

Contents

1 Basic	1
1.1 .vimrc	1
1.2 Increase Stack Size	1
2 flow	1
2.1 DMST	1
2.2 ISAP	2
2.3 MinCostFlow	2
2.4 SW min-cut	3
2.5 HLPPA	3
2.6 Hungarian	4
2.7 Hungarian Unbalanced	4
2.8 Gusfield	5
2.9 Max flow with lower/upper bound	5
2.10 Relabel to Front	5
2.11 Flow Method	6
3 Math	6
3.1 FFT	6
3.2 NTT	6
3.3 Fast Walsh Transform	7
3.4 BigInt	7
3.5 Linear Recurrence	8
3.6 Miller Rabin	9
3.7 Simplex	9
3.8 Faulhaber	10
3.9 Chinese Remainder	10
3.10 Pollard Rho	10
3.11 Poly Generator	11
3.12 Matrix Pseudo Inverse	11
3.13 ax+by=gcd	11
3.14 Mod	11
3.15 Primes and μ function	11
3.16 Result	12
4 Geometry	12
4.1 halfPlaneIntersection	12
4.2 Intersection of 2 lines	12
4.3 Intersection of 2 circles	12
4.4 Intersection of 2 segments	12
4.5 KD Tree	13
4.6 Poly Union	13
4.7 Lower Concave Hull	14
4.8 Min Enclosing Circle	14
4.9 Minkowski sum	14
4.10 Min/Max Enclosing Rectangle	15
5 Graph	16
5.1 HeavyLightDecomp	16
5.2 DominatorTree	17
5.3 MaxClique	17
5.4 Strongly Connected Component	18
5.5 Minimum General Weighted Matching	18
6 String	19
6.1 PalTree	19
6.2 SuffixArray	19
6.3 SAIS	19
6.4 SuffixAutomata	20
6.5 Aho-Corasick	20
6.6 Z Value	20
6.7 ZValue Palindrome	21
6.8 Smallest Rotation	21
6.9 Baker Bird	21
6.10 Cyclic LCS	22
7 Data Structure	22
7.1 Treap	22
7.2 Link-Cut Tree	23
7.3 Disjoint Set	23
7.4 Pairing Heap	24
7.5 Leftist Heap	24
7.6 Black Magic	24
8 Others	24
8.1 Find max tangent(x,y is increasing)	24
8.2 Exact Cover Set	24

1 Basic

1.1 .vimrc

```
syn on
se ai nu ru cul mouse=a
se cin et ts=2 sw=2 sts=2
so $VIMRUNTIME/mswin.vim
colo desert
se gfn=Monospace\ 14
```

1.2 Increase Stack Size

```
//stack resize
asm( "mov %0,%esp\n" :: "g"(mem+10000000) );
//change esp to rsp if 64-bit system

//stack resize (linux)
#include <sys/resource.h>
void increase_stack_size() {
    const rlim_t ks = 64*1024*1024;
    struct rlimit rl;
    int res=getrlimit(RLIMIT_STACK, &rl);
    if(res==0){
        if(rl.rlim_cur<ks){
            rl.rlim_cur=ks;
            res=setrlimit(RLIMIT_STACK, &rl);
        }
    }
}
```

2 flow

2.1 DMST

```
/*
 * Edmond's algorithm for Directed MST
 * runs in O(VE)
 */
const int MAXV = 10010;
const int MAXE = 10010;
const int INF = 2147483647;
struct Edge{
    int u, v, c;
    Edge(){}
    Edge(int x, int y, int z) :
        u(x), v(y), c(z){}
};
int V, E, root;
Edge edges[MAXE];
inline int newV(){
    V++;
    return V;
}
inline void addEdge(int u, int v, int c){
    E++;
    edges[E] = Edge(u, v, c);
}
bool con[MAXV];
int mnInW[MAXV], prv[MAXV], cyc[MAXV], vis[MAXV];
inline int DMST(){
    fill(con, con+V+1, 0);
    int r1 = 0, r2 = 0;
    while(1){
        fill(mnInW, mnInW+V+1, INF);
        fill(prv, prv+V+1, -1);
        REP(i, 1, E){
            int u=edges[i].u, v=edges[i].v, c=edges[i].c;
            if(u != v && v != root && c < mnInW[v])
                mnInW[v] = c, prv[v] = u;
        }
        fill(vis, vis+V+1, -1);
        fill(cyc, cyc+V+1, -1);
        r1 = 0;
        bool jf = 0;
```

```

REP(i, 1, V){
    if(con[i]) continue;
    if(prv[i] == -1 && i != root) return -1;
    if(prv[i] > 0) r1 += mnInW[i];
    int s;
    for(s = i; s != -1 && vis[s] == -1; s = prv[s])
        vis[s] = i;
    if(s > 0 && vis[s] == i){
        // get a cycle
        jf = 1;
        int v = s;
        do{
            cyc[v] = s, con[v] = 1;
            r2 += mnInW[v];
            v = prv[v];
        }while(v != s);
        con[s] = 0;
    }
}
if(!jf) break;
REP(i, 1, E){
    int &u = edges[i].u;
    int &v = edges[i].v;
    if(cyc[v] > 0) edges[i].c -= mnInW[edges[i].v];
    if(cyc[u] > 0) edges[i].u = cyc[edges[i].u];
    if(cyc[v] > 0) edges[i].v = cyc[edges[i].v];
    if(u == v) edges[i--] = edges[E--];
}
}
return r1+r2;
}

```

2.2 ISAP

```

#define SZ(c) ((int)(c).size())
struct Maxflow {
    static const int MAXV = 20010;
    static const int INF = 1000000;
    struct Edge {
        int v, c, r;
        Edge(int _v, int _c, int _r):
            v(_v), c(_c), r(_r) {}
    };
    int s, t;
    vector<Edge> G[MAXV*2];
    int iter[MAXV*2], d[MAXV*2], gap[MAXV*2], tot;
    void flowinit(int x) {
        tot = x+2;
        s = x+1, t = x+2;
        for(int i = 0; i <= tot; i++) {
            G[i].clear();
            iter[i] = d[i] = gap[i] = 0;
        }
    }
    void addEdge(int u, int v, int c) {
        G[u].push_back(Edge(v, c, SZ(G[v])));
        G[v].push_back(Edge(u, 0, SZ(G[u]) - 1));
    }
    int dfs(int p, int flow) {
        if(p == t) return flow;
        for(int &i = iter[p]; i < SZ(G[p]); i++) {
            Edge &e = G[p][i];
            if(e.c > 0 && d[p] == d[e.v]+1) {
                int f = dfs(e.v, min(flow, e.c));
                if(f) {
                    e.c -= f;
                    G[e.v][e.r].c += f;
                    return f;
                }
            }
        }
        if(--gap[d[p]] == 0) d[s] = tot;
        else {
            d[p]++;
            iter[p] = 0;
            ++gap[d[p]];
        }
        return 0;
    }
    int maxflow() {

```

```

//puts("MF");
int res = 0;
gap[0] = tot;
for(res = 0; d[s] < tot; res += dfs(s, INF));
return res;
}
} flow;
Maxflow::Edge e(1, 1, 1);

```

2.3 MinCostFlow

```

/*
A template for Min Cost Max Flow
tested with TIOJ 1724
*/
struct MinCostMaxFlow{
    static const int MAXV = 20010;
    static const int INF = 1000000000;
    struct Edge{
        int v, cap, w, rev;
        Edge(){}
        Edge(int t2, int t3, int t4, int t5)
            : v(t2), cap(t3), w(t4), rev(t5) {}
    };
    int V, s, t;
    vector<Edge> g[MAXV];
    void init(int n){
        V = n+2;
        s = n+1, t = n+2;
        for(int i = 1; i <= V; i++) g[i].clear();
    }
    void addEdge(int a, int b, int cap, int w){
        g[a].push_back(Edge(b, cap, w, (int)g[b].size()));
        g[b].push_back(Edge(a, 0, -w, (int)g[a].size()-1));
    }
    int d[MAXV], id[MAXV], mom[MAXV];
    bool inqu[MAXV];
    int qu[2000000], ql, qr;
    //the size of qu should be much large than MAXV
    int mncmxf(){
        int mxf = 0, mnc = 0;
        while(1){
            fill(d+1, d+1+V, INF);
            fill(inqu+1, inqu+1+V, 0);
            fill(mom+1, mom+1+V, -1);
            mom[s] = s;
            d[s] = 0;
            ql = 1, qr = 0;
            qu[++qr] = s;
            inqu[s] = 1;
            while(ql <= qr){
                int u = qu[ql++];
                inqu[u] = 0;
                for(int i = 0; i < (int) g[u].size(); i++){
                    Edge &e = g[u][i];
                    int v = e.v;
                    if(e.cap > 0 && d[v] > d[u]+e.w){
                        d[v] = d[u]+e.w;
                        mom[v] = u;
                        id[v] = i;
                        if(!inqu[v]) qu[++qr] = v, inqu[v] = 1;
                    }
                }
            }
            if(mom[t] == -1) break;
            int df = INF;
            for(int u = t; u != s; u = mom[u])
                df = min(df, g[mom[u]][id[u]].cap);
            for(int u = t; u != s; u = mom[u]){
                Edge &e = g[mom[u]][id[u]];
                e.cap -= df;
                g[e.v][e.rev].cap += df;
            }
            mxf += df;
            mnc += df*d[t];
        }
        return mnc;
    }
} flow;

```

2.4 SW min-cut

```

struct SW{ // O(V^3)
    static const int MXN = 514;
    int n,vst[MXN],del[MXN];
    int edge[MXN][MXN],wei[MXN];
    void init(int _n){
        n = _n;
        FZ(edge);
        FZ(del);
    }
    void add_edge(int u, int v, int w){
        edge[u][v] += w;
        edge[v][u] += w;
    }
    void search(int &s, int &t){
        FZ(vst); FZ(wei);
        s = t = -1;
        while (true){
            int mx=-1, cur=0;
            for (int i=0; i<n; i++){
                if (!del[i] && !vst[i] && mx<wei[i])
                    cur = i, mx = wei[i];
            }
            if (mx == -1) break;
            vst[cur] = 1;
            s = t;
            t = cur;
            for (int i=0; i<n; i++){
                if (!vst[i] && !del[i]) wei[i] += edge[cur][i];
            }
        }
    }
    int solve(){
        int res = 2147483647;
        for (int i=0,x,y; i<n-1; i++){
            search(x,y);
            res = min(res,wei[y]);
            del[y] = 1;
            for (int j=0; j<n; j++){
                edge[x][j] = (edge[j][x] += edge[y][j]);
            }
        }
        return res;
    }
}graph;

```

2.5 HLPPA

```

/* Highest-Label Preflow Push Algorithm */
// tested with sgu-212 (more testing suggested)
int n,m,src,sink;
int deg[MXN],adj[MXN][MXN],res[MXN][MXN]; //
    residual capacity
// graph (i.e. all things above) should be constructed
    beforehand
int ef[MXN],ht[MXN]; // excess flow, height
int apt[MXN]; // the next adj index to try push
int htodo; // highest label to check with
int hcnt[MXN*2]; // number of nodes with height h
queue<int> ovque[MXN*2]; // used to implement highest-
    label selection
bool inque[MXN];
inline void push(int v,int u) {
    int a=min(ef[v],res[v][u]);
    ef[v]-=a; ef[u]+=a;
    res[v][u]-=a; res[u][v]+=a;
    if(!inque[u]) {
        inque[u]=1;
        ovque[ht[u]].push(u);
    }
}
inline void relabel(int v) {
    int i,u,oldh;
    oldh=ht[v]; ht[v]=2*n;
    for(i=0;i<deg[v];i++){
        u=adj[v][i];
        if(res[v][u]) ht[v]=min(ht[u]+1,ht[v]);
    }
    // gap speedup
    hcnt[oldh]--; hcnt[ht[v]]++;
    if(0<oldh&&oldh<n&&hcnt[oldh]==0) {

```

```

        for(i=0;i<n;i++){
            if(ht[i]>oldh&&ht[i]<n) {
                hcnt[ht[i]]--;
                hcnt[n]++;
                ht[i]=n;
            }
        }
    }
    // update queue
    htodo=ht[v]; ovque[ht[v]].push(v); inque[v]=1;
}
inline void initPreflow() {
    int i,u;
    for(i=0;i<n;i++){
        ht[i]=ef[i]=0;
        apt[i]=0; inque[i]=0;
    }
    ht[src]=n;
    for(i=0;i<deg[src];i++){
        u=adj[src][i];
        ef[u]=res[src][u];
        ef[src]-=ef[u];
        res[u][src]=ef[u];
        res[src][u]=0;
    }
    htodo=n-1;
    for(i=0;i<2*n;i++){
        hcnt[i]=0;
        while(!ovque[i].empty()) ovque[i].pop();
    }
    for(i=0;i<n;i++){
        if(i==src||i==sink) continue;
        if(ef[i]) {
            inque[i]=1;
            ovque[ht[i]].push(i);
        }
        hcnt[ht[i]]++;
    }
    // to ensure src & sink is never added to queue
    inque[src]=inque[sink]=1;
}
inline void discharge(int v) {
    int u;
    while(ef[v]) {
        if(apt[v]==deg[v]) {
            relabel(v);
            apt[v]=0;
            continue;
        }
        u=adj[v][apt[v]];
        if(res[v][u]&&ht[v]==ht[u]+1) push(v,u);
        else apt[v]++;
    }
}
inline void hlppa() {
    int v;
    list<int>::iterator it;
    initPreflow();
    while(htodo>=0) {
        if(!ovque[htodo].size()) {
            htodo--;
            continue;
        }
        v=ovque[htodo].front();
        ovque[htodo].pop();
        inque[v]=0;
        discharge(v);
    }
}

