1 Number Theory

Exgcd.cpp

```cpp
#define pii pair<int, int>
pii exgcd(int a, int b){
    if(b == 0) return mp(1, 0);
    pii ans = exgcd(b, a % b);
    return mp(ans.S, ans.F - a / b * ans.S);
}
```

catalan.cpp

```
C_0=1
C_n = $\sum C_i * C_{n-1-i}$ = $\frac{1}{n+1} C_n^{2*n}$
```

fft.cpp

```cpp
using cp=complex<double>;
const double pi=acos(-1);
constexpr ll mxN=2e5+1;
ll n,m;
vc<cp> a(mxN<<2,{0,0}),b(mxN<<2,{0,0});
void fft(ll sz,vc<cp> &arr,ll inv){
  for(ll i=1,j=0;i<sz;++i){
    ll bit=sz>>1;
    for(;j&bit;j^=bit,bit>>=1); j^=bit;
    if(i<j) swap(arr[i],arr[j]);
  }
  for(ll l=2;l<=sz;l<<=1){
    double ang=2*pi/l*(inv?-1:1);
    cp wlen=cp(cos(ang),sin(ang));
    for(ll i=0;i<sz;i+=l){
      cp w=cp(1,0);
      for(ll j=0;j<l/2;++j){
        cp x=arr[i+j],y=arr[i+j+l/2]*w;
        arr[i+j]=x+y,arr[i+j+l/2]=x-y;
        w*=wlen;
      }
    }
  }
  if(inv) for(auto&v:arr) v/=sz;
}
```

findPrime.cpp

```cpp
mobius[1]=1;
forn(i,2,mxN){
  if(!prime[i]) prime[i]=i,mobius[i]=-1,etf[i]=i-1,st[++sz
      ]=i;
  forn(j,1,sz+1){
    if(i*st[j]>=mxN) break;
    prime[i*st[j]]=st[j];
    if(i%st[j]==0){
      mobius[i*st[j]]=0;
      etf[i*st[j]]=etf[i]*st[j];
      break;
    }
    else mobius[i*st[j]]=mobius[i]*-1,etf[i*st[j]]=etf[i]*(
        st[j]-1);
  }
}
```

matrixMultiple.cpp

```cpp
vector<vector<int>> matrixMultiply(const vector<vector<int
    >>& A, const vector<vector<int>>& B) {
    int rowA = A.size();
    int colA = A[0].size();
    int colB = B[0].size();
    vector<vector<int>> C(rowA, vector<int>(colB, 0));
    for (int i = 0; i < rowA; ++i) {
        for (int j = 0; j < colB; ++j) {
            for (int k = 0; k < colA; ++k) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
    return C;
}
```

2 Tree

LCA.cpp

```cpp

ll pa[mxN][20],d[mxN]{};
vc<ll> adj[mxN]
void dfs(ll u=1,ll p=0){d[u]=d[p]+1,pa[u][0]=p; for(auto&v:
    adj[u]) if(v!=p) dfs(v,u);}
ll lca(ll u,ll v){
    if(d[u]<d[v]) swap(u,v); forr(i,20,0) if(d[u]-(1<<i)>=d
        [v]) u=pa[u][i]; if(u==v) return u;
    forr(i,20,0) if(pa[u][i]!=pa[v][i]) u=pa[u][i],v=pa[v][
        i]; return pa[u][0];
}

void init(int n) {
  dfs(1, 0, 1);
  for (int j = 1; j < 20; j++) {
    for (int i = 1; i <= n; i++) {
      parent[i][j] = parent[parent[i][j - 1]][j - 1];
    }
  }
}
```

MST.cpp

```cpp
//kruskal
//vertex is 1 index
#define mls multiset

ll n,pa[mxN];//memset(pa,-1,sizeof(pa));
mls<ar<ll,3>> adj;

ll find(ll x){return pa[x]<0?x:pa[x]=find(pa[x]);}
void un(ll x,ll y){
    x=find(x),y=find(y); if(x==y) return;
    if(-pa[x]>-pa[y]) swap(x,y); pa[y]+=pa[x],pa[x]=y;
}
void kruskal(){
    ll cnt=0,wi=0;
    while(!adj.empty() && cnt<n-1){
        auto [w,u,v]=*adj.begin(); adj.erase(adj.begin());
        if(find(u)!=find(v)) wi+=w,un(u,v);
    }
}
```

Tree Diameter.cpp

```cpp
#define emp emplace_back()
ll n,res=0;
vc<ll> adj[mxN],dp(mxN);

void dfs(ll u=1,ll p=0){dp[u]=0; for(auto&v:adj[u]) if(v!=p
    ) dfs(v,u),res=max(res,dp[u]+dp[v]+1),dp[u]=max(dp[u],
    dp[v]+1);}

int main()
{
    cin>>n; forn(i,0,n-1){ll u,v; cin>>u>>v; adj[u].emp(v),
        adj[v].emp(u);}
    dfs();
    cout << res << '\n';
}
```

