MIST

Start->

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

#define ll long long
const int M=1e9+7;
const ll infinity = LLONG_MAX;
int dx[]={1,0,-1,0,1,1,-1,-1};//Right,Down,Left,Up,Right-Up,Right-Down,Left-Up,Left-Down
int dy[]={0,-1,0,1,1,-1,1,-1};
inline ll lcm(ll a,ll b) {return (a*b)/__gcd(a,b);}

#define rall(v) v.rbegin(),v.rend()
#define all(v) v.begin(),v.end()
#define print(x) cout<<x<<'\n';
#define YES cout<<"YES\n";
#define NO cout<<"NO\n";

void solve()
{

}

int main()
{
   ios_base::sync_with_stdio(false);
   cin.tie(NULL);
   cout.tie(NULL);

   int t=1;
   cin>>t;

   for(int i=1;i<=t;i++)
   {
      //cout<<"Case "<<i<<": ";
      solve();
   }
}
```

bfs matrix→

```
void bfs(int i,int j)
{
   vis[i][j]=1;
   queue<pair<int,int>> q;
   q.push({i,j});

   while(!q.empty())
   {
      if(vis[hx][hy]) return;
      pair<int,int> p = q.front();
      q.pop();
      int x=p.first,y=p.second;

      for(int a=0;a<4;a++)
      {
         int new_x=x+dx[a];
         int new_y=y+dy[a];
         if(!vis[new_x][new_y] && new_x>=0 && new_x<n
&& new_y>=0 && new_y<m && g[new_x][new_y]!='#')
         {
            vis[new_x][new_y]=1;
            q.push({new_x,new_y});
            dis[new_x][new_y]=dis[x][y]+1;
         }
      }
   }
}
```

bfs path finding→

```
vector<vector<int>> graph;
map<int,int> mp;//parent,child
vector<bool> vis;
int n,m;

void bfs()
{
   vector<int> parent,path;
   queue<int> q;
   q.push(0);
   mp[0]=-1;
   vis[0]=true;

   while(q.size())
   {
      int parent=q.front();
      q.pop();

      for(auto child : graph[parent])
      {
         if(!vis[child])
         {
            vis[child]=true;
            mp[child]=parent;
            q.push(child);
         }
      }
   }

   for(auto v=n-1;v!=-1;v=mp[v])
   {
      path.push_back(v);
   }

   if(!vis[n-1])
   {
      cout<<"IMPOSSIBLE\n";
      return;
   }
   cout<<path.size()<<endl;
   for(int i=path.size()-1;i>=0;i--)
   {
      cout<<path[i]+1<<" ";
   }
}
```

dfs→

```
void dfs(int index)
{
   vis[index]=1;
   cnt++;
   for(auto num : g[index])
   {
      if(!vis[num]) dfs(num);
   }
}
```

dfs matrix→

```
void dfs(int x,int y){
   vis[x][y]=true;

   for(int i=0;i<4;i++)
   {
      int new_x=x+dx[i];
      int new_y=y+dy[i];

      if(new_x<n && new_y<m && new_x>=0 && new_y>=0
&& !vis[new_x][new_y] && g[new_x][new_y]=='@')
      {
         dfs(new_x,new_y);
      }
   }
}
```

dijkstra →

```
const ll infinity = LLONG_MAX;
ll n;
vector<vector<pair<ll, ll>>> graph;

//It doesn't work for nagative cycle
void dijkstra(ll source) {
   ll mx = 0, cnt = 0;
   vector<ll> distance(n, infinity);
   distance[source] = 0;
   priority_queue<pair<ll, ll>, vector<pair<ll, ll>>,
greater<pair<ll, ll>>> pq;
   pq.push({0, source});

   while (!pq.empty()) {
      ll parent_node = pq.top().second;
      ll parent_weight = pq.top().first;
      pq.pop();

      if (parent_weight > distance[parent_node]) continue;

      for (auto child : graph[parent_node]) {
         ll child_node = child.first;
         ll child_weight = child.second;

         if (child_weight + parent_weight <
distance[child_node]) {
            distance[child_node] = child_weight +
parent_weight;
            pq.push({distance[child_node], child_node});
         }
      }
   }
}
```

Dijkstra for probability →

