```
///*************************DIJSKTRA*******************////////////////
///ini es el nodo inicial del cual se hara el dijsktra

void dijkstra(){
    priority_queue< pair<int,int> , vector<pair<int,int> > , greater<pair<int,int> >  > pq;

    vector<int> dist( n , INT_MAX/2);

    pq.push(make_pair( 0 ,  ini)  );
    dist[ini] = 0;

    while(!pq.empty()){
        pair<int,int> u = pq.top();
        pq.pop();

        if(u.first != dist[u.second])   continue;

        for(int i =  0 ; i<AdjList[u.second].size() ; i++){
            pair<int,int> v = AdjList[u.second][i];

            if(dist[v.first] > dist[u.second] + v.second){
                dist[v.first] = dist[u.second]  + v.second;
                pq.push(make_pair(  dist[v.first] , v.first )   );
            }

        }
    }

}

///*********************FIN*DIJSKTRA*******************////////////////

///**************************DSU***********************/////////////////

int F[MAXN];

void ini(){
    for(int i = 0 ; i<MAXN ;i++)    F[i] = i;
}

int FIND(int nodo){ /// te encuentra el padre
    if(F[nodo] == nodo) return nodo;
    else    return F[nodo] = FIND(F[nodo]);
}

void UNION(int u , int v){ ///asigna a F[padre de u] = padre de v;
    F[FIND(u)] = FIND(v);
}

///**********************FIN*DSU***********************/////////////////
```

```
///**********************RMQ***************************//////////////

int SparseTable[MAXN][20];/// la tabla que se usara para el RMQ

int L[MAXN];///los valores para optener el RMQ

int n;///tamaño del array

void ini_SparseTable(){
   int N = n;
   for(int i =  N-1;i>=0 ;i--){
      SparseTable[i][0] = i;
      for(int j = 1 ;i + (1<<j)<=N ; j++) {
         if(L[ SparseTable[i + (1<<(j-1))][j-1] ] <L[  SparseTable[i][j-1] ]){
            SparseTable[i][j] = SparseTable[i + (1<<(j-1))][j-1];
         }else{
            SparseTable[i][j] = SparseTable[i][j-1];
         }
      }
   }
}

int RMQ_query(int u, int v){

   if(u>v) swap(u,v);
   int tam = log2(v-u+1);
   if(L[SparseTable[u][tam] ]<L[  SparseTable[v-(1<<tam)+1][tam] ])    return SparseTable[u][tam] ;
   else    return SparseTable[v - (1<<tam) +1 ][tam] ;
}


///******************FIN*RMQ***************************//////////////


///***********EXTENDIDO_DE_EUCLIDES****************//////////////

pair<int, int> EuEx(int a, int b){/// a es el valor y b es el modulo
   if(b == 0)  return make_pair(1 , 0);
   pair<int,int> u = EuEx(b ,a%b);
   return make_pair( u.second , u.first + (a/b)*u.second );
}

int inverso_modular(int val, int mod){
   return  EuEx(val , mod).first;
}

///*******FIN*EXTENDIDO_DE_EUCLIDES****************//////////////
```

```
///****************GAUSS********************////////////

double M[MAXN][MAXN];/// Es mi matriz

int n , m ;///n = numero de ecuaciones , m = numero de incognitas

void elimination_gaussian(){

    int col = 0;

    for(int i = 0 ; i<n && col<m ;col++){
        for(int j = i ; j<n ;j++){
            if(M[i][j]!=0){
                for(int k = 0 ; k<=m ;k++){
                    swap(M[i][k] , M[j][k]);
                }
                break;
            }
        }

        if(M[i][col] == 0)  continue;

        double temp = M[i][col];
        for(int k = 0 ; k<=m ;k++)  M[i][k]/=temp;

        for(int j = 0 ; j<n ;j++) if(i!=j){
            temp = M[j][col];
            for(int k = 0 ; k<=m ;k++){
                M[j][k] -= M[i][k]*temp;
            }
        }
        i++;
    }

}

int main(){

    freopen("in.c","r",stdin);

    cin>>n>>m;

    for(int i = 0 ; i<n ;i++)
        for(int j = 0 ; j<=m ;j++)
            scanf("%lf" , &M[i][j]);

    elimination_gaussian();

    for(int i = 0 ; i<n ;i++){
        for(int j = 0 ; j<=m ;j++)
            cout<<M[i][j]<<" ";
        cout<<endl;
    }
}

///************FIN*GAUSS*******************////////////
```

```
///***********************LCA*********************///

vector<vector<int> >  AdjList;

int E[2*MAXN-1] , L[2*MAXN-1] , R[MAXN];

/*
L = nivel
*/

int ind = 0;
void dfs(int nodo, int padre , int level){
    R[nodo] = min(ind ,R[nodo] );
    E[ind] = nodo;
    L[ind++] = level;

    for(int i = 0 ; i<AdjList[nodo].size() ;i++){
        int v = AdjList[nodo][i];

        if(v!=padre){
            dfs(v , nodo , level+1);
            E[ind] = nodo;
            L[ind++] = level;
        }
    }
}


///------------------RMQ--------------//////////////////////

int SparseTable[MAXN][20];

int n;

void ini_SparseTable(){
    int N = ind;
    for(int i =  N-1;i>=0 ;i--){