3 Graph

All points minimum distance.cpp

```cpp
for (k = 1; k <= n; k++) {
  for (x = 1; x <= n; x++) {
    for (y = 1; y <= n; y++) {
      f[x][y] = min(f[x][y], f[x][k] + f[k][y]);
    }
  }
}
```

Articulation Point.cpp

```cpp
//undirected graph articulation point
ll n,tim=0,art=0;
vc<ll> dfn(mxN,0),low(mxN);
void tarjan(ll u,ll p){
    ll cnt=0;
    dfn[u]=low[u]=++tim;
    for(auto&v:adj[u]){
        if(!dfn[v]){
            tarjan(v,u);
            low[u]=min(low[u],low[v]);
            if(low[v]>=dfn[v] && u!=root){ //root=dfn[1]
                ++art;
            }
            cnt++;
        }
        else low[u]=min(low[u],dfn[v]);
    }
    if(u==root && cnt>1) art++;
}
```

BFS topoSort.cpp

```cpp
void topo_sort() {
  int n, m;
  cin >> n >> m;
  vector<vector<int>> graph(n+1);
  for (int i = 0; i < m; i++) {
    int a, b;
    cin >> a >> b;
    graph[a].push_back(b);
  }
  vector<int> rudu(n+1, 0);
  for (auto &node : graph) {
    for (auto &now : node) {
      rudu[now]++;
    }
  }
  queue<int> que;
  for (int i = 1; i <= n; i++) {
    if (rudu[i] == 0) que.push(i);
  }
  vector<int> topo;
  while (!que.empty()) {
    int top = que.front();
    que.pop();
    topo.push_back(top);
    for (auto &i : graph[top]) {
      rudu[i]--;
      if (rudu[i] == 0) que.push(i);
    }
  }
  if (topo.size() == n) {
    for (auto &i : topo) cout << i << " ";
  }
  else
    cout << "IMPOSSIBLE";
}
```

BellmanFord.cpp

```cpp
//SFPA

bool sfpa(){
    vc<ll> vs(n,0);
    queue<ll> q; q.emplace(start);
    while(!q.empty()){
        ll u=q.front(); q.pop(),vs[u]=0;
        for(auto&[v,w]:adj[u]){
            if(d[v]>d[u]+w){
                d[v]=d[u]+w;
                if(!vs[v]){vs[v]=1,cnt[v]++; if(cnt[v]>n)
                    return false; q.emplace(v);}
            }
        }
    }
    return true;
}
```

DFS topoSort Find Cycle.cpp

```cpp
const int N = 100005;
vector<vector<int>> graph(N+1);
vector<bool> been(N+1, 0);
vector<bool> onStack(N+1, 0);
vector<int> cycle;
vector<int> topo;
bool dfs(int ind) {
  been[ind] = onStack[ind] = 1;
  for (auto &i : graph[ind]) {
    if (been[i] == 0) {
      if (onStack[i]) {
        cycle.pb(i);
        return true;
      }
      if (dfs(i)) return true;
    }
    else if (onStack[i]) {
      been[ind] = 0;
      cycle.pb(i);
      return dfs(i);
    }
  }
  topo.pb(ind);
  onStack[ind] = 0;
  return 0;
}

void solve() {
  int n, m; cin >> n >> m;
  for (int i = 0; i < m; i++) {
    int a, b; cin >> a >> b;
    graph[a].pb(b);
  }
  bool ch = 0;
  for (int i = 1; i <= n; i++) {
    if (been[i] == 0) {
      if (dfs(i)) {
        ch = 1;
        break;
      }
    }
  }
  if (ch) {
    cout << "cycle found: " << endl;
    for (auto &i : cycle) cout << i << " ";
    cout << cycle.front();
  }
  reverse(topo.begin(), topo.end());
  for (auto &i : topo) cout << i << " ";
  return;
}
```

Dijkstra.cpp

```cpp
//不能處裡含有"負邊"的圖
#define pii pair<int, int>
//#define int long long
#define MAX INT_MAX //LLONG_MAX
vector<vector<pii>> graph;
vector<int> dist;
vector<int> parent;
void init(int n) {
  graph = vector<vector<pii>>(n+1);
  dist = vector<int>(n+1, MAX);
  parent = vector<int>(n+1, -1);
  return;
}

void dijkstra(int from/*, int target*/) {
  dist[from] = 0;
  priority_queue<pii, vector<pii>, greater<pii>> pque;
  pque.push({0, from});
  while (!pque.empty()) {
    int node = pque.top().second;
    int pdist = pque.top().first;
    pque.pop();
    // if (node == target) break;
    if (pdist != dist[node]) continue;
    for (auto &[to, edist] : graph[node]) {
      if (pdist + edist < dist[to]) {
        dist[to] = pdist + edist;
        pque.push({dist[to], to});
        parent[to] = node;
      }
    }
  }
  return;
}
```