```
class Solution {
public:
   int n;
   vector<vector<pair<int,float>>> graph;

   double dijkstra(int src,int dst)
   {
      priority_queue<pair<double,int>> pq;
      pq.push({1.0,src});
      vector<double> max_prob(n,0.0);
      max_prob[src]=1.0;

      while(!pq.empty())
      {
         int parent_node=pq.top().second;
         double parent_probability=pq.top().first;
         pq.pop();

         if(parent_node==dst) return parent_probability;

         for(auto child : graph[parent_node])
         {
            int child_node=child.first;
            double child_probability=child.second;


if(parent_probability*child_probability>max_prob[child_no
de])
            {

max_prob[child_node]=parent_probability*child_probabili
ty;
               pq.push({max_prob[child_node],child_node});
            }
         }
      }
      return 0.0;
   }
   double maxProbability(int n, vector<vector<int>>&
edges, vector<double>& succProb, int start_node, int
end_node) {
      this->n=n;

      graph.resize(n);

      for(int i=0;i<edges.size();i++)
      {
         graph[edges[i][0]].push_back({edges[i][1],
succProb[i]});
         graph[edges[i][1]].push_back({edges[i][0],
succProb[i]});
      }

      return dijkstra(start_node,end_node);
   }
};
```

Dijkstra matrix →

```
int n,m;
vector<vector<int>> graph;

void dijkstra()
{
   vector<vector<ll>> dist(n, vector<ll>(m, infinity));
   priority_queue<pair<ll, pair<int, int>>, vector<pair<ll,
pair<int, int>>>, greater<pair<ll, pair<int, int>>>> pq;

   pq.push({graph[0][0],{0,0}});
   dist[0][0]=graph[0][0];

   while(!pq.empty())
   {
      int node_i=pq.top().second.first;
      int node_j=pq.top().second.second;
      int node_weight=pq.top().first;

      pq.pop();
      if(node_weight>dist[node_i][node_j]) continue;

      for(int i=0;i<4;i++)
      {
         int neighbour_i=node_i+dx[i];
         int neighbour_j=node_j+dy[i];

         if(neighbour_i>=0 && neighbour_j>=0 &&
neighbour_i<n && neighbour_j<m &&
graph[neighbour_i][neighbour_j]+node_weight<dist[neigh
bour_i][neighbour_j])
         {
            int
neighbour_weight=graph[neighbour_i][neighbour_j];

dist[neighbour_i][neighbour_j]=neighbour_weight+node_
weight;

pq.push({dist[neighbour_i][neighbour_j],{neighbour_i,neig
hbour_j}});
         }
      }
   }

   cout<<dist[n-1][m-1]<<endl;
}
```

Floyd Warshall →

```
//dijkstra cannot work for negative cycle but it can work in
negative cycle

class Solution {
public:
vector<vector<int>> graph;

bool floydwarshall(int n)
   {
      for(int k=0;k<n;k++)
         for(int i=0;i<n;i++)
            for(int j=0;j<n;j++)
               if(graph[i][k]<INT_MAX &&
graph[k][j]<INT_MAX)

graph[i][j]=min(graph[i][j],graph[i][k]+graph[k][j]);

      //detect negative cycle
      for(int i=0;i<n;i++)
         if(graph[i][i]<0)
            return true;

      return false;
   }
   int networkDelayTime(vector<vector<int>>& times, int n,
int k) {
      graph.assign(n,vector<int>(n,INT_MAX));
      for(int i =0;i<n;i++) graph[i][i]=0;
      for(auto x : times)
      {
         int u= x[0]-1;
         int v= x[1]-1;
         int w=x[2];
         graph[u][v]=w;
      }
      floydwarshall(n);
      int time=-1;
      k--;
      /*for(int i =0;i<n;i++)
      {
         for(int j=0;j<n;j++)
         {
            if(graph[i][j]==INT_MAX) cout<<"  ";
            else cout<<graph[i][j]<<" ";
         }
         cout<<endl;
      }*/
      for(int i =0;i<n;i++)
      {
         if(graph[k][i]==INT_MAX) return -1;
         if(graph[k][i]!=INT_MAX)time=max(graph[k][i],time);
      }

      return time;
   }
};
```

Bellman Foard →

Used for detecting negative cycle

```
ll n;

