

## Team notebook

BUET HellBent

## Contents

<b>1 DP</b>	<b>2</b>	3.12 Point Rotation Trick . . . . .	13	6.3 Hash Table . . . . .	21
1.1 1D-1D optimization . . . . .	2	3.13 Simpson . . . . .	14	6.4 Integration Table . . . . .	21
1.2 ConnctedComponentDP . . . . .	2	3.14 Visibility Polygon . . . . .	14	6.5 Snippets . . . . .	22
1.3 Convex Hull Trick Linear . . . . .	2	3.15 Voronoi . . . . .	14	<b>7 String</b>	<b>22</b>
1.4 Convex Hull Trick Online . . . . .	2	<b>4 Graph</b>	<b>15</b>	7.1 Aho Corasick . . . . .	22
1.5 Digit DP Template . . . . .	2	4.1 BlockCutTree . . . . .	15	7.2 KMP . . . . .	22
1.6 Knuth Optimisation . . . . .	2	4.2 Blossom . . . . .	15	7.3 Manacher . . . . .	23
1.7 SOS DP . . . . .	3	4.3 Directed MST . . . . .	15	7.4 Palindromic Tree . . . . .	23
1.8 SOS(on the fly) . . . . .	3	4.4 Dominator Tree . . . . .	16	7.5 Suffix Array . . . . .	23
<b>2 Data Structures</b>	<b>3</b>	4.5 EulerPath . . . . .	16	7.6 Suffix Automata . . . . .	23
2.1 2D BIT Range update Range query . . .	3	4.6 Flow with demands . . . . .	16	7.7 Z Algorithm . . . . .	23
2.2 Centroid Decomposition . . . . .	3	4.7 Hopcroft Karp . . . . .	17		
2.3 HLD . . . . .	3	4.8 Hungarian Algorithm . . . . .	17		
2.4 Li Chao Tree . . . . .	4	4.9 Maxflow . . . . .	17		
2.5 Persistent Segment Tree . . . . .	4	4.10 Mincost Maxflow . . . . .	18		
2.6 RMQ(2D) . . . . .	4	4.11 SCC + 2SAT . . . . .	18		
2.7 SegTree Range Inc, Max Query . . . . .	4	<b>5 Math</b>	<b>18</b>		
<b>3 Geometry</b>	<b>4</b>	5.1 FFT . . . . .	18		
3.1 Circle Cover . . . . .	4	5.2 FWHT . . . . .	19		
3.2 Circle Polygon Common . . . . .	5	5.3 Gaussian Elimination . . . . .	19		
3.3 Circle Union Area . . . . .	5	5.4 Linear sieve . . . . .	19		
3.4 Geometry 2D Basic . . . . .	5	5.5 NTT . . . . .	19		
3.5 Geometry 2D Polygon . . . . .	7	5.6 Number Theory . . . . .	19		
3.6 Geometry 3D Basic . . . . .	8	5.7 Pollard Rho . . . . .	20		
3.7 Geometry 3D Convex Hull . . . . .	11	5.8 Prime Counting Function . . . . .	20		
3.8 Geometry Hull . . . . .	11	5.9 Primitive Root . . . . .	21		
3.9 Line Polygon Intersection . . . . .	13	5.10 Stern Brocot Tree . . . . .	21		
3.10 Min Enclosing Circle . . . . .	13	5.11 Sum of Floors . . . . .	21		
3.11 Minkowski Sum . . . . .	13	<b>6 Miscellaneous</b>	<b>21</b>		
		6.1 Fast IO C++ . . . . .	21		
		6.2 Fast IO JAVA . . . . .	21		

# 1 DP

## 1.1 1D-1D optimization

```
//Batch Scheduling
int dp[maxn],t[maxn],f[maxn],n,s;
int w(int i,int x){
    return(t[x]-t[i]+s)*(f[n]-f[i]);
}
int main(){
    scanf("%d %d",&n,&s);
    for(int i=1 ; i<=n ; i++){
        scanf("%d %d",&t[i],&f[i]);
        t[i] += t[i-1] ; f[i] += f[i-1] ;
    }
    vector < pair<int,int> > vp ; // pos , best-k
    vp.pb( mp( 0 , 0 ) ) ;
    for(int x=1 ; x<=n ; x++){
        int idx=upper_bound(vp.begin(),
            vp.end(),mp(x,n+1)) - vp.begin() ;
        idx-- ;
        dp[x]=dp[vp[idx].yy]+w(vp[idx].yy,x);
        while( (int)vp.size() > 0 ){
            if(vp.back().xx>x&&dp[x]+w(x,vp.back().xx)<=
                dp[vp.back().yy]+w(vp.back().yy,
                    vp.back().xx))vp.pop_back();
            else break ;
        }
        if(vp.size()==0) vp.push_back(mp(0,x));
        else{
            int lo = max(vp.back().xx,x+1),hi=n;
            if(lo>hi||dp[vp.back().yy]+
                w(vp.back().yy,hi)<=dp[x]+w(x,hi))continue;
            while( lo < hi ){
                int mid = (lo+hi)/2 ;
                if( dp[vp.back().yy]+w(vp.back().yy,mid)<=
                    dp[x]+w(x,mid))lo=mid+1;
                else hi=mid;
            }
            vp.pb( mp( lo , x ) ) ;
        }
    }
    printf("%d\n",dp[n]);
}
```

## 1.2 ConnctedComponentDP

```
i64 f(int n,int r,int k,int c,int st,int en) {
    if(n==0||c<0||st<0||en<0) return 0 ;
    r = (r+a[n]*(2*c+st+en))%M ;
    if(dp[n][r][k][c][st][en]!=-1)
        return dp[n][r][k][c][st][en];
    i64 ans = f(n-1,r,k,c,st,en);/*it is not used*/
    if(k==1) {
        if( c==0 && (st||en) && r==0 ) ans++;
        /* if this is the last element to take
        then is should either connect st and en ,
        or be the first element or last */
    }
    else{
        if(st==0) ans+=(f(n-1,r,k-1,c,1,en)+f(n-1,r,
            k-1,c-1,1,en)*c); // this is starting element
        if(en==0) ans += ( f(n-1,r,k-1,c,st,1) +
            f(n-1,r,k-1,c-1,st,1)*c); // this is ending element
```

```
ans += f( n-1 , r , k-1 , c+1 , st , en );
// created & independent
ans += f(n-1,r,k-1,c,st,en)*2*c+
    f(n-1,r,k-1,c,st,en)*(st+en);
/*created and connected with some other
component possibly start or end component */
ans += (f(n-1,r,k-1,c-1,st,en)*c*(c-1)+
    f(n-1,r,k-1,c-1,st,en)*c*(st+en));
/*created and connected between two component */
}
return dp[n][r][k][c][st][en] = ans%mod;
}
```

## 1.3 Convex Hull Trick Linear

```
//Min:M inc, x dec, useless(s-1, s-2, s-3)
// M dec, x inc, useless(s-3, s-2, s-1)
//Max:M inc, x inc, useless(s-3, s-2, s-1)
// M dec, x dec, useless(s-1, s-2, s-3)
struct CHT {
    vector<LL> M; vector<LL> C; int ptr = 0;
    bool useless(int l1, int l2, int l3) {
        return (C[l3]-C[l1])*(M[l1]-M[l2])
            <= (C[l2]-C[l1])*(M[l1]-M[l3]);
    } //Use double comp if M,C is LL range
    LL f(int id, LL x) {return M[id]*x+C[id];}
    void add(LL m, LL c) {
        M.push_back(m); C.push_back(c);
        int s = M.size();
        while (s >= 3 && useless(s-3, s-2, s-1)) {
            M.erase(M.end()-2); C.erase(C.end()-2); s--;
        }
    }
    LL query(LL x) {
        if (ptr >= M.size()) ptr = M.size()-1;
        while (ptr<M.size()-1 && f(ptr,x)>f(ptr+1,x))
            ptr++;
        return f(ptr, x);
    }
};
```

## 1.4 Convex Hull Trick Online

```
//cht for max, for min, insert(-m,-c) and negate
bool Q;
struct Line {
    mutable ll m, c, p;
    bool operator<(const Line& o) const {
        return Q ? p < o.p : m < o.m;
    }
};
struct LineContainer : multiset<Line> {
    // (for doubles, use inf = 1/.0, div(a,b) = a/b)
    const ll inf = LLONG_MAX;
    ll div(ll a, ll b) { // floored division
        return a / b - ((a ^ b) < 0 && a % b); }
    bool isect(iterator x, iterator y) {
        if (y == end()) { x->p = inf; return false; }
        if (x->m==y->m) x->p=x->c > y->c ? inf : -inf;
        else x->p = div(y->c - x->c, x->m - y->m);
        return x->p >= y->p;
    }
    void addLine(ll m, ll c) {
        auto z = insert({m, c, 0}), y = z++, x = y;
```

```
while (isect(y, z)) z = erase(z);
if (x!=begin() && isect(--x, y))
    isect(x,y=erase(y));
while ((y = x) != begin() && (--x)->p >= y->p)
    isect(x, erase(y));
}
ll query(ll x) {
    assert(!empty());
    Q = 1; auto l = *lower_bound({0,0,x}); Q = 0;
    return l.m * x + l.c;
}
bool isEmpty(){ return (empty()) ; }
void Clear() { clear() ; }
}ch;
```

## 1.5 Digit DP Template

```
int call(int pos, int st, int sm,
    bool strt, bool alLarge, bool alSmall) {
    if(sm > limit) return 0;
    if(pos == 200) return 1;
    if(dp[pos][st][sm][strt][alLarge][alSmall]!=-1)
        return dp[pos][st][sm][strt][alLarge][alSmall];
    int ret = 0;
    for(int i = 0; i < base; i++) {
        if(alLarge == false && i<L[pos]-'a') continue;
        if(alSmall == false && i>R[pos]-'a') continue;
        int npos = pos + 1;
        bool nstrt = strt|(i>0);
        bool nalLarge = alLarge|(i>L[pos]-'a');
        bool nalSmall = alSmall|(i<R[pos]-'a');
        int nst = st;
        if(nstrt) nst = Next(nst, 'a' + i);
        int nsm = sm + sts[nst].cnt;
        add(ret,
            call(npos,nst,nsm,nstrt,nalLarge,nalSmall));
    }
    dp[pos][st][sm][strt][alLarge][alSmall] = ret;
    return ret;
}
```

## 1.6 Knuth Optimisation

```
//There are n points on the segment (0, l). You
//have to mark the points in some order. Cost of
//picking a point. Cost of marking is the distance
//between closest marked points to the left and
//to the right. Minimise cost.
LL a[N], dp[N][N], opt[N][N];
LL Knuth(int l, int n) {
    a[0] = 0; a[+n] = 1;
    for (int i=1;i<=n;i++) opt[i-1][i]=i-1;
    for (int len=2; len<=n; len++)
        for (int l=0; l+len<=n; l++) {
            int r = l+len, optl = opt[l][r-1];
            int optr = opt[l+1][r]; dp[l][r] = INF;
            for (int i=optl; i<=optr; i++) {
                LL c = dp[l][i] + dp[i][r] + a[r] - a[l];
                if(c<dp[l][r]) dp[l][r] = c, opt[l][r]=i;
            }
        }
    return dp[0][n];
}
```

## 1.7 SOS DP

```
for (int i=0; i<(1<<K); i++) dp[i] = a[i];
for (int i=0; i<K; i++)
    for (int mask = 0; mask < (1<<K); mask++) {
        if ((mask & (1<<i))) /// == 0 for supermasks
            a[mask] += a[mask^(1<<i)];
```

## 1.8 SOS(on the fly)

```
public class TestProctoring {
    public double expectedTime(int[] p, int[] q) {
        int n = p.length;
        double[] prob = new double[n];
        for (int i = 0; i < n; i++) {
            prob[i] = p[i] * 1.0 / q[i];
        }
        double[][] t = new double[n+1][1<<n];
        double[] dp = new double[1<<n];

        /* t[i][mask] is sum of all submask of mask where
        difference of mask and submask is before i'th
        bit( 0 based ) , that means difference can be in 0
        to i-1 th bit t[0][mask] contains nothing other
        than just value of this mask t[n][mask] contains
        result of all submask of this mask */
        for (int mask = 1; mask < 1 << n; mask++){
            double fail = 1; double mult = 1;
            double am = 1;
            for (int j = 0; j < n; j++) {
                t[j+1][mask] = t[j][mask];
                if (((mask>>j)&1) == 1) {
                    t[j+1][mask] += t[j][mask^(1<<j)];
                    fail *= (1 - prob[j]);
                    mult *= prob[j];
                    am *= (1 - prob[j]) / prob[j];
                }
            }
            dp[mask] = (1+mult*t[n][mask]) / (1 - fail);
            for (int j = 0; j <= n; j++) {
                t[j][mask] += dp[mask] * am;
            }
        }
        return dp[(1<<n)-1];
    }
}
```

## 2 Data Structures

### 2.1 2D BIT Range update Range query

```
const int mx = 1002, my = 1002;
long long bit[4][mx][my];
void update( int x, int y, int val, int i ) {
    int y1;
    while( x<=mx ) {
        y1=y;
        while( y1<=my)
            bit[i][x][y1] += val, y1 += (y1&-y1);
        x += (x&-x);
    }
}

long long query( int x, int y, int i ) {
    long long ans=0; int y1;
    while( x>0 ) {
```

```
        y1 = y;
        while( y1>0 )
            ans += bit[i][x][y1], y1 -= (y1&-y1);
        x -= (x&-x);
    }
    return ans;
}

// add value k from (x1,y1) to (x2,y2) inclusive
void add( int x1, int y1, int x2, int y2, int k ) {
    update(x1,y1,k,0);
    update(x1,y2+1,-k,0);
    update(x2+1,y1,-k,0);
    update(x2+1,y2+1,k,0);
    update(x1,y1,k*(1-y1),1);
    update(x1,y2+1,k*y2,1);
    update(x2+1,y1,k*(y1-1),1);
    update(x2+1,y2+1,-y2*k,1);
    update(x1,y1,k*(1-x1),2);
    update(x1,y2+1,k*(x1-1),2);
    update(x2+1,y1,k*x2,2);
    update(x2+1,y2+1,-x2*k,2);
    update(x1,y1,(x1-1)*(y1-1)*k,3);
    update(x1,y2+1,-y2*(x1-1)*k,3);
    update(x2+1,y1,-x2*(y1-1)*k,3);
    update(x2+1,y2+1,x2*y2*k,3);
}

// get value from (x1,y1) to (x2,y2) inclusive
long long get( int x1, int y1, int x2, int y2 ) {
    LL v1=query(x2,y2,0)*x2*y2 +
        query(x2,y2,1)*x2 +
        query(x2,y2,2)*y2 +
        query(x2,y2,3);
    LL v2=query(x2,y1-1,0)*x2*(y1-1) +
        query(x2,y1-1,1)*x2 +
        query(x2,y1-1,3) +
        query(x2,y1-1,2)*(y1-1);
    LL v3=query(x1-1,y2,0)*(x1-1)*y2 +
        query(x1-1,y2,2)*y2+
        query(x1-1,y2,1)*(x1-1) +
        query(x1-1,y2,3);
    LL v4=query(x1-1,y1-1,0)*(x1-1)*(y1-1) +
        query(x1-1,y1-1,1)*(x1-1) +
        query(x1-1,y1-1,2)*(y1-1) +
        query(x1-1,y1-1,3);
    LL ans=v1-v2-v3+v4;
    return ans;
}
```

### 2.2 Centroid Decomposition

```
vector<int> g[N]; int n, child[N], done[N];
void dfs_size(int u, int par) {
    child[u] = 1;
    for (int v: g[u]) {
        if (done[v] or v == par) continue;
        dfs_size(v, u); child[u] += child[v];
    }
}

int dfs_find_centroid(int u, int par, int sz) {
    for (int v: g[u]) {
        if (!done[v] and v != par and child[v] > sz) {
            return dfs_find_centroid(v, u, sz);
        }
    }
}
```

```
    }
    return u;
}

void solve (int u) {/**problem specific things */}
void dfs_decompose(int u) {
    dfs_size(u, -1);
    int centroid=dfs_find_centroid(u,-1,child[u]/2);
    solve(centroid);
    done[centroid] = 1;
    for (int v : g[centroid]) {
        if (!done[v]) dfs_decompose(v);
    }
}
```

### 2.3 HLD

```
namespace hld{
    int in[maxn] , out[maxn] , sub[maxn] , t = 1,
    nxt[maxn] , depth[maxn], par[maxn] , n ;
    vector<int> g[maxn] ;
    void init(int _n){
        n = _n;
        for(int i=0 ; i<=n ; i++) g[i].clear() ;
    }
    void addEdge(int u, int v){
        g[u].pb(v) ; g[v].pb(u) ;
    }
    void dfsSZ(int u){
        sub[u] = 1 ;
        for(int i=0 ; i<g[u].size() ; i++){
            int v = g[u][i] ;
            for(int j=0 ; j<g[v].size() ; j++){
                if( g[v][j] == u ){
                    g[v].erase(g[v].begin()+j);
                    break ;
                }
            }
            dfsSZ(v) ;
            sub[u] += sub[v] ;
            if(sub[v]>sub[g[u][0]])swap(g[u][0],g[u][i]);
        }
    }
    void dfsHLD(int u){
        in[u] = ++t ;
        for(int i=0 ; i<g[u].size() ; i++){
            int v = g[u][i] ; par[v] = u ;
            depth[v] = depth[u] + 1 ;
            if( i==0 ) nxt[v] = nxt[u] ;
            else nxt[v] = v ;
            dfsHLD(v) ;
        }
        out[u] = t ;
    }
    void preprocess(int root){
        dfsSZ(root) ; t = 0 ; nxt[root] = root ;
        depth[root] = 1 ; dfsHLD(root) ;
    }
    int hldQuery( int u , int v ){
        int ans = -INF ;
        while( nxt[u] != nxt[v] ){
            while( depth[nxt[u]]<depth[nxt[v]] ){
                ans=max(ans,query(1,1,n,in[nxt[v]],in[v]));
            }
        }
    }
}
```

```
// do your thing here ( from in[v] to in[ nxt[v]] )
    v = par[nxt[v]];
}
else{
    ans=max(ans,query(1,1,n,in[nxt[u]],in[u]));
// do your thing here(from in[u] to in[nxt[u]])
    u = par[nxt[u]];
}
}
int lc ;
if( depth[u] > depth[v] ) swap(u,v) ;
lc = u ;
//here lc is the lca
//if you are working on node ,
//not on edge, then update/query upto u also
//otherwise update/query from in[u]+1 to in[v]
ans = max( ans , query(1,1,n,in[u]+1,in[v]) );
return ans ;
}
void hldUpdate( int u , int v , int val ){
    while( nxt[u] != nxt[v] ){
        if( depth[ nxt[u] ] < depth[ nxt[v] ] ){
            update(1,1,n,in[ nxt[v] ] , in[v] , val );
        }
// do your thing here ( from in[v] to in[ nxt[v] ] )
        v = par[ nxt[v] ] ;
    }
    else{
        update(1,1,n,in[ nxt[u] ] , in[u] , val );
// do your thing here (from in[u] to in[nxt[u]])
        u = par[ nxt[u] ] ;
    }
}
int lc ;
if( depth[u] > depth[v] ) swap(u,v) ;
lc = u ;
//here lc is the lca
//if you are working on node , not on edge,
//then update/query upto u also
//otherwise update/query from in[u]+1 to in[v]
update(1,1,n,in[u]+1,in[v],val) ;
return ;
}
}
```