```

2.6 Hungarian

```

#define NIL -1
#define INF 100000000
int n,matched;
int cost[MAXN][MAXN];
bool sets[MAXN]; // whether x is in set S
bool sett[MAXN]; // whether y is in set T
int xlabel[MAXN],ylabel[MAXN];
int xy[MAXN],yx[MAXN]; // matched with whom
int slack[MAXN]; // given y: min{xlabel[x]+ylabel[y]-cost[x][y]} | x not in S
int prev[MAXN]; // for augmenting matching
inline void relabel() {
    int i,delta=INF;
    for(i=0;i<n;i++) if(!sett[i]) delta=min(slack[i],delta);
    for(i=0;i<n;i++) if(sets[i]) xlabel[i]-=delta;
    for(i=0;i<n;i++) {
        if(sett[i]) ylabel[i]+=delta;
        else slack[i]-=delta;
    }
}
inline void add_sets(int x) {
    int i;
    sets[x]=1;
    for(i=0;i<n;i++) {
        if(xlabel[x]+ylabel[i]-cost[x][i]<slack[i]) {
            slack[i]=xlabel[x]+ylabel[i]-cost[x][i];
            prev[i]=x;
        }
    }
}
inline void augment(int final) {
    int x=prev[final],y=final,tmp;
    matched++;
    while(1) {
        tmp=xy[x]; xy[x]=y; yx[y]=x; y=tmp;
        if(y==NIL) return;
        x=prev[y];
    }
}
inline void phase() {
    int i,y,root;
    for(i=0;i<n;i++) { sets[i]=sett[i]=0; slack[i]=INF; }
    for(root=0;root<n&&xy[root]!=NIL;root++);
    add_sets(root);
    while(1) {
        relabel();
        for(y=0;y<n;y++) if(!sett[y]&&slack[y]==0) break;
        if(yx[y]==NIL) { augment(y); return; }
        else { add_sets(yx[y]); sett[y]=1; }
    }
}
inline int hungarian() {
    int i,j,c=0;
    for(i=0;i<n;i++) {
        xy[i]=yx[i]=NIL;
        xlabel[i]=ylabel[i]=0;
        for(j=0;j<n;j++) xlabel[i]=max(cost[i][j],xlabel[i]);
    }
    for(i=0;i<n;i++) phase();
    for(i=0;i<n;i++) c+=cost[i][xy[i]];
    return c;
}

```

2.7 Hungarian Unbalanced

```

const int nil = -1;
const int inf = 1000000000;
int xn,yn,matched;
int cost[MAXN][MAXN];
bool sets[MAXN]; // whether x is in set S
bool sett[MAXN]; // whether y is in set T
int xlabel[MAXN],ylabel[MAXN];
int xy[MAXN],yx[MAXN]; // matched with whom
int slack[MAXN]; // given y: min{xlabel[x]+ylabel[y]-cost[x][y]} | x not in S

```

```

int prev[MAXN]; // for augmenting matching
inline void relabel() {
    int i,delta=inf;
    for(i=0;i<yn;i++) if(!sett[i]) delta=min(slack[i],delta);
    for(i=0;i<xn;i++) if(sets[i]) xlabel[i]-=delta;
    for(i=0;i<yn;i++) {
        if(sett[i]) ylabel[i]+=delta;
        else slack[i]-=delta;
    }
}
inline void add_sets(int x) {
    int i;
    sets[x]=1;
    for(i=0;i<yn;i++) {
        if(xlabel[x]+ylabel[i]-cost[x][i]<slack[i]) {
            slack[i]=xlabel[x]+ylabel[i]-cost[x][i];
            prev[i]=x;
        }
    }
}
inline void augment(int final) {
    int x=prev[final],y=final,tmp;
    matched++;
    while(1) {
        tmp=xy[x]; xy[x]=y; yx[y]=x; y=tmp;
        if(y==nil) return;
        x=prev[y];
    }
}
inline void phase() {
    int i,y,root;
    for(i=0;i<xn;i++) sets[i]=0;
    for(i=0;i<yn;i++) { sett[i]=0; slack[i]=inf; }
    for(root=0;root<xn&&xy[root]!=nil;root++);
    add_sets(root);
    while(1) {
        relabel();
        for(y=0;y<yn;y++) if(!sett[y]&&slack[y]==0) break;
        if(yx[y]==nil) { augment(y); return; }
        else { add_sets(yx[y]); sett[y]=1; }
    }
}
inline int hungarian() {
    int i,j,c=0;
    matched=0;
    // we must have "xn<yn"
    bool swapxy=0;
    if(xn>yn) {
        swapxy=1;
        int mn=max(xn,yn);
        swap(xn,yn);
        for(int i=0;i<mn;i++)
            for(int j=0;j<i;j++)
                swap(cost[i][j],cost[j][i]);
    }
    for(i=0;i<xn;i++) {
        xy[i]=nil;
        xlabel[i]=0;
        for(j=0;j<yn;j++) xlabel[i]=max(cost[i][j],xlabel[i]);
    }
    for(i=0;i<yn;i++) {
        yx[i]=nil;
        ylabel[i]=0;
    }
    for(i=0;i<xn;i++) phase();
    for(i=0;i<xn;i++) c+=cost[i][xy[i]];
    // recover cost matrix (if necessary)
    if(swapxy) {
        int mn=max(xn,yn);
        swap(xn,yn);
        for(int i=0;i<mn;i++)
            for(int j=0;j<i;j++)
                swap(cost[i][j],cost[j][i]);
    }
    // need special recovery if we want more info than matching value
    return c;
}

```

2.8 Gusfield

```
#define SOURCE 0
#define SINK 1
const unsigned int inf=4000000000u;
int n,m,deg[MAXN],adj[MAXN][MAXN];
unsigned int res[MAXN][MAXN],cap[MAXN][MAXN];
int nei[MAXN],gdeg[MAXN],gadj[MAXN][MAXN];
unsigned int gres[MAXN][MAXN];
unsigned int cut[MAXN][MAXN];
unsigned int cutarr[MAXN*MAXN];
int cutn,ql,qr,que[MAXN],pred[MAXN];
unsigned int aug[MAXN];
bool cutset[MAXN];
int visited[MAXN],visid=0;
inline void augment(int src,int sink) {
    int v=sink; unsigned a=aug[sink];
    while(v!=src) {
        res[pred[v]][v]-=a;
        res[v][pred[v]]+=a;
        v=pred[v];
    }
}
inline bool bfs(int src,int sink) {
    int i,v,u; ++visid;
    ql=qr=0; que[qr++]=src;
    visited[src]=visid; aug[src]=inf;
    while(ql<qr) {
        v=que[ql++];
        for(i=0;i<deg[v];i++) {
            u=adj[v][i];
            if(visited[u]==visid||res[v][u]==0) continue;
            visited[u]=visid; pred[u]=v;
            aug[u]=min(aug[v],res[v][u]);
            que[qr++]=u;
            if(u==sink) return 1;
        }
    }
    return 0;
}
void dfs_src(int v) {
    int i,u;
    visited[v]=visid;
    cutset[v]=SOURCE;
    for(i=0;i<deg[v];i++) {
        u=adj[v][i];
        if(visited[u]<visid&&res[v][u]) dfs_src(u);
    }
}
inline unsigned int maxflow(int src,int sink) {
    int i,j;
    unsigned int f=0;
    for(i=0;i<n;i++) {
        for(j=0;j<deg[i];j++) res[i][adj[i][j]]=cap[i][adj[i][j]];
        cutset[i]=SINK;
    }
    while(bfs(src,sink)) {
        augment(src,sink);
        f+=aug[sink];
    }
    ++visid;
    dfs_src(src);
    return f;
}
inline void gusfield() {
    int i,j;
    unsigned int f;
    for(i=0;i<n;i++) { nei[i]=0; gdeg[i]=0; }
    for(i=1;i<n;i++) {
        f=maxflow(i,nei[i]);
        gres[i][nei[i]]=gres[nei[i]][i]=f;
        gadj[i][gdeg[i]++]=nei[i];
        gadj[nei[i]][gdeg[nei[i]]++]=i;
        for(j=i+1;j<n;j++)
            if(nei[j]==nei[i]&&cutset[j]==SOURCE) nei[j]=i;
    }
}
void dfs(int v,int pred,int src,unsigned int cur) {
    int i,u;
    cut[src][v]=cur;

```

```
for(i=0;i<gdeg[v];i++) {
    u=gadj[v][i];
    if(u==pred) continue;
    dfs(u,v,src,min(cur,gres[v][u]));
}
}
inline void find_all_cuts() {
    int i;
    cutn=0; gusfield();
    for(i=0;i<n;i++) dfs(i,-1,i,inf);
}
}
```

2.9 Max flow with lower/upper bound

```
// Max flow with lower/upper bound on edges
// source = 1 , sink = n
int in[ N ] , out[ N ];
int l[ M ] , r[ M ] , a[ M ] , b[ M ];
int solve(){
    flow.init( n );
    for( int i = 0 ; i < m ; i ++ ){
        in[ r[ i ] ] += a[ i ];
        out[ l[ i ] ] += a[ i ];
        flow.addEdge( l[ i ] , r[ i ] , b[ i ] - a[ i ] );
        // flow on edge from l[ i ] to r[ i ] should
        // be in [a[ i ], b[ i ]].
    }
    int nd = 0;
    for( int i = 1 ; i <= n ; i ++ ){
        if( in[ i ] < out[ i ] ){
            flow.addEdge( i , flow.t , out[ i ] - in[ i ] );
            nd += out[ i ] - in[ i ];
        }
        if( out[ i ] < in[ i ] )
            flow.addEdge( flow.s , i , in[ i ] - out[ i ] );
    }
    // original sink to source
    flow.addEdge( n , 1 , INF );
    if( flow.maxflow() != nd )
        // no solution
        return -1;
    int ans = flow.G[ 1 ].back().c; // source to sink
    flow.G[ 1 ].back().c = flow.G[ n ].back().c = 0;
    // take out super source and super sink
    for( size_t i = 0 ; i < flow.G[ flow.s ].size() ; i
        ++ ){
        flow.G[ flow.s ][ i ].c = 0;
        Edge &e = flow.G[ flow.s ][ i ];
        flow.G[ e.v ][ e.r ].c = 0;
    }
    for( size_t i = 0 ; i < flow.G[ flow.t ].size() ; i
        ++ ){
        flow.G[ flow.t ][ i ].c = 0;
        Edge &e = flow.G[ flow.t ][ i ];
        flow.G[ e.v ][ e.r ].c = 0;
    }
    flow.addEdge( flow.s , 1 , INF );
    flow.addEdge( n , flow.t , INF );
    flow.reset();
    return ans + flow.maxflow();
}
```

2.10 Relabel to Front

```
/* Relabel-to-Front */
// tested with sgu-212 (more testing suggested)
int n,m,layer,src,sink,lv1[MAXN];
Edge ed[MAXN];
int deg[MAXN],adj[MAXN][MAXN];
int res[MAXN][MAXN]; // residual capacity
// graph (i.e. all things above) should be constructed
// beforehand
list<int> lst; // discharge list
int ef[MAXN],ht[MAXN];
// excess flow, height
int apt[MAXN]; // the next adj index to try push
inline void push(int v,int u) {

```

```

int a=min(ef[v],res[v][u]);
ef[v]-=a; ef[u]+=a;
res[v][u]-=a; res[u][v]+=a;
}
inline void relabel(int v) {
int i,u;
ht[v]=2*n;
for(i=0;i<deg[v];i++) {
u=adj[v][i];
if(res[v][u]) ht[v]=min(ht[u]+1,ht[v]);
}
}
inline void initPreflow() {
int i,u;
lst.clear();
for(i=0;i<n;i++) {
ht[i]=ef[i]=0; apt[i]=0;
if(i!=src&&i!=sink) lst.push_back(i);
}
ht[src]=n;
for(i=0;i<deg[src];i++) {
u=adj[src][i];
ef[u]=res[src][u];
ef[src]-=ef[u];
res[u][src]=ef[u];
res[src][u]=0;
}
}
inline void discharge(int v) {
int u;
while(ef[v]) {
if(apt[v]==deg[v]) {
relabel(v);
apt[v]=0;
continue;
}
u=adj[v][apt[v]];
if(res[v][u]&&ht[v]==ht[u]+1) push(v,u);
else apt[v]++;
}
}
inline void relabelToFront() {
int oldh,v;
list<int>::iterator it;
initPreflow();
for(it=lst.begin();it!=lst.end();it++) {
v=*it; oldh=ht[v]; discharge(v);
if(ht[v]>oldh) {
lst.push_front(v);
lst.erase(it);
it=lst.begin();
}
}
}
}

```

2.11 Flow Method

Maximize $c^T x$ subject to $Ax \leq b$, $x \geq 0$;
with the corresponding symmetric dual problem,
Minimize $b^T y$ subject to $A^T y \geq c$, $y \geq 0$.

Maximize $c^T x$ subject to $Ax \leq b$;
with the corresponding asymmetric dual problem,
Minimize $b^T y$ subject to $A^T y = c$, $y \geq 0$.

Minimum vertex cover on bipartite graph =
Maximum matching on bipartite graph =
Max flow with source to one side, other side to sink

To reconstruct the minimum vertex cover, dfs from each unmatched vertex on the left side **and** with unused edges only. Equivalently, dfs from source with unused edges only **and** without visiting sink. Then, a vertex is chosen
iff. it is on the left side **and** without visited **or** on the right side **and** visited through dfs.

