```
#include <ext/pb_ds/assoc_container.hpp> // Common file
#include     <ext/pb_ds/tree_policy.hpp>     //     Including
tree_order_statistics_node_update
using namespace __gnu_pbds;
//#include  <sys/resource.h>       //  for  linux  stack  memory
increase
//#define gc getchar_unlocked   // for linux fast io
//#define gc getchar            // for windows fast io

template <class T> inline T mul(T p, T e, T M)
{
    long long ret = 0;
    for(; e > 0; e >>= 1){
        if(e & 1) ret = (ret + p) ;
        p = (p + p) ;
        if(ret>=M) ret-=M;
        if(p>=M) p-=M;
    }
    return (T)ret;
}
template<class T> inline void input(T &x)
{
    register char c = getchar();
    x = 0;
    int neg = 0;
    for(; ((c<48 || c>57) && c != '-'); c = getchar());
    if(c=='-'){
        neg = 1;
        c = getchar();
    }
    for(; c>47 && c<58 ; c = getchar()){
        x = (x<<1) + (x<<3) + c - 48;
    }
    if(neg) x = -x;
}
/// HASH
template<class T>inline T gethash(T h[],T power[],int l,int r)
{
    if(r<l) return (T) 0;
    return h[l]-h[r+1]*power[r-l+1];
}
mod[]  = {1000000007,1000000009,1000000021,1000000033};
mod[]  = {1000000097,1000000093,1000000097,1000000103};
base[] = {1000003,1000033,1000037,1000039,1000081,1000099};
MOD 999995959, 999998777, 101949101, 104282401, 70139317
MOD 1000000007ULL, MD2 1000000009ULL, 1000000021ULL
BASE 10000019ULL, 10000079ULL, 10000103ULL, 1000117, 1000121
```

```
/// GEOMETRY
typedef long double ld; ld INF = 1e100; ld EPS = 1e-12;
struct PT
{
    ld x, y;
    PT() {}
    PT(ld x, ld y) : x(x), y(y) {}
    PT(const PT &p) : x(p.x), y(p.y)    {}
    PT operator + (const PT &p)  const{
        return PT(x+p.x, y+p.y);
    }
    PT operator - (const PT &p)  const{
        return PT(x-p.x, y-p.y);
    }
    PT operator * (ld c)     const{
        return PT(x*c,   y*c  );
    }
    PT operator / (ld c)     const{
        return PT(x/c,   y/c  );
    }
    bool operator <(const PT &p) const{
        return x < p.x || (x == p.x && y < p.y);
    }
};

ld dot(PT p, PT q){
    return p.x*q.x+p.y*q.y;
}
ld dist2(PT p, PT q){
    return dot(p-q,p-q);
}
ld cross(PT p, PT q){
    return p.x*q.y-p.y*q.x;
}
ostream &operator<<(ostream &os, const PT &p){
    os << "(" << p.x << "," << p.y << ")";
}
// rotate a point CCW or CW around the origin
PT RotateCCW90(PT p){
    return PT(-p.y,p.x);
}
PT RotateCW90(PT p){
    return PT(p.y,-p.x);
}
PT RotateCCW(PT p, ld t){
    return PT(p.x*cos(t)-p.y*sin(t), p.x*sin(t)+p.y*cos(t));
}
```

```
/* This code computes the area or centroid of a (possibly nonconvex)
polygon,assuming that the coordinates are listed in a clockwise or
counterclockwise fashion.  Note that the centroid is often known as
the "center of gravity" or "center of mass". */

ld ComputeSignedArea(const vector<PT> &p){
    ld area = 0;
    for(int i = 0; i < p.size(); i++)
    {
        int j = (i+1) % p.size();
        area += p[i].x*p[j].y - p[j].x*p[i].y;
    }
    return area / 2.0;
}