        SparseTable[i][0] = i;
        for(int j = 1 ;i + (1<<j)<=N ; j++) {
            if(L[ SparseTable[i + (1<<(j-1))][j-1] ] <L[  SparseTable[i][j-1] ]){
                SparseTable[i][j] = SparseTable[i + (1<<(j-1))][j-1];
            }else{
                SparseTable[i][j] = SparseTable[i][j-1];
            }
        }
    }

    /*for(int i =  0;i<n ;i++){
        for(int j = 0 ;i + (1<<j)<=n ; j++) {
            cout<<SparseTable[i][j]<<" ";
        }
        cout<<endl;
    }*/
}
```

```
int RMQ_query(int u, int v){

    if(u>v) swap(u,v);
    if(L[SparseTable[u][tam] ]<L[  SparseTable[v-(1<<tam)+1][tam] ])    return SparseTable[u][tam] ;
    int tam = log2(v-u+1);
    else    return SparseTable[v - (1<<tam) +1 ][tam] ;
}


///---------------------------------------------//////////

int main(){

    freopen("in.c","r",stdin);
    int m , x,y;

    cin>>n;
    AdjList.assign(n, vector<int>() );
    for(int i = 0 ; i<n-1 ;i++){
        scanf("%d%d" ,&x,&y);
        x-- , y--;
        AdjList[x].push_back(y);
        AdjList[y].push_back(x);
    }

    for(int i = 0 ; i<n ;i++)   R[i] = (3*MAXN);

    dfs( 0 , -1 , 0);


    ini_SparseTable();

    /*or(int i = 0 ; i<ind ;i++) cout<<E[i]+1<<" ";
    cout<<endl;
    for(int i = 0 ; i<ind ;i++) cout<<L[i]<<" ";
    cout<<endl;
    for(int i = 0 ; i<n ;i++) cout<<R[i]<<" ";
    cout<<endl;*/
    int k;

    cin>>k;

    for(int i = 0 ; i<k ;i++){
        scanf("%d%d",&x,&y);
        x--,y--;
        int LCA = E[RMQ_query(R[x],R[y])];
        cout<<x+1<<"--"<<y+1<<"--->"<<LCA+1<<endl;
    }

    return 0;
}

///******************FIN*LCA*********************///
```

```
///****************************SCC*********************//////

int n; /// numero de nodos

vector<vector<int> > AdjList , AdjList_T;

bool cmp[MAXN];
int comp;
int nodo_SCC[MAXN];

vector<int> pila;

void dfs(int nodo , int super_nodo){
    cmp[nodo] = true;
    nodo_SCC[nodo] = super_nodo;
    for(int i = 0 ; i<AdjList[nodo].size() ; i++){
        int v = AdjList[nodo][i];
        if(!cmp[v])  dfs(v , super_nodo);
    }
    pila.push_back(nodo);
}

void SCC(){
    memset(cmp , 0 , sizeof cmp);
    for(int i = 0 ; i<n ;i++){
        if(!cmp[i]) dfs(i , 0);
    }

    swap(AdjList , AdjList_T);
    memset(cmp , 0 , sizeof cmp);

    for(int i = pila.size()-1 ; i>=0 ;i--){
        int v = pila[i];
        if(!cmp[v]) {
            dfs(v , comp);
            comp++;
        }
    }
    swap(AdjList , AdjList_T);

    ///APUNTES
    ///comp tiene el numero de supernodos :)
    ///ahora puedo trabajar el grafo como si fueran componentes
}

int main(){

    freopen("in.c","r",stdin);
    int m,x,y;

    cin>>n>>m;

    AdjList.assign(  n , vector<int>() );
    AdjList_T.assign(  n , vector<int>() );

    for(int i = 0 ; i<m ;i++){
        scanf("%d%d",&x,&y);
        x-- , y--;
        AdjList[x].push_back(y);
        AdjList_T[y].push_back(x);
    }

    SCC();

    for(int i = 0 ; i<n ;i++){
        cout<<i+1<<"-->"<<nodo_SCC[i]<<endl;
    }

    return 0;
}

///************************FIN*SCC*********************//////


///**************************BIT*********************/////

int BIT[MAXN];

int read(int ind){
    if(ind == 0)  return 0;
    int ans = 0;
    while(ind>0){
        ans += BIT[ind];
        ind -= ind&(-ind);
    }
    return ans;
}

void update(int ind, int val){
    while(ind<MAXN){
        BIT[ind] += val;
        ind += ind&(-ind);
    }
}

///***********************FIN*BIT********************////
```

```
////////////////////CENTRO DE MASA//////////////////////////////
#define MAXN 30009

struct node{
    int x;
    int damage;
    int large;

    node(){}
    node(int _x, int _damage , int _large){
        x = _x;
        damage = _damage;
        large = _large;
    }
};

#define vn vector<node>
vector< vn > AdjList;

int n,k,x,y,d,l,hijos[MAXN],mini,CM,cont,cc , ans;
bool cmp_global[MAXN];

int dfs(int nodo, int padre){// es para hallar el numero de nodos de los hijos colgado de un centro

    int ans = 1;
    f(i,0,AdjList[nodo].size()){

        int v = AdjList[nodo][i].x;
        if(!cmp_global[v]  &&  v!=padre )
            ans += dfs(v,nodo);

    }
    return hijos[nodo] = ans;
}

void dfs1(int nodo, int padre, int num_nodos){// tengo que inicializar mini con INF

    int sum = 0,maxi = -1;

    f(i,0,AdjList[nodo].size()){
        int v = AdjList[nodo][i].x;
        if(!cmp_global[v] && v!=padre){

            dfs1(v,nodo, num_nodos);
            maxi = max(maxi , hijos[v]);
            sum += hijos[v];