有源匯，有下界，最大流，無費用。

先從 t 連向 s ，容量設為無限大。這樣就變成了無源匯的情況。將每條有下界的邊先滿上下界的流量，然後更新盈餘量（入的流量-出的流量）。新建超級源 ss 和超級匯 tt ，若某個點 u 的盈餘量 >0 則 $ss \rightarrow u$ ，容量為 u 的盈餘量。否則 $u \rightarrow tt$ ，容量為 u 的盈餘量的相反數。如果一個點的盈餘量 >0 ，則它是一定要流出去的，所以要從 ss 連向它，使它去找這些流量的出路。建完了圖以後求一遍最大流，如果從 ss 連出的所有邊都滿流，則有解。在得到的殘留網路（原圖）上再求一次最大流即可。

3 Math

3.1 FFT

```

// const int MAXN = 262144;
// (must be 2^k)
// before any usage, run pre_fft() first
//
// To implement poly. multiply:
//
// fft( n , a );
// fft( n , b );
// for( int i = 0 ; i < n ; i++ )
//   c[ i ] = a[ i ] * b[ i ];
// fft( n , c , 1 );
//
// then you have the result in c :: [cplx]
typedef long double ld;
typedef complex<ld> cplx;
const ld PI = acos(-1);
const cplx I(0, 1);
cplx omega[MAXN+1];
void pre_fft(){
for(int i=0; i<=MAXN; i++){
omega[i] = exp(i * 2 * PI / MAXN * I);
}
// n must be 2^k
void fft(int n, cplx a[], bool inv=false){
int basic = MAXN / n;
int theta = basic;
for (int m = n; m >= 2; m >>= 1) {
int mh = m >> 1;
for (int i = 0; i < mh; i++) {
cplx w = omega[inv ? MAXN-(i*theta%MAXN) : i*theta%MAXN];
for (int j = i; j < n; j += m) {
int k = j + mh;
cplx x = a[j] - a[k];
a[j] += a[k];
a[k] = w * x;
}
}
theta = (theta * 2) % MAXN;
}
int i = 0;
for (int j = 1; j < n - 1; j++) {
for (int k = n >> 1; k > (i ^ k); k >>= 1);
if (j < i) swap(a[i], a[j]);
}
if (inv)
for (i = 0; i < n; i++)
a[i] /= n;
}

```

3.2 NTT

```

LL P=2013265921,root=31;
int MAXNUM=4194304;
// Remember coefficient are mod P
/*
p=a*2^n+1
n  2^n      p      a      root
5  32      97      3      5

```



```

6  64      193      3      5
7  128     257      2      3
8  256     257      1      3
9  512     7681     15     17
10 1024    12289    12     11
11 2048    12289     6     11
12 4096    12289     3     11
13 8192    40961     5      3
14 16384   65537     4      3
15 32768   65537     2      3
16 65536   65537     1      3
17 131072  786433     6     10
18 262144  786433     3     10 (605028353,
    2308, 3)
19 524288  5767169    11     3
20 1048576 7340033     7      3
21 2097152 23068673    11     3
22 4194304 104857601   25     3
23 8388608 167772161  20     3
24 16777216 167772161 10     3
25 33554432 167772161 5      3 (1107296257, 33,
    10)
26 67108864 469762049 7      3
27 134217728 2013265921 15     31
*/
LL bigmod(LL a, LL b){
    if(b==0) return 1;
    return (bigmod((a*a)%P, b/2)*(b%2?a:1LL))%P;
}
LL inv(LL a, LL b){
    if(a==1) return 1;
    return (((LL)(a-inv(b%a,a))*b+1)/a)%b;
}
std::vector<LL> ps(MAXNUM);
std::vector<LL> rev(MAXNUM);
struct poly{
    std::vector<LL> co;
    int n; //polynomial degree = n
    poly(int d){n=d; co.resize(n+1, 0);}
    void trans2(int NN){
        int r=0, st, N;
        unsigned int a, b;
        while((1<<r)<(NN>>1)) ++r;
        for(N=2; N<=NN; N<=1, --r){
            for(st=0; st<NN; st+=N){
                int i, ss=st+(N>>1);
                for(i=(N>>1)-1; i>=0; --i){
                    a=co[st+i]; b=(ps[i<<r]*co[ss+i])%P;
                    co[st+i]=a+b; if(co[st+i]>=P) co[st+i]-=P;
                    co[ss+i]=a-P-b; if(co[ss+i]>=P) co[ss+i]-=P;
                }
            }
        }
    }
}
void trans1(int NN){
    int r=0, st, N;
    unsigned int a, b;
    for(N=NN; N>1; N>=1, ++r){
        for(st=0; st<NN; st+=N){
            int i, ss=st+(N>>1);
            for(i=(N>>1)-1; i>=0; --i){
                a=co[st+i]; b=co[ss+i];
                co[st+i]=a+b; if(co[st+i]>=P) co[st+i]-=P;
                co[ss+i]=((a-P-b)*ps[i<<r])%P;
            }
        }
    }
}
poly operator*(const poly& _b) const{
    poly a=*this, b=_b;
    int k=n+b.n, i, N=1;
    while(N<=k) N*=2;
    a.co.resize(N, 0); b.co.resize(N, 0);
    int r=bigmod(root, (P-1)/N), Ni=inv(N, P);
    ps[0]=1;
    for(i=1; i<N; ++i) ps[i]=(ps[i-1]*r)%P;
    a.trans1(N); b.trans1(N);
    for(i=0; i<N; ++i) a.co[i]=((LL)a.co[i]*b.co[i])%P;
    r=inv(r, P);
    for(i=1; i<N/2; ++i) std::swap(ps[i], ps[N-i]);
    a.trans2(N);
    for(i=0; i<N; ++i) a.co[i]=((LL)a.co[i]*Ni)%P;
}

```

```

    a.n=n+_b.n; return a;
}
};

```

3.3 Fast Walsh Transform

```

/*
 * xor convolution:
 * x = (x0,x1) , y = (y0,y1)
 * z = ( x0y0 + x1y1 , x0y1 + x1y0 )
 * =>
 * x' = ( x0+x1 , x0-x1 ) , y' = ( y0+y1 , y0-y1 )
 * z' = ( ( x0+x1 )( y0+y1 ) , ( x0-x1 )( y0-y1 ) )
 * z = (1/2) * z'
 * or convolution:
 * x = ( x0 , x0+x1 )
 * and convolution:
 * x = ( x0+x1 , x1 )
 */
typedef long long LL;
const int MAXN = (1<<20)+10;
const LL MOD = 1e9+7;
inline LL pw( LL x , LL k ) {
    LL res = 1;
    for( LL bs = x ; k ; k >= 1, bs = (bs * bs)%MOD ){
        if( k&1 ) res = ( res * bs ) % MOD;
    }
    return res;
}
inline LL inv( LL x ) {
    return pw( x , MOD-2 );
}
inline void fwt( LL x[ MAXN ] , int N , bool inv=0 ) {
    for( int d = 1 ; d < N ; d <= 1 ) {
        int d2 = d<<1;
        for( int s = 0 ; s < N ; s += d2 ) {
            for( int i = s , j = s+d ; i < s+d ; i++, j++ ){
                LL ta = x[ i ] , tb = x[ j ];
                x[ i ] = ta+tb;
                x[ j ] = ta-tb;
                if( x[ i ] >= MOD ) x[ i ] -= MOD;
                if( x[ j ] < 0 ) x[ j ] += MOD;
            }
        }
    }
    if( inv )
        for( int i = 0 ; i < N ; i++ ) {
            x[ i ] *= inv( N );
            x[ i ] %= MOD;
        }
}

```

3.4 BigInt

```

struct BigInt{
    static const int LEN = 60;
    static const int BIGMOD = 10000;
    int s;
    int vl, v[LEN];
    // vector<int> v;
    BigInt() : s(1) { vl = 0; }
    BigInt(long long a) {
        s = 1; vl = 0;
        if (a < 0) { s = -1; a = -a; }
        while (a) {
            push_back(a % BIGMOD);
            a /= BIGMOD;
        }
    }
    BigInt(string str) {
        s = 1; vl = 0;
        int stPos = 0, num = 0;
        if (!str.empty() && str[0] == '-') {
            stPos = 1;
            s = -1;
        }
        for (int i=SZ(str)-1, q=1; i>=stPos; i--) {
            num += (str[i] - '0') * q;
        }
    }
}

```

```

    if ((q * 10) >= BIGMOD) {
        push_back(num);
        num = 0; q = 1;
    }
    if (num) push_back(num);
    n();
}
int len() const {
    return vl;
    // return SZ(v);
}
bool empty() const { return len() == 0; }
void push_back(int x) {
    v[vl++] = x;
    // v.PB(x);
}
void pop_back() {
    vl--;
    // v.pop_back();
}
int back() const {
    return v[vl-1];
    // return v.back();
}
void n() {
    while (!empty() && !back()) pop_back();
}
void resize(int nl) {
    vl = nl;
    fill(v, v+vl, 0);
    // v.resize(nl);
    // fill(ALL(v), 0);
}
void print() const {
    if (empty()) { putchar('0'); return; }
    if (s == -1) putchar('-');
    printf("%d", back());
    for (int i=len()-2; i>=0; i--) printf("%.4d", v[i]);
}
friend std::ostream& operator << (std::ostream& out,
    const Bigint &a) {
    if (a.empty()) { out << "0"; return out; }
    if (a.s == -1) out << "-";
    out << a.back();
    for (int i=a.len()-2; i>=0; i--) {
        char str[10];
        snprintf(str, 5, "%.4d", a.v[i]);
        out << str;
    }
    return out;
}
int cp3(const Bigint &b) const {
    if (s != b.s) return s - b.s;
    if (s == -1) return -(*this).cp3(-b);
    if (len() != b.len()) return len() - b.len(); //int
    for (int i=len()-1; i>=0; i--)
        if (v[i] != b.v[i]) return v[i] - b.v[i];
    return 0;
}
bool operator<(const Bigint &b) const {
    return cp3(b)<0; }
bool operator<=(const Bigint &b) const {
    return cp3(b)<=0; }
bool operator==(const Bigint &b) const {
    return cp3(b)==0; }
bool operator!=(const Bigint &b) const {
    return cp3(b)!=0; }
bool operator>(const Bigint &b) const {
    return cp3(b)>0; }
bool operator>=(const Bigint &b) const {
    return cp3(b)>=0; }
Bigint operator - () const {
    Bigint r = (*this);
    r.s = -r.s;
    return r;
}
Bigint operator + (const Bigint &b) const {
    if (s == -1) return -(-(*this)+(-b));
    if (b.s == -1) return (*this)-(-b);
    Bigint r;
    int nl = max(len(), b.len());

```

```

    r.resize(nl + 1);
    for (int i=0; i<nl; i++) {
        if (i < len()) r.v[i] += v[i];
        if (i < b.len()) r.v[i] += b.v[i];
        if (r.v[i] >= BIGMOD) {
            r.v[i+1] += r.v[i] / BIGMOD;
            r.v[i] %= BIGMOD;
        }
    }
    r.n();
    return r;
}
Bigint operator - (const Bigint &b) const {
    if (s == -1) return -(-(*this)-(-b));
    if (b.s == -1) return (*this)+(-b);
    if ((*this) < b) return -(b-(*this));
    Bigint r;
    r.resize(len());
    for (int i=0; i<len(); i++) {
        r.v[i] += v[i];
        if (i < b.len()) r.v[i] -= b.v[i];
        if (r.v[i] < 0) {
            r.v[i] += BIGMOD;
            r.v[i+1]--;
        }
    }
    r.n();
    return r;
}
Bigint operator * (const Bigint &b) {
    Bigint r;
    r.resize(len() + b.len() + 1);
    r.s = s * b.s;
    for (int i=0; i<len(); i++) {
        for (int j=0; j<b.len(); j++) {
            r.v[i+j] += v[i] * b.v[j];
            if (r.v[i+j] >= BIGMOD) {
                r.v[i+j+1] += r.v[i+j] / BIGMOD;
                r.v[i+j] %= BIGMOD;
            }
        }
    }
    r.n();
    return r;
}
Bigint operator / (const Bigint &b) {
    Bigint r;
    r.resize(max(1, len()-b.len()+1));
    int oriS = s;
    Bigint b2 = b; // b2 = abs(b)
    s = b2.s = r.s = 1;
    for (int i=r.len()-1; i>=0; i--) {
        int d=0, u=BIGMOD-1;
        while(d<u) {
            int m = (d+u+1)>>1;
            r.v[i] = m;
            if ((r*b2) > (*this)) u = m-1;
            else d = m;
        }
        r.v[i] = d;
    }
    s = oriS;
    r.s = s * b.s;
    r.n();
    return r;
}
Bigint operator % (const Bigint &b) {
    return (*this)-(*this)/b*b;
}
};

```

3.5 Linear Recurrence

```

LL n, m;
LL dp[ N + N ];
void pre_dp() {
    dp[ 0 ] = 1;
    LL bdr = min( m + m , n );
    for( LL i = 1 ; i <= bdr ; i ++ )
        for( LL j = i - 1 ; j >= max( 0LL , i - m ) ; j -- )

```



```

        dp[ i ] = add( dp[ i ] , dp[ j ] );
    }
vector<LL> Mul( vector<LL>& v1, vector<LL>& v2 ){
    int _sz1 = (int)v1.size();
    int _sz2 = (int)v2.size();
    assert( _sz1 == m );
    assert( _sz2 == m );
    vector<LL> _v( m + m );
    for( int i = 0 ; i < m + m ; i ++ ) _v[ i ] = 0;
    // expand
    for( int i = 0 ; i < _sz1 ; i ++ )
        for( int j = 0 ; j < _sz2 ; j ++ )
            _v[ i + j + 1 ] = add( _v[ i + j + 1 ] ,
                                  mul( v1[ i ] , v2[ j ] ) );
    // shrink
    for( int i = 0 ; i < m ; i ++ )
        for( int j = 1 ; j <= m ; j ++ )
            _v[ i + j ] = add( _v[ i + j ] , _v[ i ] );
    for( int i = 0 ; i < m ; i ++ )
        _v[ i ] = _v[ i + m ];
    _v.resize( m );
    return _v;
}
vector<LL> I, A;
void solve(){
    pre_dp();
    if( n <= m + m ){
        printf( "%lld\n" , dp[ n ] );
        exit( 0 );
    }
    I.resize( m );
    A.resize( m );
    for( int i = 0 ; i < m ; i ++ ) I[ i ] = A[ i ] = 1;
    // dp[ n ] = /Sum_{i=0}^{m-1} A_i * dp[ n - i - 1 ]
    LL dlt = ( n - m ) / m;
    LL rdlt = dlt * m;
    while( dlt ){
        if( dlt & 1LL ) I = Mul( I , A );
        A = Mul( A , A );
        dlt >>= 1;
    }
    LL ans = 0;
    for( int i = 0 ; i < m ; i ++ )
        ans = add( ans, mul( I[ i ], dp[ n - i - 1 - rdlt ] ) );
    printf( "%lld\n" , ans );
}

