    Line l2 = dq[dsz-2];

    pdd it12 = interPnt(l1.first, l1.second, l2.first,
        , l2.second);

    if((l1.second - l1.first) % (it12 - l1.first) < 0){
        dq.pop_back();
        dsz --;
    } else break;
}

while(dsz >= 2){
    Line l1 = dq[0];
    Line l2 = dq[1];

    pdd it12 = interPnt(l1.first, l1.second, l2.first,
        , l2.second);

    if((l1.second - l1.first) % (it12 - l1.first) < 0){
        dq.pop_front();
        dsz --;
    } else break;
}

Line l1 = dq[dsz - 1];
if(!std::isnan(interPnt(l1.first, l1.second, l1.first,
    , l1.second)._x)){
    dq._PB(l);
}

}

int dsz = dq.size();
while(dsz >= 2){
    Line l1 = dq[dsz - 1];
    Line l2 = dq[dsz - 2];
    Line l = dq[0];
    pdd it12 = interPnt(l1.first, l1.second, l2.first,
        , l2.second);
    if((l1.second - l1.first) % (it12 - l1.first) < 0){
        dq.pop_back();
        dsz --;
    } else break;
}
}

int main(){

    int N;
    cin >> N;
    for(int i=0; i<N; i++){
        double x1, x2, y1, y2;
        cin >> x1 >> y1 >> x2 >> y2;
        lnlst._PB(_MP(pdd(x1, y1), pdd(x2, y2)));
    }

    halfLineInter();

    int dsz = dq.size();
    for(int i=0; i<dsz; i++){
        int j = (i+1) % dsz;
        pdd it = interPnt(dq[i].first, dq[i].second, dq[j].
            first, dq[j].second);
        cout << it._x << ' ' << it._y << endl;
    }
}

```

## 7 Stringology

### 7.1 Suffix Array

```

const int MAX = 1020304;
int ct[MAX], he[MAX], rk[MAX], sa[MAX], tsa[MAX], tp[
    MAX][2];

```

```

void suffix_array(char *ip){

    int len = strlen(ip);
    int alp = 256;

    memset(ct, 0, sizeof(ct));
    for(int i=0; i<len; i++) ct[ip[i]+1]++;
    for(int i=1; i<alp; i++) ct[i]+=ct[i-1];
    for(int i=0; i<len; i++) rk[i]=ct[ip[i]];

    for(int i=1; i<len; i*=2){
        for(int j=0; j<len; j++){
            if(j+i>len) tp[j][1]=0;
            else tp[j][1]=rk[j+i]+1;

            tp[j][0]=rk[j];
        }
        memset(ct, 0, sizeof(ct));
        for(int j=0; j<len; j++) ct[tp[j][1]+1]++;
        for(int j=1; j<len+2; j++) ct[j]+=ct[j-1];
        for(int j=0; j<len; j++) tsa[ct[tp[j][1]]+1]=j;

        memset(ct, 0, sizeof(ct));
        for(int j=0; j<len; j++) ct[tp[j][0]+1]++;
        for(int j=1; j<len+1; j++) ct[j]+=ct[j-1];
        for(int j=0; j<len; j++) sa[ct[tp[j][0]]+1]=tsa[
            j];

        rk[sa[0]]=0;
        for(int j=1; j<len; j++){
            if( tp[sa[j]][0] == tp[sa[j-1]][0] &&
                tp[sa[j]][1] == tp[sa[j-1]][1] )
                rk[sa[j]] = rk[sa[j-1]];
            else
                rk[sa[j]] = j;
        }
    }

    for(int i=0, h=0; i<len; i++){
        if(rk[i]==0) h=0;
        else{
            int j=sa[rk[i]-1];
            h=max(0, h-1);
            for(; ip[i+h]==ip[j+h]; h++);
        }
        he[rk[i]]=h;
    }
}

```

### 7.2 Suffix Array (SAIS TWT514)