## 2.4 Li Chao Tree

```
LL val(line l, LL x){ return l.m*x + l.c;}
void init(int cn) {
    ch[cn][0] = ch[cn][1] = -1;
}
// to add line call(new line,0,MIN_X,MAX_X)
void add_line(int cn, int b, int e, line l){
    if(tot==0){ tr[0] = 1; init(0); tot++; return;}
    if(l.m == tr[cn].m && l.c == tr[cn].c) return;
    int m = (b + e)>>1;
    bool lft = val(l, b) < val(tr[cn], b);
    bool mid = val(l, m) < val(tr[cn], m);
    bool rgt = val(l, e) < val(tr[cn], e);
    if(lft == rgt){ if(lft) tr[cn] = l; return;}
    if(mid) swap(tr[cn], l);
    if(b==e) return;
    else if(lft != mid){
        if(ch[cn][0] == -1){
```

```
tr[tot] = 1; init(tot); ch[cn][0] = tot++;
return;
}
add_line(ch[cn][0], b, m, l);
}
else{
    if(ch[cn][1] == -1){
        tr[tot] = 1; init(tot); ch[cn][1] = tot++;
        return;
    }
    add_line(ch[cn][1],m+1,e,l);
}
}
}
// call get(0,1,MAX_X,x)
LL get(int cn, int b, int e, LL x){
    if(cn== -1) return MAXY;
    int m = (b + e)>>1;
    if(b==e) return val(tr[cn],x);
    if(x <= m)
        return min(val(tr[cn],x),get(ch[cn][0],b,m,x));
    return min(val(tr[cn],x),get(ch[cn][1],m+1,e,x));
}
```

## 2.5 Persistent Segment Tree

```
int a[N], root[N];
struct node { int sm, l, r;} node[N*LOG];
int tot_nodes = 0;
int upd(int cn, int b, int e, int i, int val) {
    int cur = ++tot_nodes;
    if(b==e) {
        node[cur].sm=node[cn].sm + val; return cur;
    }
    int mid = (b+e)/2;
    if (i <= mid) {
        node[cur].l = upd(node[cn].l, b, mid, i, val);
        node[cur].r = node[cn].r;
    }
    else {
        node[cur].r = upd(node[cn].r, mid+1,e,i, val);
        node[cur].l = node[cn].l;
    }
    node[cur].sm = node[node[cur].l].sm
        + node[node[cur].r].sm;
    return cur;
}
int query(int cn , int b , int e , int i, int j) {
    if (b > j or e < i or !cn) return 0 ;
    if (b >= i and e <= j) return node[cn].sm;
    int mid = (b+e)/2;
    return query(node[cn].l,b,mid,i,j)
        + query(node[cn].r,mid+1,e,i,j);
}
```

## 2.6 RMQ(2D)

```
int st[K][K][N][N]; int lg[N];
void pre() {
    lg[1] = 0;
    for (int i=2; i<N; i++) lg[i] = lg[i/2]+1;
}
int query(int l1, int r1, int l2, int r2) {
    int xx = lg[l2-l1+1], yy = lg[r2-r1+1];
    return max(max(st[xx][yy][l1][r1],
```

```
st[xx][yy][l2-(1<<xx)+1][r1]),
max(st[xx][yy][l1][r2-(1<<yy)+1],
st[xx][yy][l2-(1<<xx)+1][r2-(1<<yy)+1]));
}
void build() {
    for (int x=0; x<K; x++) {
        for (int y=0; y<K; y++) {
            for (int i=1; i<=n; i++) {
                for (int j=1; j<=m; j++) {
                    if (i+(1<<x)-1>n || j+(1<<y)-1>m)
                        continue;
                    if (!x&&!y) st[0][0][i][j]=flag[i][j];
                    else if (x>0) st[x][y][i][j] =
max(st[x-1][y][i][j],st[x-1][y][i+(1<<(x-1))][j]);
                    else if (y>0) st[x][y][i][j] =
max(st[x][y-1][i][j],st[x][y-1][i][j+(1<<(y-1))]);
                }
            }
        }
    }
}
```

## 2.7 SegTree Range Inc, Max Query

```
LL tr[4*N], lz[4*N];
void propagate(int u, int st, int en) {
    if (!lz[u]) return;
    tr[u] += lz[u];
    if (st!=en) {lz[2*u]+=lz[u]; lz[2*u+1]+=lz[u];}
    lz[u] = 0;
}
void update(int u,int st,int en,int l,int r,LL x){
    propagate(u, st, en);
    if (r<st || en<l) return;
    else if (l<=st && en<=r){
        lz[u]+=x; propagate(u, st, en);
    }
    else {
        int mid = (st+en)/2;
        update(2*u, st, mid, l, r, x);
        update(2*u+1, mid+1, en, l, r, x);
        tr[u] = max(tr[2*u], tr[2*u+1]);
    }
}
LL query(int u, int st, int en, int l, int r) {
    propagate(u, st, en);
    if (r<st || en<l) return -inf;
    else if (l<=st && en<=r) return tr[u];
    else {
        int mid = (st+en)/2;
        return max(query(2*u, st, mid, l, r),
            query(2*u+1, mid+1, en, l, r));
    }
}
```

## 3 Geometry

### 3.1 Circle Cover

```
///Check if the all of the area of circ(0, R) in
///Circ(00, RR) is covered by some other circle
bool CoverCircle(PT O, double R, vector<PT> &cen,
    vector<double> &rad, PT OO, double RR) {
    int n = cen.size();
```



```
vector<pair<double, double>> arcs;
for (int i=0; i<n; i++) {
    PT P = cen[i]; double r = rad[i];
    if (i!=0 && R + sqrt(dist2(0, P))<r) return 1;
    if (i==0 && r + sqrt(dist2(0, P))<R) return 1;
    vector<PT> inter =
        CircleCircleIntersection(0, P, R, r);
    if (inter.size() <= 1) continue;
    PT X = inter[0], Y = inter[1];
    if (cross(0, X, Y) < 0) swap(X, Y);
    if (!(cross(0, X, P) >= 0 &&
        cross(0, Y, P) <= 0)) swap(X, Y);
    if (i==0) swap(X, Y);
    X = X-0; Y=Y-0;
    double ll = atan2(X.y, X.x);
    double rr = atan2(Y.y, Y.x);
    if (rr < ll) rr += 2*PI;
    arcs.emplace_back(ll, rr);
}
if (arcs.empty()) return false;
sort(arcs.begin(), arcs.end());
double st = arcs[0].ff, en = arcs[0].ss, ans = 0;
for (int i=1; i<arcs.size(); i++) {
    if (arcs[i].first <= en + EPS)
        en = max(en, arcs[i].second);
    else st = arcs[i].first, en = arcs[i].second;
    ans = max(ans, en-st);
}
return ans >= 2*PI;
}
```

### 3.2 Circle Polygon Common

```
LD areaCT(Point pa, Point pb, LD r) {
    if (pa.Norm() < pb.Norm()) swap(pa, pb);
    if (dcmp(pb.Norm()) == 0) return 0;
    LD a=pb.Norm(), b=pa.Norm(), c=(pb-pa).Norm();
    LD sinB = fabs(pb.cross(pb-pa)/a/c);
    LD cosB = pb.dot(pb-pa)/a/c;
    LD sinC = fabs(det(pa, pb)/a/b);
    LD cosC = pa.dot(pb)/a/b;
    LD B = atan2(sinB, cosB), C = atan2(sinC, cosC);
    LD S = 0.;
    if (a > r) {
        S = C / 2 * r * r;
        LD h = a * b * sinC / c;
        if (h < r && B < PI / 2) {
            S -= (Acos(h/r)*r*r - h*sqrt(r*r-h*h));
        }
    } else if (b > r) {
        LD theta = PI - B - Asin((sinB/r * a, -1, +1));
        S = a*r*sin(theta)/2 + (C-theta)/2*r*r;
    } else {
        S = sinC * a * b / 2;
    }
    return S;
}
LD poly_cross(vector<Point> P, Point cen, LD r) {
    int n = P.size(); LD ans = 0;
    for(int i = 0; i < n; i++) {
        LD cr=fabs(areaCT(P[i]-cen, P[(i+1)%n]-cen, r))*
            dcmp((P[i]-cen).cross(P[(i+1)%n]-cen));
        ans += cr;
    }
}
```

```
}
return ans;
}

3.3 Circle Union Area

struct Point {
    LD x,y ;
    LD operator*(const Point &a)const {
        return x*a.y-y*a.x;}
    LD operator/(const Point &a)const {
        return sqrt((a.x-x)*(a.x-x)+(a.y-y)*(a.y-y));}
}po[N];
LD r[N];
int sgn(LD x) {return fabs(x)<EPS?0:(x>0.0?1:-1);}
pair<LD, bool> ARG[2*N] ;
LD cir_union(Point c[], LD r[], int n) {
    LD sum = 0.0 , sum1 = 0.0 , d, p1, p2, p3 ;
    for(int i = 0 ; i < n ; i++) {
        bool f = 1 ;
        for(int j = 0 ; f&&j<n ; j++)
            if(i!=j && sgn(r[j]-r[i]-c[i]/c[j])!=-1)f=0;
        if(!f) swap(r[i], r[-i]), swap(c[i], c[-i]);
    }
    for(int i = 0; i < n; i++) {
        int k = 0, cnt = 0;
        for(int j = 0 ; j < n; j++) {
            if(i!=j&&sgn((d=c[i]/c[j]-r[i]-r[j])<=0){
                p3=acos((r[i]*r[i]+d*d-r[j]*r[j])/
                    (2.0*r[i]*d));
                p2=atan2(c[j].y-c[i].y, c[j].x-c[i].x);
                p1 = p2-p3; p2 = p2+p3;
                if(sgn(p1+PI)==-1) p1+=2*PI, cnt++;
                if(sgn(p2-PI)==1) p2-=2*PI, cnt++;
                ARG[k++] = make_pair(p1, 0);
                ARG[k++] = make_pair(p2, 1);
            }
        }
        if(k) {
            sort(ARG, ARG+k);
            p1 = ARG[k-1].first-2*PI;
            p3 = r[i]*r[i];
            for(int j = 0 ; j < k ; j++) {
                p2 = ARG[j].first;
                if(cnt==0) {
                    sum+=(p2-p1-sin(p2-p1))*p3 ;
                    sum1+=(c[i]+Point(cos(p1), sin(p1))*
                        r[i])*(c[i]+
                        Point(cos(p2), sin(p2))*r[i]);
                }
                p1 = p2;
                ARG[j].second ? cnt--:cnt++;
            }
            else sum += 2*PI*r[i]*r[i];
        }
        return (sum+fabs(sum1))*0.5 ;
    }
}

3.4 Geometry 2D Basic

const LD EPS = 1e-9;
const LD PI = acos(-1);
LD Sq(LD x) {return x * x;}
```

```
LD Acos(LD x){return acos(min(1.0L,max(-1.0L,x)));}
LD Asin(LD x){return asin(min(1.0L,max(-1.0L,x)));}
LD Sqrt(LD x) {return sqrt(max(0.0L, x));}
int dcmp(LD x) {
    if(fabs(x) < EPS) return 0;
    return (x > 0.0 ? +1 : -1);
}
struct Point {
    LD x, y;
    Point() {}
    Point(LD a, LD b) : x(a), y(b) {}
    Point(const Point& a) : x(a.x), y(a.y) {}
    void operator=(const Point& a) { x=a.x; y=a.y;}
    Point operator+(const Point& a) const
        { Point p(x + a.x, y + a.y); return p; }
    Point operator-(const Point& a) const
        { Point p(x - a.x, y - a.y); return p; }
    Point operator*(LD a)const
        { Point p(x*a,y*a); return p; }
    Point operator/(LD a) const
        { assert(a > EPS); Point p(x/a,y/a); return p; }
    bool IsZero() const {
        return abs(x) < EPS && abs(y) < EPS;
    }
    bool operator==(const Point& a) const {
        return (*this - a).IsZero();
    }
    LD cross(const Point& a) const {
        return x * a.y - y * a.x;
    }
    LD cross(Point a, Point b) const {
        a = a-*this; b = b-*this; return a.cross(b);
    }
    LD dot(const Point& a) const {
        return x * a.x + y * a.y;
    }
    LD Norm() const { return Sqrt(Sq(x) + Sq(y));}
    void NormalizeSelf() { *this = *this / Norm();}
    Point Normalize() {
        Point res(*this); res.NormalizeSelf();
        return res;
    }
    LD Dist(const Point& a) const
        {return (*this-a).Norm();}
    LD Angle() const { return atan2(y, x);}
    void RotateSelf(LD angle) {
        LD c = cos(angle), s = sin(angle);
        LD nx = x*c-y*s, ny = y*c+x*s; y = ny, x = nx;
    }
    Point Rotate(LD angle) const {
        Point res(*this); res.RotateSelf(angle);
        return res;
    }
    static bool LexCmp(const Point&a, const Point&b){
        if(abs(a.x - b.x) > EPS) return a.x < b.x;
        return a.y < b.y;
    }
    LD SqNorm() { return x * x + y * y;}
};
struct Circle {
    Point center; LD r;
    Circle(LD x, LD y, LD rad) {
        center = Point(x, y); r = rad;
    }
}
```

```

}
Circle(const Point& a, LD rad):center(a),r(rad){}
Point PointAtAngle(LD ang) const {
    return center+Point{r*cos(ang),r*sin(ang)};
}
bool operator==(const Circle& c) const {
    return center == c.center && abs(r-c.r) < EPS;
}
};
struct Line {
    Point p[2]; bool is_seg;
    Line(Point a, Point b, bool is_seg_ = false) {
        p[0] = a; p[1] = b; is_seg = is_seg_;
    }
    Line() {}
    // Ax + By + C = 0, not tested
    Line(LD A, LD B, LD C, bool is_seg_ = false) {
        if(fabs(A) > EPS)
            p[0]=Point(-C/A,0.),p[1]=Point(-(B+C)/A,1.);
        else
            p[0]=Point(0.,-C/B),p[1]=Point(1.,-(A+C)/B);
        is_seg = is_seg_;
    }
    Point& operator[](int a) { return p[a];}
    Point Dir() { return p[1] - p[0];}
    Point NormalVector() {
        Point perp = p[1]-p[0];perp.RotateSelf(PI/2);
        perp.NormalizeSelf(); return perp;
    }
    Line shift(Point q) { // not tested
        return Line(p[0] + q, p[1] + q, is_seg);
    }
    //(A,B,C) such that A^2+B^2=1, (A,B) >(0,0)
    vector<LD> LineEqNormLD() { // seems ok
        LD A = p[1].y - p[0].y, B = p[0].x - p[1].x;
        LD C = -(A * p[0].x + B * p[0].y);
        assert(abs(A*p[1].x + B*p[1].y + C) < EPS);
        LD norm = Sqrt(Sq(A) + Sq(B));
        vector<LD> res{A, B, C};
        for (auto& x : res) x /= norm;
        if (A < -EPS || (abs(A) < EPS && B < -EPS))
            for (auto& x : res) { x *= -1; }
        return res;
    }
    // assumes that coordinates are integers!
    vector<int> LineEqNormInt() { // seems ok
        int A = round(p[1].y - p[0].y);
        int B = round(p[0].x - p[1].x);
        int C = -(A * p[0].x + B * p[0].y);
        int gcd = abs(__gcd(A, __gcd(B, C)));
        vector<int> res{A, B, C};
        for (auto& x : res) { x /= gcd; }
        if (A < 0 || (A == 0 && B < 0))
            for (auto& x : res) { x *= -1; }
        return res;
    }
};
namespace Utils {
LD Angle(Point P, Point Q, Point R) { // angle PQR
    LD ang2 = (P - Q).Angle(), ang1 = (R-Q).Angle();
    LD ans = ang1 - ang2;

```

```

    if (ans < EPS) ans += 2 * PI;
    return ans;
}
bool PtBelongToLine(Point p, Line l) {
    return abs(l[0].cross(l[1], p)) < EPS;
}
bool PtBelongToSeg(Point p, Line l) { // seems ok
    return abs(p.Dist(l[0])+p.Dist(l[1])
        -l[0].Dist(l[1])) < EPS;
}
bool AreParallel(Line l1, Line l2) { // seems ok
    LD t=l1[0].cross(l2[0],l1[1])
        -l1[0].cross(l2[1],l1[1]);
    return abs(t) < EPS;
}
bool AreCollinear(Line l1, Line l2) { // not tested
    return AreParallel(l1,l2) &&
        PtBelongToLine(l2[0],l1);
}
Point ProjPtToLine(Point p, Line l) { //Tested
    Point dir = l[1]-l[0];
    return l[0]+dir*(dir.dot(p-l[0])/dir.dot(dir));
}
Point ReflectPtWRTLine(Point p, Line l) {
    Point proj = ProjPtToLine(p, l);return proj*2-p;
}
Point ProjPtToSegment(Point p, Line l){ //!tested
    LD base = (l[1]-l[0]).SqNorm();
    if (fabs(base) < EPS) return l[0];
    LD param = (p-l[0]).dot(l[1]-l[0])/base;
    if (param < 0) return l[0];
    if (param > 1) return l[1];
    return l[0] + (l[1]-l[0]) * param;
}
LD PtToLine(Point p, Line l) { // not tested
    Point v1 = l[1] - l[0], v2 = p - l[0];
    return fabs(v1.cross(v2))/v1.Norm();
}
LD PtToSegment(Point p, Line l) {
    if (l[0] == l[1]) return (p-l[0]).Norm();
    Point v1 = l[1]-l[0], v2 = p-l[0], v3 = p-l[1];
    if ((v1.dot(v2)) < -EPS) return v2.Norm();
    else if ((v1.dot(v3)) > EPS) return v3.Norm();
    else return fabs(v1.cross(v2))/v1.Norm();
}
vector<Point> InterLineLine(Line& a, Line& b){ //ok
    Point vec_a = a[1] - a[0];
    Point vec_b1 = b[1] - a[0];
    Point vec_b0 = b[0] - a[0];
    LD tr_area = vec_b1.cross(vec_b0);
    LD quad_area = vec_b1.cross(vec_a)
        + vec_a.cross(vec_b0);
    if (abs(quad_area) < EPS){ //parallel/coinciding
        if (PtBelongToLine(a[0], b)) {
            return {a[0], a[1]};
        } else {
            return {};
        }
    }
    return {a[0] + vec_a * (tr_area / quad_area)};
}
//SZ(res)==0:empty,SZ(res)=1=> intersection is

```

```

//a pt,SZ(res) == 2 => intersection is a segment
vector<Point> InterSegs(Line l1, Line l2) { // ok
    if(!Point::LexCmp(l1[0],l1[1]))
        swap(l1[0], l1[1]);
    if(!Point::LexCmp(l2[0],l2[1]))
        swap(l2[0], l2[1]);
    if(AreParallel(l1, l2)) {
        if(!PtBelongToLine(l2[0],l1))
            return vector<Point>();
        vector<Point> ends(2);
        for (int tr = 0; tr < 2; tr++)
            if (Point::LexCmp(l1[tr], l2[tr]) ^ tr)
                ends[tr] = l2[tr];
        else ends[tr] = l1[tr];
        if ((ends[1] - ends[0]).IsZero())
            ends.pop_back();
        if (SZ(ends)==2&&Point::LexCmp(ends[1],ends[0]))
            return vector<Point>();
        return ends;
    }
    else {
        vector<Point> p = InterLineLine(l1, l2);
        if(PtBelongToSeg(p[0],l1) &&
            PtBelongToSeg(p[0],l2))
            return p;
        return vector<Point>();
    }
}
LD SegmentToSegmentDistance(Line l1,Line l2){ //nt
    vector<Point> inter = InterSegs(l1, l2);
    if(inter.size() > 0) return 0.0;
    LD an=min(PtToSegment(l1[0],l2),
        PtToSegment(l1[1],l2));
    an = min(an, PtToSegment(l2[0], l1));
    an = min(an, PtToSegment(l2[1], l1));
    return an;
}
//0,1,2,3 pts.If 3 pts it means they are equal
vector<Point>InterCircleCircle(Circle a,Circle b){
    if (a.r + EPS < b.r) swap(a, b);
    if (a == b) {
        return vector<Point>{a.PointAtAngle(0),
            a.PointAtAngle(2 * PI / 3),
            a.PointAtAngle(4 * PI / 3)};
    }
    Point diff=b.center-a.center;LD dis=diff.Norm();
    LD ang = diff.Angle();
    LD longest=max(max(a.r,b.r),dis),per=a.r+b.r+dis;
    if (2 * longest > per + EPS)
        return vector<Point>();
    if (abs(2 * longest - per) < 2 * EPS)
        return vector<Point>{a.PointAtAngle(ang)};
    LD d=Acos((Sq(a.r)+Sq(dis)-Sq(b.r))/(2*a.r*dis));
    return vector<Point>{a.PointAtAngle(ang - d),
        a.PointAtAngle(ang+d)};
}
vector<Point>InterCircleLine(Circle c,Line l){ //ok
    Point proj = ProjPtToLine(c.center, l);
    LD dis_proj = c.center.Dist(proj);
    if (dis_proj > c.r + EPS) return vector<Point>();
    LD a = Sqrt(Sq(c.r) - Sq(dis_proj));
    Point dir = l[1] - l[0];