Edmond.cpp

```

constexpr ll mxN=505;
ll n,m,adj[mxN][mxN]{},g[mxN][mxN]{};//adj=capacity
vc<ll> pa;
ll bfs(){
  queue<pll> q; q.emplace(pll{0,inf});
  while(!q.empty()){
    auto [u,flow]=q.front(); q.pop();
    forn(i,0,n)
      if(g[u][i] && adj[u][i] && pa[i]==-1){
        pa[i]=u;
        if(i==n-1) return min(flow,adj[u][i]);
        q.emplace(pll{i,min(flow,adj[u][i])});
      }
  }
  return 0;
}
void solve(istream &cin){
  cin>>n>>m;
  pa.resize(n);
  forn(i,1,m+1){ll u,v; cin>>u>>v,u--,v--; adj[u][v]=g[u][v]=1,adj[v][u]=g[v][u]=1;}
  ll f=0;
  fill(all(pa),-1),pa[0]=0;
  for(ll nf;nf=bfs();f+=nf){
    ll cur=n-1;
    while(cur) adj[pa[cur]][cur]-=nf,adj[cur][pa[cur]]+=nf,cur=pa[cur];
    fill(all(pa),-1),pa[0]=0;
  }
  vc<pll> res;
  forn(i,0,n) forn(j,0,n) if(g[i][j] && pa[i]!=-1 && pa[j]==-1) res.emp(pll{i,j});
  cout << res.size() << '\n';
  for(auto&[u,v]:res) cout << u+1 << ' ' << v+1 << '\n';
}
int main()
{
    fast_io;
    ll testcase;
    // cin>>testcase;
    testcase=1;
    while(testcase--)
        solve(cin);
    return 0;
}
```

Find Bridge.cpp

```
int low[MAXN], dfn[MAXN], dfs_clock;
bool isbridge[MAXN];
vector<int> G[MAXN];
int cnt_bridge;
int father[MAXN];

void tarjan(int u, int fa) {
  father[u] = fa;
  low[u] = dfn[u] = ++dfs_clock;
  for (int i = 0; i < G[u].size(); i++) {
    int v = G[u][i];
    if (!dfn[v]) {
      tarjan(v, u);
      low[u] = min(low[u], low[v]);
      if (low[v] > dfn[u]) {
        isbridge[v] = true;
        ++cnt_bridge;
      }
    } else if (dfn[v] < dfn[u] && v != fa) {
      low[u] = min(low[u], dfn[v]);
    }
  }
}
```

Strong Connect Parts.cpp

```
int dfn[N], low[N], dfncnt, s[N], in_stack[N], tp;
int scc[N], sc;  // 结点 i 所在 SCC 的编号
int sz[N];       // 强连通 i 的大小

void tarjan(int u) {
  low[u] = dfn[u] = ++dfncnt, s[++tp] = u, in_stack[u] = 1;
  for (int i = h[u]; i; i = e[i].nex) {
    const int &v = e[i].t;
    if (!dfn[v]) {
      tarjan(v);
      low[u] = min(low[u], low[v]);
    } else if (in_stack[v]) {
      low[u] = min(low[u], dfn[v]);
    }
  }
  if (dfn[u] == low[u]) {
    ++sc;
    while (s[tp] != u) {
      scc[s[tp]] = sc;
      sz[sc]++;
      in_stack[s[tp]] = 0;
      --tp;
    }
    scc[s[tp]] = sc;
    sz[sc]++;
    in_stack[s[tp]] = 0;
    --tp;
  }
}
```