        }
    }

    maxi = max(maxi,  num_nodos - sum -1);

    if(mini>=maxi){
        CM = nodo;
        mini = maxi;
    }
}
```

```
vector<pair<pii,int> > acum;

void dfs2(int nodo, int padre, int damage, int large){

    cont++;
    acum.pb(make_pair(pii(damage, large) , cc   )   );
    f(i,0,AdjList[nodo].size()){

        int v = AdjList[nodo][i].x , dam = AdjList[nodo][i].damage , lar = AdjList[nodo][i].large;
        if(!cmp_global[v] && v!=padre)  dfs2(v,nodo, damage + dam , large + lar);

    }

}

vector<pii> pre_process(){

    vector<pii> save1;
    acum.clear();
    cc = 0;
    f(i,0,AdjList[CM].size()){

        int v = AdjList[CM][i].x , damage = AdjList[CM][i].damage ;
        int large = AdjList[CM][i].large;

        if(!cmp_global[v] ){
            cont = 0;
            dfs2(v , -1 , damage , large);
            save1.pb(pii(v, cont));
            cc++;

        }

    }

    return save1;
}

void sol(int nodo, int num_nodos){
    if(num_nodos==1)    return;

    mini = oo;
    //Optengo el CM
    dfs(nodo,-1);
    dfs1(nodo,-1,num_nodos);
    ///////////////////////////////

    cmp_global[CM] = true;

    vector<pii> save1 = pre_process();

    sort(all(acum));

    f(i,0,acum.size())
        if(acum[i].fst.fst<=k)
```

```
            ans = max(ans, acum[i].fst.snd);

    /////////////////////7
    vector<pii> temp;

    vector<pii> dp1,dp2;

    temp.pb(pii(acum[0].fst.snd , acum[0].snd));
    temp.pb(pii(acum[0].fst.snd , acum[0].snd));

    dp1.pb(temp[0]);
    dp2.pb(temp[1]);

    f(i,1,acum.size()){

        temp.pb(pii(acum[i].fst.snd , acum[i].snd));
        sort(rall(temp));

        vector<pii> temp1;
        temp1.pb(temp[0]);

        if(temp[1].snd != temp[0].snd)  temp1.pb(temp[1]);
        else if(temp[2].snd != temp[0].snd) temp1.pb(temp[2]);
        else    temp1.pb(temp[1]);

        temp = temp1;

        dp1.pb(temp[0]);
        dp2.pb(temp[1]);
    }

    f(i,1,acum.size()){
        int L = 0 , R = i-1 , mid;
        while( R-L > 1){
            mid = (L+R)/2;
            if(acum[i].fst.fst + acum[mid].fst.fst <= k)   L = mid;
            else    R = mid;
        }

        if(acum[R].fst.fst + acum[i].fst.fst <= k)  {
            if(dp1[R].snd!=acum[i].snd) ans = max(ans, dp1[R].fst + acum[i].fst.snd);
            if(dp2[R].snd!=acum[i].snd) ans = max(ans, dp2[R].fst + acum[i].fst.snd);
        }
        if(acum[L].fst.fst + acum[i].fst.fst <= k)  {
            if(dp1[L].snd!=acum[i].snd) ans = max(ans, dp1[L].fst + acum[i].fst.snd);
            if(dp2[L].snd!=acum[i].snd) ans = max(ans, dp2[L].fst + acum[i].fst.snd);
        }

    }

    ////////////////////////

    f(i,0,save1.size())
        sol(save1[i].fst , save1[i].snd);

}
```

```
int main() {
    freopen("in.c","r",stdin);

    int TC,NC = 1;
    scanf("%d",&TC);

    while(TC--){

        scanf("%d%d",&n,&k);
        AdjList.assign(n+2 , vn());

        f(i,0,n-1){
            scanf("%d%d%d%d",&x,&y,&d,&l);
            x--;y--;
            AdjList[x].pb( node(y,d,l)  );
            AdjList[y].pb( node(x,d,l)  );

        }

        clr(cmp_global,0);
        ans = 0;

        sol(0 ,  n ) ;

        cout<<"Case "<<NC++<<": "<<ans<<endl;

    }

    return 0;
}
```

```
/////////////////////CLOSEST PAIR///////////////////////////////////////////
ll dist(pii x, pii y){
    ll ans = ((ll)(x.fst-y.fst)*(ll)(x.fst-y.fst)) + ((ll)(x.snd-y.snd)*(ll)(x.snd-y.snd));
    return ans;
}

vector<pii> save;
int n;
ll dist_min;
void ClosestPair(int b,int e){
    if(b==e) return;
    else
    {
        int mid = (b+e)/2 ;

        ClosestPair(b,mid);
        ClosestPair(mid+1,e);

        ll x_mid = save[mid].fst;

        vector<pii > save1;//ordenar por Y

        f(i,b,e+1)
        if( (ll)(save[i].fst - x_mid)*(ll)(save[i].fst - x_mid) <= dist_min )
            save1.pb(pii(save[i].snd,save[i].fst) );

        sort(all(save1));
        f(i,0,save1.size() )
        {
            int ind1 = i+1; // ind1++
            int cont = 0;
            while(ind1<save1.size() && cont<6)
            {
                dist_min = min(dist_min , dist( save1[i], save1[ind1] ) );
                ind1++;
                cont++;
            }
        }
    }
}
```

```
//////////////////////GEOMETRIA///////////////////////////////////////////
#define EPS 1e-8
#define PI acos(-1)
#define Vector Point

struct Point
{
    double x, y;
    Point(){}
    Point(double a, double b) { x = a; y = b; }
    double mod2() { return x*x + y*y; }
    double mod()  { return sqrt(x*x + y*y); }