```

```

LD dir_norm = dir.Norm();
vector<Point> candS{proj + dir * (a / dir_norm),
    proj - dir * (a / dir_norm)};
if (cands[0].Dist(cands[1])<EPS)
    return vector<Point>{proj};
return candS;
}
vector<Point>InterCircleSeg(Circle c, Line l){//ok
vector<Point> from_line = InterCircleLine(c, l);
vector<Point> res;
for(auto p:from_line)
    if(PtBelongToSeg(p,l)) res.pb(p);
return res;
}
vector<Point>TangencyPtsToCircle(Circle c,Point p){
LD d = c.center.Dist(p);//ok
if (d < c.r - EPS) return {};
if (d < c.r + EPS) return {p};
LD from_cent = (p - c.center).Angle();
LD ang_dev = Acos(c.r / d);
return {c.PointAtAngle(from_cent - ang_dev),
    c.PointAtAngle(from_cent + ang_dev)};
}
vector<Line> OuterTangents(Circle c1, Circle c2) {
if(c1 == c2) { return {}; }//is it best choice?
if(c1.r < c2.r) { swap(c1, c2); }
if(c2.r + c1.center.Dist(c2.center) < c1.r-EPS)
    return {};
if (abs(c1.r - c2.r) < EPS) {
    Point diff = c2.center - c1.center;
    Point R = diff.Rotate(PI/2)*(c1.r/diff.Norm());
    return {{c1.center + R, c2.center + R},
        {c1.center - R, c2.center - R}};
}
Point I = c1.center +
    (c2.center-c1.center)*(c1.r/(c1.r-c2.r));
if (c2.r+c1.center.Dist(c2.center)<c1.r+EPS) {
    return {{I,I+(c2.center-c1.center).Rotate(PI/2)}};
}
vector<Point> to1 = TangencyPtsToCircle(c1, I);
vector<Point> to2 = TangencyPtsToCircle(c2, I);
vector<Line>res{{to1[0],to2[0]},{to1[1],to2[1]}};
assert(Utills::PtBelongToLine(I, res[0]));
assert(Utills::PtBelongToLine(I, res[1]));
return res;
}
vector<Line> InnerTangents(Circle c1, Circle c2) {
if (c1 == c2) return {};//surely best choice
if (c1.r < c2.r) { swap(c1, c2); }
LD d = c1.center.Dist(c2.center);
if (d < c1.r + c2.r - EPS) { return {}; }
Point I = c1.center +
    (c2.center-c1.center)*(c1.r/(c1.r+c2.r));
if(d < c1.r + c2.r + EPS) return {{I,I+
    (c2.center-c1.center).Rotate(PI/2)}};
vector<Point> to1 = TangencyPtsToCircle(c1, I);
vector<Point> to2 = TangencyPtsToCircle(c2, I);
vector<Line>res{{to1[0],to2[0]},{to1[1],to2[1]}};
assert(Utills::PtBelongToLine(I, res[0]));
assert(Utills::PtBelongToLine(I, res[1]));
return res;
}

```

```

LD DiskInterArea(Circle c1, Circle c2) {
if (c1.r < c2.r) swap(c1, c2);
LD d = c1.center.Dist(c2.center);
if (c1.r + c2.r < d + EPS) return 0;
if (c1.r - c2.r > d - EPS) return PI * Sq(c2.r);
LD al=Acos((Sq(d)+Sq(c1.r)-Sq(c2.r))/(2*d*c1.r));
LD be=Acos((Sq(d)+Sq(c2.r)-Sq(c1.r))/(2*d*c2.r));
return al * Sq(c1.r) + be * Sq(c2.r) -
    sin(2*al)*Sq(c1.r)/2-sin(2*be)*Sq(c2.r)/2;
}
Line RadicalAxis(Circle c1, Circle c2) {
LD d = c1.center.Dist(c2.center);
LD a = (Sq(c1.r) - Sq(c2.r) + Sq(d)) / (2 * d);
Point Q = c1.center+(c2.center-c1.center)*(a/d);
Point R = Q+(c2.center-c1.center).Rotate(PI/2);
return Line(Q, R);
}
vector<Point> CirThroughAPtAndTngntToALineWithRad
    (Point p, Line l, LD r) { //not tested
vector<Point> sol;
Point norm = l.NormalVector();
Line l1=l.shift(norm*r);
Line l2=l.shift(norm*(-r));
sol = InterCircleLine(Circle(p, r), l1);
vector<Point> t=InterCircleLine(Circle(p, r),l2);
for(auto pp : t) sol.push_back(pp);
return sol;
}
vector<Point>CirTngntToTwoLinesWithRad
    (Line l1,Line l2,LD r) { // not tested
vector<Point> sol;
Point e1 = l1.NormalVector();
Point e2 = l2.NormalVector();
Line L1[2]={l1.shift(e1*r),l1.shift(e1*(-r))},
L2[2]={l2.shift(e2 * r),l2.shift(e2 * (-r))};
for(int i = 0; i < 2; i++) {
    for(int j = 0; j < 2; j++) {
        vector<Point> t = InterLineLine(L1[i],L2[j]);
        for(auto pp : t) sol.push_back(pp);
    }
}
return sol;
}
vector<Point> CirTanToTwoDisjointCirclesWithRadius
    (Circle c1, Circle c2, LD r) { // not tested
c1.r += r; c2.r += r;
return InterCircleCircle(c1, c2);
}
// CENTERS BEGIN
Point Bary(Point A,Point B,Point C,LD a,LD b,LD c){
return (A * a + B * b + C * c) / (a + b + c);
}
Point Centroid(Point A, Point B, Point C) {
return Bary(A, B, C, 1, 1, 1);
}
Point Circumcenter(Point A, Point B, Point C) {
LD a = (B - C).SqNorm(), b = (C - A).SqNorm();
LD c = (A - B).SqNorm();
return Bary(A,B,C,a*(b+c-a),b*(c+a-b),c*(a+b-c));
}
Point Incenter(Point A, Point B, Point C) {

```

```

return Bary(A,B,C,(B-C).Norm(),
    (A-C).Norm(),(A-B).Norm());
}
Point Orthocenter(Point A, Point B, Point C) {
LD a=(B-C).SqNorm(),b=(C-A).SqNorm();
LD c=(A-B).SqNorm();
return Bary(A,B,C,(a+b-c)*(c+a-b),
    (b+c-a)*(a+b-c),(c+a-b)*(b+c-a));
}
Point Excenter(Point A,Point B,Point C){//opp to A
LD a=(B-C).Norm(),b=(A-C).Norm(),c=(A-B).Norm();
return Bary(A, B, C, -a, b, c);
}

```

### 3.5 Geometry 2D Polygon

```

// Cut Polygon (not tested)
void ints(vector<Point> &V,Point a,Point b,Line l){
Point p = l[0], q = l[1];
LD na = (a-p).cross(q-p), nb = (b-p).cross(q-p);
if (na*nb < 0.0)
    V.push_back(a + (b-a)*(na/(na-nb)));
}
vector<Point> cut(vector<Point> polygon,Line l,
    int sign){
vector<Point> np; int sz = polygon.size();
for(int i = 0 ; i < sz ; i++) {
    Point p = polygon[i], q = polygon[(i+1)%sz];
    if(dcmp(l.Dir().cross(p))*sign>=0) np.pb(p);
    ints(np, p, q, l);
}
return np;
}
//diameter of a convex polygon p (not tested)
LD rotating_calipers(vector<Point> p) {
int q = 1, n = p.size(); LD ans = 0;
for( int i = 0; i < n; i++) {
    while(p[i].cross(p[(i+1)%n],p[(q+1)%n]) >
        p[i].cross(p[(i+1)%n],p[q])) q = (q+1)%n;
    LD t1=(p[i]-p[q]).Norm();
    LD t2=(p[(i+1)%n]-p[q]).Norm();
    ans = max(ans, max(t1, t2));
}
return ans;
}
//minimum area rect for convex polygon(!tested)
LD rec_rotating_calipers(vector<Point> p) {
int n=p.size(),l=0,r=0,q=1;
LD ans1=1e15, ans2=1e15;
for( int i = 0; i < n; i++) {
    while(dcmp(p[i].cross(p[(i+1)%n],p[(q+1)%n])
        -p[i].cross(p[(i+1)%n],p[q])) > 0) q=(q+1)%n;
    while(dcmp((p[(i+1)%n]-p[i]).dot
        (p[(r+1)%n]-p[r]))>0) r=(r+1)%n;
    if (!i) l = q;
    while(dcmp((p[(i+1)%n]-p[i]).dot
        (p[(l+1)%n]-p[l]))<0) l=(l+1)%n;
    LD d = (p[(i+1)%n]-p[i]).Norm();
    LD h = p[i].cross(p[(i+1)%n],p[q])/d;
    LD w = (((p[(i+1)%n]-p[i]).dot(p[r]-p[i]))
        -((p[(i+1)%n]-p[i]).dot(p[l]-p[i])))/d;
    ans1 = min(ans1,2*(h+w)),ans2 = min(ans2,h*w);
}

```



```

}
return ans2;
}
struct Polygon {
vector<Point> pts;
Polygon(vector<Point> pts_) : pts(pts_) {}
Polygon() : Polygon(vector<Point>()) {}
void Add(Point p) { pts.push_back(p); }
// positive for counterclockwise
LD Area() {
LD area = 0;
for(int i = 0; i < SZ(pts); i++)
area += pts[i].cross(pts[(i + 1) % SZ(pts)]);
area /= 2; return area;
}
void OrientCounterclockwise() {
if (Area() < 0) reverse(pts.begin(), pts.end());
}
int next(int a) {
if (a + 1 < SZ(pts)) return a + 1;
return 0;
}
pair<int, int> FurthestPair() {
MakeConvexHull(); OrientCounterclockwise();
int furth = 1;
pair<int, int> best_pair = make_pair(0, 0);
LD best_dis = 0;
for (int i = 0; i < SZ(pts); i++) {
Point side = pts[next(i)] - pts[i];
while(side.cross(pts[furth] - pts[i])
< side.cross(pts[next(furth)] - pts[i])){
furth = next(furth);
}
vector<int> vec{i, next(i)};
for (auto ind : vec) {
if (pts[ind].Dist(pts[furth]) > best_dis){
best_pair = make_pair(ind, furth);
best_dis = pts[ind].Dist(pts[furth]);
}
}
}
return best_pair;
}
void MakeConvexHull() {
vector<Point> one_way_hull[2];
sort(pts.begin(), pts.end(), Point::LexCmp);
for (int dir = -1; dir <= 1; dir += 2) {
int hull_num = (dir + 1) / 2;
auto& H = one_way_hull[hull_num];
one_way_hull[hull_num].push_back(pts[0]);
if (SZ(pts) > 1) {
H.push_back(pts[1]);
}
for(int i = 2; i < SZ(pts); i++) {
while(SZ(H) >= 2 && dir * (pts[i] - H[SZ(H)-2]).
cross(H.back() - H[SZ(H)-2]) > -EPS){
H.pop_back();
}
H.push_back(pts[i]);
}
}
pts.clear();

```

```

for(auto p:one_way_hull[1])pts.push_back(p);
for(int i = SZ(one_way_hull[0])-2; i >= 1; i--)
pts.push_back(one_way_hull[0][i]);
}
// without sides
vector<vector<bool>> InsideDiagonalsMatrix() {
int n = pts.size();
vector<vector<bool>> res(n, vector<bool>(n));
for (int i = 0; i < n; i++) {
for (int j = 0; j < n; j++) {
Line diag(pts[i], pts[j]);
if(i==j || abs(i-j)==1 || abs(i-j)==n-1)
continue;
res[i][j] = 1;
for (int k = 0; k < n; k++) {
int kk = next(k);
Line side(pts[k], pts[kk]);
if(k==i || k==j || kk==i || kk==j)
continue;
vector<Point>inter =
Utils::InterSegs(diag,side);
if (SZ(inter)) res[i][j] = 0;
}
int act = next(i), passed_j = 0;
LD areas[2] = {0, 0};
while (act != i) {
passed_j |= (act == j);
LD t =
pts[i].cross(pts[act],pts[next(act)]);
areas[passed_j] += t; act = next(act);
}
if (areas[0] * areas[1] < EPS)
res[i][j] = 0;
}
}
return res;
}
// CLIP START
bool InUpper(Point a) {
if (abs(a.y) > EPS) return a.y > 0;
return a.x > 0;
}
bool angle_cmp(const Point a, const Point b) {
bool u = InUpper(a), v = InUpper(b);
return u!=v ? u : a.cross(b)>0;
}
LD cross(Point a, Point b, Point c, Point d) {
return (d-c).cross(a-c) / (d - c).cross(a - b);
}
struct ClipLine { // valid side is on left
ClipLine(Point A, Point B) {
al = A, bl = B, a = A, b = B;
}
Point al,bl; // original line points
mutable Point a,b; // actual intersection points
Point dir() const { return bl - al; }
bool operator<(const ClipLine& l) const {
return angle_cmp(dir(),l.dir());
}
}
Point cross(const ClipLine& l) {
return al+(bl-al)*::cross(al,bl,l.al,l.bl);
}

```

```

}
bool left(Point p){return(bl-al).cross(p-al)>0;}
};
struct Clip {
Clip(LD r) : area(4*r*r) {
Point a{-r,-r}, b{r,-r}, c{r,r}, d{-r,r};
lines = {ClipLine(a,b), ClipLine(b,c),
ClipLine(c,d), ClipLine(d,a)};
}
void insert(Line l){insert(ClipLine(l[0],l[1]));}
void insert(ClipLine l) {
assert(abs(l.dir().SqNorm()) > EPS); find(l);
while(size()&&!l.left(it->a)&&!l.left(it->b))
erase();
if(size())
while(prev(),size() && !l.left(it->a) &&
!l.left(it->b)) erase();
if(size()&&!l.left(it->a) || !l.left(it->b)){
l.a = l.cross(*it);
area -= l.a.cross(it->b)*.5;
it->b = l.a; next();
l.b = l.cross(*it);
if ((l.a-l.b).SqNorm() < EPS) l.b = l.a;
area -= it->a.cross(l.b) * .5;
it->a = l.b;
if (!(l.a - l.b).IsZero()) {
area += l.a.cross(l.b)*.5;
lines.insert(l);
}
}
}
void find(const ClipLine &l) {
it = lines.lower_bound(l);
if(it == lines.end()) { it = lines.begin(); }
}
void recalculate() {
area = 0;
for(const ClipLine &l : lines)
area+=l.a.cross(l.b);
area *= .5;
}
int size() { return lines.size(); }
void next(){if(++it==lines.end())
it = lines.begin();}
void prev(){
if(it==lines.begin()) it=lines.end();--it;
}
void erase() {
assert(it!=lines.end());
area -= it->a.cross(it->b)*.5;
it = lines.erase(it);
if(it==lines.end()) it = lines.begin();
}
typename set<ClipLine>::iterator it;
set<ClipLine> lines;
LD area;
};

```

### 3.6 Geometry 3D Basic

```

struct Point3 {
LD x, y, z;
Point3() {}

```



```

Point3(LD a, LD b, LD c) : x(a), y(b), z(c){}
void operator=(const Point3& a) {
    x=a.x,y=a.y,z=a.z;
}
Point3 operator+(Point3 a) {
    Point3 p{x + a.x, y + a.y, z + a.z}; return p;
}
Point3 operator-(Point3 a) {
    Point3 p{x - a.x, y - a.y, z - a.z}; return p;
}
Point3 operator*(LD a) {
    return Point3(x*a,y*a,z*a);
}
Point3 operator/(LD a) {
    assert(a > EPS); Point3 p{x/a, y/a, z/a};
    return p;
}
LD& operator[](int a) {
    if (a == 0) return x;
    if (a == 1) return y;
    if (a == 2) return z;
    assert(false);
}
bool IsZero() {
    return abs(x)<EPS&& abs(y)<EPS && abs(z) < EPS;
}
bool operator==(Point3 a) {
    return (*this - a).IsZero();
}
LD dot(Point3 a) {
    return x * a.x + y * a.y + z * a.z;
}
LD Norm() {
    return Sqrt(x * x + y * y + z * z);
}
LD SqNorm() {
    return x * x + y * y + z * z;
}
void NormalizeSelf() {
    *this = *this/Norm();
}
Point3 Normalize() {
    Point3 res(*this); res.NormalizeSelf();
    return res;
}
LD Dis(Point3 a) {
    return (*this - a).Norm();
}
pair<LD, LD> SphericalAngles() {
    return {atan2(z,Sqrt(x*x+y*y)),atan2(y,x)};
}
LD Area(Point3 p) {
    return Norm() * p.Norm() * sin(Angle(p)) / 2;
}
// LD Angle(Point3 p) {
//     LD a = Norm(), b = p.Norm(), c = Dis(p);
//     return Acos((a*a+b*b-c*c)/(2*a*b));
// }
LD Angle(Point3 b) { // not tested
    Point3 a(*this);
    return Acos(abs(a.dot(b))/a.Norm()/b.Norm());
}
LD Angle(Point3 p, Point3 q){return p.Angle(q);}

```

```

Point3 cross(Point3 p) {
    Point3 q(*this);
    return {q[1]*p[2] - q[2]*p[1], q[2]*p[0] -
            q[0] * p[2], q[0] * p[1] - q[1] * p[0]};
}
bool LexCmp(Point3& a, const Point3& b) {
    if (abs(a.x - b.x) > EPS) { return a.x < b.x;}
    if (abs(a.y - b.y) > EPS) { return a.y < b.y;}
    return a.z < b.z;
}
};
struct Line3 {
    Point3 p[2];
    Line3() {}
    Line3(Point3 a, Point3 b) { p[0] = a, p[1] = b;}
    Point3& operator[](int a) { return p[a]; }
};
struct Plane {
    Point3 p[3];
    Point3& operator[](int a) { return p[a]; }
    Plane(Point3 p0, Point3 p1, Point3 p2) {
        p[0] = p0; p[1] = p1; p[2] = p2;
    }
    // Ax + By + Cz = D
    Plane(Point3 normal, LD D) {
        /// to do, update p[0], p[1], p[2]
    }
    Point3 GetNormal() {
        Point3 cross = (p[1]-p[0]).cross(p[2]-p[0]);
        return cross.Normalize();
    }
    void GetPlaneEq(LD& A, LD& B, LD& C, LD& D) {
        Point3 normal = GetNormal();
        A = normal[0], B = normal[1], C = normal[2];
        D = normal.dot(p[0]);
        assert(abs(D - normal.dot(p[1])) < EPS);
        assert(abs(D - normal.dot(p[2])) < EPS);
    }
    vector<Point3> GetOrtonormalBase() {
        Point3 normal = GetNormal();
        Point3 cand = {-normal.y, normal.x, 0};
        if (abs(cand.x) < EPS && abs(cand.y) < EPS)
            cand = {0, -normal.z, normal.y};
        cand.NormalizeSelf();
        Point3 third = Plane{Point3{0, 0, 0},
                             normal, cand}.GetNormal();
        return {normal, cand, third};
    }
};
struct Circle3 {
    Plane pl; Point3 cent; LD r;
};
struct Sphere {
    Point3 cent; LD r;
};
namespace Utils3 {
    //angle PQR
    LD Angle(Point3 P, Point3 Q, Point3 R) {
        return (P - Q).Angle(R - Q);
    }
    LD Area(Point3 p, Point3 q, Point3 r) { // ok
        q = q-p; r = r-p; return q.Area(r);
    }
}