bool bell_man_ford(vector<vector<ll>> &edges)
{
   //vector<vector<ll>> edges(m,vector<ll>(3,0));
   vector<ll> distance(n,infinity);
   distance[0]=0;

   for(ll i=0;i<n-1;i++)
   {
      for(auto edge : edges)
      {
         ll source=edge[0];
         ll destination=edge[1];
         ll weight=edge[2];

         if(distance[destination]>distance[source]+weight)
distance[destination]=distance[source]+weight;
      }
   }
   for(auto edge : edges)
   {
      ll source=edge[0];
      ll destination=edge[1];
      ll weight=edge[2];

      if(distance[destination]>distance[source]+weight)
return true;
   }
   return false;
}
```

Krushkal →

```
class KruskalMST {
public:
   int n; // number of nodes
   vector<int> parent;
   vector<tuple<int, int, int>> edges;

   KruskalMST(int nodes) {
      n = nodes;
      parent.resize(n);
      iota(parent.begin(), parent.end(), 0); // parent[i] = i
   }

   void add_edge(int u, int v, int w) {
      edges.push_back({w, u, v});
   }

   int find_parent(int u) {
      if (parent[u] == u) return u;
      return parent[u] = find_parent(parent[u]);
   }

   bool union_sets(int u, int v) {
      int pu = find_parent(u);
      int pv = find_parent(v);
      if (pu != pv) {
         parent[pu] = pv;
         return true;
      }
      return false;
   }

   int get_mst_weight() {
      sort(all(edges));
      int total_weight = 0;
      for (auto [w, u, v] : edges) {
         if (union_sets(u, v)) {
            total_weight += w;
         }
      }
      return total_weight;
   }
};
```

Segment Tree →

```
vector<int>v,seg;//v(n+1) since it's 1 base indexing
seg(4*n)

int build(int start,int end,int node)
{
   if(start==end) return seg[node]=v[start];

   int mid=(start+end)/2;
   int left = build(start,mid,2*node);
   int right = build(mid+1,end,2*node+1);

   return seg[node]=left+right; //
seg[node]=seg[2*node]+seg[2*node+1] can be used
}

void update(int start,int end,int node,int ind,int val)
{
   if(start==end && end==ind) seg[node]+=val; //end == ind
is optional
```

```
   int mid=(start+end)/2;
   if(ind<=mid) update(start,mid,2*node,ind,val); //update
the left sub-tree
   else update(mid+1,end,2*node+1,ind,val); //update the
right sub-tree

   seg[node]=seg[2*node]+seg[2*node+1]; //update the
root for left and right sub-tree
}

int query(int start,int end,int node,int l,int r)
{
   //l for left of range in query r for right of range in query
   if(r<start || l>end) return 0; // l..r...start...end or
start...end...l...r
   if(l<=start && r>=end) return seg[node];
//l...start...end...r

   int mid=(start+end)/2;
   int left = query(start,mid,2*node,l,r);
   int right = query(mid+1,end,2*node+1,l,r);
   return left+right;
}
```

Lazy Segment Tree →

```
vector<ll> seg, lazy;

void propagate(int node, int range_i, int range_j) {
   if (lazy[node] != 0) {
      seg[node] += (range_j - range_i + 1) * lazy[node];
      if (range_i != range_j) {
         lazy[2*node] += lazy[node];
         lazy[2*node+1] += lazy[node];
      }
      lazy[node] = 0;
   }
}

void update(int i, int j, int val, int node, int range_i, int
range_j) {
   propagate(node, range_i, range_j);
   if (range_i > j || range_j < i) return;
   if (range_i >= i && range_j <= j) {
      lazy[node] += val;
      propagate(node, range_i, range_j);
      return;
   }
   int mid = (range_i + range_j) / 2;
   update(i, j, val, 2*node, range_i, mid);
   update(i, j, val, 2*node+1, mid+1, range_j);
   seg[node] = seg[2*node] + seg[2*node+1];
}

ll query(int i, int j, int node, int range_i, int range_j) {
   propagate(node, range_i, range_j);
   if (range_i > j || range_j < i) return 0;
   if (range_i >= i && range_j <= j) return seg[node];
   int mid = (range_i + range_j) / 2;
   return query(i, j, 2*node, range_i, mid) + query(i, j,
2*node+1, mid+1, range_j);
}
```

Binary exponentiation → (recursion)