    double arg()  { return atan2(y, x); }
    Point ort()   { return Point(-y, x); }
    Point unit()  { double k = mod(); return Point(x/k, y/k); }
};

Point operator +(const Point &a, const Point &b) { return Point(a.x + b.x, a.y + b.y); }
Point operator -(const Point &a, const Point &b) { return Point(a.x - b.x, a.y - b.y); }
Point operator /(const Point &a, double k) { return Point(a.x/k, a.y/k); }
Point operator *(const Point &a, double k) { return Point(a.x*k, a.y*k); }

ostream &operator<<(ostream &os, const Point &p) {
  os << "(" << p.x << "," << p.y << ")";
}

Point RotateCCW90(Point p)   { return Point(-p.y,p.x); }
Point RotateCW90(Point p)    { return Point(p.y,-p.x); }
Point RotateCCW(Point p, double t) {
  return Point(p.x*cos(t)-p.y*sin(t), p.x*sin(t)+p.y*cos(t));
}

bool operator ==(const Point &a, const Point &b)
{
    return abs(a.x - b.x) < EPS && abs(a.y - b.y) < EPS;
}
bool operator !=(const Point &a, const Point &b)
{
    return !(a==b);
}
bool operator <(const Point &a, const Point &b)
{
    if(abs(a.x - b.x) > EPS) return a.x < b.x;
    return a.y + EPS < b.y;
}

//### FUNCIONES BASICAS #############################################################

double dist(const Point &A, const Point &B)    { return hypot(A.x - B.x, A.y - B.y); }
double cross(const Vector &A, const Vector &B) { return A.x * B.y - A.y * B.x; }
double dot(const Vector &A, const Vector &B)   { return A.x * B.x + A.y * B.y; }
double area(const Point &A, const Point &B, const Point &C) { return cross(B - A, C - A); }
double dist2(Point p, Point q)   { return dot(p-q,p-q); }
```

```
// project point c onto line through a and b
// assuming a != b
Point ProjectPointLine(Point a, Point b, Point c) {
  return a + (b-a)*dot(c-a, b-a)/dot(b-a, b-a);
}

// project point c onto line segment through a and b
Point ProjectPointSegment(Point a, Point b, Point c) {
  double r = dot(b-a,b-a);
  if (fabs(r) < EPS) return a;
  r = dot(c-a, b-a)/r;
  if (r < 0) return a;
  if (r > 1) return b;
  return a + (b-a)*r;
}

// compute distance from c to segment between a and b
double DistancePointSegment(Point a, Point b, Point c) {
  return sqrt(dist2(c, ProjectPointSegment(a, b, c)));
}

// compute distance between point (x,y,z) and plane ax+by+cz=d
double DistancePointPlane(double x, double y, double z,
                          double a, double b, double c, double d)
{
  return fabs(a*x+b*y+c*z-d)/sqrt(a*a+b*b+c*c);
}

// determine if lines from a to b and c to d are parallel or collinear
bool LinesParallel(Point a, Point b, Point c, Point d) {
  return fabs(cross(b-a, c-d)) < EPS;
}

bool LinesCollinear(Point a, Point b, Point c, Point d) {
  return LinesParallel(a, b, c, d)
      && fabs(cross(a-b, a-c)) < EPS
      && fabs(cross(c-d, c-a)) < EPS;
}

// Heron triangulo y cuadrilatero ciclico
// http://mathworld.wolfram.com/CyclicQuadrilateral.html
// http://www.spoj.pl/problems/QUADAREA/

double areaHeron(double a, double b, double c){
        double s = (a + b + c) / 2;
        return sqrt(s * (s-a) * (s-b) * (s-c));
}

double circumradius(double a, double b, double c) { return a * b * c / (4 * areaHeron(a, b, c)); }

double areaHeron(double a, double b, double c, double d)
{
        double s = (a + b + c + d) / 2;
        return sqrt((s-a) * (s-b) * (s-c) * (s-d));
}
```

```
double circumradius(double a, double b, double c, double d) { return sqrt((a*b + c*d) * (a*c + b*d) * (a*d + b*c))
/ (4 * areaHeron(a, b, c, d)); }

//### DETERMINA SI P PERTENECE AL SEGMENTO AB
#############################################
bool between(const Point &A, const Point &B, const Point &P)
{
   return  P.x + EPS >= min(A.x, B.x) && P.x <= max(A.x, B.x) + EPS &&
           P.y + EPS >= min(A.y, B.y) && P.y <= max(A.y, B.y) + EPS;
}

bool onSegment(const Point &A, const Point &B, const Point &P)
{
   return abs(area(A, B, P)) < EPS && between(A, B, P);
}

//### DETERMINA SI EL SEGMENTO P1Q1 SE INTERSECTA CON EL SEGMENTO P2Q2
#####################
//funciona para cualquiera P1, P2, P3, P4
bool intersects(const Point &P1, const Point &P2, const Point &P3, const Point &P4)
{
   double A1 = area(P3, P4, P1);
   double A2 = area(P3, P4, P2);
   double A3 = area(P1, P2, P3);
   double A4 = area(P1, P2, P4);

   if( ((A1 > 0 && A2 < 0) || (A1 < 0 && A2 > 0)) &&
      ((A3 > 0 && A4 < 0) || (A3 < 0 && A4 > 0)))
           return true;

   else if(A1 == 0 && onSegment(P3, P4, P1)) return true;
   else if(A2 == 0 && onSegment(P3, P4, P2)) return true;
   else if(A3 == 0 && onSegment(P1, P2, P3)) return true;