```

```

}
LD DistPtLine(Point3 p, Line3 l){ // not tested
    return ((l[1]-l[0]).cross((p-l[0]))).Norm()/
            (l[1]-l[0]).Norm();
}
Point3 ProjPtToLine3(Point3 p, Line3 l) { // ok
    Point3 diff = l[1]-l[0]; diff.NormalizeSelf();
    return l[0] + diff * (p - l[0]).dot(diff);
}
Point3 ProjPtSeg3(Point3 p, Line3 l) {///!tested
    LD r = (l[1]-l[0]).dot(l[1]-l[0]);
    if(abs(r) < EPS) return l[0];
    r = (p-l[0]).dot(l[1]-l[0])/r;
    if(r < 0) return l[0];
    if (r > 1) return l[1];
    return l[0] + (l[1]-l[0]) * r;
}
LD DistPtSeg3(Point3 p, Line3 l) {
    Point3 q = ProjPtSeg3(p, l); return p.Dis(q);
}
LD DisPtLine3(Point3 p, Line3 l) { // ok
    LD dis2 = p.Dis(ProjPtToLine3(p, l));
    return dis2;
}
bool PtBelongToLine3(Point3 p, Line3 l) {
    return DisPtLine3(p, l) < EPS;
}
bool Lines3Equal(Line3 p, Line3 l) {
    return PtBelongToLine3(p[0],l) &&
           PtBelongToLine3(p[1],l);
}
bool OrientPointPlane(Point3 t,Plane p){///!tested
    LD dot = p.GetNormal().dot(t - p[0]);
    return dcmp(dot);
}
Point3 ProjPtToPlane(Point3 p, Plane pl) {
    Point3 normal = pl.GetNormal();
    return p - normal * normal.dot(p - pl[0]);
}
LD DisPtPlane(Point3 p, Plane pl) {
    Point3 normal = pl.GetNormal();
    return abs(normal.dot(p - pl[0]));
}
bool PtBelongToPlane(Point3 p, Plane pl) {
    return DisPtPlane(p, pl) < EPS;
}
bool Line3BelongToPlane(Line3 l, Plane pl) {
    return PtBelongToPlane(l[0], pl) &&
           PtBelongToPlane(l[1], pl);
}
Plane ShiftUpDown(Plane p, LD dist) { ///!tested
    Point3 n = p.GetNormal();
    LD d = p.GetNormal().dot(p[0]);
    return Plane(n, d + dist * n.Norm());
}
Plane ParallelPlane(Plane pl, Point3 A) {
    Point3 diff = A - ProjPtToPlane(A, pl);
    return Plane{pl[0]+diff,pl[1]+diff,pl[2]+diff};
}
//undefined for parallel line and plane(!tested)
Point3 InterLinePlane(Line3 l, Plane p) {

```

```

Point3 norm = p.GetNormal();
LD D = norm.dot(p[0]);
LD k =
    (D - (norm.dot(l[0])))/(norm.dot(l[1]-l[0]));
return l[0] + (l[1]-l[0])*k;
}
// not tested, assumes planes are not parallel
Line3 InterPlanePlane(Plane p1, Plane p2) {
    Point3 n1=p1.GetNormal(), n2 = p2.GetNormal();
    LD d1 = n1.dot(p1[0]), d2 = n2.dot(p2[0]);
    Point3 dir = n1.cross(n2);
    assert(!dir.IsZero()); // parallel plane
    Point3 u =
        (n2*d1 - n1*d2).cross(dir)/dir.dot(dir);
    return Line3(u, u + dir);
}
Point PlanePtTo2D(Plane pl, Point3 p) { // ok
    assert(PtBelongToPlane(p, pl));
    vector<Point3> base = pl.GetOrthonormalBase();
    Point3 control{0, 0, 0};
    for (int tr = 0; tr < 3; tr++) {
        control=control+base[tr] * p.dot(base[tr]);
    }
    assert(PtBelongToPlane(pl[0] + base[1], pl));
    assert(PtBelongToPlane(pl[0] + base[2], pl));
    assert((p - control).IsZero());
    return {p.dot(base[1]), p.dot(base[2])};
}
Line PlaneLineTo2D(Plane pl, Line3 l) {
    return
        {PlanePtTo2D(pl,l[0]),PlanePtTo2D(pl,l[1])};
}
Point3 PlanePtTo3D(Plane pl, Point p) { // ok
    vector<Point3> base = pl.GetOrthonormalBase();
    return base[0]*base[0].dot(pl[0]) +
        base[1]*p.x+base[2]*p.y;
}
Line3 PlaneLineTo3D(Plane pl, Line l) {
    return Line3{PlanePtTo3D(pl,l[0]),
        PlanePtTo3D(pl, l[1])};
}
Line3 ProjLineToPlane(Line3 l, Plane pl) { // ok
    return Line3{ProjPtToPlane(l[0], pl),
        ProjPtToPlane(l[1], pl)};
}
Point3 ClosestPtOnL1FromL2(Line3 l1,Line3 l2){
    Point3 n = (l1[1]-l1[0]).cross(l2[1]-l2[0]);
    Point3 n3 = (l2[1]-l2[0]).cross(n); //!tested
    //p is the plane including line l2 and n
    Plane p = Plane(n3, n3.dot(l2[0]));
    return InterLinePlane(l1, p);
}
vector<Point3> InterLineLine(Line3 k, Line3 l) {
    if (Lines3Equal(k, l)) { return {k[0], k[1]}; }
    if (PtBelongToLine3(l[0], k)) {return {l[0]};}
    Plane pl{l[0], k[0], k[1]};
    if (!PtBelongToPlane(l[1], pl)) { return {}; }
    Line k2 = PlaneLineTo2D(pl, k);
    Line l2 = PlaneLineTo2D(pl, l);
    vector<Point> inter=Utils::InterLineLine(k2,l2);
    vector<Point3> res;
    for (auto P:inter) res.pb(PlanePtTo3D(pl, P));
}

```

```

return res;
}
LD DisLineLine (Line3 l1, Line3 l2){ //!tested
    Point3 dir = (l1[1]-l1[0]).cross(l2[1]-l2[0]);
    if(dcmp(dir.Norm())==0)
        return DistPtLine(l2[0],l1);
    return abs((l2[0]-l1[0]).dot(dir))/dir.Norm();
}
/**
LD DisLineLine(Line3 l, Line3 k) { //ok(para fix)
    Plane together{l[0],l[1],l[0]+k[1]-k[0]};
    Line3 proj = ProjLineToPlane(k, together);
    Point3 inter =
        (Utils3::InterLineLine(l,proj))[0];
    Point3 on_k_inter = k[0] + inter - proj[0];
    return inter.Dis(on_k_inter);
}
**/
LD Det(Point3 a, Point3 b, Point3 d) { // ok
    Point3 pts[3] = {a, b, d};
    LD res = 0;
    for (int sign : {-1, 1}) {
        for (int st_col=0; st_col<3; st_col++) {
            int c = st_col;
            LD prod = 1;
            for (int r=0; r<3; r++){
                prod *= pts[r][c];
                c = (c + sign + 3) % 3;
            }
            res += sign * prod;
        }
    }
    return res;
}
Point3 PtFromSphericalAng(LD al, LD be) { // ok
    return{cos(al)*cos(be),cos(al)*sin(be),sin(al)};
}
//img of B in rot wrt line
//passing thru orig s.t.A1->A2
Point3 RotateAccordingly
    (Point3 A1,Point3 A2,Point3 B1) { // ok
    Plane pl{A1, A2, {0, 0, 0}};
    Point A12 = PlanePtTo2D(pl, A1);
    Point A22 = PlanePtTo2D(pl, A2);
    complex<LD> rat = complex<LD>(A22.x, A22.y) /
        complex<LD>(A12.x, A12.y);
    Plane plb = ParallelPlane(pl, B1);
    Point B2 = PlanePtTo2D(plb, B1);
    complex<LD>Brot = rat*complex<LD>(B2.x, B2.y);
    return
        PlanePtTo3D(plb,{Brot.real(),Brot.imag()});
}
vector<Point3> InterLineSphere(Line3 l,Sphere s){
    vector<Point3> ints; // not tested
    LD h2 = Sq(s.r) - Sq(DisPtLine3(s.cent, l));
    if(dcmp(h2) < 0) return ints;
    if(dcmp(h2) == 0){
        ints.push_back(ProjPtToLine3(s.cent, l));
        return ints;
    }
    Point3 v = ProjPtToLine3(s.cent, l);
    Point3 dir = l[1] - l[0];
    Point3 h = dir * Sqrt(h2)/dir.Norm();
}

```

```

ints.push_back(v+h); ints.push_back(v-h);
return ints;
}
vector<Circle3> InterPlaneSphere
    (Plane pl,Sphere s){ //ok
    Point3 proj = ProjPtToPlane(s.cent, pl);
    LD dis = s.cent.Dis(proj);
    if (dis > s.r + EPS) {
        return {};
    }
    if (dis > s.r - EPS) {
        return {{pl, proj, 0}}; // is it best choice?
    }
    return {{pl,proj,Sqrt(s.r*s.r-dis*dis)}};
}
bool PtBelongToSphere(Sphere s, Point3 p) {
    return abs(s.r - s.cent.Dis(p)) < EPS;
}
LD DisOnSphere(Sphere sph, Point3 A, Point3 B) {
    assert(PtBelongToSphere(sph, A));
    assert(PtBelongToSphere(sph, B));
    LD ang = Angle(A, sph.cent, B);
    return ang * sph.r;
}
bool InsideATriangle
    (Point3 a,Point3 b,Point3 c,Point3 p){
    Plane abc = Plane(a, b, c);
    if(!Utils3::PtBelongToPlane(p, abc))return 0;
    Point3 n = abc.GetNormal();
    vector<int> sign(3);
    for(int i = 0; i < 3; i++) {
        LD t = n.dot((abc[(i+1)%3]-abc[i]).
            cross(p-abc[i]));
        sign[i] = dcmp(t);
    }
    if(sign[0]>=0&&sign[1]>=0&&sign[2]>=0) return 1;
    if(sign[0]<=0&&sign[1]<=0&&sign[2]<=0) return 1;
    return 0;
}
LD PtDistOn3dTriangle
    (Point3 a,Point3 b,Point3 c,Point3 p){
    Plane abc = Plane(a,b,c);
    Point3 p_ = Utils3::ProjPtToPlane(p, abc);
    LD ret = 1e19;
    if(InsideATriangle(a,b,c,p_))
        ret = min(ret, p.Dis(p_));
    ret = min(ret,Utils3::DistPtSeg3(p, Line3(a,b)));
    ret = min(ret,Utils3::DistPtSeg3(p, Line3(b,c)));
    ret = min(ret,Utils3::DistPtSeg3(p, Line3(a,c)));
    return ret;
}
struct Face{
    Point3 a, b, c;
    Face(){
        Face(Point3 a,Point3 b,Point3 c):a(a),b(b),c(c){}
        Face(const Face &f) : a(f.a), b(f.b), c(f.c) {}
    };
LD ployhedronVolume(vector<Face> &vec) { //!tested
    if(vec.size() == 0) return 0;
    Point3 reff = vec[0].a; LD vol = 0;
    for(int i = 1; i < vec.size(); i++) {
}

```

```

    Point3 ar = (vec[i].b-vec[i].a).
        cross(vec[i].c - vec[i].a);
    vol += abs(ar.dot(reff-vec[i].a));
}
return vol/6.0;
}
vector<Face>Convex3dHull(vector<Point3> &V){//nt
vector <Face> Faces;
for(int i = 0; i < V.size(); i++) {
    for(int j = i+1; j < V.size(); j++) {
        for(int k = j+1; k < V.size(); k++) {
            if(((V[j]-V[i]).cross(V[k]-V[i])).Norm()
                < EPS) continue;
            bool up = 0, down = 0;
            Plane P(V[i], V[j], V[k]);
            Point3 normal = P.GetNormal();
            for(int l = 0; l < V.size(); l++) {
                if (l == i or l == j or l == k)
                    continue;
                if(InsideATriangle(V[i],V[j],V[k],V[l])){
                    up = down = 1;
                    break;
                }
                else if(normal.dot(V[l]-V[i])<0) down=1;
                else up = 1;
            }
            if(up == 0 or down == 0) {
                Face temp;
                temp.a=V[i],temp.b=V[j],temp.c=V[k];
                Faces.push_back(temp);
            }
        }
    }
}
return Faces;
}
struct PointS {
    LD lat, lon;
    PointS(LD latt, LD lonn) {lat=latt; lon=lonn;}
    Point3 toEucl() {
        return Point3{cos(lat)*cos(lon),
            cos(lat)*sin(lon),sin(lat)};
    }
    PointS(Point3 p) {
        p.NormalizeSelf(); lat = Asin(p.z);
        lon = Acos(p.y / cos(lat));
    }
};
LD DistS(Point3 a, Point3 b) {
    return atan2l(b.cross(a).Norm(), a.dot(b));
}
struct CircleS {
    Point3 o; // center of circle on sphere
    LD r; // arc len
    LD area() const { return 2*PI*(1 - cos(r)); }
};
CircleS From3(Point3 a,Point3 b,Point3 c){
    int tmp = 1; //any 3 dif pts
    if((a-b).Norm()>(c-b).Norm()){
        swap(a,c);tmp = -tmp;
    }
    if((b-c).Norm()>(a-c).Norm()){
        swap(a,b);tmp = -tmp;
    }
}

```

```

}
Point3 v=(c-b).cross(b-a);
v = v * (tmp / v.Norm());
return CircleS{v, DistS(a,v)};
}
CircleS From2(Point3 a,Point3 b){//nei same nor opp
    Point3 mid = (a + b) / 2;
    mid = mid / mid.Norm();
    return From3(a, mid, b);
}
//angle at A, no two points opposite
LD Angle(Point3 A, Point3 B, Point3 C) {
    LD a = B.dot(C), b = C.dot(A), c = A.dot(A);
    return Acos((b-a*c)/Sqrt((1-Sq(a))*(1-Sq(c))));
}
// no two poins opposite
LD TriangleArea(Point3 A, Point3 B, Point3 C) {
    LD a = Angle(C,A,B), b = Angle(A,B,C);
    LD c = Angle(B,C,A);
    return a + b + c - PI;
}
// what about c1==c2 case?
vector<Point3>IntersectionS
    (CircleS c1, CircleS c2) {
    Point3 n = c2.o.cross(c1.o);
    Point3 w = c2.o * cos(c1.r) - c1.o * cos(c2.r);
    LD d = n.SqNorm();
    if (d < EPS) {
        cerr<<"parallel circles?\n";
        return {};
    }
    LD a = w.SqNorm() / d; vector<Point3> res;
    if (a >= 1 + EPS) return res;
    Point3 u = n.cross(w) / d;
    if (a > 1 - EPS) {
        res.pb(u); return res;
    }
    LD h = Sqrt((1 - a) / d);
    res.pb(u + n * h);
    res.pb(u - n * h);
    return res;
}

```

### 3.7 Geometry 3D Convex Hull

```

typedef vector<Point3> face;
typedef vector<Point3> edge;
typedef vector<face> hull;
#define INSIDE (-1)
#define ON (0)
#define OUTSIDE (1)
int side(Point3 a, Point3 b, Point3 c, Point3 p){
    Point3 norm = (b-a).cross(c-a);
    Point3 me = p-a;
    return dcmp(me.dot(norm));
}
hull find_hull(vector<Point3> P) {
    random_shuffle(P.begin(), P.end());
    int n = P.size();
    for(int j = 2; j < n; j++) {
        Point3 n = (P[1]-P[0]).cross(P[j]-P[0]);
        if(n.Norm() > EPS) {swap(P[j], P[2]);break;}
    }
}

```

```

for(int j = 3; j < n; j++) {
    if(side(P[0],P[1],P[2],P[j])) {
        swap(P[j], P[3]); break;
    }
}
if(side(P[0],P[1],P[2],P[3]) == OUTSIDE)
    swap(P[0], P[1]);
hull H{ {P[0],P[1],P[2]}, {P[0],P[3],P[1]},
    {P[0],P[2],P[3]}, {P[3],P[2],P[1]}};
auto make_degrees = [&](const hull& H) {
    map<edge,int> ans;
    for(const auto & f : H) {
        for(int i = 0; i < 3; i++){
            Point3 a = f[i], b = f[(i+1)%3];
            ans[{a,b}]++;
        }
    }
    return ans;
};
for(int j = 4; j < n; j++) {
    hull H2; H2.reserve((int)H.size());
    vector<face> plane;
    for(const auto & f : H) {
        int s = side(f[0],f[1],f[2],P[j]);
        if (s == INSIDE || s == ON) H2.pb(f);
    }
    //For any edge that now only has 1 incident
    //face (it's other face deleted) add a new
    //face with this vertex and that edge.
    map<edge, int> D = make_degrees(H2);
    const auto tmp = H2;
    for (const auto & f : tmp) {
        for(int i = 0; i < 3; i++) {
            Point3 a = f[i], b = f[(i+1)%3];
            int d = D[{a,b}] + D[{b,a}];
            if (d==1) H2.pb({a, P[j], b});//a new face
        }
    }
    H = H2;
}
return H;
}

```

### 3.8 Geometry Hull

```

int dcmp(int x) {
    if(x < 0) return -1;
    return x > 0;
}
struct Point {
    int x, y;
    Point() {}
    Point(int a, int b) : x(a), y(b) {}
    Point(const Point& a) : x(a.x), y(a.y) {}
    void operator=(const Point& a){x=a.x;y=a.y;}
    Point operator+(const Point& a) const {
        Point p(x + a.x, y + a.y); return p;
    }
    Point operator-(const Point& a) const {
        Point p(x - a.x, y - a.y); return p;
    }
    Point operator*(int a)const {
        return Point(x*a,y*a);
    }
}

```

```

}
Point operator/(int a)const{
    return Point(x/a, y/a);
}
int cross(const Point& a)const {
    return x * a.y - y * a.x;
}
int cross(Point a, Point b) const {
    a = a - *this; b = b - *this; return a.cross(b);
}
int DotProd(const Point& a) const {
    return x * a.x + y * a.y;
}
Point Rotate90() { return Point(-y, x); }
bool operator < (const Point &p) const{
    return make_pair(x, y) < make_pair(p.x, p.y);
}
bool operator > (const Point &p) const {
    return make_pair(x, y) > make_pair(p.x, p.y);
}
int SqNorm() { return x * x + y * y; }
};
bool OnSegment(Point p, Point a, Point b) {
    return (a-p).cross(b-p)==0&&(a-p).DotProd(b-p)<0;
}
int isPointInPolygon(Point p,vector<Point> &poly){
    int wn = 0, n = poly.size();
    for(int i = 0; i < n; i++) {
        if(OnSegment(p,poly[i],poly[(i+1)%n]))
            return -1;//on edge
        int k=(poly[(i+1)%n]-poly[i]).cross(p-poly[i]);
        int d1 = poly[i].y-p.y;
        int d2 = poly[(i+1)%n].y-p.y;
        if (k > 0 && d1 <= 0 && d2 > 0) wn++;
        if (k < 0 && d2 <= 0 && d1 > 0) wn--;
    }
    if (wn != 0) return 1; //inside
    return 0; //outside
}
// returns 1 if p is on or inside triangle(a,b,c)
bool PointInTriangle
    (Point a,Point b,Point c,Point p) {
    int d1 = dcmp((b-a).cross(p-b));
    int d2 = dcmp((c-b).cross(p-c));
    int d3 = dcmp((a-c).cross(p-a));
    return !(((d1 < 0) || (d2 < 0) || (d3 < 0)) &&
        ((d1 > 0) || (d2 > 0) || (d3 > 0)));
}
struct ConvexHull {
    vector<Point> hull, lr, ur; int n;
    /// builds convex hull of a set of points
    bool ccw(Point p, Point q, Point r) {
        return p.cross(q, r) > 0;
    }
    int cross(Point p, Point q, Point r) {
        return (q-p).cross(r-q);
    }
    Point LineLineIntersection
        (Point p1, Point p2, Point q1, Point q2) {
        int a1 = cross(q1,q2,p1),a2 = -cross(q1,q2,p2);
        return (p1 * a2 + p2 * a1) / (a1 + a2);
    }
}