```

3.6 Miller Rabin

```

// n < 4,759,123,141      3 : 2, 7, 61
// n < 1,122,004,669,633  4 : 2, 13, 23, 1662803
// n < 3,474,749,660,383  6 : pimes <= 13
// n < 2^64               7 :
// 2, 325, 9375, 28178, 450775, 9780504, 1795265022
// Make sure testing integer is in range [2, n-2] if
// you want to use magic.
LL power(LL x, LL p, LL mod){
    LL 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(LL a, LL n, LL u, int t){
    LL x=power(a,u,n);
    for(int i=0;i<t;i++){
        LL nx=mult(x,x,n);
        if(nx==1&&x!=1&&x!=n-1) return 1;
        x=nx;
    }
    return x!=1;
}
bool miller_rabin(LL 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;
    LL u=n-1;

```

```

    int t=0;
    // n-1 = u*2^t
    while(!(u&1)) {
        u>>=1;
        t++;
    }
    while(s--){
        LL a=randll()%n+1;
        if(witness(a,n,u,t)) return 0;
    }
    return 1;
}

```

3.7 Simplex

```

const int MAXN = 111;
const int MAXM = 111;
const double eps = 1E-10;
double a[MAXN][MAXM], b[MAXN], c[MAXN], d[MAXN][MAXM];
double x[MAXN];
int ix[MAXN + MAXM]; // !!! array all indexed from 0
// max{cx} subject to {Ax<=b, x>=0}
// n: constraints, m: vars !!!
// x[] is the optimal solution vector
//
// usage :
// value = simplex(a, b, c, N, M);
double simplex(double a[MAXN][MAXM], double b[MAXN],
               double c[MAXN], int n, int m){
    ++m;
    int r = n, s = m - 1;
    memset(d, 0, sizeof(d));
    for( int i = 0; i < n + m; ++i) ix[i] = i;
    for( int i = 0; i < n; ++i) {
        for( int j = 0; j < m - 1; ++j) d[i][j] = -a[i][j];
        d[i][m - 1] = 1;
        d[i][m] = b[i];
        if( d[r][m] > d[i][m] ) r = i;
    }
    for( int j = 0; j < m - 1; ++j) d[n][j] = c[j];
    d[n + 1][m - 1] = -1;
    for( double dd;; ) {
        if( r < n ) {
            int t = ix[s]; ix[s] = ix[r + m]; ix[r + m] = t;
            d[r][s] = 1.0 / d[r][s];
            for( int j = 0; j <= m; ++j)
                if( j != s ) d[r][j] *= -d[r][s];
            for( int i = 0; i <= n + 1; ++i) if( i != r ) {
                for( int j = 0; j <= m; ++j) if( j != s )
                    d[i][j] += d[r][j] * d[i][s];
                d[i][s] *= d[r][s];
            }
        }
        r = -1; s = -1;
        for( int j = 0; j < m; ++j)
            if( s < 0 || ix[s] > ix[j] ) {
                if( d[n + 1][j] > eps ||
                    (d[n + 1][j] > -eps && d[n][j] > eps) )
                    s = j;
            }
        if( s < 0 ) break;
        for( int i = 0; i < n; ++i) if( d[i][s] < -eps ) {
            if( r < 0 ||
                (dd = d[r][m] / d[r][s] - d[i][m] / d[i][s]) <
                  -eps ||
                (dd < eps && ix[r + m] > ix[i + m]) )
                r = i;
        }
        if( r < 0 ) return -1; // not bounded
    }
    if( d[n + 1][m] < -eps ) return -1; // not executable
    double ans = 0;
    for( int i = 0; i < m; i ++ ) x[i] = 0;
    for( int i = m; i < n + m; ++i) { // the missing
        enumerated x[i] = 0
        if( ix[i] < m - 1 ) {
            ans += d[i - m][m] * c[ix[i]];
            x[ix[i]] = d[i - m][m];
        }
    }
}

```

```

return ans;
}

```

3.8 Faulhaber

```

/* faulhaber 's formula -
 * cal power sum formula of all p=1~k in O(k^2) */
#define MAXK 2500
const int mod = 1000000007;
int b[MAXK];
// bernoulli number
int inv[MAXK+1];
// inverse
int cm[MAXK+1][MAXK+1]; // combinactories
int co[MAXK][MAXK+2];
// coeeficient of x^j when p=i
int add(int a,int b) { return a+b<mod?a+b:a+b-mod; }
int sub(int a,int b) { return a<b?a-b+mod:a-b; }
inline int getinv(int x) {
    int a=x,b=mod,a0=1,a1=0,b0=0,b1=1;
    while(b) {
        int q,t;
        q=a/b; t=b; b=a-b*q; a=t;
        t=b0; b0=a0-b0*q; a0=t;
        t=b1; b1=a1-b1*q; a1=t;
    }
    return a0<0?a0+mod:a0;
}
inline void pre() {
    /* combinational */
    for(int i=0;i<=MAXK;i++) {
        cm[i][0]=cm[i][i]=1;
        for(int j=1;j<i;j++)
            cm[i][j]=add(cm[i-1][j-1],cm[i-1][j]);
    }
    /* inverse */
    for(int i=1;i<=MAXK;i++) inv[i]=getinv(i);
    /* bernoulli */
    b[0]=1; b[1]=getinv(2); // with b[1] = 1/2
    for(int i=2;i<MAXK;i++) {
        if(i&1) { b[i]=0; continue; }
        b[i]=1;
        for(int j=0;j<i;j++)
            b[i]=sub(b[i],
                (LL)cm[i][j]*b[j]%mod*inv[i-j+1]%mod);
    }
    /* faulhaber */
    // sigma_x=1~n {x^p} = 1/(p+1) * sigma_j=0~p { C(p+1,
        j) * B_j * n^(p-j+1)}
    for(int i=1;i<MAXK;i++) {
        co[i][0]=0;
        for(int j=0;j<i;j++)
            co[i][i-j+1]=
                (LL)inv[i+1]%mod*cm[i+1][j]%mod*b[j]%mod;
    }
}
inline int power(int x,int p) {
    int s=1,m=x;
    while(p) {
        if(p&1) s=(LL)s*m%mod;
        p>>=1; m=(LL)m*m%mod;
    }
    return s;
}
/* sample usage: return f(n,p) = sigma_x=1~n (x^p) */
inline int solve(int n,int p) {
    int sol=0,m=n;
    for(int i=1;i<=p+1;i++) {
        sol=add(sol,(LL)co[p][i]*m%mod);
        m=(LL)m*n%mod;
    }
    return sol;
}

```

3.9 Chinese Remainder

```

int pfn;
// number of distinct prime factors

```

```

int pf[MAXN]; // prime factor powers
int rem[MAXN]; // corresponding remainder
int pm[MAXN];
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]=(LL)m*inverse(m,pf[i])%mod;
        s=(s+(LL)pm[i]*rem[i])%mod;
    }
    return s;
}

```

3.10 Pollard Rho

```

// does not work when n is prime
LL modit(LL x,LL mod) {
    if(x>=mod) x-=mod;
    //if(x<0) x+=mod;
    return x;
}
LL mult(LL x,LL y,LL mod) {
    LL s=0,m=x%mod;
    while(y) {
        if(y&1) s=modit(s+m,mod);
        y>>=1;
        m=modit(m+m,mod);
    }
    return s;
}
LL f(LL x,LL mod) {
    return modit(mult(x,x,mod)+1,mod);
}
LL pollard_rho(LL n) {
    if(!(n&1)) return 2;
    while(true) {
        LL y=2, x=rand()%(n-1)+1, res=1;
        for(int sz=2; res==1; sz*=2) {
            for(int i=0; i<sz && res<=1; i++) {
                x = f(x, n);
                res = __gcd(abs(x-y), n);
            }
            y = x;
        }
    }
}

```

```

    if (res!=0 && res!=n) return res;
}
}

```

3.11 Poly Generator

```

class PolyGen {
    /* 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<LL> coef;
    // initialize and calculate f(x), vector _fx should
    // be filled with f(0) to f(n)
    PolyGen(int _n,vector<LL> _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)
        LL eval(int x) {
            LL m=1,ret=0;
            for(int i=0;i<=n;i++){
                ret+=coef[i]*m;
                m=m*(x-i)/(i+1);
            }
            return ret;
        }
    };
};

```

3.12 Matrix Pseudo Inverse

```

Mat pinv( Mat m ){
    Mat res = I;
    FZ( used );
    for( int i = 0 ; i < W ; i ++ ){
        int piv = -1;
        for( int j = 0 ; j < W ; j ++ ){
            if( used[ j ] ) continue;
            if( abs( m.v[ j ][ i ] ) > EPS ){
                piv = j;
                break;
            }
        }
        if( piv == -1 ) continue;
        used[ i ] = true;
        swap( m.v[ piv ], m.v[ i ] );
        swap( res.v[ piv ], res.v[ i ] );
        LD rat = m.v[ i ][ i ];
        for( int j = 0 ; j < W ; j ++ ){
            m.v[ i ][ j ] /= rat;
            res.v[ i ][ j ] /= rat;
        }
        for( int j = 0 ; j < W ; j ++ ){
            if( j == i ) continue;
            rat = m.v[ j ][ i ];
            for( int k = 0 ; k < W ; k ++ ){
                m.v[ j ][ k ] -= rat * m.v[ i ][ k ];
                res.v[ j ][ k ] -= rat * res.v[ i ][ k ];
            }
        }
    }
    for( int i = 0 ; i < W ; i ++ ){
        if( used[ i ] ) continue;
        for( int j = 0 ; j < W ; j ++ )
            res.v[ i ][ j ] = 0;
    }
    return res;
}

```

3.13 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);
}
}

```

3.14 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);
}

```

3.15 Primes and μ function

```

/*
 * 12721
 * 13331
 * 14341
 * 75577
 * 123457
 * 222557
 * 556679
 * 999983
 * 1097774749
 * 1076767633
 * 100102021
 * 999997771
 * 1001010013
 * 1000512343
 * 987654361
 * 999991231
 * 999888733
 * 98789101
 * 987777733
 * 999991921
 * 1010101333
 * 1010102101
 * 1000000000039
 * 100000000000037
 * 2305843009213693951
 * 4611686018427387847
 * 9223372036854775783
 * 18446744073709551557
 */
int mu[ N ], p_tbl[ N ];
vector<int> primes;
void sieve() {
    mu[ 1 ] = p_tbl[ 1 ] = 1;
    for( int i = 2 ; i < N ; i ++ ){
        if( !p_tbl[ i ] ){
            p_tbl[ i ] = i;
            primes.push_back( i );
            mu[ i ] = -1;
        }
        for( int p : primes ){
            int x = i * p;
            if( x >= M ) break;
            p_tbl[ x ] = p;
        }
    }
}

```

```

    mu[ x ] = -mu[ i ];
    if( i % p == 0 ){
        mu[ x ] = 0;
        break;
    }
}
}
vector<int> factor( int x ){
    vector<int> fac{ 1 };
    while( x > 1 ){
        int fn = SZ(fac), p = p_tbl[ x ], pos = 0;
        while( x % p == 0 ){
            x /= p;
            for( int i = 0 ; i < fn ; i ++ )
                fac.PB( fac[ pos ++ ] * p );
        }
    }
    return fac;
}

```

3.16 Result

```

/*
Lucas' Theorem:
For non-negative integer n,m and prime P,
 $C(m,n) \bmod P = C(m/M,n/M) * C(m \% M,n \% M) \bmod P$ 
= mult_i ( C(m_i,n_i) )
where m_i is the i-th digit of m in base P.
--
Sum of Two Squares Thm (Legendre)
For a given positive integer N, let
D1 = (# of d \in \mathbb{N} dividing N that d=1(mod 4))
D3 = (# of d \in \mathbb{N} dividing N that d=3(mod 4))
then N can be written as a sum of two squares in
exactly  $R(N) = 4(D1-D3)$  ways.
--
Difference of D1-D3 Thm
let  $N=2^t * [p_1^{e_1} * \dots * p_r^{e_r}] * [q_1^{f_1} * \dots * q_s^{f_s}]$ 
    <-mod 4 = 1 prime->    <-mod 4 = 3 prime->
then  $D1 - D3 = (e_1+1)(e_2+1)\dots(e_r+1)$  if fi all even
                0                      if any fi is odd
--
Pick's Theorem
 $A = i + b/2 - 1$ 
*/

```

4 Geometry

4.1 halfPlaneIntersection

4.2 Intersection of 2 lines

```

Pt interPnt( Line l1, Line l2, bool &res ){
    Pt p1, p2, q1, q2;
    tie(p1, p2) = l1;
    tie(q1, q2) = l2;
    double f1 = (p2 - p1) ^ (q1 - p1);
    double f2 = (p2 - p1) ^ (p1 - q2);
    double f = (f1 + f2);
    if( fabs(f) < eps ){
        res = false;
        return {0, 0};
    }
    res = true;
    return q1 * (f2 / f) + q2 * (f1 / f);
}
bool isin( Line l0, Line l1, Line l2 ){
    // Check inter(l1, l2) in l0
    bool res;
    Pt p = interPnt(l1, l2, res);
    return ( (l0.SE - l0.FI) ^ (p - l0.FI) ) > eps;
}
/* If no solution, check: 1. ret.size() < 3
 * Or more precisely, 2. interPnt(ret[0], ret[1])
 * in all the lines. (use (l.S - l.F) ^ (p - l.F) > 0