```

struct SA{
    static const int MXN = 300010;
    bool _t[MXN*2];
    int _s[MXN*2], _sa[MXN*2], _c[MXN*2], x[MXN], _p[
        MXN], _q[MXN*2], hei[MXN], r[MXN];
    int operator [] (int i){ return _sa[i]; }
    void build(int *s, int n, int m){
        memcpy(_s, s, sizeof(int) * n);
        sais(_s, _sa, _p, _q, _t, _c, n, m);
        mkhei(n);
    }
    void mkhei(int n){
        REP(i, n) r[_sa[i]] = i;
        hei[0] = 0;
        REP(i, n) if(r[i]) {
            int ans = i>0 ? max(hei[r[i-1]] - 1, 0) :
                0;
            while(_s[i+ans] == _s[_sa[r[i]-1]+ans]) ans
                ++;
            hei[r[i]] = ans;
        }
    }
    void sais(int *s, int *sa, int *p, int *q, bool *t,
        int *c, int n, int z){
        bool uniq = t[n-1] = true, neq;
        int nn = 0, nmzx = -1, *nsa = sa + n, *ns = s +
            n, lst = -1;
#define MS0(x,n) memset((x),0,n*sizeof(*(x)))

```

```

#define MAGIC(XD) MS0(sa, n); \
    memcpy(x, c, sizeof(int) * z); \
    XD; \
    memcpy(x + 1, c, sizeof(int) * (z - 1)); \
    REP(i,n) if(sa[i] && !t[sa[i]-1]) sa[x[s[sa[i]-1]]++] = sa[i]-1; \
    memcpy(x, c, sizeof(int) * z); \
    for(int i = n - 1; i >= 0; i--) if(sa[i] && t[sa[i]-1]) sa[--x[s[sa[i]-1]]] = sa[i]-1;
MS0(c, z);
REP(i,n) uniq &= ++c[s[i]] < 2;
REP(i,z-1) c[i+1] += c[i];
if (uniq) { REP(i,n) sa[--c[s[i]]] = i; return;
}
for(int i = n - 2; i >= 0; i--) t[i] = (s[i]==s[i+1] ? t[i+1] : s[i]<s[i+1]);
MAGIC(REP1(i,1,n-1) if(t[i] && !t[i-1]) sa[--x[s[i]]]=p[q[i]=nn++]=i);
REP(i, n) if (sa[i] && t[sa[i]] && !t[sa[i]-1])
{
    neq=lst<0||memcmp(s+sa[i],s+lst,(p[q[sa[i]]+1]-sa[i])*sizeof(int));
    ns[q[lst=sa[i]]]=nmxz+=neq;
}
sais(ns, nsa, p + nn, q + n, t + n, c + z, nn, nmxz + 1);
MAGIC(for(int i = nn - 1; i >= 0; i--) sa[--x[s[p[nsa[i]]]]] = p[nsa[i]]);
}
};

int main(){
    // s is int array
    SA *sa = new SA();
    sa->build(s,n,128);
}

```

### 7.3 Aho-Corasick Algorithm

```

class ACautomata{
public:
    class Node{
    public:
        int cnt,dp;
        Node *go[26], *fail;
        Node (){
            cnt = 0;
            dp = -1;
            memset(go,0,sizeof(go));
            fail = 0;
        }
    };

    Node *root, pool[1048576];
    int nMem;

    Node* new_Node(){
        pool[nMem] = Node();
        return &pool[nMem++];
    }

    void init(){
        nMem = 0;
        root = new_Node();
    }

    void add(const string &str){
        insert(root,str,0);
    }

    void insert(Node *cur, const string &str, int pos){
        if (pos >= (int)str.size()){
            cur->cnt++;
            return;
        }
        int c = str[pos]-'a';
        if (cur->go[c] == 0){
            cur->go[c] = new_Node();
        }
        insert(cur->go[c],str,pos+1);
    }

    void make_fail(){

```

```

queue<Node*> que;
que.push(root);
while (!que.empty()){
    Node* fr=que.front();
    que.pop();
    for (int i=0; i<26; i++){
        if (fr->go[i]){
            Node *ptr = fr->fail;
            while (ptr && !ptr->go[i])
                ptr = ptr->fail;
            if (!ptr)
                fr->go[i]->fail = root;
            else
                fr->go[i]->fail = ptr->go[i];
            que.push(fr->go[i]);
        }
    }
}
};

```

### 7.4 Z value

```

char s[MAXLEN];
int len,z[MAXLEN];
void Z_value() {
    int i,j,left,right;
    left=right=0; z[0]=len;
    for(i=1;i<len;i++) {
        j=max(min(z[i-left],right-i),0);
        for(;i+j<len&&s[i+j]==s[j];j++);
        z[i]=j;
        if(i+z[i]>right) {
            right=i+z[i];
            left=i;
        }
    }
}

```

### 7.5 Z value (palindrome ver.)