```

```

void init(vector<Point> &poly) {
    hull.clear(), lr.clear(), ur.clear();
    sort(poly.begin(),poly.end());
    for(int i = 0; i < poly.size(); i++) {
        while(lr.size() >= 2 &&
            !ccw(lr[lr.size()-2],lr.back(),poly[i]))
            lr.pop_back();
        lr.push_back(poly[i]);
    }
    for(int i = (int)poly.size()-1; i >= 0; i--){
        while(ur.size() >= 2 &&
            !ccw(ur[ur.size()-2],ur.back(),poly[i]))
            ur.pop_back();
        ur.push_back(poly[i]);
    }
    hull = lr;
    for(int i = 1; i+1 < ur.size(); i++)
        hull.push_back(ur[i]);
    n = hull.size();
}
int sign(int x) {
    if (x < 0) return -1; return x > 0;
}
int crossOp(Point p, Point q, Point r) {
    int c = (q-p).cross(r-q);if (c < 0) return -1;
    return (c > 0);
}
//tests if p is inside or on the convex poly
//if Pt p is on side a,b is the idx of two ends
bool contain(Point p,int&a,int&b){
    if(p.x < lr[0].x || p.x>lr.back().x) return 0;
    int id = lower_bound(lr.begin(),
        lr.end(),Point(p.x,-INF)) - lr.begin();
    if(lr[id].x == p.x){
        if(lr[id].y > p.y) return 0;
    } else {
        if(crossOp(lr[id-1],lr[id],p) < 0) return 0;
        if(crossOp(lr[id-1],lr[id],p) == 0){
            a = id - 1; b = id;
            return 1;
        }
    }
}
id = lower_bound(ur.begin(),ur.end(),Point
    (p.x,INF),greater<Point>()) - ur.begin();
if(ur[id].x == p.x){
    if(ur[id].y < p.y) return 0;
} else {
    if(crossOp(ur[id-1],ur[id],p) < 0) return 0;
    if(crossOp(ur[id-1],ur[id],p) == 0) {
        a = id - 1 + lr.size() - 1;
        b = id + lr.size() - 1;
        return 1;
    }
}
return 1;
}
int find(vector<Point> &vec, Point dir) {
    int l = 0, r = vec.size();
    while(l+5<r){
        int L = (l*2+r)/3, R = (l+r*2)/3;
        if(vec[L].DotProd(dir)>vec[R].DotProd(dir))
            r=R;
    }
}

```

```

else
    l=L;
}
int ret = 1;
for(int k = l+1; k < r; k++)
    if(vec[k].DotProd(dir)>vec[ret].DotProd(dir))
        ret = k;
return ret;
}
///rays frm inf in dir, returns the furthest Pt
int findFarest(Point dir){
    if(sign(dir.y) > 0 || sign(dir.y) == 0 &&
        sign(dir.x) > 0){
        return ((int)lr.size()-1 + find(ur,dir))%n;
    } else {
        return find(lr,dir);
    }
}
Point get(int l, int r, Point p1, Point p2){
    int sl = crossOp(p1,p2,hull[l%n]);
    while(l+1<r){
        int m = (l+r)>>1;
        if(crossOp(p1,p2,hull[m%n]) == sl) l = m;
        else r = m;
    }
    return LineLineIntersection
        (p1,p2,hull[l%n],hull[(l+1)%n]);
}
//Ints between line and convex polygon. O(log(n))
//touching the hull does not count as intersection
vector<Point>Line_Hull_Intersection
    (Point p1, Point p2){
    int X = findFarest((p2-p1).Rotate90());
    int Y = findFarest((p1-p2).Rotate90());
    if(X > Y) swap(X,Y);
    if(crossOp(p1,p2,hull[X])*
        crossOp(p1,p2,hull[Y]) < 0){
        return {get(X,Y,p1,p2),get(Y,X+n,p1,p2)};
    } else {
        return {};
    }
}
void update_tangent(Point p,int id,int&a,int&b){
    if(crossOp(p,hull[a],hull[id]) > 0) a = id;
    if(crossOp(p,hull[b],hull[id]) < 0) b = id;
}
void binary_search(int l,int r,Point p,
    int&a,int&b){
    if(l==r) return;
    update_tangent(p,l%n,a,b);
    int sl = crossOp(p,hull[l%n],hull[(l+1)%n]);
    while(l+1<r){
        int m = l+r>>1;
        if(crossOp(p,hull[m%n],hull[(m+1)%n]) == sl)
            l=m;
        else r=m;
    }
    update_tangent(p,r%n,a,b);
}
void get_tangent(Point p,int&a,int&b){
    if(contain(p,a,b)) return;
    a = b = 0;
    int id = lower_bound(lr.begin(), lr.end(),p)
}

```



```

        - lr.begin();
    binary_search(0,id,p,a,b);
    binary_search(id,lr.size(),p,a,b);
    id = lower_bound(ur.begin(), ur.end(),p,
        greater<Point>()) - ur.begin();
    binary_search((int)lr.size() - 1,
        (int) lr.size() - 1 + id,p,a,b);
    binary_search((int) lr.size() - 1 + id,
        (int) lr.size() - 1 + ur.size(),p,a,b);
}
};

```

### 3.9 Line Polygon Intersection

```

//Dist from O to OX intersect AB, div by len(OX)
double distance(PT O, PT X, PT A, PT B) {
    B = B-A; A = A-O; X=X-O;
    return 1.0*cross(A, B)/cross(X, B);
}
int sign(LL a) { return a==0 ? 0 : (a>0 ? 1:-1);}
//Special Pts are given by pair<double,int>(a,b)
//Let O be a special pt with OX,OY incident edges.
//Then a = AO/AB, b is an integer denoting type.
//CS: OX and OY are on the same side: ignore
//CS: OX and OY are on the different side: b=0
//Cs: OX on line, OY on side, b=sign(cross(A,B,Y))
double LinePoly(PT A, PT B, const vector<PT> &p) {
    int n = p.size();
    vector<pair<double, int>> special;
    for (int i=0; i<n; i++) {
        PT X = p[i], Y = p[(i+1)%n], W = p[(i-1+n)%n];
        LL crx = cross(A, B, X), cry = cross(A, B, Y),
        crw = cross(A, B, W);
        if (crx == 0) {
            double f;
            if (B.x != A.x) f = 1.0*(X.x-A.x)/(B.x-A.x);
            else f = 1.0*(X.y-A.y)/(B.y-A.y);
            if (sign(crw) && sign(cry))
                if (sign(crw) != sign(cry))
                    special.pb({f, 0});
            else if (sign(cry)) special.pb({f, sign(cry)});
            else if (sign(crw)) special.pb({f, sign(crw)});
        }
        else if (sign(crx) == -sign(cry)) {
            double f = distance(A, B, X, Y);
            special.push_back({f, 0});
        }
    }
    sort(special.begin(), special.end());
    bool active = false;
    int sgn = 0; //1st side sign if curntly linear,
    double prv = 0, ans = 0;
    for (auto &pr: special) {
        double d = pr.first; int tp = pr.second;
        if (sgn) {
            assert(sgn && tp);
            if (sgn != tp) active = !active;
            ans += d - prv; sgn = 0;
        }
        else {
            if (active) ans += d - prv;
            if (tp == 0) active = !active;
        }
    }
}

```

```

        else sgn = tp;
    }
    prv = d;
}
return ans*sqrt(dot(B-A, B-A));
}

```

### 3.10 Min Enclosing Circle

```

bool is_colinear(Point a, Point b, Point c) {
    return fabs((b-a).cross(c-a)) < EPS;
}
bool on(Point a, Point b, Point x) {
    LD t = (x-a).Norm()+(x-b).Norm()-(a-b).Norm();
    return fabs(t) < EPS;
}
bool in_circle(const Point& v, const Circle& C) {
    return (v - C.center).Norm() <= C.r + EPS;
}
Circle better(Circle A, Circle B) {
    if (A.r < B.r) return A;
    return B;
}
Circle find_circle(Point a) {return Circle(a,0);}
Circle find_circle(Point a, Point b) {
    return Circle((a+b)/2, (a-b).Norm()/2);
}
Circle find_circle(Point a, Point b, Point c,
    bool force_on = false) {
    if(is_colinear(a,b,c)) {
        if(on(a,b,c))
            return Circle((a+b)/2, (a-b).Norm()/2);
        if(on(a,c,b))
            return Circle((a+c)/2, (a-c).Norm() / 2);
        if(on(c,b,a))
            return Circle((c+b)/2, (c-b).Norm() / 2 );
    }
    Point u = (b-a), v = (c-a);
    Point uperp = u.Rotate90(), vperp = v.Rotate90();
    Point ab = (a+b)/2, ac = (a+c)/2;
    Point ans=InterLineLine(ab,ab+uperp,ac,ac+vperp);
    LD rad = ((ans-a).Norm()+
        (ans-b).Norm()+(ans-c).Norm())/3.01;
    Circle C = Circle(ans,rad);
    if (force_on) return C;
    Circle C_ab = find_circle(a,b);
    Circle C_bc = find_circle(b,c);
    Circle C_ac = find_circle(a,c);
    if(in_circle(c, C_ab)) C = better(C, C_ab);
    if(in_circle(a, C_bc)) C = better(C, C_bc);
    if(in_circle(b, C_ac)) C = better(C, C_ac);
    return C;
}
Circle find_circle(vector<Point> P, int N, int K){
    if (K >= 3)
        return find_circle(P[N-1],P[N-2],P[N-3],true);
    if (N == 1) return find_circle(P[0]);
    if (N == 2) return find_circle(P[0],P[1]);
    int i = rand()%(N-K);
    swap(P[i], P[N-1-K]); swap(P[N-1-K], P[N-1]);
    auto C = find_circle(P, N-1, K);
    swap(P[N-1-K], P[N-1]); swap(P[i], P[N-1-K]);
    if (in_circle(P[i],C)) return C;
    swap(P[i], P[N-1-K]);
}

```

```

C = find_circle(P, N, K+1);
swap(P[i], P[N-1-K]);
return C;
}

```

### 3.11 Minkowski Sum

```

PT dir;
bool half(PT p){
    return cross(dir, p) < 0 ||
        (cross(dir, p) == 0 && dot(dir, p) > 0);
}
bool polarComp(PT p, PT q) {
    return make_tuple(half(p), 0)
        < make_tuple(half(q), cross(p, q));
}
void process(vector<PT> &P) {
    int mnid = 0;
    for (int i=0; i<P.size(); i++)
        if (P[i] < P[mnid])
            mnid = i;
    rotate(P.begin(), P.begin()+mnid, P.end());
}
vector<PT> MinkowskiSum(vector<PT>A, vector<PT>B){
    process(A); process(B);
    int n = A.size(), m = B.size();
    vector<PT> P(n), Q(m);
    for(int i=0; i<n; i++) P[i] = A[(i+1)%n] - A[i];
    for(int i=0; i<m; i++) Q[i] = B[(i+1)%m] - B[i];
    dir = PT(0, -1);
    vector<PT> C(n+m+1);
    merge(P.begin(), P.end(), Q.begin(), Q.end(),
        C.begin()+1, polarComp);
    C[0] = A[0] + B[0];
    for(int i=1; i<C.size(); i++) C[i]=C[i]+C[i-1];
    C.pop_back();
    return C;
}
}

```

### 3.12 Point Rotation Trick

```

struct pnt{
    int x, y, idx;
    bool operator<(const pnt &p)const{
        return pi(x, y) < pi(p.x, p.y);
    }
}a[5005];
struct line{
    int dx, dy, i1, i2;
};
vector<line> v;
int n, rev[5005];
lint p, q;
LL ccw(pnt a, pnt b, pnt c){
    int dx1 = b.x - a.x;
    int dy1 = b.y - a.y;
    int dx2 = c.x - a.x;
    int dy2 = c.y - a.y;
    return abs(1ll * dx1 * dy2 - 1ll * dy1 * dx2);
}
void solve(int c1, int c2, LL l){
    ans = max(ans, ccw(a[c1], a[c2], a[0]));
    ans = max(ans, ccw(a[c1], a[c2], a[n-1]));
}
int main(){
}

```

```

cin >> n;
for(int i=0; i<n;i++) cin >> a[i].x >> a[i].y;
sort(a, a+n);
for(int i=0; i < n;i++) a[i].idx = rev[i] = i;
for(int i=0; i<n; i++)
    for(int j=i+1; j<n; j++)
        v.pb({a[j].x-a[i].x,a[j].y-a[i].y,
               a[i].idx,a[j].idx});

sort(v.begin(), v.end(), [&]
    (const line &a, const line &b){
        LL cw = 1ll*a.dx*b.dy - 1ll * b.dx * a.dy;
        if(cw != 0) return cw > 0;
        return pi(a.i1, a.i2) < pi(b.i1, b.i2);
    });
LL ret = 0;
for(int i=0; i<v.size(); i++){
    int c1 = rev[v[i].i1], c2 = rev[v[i].i2];
    if(c1 > c2) swap(c1, c2);
    solve(c1, c2, p);
    swap(a[c1], a[c2]);
    swap(rev[v[i].i1], rev[v[i].i2]);
}
}

```

### 3.13 Simpson

```

//We divide the integration segment[a;b] into 2n
//equal parts # of steps (already multiplied by 2)
double simpson_integration(double a, double b){
    double h = (b - a) / N;
    double s = f(a) + f(b); // a = x_0 and b = x_2n
    for (int i = 1; i <= N - 1; ++i) {
        double x = a + h * i;
        s += f(x) * ((i & 1) ? 4 : 2);
    }
    s *= h / 3;
    return s;
}

```

### 3.14 Visibility Polygon

```

bool half(PT p) {
    return p.y > 0 || (p.y == 0 && p.x > 0);
}

int compare(PT a, PT b) {
    auto l = make_tuple(half(b), 0);
    auto r = make_tuple(half(a), cross(a, b));
    return l==r ? 0 : (l<r ? -1 : 1);
}

double distance(PT X, PT A, PT B) {
    B = B-A; assert(cross(X, B));
    return sqrt(dot(X, X))*cross(A, B)/cross(X, B);
}

bool compareDis(PT X, PT A, PT B, PT AA, PT BB) {
    B = B-A; BB = BB-AA;
    return 1.0*cross(A, B)/cross(X, B) <
        1.0*cross(AA, BB)/cross(X, BB);
}

pair<double, double> shoot(PT X, double len) {
    double rat = len/sqrt(dot(X, X));
    return make_pair(X.x*rat, X.y*rat);
}

vector<pair<double, double>>

```

```

getVisibilityPolygon(PT Z, vector<PT> &p) {
    for (PT &X: p) X = X-Z;
    int n = p.size(); PT O(0, 0);
    auto comp=[](PT a,PT b){return compare(a,b)<0;};
    map<PT,vector<int>,decltype(comp)> events(comp);
    for (int i=1; i<=n; i++) {
        PT X = p[i-1], Y = p[i%n];
        if (cross(O, X, Y) < 0) swap(X, Y);
        if (compare(X, Y) == 0) continue;
        events[X].push_back(i);events[Y].push_back(-i);
    }
    PT dir, last = events.rbegin() -> first;
    auto comp2 = [&dir, &p, &n](int i, int j){return
        compareDis(dir,p[i-1],p[i%n],p[j-1],p[j%n]);};
    multiset<int, decltype(comp2)> st(comp2);
    vector<bool> open(n+1);
    for (auto pr: events) {
        for (int v: pr.second) if (v>0) open[v] = 1;
        for (int v: pr.second) if (v<0) open[-v] = 0;
    }
    vector<int> pending;
    vector<pair<double, double>> poly;
    for (int i=1; i<=n; i++)
        if (open[i]) pending.push_back(i);
    for (auto pr: events) {
        PT nw = pr.first;
        dir = nw+last;
        for (int i: pending) st.insert(i);
        pending.clear();
        int i = *st.begin();
        poly.push_back(shoot(last,
            distance(last,p[i-1],p[i%n])));
        poly.push_back(shoot(nw,
            distance(nw, p[i-1], p[i%n])));
        for (int i: pr.second) {
            if (i < 0) st.erase(-i);
            else pending.push_back(i);
        }
        last = nw;
    }
    return poly;
}

```

### 3.15 Voronoi

```

LD ccw(Point p, Point q, Point r) {
    return (q-p).cross(r-q);
}

// ax + by = c
struct Line{
    LD a, b, c;
    Point u, d;
    Line(LD a, LD b,LD c):a(a), b(b), c(c) {
        // careful that u, d is not updated here.
    }
    Line(Point u_, Point d_) {
        u = u_, d = d_; //anti-clock dir is the region
        a = d.y, b = -d.x, c = -u.y*d.x + u.x*d.y;
        // ax + by <= c
    }
    bool operator < (const Line &l)const{
        bool flag1 = mp(a, b) > mp(0.0L, 0.0L);
        bool flag2 = mp(l.a, l.b) > mp(0.0L, 0.0L);
    }
}

```

```

if(flag1 != flag2) return flag1 > flag2;
LD t = ccw(Point(0.0L, 0.0L),
    Point(a, b), Point(l.a, l.b));
return dcmp(t) == 0 ? c*hypot(l.a, l.b) <
    l.c * hypot(a, b):t>0;
}

Point slope() { return Point(a, b);}
};

Point cross(Line a, Line b){
    LD det = a.a * b.b - b.a * a.b;
    return Point((a.c * b.b - a.b * b.c) / det,
        (a.a * b.c - a.c * b.a) / det);
}

bool bad(Line a, Line b, Line c){
    if(ccw(Point(0, 0), a.slope(), b.slope()) <= 0)
        return false;
    Point crs = cross(a, b);
    return crs.x * c.a + crs.y * c.b >= c.c;
}

// ax + by <= c;
bool hpi(vector<Line> v, vector<Point> &solution){
    sort(v.begin(), v.end());
    deque<Line> dq;
    for(auto &i : v) {
        if(!dq.empty()&&!dcmp(ccw(Point(0,0),
            dq.back().slope(), i.slope())) continue;
        while(dq.size()>2&&bad(dq[dq.size()-2],
            dq.back(), i)) dq.pop_back();
        while(dq.size()>2&&bad(i,dq[0],dq[1]))
            dq.pop_front();
        dq.pb(i);
    }
    while(dq.size()>2&&bad(dq[dq.size()-2],
        dq.back(),dq[0])) dq.pop_back();
    while(dq.size()>2&&bad(dq.back(),
        dq[0],dq[1])) dq.pop_front();
    vector<Point> tmp;
    for(int i=0; i< dq.size(); i++){
        Line cur = dq[i], nxt = dq[(i+1)%dq.size()];
        if(ccw(Point(0,0),cur.slope(),nxt.slope())
            <= EPS) return false;
        tmp.pb(cross(cur, nxt));
    }
    solution = tmp; return true;
}

int main() {
    int n; cin >> n; vector<Point> P(n);
    for(int i=0;i<n;i++) cin >> P[i].x >> P[i].y;
    LD R = 1e9;
    vector<vector<Point>> voronoi_diagram;
    for(int i = 0; i < n; i++) {
        vector<Line> lines;
        lines.pb(Line(1,0,R)); // x <= R
        lines.pb(Line(-1,0,R)); // x >= -R => -x <= R
        lines.pb(Line(0,1,R)); // y <= R
        lines.pb(Line(0,-1,R)); // y >= -R => -y <= R
        for(int j = 0; j < n; j++) {
            if(P[i] == P[j]) continue;
            Point u=(P[i]+P[j])*0.5,dir = P[j]-P[i];
            Point dir_90 = dir.Rotate90();
            Point v = u + dir_90;
            LD a = dir_90.y, b = -dir_90.x;

```

```

        LD = c = -u.y*dir_90.x + u.x*dir_90.y;
        lines.pb(Line(a,b,c));
    }
    vector<Point> polygon;
    hpi(lines, polygon);
    voronoi_diagram.pb(polygon);
}
}