```

```

*/
/* --^-- Line.FI --^-- Line.SE --^-- */
vector<Line> halfPlaneInter( vector<Line> lines ){
    int sz = lines.size();
    vector<double> ata(sz), ord(sz);
    for( int i=0; i<sz; i++ ){
        ord[i] = i;
        Pt d = lines[i].SE - lines[i].FI;
        ata[i] = atan2(d.Y, d.X);
    }
    sort( ord.begin(), ord.end(), [&](int i, int j) {
        if( fabs(ata[i] - ata[j]) < eps ){
            return ( (lines[i].SE - lines[i].FI) ^
                    (lines[j].SE - lines[j].FI) ) < 0;
        }
        return ata[i] < ata[j];
    });
    vector<Line> fin;
    for( int i=0; i<sz; i++ ){
        if ( !i or fabs(ata[ord[i]] - ata[ord[i-1]]) > eps )
            fin.PB(lines[ord[i]]);
    }
    deque<Line> dq;
    for( int i=0; i<(int)(fin.size()); i++ ){
        while((int)(dq.size()) >= 2 and
            not isin(fin[i], dq[(int)(dq.size())-2],
                    dq[(int)(dq.size())-1]))
            dq.pop_back();
        while((int)(dq.size()) >= 2 and
            not isin(fin[i], dq[0], dq[1]))
            dq.pop_front();
        dq.push_back(fin[i]);
    }
    while( (int)(dq.size()) >= 3 and
        not isin(dq[0], dq[(int)(dq.size())-2],
                dq[(int)(dq.size())-1]))
        dq.pop_back();
    while( (int)(dq.size()) >= 3 and
        not isin(dq[(int)(dq.size())-1], dq[0], dq[1]))
        dq.pop_front();
    vector<Line> res(dq.begin(), dq.end());
    return res;
}

```

4.3 Intersection of 2 circles

```

vector<Pt> interCircle( Pt o1 , D r1 , Pt o2 , D r2 ){
    D d2 = ( o1 - o2 ) * ( o1 - o2 );
    D d = sqrt(d2);
    if( d > r1 + r2 ) return {};
    Pt u = (o1+o2)*0.5 + (o1-o2)*((r2*r2-r1*r1)/(2*d2));
    D A = sqrt((r1+r2+d)*(r1-r2+d)*(r1+r2-d)*(r1-r2-d));
    Pt v = Pt( o1.Y-o2.Y , -o1.X + o2.X ) * A / (2*d2);
    return {u+v, u-v};
}

```

4.4 Intersection of 2 segments

```

int ori( const PLL& o , const PLL& a , const PLL& b ){
    LL ret = ( a - o ) ^ ( b - o );
    return ret / max( 1ll , abs( ret ) );
}
// p1 == p2 || q1 == q2 need to be handled
bool banana( const PLL& p1 , const PLL& p2 ,
              const PLL& q1 , const PLL& q2 ){
    if( ( ( p2 - p1 ) ^ ( q2 - q1 ) ) == 0 ){ // parallel
        if( ori( p1 , p2 , q1 ) ) return false;
        return ( ( p1 - q1 ) * ( p2 - q1 ) ) <= 0 ||
            ( ( p1 - q2 ) * ( p2 - q2 ) ) <= 0 ||
            ( ( q1 - p1 ) * ( q2 - p1 ) ) <= 0 ||
            ( ( q1 - p2 ) * ( q2 - p2 ) ) <= 0;
    }
    return (ori( p1, p2, q1 ) * ori( p1, p2, q2 ) <= 0) &&
        (ori( q1, q2, p1 ) * ori( q1, q2, p2 ) <= 0);
}

```

4.5 KD Tree

```
const int MXN = 100005;
struct KDTree {
    struct Node {
        int x,y,x1,y1,x2,y2;
        int id,f;
        Node *L, *R;
    }tree[MXN];
    int n;
    Node *root;
    LL dis2(int x1, int y1, int x2, int y2) {
        LL dx = x1-x2;
        LL dy = y1-y2;
        return dx*dx+dy*dy;
    }
    static bool cmpx(Node& a, Node& b){ return a.x<b.x; }
    static bool cmpy(Node& a, Node& b){ return a.y<b.y; }
    void init(vector<pair<int,int>> ip) {
        n = ip.size();
        for (int i=0; i<n; i++) {
            tree[i].id = i;
            tree[i].x = ip[i].first;
            tree[i].y = ip[i].second;
        }
        root = build_tree(0, n-1, 0);
    }
    Node* build_tree(int L, int R, int dep) {
        if (L>R) return nullptr;
        int M = (L+R)/2;
        tree[M].f = dep%2;
        nth_element(tree+L, tree+M, tree+R+1, tree[M].f ?
            cmpy : cmpx);
        tree[M].x1 = tree[M].x2 = tree[M].x;
        tree[M].y1 = tree[M].y2 = tree[M].y;

        tree[M].L = build_tree(L, M-1, dep+1);
        if (tree[M].L) {
            tree[M].x1 = min(tree[M].x1, tree[M].L->x1);
            tree[M].x2 = max(tree[M].x2, tree[M].L->x2);
            tree[M].y1 = min(tree[M].y1, tree[M].L->y1);
            tree[M].y2 = max(tree[M].y2, tree[M].L->y2);
        }
        tree[M].R = build_tree(M+1, R, dep+1);
        if (tree[M].R) {
            tree[M].x1 = min(tree[M].x1, tree[M].R->x1);
            tree[M].x2 = max(tree[M].x2, tree[M].R->x2);
            tree[M].y1 = min(tree[M].y1, tree[M].R->y1);
            tree[M].y2 = max(tree[M].y2, tree[M].R->y2);
        }
        return tree+M;
    }
    int touch(Node* r, int x, int y, LL d2){
        LL dis = sqrt(d2)+1;
        if (x<r->x1-dis || x>r->x2+dis ||
            y<r->y1-dis || y>r->y2+dis)
            return 0;
        return 1;
    }
    void nearest(Node* r, int x, int y,
        int &mID, LL &md2){
        if (!r || !touch(r, x, y, md2)) return;
        LL d2 = dis2(r->x, r->y, x, y);
        if (d2 < md2 || (d2 == md2 && mID < r->id)) {
            mID = r->id;
            md2 = d2;
        }
        // search order depends on split dim
        if ((r->f == 0 && x < r->x) ||
            (r->f == 1 && y < r->y)) {
            nearest(r->L, x, y, mID, md2);
            nearest(r->R, x, y, mID, md2);
        } else {
            nearest(r->R, x, y, mID, md2);
            nearest(r->L, x, y, mID, md2);
        }
    }
    int query(int x, int y) {
        int id = 1029384756;
        LL d2 = 102938475612345678LL;
        nearest(root, x, y, id, d2);
    }
};
```

```
        return id;
    }
}tree;
```

4.6 Poly Union

```
#define eps 1e-8
class PY{ public:
    int n;
    Pt pt[5];
    Pt& operator[](const int x){ return pt[x]; }
    void input(){
        int i; n=4;
        for(i=0;i<n;i++) scanf("%lf%lf",&pt[i].x,&pt[i].y);
    }
    double getArea(){
        int i; double s=pt[n-1]^pt[0];
        for(i=0;i<n-1;i++) s+=pt[i]^pt[i+1];
        return s/2;
    }
};
PY py[500];
pair<double,int> c[5000];
inline double segP(Pt &p,Pt &p1,Pt &p2){
    if(SG(p1.x-p2.x)==0) return (p.y-p1.y)/(p2.y-p1.y);
    return (p.x-p1.x)/(p2.x-p1.x);
}
double polyUnion(int n){
    int i,j,ii,jj,ta,tb,r,d;
    double z,w,s,sum,tc,td;
    for(i=0;i<n;i++) py[i][py[i].n]=py[i][0];
    sum=0;
    for(i=0;i<n;i++){
        for(ii=0;ii<py[i].n;ii++){
            r=0;
            c[r++]=make_pair(0.0,0.0);
            c[r++]=make_pair(1.0,0.0);
            for(j=0;j<n;j++){
                if(i==j) continue;
                for(jj=0;jj<py[j].n;jj++){
                    ta=SG(tri(py[i][ii],py[i][ii+1],py[j][jj]));
                    tb=SG(tri(py[i][ii],py[i][ii+1],py[j][jj+1]));
                    if(ta==0 && tb==0){
                        if((py[j][jj+1]-py[i][jj])*(py[i][ii+1]-py[i][ii])>0 && j<i){
                            c[r++]=make_pair(segP(py[j][jj],py[i][ii],py[i][ii+1]),1);
                            c[r++]=make_pair(segP(py[j][jj+1],py[i][ii+1],py[i][ii]),-1);
                        }
                    } else if(ta>0 && tb<0){
                        tc=tri(py[j][jj],py[j][jj+1],py[i][ii]);
                        td=tri(py[j][jj],py[j][jj+1],py[i][ii+1]);
                        c[r++]=make_pair(tc/(tc-td),1);
                    } else if(ta<0 && tb>0){
                        tc=tri(py[j][jj],py[j][jj+1],py[i][ii]);
                        td=tri(py[j][jj],py[j][jj+1],py[i][ii+1]);
                        c[r++]=make_pair(tc/(tc-td),-1);
                    }
                }
            }
        }
        sort(c,c+r);
        z=min(max(c[0].first,0.0),1.0);
        d=c[0].second; s=0;
        for(j=1;j<r;j++){
            w=min(max(c[j].first,0.0),1.0);
            if(!d) s+=w-z;
            d+=c[j].second; z=w;
        }
        sum+=(py[i][ii]^py[i][ii+1])*s;
    }
    return sum/2;
}
int main(){
    int n,i,j,k;
    double sum,ds;
    scanf("%d",&n); sum=0;
    for(i=0;i<n;i++){
```

```

py[i].input();
ds=py[i].getArea();
if(ds<0){
    for(j=0,k=py[i].n-1;j<k;j++,k--) swap(py[i][j],
        py[i][k]);
    ds=-ds;
} sum+=ds;
} printf("%.9f\n",sum/polyUnion(n));
}

```

4.7 Lower Concave Hull

```

/****
maintain a "concave hull" that support the following
1. insertion of a line
2. query of height(y) on specific x on the hull
****/
/* set as needed */
typedef long double LD;
const LD eps=1e-9;
const LD inf=1e19;
class Seg {
public:
    LD m,c,x1,x2; // y=mx+c
    bool flag;
    Seg(
        LD _m,LD _c,LD _x1=-inf,LD _x2=inf,bool _flag=0)
        :m(_m),c(_c),x1(_x1),x2(_x2),flag(_flag) {}
    LD evaly(LD x) const {
        return m*x+c;
    }
    const bool operator<(LD x) const {
        return x2-eps<x;
    }
    const bool operator<(const Seg &b) const {
        if(flag!b.flag) return *this<b.x1;
        return m+eps<b.m;
    }
};
class LowerConcaveHull { // maintain a hull like: \_/_/
public:
    set<Seg> hull;
    /* functions */
    LD xintersection(Seg a,Seg b) {
        return (a.c-b.c)/(b.m-a.m);
    }
    inline set<Seg>::iterator replace(set<Seg> &
        hull,set<Seg>::iterator it,Seg s) {
        hull.erase(it);
        return hull.insert(s).first;
    }
    void insert(Seg s) {
        // insert a line and update hull
        set<Seg>::iterator it=hull.find(s);
        // check for same slope
        if(it!=hull.end()) {
            if(it->c+eps>=s.c) return;
            hull.erase(it);
        }
        // check if below whole hull
        it=hull.lower_bound(s);
        if(it!=hull.end()&&
            s.evaly(it->x1)<=it->evaly(it->x1)+eps) return;
        // update right hull
        while(it!=hull.end()) {
            LD x=xintersection(s,*it);
            if(x>=it->x2-eps) hull.erase(it++);
            else {
                s.x2=x;
                it=replace(hull,it,Seg(it->m,it->c,x,it->x2));
                break;
            }
        }
        // update left hull
        while(it!=hull.begin()) {
            LD x=xintersection(s,*(--it));
            if(x<=it->x1+eps) hull.erase(it++);
            else {
                s.x1=x;
                it=replace(hull,it,Seg(it->m,it->c,it->x1,x));
            }
        }
    }
};

```

```

        break;
    }
}
// insert s
hull.insert(s);
}
void insert(LD m,LD c) { insert(Seg(m,c)); }
LD query(LD x) { // return y @ given x
    set<Seg>::iterator it =
        hull.lower_bound(Seg(0.0,0.0,x,x,1));
    return it->evaly(x);
}
};

```

4.8 Min Enclosing Circle

```

struct Mec{
    // return pair of center and r
    static const int N = 101010;
    int n;
    Pt p[ N ], cen;
    double r2;
    void init( int _n , Pt _p[] ){
        n = _n;
        memcpy( p , _p , sizeof(Pt) * n );
    }
    double sqr(double a){ return a*a; }
    Pt center(Pt p0, Pt p1, Pt p2) {
        Pt a = p1-p0;
        Pt b = p2-p0;
        double c1=norm2( a ) * 0.5;
        double c2=norm2( b ) * 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 Pt(x,y);
    }
    pair<Pt,double> solve(){
        random_shuffle(p,p+n);
        r2=0;
        for (int i=0; i<n; i++){
            if (norm2(cen-p[i]) <= r2) continue;
            cen = p[i];
            r2 = 0;
            for (int j=0; j<i; j++){
                if (norm2(cen-p[j]) <= r2) continue;
                cen = Pt((p[i].X+p[j].X)*0.5, (p[i].Y+p[j].Y)
                    *0.5);
                r2 = norm2(cen-p[j]);
                for (int k=0; k<j; k++){
                    if (norm2(cen-p[k]) <= r2) continue;
                    cen = center(p[i],p[j],p[k]);
                    r2 = norm2(cen-p[k]);
                }
            }
        }
        return {cen,sqrt(r2)};
    }
} mec;