```

const int MAX = 1000;
int len;
char ip[MAX];
char op[MAX*2];
int zv[MAX*2];

int main(){
    cin >> ip;
    len = strlen(ip);

    int l2 = len*2 - 1;
    for(int i=0; i<l2; i++){
        if(i&1) op[i] = '@';
        else op[i] = ip[i/2];
    }
    int l=0, r=0;
    zv[0] = 1;

    for(int i=1; i<l2; i++){
        if (i > r){
            l = r = i;
            while( l>0 && r<l2-1 && op[l-1] == op[r+1] ){
                l--;
                r++;
            }
            zv[i] = (r-l+1);
        }else{
            int md = (l+r)/2;
            int j = md + md - i;
            zv[i] = zv[j];
            int q = zv[i] / 2;
            int nr = i + q;
            if( nr == r ){
                l = i + i - r;
            }
            while( l>0 && r<l2-1 && op[l-1] == op[r+1] ){

```

```

        l --;
        r ++;
    }
    zv[i] = r - l + 1;
} else if (nr > r){
    zv[i] = (r - i) * 2 + 1;
}
}
}
return 0;
}

```

## 7.6 Suffix Automaton

```

class SAM{ //SuffixAutomaton
public:
    class State{
    public:
        State *par, *go[26];
        int val;
        State (int _val) :
            par(0), val(_val){
            MSET(go,0);
        }
    };
    State *root, *tail;

    void init(const string &str){
        root = tail = new State(0);
        for (int i=0; i<SZ(str); i++){
            extend(str[i]-'a');
        }
    }
    void extend(int w){
        State *p = tail, *np = new State(p->val+1);
        for ( ; p && p->go[w]==0; p=p->par)
            p->go[w] = np;
        if (p == 0){
            np->par = root;
        } else {
            if (p->go[w]->val == p->val+1){
                np->par = p->go[w];
            } else {
                State *q = p->go[w], *r = new State(0);
                *r = *q;
                r->val = p->val+1;
                q->par = np->par = r;
                for ( ; p && p->go[w]==q; p=p->par)
                    p->go[w] = r;
            }
        }
        tail = np;
    }
};

```

## 8 Problems

### 8.1 Qtree IV

```

const int MX = 100005;
const int INF = 1029384756;

int N, fa[MX], faW[MX], sz[MX], belong[MX], color[MX], at[MX];
int fr, bk, que[MX];
vector<PII> E[MX];
multiset<int> D[MX];
multiset<int> ans;

struct Chain{
    int n;
    vector<int> V;
    struct Node{
        int mxL, mxR, mx;
    };
    Node *tree;
};

```

```

int *d;