```

## 4 Graph

### 4.1 BlockCutTree

```

namespace BCT{
    const int mx = 100005; //max(edge,node)
    bool isCutPoint[mx];
    int low[mx], pre[mx], cnt2vcc, used[mx], Timer = 0;
    vector<int> biComp[mx]; int n, m;
    struct Edge{ int v, id; };
    vector<Edge> g[mx]; vector<int> bridges;
    stack<int> stk;
    void init(int n, int m){
        n = _n; m = _m;
        for(int i=1; i<=max(n,m); i++){
            g[i].clear(); biComp[i].clear();
        }
        bridges.clear(); // for bridge
    }
    void addEdge(int u, int v, int id){
        g[u].pb({v,id}); g[v].pb({u,id});
    }
    void dfs(int u, int par){
        pre[u] = ++Timer; low[u] = pre[u]; int chCnt=0;
        for(int i=0; i<g[u].size(); i++){
            int edgeId = g[u][i].id;
            if(used[edgeId]) continue;
            used[edgeId] = true; stk.push(edgeId);
            int v = g[u][i].v;
            if(pre[v]==-1){
                dfs(v, u);
                low[u] = min(low[u], low[v]);
                if(low[v] == pre[v]) bridges.pb(edgeId);
                if(low[v] >= pre[u]){
                    cnt2vcc++;
                    while(stk.size()>0)/*making component*/{
                        biComp[cnt2vcc].pb(stk.top());
                        stk.pop();
                    }
                    if(biComp[cnt2vcc].back()==edgeId) break;
                }
                if(par!=0) isCutPoint[u]=true;
                //checking if non-root
            }
            chCnt++;
        }
        else low[u] = min(low[u], pre[v]);
    }
    if(chCnt > 1 && par==0) isCutPoint[u] = true;
    //checking for root
}
int find2VCC(){
    int i, j; Timer = 0;
    for(i=1; i<=m; i++) used[i] = false;
    for(i=1; i<=n; i++){
        isCutPoint[i] = false; pre[i] = -1;
    }
}

```

```

    }
    cnt2vcc = 0;
    for(i=1; i<=n; i++){
        if(pre[i]==-1) dfs(i,0);
    }
}
struct Edge{
    int u, v, id;
}edge[maxn];
int main(){
    //BCT::addEdge(u,v,i);
    BCT::find2VCC();
    int cntVcc = BCT::cnt2vcc; int ans1;
    unsigned long long int ans2;
    if(cntVcc==1){
        ans1 = 2; ans2 = (n*(n-1))/2LL;
    }
    else{
        ans1 = 0, ans2=1LL;
        for(i=1; i<=cntVcc; i++){
            set<int> nodes;
            for(j=0; j<BCT::biComp[i].size(); j++){
                int id = BCT::biComp[i][j];
                nodes.insert(edge[id].u);
                nodes.insert(edge[id].v);
            }
            set<int>::iterator it = nodes.begin();
            int artCnt = 0;
            while(it!=nodes.end()){
                if(BCT::isCutPoint[*it]) artCnt++;
                it++;
            }
            if(artCnt==1){
                ans1++;
                ans2 += (1LL*(nodes.size() - artCnt));
            }
        }
    }
}

```

### 4.2 Blossom

```

const int N = 2020 + 1;
struct GM { // 1-based Vertex index
    int vis[N], par[N], orig[N], match[N], aux[N], t;
    vector<int> conn[N]; queue<int> Q;
    void addEdge(int u, int v){
        conn[u].push_back(v); conn[v].push_back(u);
    }
    void init(int n){
        N = n; t = 0;
        for(int i=0; i<=n; i++){
            conn[i].clear(); match[i]=aux[i]=par[i]=0;
        }
    }
    void augment(int u, int v){
        int pv = v, nv;
        do{
            pv=par[pv]; nv=match[pv];
            match[pv]=pv; match[pv]=v; v=nv;
        } while(u != pv);
    }
    int lca(int v, int w){

```

```

        ++t;
        while(true){
            if(v){
                if(aux[v] == t) return v;
                aux[v] = t; v = orig[par[match[v]]];
            }
            swap(v, w);
        }
    }
    void blossom(int v, int w, int a){
        while(orig[v] != a){
            par[v] = w; w = match[v];
            if(vis[w] == 1) Q.push(w); vis[w] = 0;
            orig[v] = orig[w] = a; v = par[w];
        }
    }
    bool bfs(int u){
        fill(vis+1, vis+1+N, -1);
        iota(orig+1, orig+1+N, 1);
        Q = queue<int>();
        Q.push(u); vis[u] = 0;
        while(!Q.empty()){
            int v = Q.front(); Q.pop();
            for(int x: conn[v]){
                if(vis[x] == -1){
                    par[x] = v; vis[x] = 1;
                    if(!match[x]) return augment(u, x), true;
                    Q.push(match[x]); vis[match[x]] = 0;
                }
                else if(vis[x] == 0 && orig[v] != orig[x]){
                    int a = lca(orig[v], orig[x]);
                    blossom(x, v, a); blossom(v, x, a);
                }
            }
        }
        return false;
    }
    int Match(){
        int ans = 0;
        vector<int> V(N-1); iota(V.begin(), V.end(), 1);
        shuffle(V.begin(), V.end(), mt19937(0x9494949));
        for(auto x: V) if(!match[x]){
            for(auto y: conn[x]) if(!match[y]){
                match[x] = y; match[y] = x; ++ans; break;
            }
        }
        for(int i=1; i<=N; i++){
            if(!match[i] && bfs(i)) ++ans;
        }
        return ans;
    }
};

```

### 4.3 Directed MST

```

struct Edge{
    int u, v, w; Edge(){ }
    Edge(int a, int b, int c){ u = a, v = b, w = c; }
};
//Directed minimum spanning tree in O(n * m)
//Mks a rooted tree of min weight frm da root node
//Returns -1 if no solution from root
int directed_MST(int n, vector<Edge> E, int root){
    const int INF = (1 << 30) - 30;

```

```

int i, j, k, l, x, y, res = 0;
vector<int> cost(n), parent(n), label(n), comp(n);
for (; ){
    for (i = 0; i < n; i++) cost[i] = INF;
    for (auto e: E){
        if (e.u != e.v && cost[e.v] > e.w){
            cost[e.v] = e.w; parent[e.v] = e.u;
        }
    }
    cost[root] = 0;
    for (i = 0; i < n && cost[i] != INF; i++){
        if (i != n) return -1; // No solution
    }
    for (i = 0, k = 0; i < n; i++) res += cost[i];
    for (i = 0; i < n; i++) label[i] = comp[i] = -1;
    for (i = 0; i < n; i++){
        for(x=i; x!=root && comp[x] == -1; x=parent[x])
            comp[x] = i;
        if (x != root && comp[x] == i){
            for (k++; label[x] == -1; x=parent[x])
                label[x] = k-1;
        }
    }
    if (k == 0) break;
    for (i = 0; i < n; i++){
        if (label[i] == -1) label[i] = k++;
    }
    for (auto &e: E){
        x = label[e.u], y = label[e.v];
        if (x != y) e.w -= cost[e.v];
        e.u = x, e.v = y;
    }
    root = label[root], n = k;
}
return res;
}

```

#### 4.4 Dominator Tree

```

struct ChudirBhai {
    int n, T; VVI g, tree, rg, bucket;
    VI sdom, par, dom, dsu, label, arr, rev;
    ChudirBhai(int n): n(n), g(n+1), tree(n+1), rg(n+1),
        bucket(n+1), sdom(n+1), par(n+1), dom(n+1),
        dsu(n+1), label(n+1), arr(n+1), rev(n+1), T(0) {
        for(int i=1; i<=n; i++)
            sdom[i] = dom[i] = dsu[i] = label[i] = i;
    }
    void addEdge(int u, int v) {g[u].push_back(v);}
    void dfs0(int u) {
        T++; arr[u] = T, rev[T] = u;
        label[T] = T, sdom[T] = T, dsu[T] = T;
        for(int i = 0; i < g[u].size(); i++) {
            int w = g[u][i];
            if(!arr[w]) dfs0(w), par[arr[w]] = arr[u];
            rg[arr[w]].push_back(arr[u]);
        }
    }
    int Find(int u, int x = 0) {
        if(u == dsu[u]) return x? -1: u;
        int v = Find(dsu[u], x+1);
        if(v < 0) return u;
        if(sdom[label[dsu[u]]] < sdom[label[u]])
            label[u] = label[dsu[u]];
    }
}

```

```

dsu[u] = v;
return x? v: label[u];
}
void Union(int u, int v) { dsu[v] = u; }
VVI buildAndGetTree(int s) {
    dfs0(s);
    for(int i = n; i >= 1; i--) {
        for(int j = 0; j < rg[i].size(); j++){
            sdom[i] = min(sdom[i], sdom[Find(rg[i][j])]);
            if(i > 1) bucket[sdom[i]].push_back(i);
            for(int j = 0; j < bucket[i].size(); j++) {
                int w = bucket[i][j], v = Find(w);
                if(sdom[v] == sdom[w]) dom[w] = sdom[w];
                else dom[w] = v;
            }
            if(i > 1) Union(par[i], i);
        }
        for(int i = 2; i <= n; i++) {
            if(dom[i] != sdom[i]) dom[i] = dom[dom[i]];
            tree[rev[i]].push_back(rev[dom[i]]);
            tree[rev[dom[i]]].push_back(rev[i]);
        }
        return tree;
    }
};

```

#### 4.5 EulerPath

```

int c[maxn], d[maxn];
map<int, multiset<int>> g; map<int, int> vis;
void dfs1(int u) {
    vis[u] = 1;
    for( auto v : g[u] )
        if( vis.find(v) == vis.end() ) dfs1(v);
}
//just call dfs2 with the node you want to start
//your path at first you need to make sure,
//the graph is connected and euler path exists
vector<int> ans;
void dfs2(int u) {
    while( (int)g[u].size() != 0 ) {
        int v = *g[u].begin();
        g[u].erase(g[u].find(v));
        g[v].erase(g[v].find(u)); dfs2(v);
    }
    ans.pb(u);
}
int main() {
    int n; scanf("%d", &n);
    for(int i=1; i<n; i++) scanf("%d", &c[i]);
    for(int i=1; i<n; i++) scanf("%d", &d[i]);
    for(int i=1; i<n; i++) {
        if( c[i] > d[i] ) {
            printf("-1\n"); return 0;
        }
        g[c[i]].insert(d[i]); g[d[i]].insert(c[i]);
    }
    int src = c[1], cnt = 0;
    for( auto it : g ) {
        if( (int)it.second.size() & 1 ) {
            cnt++; src = it.first;
        }
    }
}

```

```

dfs1( src );
if(vis.size() != g.size() || (cnt != 0 && cnt != 2)) {
    printf("-1\n");
    return 0;
}
//call for printing euler path
dfs2(src);
for(int i=0; i<ans.size(); i++){
    printf("%d", ans[i]);
    if( i == (int)ans.size() - 1 ) printf("\n");
    else printf(" ");
}
}

```

#### 4.6 Flow with demands

**Finding an arbitrary flow** Consider flow networks, where we additionally require the flow of each edge to have a certain amount, i.e. we bound the flow from below by a **demand** function  $d(e)$ :

$$d(e) \leq f(e) \leq c(e)$$

So next each edge has a minimal flow value, that we have to pass along the edge.

We make the following changes in the network. We add a new source  $s'$  and a new sink  $t'$ , a new edge from the source  $s'$  to every other vertex, a new edge for every vertex to the sink  $t'$ , and one edge from  $t$  to  $s$ . Additionally we define the new capacity function  $c'$  as:

- $c'((s', v)) = \sum_{u \in V} d((u, v))$  for each edge  $(s', v)$ .
- $c'((v, t')) = \sum_{w \in V} d((v, w))$  for each edge  $(v, t')$ .
- $c'((u, v)) = c((u, v)) - d((u, v))$  for each edge  $(u, v)$  in the old network.
- $c'((t, s)) = \infty$

If the new network has a saturating flow (a flow where each edge outgoing from  $s'$  is completely filled, which is equivalent to every edge incoming to  $t'$  is completely filled), then the network with demands has a valid flow, and the actual flow can be easily reconstructed from the new network. Otherwise there doesn't exist a flow that satisfies all conditions. Since a saturating flow has to be a maximum flow, it can be found by any maximum flow algorithm.



**Minimal flow** Note that along the edge  $(t, s)$  (from the old sink to the old source) with the capacity  $\infty$  flows the entire flow of the corresponding old network. I.e. the capacity of this edge effects the flow value of the old network. By giving this edge a sufficient large capacity (i.e.  $\infty$ ), the flow of the old network is unlimited. By limiting this edge by smaller capacities, the flow value will decrease. However if we limit this edge by a too small value, than the network will not have a saturated solution, e.g. the corresponding solution for the original network will not satisfy the demand of the edges. Obviously here can use a binary search to find the lowest value with which all constraints are still satisfied. This gives the minimal flow of the original network.

#### 4.7 Hopcroft Karp

```
const int maxN = 50000+5, maxM = 50000+5;
struct HopcroftKarp {
    int n, vis[maxN], lev[maxN], ml[maxN], mr[maxM];
    vector<int> edge[maxN]; //edges for lft art only
    HopcroftKarp(int n) : n(n) { //n=nodes in lft prt
        for (int i = 1; i <= n; ++i) edge[i].clear();
    }
    void add(int u, int v) {edge[u].push_back(v);}
    bool dfs(int u) {
        vis[u] = true;
        for(auto it=edge[u].begin(); it!=edge[u].end(); ++it){
            int v = mr[*it];
            if (v==-1 || (!vis[v] && lev[u] < lev[v] &&
                dfs(v))) {
                ml[u] = *it; mr[*it] = u; return true;
            }
        }
        return false;
    }
    int matching() { // n for left
        memset(vis, 0, sizeof vis);
        memset(lev, 0, sizeof lev);
        memset(ml, -1, sizeof ml);
        memset(mr, -1, sizeof mr);
        for (int match = 0;;) {
            queue<int> que;
            for (int i = 1; i <= n; ++i) {
                if (ml[i] == -1) lev[i] = 0, que.push(i);
                else lev[i] = -1;
            }
            while (!que.empty()) {
                int u = que.front();
                que.pop();
                for (auto it = edge[u].begin();
                    it != edge[u].end(); ++it) {
                    int v=mr[*it];
                    if (v!=-1 && lev[v] < 0)
                        lev[v] = lev[u]+1, que.push(v);
                }
            }
            for (int i = 1; i <= n; ++i) vis[i] = false;
            int d = 0;
            for (int i=1; i<=n; ++i)
                if (ml[i]==-1 && dfs(i)) ++d;
            if (d == 0) return match;
            match += d;
        }
    }
};
```

```
    }
    for (int i = 1; i <= n; ++i) vis[i] = false;
    int d = 0;
    for (int i=1; i<=n; ++i)
        if (ml[i]==-1 && dfs(i)) ++d;
    if (d == 0) return match;
    match += d;
}
};
```

#### 4.8 Hungarian Algorithm

```
namespace wm{
    bool vis[N]; int U[N], V[N], P[N];
    int way[N], minv[N], match[N], ar[N][N];
    //n=no of row, m=no of col, 1 based,
    //flag=MAXIMIZE/MINIMIZE
    //match[i] = the column to which row i is matched
    int hungarian(int n, int m, int mat[N][N], int flag){
        clr(U), clr(V), clr(P), clr(ar), clr(way);
        for (int i = 1; i <= n; i++){
            for (int j = 1; j <= m; j++){
                ar[i][j] = mat[i][j];
                if (flag == MAXIMIZE) ar[i][j] = -ar[i][j];
            }
        }
        if (n > m) m = n;
        int i, j, a, b, c, d, r, w;
        for (i = 1; i <= n; i++){
            P[0] = i, b = 0;
            for (j=0; j<=m; j++) minv[j]=inf, vis[j] = 0;
            do{
                vis[b] = true; a = P[b], d = 0, w = inf;
                for (j = 1; j <= m; j++){
                    if (!vis[j]){
                        r = ar[a][j] - U[a] - V[j];
                        if (r < minv[j]) minv[j] = r, way[j]=b;
                        if (minv[j] < w) w = minv[j], d = j;
                    }
                }
                for (j = 0; j <= m; j++){
                    if (vis[j]) U[P[j]] += w, V[j] -= w;
                    else minv[j] -= w;
                }
                b = d;
            } while (P[b] != 0);
            do{
                d = way[b]; P[b] = P[d], b = d;
            } while (b != 0);
        }
        for (j = 1; j <= m; j++) match[P[j]] = j;
        return (flag == MINIMIZE) ? -V[0] : V[0];
    }
}
```

#### 4.9 Maxflow

```
/* 0 based for directed graphs */
const LL INF = (~0ULL) >> 1, N = 30010;
namespace flow{
    struct Edge{
        int u, v; LL cap, flow;
```

```
Edge(){
    Edge(int a, int b, LL c, LL f){
        u = a, v = b, cap = c, flow = f;
    }
};
vector<int> adj[N]; vector<Edge> E;
int n, s, t, ptr[N], len[N], dis[N], Q[N];
void init(int nodes, int src, int sink){
    clr(len); E.clear();
    n = nodes, s = src, t = sink;
    for(int i=0; i<N; i++) adj[i].clear();
}
void addEdge(int a, int b, LL c){
    adj[a].push_back(E.size());
    E.push_back(Edge(a, b, c, 0));
    len[a]++; adj[b].push_back(E.size());
    E.push_back(Edge(b, a, 0, 0)); len[b]++;
}
bool bfs(){
    int i, j, k, id, f = 0, l = 0;
    memset(dis, -1, sizeof(dis[0]) * n);
    dis[s] = 0, Q[l++] = s;
    while (f < l && dis[t] == -1){
        i = Q[f++];
        for (k = 0; k < len[i]; k++){
            id = adj[i][k];
            if (dis[E[id].v] == -1 &&
                E[id].flow < E[id].cap){
                Q[l++] = E[id].v;
                dis[E[id].v] = dis[i] + 1;
            }
        }
    }
    return (dis[t] != -1);
}
LL dfs(int i, LL f){
    if (i == t || !f) return f;
    while (ptr[i] < len[i]){
        int id = adj[i][ptr[i]];
        if (dis[E[id].v] == dis[i] + 1){
            LL ff = E[id].cap - E[id].flow;
            LL x = dfs(E[id].v, min(f, ff));
            if (x) {
                E[id].flow += x, E[id^1].flow -= x;
                return x;
            }
        }
        ptr[i]++;
    }
    return 0;
}
LL dinic(){
    LL res = 0;
    while (bfs()){
        memset(ptr, 0, n * sizeof(ptr[0]));
        while (LL f = dfs(s, INF)) {
            res += f;
        }
    }
    return res;
}
```

## 4.10 Mincost Maxflow

```

//0 Based, dir graphs (for undir add two diredge)
namespace mcmf{
    const int N = 1000010; const LL INF = 1LL << 60;
    LL cap[N], flow[N], cost[N], dis[N];
    int n,m,s,t,Q[10000010];
    int adj[N],link[N],last[N],from[N],vis[N];
    void init(int nodes, int source, int sink){
        m = 0, n = nodes, s = source, t = sink;
        for (int i = 0; i <= n; i++) last[i] = -1;
    }
    void addEdge(int u,int v,LL c,LL w){
        adj[m]=v, cap[m]=c, flow[m]=0, cost[m]=+w,
        link[m]=last[u], last[u]=m++;
        adj[m]=u, cap[m]=0, flow[m]=0, cost[m]=-w,
        link[m]=last[v], last[v]=m++;
    }
    bool spfa(){
        int i, j, x, f = 0, l = 0;
        for (i=0; i<=n; i++) vis[i] = 0, dis[i] = INF;
        dis[s] = 0, Q[l++] = s;
        while (f < l){
            i = Q[f++];
            for (j = last[i]; j != -1; j = link[j]){
                if (flow[j] < cap[j]){
                    x = adj[j];
                    if (dis[x] > dis[i] + cost[j]){
                        dis[x] = dis[i] + cost[j], from[x] = j;
                        if (!vis[x]){
                            vis[x] = 1;
                            if (f && rand() & 7) Q[--f] = x;
                            else Q[l++] = x;
                        }
                    }
                }
            }
            vis[i] = 0;
        }
        return (dis[t] != INF);
    }
    pair <LL, LL> solve(){
        int i, j; LL mincost = 0, maxflow = 0;
        while (spfa()){
            LL aug = INF;
            for(i=t,j=from[i];i!=s;i=adj[j^1],j=from[i])
                aug = min(aug, cap[j]-flow[j]);
            for(i=t,j=from[i];i!=s;i=adj[j^1],j=from[i])
                flow[j] += aug, flow[j ^ 1] -= aug;
            maxflow += aug, mincost += aug * dis[t];
        }
        return make_pair(mincost, maxflow);
    }
}