dinic.cpp

```

constexpr ll mxN=505;
class fe{
  public:
    ll u,v,cp,fw;
    fe(){}
    fe(ll U,ll V,ll CP):u(U),v(V),cp(CP){fw=0;}
};
ll n,m,sz=0;
vc<fe> edg;
queue<ll> q;
vc<ll> adj[mxN],ptr,dep;
bool bfs(){
  while(!q.empty()){
    ll u=q.front(); q.pop();
    for(auto&id:adj[u]){
      if(edg[id].cp==edg[id].fw || dep[edg[id].v]!=-1)
          continue;
      dep[edg[id].v]=dep[u]+1,q.emplace(edg[id].v);
    }
  }
  return dep[n-1]!=-1;
}
ll dinic(ll u,ll push){
  if(!push) return 0;
  if(u==n-1) return push;
  for(ll &cid=ptr[u];cid<adj[u].size();++cid){
    ll id=adj[u][cid];
    ll v=edg[id].v;
    if(dep[v]!=dep[u]+1) continue;
    ll tr=dinic(v,min(push,edg[id].cp-edg[id].fw));
    if(!tr) continue;
    edg[id].fw+=tr,edg[id^1].fw-=tr;
    return tr;
  }
  return 0;
}
void solve(istream &cin){
  cin>>n>>m;
  ptr.resize(n),dep.resize(n);
  forn(i,1,m+1){
    ll u,v,c; cin>>u>>v>>c,u--,v--;
    edg.emp(fe{u,v,c}),edg.emp(fe{v,u,0});
    adj[u].emp(sz),adj[v].emp(sz+1),sz+=2;
  }
  ll res=0;
  for(;;){
    ll flow;
    fill(all(dep),-1);
    q.emplace(0),dep[0]=0;
    if(!bfs()) break;
    fill(all(ptr),0);
    for(;flow=dinic(0,inf);res+=flow);
  }
  cout << res << '\n';
}
int main()
{
    fast_io;
    ll testcase;
    // cin>>testcase;
    testcase=1;
    while(testcase--)
        solve(cin);
    return 0;
}
```

4 Geometry

Convex Hull.cpp

```cpp
#pragma GCC optimize("Ofast,unroll-loops,no-stack-protector,fast-math")
#define ll long long
#define pb push_back
#define mod (ll)(1e9 + 7)
#include <bits/stdc++.h>

using namespace std;

struct dat
{
    ll x, y;
    dat(ll x, ll y) : x(x), y(y) {}
    dat() = default;
    dat operator+(const dat &b)const { return dat(x + b.x, y + b.y); }
    dat operator-(const dat &b)const { return dat(x - b.x, y - b.y); }
    ll operator^(const dat &b)const { return x * b.y - y * b.x; }
    bool operator<(const dat &b)const { return x == b.x ? y < b.y : x < b.x; }
};

dat input[200005];

int main()
{
    ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    ll n;
    cin >> n;
    for (ll i = 0; i < n; i++)
    {
        ll tx, ty;
        cin >> tx >> ty;
        input[i] = dat(tx, ty);
    }
    sort(input, input + n);
    vector<dat> ans(2 * n);
    ll place = 0;
    for (ll i = 0; i < n; i++)
    {
        while (place > 1 && ((input[i] - ans[place - 1]) ^ (input[i] - ans[place - 2])) > 0)
            place--;
        ans[place++] = input[i];
    }
    ll np = place;
    for (ll i = n - 2; i >= 0; i--)
    {
        while (place > np && ((input[i] - ans[place - 1]) ^ (input[i] - ans[place - 2])) > 0)
        {
            place--;
        }
        ans[place++] = input[i];
    }
    place--;
    cout << place << "\n";
    for (ll i = 0; i < place; i++)
    {
        cout << ans[i].x << " " << ans[i].y;
        cout << "\n";
    }
    //cout.flush(); system("pause");
    return 0;
}
```

Determine whether a polygon is convex.cpp

```cpp
#include <vector>
using namespace std;

struct Point {
    double x, y;
};

double cross(const Point& a, const Point& b, const Point& c) {
    // 計算向量ab x bc
    double x1 = b.x - a.x, y1 = b.y - a.y;
    double x2 = c.x - b.x, y2 = c.y - b.y;
    return x1 * y2 - y1 * x2;
}

bool isConvex(const vector<Point>& poly) {
    int n = poly.size();
    if (n < 3) return false;
    int sign = 0;
    for (int i = 0; i < n; ++i) {
        double c = cross(poly[i], poly[(i + 1) % n], poly[(i + 2) % n]);
        if (c != 0) {
            if (sign == 0)
                sign = (c > 0) ? 1 : -1;
            else if ((c > 0 && sign < 0) || (c < 0 && sign > 0))
                return false;
        }
    }
    return true;
}
```

Line Crossing.cpp

```cpp
struct point
{
  ll x, y;
  point() = default;
  point(ll a, ll b) : x(a), y(b) {}
  friend istream &operator>>(istream &in, point &b) { in >> b.x >> b.y; return in; }
  friend ostream &operator<<(ostream &ou, const point &b) { ou << b.x << " " << b.y; return ou; }
  point operator-(point &b) { return point(x - b.x, y - b.y); }
  inline ll cross(const point &b) { return x * b.y - y * b.x; }
};

inline int ll_prot(ll temp) { return temp > 0 ? 1 : temp < 0 ? -1 : 0; }

inline bool check_touch(point a, point b, point c, point d)
{
  if (max(a.x, b.x) < min(c.x, d.x)) return false;
  if (max(a.y, b.y) < min(c.y, d.y)) return false;
  if (max(c.x, d.x) < min(a.x, b.x)) return false;
  if (max(c.y, d.y) < min(a.y, b.y)) return false;
  if (ll_prot((a - b).cross(a - c)) * ll_prot((a - b).cross(a - d)) > 0) return false;
  if (ll_prot((c - d).cross(c - b)) * ll_prot((c - d).cross(c - a)) > 0) return false;
  return true;
}
```