   else if(A4 == 0 && onSegment(P1, P2, P4)) return true;
   else return false;
}

//### DETERMINA SI A, B, M, N PERTENECEN A LA MISMA RECTA ##############################
bool sameLine(Point P1, Point P2, Point P3, Point P4)
{
        return area(P1, P2, P3) == 0 && area(P1, P2, P4) == 0;
}
//### SI DOS SEGMENTOS O RECTAS SON PARALELOS
####################################################
bool isParallel(const Point &P1, const Point &P2, const Point &P3, const Point &P4)
{
        return cross(P2 - P1, P4 - P3) == 0;
}

//### PUNTO DE INTERSECCION DE DOS RECTAS NO PARALELAS
##################################
Point lineIntersection(const Point &A, const Point &B, const Point &C, const Point &D)
{
   return A + (B - A) * (cross(C - A, D - C) / cross(B - A, D - C));
}
```

```
Point circumcenter(const Point &A, const Point &B, const Point &C)
{
        return (A + B + (A - B).ort() * dot(C - B, A - C) / cross(A - B, A - C)) / 2;
}

Point ComputeCircleCenter(Point a, Point b, Point c) {
  b=(a+b)/2;
  c=(a+c)/2;
  return lineIntersection(b, b+RotateCW90(a-b), c, c+RotateCW90(a-c));
}

//### FUNCIONES BASICAS DE POLIGONOS ######################################################
bool isConvex(const vector <Point> &P)
{
    int n = P.size(), pos = 0, neg = 0;
    for(int i=0; i<n; i++)
    {
        double A = area(P[i], P[(i+1)%n], P[(i+2)%n]);
        if(A < 0) neg++;
        else if(A > 0) pos++;
    }
    return neg == 0 || pos == 0;
}

double area(const vector <Point> &P)
{
    int n = P.size();
    double A = 0;
    for(int i=1; i<=n-2; i++)
        A += area(P[0], P[i], P[i+1]);
    return abs(A/2);
}

bool pointInPoly(const vector <Point> &P, const Point &A)
{
    int n = P.size(), cnt = 0;
    for(int i=0; i<n; i++)
    {
        int inf = i, sup = (i+1)%n;
        if(P[inf].y > P[sup].y) swap(inf, sup);
        if(P[inf].y <= A.y && A.y < P[sup].y)
            if(area(A, P[inf], P[sup]) > 0)
                cnt++;
    }
    return (cnt % 2) == 1;
}
```

```
//### CONVEX HULL #######################################################################
// O(nh)
/*vector <Point> ConvexHull(vector <Point> S)
{
        sort(all(S));

        int it=0;
        Point primero = S[it], ultimo =  primero;

        int n = S.size();

        vector <Point> convex;
        do
        {
                convex.push_back(S[it]);
                it = (it + 1)%n;

                for(int i=0; i<S.size(); i++)
                {
                        if(S[i]!=ultimo && S[i]!=S[it])
                        {
                                if(area(ultimo, S[it], S[i]) < EPS) it = i;
                        }
                }

                ultimo=S[it];
        }while(ultimo!=primero);

        return convex;
}*/

// O(n log n)
vector <Point> ConvexHull(vector <Point> P)
{
   sort(P.begin(),P.end());
   int n = P.size(),k = 0;
   Point H[2*n];

   for(int i=0;i<n;++i){
      while(k>=2 && area(H[k-2],H[k-1],P[i]) <= 0) --k;
      H[k++] = P[i];
   }

   for(int i=n-2,t=k;i>=0;--i){
      while(k>t && area(H[k-2],H[k-1],P[i]) <= 0) --k;
      H[k++] = P[i];
   }

   return vector <Point> (H,H+k-1);
}
```

```
//### DETERMINA SI P ESTA EN EL INTERIOR DEL POLIGONO CONVEXO A ########################

// O (log n)
bool isInConvex(vector <Point> &A, const Point &P)
{
        int n = A.size(), lo = 1, hi = A.size() - 1;

        if(area(A[0], A[1], P) <= 0) return 0;
        if(area(A[n-1], A[0], P) <= 0) return 0;

        while(hi - lo > 1)
        {
                int mid = (lo + hi) / 2;

                if(area(A[0], A[mid], P) > 0) lo = mid;
                else hi = mid;
        }

        return area(A[lo], A[hi], P) > 0;
}

// O(n)
Point norm(const Point &A, const Point &O)
{
   Vector V = A - O;
   V = V * 10000000000.0 / V.mod();
   return O + V;
}

bool isInConvex(vector <Point> &A, vector <Point> &B)
{
   if(!isInConvex(A, B[0])) return 0;
   else
   {
      int n = A.size(), p = 0;

      for(int i=1; i<B.size(); i++)
      {
         while(!intersects(A[p], A[(p+1)%n], norm(B[i], B[0]), B[0])) p = (p+1)%n;

         if(area(A[p], A[(p+1)%n], B[i]) <= 0) return 0;
      }

      return 1;
   }
}
```

```
//##### SMALLEST ENCLOSING CIRCLE O(n) ################################################
// http://www.cs.uu.nl/docs/vakken/ga/slides4b.pdf
// http://www.spoj.pl/problems/ALIENS/

pair <Point, double> enclosingCircle(vector <Point> P)
{
    random_shuffle(P.begin(), P.end());

    Point O(0, 0);
    double R2 = 0;

    for(int i=0; i<P.size(); i++)