```

## 4.11 SCC + 2SAT

/\*at first take a graph of size  $2*n$  (for each variable two nodes). for each clause of type (a or b), add two diredge  $a \rightarrow b$  and  $!b \rightarrow a$ . if both  $x_i$  and  $!x_i$  is in same connected component for some  $i$ , then this equations are unsatisfiable. Otherwise there is a solution. Assume,  $f$  is satisfiable. Now we want to give values to each var in order to

satisfy  $f$ . It can be done with a top sort of vertices of the graph we made. If  $!x_i$  is after  $x_i$  in topological sort,  $x_i$  should be FALSE. It should be TRUE otherwise. say we have equation with three var  $x_1, x_2, x_3$ . ( $x_1$  or  $!x_2$ ) and ( $x_2$  or  $x_3$ ) = 1. so we add  $x_1, x_2, x_3$  and  $x_4$  (as  $!x_1$ ),  $x_5$  ( $!x_2$ ) and  $x_6$  ( $!x_3$ ). Add edge  $x_4 \rightarrow x_2, x_2 \rightarrow x_1, x_5 \rightarrow x_3, x_6 \rightarrow x_2$ .

you need to pass array to the function findSCC, in which result will be returned every node will be given a number, for nodes of a single connected component the number will be same this number representing nodes will be topsorted\*/

```

class SCC{
public:
    vector<int> *g1, *g2; int maxNode, *vis1, *vis2;
    stack<int> st;
    SCC(int MaxNode){
        maxNode = MaxNode; vis1 = new int[maxNode+2];
        vis2 = new int[maxNode+2];
        g1 = new vector<int>[maxNode+2];
        g2 = new vector<int>[maxNode+2];
    }
    void addEdge(int u,int v){
        g1[u].push_back(v); g2[v].push_back(u);
    }
    void dfs1(int u){
        if(vis1[u]==1) return; vis1[u]=1;
        for(int i=0; i<g1[u].size(); i++) dfs1(g1[u][i]);
        st.push(u); return;
    }
    void dfs2(int u, int cnt, int *ans){
        if(vis2[u]==1) return; vis2[u] = 1;
        for(int i=0; i<g2[u].size(); i++)
            dfs2(g2[u][i], cnt, ans);
        ans[u] = cnt;
    }
    int findSCC(int *ans){
        for(int i=1; i<=maxNode; i++) vis1[i] = 0;
        for(int i=1; i<=maxNode; i++)
            if(vis1[i]==0) dfs1(i);
        int cnt = 0;
        for(int i=1; i<=maxNode; i++) vis2[i] = 0;
        while(!st.empty()){
            int u = st.top();
            if(vis2[u]==0) {++cnt; dfs2(u, cnt, ans);}
            st.pop();
        }
        for(int i=1; i<=maxNode; i++) {
            g1[i].clear(); g2[i].clear();
        }
        delete vis1; delete vis2; return cnt;
    }
};

```

## 5 Math

### 5.1 FFT

```

struct FFT {
    struct node {
        double x,y;
        node() {}
        node(double a, double b): x(a), y(b) {}
    }
};

```

```

node operator+(node a) const {
    return node(x+a.x, y+a.y);
}
node operator-(node a) const {
    return node(x-a.x, y-a.y);
}
node operator*(node a) const {
    return node(x*a.x-y*a.y, x*a.y+a.x*y);
}
};

int M; vector<node> A, B, w[2]; vector<int> rev;
const long double pi = acos(-1);
void init(int n) {
    M = 1; while(M < n) M <= 1; M <= 1;
    A.resize(M); B.resize(M); w[0] = w[1] = rev = B;
    for (int i=0; i<M; i++) {
        int j=i, y=0;
        for (int x=1; x<M; x<=1, j>=1) (y<=1) += j&1;
        rev[i]=y;
    }
    for (int i=0; i<M; i++) {
        w[0][i] = node( cos(2*pi*i/M), sin(2*pi*i/M));
        w[1][i] = node( cos(2*pi*i/M), -sin(2*pi*i/M));
    }
}

void ftransform( vector<node> &A, int p ) {
    for (int i=0; i<M; i++)
        if (i<rev[i]) swap(A[i], A[rev[i]]);
    for (int i=1; i<M; i<=1)
        for (int j=0, t=M/(i<=1); j<M; j+=i<=1)
            for (int k=0, l=0; k<i; k++, l+=t) {
                node x=w[p][l]*A[i+j+k], y=A[j+k];
                A[j+k]=y+x; A[j+k+i]=y-x;
            }
    if (p) for (int i=0; i<M; i++) A[i].x/=M;
}

void multiply(VI &P, VI &Q, VI &res) {
    init(max(P.size(), Q.size()));
    for(int i=0; i<M; i++)
        A[i].x=A[i].y=B[i].x=B[i].y=0;
    for(int i = 0; i < P.size(); i++) A[i].x = P[i];
    for(int i = 0; i < Q.size(); i++) B[i].x = Q[i];
    ftransform(A, 0); ftransform(B, 0);
    for (int k=0; k<M; k++) A[k] = A[k]*B[k];
    ftransform(A, 1);
    res.resize(M);
    for (int i=0; i<M; i++) res[i] = round(A[i].x);
}

//use long double in fft if RT >= 13
const int RT = 15; //Upto M <= 4^RT
vector<LL> anymod(vector<LL> &a, vector<LL> &b, LL mod){
    init(max(a.size(), b.size()));
    vector<node> al(M), ar(M), bl(M), br(M);
    for (int i=0; i<a.size(); i++) {
        LL k = a[i]%mod; al[i] = node(k >> RT, 0);
        ar[i] = node(k & ((1<<RT)-1), 0);
    }
    for (int i=0; i<b.size(); i++) {
        LL k = b[i]%mod; bl[i] = node(k >> RT, 0);
        br[i] = node(k & ((1<<RT)-1), 0);
    }
    ftransform(al, 0); ftransform(ar, 0);
    ftransform(bl, 0); ftransform(br, 0);
    for (int i=0; i<M; i++) {
        node ll = al[i] * bl[i], lr = ar[i] * br[i];
    }
}

```

```

    node rl = ar[i] * bl[i], rr = ar[i] * br[i];
    al[i] = ll; ar[i] = lr;
    bl[i] = rl; br[i] = rr;
}
ftransform(al, true); ftransform(ar, true);
ftransform(bl, true); ftransform(br, true);
vector<LL> ans(M);
for (int i=0; i<M; i++) {
    LL right = round(br[i].x);
    right %= mod;
    LL mid=round(round(bl[i].x) + round(ar[i].x));
    mid = ((mid%mod)<<RT)%mod;
    LL left=round(al[i].x);
    left = ((left%mod)<<(2*RT))%mod;
    ans[i] = (left+mid+right)%mod;
}
return ans;
};

```

## 5.2 FWHT

```

void FWHT(vector<LL> &p, bool inv) {
    int n = p.size(); assert((n&(n-1))==0);
    for (int len=1; 2*len<=n; len <= 1) {
        for (int i = 0; i < n; i += len+len){
            for (int j = 0; j < len; j++) {
                LL u = p[i+j], v = p[i+len+j];
                ///XOR p[i+j]=u+v; p[i+len+j]=u-v;
                ///OR if(!inv) p[i+j]=v, p[i+len+j]=u+v;
                ///OR else p[i+j]=-u+v, p[i+len+j]=u;
                ///AND if(!inv) p[i+j]=u+v, p[i+len+j]=u;
                ///AND else p[i+j]=v, p[i+len+j]=u-v;
            }
        }
    }
    ///XOR if(inv) for(int i=0;i<n;i++) p[i]/=n;
}

vector<LL> convo(vector<LL> a, vector<LL> b) {
    int n = 1, sz = max(a.size(), b.size());
    while(n<sz) n*=2;
    a.resize(n); b.resize(n); vector<LL> res(n, 0);
    FWHT(a, 0); FWHT(b, 0);
    for(int i=0;i<n;i++) res[i] = a[i] * b[i];
    FWHT(res, 1);
    return res;
}

```

## 5.3 Gaussian Elimination

```

///n = no of eqn, m = no of var, ar[i][m] = rhs
///returns -1 if no sol, else no of free variables
int gauss(int n,int m,double **ar,VD&res){
    res.assign(m, 0); vector<int> pos(m, -1);
    int i, j, k, l, p, free_var = 0;
    for (j = 0, i = 0; j < m && i < n; j++){
        for (k = i, p = i; k < n; k++){
            if (abs(ar[k][j]) > abs(ar[p][j])) p = k;
        }
        if (abs(ar[p][j]) > EPS){
            pos[j] = i;
            for (l = j; l <= m; l++)
                swap(ar[p][l], ar[i][l]);
            for (k = 0; k < n; k++){

```

```

                if (k != i){
                    double x = ar[k][j] / ar[i][j];
                    for (l=j; l<=m; l++)
                        ar[k][l] -= (ar[i][l] * x);
                }
                i++;
            }
        }
        for (i = 0; i < m; i++){
            if (pos[i] == -1) free_var++;
            else res[i] = ar[pos[i]][m] / ar[pos[i]][i];
        }
        for (i = 0; i < n; i++) {
            double val = 0.0;
            for (j = 0; j < m; j++)
                val += (res[j] * ar[i][j]);
            if (abs(val - ar[i][m]) > EPS) return -1;
        }
        return free_var;
}

int gauss(int n,int m, bitset<MAXCOL>ar[MAXROW],
           bitset<MAXCOL>&res){
    res.reset(); vector<int>pos(m, -1);
    int i, j, k, l, v, p, free_var = 0;
    for (j = 0, i = 0; j < m && i < n; j++){
        for (k = i, p = i; k < n; k++){
            if (ar[k][j]) {p = k; break;}
        }
        if (ar[p][j]){
            pos[j] = i; swap(ar[p], ar[i]);
            for (k = 0; k < n; k++){
                if (k != i && ar[k][j]) ar[k] ^= ar[i];
            }
            i++;
        }
    }
    for (i = 0; i < m; i++){
        if (pos[i] == -1) free_var++;
        else res[i] = ar[pos[i]][m];
    }
    for (i = 0; i < n; i++) {
        for (j=0, v=0; j<m; j++) v^=(res[j]&ar[i][j]);
        if (v != ar[i][m]) return -1;
    }
    return free_var;
}

```

## 5.4 Linear sieve

```

vector<int> pr; int lp[N+1]; //lowest prime factor
void sieve() {
    for (int i=2; i<N; ++i) {
        if (lp[i] == 0) {lp[i] = i; pr.push_back(i);}
        for (int j=0; j<pr.size() && i*pr[j]<N; ++j)
            lp[i * pr[j]] = pr[j];
    }
}

```

## 5.5 NTT

```

struct NTT {
    vector<int>A, B, w[2], rev;

```

```

    int P, M, G;
    NTT(int mod, int g) {P = mod; G = g;}
    int Pow(int a, int b) {
        int res=1;
        for (;b; b>>=1,a=a*1LL*a%P)
            if (b&1) res=res*1LL*a%P;
        return res;
    }
    void init( int n ) {
        for (M=1; M<n; M<=1); M<=1;
        A.resize(M); B.resize(M); w[0]=w[1]=rev=B;
        for (int i=0; i<M; i++) {
            int x=i, &y=rev[i];
            y=0;
            for (int k=1; k<M; k<=1,x>>=1)(y<=1)|=x&1;
        }
        int x=Pow(G, (P-1)/M), y=Pow(x, P-2);
        w[0][0]=w[1][0]=1;
        for (int i=1; i<M; i++) {
            w[0][i]=w[0][i-1]*1LL*x%P;
            w[1][i]=w[1][i-1]*1LL*y%P;
        }
    }
    void ntransform(vector<int> &a, int f) {
        for (int i=0; i<M; i++)
            if (i<rev[i]) swap(a[i], a[rev[i]]);
        for (int i=1; i<M; i<=1)
            for (int j=0,t=M/(i<1); j<M; j+=i<1) {
                for (int k=0,l=0; k<i; k++,l+=t) {
                    int x=a[j+k+i]*1ll*w[f][l]%P, y=a[j+k];
                    a[j+k+i]=y-x<0?y-x+P:y-x;
                    a[j+k]=y+x>P?y+x-P:y+x;
                }
            }
        if (f) {
            int x=Pow(M, P-2);
            for (int i=0; i<M; i++) a[i]=a[i]*1ll*x%P;
        }
    }
    void multiply(VI &X, VI &Y, VI &res) {
        init(max(X.size(), Y.size()));
        for (int i = 0; i < M; i++) A[i]=B[i]=0;
        for (int i = 0; i < X.size(); i++) A[i]=X[i];
        for (int i = 0; i < Y.size(); i++) B[i]=Y[i];
        ntransform(A,0); ntransform(B,0);
        res.clear(); res.resize(M);
        for (int i=0;i<M;i++) res[i]=A[i]*1LL*B[i]%P;
        ntransform(res,1);
    }
};

```

## 5.6 Number Theory

```

LL gcd(LL u, LL v) {
    if (u == 0) return v; if (v == 0) return u;
    int shift = __builtin_ctzll(u | v);
    u >>= __builtin_ctzll(u);
    do {
        v >>= __builtin_ctzll(v);
        if (u > v) swap(u, v);
        v = v - u;
    } while (v);
    return u << shift;
}

LL lcm(LL a, LL b) {return (a/gcd(a, b))*b;}

```

```

LL power(LL a, LL b, LL m) {
    a = (a%m+m)%m; LL ans = 1;
    while (b) {
        if (b & 1) ans = (ans*a)%m;
        a = (a*a)%m;
        b >>= 1;
    }
    return ans;
}
//returns g = gcd(a, b); finds x, y st d = ax+ by
LL egcd(LL a, LL b, LL &x, LL &y) {
    LL xx = y = 0; LL yy = x = 1;
    while (b) {
        LL q = a/b;
        LL t = b; b = a%b; a = t;
        t = xx; xx = x-q*xx; x = t;
        t = yy; yy = y-q*yy; y = t;
    }
    return a;
}
//Solves ax=b(mod m)
vector<LL>SolveCongruence(LL a,LL b,LL m){
    LL x, y, g = egcd(a, m, x, y); vector<LL> ans;
    if (b%g == 0) {
        x = (x*(b/g))%m; if (x<0) x+=m;
        for (LL i=0; i<g; i++) {
            ans.push_back(x); x=(x+m/g)%m;
        }
    }
    return ans;
}
LL inverse(LL a, LL m) {
    LL x, y, g = egcd(a, m, x, y);
    if (g > 1) return -1;
    return (x%m+m)%m;
}
//find z st z%m1=r1,z%m2=r2. Here, z is unique mod
//M=lcm(m1,m2), on failure, M=-1,
PLL CRT(LL m1, LL r1, LL m2, LL r2) {
    LL s, t, g = egcd(m1, m2, s, t);
    if (r1%g != r2%g) return PLL(0, -1);
    LL M = m1*m2;
    LL ss = ((s*r2)%m2)*m1, tt = ((t*r1)%m1)*m2;
    LL ans = ((ss+tt)%M+M)%M;
    return PLL(ans/g, M/g);
}
PLL CRT(const vector<LL> &m, const vector<LL> &r) {
    PLL ans = PLL(r[0], m[0]);
    for (LL i = 1; i < m.size(); i++) {
        ans = CRT(ans.second, ans.first, m[i], r[i]);
        if (ans.second == -1) break;
    }
    return ans;
}
//computes x and y such that ax + by = c
bool LinearDiophantine(LL a,LL b,LL c,LL &x,LL &y){
    if (!a && !b) {x=y=0; return !c;}
    if (!a) {x=0;y=c/b; return !(c%b);}
    if (!b) {x=c/a;y=0; return !(c%a);}
    LL g = gcd(a, b);
    x=c/g*inverse(a/g, b/g); y=(c-a*x)/b;
    return !(c%g);
}

```

```

// Min sol to a^x = b (mod M),use unmap for speed
int DiscreteLog(int a, int b, int M) {
    map<int, int> id; LL cur=1, RT=sqrt(M)+5;
    for (int i=0;i<RT;i++) id[cur]=i,cur=(cur*a)%M;
    int pp = power(cur, M-2, M);
    cur = b;
    for (int i=0; i*RT<M; i++) {
        auto it = id.find(cur);
        if (it != id.end()) return i*RT+it->second;
        cur = (cur*pp)%M;
    }
    return -1;
}

```

## 5.7 Pollard Rho

```

LL mult(LL a, LL b, LL mod) {
    assert(b < mod && a < mod);
    long double x = a;
    uint64_t c = x * b / mod;
    int64_t r = (int64_t)(a*b-c*mod) % (int64_t)mod;
    return r < 0 ? r + mod : r;
}
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;
}
vector<LL>bases = {2,325,9375,28178,450775,9780504,
1795265022}; //2, 13, 23, 1662803 for 10^12
bool miller_rabin(LL n) {
    if (n<2) return 0; if (n%2==0) return n==2;
    LL u = n-1; int t = 0;
    while(u%2==0) u/=2, t++; // n-1 = u*2^t
    for (LL v: bases) {
        LL a = v%(n-1) + 1;
        if(witness(a, n, u, t)) return 0;
    }
    return 1;
}
mt19937_64 rng(7852365);
//returns n if prime or 1, or proper divisor of n
LL pollard_rho(LL n) {
    if (n==1) return 1; if (n%2==0) return 2;
    if (miller_rabin(n)) return n;
    while (true) {
        LL x=uniform_int_distribution<LL>(1,n-1)(rng);
        LL y = 2, res = 1;
        for (int sz=2; res==1; sz*=2) {
            for (int i=0; i<sz && res<=1; i++) {
                x = mult(x, x, n) + 1;

```

```

                res = gcd(abs(x-y), n);
            }
            y = x;
        }
        if (res!=0 && res!=n) return res;
    }
}

```

## 5.8 Prime Counting Function

```

#define MAXN 500
#define MAXM 100010
#define MAXP 666666
#define MAX 10000010
#define chkbit(ar, i) \
    (((ar[(i) >> 6] & (1 << (((i) >> 1) & 31))))
#define setbit(ar, i) \
    (((ar[(i) >> 6] |= (1 << (((i) >> 1) & 31))))
#define isprime(x)\
    (((x)&&((x)&1)&&(!chkbit(ar,(x))))||((x)==2))
namespace pcf{
    long long dp[MAXN][MAXM];
    unsigned int ar[(MAX>>6)+5] = {0};
    int len=0, primes[MAXP], counter[MAX];
    void Sieve(){
        setbit(ar,0), setbit(ar,1);
        for (int i=3;(i*i)<MAX;i++,i++){
            if(!chkbit(ar, i)){
                int k=i<<1;
                for(int j=(i*i);j<MAX;j+=k) setbit(ar,j);
            }
        }
        for(int i=1;i<MAX;i++){
            counter[i]=counter[i - 1];
            if(isprime(i)) primes[len++]=i,counter[i]++;
        }
    }
    void init(){
        Sieve();
        for(int n=0;n<MAXN;n++){
            for(int m=0;m<MAXM;m++){
                if(!n) dp[n][m]=m;
                else dp[n][m] =
                    dp[n-1][m]-dp[n-1][m/primes[n-1]];
            }
        }
    }
    LL phi(LL m,int n){
        if(n==0) return m;
        if(primes[n-1]>=m) return 1;
        if(m<MAXM && n<MAXN) return dp[n][m];
        return phi(m,n-1) - phi(m/primes[n-1],n-1);
    }
    LL Lehmer(long long m){
        if(m<MAX) return counter[m];
        LL w,res=0;
        int i,a,s,c,x,y;
        s=sqrt(0.9+m), y=c=cbrt(0.9+m);
        a=counter[y], res=phi(m,a)+a-1;
        for(i=a;primes[i]<=s;i++) res =
            res-Lehmer(m/primes[i])+Lehmer(primes[i])-1;
        return res;
    }
}