void init(){
    n = V.size();
    for (int i=0; i<n; i++){
        at[V[i]] = i;
        d = new int[n];
        for (int i=1; i<n; i++){
            d[i] = d[i-1] + faW[V[i-1]];
            tree = new Node[4*n];
        }
    }
    int max3(int a, int b, int c){
        return max(a, max(b, c));
    }
    void pushUp(int L, int R, int id){
        int M = (L+R)/2;
        int lc = id*2+1;
        int rc = id*2+2;
        tree[id].mxL = max3(-INF, tree[lc].mxL, d[M+1]-d[L]
            ]+tree[rc].mxL);
        tree[id].mxR = max3(-INF, tree[rc].mxR, d[R]-d[M]+
            tree[lc].mxR);
        tree[id].mx = max3(tree[lc].mx, tree[rc].mx, tree[
            lc].mxR + d[M+1]-d[M] + tree[rc].mxL);
    }
    void build_tree(int L, int R, int id){
        if (L == R){
            multiset<int>::reverse_iterator ptr=D[V[L]].
                rbegin();
            tree[id].mxL = tree[id].mxR = tree[id].mx = *ptr;
            ptr++;
            tree[id].mx = max(-INF, tree[id].mx+(*ptr));
            return ;
        }
        int M = (L+R)/2;
        build_tree(L, M, id*2+1);
        build_tree(M+1, R, id*2+2);
        pushUp(L, R, id);
    }
    void update_tree(int L, int R, int fn, int id){
        if (L == R){
            multiset<int>::reverse_iterator ptr=D[V[L]].
                rbegin();
            tree[id].mxL = tree[id].mxR = tree[id].mx = *ptr;
            ptr++;
            tree[id].mx = max(-INF, tree[id].mx+(*ptr));
            return ;
        }
        int M=(L+R)/2;
        if (fn <= M) update_tree(L, M, fn, id*2+1);
        else update_tree(M+1, R, fn, id*2+2);
        pushUp(L, R, id);
    }
    int update(int x){
        int u=V.back();
        int p=fa[u];
        if (p) D[p].erase(D[p].find(faW[u]+tree[0].mxR));
        ans.erase(ans.find(tree[0].mx));
        update_tree(0, n-1, at[x], 0);
        ans.insert(tree[0].mx);
        if (p) D[p].insert(faW[u]+tree[0].mxR);
        return p;
    }
}chain[MX];

void DFS(int u){
    Chain &c = chain[belong[u]];
    c.init();
    for (int i=0, v; i<c.n; i++){
        u = c.V[i];
        FOR(it, E[u]){
            v = it->_F;
            if (fa[u] == v || (i && v == c.V[i-1])) continue;
            DFS(v);
            D[u].insert(chain[belong[v]].tree[0].mxR+it->_S);
        }
        D[u].insert(-INF);
        D[u].insert(-INF);
        D[u].insert(0);
    }
    c.build_tree(0, c.n-1, 0);
}

```

```

    ans.insert(c.tree[0].mx);
}
int main(int argc, char** argv){
    scanf("%d", &N);
    for (int i=0,u,v,w; i<N-1; i++){
        scanf("%d%d%d", &u, &v, &w);
        E[u]._PB(_MP(v,w));
        E[v]._PB(_MP(u,w));
    }
    fr=bk=0; que[bk++] = 1;
    while (fr < bk){
        int u=que[fr++],v;
        FOR(it,E[u]){
            v = it->_F;
            if (v == fa[u]) continue;
            que[bk++] = v;
            fa[v] = u;
            faW[v] = it->_S;
        }
    }
    for (int i=bk-1,u,v,pos; i>=0; i--){
        u = que[i];
        sz[u] = 1;
        pos = 0;
        FOR(it,E[u]){
            v = it->_F;
            if (v == fa[u]) continue;
            sz[u] += sz[v];
            if (sz[v] > sz[pos])
                pos=v;
        }
        if (pos == 0) belong[u] = u;
        else belong[u] = belong[pos];
        chain[belong[u]].V._PB(u);
    }
    DFS(1);
    int nq;
    scanf("%d", &nq);
    char cmd[10];
    while (nq--){
        scanf("%s", cmd);
        if (cmd[0] == 'C'){
            int x;
            scanf("%d", &x);
            if (color[x]){
                D[x].insert(0);
            } else {
                D[x].erase(D[x].find(0));
            }
            color[x] ^= 1;
            while (x){
                x = chain[belong[x]].update(x);
            }
        } else {
            if (*ans.rbegin() != -INF){
                printf("%d\n", max(0,*ans.rbegin()));
            } else {
                puts("They have disappeared.");
            }
        }
    }
    return 0;
}

```

## 8.2 Find the maximun tangent (x,y is increasing)

```

typedef long long LL;
const int MAXN = 100010;
struct Coord{
    LL x, y;
    Coord operator - (Coord ag) const{
        Coord res;
        res.x = x - ag.x;
        res.y = y - ag.y;
        return res;
    }
}sum[MAXN], pnt[MAXN], ans, calc;

```