Ray method to determine whether a point is inside a polygon.cpp

```cpp
#include <vector>
using namespace std;

struct Point {
    double x, y;
};

bool onSegment(Point a, Point b, Point p) {
    // 判斷點p是否在線段ab上
    return (min(a.x, b.x) <= p.x && p.x <= max(a.x, b.x)) &&
           (min(a.y, b.y) <= p.y && p.y <= max(a.y, b.y)) &&
           ((b.x - a.x) * (p.y - a.y) == (b.y - a.y) * (p.x - a.x));
}

bool pointInPolygon(const vector<Point>& poly, Point p) {
    int n = poly.size();
    int cnt = 0;
    for (int i = 0; i < n; ++i) {
        Point a = poly[i], b = poly[(i + 1) % n];
        // 檢查是否在邊上
        if (onSegment(a, b, p)) return true;
        // 保證a.y <= b.y
        if (a.y > b.y) swap(a, b);
        // 射線與邊相交
        if (a.y <= p.y && p.y < b.y) {
            double x = a.x + (b.x - a.x) * (p.y - a.y) / (b.y - a.y);
            if (x > p.x) cnt++;
        }
    }
    return cnt % 2 == 1;
}
```

Rotating Calipers 求凸包直徑.cpp

```cpp
struct Point {
    double x, y;
};

double dist2(const Point& a, const Point& b) {
    double dx = a.x - b.x, dy = a.y - b.y;
    return dx * dx + dy * dy;
}

// 假設 convex 是已經逆時針排序的凸包
double convexHullDiameter(const vector<Point>& convex) {
    int n = convex.size();
    if (n == 1) return 0;
    if (n == 2) return dist2(convex[0], convex[1]);
    double res = 0;
    for (int i = 0, j = 1; i < n; ++i) {
        while (dist2(convex[i], convex[(j + 1) % n]) >
             dist2(convex[i], convex[j]))
            j = (j + 1) % n;
        res = max(res, dist2(convex[i], convex[j]));
    }
    return sqrt(res);
}
```

all solutions for Diophantine Equation in given range.cpp

```cpp
/*
    1. General solution to the linear Diophantine equation:
        a * x + b * y = c
       If (x0, y0) is a particular solution and g = gcd(a, b):
           x = x0 + k * (b / g)
           y = y0 - k * (a / g)
           for any integer k
    2. Number of integer solutions (x, y) within bounds:
           minx <= x <= maxx
           miny <= y <= maxy
       If lx and rx are the valid range for x:
           Number of solutions = (rx - lx) / abs(b / g) + 1
*/
void shift_solution(int & x, int & y, int a, int b, int cnt) {
    x += cnt * b;
    y -= cnt * a;
}

int find_all_solutions(int a, int b, int c, int minx, int maxx, int miny, int maxy) {
    int x, y, g;
    if (!find_any_solution(a, b, c, x, y, g))
        return 0;
    a /= g;
    b /= g;

    int sign_a = a > 0 ? +1 : -1;
    int sign_b = b > 0 ? +1 : -1;

    shift_solution(x, y, a, b, (minx - x) / b);
    if (x < minx)
        shift_solution(x, y, a, b, sign_b);
    if (x > maxx)
        return 0;
    int lx1 = x;

    shift_solution(x, y, a, b, (maxx - x) / b);
    if (x > maxx)
        shift_solution(x, y, a, b, -sign_b);
    int rx1 = x;

    shift_solution(x, y, a, b, -(miny - y) / a);
    if (y < miny)
        shift_solution(x, y, a, b, -sign_a);
    if (y > maxy)
        return 0;
    int lx2 = x;

    shift_solution(x, y, a, b, -(maxy - y) / a);
    if (y > maxy)
        shift_solution(x, y, a, b, sign_a);
    int rx2 = x;

    if (lx2 > rx2)
        swap(lx2, rx2);
    int lx = max(lx1, lx2);
    int rx = min(rx1, rx2);

    if (lx > rx)
        return 0;
    return (rx - lx) / abs(b) + 1;
}
```