```

4.9 Minkowski sum

```

/* convex hull Minkowski Sum*/
#define INF 100000000000000LL
int pos( const Pt& tp ){
    if( tp.Y == 0 ) return tp.X > 0 ? 0 : 1;
    return tp.Y > 0 ? 0 : 1;
}
#define N 300030
Pt pt[ N ], qt[ N ], rt[ N ];
LL Lx,Rx;
int dn,un;
inline bool cmp( Pt a, Pt b ){
    int pa=pos( a ),pb=pos( b );
    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;
    LL 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("%lld%lld",&pt[i].X,&pt[i].Y);
    scanf("%d",&m);
    for(i=0;i<m;i++) scanf("%lld%lld",&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("%lld%lld",&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("%lld %lld",&p.X,&p.Y);
        p.X*=3; p.Y*=3;
        puts(inConvex(p)?"YES":"NO");
    }
}

```

```

}
}

```

4.10 Min/Max Enclosing Rectangle

```

/***** NEED REVISION *****/
/* uva819 - gifts large and small */
#define MAXN 100005
const double eps=1e-8;
const double inf=1e15;
class Coor {
public:
    double x,y;
    Coor() {}
    Coor(double xi,double yi) { x=xi; y=yi; }
    Coor& operator+=(const Coor &b) { x+=b.x; y+=b.y;
        return *this; }
    const Coor operator+(const Coor &b) const { return (
        Coor)*this+=b; }
    Coor& operator-=(const Coor &b) { x-=b.x; y-=b.y;
        return *this; }
    const Coor operator-(const Coor &b) const { return (
        Coor)*this-=b; }
    Coor& operator*=(const double b) { x*=b; y*=b; return
        *this; }
    const Coor operator*(const double b) const { return (
        Coor)*this*=b; }
    Coor& operator/=(const double b) { x/=b; y/=b; return
        *this; }
    const Coor operator/(const double b) const { return (
        Coor)*this/=b; }
    const bool operator<(const Coor& b) const { return y<
        b.y-eps || fabs(y-b.y)<eps&&x<b.x; }
    const double len2() const { return x*x+y*y; }
    const double len() const { return sqrt(len2()); }
    const Coor perp() const { return Coor(y,-x); }
    Coor& standardize() {
        if(y<0 || y==0&&x<0) {
            x=-x;
            y=-y;
        }
        return *this;
    }
    const Coor standardize() const { return ((Coor)*this)
        .standardize(); }
};

double dot(const Coor &a,const Coor &b) { return a.x*b.
    x+a.y*b.y; }
double dot(const Coor &o,const Coor &a,const Coor &b) {
    return dot(a-o,b-o); }
double cross(const Coor &a,const Coor &b) { return a.x*
    b.y-a.y*b.x; }
double cross(const Coor &o,const Coor &a,const Coor &b)
    { return cross(a-o,b-o); }

Coor cmpo;
const bool cmpf(const Coor &a,const Coor &b) {
    return cross(cmpo,a,b)>eps || fabs(cross(cmpo,a,b))<eps
        &&
        dot(a,cmpo,b)<eps;
}

class Polygon {
public:
    int pn;
    Coor p[MAXN];
    void convex_hull() {
        int i,tn=pn;
        for(i=1;i<pn;++i) if(p[i]<p[0]) swap(p[0],p[i]);
        cmpo=p[0];
        std::sort(p+1,p+pn,cmpf);
        for(i=pn=1;i<tn;++i) {
            while(pn>2&&cross(p[pn-2],p[pn-1],p[i])<=eps) --
                pn;
            p[pn++]=p[i];
        }
        p[pn]=p[0];
    }
};

Polygon pol;
double minarea,maxarea;
int slpn;

```

```

Coor slope[MAXN*2];
Coor lrec[MAXN*2], rrec[MAXN*2], trec[MAXN*2], brec[MAXN*2];
inline double xproject(Coor p, Coor slp) { return dot(p, slp)/slp.len(); }
inline double yproject(Coor p, Coor slp) { return cross(p, slp)/slp.len(); }
inline double calcarea(Coor lp, Coor rp, Coor bp, Coor tp, Coor slp) {
    return (xproject(rp, slp)-xproject(lp, slp))*(yproject(tp, slp)-yproject(bp, slp)); }
inline void solve() {
    int i, lind, rind, tind, bind, tn;
    double pro, area1, area2, l, r, m1, m2;
    Coor s1, s2;
    pol.convex_hull();
    slpn=0; /* generate all critical slope */
    slope[slpn]=Coor(1.0, 0.0);
    slope[slpn]=Coor(0.0, 1.0);
    for(i=0; i<pol.pn; i++) {
        slope[slpn]=(pol.p[i+1]-pol.p[i]).standardize();
        if(slope[slpn].x>0) slpn++;
        slope[slpn]=(pol.p[i+1]-pol.p[i]).perp().standardize();
        if(slope[slpn].x>0) slpn++;
    }
    cmpo=Coor(0,0);
    std::sort(slope, slope+slpn, cmpf);
    tn=slpn;
    for(i=slpn-1; i>0; i--) {
        if(cross(cmpo, slope[i], slope[i+1])>0) slope[slpn]=slope[i];
    }
    lind=rind=0; /* find critical touchpoints */
    for(i=0; i<pol.pn; i++) {
        pro=xproject(pol.p[i], slope[0]);
        if(pro>xproject(pol.p[lind], slope[0])) lind=i;
        if(pro>xproject(pol.p[rind], slope[0])) rind=i;
    }
    tind=bind=0;
    for(i=0; i<pol.pn; i++) {
        pro=yproject(pol.p[i], slope[0]);
        if(pro>yproject(pol.p[bind], slope[0])) bind=i;
        if(pro>yproject(pol.p[tind], slope[0])) tind=i;
    }
    for(i=0; i<slpn; i++) {
        while(xproject(pol.p[lind+1], slope[i])<=xproject(pol.p[lind], slope[i])+eps)
            lind=(lind==pol.pn-1?0:lind+1);
        while(xproject(pol.p[rind+1], slope[i])>=xproject(pol.p[rind], slope[i])-eps)
            rind=(rind==pol.pn-1?0:rind+1);
        while(yproject(pol.p[bind+1], slope[i])<=yproject(pol.p[bind], slope[i])+eps)
            bind=(bind==pol.pn-1?0:bind+1);
        while(yproject(pol.p[tind+1], slope[i])>=yproject(pol.p[tind], slope[i])-eps)
            tind=(tind==pol.pn-1?0:tind+1);
        lrec[i]=pol.p[lind];
        rrec[i]=pol.p[rind];
        brec[i]=pol.p[bind];
        trec[i]=pol.p[tind];
    }
    minarea=inf; /* find minimum area */
    for(i=0; i<slpn; i++) {
        area1=calcarea(lrec[i], rrec[i], brec[i], trec[i], slope[i]);
        if(area1<minarea) minarea=area1;
    }
    maxarea=minarea; /* find maximum area */
    for(i=0; i<slpn-1; i++) {
        l=0.0; r=1.0;
        while(l<r-eps) {
            m1=l+(r-l)/3;
            m2=l+(r-l)*2/3;
            s1=slope[i]*(1.0-m1)+slope[i+1]*m1;
            area1=calcarea(lrec[i], rrec[i], brec[i], trec[i], s1);
            s2=slope[i]*(1.0-m2)+slope[i+1]*m2;
            area2=calcarea(lrec[i], rrec[i], brec[i], trec[i], s2);
            if(area1<area2) l=m1;
            else r=m2;
        }
    }
}

```

```

}
s1=slope[i]*(1.0-l)+slope[i+1]*l;
area1=calcarea(lrec[i], rrec[i], brec[i], trec[i], s1);
if(area1>maxarea) maxarea=area1;
}
}
int main() {
    int i, casenum=1;
    while(scanf("%d", &pol.pn)==1 && pol.pn) {
        for(i=0; i<pol.pn; i++)
            scanf("%lf %lf", &pol.p[i].x, &pol.p[i].y);
        solve();
        //minarea, maxarea
    }
}

```

5 Graph

5.1 HeavyLightDecomp

```

#define SZ(c) (int)(c).size()
#define ALL(c) (c).begin(), (c).end()
#define REP(i, s, e) for(int i = (s); i <= (e); i++)
#define REPD(i, s, e) for(int i = (s); i >= (e); i--)
typedef tuple<int, int> tii;
const int MAXN = 100010;
const int LOG = 19;
struct HLD {
    int n;
    vector<int> g[MAXN];
    int sz[MAXN], dep[MAXN];
    int ts, tid[MAXN], tdi[MAXN], tl[MAXN], tr[MAXN];
    // ts : timestamp, useless after yutruli
    // tid[ u ] : pos. of node u in the seq.
    // tdi[ i ] : node at pos i of the seq.
    // tl, tr[ u ] : subtree interval in the seq. of node u
    int mom[MAXN][LOG], head[MAXN];
    // head[ u ] : head of the chain contains u
    void dfsz(int u, int p) {
        dep[u] = dep[p] + 1;
        mom[u][0] = p;
        sz[u] = 1;
        head[u] = u;
        for(int& v: g[u]) if(v != p) {
            dep[v] = dep[u] + 1;
            dfsz(v, u);
            sz[u] += sz[v];
        }
    }
    void dfsH(int u) {
        //printf("dfsH %d\n", u);
        ts++;
        tid[u] = tl[u] = tr[u] = ts;
        tdi[tid[u]] = u;
        sort(ALL(g[u]), [&](int a, int b) { return sz[a] > sz[b]; });
        bool flag = 1;
        for(int& v: g[u]) if(v != mom[u][0]) {
            if(flag) head[v] = head[u], flag = 0;
            dfsH(v);
            tr[u] = tr[v];
        }
    }
    inline int lca(int a, int b) {
        if(dep[a] > dep[b]) swap(a, b);
        //printf("lca %d %d\n", a, b);
        int diff = dep[b] - dep[a];
        REPD(k, LOG-1, 0) if(diff & (1<<k)) {
            //printf("b %d\n", mom[b][k]);
            b = mom[b][k];
        }
        if(a == b) return a;
        REPD(k, LOG-1, 0) if(mom[a][k] != mom[b][k]) {
            a = mom[a][k];
            b = mom[b][k];
        }
    }
}

```

```

    return mom[a][0];
}
void init( int _n ){
    n = _n;
    REP( i , 1 , n ) g[ i ].clear();
}
void addEdge( int u , int v ){
    g[ u ].push_back( v );
    g[ v ].push_back( u );
}
void yutruli(){
    dfssz(1, 0);
    ts = 0;
    dfshl(1);
    REP(k, 1, LOG-1) REP(i, 1, n)
        mom[i][k] = mom[mom[i][k-1]][k-1];
}
vector< tii > getPath( int u , int v ){
    vector< tii > res;
    while( tid[ u ] < tid[ head[ v ] ] ){
        res.push_back( tii(tid[ head[ v ] ] , tid[ v ] ) );
        v = mom[ head[ v ] ][ 0 ];
    }
    res.push_back( tii( tid[ u ] , tid[ v ] ) );
    reverse( ALL( res ) );
    return res;
}
/*
 * res : list of intervals from u to v
 * u must be ancestor of v
 * usage :
 * vector< tii > path = tree.getPath( u , v )
 * for( tii tp : path ) {
 *     int l , r; tie( l , r ) = tp;
 *     upd( l , r );
 *     uu = tree.tdi[ l ] , vv = tree.tdi[ r ];
 *     uu ~> vv is a heavy path on tree
 * }
 */
}
} tree;

```

5.2 DominatorTree

```

const int MAXN = 100010;
struct DominatorTree{
#define REP(i,s,e) for(int i=(s);i<=(e);i++)
#define REPD(i,s,e) for(int i=(s);i>=(e);i--)
    int n , m , s;
    vector< int > g[ MAXN ] , pred[ MAXN ];
    vector< int > cov[ MAXN ];
    int dfn[ MAXN ] , nfd[ MAXN ] , ts;
    int par[ MAXN ];
    int sdom[ MAXN ] , idom[ MAXN ];
    int mom[ MAXN ] , mn[ MAXN ];

    inline bool cmp( int u , int v )
    { return dfn[ u ] < dfn[ v ]; }

    int eval( int u ){
        if( mom[ u ] == u ) return u;
        int res = eval( mom[ u ] );
        if(cmp( sdom[ mn[ mom[ u ] ] ] , sdom[ mn[ u ] ] ))
            mn[ u ] = mn[ mom[ u ] ];
        return mom[ u ] = res;
    }

    void init( int _n , int _m , int _s ){
        ts = 0; n = _n; m = _m; s = _s;
        REP( i , 1 , n ) g[ i ].clear(), pred[ i ].clear();
    }
    void addEdge( int u , int v ){
        g[ u ].push_back( v );
        pred[ v ].push_back( u );
    }
    void dfs( int u ){
        ts++;
        dfn[ u ] = ts;
        nfd[ ts ] = u;
        for( int v : g[ u ] ) if( dfn[ v ] == 0 ){