```



## 5.9 Primitive Root

```

/** Find primitive root of p assuming p is prime.
if not, we must add calculation of phi(p)
Complexity : O(Ans * log (phi(n)) * log n +sqrt(p))
Returns -1 if not found
*/
int primitive_root(int p) {
    vector<int> factor; int phi = p-1, n = phi;
    for (int i=2; i*i<=n; ++i)
        if (n%i == 0) {
            factor.push_back(i);
            while (n%i==0) n/=i;
        }
    if (n>1) factor.push_back(n);
    for (int res=2; res<=p; ++res) {
        bool ok = true;
        for (int i=0; i<factor.size() && ok; ++i)
            ok &= power(res, phi/factor[i], p) != 1;
        if (ok) return res;
    }
    return -1;
}

int nttdata(int mod,int &root,int &inv, int &pw) {
    int c = 0, n = mod-1; while (n%2==0) c++, n/=2;
    pw = (mod-1)/n; int g = primitive_root(mod);
    root = power(g, n, mod);
    inv = power(root, mod-2, mod);
    return c;
}

```

## 5.10 Stern Brocot Tree

```

//finds x/y with min y st: L <= (x/y) < R
pair<LL,LL>solve(LL L, LL R){
    pair<LL, LL> l(0, 1), r(1, 1);
    if(L==0.0) return l; // corner case
    while(true) {
        pair<int, int> m(l.x+r.x, l.y+r.y);
        if(m.x<L*m.y){ // move to the right
            LL kl=1, kr=1;
            while(l.x+kr*r.x <= L*(l.y+kr*r.y)) kr*=2;
            while(kl!=kr){
                LL km = (kl+kr)/2;
                if(l.x+km*r.x < L*(l.y+km*r.y)) kl=km+1;
                else kr=km;
            }
            l={l.x+(kl-1)*r.x,l.y+(kl-1)*r.y};
        }
        else if(m.x>=R*m.y){ //move to left
            LL kl=1, kr=1;
            while(r.x+kr*l.x>=R*(r.y+kr*l.y)) kr*=2;
            while(kl!=kr){
                LL km = (kl+kr)/2;
                if(r.x+km*l.x>=R*(r.y+km*l.y)) kl = km+1;
                else kr = km;
            }
            r={r.x+(kl-1)*l.x,r.y+(kl-1)*l.y};
        }
        else return m;
    }
}

```

## 5.11 Sum of Floors

```

//Finds sum of floor((p*i+r)/q) from i = 0 to n-1
LL findSum(LL n, LL p, LL r, LL q) {
    if (p == 0) return (r / q) * n;
    if (p>=q||r>=q) return findSum(n,p%q,r%q,q) +
        ((p/q)*(n-1)+2*(r/q))*n/2;
    return findSum((p*n+r)/q, q, (p*n+r)%q,p);
}

```

## 6 Miscellaneous

### 6.1 Fast IO C++

```

#include<bits/stdc++.h>
using namespace std;
static const int buf_size = 4096;
inline int getChar() {
    static char buf[buf_size];
    static int len = 0, pos = 0;
    if(pos==len)
        pos=0, len=fread(buf, 1, buf_size, stdin);
    if (pos == len) return -1;
    return buf[pos++];
}

inline int readChar() {
    int c = getChar();
    while (c <= 32) c = getChar();
    return c;
}

template <class T>
inline T readInt() {
    int s = 1, c = readChar(); T x = 0;
    if (c == '-') s = -1, c = getChar();
    while ('0'<=c&&c<='9')
        x=x*10+c-'0',c=getChar();
    return s == 1 ? x : -x;
}

static int write_pos = 0;
static char write_buf[buf_size];
inline void writeChar( int x ) {
    if (write_pos == buf_size)
        fwrite(write_buf, 1, buf_size,
            stdout),
        write_pos = 0;
    write_buf[write_pos++] = x;
}

template <class T>
inline void writeInt( T x, char end ) {
    if (x < 0) writeChar('-'), x = -x;
    char s[24]; int n = 0;
    while (x || !n) s[n++] = '0' + x % 10,
        x/=10;
    while (n--) writeChar(s[n]);
    if (end) writeChar(end);
}

inline void writeWord( const char *s ) {
    while (*s) writeChar(*s++);
}

```

### 6.2 Fast IO JAVA

```

public class Main {
    public static void main(String[] args) {
        InputStream inputStream = System.in;
        OutputStream outputStream = System.out;

```

```

        InputReader in = new InputReader(inputStream);
        PrintWriter out = new PrintWriter(outputStream);
        int n = in.nextInt(); long l = in.nextLong();
        out.println(n); out.println(l);
        out.println("done"); out.close();
    }

    static class InputReader {
        public BufferedReader reader;
        public StringTokenizer tokenizer;
        public InputReader(InputStream stream) {
            reader = new BufferedReader(
                new InputStreamReader(stream),32768);
            tokenizer = null;
        }
        public String next() {
            while(tokenizer==null ||
                !tokenizer.hasMoreTokens()){
                try {
                    tokenizer =
                        new StringTokenizer(reader.readLine());
                } catch (IOException e) {
                    throw new RuntimeException(e);
                }
            }
            return tokenizer.nextToken();
        }
        public int nextInt() {
            return Integer.parseInt(next());
        }
        public long nextLong(){
            return Long.parseLong(next());
        }
    }
}

```

### 6.3 Hash Table

```

#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        x += 0x9e3779b97f4a7c15; //Random
        x=(x^(x>>30))*0xbf58476d1ce4e5b9; //Random
        x=(x^(x>>27))*0x94d049bb133111eb; //Random
        return x^(x>>31);
    }
    const uint64_t FIXED_RANDOM = chrono::
        steady_clock::now().time_since_epoch().count();
    size_t operator()(uint64_t x) const {
        return splitmix64(x + FIXED_RANDOM);
    }
    size_t operator()(pair<int, int> x) const {
        return splitmix64((uint64_t(x.first)<<32) +
            x.second + FIXED_RANDOM);
    }
};
gp_hash_table<pair<int,int>,int,custom_hash> ht;

```

### 6.4 Integration Table

$$\int \frac{1}{a^2 + x^2} dx = \frac{1}{a} \tan^{-1} \frac{x}{a} \quad (1)$$

$$\int \frac{x}{a^2 + x^2} dx = \frac{1}{2} \ln |a^2 + x^2| \quad (2)$$

$$\int \frac{1}{ax^2 + bx + c} dx = \frac{2}{\sqrt{4ac - b^2}} \tan^{-1} \frac{2ax + b}{\sqrt{4ac - b^2}} \quad (3)$$

$$\int \frac{1}{(x+a)(x+b)} dx = \frac{1}{b-a} \ln \frac{a+x}{b+x}, \quad a \neq b \quad (4)$$

$$\int \sqrt{x^2 \pm a^2} dx = \frac{1}{2} x \sqrt{x^2 \pm a^2} \pm \frac{1}{2} a^2 \ln \left| x + \sqrt{x^2 \pm a^2} \right| \quad (5)$$

$$\int \sqrt{a^2 - x^2} dx = \frac{1}{2} x \sqrt{a^2 - x^2} + \frac{1}{2} a^2 \tan^{-1} \frac{x}{\sqrt{a^2 - x^2}} \quad (6)$$

$$\int x \sqrt{x^2 \pm a^2} dx = \frac{1}{3} (x^2 \pm a^2)^{3/2} \quad (7)$$

$$\int \frac{1}{\sqrt{x^2 \pm a^2}} dx = \ln \left| x + \sqrt{x^2 \pm a^2} \right| \quad (8)$$

$$\int \frac{1}{\sqrt{a^2 - x^2}} dx = \sin^{-1} \frac{x}{a} \quad (9)$$

$$\int \frac{dx}{(a^2 + x^2)^{3/2}} = \frac{x}{a^2 \sqrt{a^2 + x^2}} \quad (10)$$

$$\int \tan ax dx = -\frac{1}{a} \ln \cos ax \quad (11)$$

$$\int \sec x dx = \ln |\sec x + \tan x| = 2 \tanh^{-1} \left( \tan \frac{x}{2} \right) \quad (12)$$

$$\int e^{bx} \sin ax dx = \frac{1}{a^2 + b^2} e^{bx} (b \sin ax - a \cos ax) \quad (13)$$

$$\int e^{bx} \cos ax dx = \frac{1}{a^2 + b^2} e^{bx} (a \sin ax + b \cos ax) \quad (14)$$

## 6.5 Snippets

```

/// Random
mt19937_64 rng(chrono::steady_clock
                ::now().time_since_epoch().count());
shuffle(V.begin(), V.end(), rng);
int x = uniform_int_distribution<int>(1, r)(rng);

/// bit manipulation
number of leading zeros: __builtin_clz(x)
number of trailing zeros: __builtin_ctz(x)
number of set bits : __builtin_popcountll(x)
bitset : bs._Find_first(), bs._Find_next(15)

/// subset(3~n)
for(int i = mask; i > 0; i = ((i-1) & mask))

/// ordered set
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef tree <int, null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update > ordered_set;
find_by_order(k): itr to kth largest 0 indexed
order_of_key(val): no of items in set < val

/// 2D Partial Sum : update (x1,y1) to (x2,y2) +x
a[x1][y1]+=x; a[x1][y2+1]-=x;
a[x2+1][y1]-=x; a[x2+1][y2+1]+=x;
recons: a[x][y] += a[x-1][y]+a[x][y-1]-a[x-1][y-1]

/// __int128:
__int128 x = 1e12; x = x * x + 1000;
while(x) {res.pb(x%10 + '0'); x/= 10;}

/// split a string by space
string str="abc def gh",buf;stringstream ss(str);
while(ss >> buf) cout << buf << endl;

/// ntt mod :
998244353 = 119 * 2^23 + 1 , primitive root = 3
985661441 = 235 * 2^22 + 1 , primitive root = 3
1012924417 = 483 * 2^21 + 1 , primitive root = 5

/// MO on tree
case-1: lca(u,v) == u , [ST(u),ST(v)]
case-2: otherws, [EN(u),ST(v)]+[ST(lca), ST(lca)]

```

## 7 String

### 7.1 Aho Corasick

```

struct AC {
struct state {
int to[ALPHA],depth,sLink,
int par,parLet,cnt,nxt[ALPHA];
}states[N];
vector<int> suff_tree[N]; int tot_nodes;
void init() {
for(int i = 0; i < N; i++) suff_tree[i].clear();
tot_nodes = 1; clr(states); //careful,memset TLE
}
int add_string(string &str) {
int cur = 1;
for(int i = 0; i < str.size(); i++) {
int c = str[i]-'a';

```

```

if(!states[cur].to[c]) {
states[cur].to[c] = ++tot_nodes;
states[tot_nodes].par = cur;
states[tot_nodes].depth=states[cur].depth+1;
states[tot_nodes].parLet = c;
}
cur = states[cur].to[c];
}
return cur;
}
void push_links() {
queue<int> qq;
qq.push(1);
while (!qq.empty()) {
int node = qq.front();
qq.pop();
if (states[node].depth <= 1)
states[node].sLink = 1;
else {
int cur = states[states[node].par].sLink;
int parLet = states[node].parLet;
while (cur > 1 and !states[cur].to[parLet]){
cur = states[cur].sLink;
}
if (states[cur].to[parLet]) {
cur = states[cur].to[parLet];
}
states[node].sLink = cur;
}
if(node!=1)
suff_tree[states[node].sLink].pb(node);
for (int i = 0 ; i < ALPHA; i++) {
if (states[node].to[i])
qq.push(states[node].to[i]);
}
}
}
int next_state(int from, int c) {
if(states[from].nxt[c])
return states[from].nxt[c];
int cur = from;
while(cur>1&&!states[cur].to[c])
cur=states[cur].sLink;
if(states[cur].to[c]) cur = states[cur].to[c];
return states[from].nxt[c] = cur;
}
void dfs(int u) {
for(int v : suff_tree[u]) {
dfs(v); states[u].cnt += states[v].cnt;
}
}
}aho;

```

### 7.2 KMP

```

vector<int> prefix_function (string s) {
int n = (int) s.length(); vector<int> pi (n);
for (int i=1; i<n; ++i) {
int j = pi[i-1];
while (j > 0 && s[i] != s[j]) j = pi[j-1];
if (s[i] == s[j]) ++j;
pi[i] = j;
}
}

```

```
    return pi;
}
```

### 7.3 Manacher

```
//p[0][i] = maxlen of hlf palin arnd half idx i
//p[1][i] = maxlen of hlf palin arnd idx i, 0 based
VI p[2];
void manacher(const string s) {
    int n = s.size(); p[0] = VI(n+1); p[1] = VI(n);
    for (int z=0; z<2; z++)
        for (int i=0, l=0, r=0; i<n; i++) {
            int t = r - i + !z;
            if (i<r) p[z][i] = min(t, p[z][l+t]);
            int L = i-p[z][i], R = i+p[z][i] - !z;
            while (L>=1 && R+1<n && s[L-1] == s[R+1])
                p[z][i]++, L--, R++;
            if (R>r) l=L, r=R;
        }
}
bool ispalin(int l, int r) {
    int mid = (l+r+1)/2, sz = r-l+1;
    return 2*p[sz%2][mid] + b>=sz;
}
```

### 7.4 Palindromic Tree

```
struct node { int next[26], len, sufflink; };
int len, sz, suff; char str[N]; node tree[N];
bool addLetter(int pos) {
    int cur = suff, curlen = 0, let = str[pos] - 'a';
    while (true) {
        curlen = tree[cur].len;
        if (pos-curlen>=1 && str[pos-1-curlen]==str[pos])
            break;
        cur = tree[cur].sufflink;
    }
    if (tree[cur].next[let]) {
        suff = tree[cur].next[let]; return false;
    }
    suff = ++sz; tree[sz].len = tree[cur].len + 2;
    tree[cur].next[let] = sz;
    if (tree[sz].len==1) {
        tree[sz].sufflink = 2; return 1;
    }
    while (true) {
        cur=tree[cur].sufflink; curlen=tree[cur].len;
        if (pos-curlen>=1 && str[pos-1-curlen]==str[pos]) {
            tree[sz].sufflink=tree[cur].next[let]; break;
        }
    }
    return true;
}
void initTree() {
    memset (tree, 0, sizeof tree); sz = 2; suff = 2;
    tree[1].len = -1; tree[1].sufflink = 1;
    tree[2].len = 0; tree[2].sufflink = 1;
}
```

### 7.5 Suffix Array

```
const int N = 1e6+7, LOG = 20, ALPHA = 128;
struct SuffixArray {
    int sa[N], data[N], rnk[N], hgt[N], n;
    int wa[N], wb[N], wws[N], wv[N];
    int lg[N], rmq[N][LOG], rev_sa[N];
    int cmp(int *r, int a, int b, int l) {
        return (r[a]==r[b]) && (r[a+1]==r[b+1]);
    }
    void DA(int *r, int *sa, int n, int m) {
        int i, j, p, *x=wa, *y=wb, *t;
        for(i=0; i<m; i++) wws[i]=0;
        for(i=0; i<n; i++) wws[x[i]]=r[i]++;
        for(i=1; i<m; i++) wws[i]+=wws[i-1];
        for(i=n-1; i>=0; i--) sa[--wws[x[i]]]=i;
        for(j=1, p=1; p<n; j*=2, m=p) {
            for(p=0, i=n-j; i<n; i++) y[p++]=i;
            for(i=0; i<n; i++) if(sa[i]>=j) y[p++]=sa[i]-j;
            for(i=0; i<n; i++) wv[i]=x[y[i]];
            for(i=0; i<m; i++) wws[i]=0;
            for(i=0; i<n; i++) wws[wv[i]]++;
            for(i=1; i<m; i++) wws[i]+=wws[i-1];
            for(i=n-1; i>=0; i--) sa[--wws[wv[i]]]=y[i];
            for(t=x, x=y, y=t, p=1, x[sa[0]]=0, i=1; i<n; i++)
                x[sa[i]]=cmp(y, sa[i-1], sa[i], j)?p-1:p++;
        }
    }
    void calhgt(int *r, int *sa, int n) {
        int i, j, k=0;
        for(i=1; i<=n; i++) rnk[sa[i]]=i;
        for(i=0; i<n; i++) hgt[rnk[i+1]]=k;
        for(k?k--:0, j=sa[rnk[i]-1]; r[i+k]==r[j+k]; k++);
    }
    void suffix_array (string &A) {
        n = A.size();
        for(int i=0; i<max(n+5, ALPHA); i++) sa[i]=data[i]=
            rnk[i]=hgt[i]=wa[i]=wb[i]=wws[i]=wv[i]=0;
        for (int i = 0; i < n; i++) data[i] = A[i];
        DA(data, sa, n+1, ALPHA);
        calhgt(data, sa, n);
        for(int i = 0; i < n; i++)
            sa[i]=sa[i+1], hgt[i]=hgt[i+1], rev_sa[sa[i]]=i;
        range_lcp_init();
    }
    void range_lcp_init() {
        for(int i = 0; i < n; i++) rmq[i][0] = hgt[i];
        for(int j = 1; j < LOG; j++) {
            for(int i = 0; i < n; i++) {
                if (i+(1<<j)-1 < n) rmq[i][j] =
                    min(rmq[i][j-1], rmq[i+(1<<(j-1))][j-1]);
                else break;
            }
        }
        lg[0] = lg[1] = 0;
        for(int i = 2; i <= n; i++) lg[i] = lg[i/2] + 1;
    }
    int query_lcp(int l, int r) {
        assert(l <= r); assert(l>=0 && l<n && r>=0 && r<n);
        if(l == r) return n-sa[l];
    }
}
```

```
l++; int k = lg[r-l+1];
return min(rmq[l][k], rmq[r-(1<<k)+1][k]);
}
}SA;
```

### 7.6 Suffix Automata

```
///# No of Occ of each state: init each state(except
//the clones) with cnt[state]=1, loop dec order of
//len[state], and do: cnt[link[state]]+=cnt[state]
///# First Occ of each state:
// for new state: firstpos(cur) = len(cur)-1
// for cloned state: firstpos(clone) = firstpos(q)
const int ALPHA = 26;
namespace SuffixAutomata {
    vector<vector<int>> to, nstate;
    vector<int> link, len;
    int n, sz, cur;
    void add(int c) {
        int p = cur;
        cur = ++sz; len[cur] = len[p] + 1;
        while (to[p][c]==0) {to[p][c]=cur; p = link[p];}
        if (to[p][c] == cur) {link[cur] = 0; return;}
        int q = to[p][c];
        if (len[q] == len[p] + 1) link[cur] = q; return;
        int cl = ++sz;
        to[cl] = to[q]; link[cl] = link[q];
        len[cl] = len[p] + 1; link[cur] = link[q] = cl;
        while (to[p][c] == q) {to[p][c]=cl; p=link[p];}
    }
    int advance(int state, int c) {
        if(nstate[state][c]!=-1) return nstate[state][c];
        int nstate;
        if(to[state][c]) nstate = to[state][c];
        else if(state) nstate = advance(link[state], c);
        else nstate = state;
        return nstate[state][c] = nstate;
    }
    void build(string &s) {
        cur = sz = 0; n = s.size();
        to.assign(2*n+1, vector<int>(ALPHA, 0));
        nstate.assign(2*n+1, vector<int>(ALPHA, -1));
        link.assign(2*n+1, 0); len.assign(2*n+1, 0);
        for(int i = 0; i < n; i++) add(s[i] - 'a');
    }
}

7.7 Z Algorithm
vector<int> z_function(string s) {
    int n = s.size();
    vector<int> z(n);
    int l = 0, r = 0;
    for (int i=1; i<n; i++) {
        if (i<=r) z[i] = min(r-i+1, z[i-l]);
        while(i+z[i]<n && s[i+z[i]]==s[z[i]]) z[i]++;
        if (i+z[i]-1>r) l = i, r = i+z[i]-1;
    }
    return z;
}
```