```

inline bool cross(Coord a, Coord b, Coord c){
    return (c.y - a.y) * (c.x - b.x) > (c.x - a.x) * (c.y - b.y);
}
int main(){
    int n, l, np, st, ed, now;
    scanf("%d %d\n", &n, &l);
    sum[0].x = sum[0].y = np = st = ed = 0;
    for (int i = 1, v; i <= n; i++){
        scanf("%d", &v);
        sum[i].y = sum[i-1].y + v;
        sum[i].x = i;
    }
    ans.x = now = 1;
    ans.y = -1;
    for (int i = 0; i <= n-1; i++){
        while (np > 1 && cross(pnt[np-2], pnt[np-1], sum[i]))
            np--;
        if (np < now && np != 0) now = np;
        pnt[np++] = sum[i];
        while (now < np && !cross(pnt[now-1], pnt[now], sum[i+1]))
            now++;
        calc = sum[i+1] - pnt[now-1];
        if (ans.y * calc.x < ans.x * calc.y){
            ans = calc;
            st = pnt[now-1].x;
            ed = i+1;
        }
    }
    double res = (sum[ed].y-sum[st].y)/(sum[ed].x-sum[st].x);
    printf("%f\n", res);
    return 0;
}

```

## 9 +1ironwood's code

### 9.1 KDTreeAndNearestPoint

```

#define INF 1100000000
class NODE{ public:
    int x,y,x1,x2,y1,y2;
    int i,f;
    NODE *L,*R;
};
inline long long dis(NODE& a,NODE& b){
    long long dx=a.x-b.x;
    long long dy=a.y-b.y;
    return dx*dx+dy*dy;
}
NODE node[100000];
bool cmpx(const NODE& a,const NODE& b){ return a.x<b.x; }
bool cmpy(const NODE& a,const NODE& b){ return a.y<b.y; }
NODE* KDTree(int L,int R,int dep){
    if(L>R) return 0;
    int M=(L+R)/2;
    if(dep%2==0){
        nth_element(node+L,node+M,node+R+1,cmpx);
        node[M].f=0;
    }else{
        nth_element(node+L,node+M,node+R+1,cmpy);
        node[M].f=1;
    }
    node[M].x1=node[M].x2=node[M].x;
    node[M].y1=node[M].y2=node[M].y;
    node[M].L=KDTree(L,M-1,dep+1);
    if(node[M].L){
        node[M].x1=min(node[M].x1,node[M].L->x1);
        node[M].x2=max(node[M].x2,node[M].L->x2);
        node[M].y1=min(node[M].y1,node[M].L->y1);
        node[M].y2=max(node[M].y2,node[M].L->y2);
    }
    node[M].R=KDTree(M+1,R,dep+1);
}

```

```

if(node[M].R){
    node[M].x1=min(node[M].x1,node[M].R->x1);
    node[M].x2=max(node[M].x2,node[M].R->x2);
    node[M].y1=min(node[M].y1,node[M].R->y1);
    node[M].y2=max(node[M].y2,node[M].R->y2);
}
return node+M;
}
inline int touch(NODE* r,int x,int y,long long d){
    long long d2;
    d2 = (long long)(sqrt(d)+1);
    if(x<r->x1-d2 || x>r->x2+d2 || y<r->y1-d2 || y>r->y2+
        d2)
        return 0;
    return 1;
}
void nearest(NODE* r,int z,long long &md){
    if(!r || !touch(r,node[z].x,node[z].y,md)) return;
    long long d;
    if(node[z].i!=r->i){
        d=dis(*r,node[z]);
        if(d<md) md=d;
    }
    if(r->f==0){
        if(node[z].x<r->x){
            nearest(r->L,z,md);
            nearest(r->R,z,md);
        }else{
            nearest(r->R,z,md);
            nearest(r->L,z,md);
        }
    }else{
        if(node[z].y<r->y){
            nearest(r->L,z,md);
            nearest(r->R,z,md);
        }else{
            nearest(r->R,z,md);
            nearest(r->L,z,md);
        }
    }
}
int main(){
    int TT,n,i;
    long long d;
    NODE* root;
    scanf("%d",&TT);
    while(TT--){
        scanf("%d",&n);
        for(i=0;i<n;i++){
            scanf("%d %d",&node[i].x,&node[i].y);
            node[i].i=i;
        }
        root=KDTree(0,n-1,0);
        for(i=0;i<n;i++){
            d=9000000000000000LL;
            nearest(root,i,d);
            ans[node[i].i]=d;
        }
    }
}

```

## 9.2 MinkowskiSum