```
ll bin_exp(ll a,ll b,ll c)
{  // (a^b)%c
   if(b==0) return 1;
   ll res=pow(a,b/2,c);
   res=(res*res)%c;
   if(b%2) return (res*a)%c;
   else return res;
}
```

Binary exponentiation →

```
ll bin_exp(ll base, ll exp, ll mod) {
   ll result = 1;
   base %= mod;
   while (exp > 0) {
      if (exp & 1) result = (result * base) % mod;
      base = (base * base) % mod;
      exp >>= 1;
   }
   return result;
}
```

Matrix exponentiation →

```
void multiplication(vector<vector<int>>
&mat,vector<vector<int>> &mat2,int m_size)
{
   vector<vector<int>> tmp(m_size, vector<int>(m_size,
0));
   for(int i=0;i<m_size;i++)
      for(int j=0;j<m_size;j++)
         for(int k=0;k<m_size;k++)
            tmp[i][j] = (tmp[i][j] + 1LL * mat[i][k] * mat2[k][j]
% M) % M;

   mat=tmp;
}

void exp(vector<vector<int>> &mat,int p,int m_size)
{
   vector<vector<int>>
result(m_size,vector<int>(m_size,0));
   for(int i=0;i<m_size;i++) result[i][i]=1;

   while(p>0){
      if(p%2) multiplication(result,mat,m_size);
      multiplication(mat,mat,m_size);
      p/=2;
   }
   mat=result;
}
```

Prime factorization →

```
void primeFactorization(int n) {
   // Print the number of 2s that divide n
   while (n % 2 == 0) {
      cout << 2 << " ";
      n /= 2;
   }

   // n must be odd at this point. So we can skip even
numbers (i.e., step by 2)
   for (int i = 3; i * i <= n; i += 2) {
      // While i divides n, print i and divide n
      while (n % i == 0) {
         cout << i << " ";
         n /= i;
      }
   }

   // This condition is to handle the case when n is a prime
number greater than 2
   if (n > 2)
      cout << n << " ";
}
```

Seive →

```
const int N=1e7+10;
vector<bool> is_prime(N,true);
void seive()
{
   is_prime[0]=is_prime[1]=false;
   for(int i=2;i*i<N;i++)
      if(is_prime[i])
         for(int j=i*i;j<N;j+=i)
            is_prime[j]=false;
}
```

Prime check →

```
bool isPrime(int n) {
   // Handle base cases
   if (n <= 1) return false;
   if (n == 2 || n == 3) return true;
   if (n % 2 == 0 || n % 3 == 0) return false;

   // Check for factors from 5 to √n
   for (int i = 5; i * i <= n; i += 6) {
      if (n % i == 0 || n % (i + 2) == 0) {
         return false;
      }
   }
   return true;
}
```

Fermet prime test →

```
bool isProbablyPrimeFermat(ll n, int iterations = 5) {
   if (n <= 3) return n > 1;
   if (n % 2 == 0) return false;

   for (int i = 0; i < iterations; ++i) {
      ll a = 2 + rand() % (n - 3); // a in [2, n-2]
      if (bin_exp(a, n - 1, n) != 1)
         return false;
   }
   return true;
}
```

Spf →

```
int spf[N];

void Spf() {
   for(int i=2;i<=N;i++) spf[i]=i;

   for(int i=2;i*i<N;i++)
      if(spf[i]==i)
         for(int j=i*i;j<N;j+=i)
            if(spf[j]==j) spf[j]=i;
}
```

All factor of a number →

```
cin>>n;
for(int i=1;i*i<=n;i++)
   if(n%i==0)
      if(n/i==i) cout<<i<<" ";
      else cout<<i<<" "<<n/i<<" ";
```

Kmp →