closest pair of points.cpp

```cpp
struct Point {
    double x, y;
    bool operator<(const Point& p) const { return x < p.x || (x == p.x && y < p.y); }
};

double dist(const Point& a, const Point& b) {
    double dx = a.x - b.x, dy = a.y - b.y;
    return sqrt(dx * dx + dy * dy);
}

double closestPair(vector<Point>& pts, int l, int r) {
    if (r - l <= 3) {
        double d = 1e20;
        for (int i = l; i < r; ++i)
            for (int j = i + 1; j < r; ++j)
                d = min(d, dist(pts[i], pts[j]));
        sort(pts.begin() + l, pts.begin() + r, [](const Point& a, const Point& b) { return a.y < b.y; });
        return d;
    }
    int m = (l + r) / 2;
    double xmid = pts[m].x;
    double d = min(closestPair(pts, l, m), closestPair(pts, m, r));
    vector<Point> tmp;
    merge(pts.begin() + l, pts.begin() + m, pts.begin() + m, pts.begin() + r, back_inserter(tmp),
          [](const Point& a, const Point& b) { return a.y < b.y; });
    copy(tmp.begin(), tmp.end(), pts.begin() + l);
    vector<Point> strip;
    for (int i = l; i < r; ++i)
        if (fabs(pts[i].x - xmid) < d)
            strip.push_back(pts[i]);
    for (int i = 0; i < strip.size(); ++i)
        for (int j = i + 1; j < strip.size() && strip[j].y - strip[i].y < d; ++j)
            d = min(d, dist(strip[i], strip[j]));
    return d;
}

// 用法：
// vector<Point> pts = ...;
// sort(pts.begin(), pts.end());
// double ans = closestPair(pts, 0, pts.size());
```

動態凸包.cpp

```cpp
struct Point {
    long long x, y;
    bool operator<(const Point& p) const {
        return x < p.x || (x == p.x && y < p.y);
    }
};

long long cross(const Point& a, const Point& b, const Point& c) {
    // 向量ab x bc
    return (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x - a.x);
}

struct DynamicConvexHull {
    set<Point> up, dn; // 上凸包、下凸包

    void insert(const Point& p) {
        insertHull(up, p, false); // 上凸包
        insertHull(dn, p, true);  // 下凸包
    }

    // 取得逆時針順序的凸包點（不重複端點）
    vector<Point> getConvexHull() const {
        vector<Point> res;
        for (auto p : up) res.push_back(p);
        vector<Point> dnvec(dn.rbegin(), dn.rend());
        for (int i = 1; i + 1 < dnvec.size(); ++i) // 去掉首尾重複
            res.push_back(dnvec[i]);
        return res;
    }

private:
    // flag: false=上凸包, true=下凸包
    void insertHull(set<Point>& hull, const Point& p, bool flag) {
```

```
        auto it = hull.insert(p).first;
        // 刪除左側不在凸包上的點
        while (it != hull.begin()) {
            auto prv = prev(it);
            if (prv == hull.begin()) break;
            auto pprv = prev(prv);
            if ((cross(*pprv, *prv, *it) > 0) ^ flag)
                hull.erase(prv);
            else break;
        }
        // 刪除右側不在凸包上的點
        while (next(it) != hull.end()) {
            auto nxt = next(it);
            if (next(nxt) == hull.end()) break;
            auto nnxt = next(nxt);
            if ((cross(*it, *nxt, *nnxt) > 0) ^ flag)
                hull.erase(nxt);
            else break;
        }
    }
};
```

多邊形重心計算.cpp

```
struct Point {
    double x, y;
};

Point polygonCentroid(const vector<Point>& poly) {
    double cx = 0, cy = 0, area = 0;
    int n = poly.size();
    for (int i = 0; i < n; ++i) {
        double cross = poly[i].x * poly[(i + 1) % n].y -
            poly[(i + 1) % n].x * poly[i].y;
        cx += (poly[i].x + poly[(i + 1) % n].x) * cross;
        cy += (poly[i].y + poly[(i + 1) % n].y) * cross;
        area += cross;
    }
    area /= 2.0;
    cx /= (6.0 * area);
    cy /= (6.0 * area);
    return {cx, cy};
}
```

多邊形面積計算（Shoelace Formula）.cpp

```
struct Point {
    double x, y;
};

double polygonArea(const vector<Point>& poly) {
    double area = 0;
    int n = poly.size();
    for (int i = 0; i < n; ++i) {
        area += poly[i].x * poly[(i + 1) % n].y;
        area -= poly[i].y * poly[(i + 1) % n].x;
    }
    return fabs(area) / 2.0;
}
```

找三點組成最大圓.cpp

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<ll, ll> P;

ll cross(P a, P b) { return a.first * b.second - a.second *
    b.first; }
ll dist2(P a, P b) {
    ll dx = a.first - b.first, dy = a.second - b.second;
    return dx * dx + dy * dy;
}
double dist(P a, P b) { return sqrt(dist2(a, b)); }