```

/* convex hull Minkowski Sum*/
#define INF 1000000000000000LL
class PT{ public:
    long long x,y;
    int POS(){
        if(y==0) return x>0?0:1;
        return y>0?0:1;
    }
};
PT pt[300000],qt[300000],rt[300000];
long long Lx,Rx;
int dn,un;
inline bool cmp(PT a,PT b){
    int pa=a.POS(),pb=b.POS();
    if(pa==pb) return (a^b)>0;
    return pa<pb;
}

```

```

int minkowskiSum(int n,int m){
    int i,j,r,p,q,fi,fj;
    for(i=1,p=0;i<n;i++){
        if(pt[i].y<pt[p].y || (pt[i].y==pt[p].y && pt[i].x<
            pt[p].x)) p=i; }
    for(i=1,q=0;i<m;i++){
        if(qt[i].y<qt[q].y || (qt[i].y==qt[q].y && qt[i].x<
            qt[q].x)) q=i; }
    rt[0]=pt[p]+qt[q];
    r=1; i=p; j=q; fi=fj=0;
    while(1){
        if((fj&&j==q) || ((!fi||i!=p) && cmp(pt[(p+1)%n]-pt
            [p],qt[(q+1)%m]-qt[q]))){
            rt[r]=rt[r-1]+pt[(p+1)%n]-pt[p];
            p=(p+1)%n;
            fi=1;
        }else{
            rt[r]=rt[r-1]+qt[(q+1)%m]-qt[q];
            q=(q+1)%m;
            fj=1;
        }
        if(r<=1 || ((rt[r]-rt[r-1])^(rt[r-1]-rt[r-2]))!=0)
            r++;
        else rt[r-1]=rt[r];
        if(i==p && j==q) break;
    }
    return r-1;
}
void initInConvex(int n){
    int i,p,q;
    long long Ly,Ry;
    Lx=INF; Rx=-INF;
    for(i=0;i<n;i++){
        if(pt[i].x<Lx) Lx=pt[i].x;
        if(pt[i].x>Rx) Rx=pt[i].x;
    }
    Ly=Ry=INF;
    for(i=0;i<n;i++){
        if(pt[i].x==Lx && pt[i].y<Ly){ Ly=pt[i].y; p=i; }
        if(pt[i].x==Rx && pt[i].y>Ry){ Ry=pt[i].y; q=i; }
    }
    for(dn=0,i=p;i!=q;i=(i+1)%n){ qt[dn++]=pt[i]; }
    qt[dn]=pt[q]; Ly=Ry=-INF;
    for(i=0;i<n;i++){
        if(pt[i].x==Lx && pt[i].y>Ly){ Ly=pt[i].y; p=i; }
        if(pt[i].x==Rx && pt[i].y<Ry){ Ry=pt[i].y; q=i; }
    }
    for(un=0,i=p;i!=q;i=(i+n-1)%n){ rt[un++]=pt[i]; }
    rt[un]=pt[q];
}
inline int inConvex(PT p){
    int L,R,M;
    if(p.x<Lx || p.x>Rx) return 0;
    L=0;R=dn;
    while(L<R-1){ M=(L+R)/2;
        if(p.x<qt[M].x) R=M; else L=M; }
    if(tri(qt[L],qt[R],p)<0) return 0;
    L=0;R=un;
    while(L<R-1){ M=(L+R)/2;
        if(p.x<rt[M].x) R=M; else L=M; }
    if(tri(rt[L],rt[R],p)>0) return 0;
    return 1;
}
int main(){
    int n,m,i;
    PT p;
    scanf("%d",&n);
    for(i=0;i<n;i++) scanf("%I64d %I64d",&pt[i].x,&pt[i].
        y);
    scanf("%d",&m);
    for(i=0;i<m;i++) scanf("%I64d %I64d",&qt[i].x,&qt[i].
        y);
    n=minkowskiSum(n,m);
    for(i=0;i<n;i++) pt[i]=rt[i];
    scanf("%d",&m);
    for(i=0;i<m;i++) scanf("%I64d %I64d",&qt[i].x,&qt[i].
        y);
    n=minkowskiSum(n,m);
    for(i=0;i<n;i++) pt[i]=rt[i];
    initInConvex(n);
}

```