```
vector<int> lps;
string txt,pat;
int n,m;

void lps_cal()
{
   int i=1,len=0;

   while(i<m)
   {
      if(pat[i]==pat[len])
      {
         len++;
         lps[i]=len;
         i++;
      }
      else
      {
         if(len==0) i++;
         else len=lps[len-1];
      }
   }
}

void kmp()
{
   m=pat.length();
   n=txt.length();
   lps_cal();
   int i=0,j=0;
   vector<int> v;

   while(i<n)
   {
      if(txt[i]==pat[j])
      {
         i++;
         j++;
      }
      else
      {
         if(j==0) i++;
         else j=lps[j-1];
      }

      if(j==m)
      {
         cout<<i-j<<endl;
         j=lps[j-1];
      }
   }

}
```

String hashing →

```
class Hash
{
   private:
      int
n,mod1=1e9+7,mod2=1e9+9,base1=2147483647,base2=
2147476381;
      string str;
      vector<int> ph1,ph2;
      vector<int> pow1,pow2;

      void prefix_hash()
      {
         int x=0,y=0;
         pow1[0]=1,pow2[0]=1;

         for(int i=0;i<n;i++)
         {
```

```
            pow1[i+1]=(1LL*pow1[i]*base1)%mod1;
            pow2[i+1]=(1LL*pow2[i]*base2)%mod2;
            x=(1LL*x*base1+str[i])%mod1;
            y=(1LL*y*base2+str[i])%mod2;
            ph1[i]=x;
            ph2[i]=y;
        }
    }

  public:
    Hash(string s)
    {
        str=s;
        n=str.size();
        ph1.resize(n);
        ph2.resize(n);
        pow1.resize(n+1);
        pow2.resize(n+1);
        prefix_hash();
    }

    pair<int,int> cal_hash(int left,int right)
    {
        if(left==0) return {ph1[right],ph2[right]};
        else
        {
            int x=((ph1[right]-(1LL*ph1[left-1]*pow1[right-
left+1])%mod1)+mod1)%mod1;
            int y=((ph2[right]-(1LL*ph2[left-1]*pow2[right-
left+1])%mod2)+mod2)%mod2;
            return {x,y};
        }
    }
};
```

Knapsack→

```
// Function to solve the 0/1 Knapsack problem using
dynamic programming
int knapsack(vector<int>& wt, vector<int>& val, int n, int
W) {
  // Create a 2D DP table with dimensions n x W+1 and
initialize it with zeros
  vector<vector<int>> dp(n, vector<int>(W + 1, 0));

  // Base condition: Fill in the first row for the weight of
the first item
  for (int i = wt[0]; i <= W; i++) {
    dp[0][i] = val[0];
  }

  // Fill in the DP table using a bottom-up approach
  for (int ind = 1; ind < n; ind++) {
    for (int cap = 0; cap <= W; cap++) {
      // Calculate the maximum value by either excluding
the current item or including it
      int notTaken = dp[ind - 1][cap];
      int taken = INT_MIN;

      // Check if the current item can be included without
exceeding the knapsack's capacity
      if (wt[ind] <= cap) {
        taken = val[ind] + dp[ind - 1][cap - wt[ind]];
      }

      // Update the DP table
      dp[ind][cap] = max(notTaken, taken);
    }
  }

  // The final result is in the last cell of the DP table
  return dp[n - 1][W];
}
```

Convex Hull->>

```
ll area(vector<pair<ll,ll>> &v)
{
  ll ar=0,n=v.size();
  for(int i=0;i<n;i++)
  {
    ar+=v[i].first*v[(i+1)%n].second-
v[i].second*v[(i+1)%n].first;
  }

  return ar;
}

ll cross(pair<ll,ll> a,pair<ll,ll> b,pair<ll,ll> c)
{
  return (b.first-a.first)*(c.second-a.second)-(c.first-
a.first)*(b.second-a.second);
}

void solve()
{
  ll n;
  while(cin>>n && n)
  {
    vector<pair<ll,ll>> v(n),lower,upper;

    for(auto &[x,y] : v) cin>>x>>y;
    sort(all(v));

    for(int i=0;i<n;i++)
    {
      while(lower.size()>=2 && cross(lower[lower.size()-
2],lower.back(),v[i])<=0) lower.pop_back();
```

```
        lower.push_back(v[i]);
      }
      for(int i=n-1;i>=0;i--)
      {
        while(upper.size()>=2 && cross(upper[upper.size()-
2],upper.back(),v[i])<=0) upper.pop_back();
        upper.push_back(v[i]);
      }

      upper.pop_back();
      lower.pop_back();

      lower.insert(lower.end(),upper.begin(),upper.end());

      double ar=area(lower);

      cout<<ar/2<<'\n';
   }

}
```