ld ComputeArea(const vector<PT> &p){
    return fabs(ComputeSignedArea(p));
}
PT ComputeCentroid(const vector<PT> &p){
    PT c(0,0);
    ld scale = 6.0 * ComputeSignedArea(p);
    for (int i = 0; i < p.size(); i++)
    {
        int j = (i+1) % p.size();
        c = c + (p[i]+p[j])*(p[i].x*p[j].y - p[j].x*p[i].y);
    }
    return c / scale;
}

ld angle(PT p1, PT p2, PT p3)
{ // angle from p1->p2 to p1->p3, returns -PI to PI
    PT va = p2-p1,vb=p3-p1;
    ld x,y;
    x=dot(va,vb);
    y=cross(va,vb);
    return(atan2(y,x));
}
double DEG(double x){
    return (180.0*x)/(pi);
}
double RAD(double x){
    return (x*(double)pi)/(180.0);
}
bool colin(int i,int j,int k)
{ ///check if points in index i,j,k are colinear
    return(x[i]*(y[j]-y[k])+x[j]*(y[k]-y[i])+x[k]*(y[i]-y[j]))==0;
}
```

```
int  Set(int N,int pos){ return N=N | (1<<pos); }
int  Reset(int N,int pos){ return N= N & ~(1<<pos); }
bool Check(int N,int pos){ return (bool)(N & (1<<pos)); }
template< class T, class X > inline T   togglebit(T a, X i){
    T t=1; return (a^(t<<i)); }

#define POPCOUNT __builtin_popcountll
#define RIGHTMOST __builtin_ctzll
#define LEFTMOST(x) (63-__builtin_clzll((x)))
#define UNIQUE(V) (V).erase(unique((V).begin(),(V).end()),(V).end())
#define Unique(V) V.resize(unique(V.begin(),V.end())-V.begin())
#define NUMDIGIT(x,y) (((int)(log10((x))/log10((y))))+1)
#define ABS(x) ((x)<0?-(x):(x))
#define FABS(x) ((x)+eps<0?-(x):(x))
#define NORM(x) if(x>=mod)x-=mod;
#define PI         acos(-1.0) OR 2.0*acos(0.0)
//~ cout << fixed << setprecision(20) << Ans << endl;
//~priority_queue<piii,vpiii, greater<piii> >pq; //for dijkstra

int toInt(string s){ int sm; stringstream ss(s); ss>>sm; return sm; }
int toLlint(string s){ LL sm; stringstream ss(s); ss>>sm; return sm;}
int cdigittoint(char ch){return ch-'0';}

template <class T> inline string toString(T t) { stringstream
ss; ss<<t; return ss.str();}
template <class T> inline long long toLong(T t) {stringstream
ss;ss<<t;long long ret;ss>>ret;return ret;}
template  <class  T>  inline  int  toInt(T  t)  {stringstream
ss;ss<<t;int ret;ss>>ret;return ret;}
template <class T> T extract(string s, T ret) {stringstream ss(s); ss
>> ret; return ret;}/// extract words or numbers from a line
template <class T> string tostring(T n) {stringstream ss; ss <<
n; return ss.str();}/// convert a number to string
template<class T> T Mod(T n,T m) {return (n%m+m)%m;} ///For
Positive Negative No.
template <typename T> T Angle(T x1,T y1,T x2, T y2){ return
atan( double(y1-y2) / double(x1-x2));}
template  <class  R>  R  Josephus(R  n,R  k){R  ans=1;for(R
i=2;i<=n;i++)ans=(ans+k-1)%i+1;return ans;}

/**Direction**/
int dx[8]={0,1,1,1,0,-1,-1,-1};int dy[8]={1,1,0,-1,-1,-1,0, 1};
///8 Direction
int dx[4]={1,0,-1,0};int dy[4]={0,1,0,-1};
///4 Direction
int dx[]={2,1,-1,-2,-2,-1,1,2}; int dy[]={1,2,2,1,-1,-2,-2,-1};
///Knight dIRECTION
int dx[]={-1,-1,+0,+1,+1,+0}; int dy[]={-1,+1,+2,+1,-1,-2};
///Hexagonal Direction
```