    {
        if((P[i] - O).mod2() > R2 + EPS)
        {
            O = P[i], R2 = 0;
            for(int j=0; j<i; j++)
            {
                if((P[j] - O).mod2() > R2 + EPS)
                {
                    O = (P[i] + P[j])/2, R2 = (P[i] - P[j]).mod2() / 4;
                    for(int k=0; k<j; k++)
                        if((P[k] - O).mod2() > R2 + EPS)
                            O = circumcenter(P[i], P[j], P[k]), R2 = (P[k] - O).mod2();
                }
            }
        }
    }
    return make_pair(O, sqrt(R2));
}

//##### CLOSEST PAIR OF POINTS ########################################################
bool XYorder(Point P1, Point P2)
{
        if(P1.x != P2.x) return P1.x < P2.x;
        return P1.y < P2.y;
}
bool YXorder(Point P1, Point P2)
{
        if(P1.y != P2.y) return P1.y < P2.y;
        return P1.x < P2.x;
}
double closest_recursive(vector <Point> vx, vector <Point> vy)
{
        if(vx.size()==1) return 1e20;
        if(vx.size()==2) return dist(vx[0], vx[1]);

        Point cut = vx[vx.size()/2];

        vector <Point> vxL, vxR;
        for(int i=0; i<vx.size(); i++)
                if(vx[i].x < cut.x || (vx[i].x == cut.x && vx[i].y <= cut.y))
                        vxL.push_back(vx[i]);
                else vxR.push_back(vx[i]);

        vector <Point> vyL, vyR;
    for(int i=0; i<vy.size(); i++)
        if(vy[i].x < cut.x || (vy[i].x == cut.x && vy[i].y <= cut.y))
            vyL.push_back(vy[i]);
        else vyR.push_back(vy[i]);

    double dL = closest_recursive(vxL, vyL);
    double dR = closest_recursive(vxR, vyR);
    double d = min(dL, dR);

    vector <Point> b;
    for(int i=0; i<vy.size(); i++)
        if(abs(vy[i].x - cut.x) <= d)
            b.push_back(vy[i]);

    for(int i=0; i<b.size(); i++)
        for(int j=i+1; j<b.size() && (b[j].y - b[i].y) <= d; j++)
            d = min(d, dist(b[i], b[j]));

    return d;
}
double closest(vector <Point> points)
{
    vector <Point> vx = points, vy = points;
    sort(vx.begin(), vx.end(), XYorder);
    sort(vy.begin(), vy.end(), YXorder);

    for(int i=0; i+1<vx.size(); i++)
        if(vx[i] == vx[i+1])
            return 0.0;

    return closest_recursive(vx,vy);
}

// INTERSECCION DE CIRCULOS
vector <Point> circleCircleIntersection(Point O1, double r1, Point O2, double r2)
{
    vector <Point> X;

    double d = dist(O1, O2);

    if(d > r1 + r2 || d < max(r2, r1) - min(r2, r1)) return X;
    else
    {
        double a = (r1*r1 - r2*r2 + d*d) / (2.0*d);
        double b = d - a;
        double c = sqrt(abs(r1*r1 - a*a));

        Vector V = (O2-O1).unit();
        Point H = O1 + V * a;

        X.push_back(H + V.ort() * c);

        if(c > EPS) X.push_back(H - V.ort() * c);
    }

    return X;
```

```
}

// LINEA AB vs CIRCULO (O, r)
// 1. Mucha perdida de precision, reemplazar por resultados de formula.
// 2. Considerar line o segment

vector <Point> lineCircleIntersection(Point A, Point B, Point O, long double r)
{
        vector <Point> X;

        Point H1 = O + (B - A).ort() * cross(O - A, B - A) / (B - A).mod2();
        long double d2 = cross(O - A, B - A) * cross(O - A, B - A) / (B - A).mod2();

        if(d2 <= r*r + EPS)
        {
                long double k = sqrt(abs(r * r - d2));

                Point P1 = H1 + (B - A) * k / (B - A).mod();
                Point P2 = H1 - (B - A) * k / (B - A).mod();

                if(between(A, B, P1)) X.push_back(P1);

                if(k > EPS && between(A, B, P2)) X.push_back(P2);
        }

        return X;
}

//### PROBLEMAS BASICOS ###############################################################
void CircumscribedCircle()
{
        int x1, y1, x2, y2, x3, y3;
        scanf("%d %d %d %d %d %d", &x1, &y1, &x2, &y2, &x3, &y3);

        Point A(x1, y1), B(x2, y2), C(x3, y3);

        Point P1 = (A + B) / 2.0;
        Point P2 = P1 + (B-A).ort();
        Point P3 = (A + C) / 2.0;
        Point P4 = P3 + (C-A).ort();

        Point CC = lineIntersection(P1, P2, P3, P4);
        double r = dist(A, CC);

        printf("(%.6lf,%.6lf,%.6lf)\n", CC.x, CC.y, r);
}

void InscribedCircle()
{
        int x1, y1, x2, y2, x3, y3;
        scanf("%d %d %d %d %d %d", &x1, &y1, &x2, &y2, &x3, &y3);

        Point A(x1, y1), B(x2, y2), C(x3, y3);

        Point AX = A + (B-A).unit() + (C-A).unit();
        Point BX = B + (A-B).unit() + (C-B).unit();
```

```
    Point CC = lineIntersection(A, AX, B, BX);
    double r = abs(area(A, B, CC) / dist(A, B));

    printf("(%.6lf,%.6lf,%.6lf)\n", CC.x, CC.y, r);