// Andrew's Monotone Chain
vector<P> convex_hull(vector<P> pts) {
    sort(pts.begin(), pts.end());
    int n = pts.size(), k = 0;
    vector<P> hull(2 * n);
    for (int i = 0; i < n; ++i) {
        while (k >= 2 && cross({hull[k-1].first-hull[k-2].
            first, hull[k-1].second-hull[k-2].second},
                               {pts[i].first-hull[k-1].
                                   first, pts[i].second-
                                   hull[k-1].second}) <=
                                   0)
            --k;
        hull[k++] = pts[i];
    }
    for (int i = n-2, t = k; i >= 0; --i) {
        while (k > t && cross({hull[k-1].first-hull[k-2].
            first, hull[k-1].second-hull[k-2].second},
                              {pts[i].first-hull[k-1].first
                                  , pts[i].second-hull[k
                                  -1].second}) <= 0)
            --k;
        hull[k++] = pts[i];
    }
    hull.resize(k-1);
    return hull;
}

double circumradius(P a, P b, P c) {
    double A = dist(b, c), B = dist(a, c), C = dist(a, b);
    double S = abs(cross({b.first-a.first, b.second-a.
        second},
                         {c.first-a.first, c.second-a.
                             second})) / 2.0;
    if (S == 0) return 0; // 共線
    return (A * B * C) / (4.0 * S);
}

int main() {
    int n;
    cin >> n;
    vector<P> pts(n);
    for (int i = 0; i < n; ++i) cin >> pts[i].first >> pts[
        i].second;
    vector<P> hull = convex_hull(pts);
    int m = hull.size();
    double ans = 0;
    // 旋轉卡殼
    for (int i = 0; i < m; ++i) {
        int j = (i + 1) % m, k = (j + 1) % m;
        do {
            while (true) {
                int k2 = (k + 1) % m;
                double r1 = circumradius(hull[i], hull[j],
                    hull[k]);
                double r2 = circumradius(hull[i], hull[j],
                    hull[k2]);
                if (r2 > r1) k = k2;
                else break;
            }
            ans = max(ans, circumradius(hull[i], hull[j],
                hull[k]));
            j = (j + 1) % m;
        } while (j != i);
    }
    cout << fixed << setprecision(6) << ans << endl;
}
```

最大曼哈頓距離.cpp

```
int maxManhattanDist(const vector<Point>& pts) {
    int res = 0;
    for (int d = 0; d < 4; ++d) {
        int mn = 1e9, mx = -1e9;
        for (auto p : pts) {
            int val;
            if (d == 0) val = p.x + p.y;
            if (d == 1) val = p.x - p.y;
            if (d == 2) val = -p.x + p.y;
            if (d == 3) val = -p.x - p.y;
            mn = min(mn, val);
            mx = max(mx, val);
        }
        res = max(res, mx - mn);
    }
    return res;
}
```

極角排序.cpp

```
struct Point {
    int x, y;
};

// 以 base 為基準點，對 points 進行極角排序
void polarSort(vector<Point>& points, Point base) {
    sort(points.begin(), points.end(), [&](const Point& a,
        const Point& b) {
        int dx1 = a.x - base.x, dy1 = a.y - base.y;
        int dx2 = b.x - base.x, dy2 = b.y - base.y;
        long long cross = 1LL * dx1 * dy2 - 1LL * dy1 * dx2
            ;
```

```
        if (cross != 0) return cross > 0;
        // 極角相同時，距離近的排前面
        return dx1 * dx1 + dy1 * dy1 < dx2 * dx2 + dy2 * dy2;
    });
}
```

離散化.cpp

```
vector<int> discretize(vector<int> vals) {
    vector<int> tmp = vals;
    sort(tmp.begin(), tmp.end());
    tmp.erase(unique(tmp.begin(), tmp.end()), tmp.end());
    for (int& v : vals)
        v = lower_bound(tmp.begin(), tmp.end(), v) - tmp.begin();
    return vals;
}
```

5 Data Structure

Array Version of Linked List.cpp

```
/*看不懂cut the node
實際用法需要理解      */
struct node{
    struct node *nxt, *prv;
    long long v;
}pos[100002];

inline node* cut(node *p) // cut the node
{
    p->nxt->prv = p->prv;
    p->prv->nxt = p->nxt;
    p->nxt = p->prv = NULL;
    return p;
}

inline node* insert(node *p, node *q) // insert node q to right side of node p
{
    q->nxt = p->nxt;
    p->nxt->prv = q;
    q->prv = p;
    p->nxt = q;
    return q;
}
```

Binary Index Tree.cpp