```

```

            par[ v ] = u;
            dfs( v );
        }
    }
    void build(){
        REP( i , 1 , n ){
            dfn[ i ] = nfd[ i ] = 0;
            cov[ i ].clear();
            mom[ i ] = mn[ i ] = sdom[ i ] = i;
        }
        dfs( s );
        REPD( i , n , 2 ){
            int u = nfd[ i ];
            if( u == 0 ) continue;
            for( int v : pred[ u ] ) if( dfn[ v ] ){
                eval( v );
                if( cmp( sdom[ mn[ v ] ] , sdom[ u ] ) )
                    sdom[ u ] = sdom[ mn[ v ] ];
            }
            cov[ sdom[ u ] ].push_back( u );
            mom[ u ] = par[ u ];
            for( int w : cov[ par[ u ] ] ){
                eval( w );
                if( cmp( sdom[ mn[ w ] ] , par[ u ] ) )
                    idom[ w ] = mn[ w ];
                else idom[ w ] = par[ u ];
            }
            cov[ par[ u ] ].clear();
        }
        REP( i , 2 , n ){
            int u = nfd[ i ];
            if( u == 0 ) continue;
            if( idom[ u ] != sdom[ u ] )
                idom[ u ] = idom[ idom[ u ] ];
        }
    }
} domT;

```

5.3 MaxClique

```

class MaxClique {
public:
    static const int MV = 210;
    int V , ans;
    int el[MV][MV/30+1];
    int dp[MV];
    int s[MV][MV/30+1];
    vector<int> sol;
    void init(int v) {
        V = v; ans = 0;
        FZ(el); FZ(dp);
    }
    /* Zero Base */
    void addEdge(int u, int v) {
        if(u > v) swap(u, v);
        if(u == v) return;
        el[u][v/32] |= (1<<(v%32));
    }
    bool dfs(int v, int k) {
        int c = 0, d = 0;
        for(int i=0; i<(V+31)/32; i++) {
            s[k][i] = el[v][i];
            if(k != 1) s[k][i] &= s[k-1][i];
            c += __builtin_popcount(s[k][i]);
        }
        if(c == 0) {
            if(k > ans) {
                ans = k;
                sol.clear();
                sol.push_back(v);
                return 1;
            }
            return 0;
        }
        for(int i=0; i<(V+31)/32; i++) {
            for(int a = s[k][i]; a; d++) {
                if(k + (c-d) <= ans) return 0;
                int lb = a&(-a), lg = 0;
                a ^= lb;
                while(lb!=1) {

```

```

        lb = (unsigned int)(lb) >> 1;
        lg ++;
    }
    int u = i*32 + lg;
    if(k + dp[u] <= ans) return 0;
    if(dfs(u, k+1)) {
        sol.push_back(v);
        return 1;
    }
}
return 0;
}
int solve() {
    for(int i=V-1; i>=0; i--) {
        dfs(i, 1);
        dp[i] = ans;
    }
    return ans;
}
};

```

5.4 Strongly Connected Component

```

struct Scc{
    int n, nScc, vst[MXN], bln[MXN];
    vector<int> E[MXN], rE[MXN], vec;
    void init(int _n){
        n = _n;
        for (int i=0; i<MXN; 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 (auto v : E[u])
            if (!vst[v]) DFS(v);
        vec.PB(u);
    }
    void rDFS(int u){
        vst[u] = 1;
        bln[u] = nScc;
        for (auto v : rE[u])
            if (!vst[v]) rDFS(v);
    }
    void solve(){
        nScc = 0;
        vec.clear();
        FZ(vst);
        for (int i=0; i<n; i++)
            if (!vst[i]) DFS(i);
        reverse(vec.begin(),vec.end());
        FZ(vst);
        for (auto v : vec){
            if (!vst[v]){
                rDFS(v);
                nScc++;
            }
        }
    }
};

```

5.5 Minimum General Weighted Matching

```

struct Graph {
    // Minimum General Weighted Matching (Perfect Match)
    static const int MXN = 105;
    int n, edge[MXN][MXN];
    int match[MXN], dis[MXN], onstk[MXN];
    vector<int> stk;
    void init(int _n) {
        n = _n;
        for( int i = 0 ; i < n ; i ++ )

```

```

        for( int j = 0 ; j < n ; j ++ )
            edge[ i ][ j ] = 0;
    }
    void add_edge(int u, int v, int w) {
        edge[u][v] = edge[v][u] = w;
    }
    bool SPFA(int u){
        if (onstk[u]) return true;
        stk.PB(u);
        onstk[u] = 1;
        for (int v=0; v<n; v++){
            if (u != v && match[u] != v && !onstk[v]){
                int m = match[v];
                if (dis[m] > dis[u] - edge[v][m] + edge[u][v]){
                    dis[m] = dis[u] - edge[v][m] + edge[u][v];
                    onstk[v] = 1;
                    stk.PB(v);
                    if (SPFA(m)) return true;
                    stk.pop_back();
                    onstk[v] = 0;
                }
            }
        }
        onstk[u] = 0;
        stk.pop_back();
        return false;
    }
    int solve() {
        // find a match
        for (int i=0; i<n; i+=2){
            match[i] = i+1;
            match[i+1] = i;
        }
        while (true){
            int found = 0;
            for( int i = 0 ; i < n ; i ++ )
                onstk[ i ] = dis[ i ] = 0;
            for (int i=0; i<n; i++){
                stk.clear();
                if (!onstk[i] && SPFA(i)){
                    found = 1;
                    while (SZ(stk)>=2){
                        int u = stk.back(); stk.pop_back();
                        int v = stk.back(); stk.pop_back();
                        match[u] = v;
                        match[v] = u;
                    }
                }
            }
            if (!found) break;
        }
        int ret = 0;
        for (int i=0; i<n; i++)
            ret += edge[i][match[i]];
        ret /= 2;
        return ret;
    }
}graph;

```

6 String

6.1 PalTree

```
const int MAXN = 200010;
struct PalT{
    struct Node{
        int nxt[ 33 ] , len , fail;
        ll cnt;
    };
    int tot , lst;
    Node nd[ MAXN * 2 ];
    char* s;
    int newNode( int l , int _fail ){
        int res = ++tot;
        memset( nd[ res ].nxt , 0 , sizeof nd[ res ].nxt );
        nd[ res ].len = l;
        nd[ res ].cnt = 0;
        nd[ res ].fail = _fail;
        return res;
    }
    void push( int p ){
        int np = lst;
        int c = s[ p ] - 'a';
        while( p - nd[ np ].len - 1 < 0
            || s[ p ] != s[ p - nd[ np ].len - 1 ] )
            np = nd[ np ].fail;

        if( nd[ np ].nxt[ c ] ){
            nd[ nd[ np ].nxt[ c ] ].cnt++;
            lst = nd[ np ].nxt[ c ];
            return ;
        }
        int nq = newNode( nd[ np ].len + 2 , 0 );
        nd[ nq ].cnt++;
        nd[ np ].nxt[ c ] = nq;
        lst = nq;
        if( nd[ nq ].len == 1 ){
            nd[ nq ].fail = 2;
            return ;
        }
        int tf = nd[ np ].fail;
        while( p - nd[ tf ].len - 1 < 0
            || s[ p ] != s[ p - nd[ tf ].len - 1 ] )
            tf = nd[ tf ].fail;

        nd[ nq ].fail = nd[ tf ].nxt[ c ];
        return ;
    }
    void init( char* _s ){
        s = _s;
        tot = 0;
        newNode( -1 , 1 );
        newNode( 0 , 1 );
        lst = 2;
        for( int i = 0 ; s[ i ] ; i++ )
            push( i );
    }
    void yutuli(){
#define REPD(i, s, e) for(int i = (s); i >= (e); i--)
        REPD( i , tot , 1 )
            nd[ nd[ i ].fail ].cnt += nd[ i ].cnt;
        nd[ 1 ].cnt = nd[ 2 ].cnt = 0ll;
    }
} pA;
int main(){
    pA.init( sa );
}
```

6.2 SuffixArray

```
const int MAX = 1020304;
int ct[MAX], he[MAX], rk[MAX];
int 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]]+]=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[tsa[j]][0]]+]=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;
}
}
```

6.3 SAIS

```
const int N = 300010;
struct SA{
#define REP(i,n) for ( int i=0; i<int(n); i++ )
#define REP1(i,a,b) for ( int i=(a); i<=int(b); i++ )
    bool _t[N*2];
    int _s[N*2], _sa[N*2], _c[N*2], x[N], _p[N], _q[N*2],
        hei[N], r[N];
    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[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]]);
}
}sa;
int H[ N ], SA[ N ];
void suffix_array(int* ip, int len) {
    // should padding a zero in the back
    // ip is int array, len is array length
    // ip[0..n-1] != 0, and ip[len] = 0
    ip[len++] = 0;
    sa.build(ip, len, 128);
    for (int i=0; i<len; i++) {
        H[i] = sa.hei[i + 1];
        SA[i] = sa._sa[i + 1];
    }
    // resulting height, sa array \in [0,len)
}

```

6.4 SuffixAutomata

```

const int MAXM = 1000010;
struct SAM{
    int tot, root, lst, mom[MAXM], mx[MAXM];
    int acc[MAXM], nxt[MAXM][33];
    int newNode(){
        int res = ++tot;
        fill(nxt[res], nxt[res]+33, 0);
        mom[res] = mx[res] = acc[res] = 0;
        return res;
    }
    void init(){
        tot = 0;
        root = newNode();
        mom[root] = 0, mx[root] = 0;
        lst = root;
    }
    void push(int c){
        int p = lst;
        int np = newNode();
        mx[np] = mx[p]+1;
        for(; p && nxt[p][c] == 0; p = mom[p])
            nxt[p][c] = np;
        if(p == 0) mom[np] = root;
        else{
            int q = nxt[p][c];
            if(mx[p]+1 == mx[q]) mom[np] = q;
            else{
                int nq = newNode();
                mx[nq] = mx[p]+1;
                for(int i = 0; i < 33; i++)
                    nxt[nq][i] = nxt[q][i];
                mom[nq] = mom[q];
                mom[q] = nq;
                mom[np] = nq;
                for(; p && nxt[p][c] == q; p = mom[p])
                    nxt[p][c] = nq;
            }
        }
        lst = np;
    }
    void print(){
        REP(i, 1, tot){
            printf("node %d : \n", i);

```

```

printf("mx %d, mom %d\n", mx[i], mom[i]);
REP(j, 1, 26) if(nxt[i][j])
    printf("nxt %c %d\n", 'a'+j-1, nxt[i][j]);
puts("-----");
}
}
void push(char *str){
    for(int i = 0; str[i]; i++)
        push(str[i]-'a'+1);
}
} sam;

```

6.5 Aho-Corasick

```

struct AAutomata{
    struct Node{
        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]);
                }
            }
        }
    }
};

```

6.6 Z Value

```

char s[MAXN];
int len,z[MAXN];
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;
    }
}
}

```

6.7 ZValue Palindrome

```

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

```

6.8 Smallest Rotation

```

string mcp(string s){
    int n = s.length();
    s += s;
    int i=0, j=1, k=0;
    while (j<n && k<n){
        if (s[i+k] == s[j+k]) k++;
        else {
            if (s[i+k] < s[j+k]) {
                j += k + 1;
            } else {
                i = j;
                j = max(j+1, j+k);
            }
            k = 0;
        }
    }
    return s.substr(i, n);
}

```

6.9 Baker Bird

```

class Node { public:
    Node *fail;
    map<char,Node*> _next;
    int out;
    Node() { fail=NULL; out=-1; }
    ~Node() {

```

```

        for(map<char,Node*>::iterator it=_next.begin();it!=
            _next.end();it++)
            delete it->second;
    }
    Node* build(char ch) {
        if(_next.find(ch)==_next.end()) _next[ch]=new Node;
        return _next[ch];
    }
    Node* next(char ch) {
        if(_next.find(ch)==_next.end()) return NULL;
        return _next[ch];
    }
};
int srn,scn,prn,pcn,mrn,mcn;
char s[MAXN][MAXN],p[MAXN][MAXN];
int rm[MAXN][MAXN]; // rank matrix
int maxrank;
int seq[MAXN]; // index of patterns for radix sort
int rank[MAXN]; // rank of pattern on row r
int cnt[SIGMA+1],tmp[MAXN];
int pre[MAXN]; // pre-matrix for kmp
int ql,q;
Node* que[MAXN*MAXN];
inline void radix_pass(int j,int *from,int *to) {
    int i;
    for(i=0;i<SIGMA;i++) cnt[i]=0;
    for(i=0;i<prn;i++) cnt[p[from[i]][j]+1]++;
    for(i=0;i<SIGMA;i++) cnt[i+1]+=cnt[i];
    for(i=0;i<prn;i++) to[cnt[p[from[i]][j]]+1]=from[i];
}
inline void radix_sort_patterns() {
    int i,j;
    for(i=0;i<prn;i++) ((pcn&1)?tmp[i]:seq[i])=i;
    for(j=pcn-1;j>=0;j--) {
        if(j&1) radix_pass(j,seq,tmp);
        else radix_pass(j,tmp,seq);
    }
    maxrank=0;
    for(i=0;i<prn;i++) {
        if(i&&strcmp(p[seq[i-1]],p[seq[i]])) ++maxrank;
        rank[seq[i]]=maxrank;
    }
}
inline void construct(Node *v,char *p,int ind) {
    while(*p) { v=v->build(*p); p++; }
    v->out=ind;
}
inline void construct_all(Node *ac) {
    for(int i=0;i<prn;i++) construct(ac,p[i],rank[i]);
}
inline void find_fail(Node *ac) {
    Node *v,*u,*f;
    map<char,Node*>::iterator it;
    char ch;
    ql=q=0; ac->fail=ac;
    for(it=ac->_next.begin();it!=ac->_next.end();it++) {
        u=it->second;
        u->fail=ac;
        que[q++]=u;
    }
    while(ql<q) {
        v=que[ql++];
        for(it=v->_next.begin();it!=v->_next.end();it++) {
            ch=it->first; u=it->second;
            f=v->fail;
            while(f!=ac&&f->next(ch)==NULL) f=f->fail;
            if(f->next(ch)) u->fail=f->next(ch);
            else u->fail=ac;
            que[q++]=u;
        }
    }
}
inline void ac_match(Node *ac,char *s,int *arr) {
    int i;
    Node *v=ac;
    for(i=0;i<scn;i++) {
        while(v!=ac&&v->next(s[i])==NULL) v=v->fail;
        if(v->next(s[i])) v=v->next(s[i]);
        if(i==pcn-1) arr[i-pcn+1]=v->out;
    }
}
inline void find_rank_matrix() {