}

vector <Point>  TangentLineThroughPoint(Point P, Point C, long double r)
{
    vector <Point> X;

    long double h2 = (C - P).mod2();
    if(h2 < r*r) return X;
    else
    {
        long double d = sqrt(h2 - r*r);

        long double m1 = (r*(P.x - C.x) + d*(P.y - C.y)) / h2;
        long double n1 = (P.y - C.y - d*m1) / r;

        long double n2 = (d*(P.x - C.x) + r*(P.y - C.y)) / h2;
        long double m2 = (P.x - C.x - d*n2) / r;

        X.push_back(C + Point(m1, n1)*r);
        if(d != 0) X.push_back(C + Point(m2, n2)*r);

        return X;
    }
}

void TangentLineThroughPoint()
{
    int xc, yc, r, xp, yp;
    scanf("%d %d %d %d %d", &xc, &yc, &r, &xp, &yp);

    Point C(xc, yc), P(xp, yp);

    double hyp = dist(C, P);
    if(hyp < r) printf("[]\n");
    else
    {
        double d = sqrt(hyp * hyp - r*r);

        double m1 = (r*(P.x - C.x) + d*(P.y - C.y)) / (r*r + d*d);
        double n1 = (P.y - C.y - d*m1) / r;
        double ang1 = 180 * atan(-m1/n1) / PI + EPS;
        if(ang1 < 0) ang1 += 180.0;

        double n2 = (d*(P.x - C.x) + r*(P.y - C.y)) / (r*r + d*d);
        double m2 = (P.x - C.x - d*n2) / r;
        double ang2 = 180 * atan(-m2/n2) / PI + EPS;
        if(ang2 < 0) ang2 += 180.0;

        if(ang1 > ang2) swap(ang1, ang2);

        if(d == 0) printf("[%.6lf]\n", ang1);
        else printf("[%.6lf,%.6lf]\n", ang1, ang2);
    }
}

void CircleThroughAPointAndTangentToALineWithRadius()
{
    int xp, yp, x1, y1, x2, y2, r;
    scanf("%d %d %d %d %d %d %d", &xp, &yp, &x1, &y1, &x2, &y2, &r);

    Point P(xp, yp), A(x1, y1), B(x2, y2);

    Vector V = (B - A).ort() * r / (B - A).mod();

    Point X[2];
    int cnt = 0;

    Point H1 = P + (B - A).ort() * cross(P - A, B - A) / (B - A).mod2() + V;
    double d1 = abs(r + cross(P - A, B - A) / (B - A).mod());

    if(d1 - EPS <= r)
    {
        double k = sqrt(abs(r * r - d1 * d1));

        X[cnt++] = Point(H1 + (B - A).unit() * k);

        if(k > EPS) X[cnt++] = Point(H1 - (B - A).unit() * k);
    }

    Point H2 = P + (B - A).ort() * cross(P - A, B - A) / (B - A).mod2() - V;
    double d2 = abs(r - cross(P - A, B - A) / (B - A).mod());

    if(d2 - EPS <= r)
    {
        double k = sqrt(abs(r * r - d2 * d2));

        X[cnt++] = Point(H2 + (B - A).unit() * k);

        if(k > EPS) X[cnt++] = Point(H2 - (B - A).unit() * k);
    }

    sort(X, X + cnt);

    if(cnt == 0) printf("[]\n");
    else if(cnt == 1) printf("[(%.6lf,%.6lf)]\n", X[0].x, X[0].y);
    else if(cnt == 2) printf("[(%.6lf,%.6lf),(%.6lf,%.6lf)]\n", X[0].x, X[0].y, X[1].x, X[1].y);
}

void CircleTangentToTwoLinesWithRadius()
{
    int x1, y1, x2, y2, x3, y3, x4, y4, r;
    scanf("%d %d %d %d %d %d %d %d %d", &x1, &y1, &x2, &y2, &x3, &y3, &x4, &y4, &r);

    Point A1(x1, y1), B1(x2, y2), A2(x3, y3), B2(x4, y4);

    Vector V1 = (B1 - A1).ort() * r / (B1 - A1).mod();
    Vector V2 = (B2 - A2).ort() * r / (B2 - A2).mod();
```

```
        Point X[4];
        X[0] = lineIntersection(A1 + V1, B1 + V1, A2 + V2, B2 + V2);
        X[1] = lineIntersection(A1 + V1, B1 + V1, A2 - V2, B2 - V2);
        X[2] = lineIntersection(A1 - V1, B1 - V1, A2 + V2, B2 + V2);
        X[3] = lineIntersection(A1 - V1, B1 - V1, A2 - V2, B2 - V2);

        sort(X, X + 4);
        printf("[(%.6lf,%.6lf),(%.6lf,%.6lf),(%.6lf,%.6lf),(%.6lf,%.6lf)]\n", X[0].x, X[0].y, X[1].x, X[1].y, X[2].x, X[2].y, X[3].x, X[3].y);
}

void CircleTangentToTwoDisjointCirclesWithRadius()
{
        int x1, y1, r1, x2, y2, r2, r;
        scanf("%d %d %d %d %d %d %d", &x1, &y1, &r1, &x2, &y2, &r2, &r);

        Point A(x1, y1), B(x2, y2);

        r1 += r;
        r2 += r;

        double d = dist(A, B);

        if(d > r1 + r2 || d < max(r1, r2) - min(r1, r2)) printf("[]\n");
        else
        {
                double a = (r1*r1 - r2*r2 + d*d) / (2.0*d);
                double b = d - a;
                double c = sqrt(abs(r1*r1 - a*a));

                Vector V = (B-A).unit();
                Point H = A + V * a;

                Point P1 = H + V.ort() * c;
                Point P2 = H - V.ort() * c;

                if(P2 < P1) swap(P1, P2);

                if(P1 == P2) printf("[(%.6lf,%.6lf)]\n", P1.x, P1.y);
                else printf("[(%.6lf,%.6lf),(%.6lf,%.6lf)]\n", P1.x, P1.y, P2.x, P2.y);
        }
}

int main(){


    return 0;
}
```

```
//////////////////////////MATRICES//////////////////////////////////

struct Matrix{
    int X[SIZE][SIZE];

    Matrix () {}
    Matrix (int k){
        memset(X, 0, sizeof(X));