```
scanf("%d",&m);
for(i=0;i<m;i++){
    scanf("%I64d %I64d",&p.x,&p.y);
    p.x*=3; p.y*=3;
    puts(inConvex(p)?"YES":"NO");
}
}
```

### 9.3 MinimumMeanCycle

```
/* minimum mean cycle */
class Edge { public:
    int v,u;
    double c;
};
int n,m;
Edge e[MAXEDGE];
double d[MAXNUM][MAXNUM];
inline void relax(double &x,double val) { if(val<x) x=
    val; }
inline void bellman_ford() {
    int i,j;
    for(j=0;j<n;j++) d[0][j]=0.0;
    for(i=0;i<n;i++) {
        for(j=0;j<n;j++) d[i+1][j]=inf;
        for(j=0;j<m;j++)
            if(d[i][e[j].v]<inf-eps) relax(d[i+1][e[j].u],d[i][
                e[j].v]+e[j].c);
    }
}
inline double karp_mmc() {
    // returns inf if no cycle, mmc otherwise
    int i,k; double mmc=inf,avg;
    bellman_ford();
    for(i=0;i<n;i++) {
        avg=0.0;
        for(k=0;k<n;k++) {
            if(d[n][i]<inf-eps) avg=max(avg,(d[n][i]-d[k][i])
                /(n-k));
            else avg=max(avg,inf);
        }
        mmc=min(mmc,avg);
    }
    return mmc;
}
```

### 9.4 PolynomialGenerator

```
class PolynomialGenerator {
    /* for a nth-order polynomial f(x), *
    * given f(0), f(1), ..., f(n) *
    * express f(x) as sigma_i{c_i*C(x,i)} */
public:
    int n;
    vector<long long> coef;
    // initialize and calculate f(x), vector _fx should
    // be
    filled with f(0) to f(n)
    PolynomialGenerator(int _n,vector<long long> _fx)
        :n(_n
        ),coef(_fx) {
        for(int i=0;i<n;i++)
            for(int j=n;j>i;j--)
                coef[j]-=coef[j-1];
    }
    // evaluate f(x), runs in O(n)
    long long eval(int x) {
        long long m=1,ret=0;
        for(int i=0;i<=n;i++) {
            ret+=coef[i]*m;
            m=m*(x-i)/(i+1);
        }
        return ret;
    }
};
```

### 9.5 SwGeneralGraphMaxMatching