```

```

Node ac;
radix_sort_patterns();
construct_all(&ac);
find_fail(&ac);
mrn=srn; mcn=scn-pcn+1;
for(int i=0;i<srn;i++) ac_match(&ac,s[i],rm[i]);
}
inline void find_pre(int *p,int plen) {
int i,x;
x=pre[0]=-1;
for(i=1;i<plen;i++) {
while(x>=0&&p[x+1]!=p[i]) x=pre[x];
if(p[x+1]==p[i]) x++;
pre[i]=x;
}
}
inline int kmp_match(int col,int *p,int plen) {
int i,x=-1,occ=0;
for(i=0;i<mrn;i++) {
while(x>=0&&p[x+1]!=rm[i][col]) x=pre[x];
if(p[x+1]==rm[i][col]) x++;
if(x==plen-1) { occ++; x=pre[x]; }
}
return occ;
}
inline int baker_bird() {
int i,occ=0;
find_rank_matrix();
find_pre(rank,prn);
for(i=0;i<mcn;i++) occ+=kmp_match(i,rank,prn);
return occ;
}

```

```

strcpy(b,tmp);
}
strcpy(tmp,a);
strcat(a,tmp);
// basic lcs
for(int i=0;i<=2*al;i++) {
dp[i][0]=0;
pred[i][0]=U;
}
for(int j=0;j<=bl;j++) {
dp[0][j]=0;
pred[0][j]=L;
}
for(int i=1;i<=2*al;i++) {
for(int j=1;j<=bl;j++) {
if(a[i-1]==b[j-1]) dp[i][j]=dp[i-1][j-1]+1;
else dp[i][j]=max(dp[i-1][j],dp[i][j-1]);
if(dp[i][j-1]==dp[i][j]) pred[i][j]=L;
else if(a[i-1]==b[j-1]) pred[i][j]=LU;
else pred[i][j]=U;
}
}
// do cyclic lcs
int clcs=0;
for(int i=0;i<al;i++) {
clcs=max(clcs,lcs_length(i));
reroot(i+1);
}
// recover a
a[al]='\0';
return clcs;
}

```

6.10 Cyclic LCS

```

#define L 0
#define LU 1
#define U 2
const int mov[3][2]={0,-1, -1,-1, -1,0};
int al,bl;
char a[MAXL*2],b[MAXL*2]; // 0-indexed
int dp[MAXL*2][MAXL];
char pred[MAXL*2][MAXL];
inline int lcs_length(int r) {
int i=r+al,j=bl,l=0;
while(i>r) {
char dir=pred[i][j];
if(dir==LU) l++;
i+=mov[dir][0];
j+=mov[dir][1];
}
return l;
}
inline void reroot(int r) { // r = new base row
int i=r,j=1;
while(j<=bl&&pred[i][j]!=LU) j++;
if(j>bl) return;
pred[i][j]=L;
while(i<2*al&&j<=bl) {
if(pred[i+1][j]==U) {
i++;
pred[i][j]=L;
} else if(j<bl&&pred[i+1][j+1]==LU) {
i++;
j++;
pred[i][j]=L;
} else {
j++;
}
}
}
int cyclic_lcs() {
// a, b, al, bl should be properly filled
// note: a WILL be altered in process
// -- concatenated after itself
char tmp[MAXL];
if(al>bl) {
swap(al,bl);
strcpy(tmp,a);
strcpy(a,b);
}

```

7 Data Structure

7.1 Treap

```

struct Treap{
int sz , val , pri , tag;
Treap *l , *r;
Treap( int _val ){
val = _val; sz = 1;
pri = rand(); l = r = NULL; tag = 0;
}
};
void push( Treap * a ){
if( a->tag ){
Treap *swp = a->l; a->l = a->r; a->r = swp;
int swp2;
if( a->l ) a->l->tag ^= 1;
if( a->r ) a->r->tag ^= 1;
a->tag = 0;
}
}
int Size( Treap * a ){ return a ? a->sz : 0; }
void pull( Treap * a ){
a->sz = Size( a->l ) + Size( a->r ) + 1;
}
Treap* merge( Treap *a , Treap *b ){
if( !a || !b ) return a ? a : b;
if( a->pri > b->pri ){
push( a );
a->r = merge( a->r , b );
pull( a );
return a;
} else {
push( b );
b->l = merge( a , b->l );
pull( b );
return b;
}
}
void split( Treap *t , int k , Treap*&a , Treap*&b ){
if( !t ){ a = b = NULL; return; }
push( t );
if( Size( t->l ) + 1 <= k ){
a = t;
split( t->r , k - Size( t->l ) - 1 , a->r , b );
}
}

```

```

    pull( a );
} else {
    b = t;
    split( t->l , k , a , b->l );
    pull( b );
}
}

```

7.2 Link-Cut Tree

```

const int MXN = 100005;
const int MEM = 100005;
struct Splay {
    static Splay nil, mem[MEM], *pmem;
    Splay *ch[2], *f;
    int val, rev, size;
    Splay () : val(-1), rev(0), size(0) {
        f = ch[0] = ch[1] = &nil;
    }
    Splay (int _val) : val(_val), rev(0), size(1) {
        f = ch[0] = ch[1] = &nil;
    }
    bool isr() {
        return f->ch[0] != this && f->ch[1] != this;
    }
    int dir() {
        return f->ch[0] == this ? 0 : 1;
    }
    void setCh(Splay *c, int d) {
        ch[d] = c;
        if (c != &nil) c->f = this;
        pull();
    }
    void push() {
        if (rev) {
            swap(ch[0], ch[1]);
            if (ch[0] != &nil) ch[0]->rev ^= 1;
            if (ch[1] != &nil) ch[1]->rev ^= 1;
            rev = 0;
        }
    }
    void pull() {
        size = ch[0]->size + ch[1]->size + 1;
        if (ch[0] != &nil) ch[0]->f = this;
        if (ch[1] != &nil) ch[1]->f = this;
    }
} Splay::nil, Splay::mem[MEM], *Splay::pmem = Splay::mem;
Splay *nil = &Splay::nil;
void rotate(Splay *x) {
    Splay *p = x->f;
    int d = x->dir();
    if (!p->isr()) p->f->setCh(x, p->dir());
    else x->f = p->f;
    p->setCh(x->ch[d], d);
    x->setCh(p, !d);
    p->pull(); x->pull();
}

vector<Splay*> splayVec;
void splay(Splay *x) {
    splayVec.clear();
    for (Splay *q=x; q=q->f) {
        splayVec.push_back(q);
        if (q->isr()) break;
    }
    reverse(begin(splayVec), end(splayVec));
    for (auto it : splayVec) it->push();
    while (!x->isr()) {
        if (x->f->isr()) rotate(x);
        else if (x->dir()==x->f->dir())
            rotate(x->f), rotate(x);
        else rotate(x), rotate(x);
    }
}
Splay* access(Splay *x) {
    Splay *q = nil;
    for (; x!=nil; x=x->f) {
        splay(x);
        x->setCh(q, 1);
    }
}

```

```

    q = x;
}
return q;
}
void evert(Splay *x) {
    access(x);
    splay(x);
    x->rev ^= 1;
    x->push(); x->pull();
}
void link(Splay *x, Splay *y) {
    // evert(x);
    access(x);
    splay(x);
    evert(y);
    x->setCh(y, 1);
}
void cut(Splay *x, Splay *y) {
    // evert(x);
    access(y);
    splay(y);
    y->push();
    y->ch[0] = y->ch[0]->f = nil;
}
int N, Q;
Splay *vt[MXN];
int ask(Splay *x, Splay *y) {
    access(x);
    access(y);
    splay(x);
    int res = x->f->val;
    if (res == -1) res = x->val;
    return res;
}
int main(int argc, char** argv) {
    scanf("%d%d", &N, &Q);
    for (int i=1; i<=N; i++)
        vt[i] = new (Splay::pmem++) Splay(i);
    while (Q--) {
        char cmd[105];
        int u, v;
        scanf("%s", cmd);
        if (cmd[1] == 'i') {
            scanf("%d%d", &u, &v);
            link(vt[u], vt[v]);
        } else if (cmd[0] == 'c') {
            scanf("%d", &v);
            cut(vt[1], vt[v]);
        } else {
            scanf("%d%d", &u, &v);
            int res = ask(vt[u], vt[v]);
            printf("%d\n", res);
        }
    }
}

```

7.3 Disjoint Set

```

struct DisjointSet {
    // save() is like recursive
    // undo() is like return
    int n, fa[ N ], sz[ N ];
    vector< pair<int*,int> > h;
    vector<int> sp;
    void init( int tn ) {
        n=tn;
        for( int i = 0 ; i < n ; i ++ ) {
            fa[ i ] = i;
            sz[ i ] = 1;
        }
        sp.clear(); h.clear();
    }
    void assign( int *k, int v ) {
        h.PB( {k, *k} );
        *k = v;
    }
    void save() { sp.PB(SZ(h)); }
    void undo() {
        assert(!sp.empty());
        int last=sp.back(); sp.pop_back();
    }
}

```

```

while( SZ(h)!=last ){
    auto x=h.back(); h.pop_back();
    *x.first = x.second;
}
}
int f( int x ){
    while( fa[ x ] != x ) x = fa[ x ];
    return x;
}
void uni( int x , int y ){
    x = f( x ); y = f( y );
    if( x == y ) return;
    if( sz[ x ] < sz[ y ] ) swap( x , y );
    assign( &sz[ x ] , sz[ x ] + sz[ y ] );
    assign( &fa[ y ] , x );
}
}djs;

```

7.4 Pairing Heap

```

#include <bits/extc++.h>
using namespace __gnu_pbds;
typedef priority_queue<int> heap;
int main(){
    heap h1 , h2;
    h1.push( 1 );
    h2.push( 4 );
    h1.join( h2 );
    h1.size(); // 2
    h2.size(); // 0
    h1.top(); // 4
}

```

7.5 Leftist Heap

```

const int MAXN = 10000;
struct Node{
    int num,lc,rc;
    Node() : num(0), lc(-1), rc(-1){}
    Node( int _v ) : num(_v), lc(-1), rc(-1){}
}tree[ MAXN ];
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)
*/

```

7.6 Black Magic

```

#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);
}

```

8 Others

8.1 Find max 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 - l; 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+l]))
            now++;
        calc = sum[i+l] - pnt[now-1];
        if (ans.y * calc.x < ans.x * calc.y){
            ans = calc;
            st = pnt[now-1].x;
            ed = i + l;
        }
    }
    double res = (sum[ed].y - sum[st].y) /
        (sum[ed].x - sum[st].x);
    printf("%f\n", res);
    return 0;
}

```

8.2 Exact Cover Set

```

// given n*m 0-1 matrix
// find a set of rows s.t.
// for each column, there's exactly one 1
#include <stdio.h>
#include <string.h>
#define N 1024 //row
#define M 1024 //column
#define NM ((N+2)*(M+2))
char A[N][M]; //n*m 0-1 matrix
int used[N]; //answer: the row used
int id[N][M];
int L[NM],R[NM],D[NM],U[NM],C[NM],S[NM],ROW[NM];
void remove(int c){
    L[R[c]]=L[c]; R[L[c]]=R[c];
    for( int i=D[c]; i!=c; i=D[i] )
        for( int j=R[i]; j!=i; j=R[j] ){
            U[D[j]]=U[j]; D[U[j]]=D[j]; S[C[j]]--;

```

```

    }
}
void resume(int c){
    for( int i=D[c]; i!=c; i=D[i] )
        for( int j=L[i]; j!=i; j=L[j] ){
            U[D[j]]=D[U[j]]=j; S[C[j]]++;
        }
    L[R[c]]=R[L[c]]=c;
}
int dfs(){
    if(R[0]==0) return 1;
    int md=100000000,c;
    for( int i=R[0]; i!=0; i=R[i] )
        if(S[i]<md){ md=S[i]; c=i; }
    if(md==0) return 0;
    remove(c);
    for( int i=D[c]; i!=c; i=D[i] ){
        used[ROW[i]]=1;
        for( int j=R[i]; j!=i; j=R[j] ) remove(C[j]);
        if(dfs()) return 1;
        for( int j=L[i]; j!=i; j=L[j] ) resume(C[j]);
        used[ROW[i]]=0;
    }
    resume(c);
    return 0;
}
int exact_cover(int n,int m){
    for( int i=0; i<=m; i++ ){
        R[i]=i+1; L[i]=i-1; U[i]=D[i]=i;
        S[i]=0; C[i]=i;
    }
    R[m]=0; L[0]=m;
    int t=m+1;
    for( int i=0; i<n; i++ ){
        int k=-1;
        for( int j=0; j<m; j++ ){
            if(!A[i][j]) continue;
            if(k==-1) L[t]=R[t]=t;
            else{ L[t]=k; R[t]=R[k]; }
            k=t; D[t]=j+1; U[t]=U[j+1];
            L[R[t]]=R[L[t]]=U[D[t]]=D[U[t]]=t;
            C[t]=j+1; S[C[t]]++; ROW[t]=i; id[i][j]=t++;
        }
    }
    for( int i=0; i<n; i++ ) used[i]=0;
    return dfs();
}

```