        for(int i=0; i<SIZE; i++)
            X[i][i] = k;
    }
};

Matrix operator *(Matrix &A, Matrix &B)
{
    Matrix M;
    for(int i=0; i<SIZE; i++)
    {
        for(int j=0; j<SIZE; j++)
        {
            long long tmp = 0;
            for(int k=0; k<SIZE; k++)
                tmp += (long long)A.X[i][k] * B.X[k][j];
            M.X[i][j] = tmp % MOD;
        }
    }
    return M;
}

Matrix pow(Matrix x, long long n)
{
    Matrix P(1);
    while(n)
    {
        if(n & 1) P = P * x;

        n >>= 1;
        x = x * x;
    }
    return P;
}

long long modpow(long long x, long long n)
{
    long long P = 1;

    while(n){
        if(n & 1) P = P * x % MOD;

        n >>= 1;
        x = x * x % MOD;
    }

    return P;
}
```

```
///////////////////////////MERGE SORT////////////////////////////////////
int cnt;

void msort(int v[],int t){

    if(t<=1) return;
    //for(int i=0; i<t; i++) cout<<v[i]<<" "; cout<<endl;
    int ta=t/2, tb=t-ta;
    int a[ta], b[tb];

    for(int i=0; i<ta; i++) a[i]=v[i];
    for(int i=0; i<tb; i++) b[i]=v[i+ta];

    msort(a,ta); msort(b,tb);
    int pa=0, pb=0, i=0;

    while(pa<ta && pb<tb){
        if(a[pa]>=b[pb])
            v[i++]=b[pb++], cnt+=ta-pa;
        else
            v[i++]=a[pa++];
    }
    while(pa<ta) v[i++]=a[pa++];
    while(pb<tb) v[i++]=b[pb++];
}

int main(){

    int a[]={1,2,3,4,5};
    cnt=0;
    msort(a,5);
    for(int i=0; i<5; i++) cout<<a[i]<<" "; cout<<endl;
    cout<<cnt;
    return 0;
}
```

```
///////////////////////////////SEGMENT TREE/////////////////////////////////

struct nodo{
    int sum, minN;
    nodo() { }
    nodo(int _sum, int _minN){
        sum = _sum;
        minN = _minN;
    }
}T[MAXN*4];

int n, a[MAXN];
void init(int b, int e, int node)
{
    if(b == e) T[node].sum = T[node].minN = a[b];
    else
    {
        int mid = (b + e)/2, le = 2*node + 1, ri = 2*node + 2;
        init(b, mid, le);
        init(mid + 1, e, ri);

        T[node].sum  = T[le].sum + T[ri].sum;
        T[node].minN = min(T[le].minN, T[ri].minN);
    }
}
void update(int b, int e, int node, int i, int val)
{
    if(i < b || i > e) return;

    if( b == e ) T[node].sum = T[node].minN = a[i] = val;
    else
    {
        int mid = (b + e)/2, le = 2*node + 1, ri = 2*node + 2;
        update(b, mid, le, i, val);
        update(mid + 1, e, ri, i, val);
        T[node].sum  = T[le].sum + T[ri].sum;
        T[node].minN  = min(T[le].minN, T[ri].minN);
    }
}

nodo query(int b, int e, int node, int i, int j)
{
    if(i <= b && e <= j) return T[node];
    int mid = (b + e) / 2, le = 2*node + 1, ri = 2*node + 2;
    if(j <= mid) return query(b, mid, le, i, j);
    else if(mid < i) return query(mid + 1, e, ri, i, j);
    else
    {
        nodo ret1 = query(b, mid, le, i, j);
        nodo ret2 = query(mid + 1, e, ri, i, j);
        nodo ret;
        ret.sum  = ret1.sum + ret2.sum;
        ret.minN = min(ret1.minN, ret2.minN);
        return ret;
    }
}
```

```
///////////////////////////////SEGMEMT TREE + LAZY/////////////////////

struct nodo{
    ll sum;
    ll add;// acumulado para el (LP)
    nodo() { }
    nodo(ll _sum, ll  _add){
        sum = _sum;
        add = _add;
    }
}T[MAXN * 4];

int a[MAXN];

void relax(int node, int b, int e){
    T[node].sum += T[node].add*((e-b)+1);
    if(b==e){
        //T[node].add=T[b].add;
    }else{
        T[node+node+1].add += T[node].add;
        T[node+node+2].add += T[node].add;
    }
    T[node].add = 0;
}

void init(int b, int e, int node){
    if(b==e)    T[node].sum = a[b];
    int mid = (b+e)/2, le = 2*node + 1, ri = 2*node + 2;
    init(b,mid,le);
    init(mid+1,e,ri);
    T[node].sum=T[le].sum+T[ri].sum;
}

void update(int b, int e, int node, int i,int j, int val)
{   relax(node,b,e);
    if(j < b || i > e) return;
    if(i <= b && e <= j){
        T[node].add += val;
        relax(node,b,e);
        return;
    }
    int mid = (b + e)/2, le = 2*node + 1, ri = 2*node + 2;

    update(b, mid, le, i,j, val);
    update(mid + 1, e, ri, i,j, val);

    T[node].sum  = T[le].sum + T[ri].sum;
}

nodo query(int b, int e,int node, int i, int j)
{
    relax(node,b,e);
    if(i <= b && e <= j) return T[node];
    int mid = (b + e) / 2, le = 2*node + 1, ri = 2*node + 2;
    if(j<=mid)  return query(b, mid, le, i, j);
    else if(mid<i)  return query(mid + 1, e, ri, i, j);
    else{
        nodo ret;
        nodo ret1=query(b, mid, le, i, j);
        nodo ret2=query(mid + 1, e, ri, i, j);

        ret.sum=ret1.sum+ret2.sum;
        return ret;
    }
}
```