```
#define N 256 // max vertex num
class Graph { public:
    // n,g[i][j]=0/1, match() => match: (i,mate[i]) (or
    // mate[i]=-1)
    int n, mate[N];
    bool g[N][N], inQ[N], inBlo[N];
    queue<int> Q;
    int start, newBase, prev[N], base[N];
    int lca(int u, int v) {
        bool path[N] = { false };
        while(true) {
            u = base[u]; path[u] = true;
            if(u == start) break;
            u = prev[mate[u]];
        }
        while(true) {
            v = base[v];
            if(path[v]) break;
            v = prev[mate[v]];
        }
        return v;
    }
    void trace(int u) {
        while(base[u] != newBase) {
            int v = mate[u];
            inBlo[base[u]] = inBlo[base[v]] = true;
            u = prev[v];
            if(base[u] != newBase) prev[u] = v;
        }
    }
    void contract(int u, int v) {
        newBase = lca(u, v);
        memset(inBlo, false, sizeof(inBlo));
        trace(u); trace(v);
        if(base[u] != newBase) prev[u] = v;
        if(base[v] != newBase) prev[v] = u;
        for(int i = 0; i < n; i++)
            if(inBlo[base[i]]) {
                base[i] = newBase;
                if(!inQ[i]) { Q.push(i); inQ[i] = true; }
            }
    }
    bool search() {
        memset(inQ, false, sizeof(inQ));
        memset(prev, -1, sizeof(prev));
        for(int i = 0; i < n; i++) base[i] = i;
        while(!Q.empty()) Q.pop();
        Q.push(start); inQ[start] = true;
        while(!Q.empty()) {
            int u = Q.front(); Q.pop();
            for(int i = 0; i < n; i++)
                if(g[u][i] && base[u] != base[i] && mate[u] !=
                    i){
                    if(i == start || (mate[i] >= 0 && prev[mate[i]
                        ] >= 0)) contract(u, i);
                    else if(prev[i] < 0) {
                        prev[i] = u;
                        if(mate[i] != -1) { Q.push(mate[i]); inQ[
                            mate[i]] = true; }
                        else { augment(i); return true; }
                    }
                }
        }
        return false;
    }
    void augment(int u) {
        while(u >= 0) {
            int v = prev[u], w = mate[v];
            mate[v] = u; mate[u] = v; u = w;
        }
    }
    int match() {
        memset(mate, -1, sizeof(mate));
        int mth = 0;
        for(int i = 0; i < n; i++) {
            if(mate[i] >= 0) continue;
            start = i;
            if(search()) mth++;
        }
    }
};
```



```

    return mth;
}
};

```

## 9.6 stoer-wagner-nm

```

// {{{ StoerWagner
const int inf=1000000000;
// should be larger than max.possible mincut
class StoerWagner {
public:
    int n,mc; // node id in [0,n-1]
    vector<int> adj[MAXN];
    int cost[MAXN][MAXN];
    int cs[MAXN];
    bool merged[MAXN],sel[MAXN];
    // --8<-- include only if cut is explicitly needed
    DisjointSet djs;
    vector<int> cut;
    //--8<-----
    StoerWagner(int _n):n(_n),mc(inf),djs(_n) {
        for(int i=0;i<n;i++)
            merged[i]=0;
        for(int i=0;i<n;i++)
            for(int j=0;j<n;j++)
                cost[i][j]=cost[j][i]=0;
    }
    void append(int v,int u,int c) {
        if(v==u) return;
        if(!cost[v][u]&&c) {
            adj[v].PB(u);
            adj[u].PB(v);
        }
        cost[v][u]+=c;
        cost[u][v]+=c;
    }
    void merge(int v,int u) {
        merged[u]=1;
        for(int i=0;i<n;i++)
            append(v,i,cost[u][i]);
        // --8<-- include only if cut is explicitly
        // needed
        djs.merge(v,u);
        //
        //--8<-----
    }
    void phase() {
        priority_queue<pii> pq;
        for(int v=0;v<n;v++) {
            if(merged[v]) continue;
            cs[v]=0;
            sel[v]=0;
            pq.push(MP(0,v));
        }
        int v,s,pv;
        while(pq.size()) {
            if(cs[pq.top().S]>pq.top().F) {
                pq.pop();
                continue;
            }
            pv=v;
            v=pq.top().S;
            s=pq.top().F;
            pq.pop();
            sel[v]=1;
            for(int i=0;i<adj[v].size();i++) {
                int u=adj[v][i];
                if(merged[u]||sel[u]) continue;
                cs[u]+=cost[v][u];
                pq.push(MP(cs[u],u));
            }
        }
        if(s<mc) {
            mc=s;
            // --8<-- include only if cut is explicitly
            // needed -----
            cut.clear();
            for(int i=0;i<n;i++)
                if(djs.getrep(i)!=djs.getrep(v)) cut.PB(i);
        }
    }
};

```

```

        //--8<-----
    }
    merge(v,pv);
}
int mincut() {
    if(mc==inf) {
        for(int t=0;t<n-1;t++)
            phase();
    }
    return mc;
}
// --8<-- include only if cut is explicitly needed
// -----
vector<int> getcut() { // return one side of the
    // cut
    mincut();
    return cut;
}
// --8<-----
};
// }}}

```