```
//BIT is always 1 index
ll n,t[mxN],a[mxN];
void built(){forn(i,1,mxN){ t[i]+=a[i]; if(i+(i&-i)<mxN) t[i+(i&-i)]+=t[i];}} //O(n)
void pad(ll p,ll val){for(;p<mxN;t[p]+=val,p+=p&-p);}
ll query(ll p){ll ans=0; for(;p;ans+=t[p],p-=p&-p); return ans;}
```

Disjoint Set Union.cpp

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

class DisjointSets {
  private:
  vector<int> parents;
  vector<int> sizes;

  public:
  DisjointSets(int size) : parents(size), sizes(size, 1) {
    for (int i = 0; i < size; i++) { parents[i] = i; }
  }

  /** @return the "representative" node in x's component */
  int find(int x) {
    return parents[x] == x ? x : (parents[x] = find(parents[x]));
  }

  /** @return whether the merge changed connectivity */
  bool unite(int x, int y) {
    int x_root = find(x);
    int y_root = find(y);
    if (x_root == y_root) { return false; }

    if (sizes[x_root] < sizes[y_root]) { swap(x_root, y_root); }
    sizes[x_root] += sizes[y_root];
    parents[y_root] = x_root;
    return true;
  }

  /** @return whether x and y are in the same connected component */
  bool connected(int x, int y) { return find(x) == find(y); }
};
```

Segment Tree.cpp

```
//range add range query // sum query
#define forr(a,b,c) for(ll a=b;a>=c;--a)
#define forn(a,b,c) for(ll a=b;a<c;++a)
constexpr ll mxN=2e5+1;
#pragma GCC target("lzcnt")
ll n,q,t[mxN<<1],lz[mxN]{};
void built(){forr(i,n-1,1)t[i]=t[i<<1]+t[i<<1|1];}
void apply(ll p,ll k,ll val){t[p]+=val*k; if(p<n) lz[p]+=val;}
void push(ll p){for(ll h=63-__builtin_clzll(p);h;h--) if(lz[p>>h]){apply((p>>h)<<1,1<<(h-1),lz[p>>h]),apply((p>>h)<<1|1,1<<(h-1),lz[p>>h]),lz[p>>h]=0;}}
void built(ll p){for(;p>>=1;)if(!lz[p]) t[p]=t[p<<1]+t[p<<1|1];}
void rad(ll l,ll r,ll val){
    ll l0=l+=n-1,r0=r+=n-1,k=1; push(l0),push(r0);
    for(;l<=r;l>>=1,r>>=1,k<<=1){if(l&1) apply(l++,k,val);
        if(r&1^1) apply(r--,k,val);}
    built(l0),built(r0);
}
ll query(ll l,ll r){ ll ans=0; push(l+=n-1),push(r+=n-1);
    for(;l<=r;l>>=1,r>>=1){if(l&1) ans+=t[l++]; if(r&1^1) ans+=t[r--];} return ans;} //range query
ll query(ll p){push(p+=n-1); return t[p];} // point query
```

6 Not that important

EPS.cpp

```
#include <cmath>
const double EPS = 1e-9;

bool eq(double a, double b) {
    return fabs(a - b) < EPS;
}

bool lt(double a, double b) {
    return a < b - EPS;
}

bool le(double a, double b) {
    return a < b + EPS;
}
```

Gauce Elimination.cpp

```
// a[N][N]是增广矩阵
int gauss()
{
    int c, r;
    for (c = 0, r = 0; c < n; c ++ )
    {
        int t = r;
        for (int i = r; i < n; i ++ )   // 找到绝对值最大的行
            if (fabs(a[i][c]) > fabs(a[t][c]))
                t = i;

        if (fabs(a[t][c]) < eps) continue;

        for (int i = c; i <= n; i ++ ) swap(a[t][i], a[r][i]);      // 将绝对值最大的行换到最顶端
        for (int i = n; i >= c; i -- ) a[r][i] /= a[r][c];
                // 将当前行的首位变成1
        for (int i = r + 1; i < n; i ++ )       // 用当前行将下面所有的列消成0
            if (fabs(a[i][c]) > eps)
```

```
                for (int j = n; j >= c; j -- )
                    a[i][j] -= a[r][j] * a[i][c];

        r ++ ;
    }

    if (r < n)
    {
        for (int i = r; i < n; i ++ )
            if (fabs(a[i][n]) > eps)
                return 2; // 无解
        return 1; // 有无穷多组解
    }

    for (int i = n - 1; i >= 0; i -- )
        for (int j = i + 1; j < n; j ++ )
            a[i][n] -= a[i][j] * a[j][n];

    return 0; // 有唯一解
}
```

Joseph.cpp

```
int josephus(int n, int k) {
  int res = 0;
  for (int i = 1; i <= n; ++i) res = (res + k) % i;
  return res;
}
//klogn version
int josephus(int n, int k) {
  if (n == 1) return 0;
  if (k == 1) return n - 1;
  if (k > n) return (josephus(n - 1, k) + k) % n;  // 线性算法
  int res = josephus(n - n / k, k);
  res -= n % k;
  if (res < 0)
    res += n;  // mod n
  else
    res += res / (k - 1);  // 还原位置
  return res;
}
```