Contents

1 Contest Setup

1.1 vimrc

```
set number      " Show line numbers
set mouse=a     " Enable inaction via mouse
set showmatch       " Highlight matching brace
set cursorline      " Show underline
set cursorcolumn    " highlight vertical column

filetype on "enable file detection
syntax on   "syntax highlight

set autoindent      " Auto-indent new lines
set shiftwidth=4    " Number of auto-indent spaces
set smartindent     " Enable smart-indent
set smarttab        " Enable smart-tabs
set softtabstop=4   " Number of spaces per Tab

" ---------Optional----------

set undolevels=10000    " Number of undo levels
set scrolloff=5     " Auto scroll

set hlsearch    " Highlight all search results
set smartcase   " Enable smart-case search
set ignorecase  " Always case-insensitive
set incsearch   " Searches for strings incrementally

highlight Comment ctermfg=cyan
set showmode

set encoding=utf-8
set fileencoding=utf-8
scriptencoding=utf-8
```

1.2 bashrc

```
alias g++="g++ -Wall -Wextra -std=c++11 -O2"
```

1.3 C++ template

```
#include <bits/stdc++.h>

using namespace std;

#define x first
#define y second

typedef long long int ll;
typedef pair<int, int> ii;

int main()
{
    return 0;
}
```

1.4 Java template

```
import java.io.*;
import java.util.*;

public class Main
{
    public static void main(String[] args)
    {
        MyScanner sc = new MyScanner();
        out = new PrintWriter(new BufferedOutputStream(System.out));
        // Start writing your solution here.

        // Stop writing your solution here.
        out.close();
    }

    public static PrintWriter out;

    public static class MyScanner
    {
        BufferedReader br;
        StringTokenizer st;

        public MyScanner()
        {
            br = new BufferedReader(new InputStreamReader(System.in));
        }

        boolean hasNext()
        {
            while (st == null || !st.hasMoreElements()) {
                try {
                    st = new StringTokenizer(br.readLine());
                } catch (Exception e) {
                    return false;
                }
            }
            return true;
        }

        String next()
        {
            if (hasNext())
                return st.nextToken();
            return null;
        }

        int nextInt()
        {
            return Integer.parseInt(next());
        }
```

```java

        long nextLong()
        {
            return Long.parseLong(next());
        }

        double nextDouble()
        {
            return Double.parseDouble(next());
        }

        String nextLine()
        {
            String str = "";
            try {
                str = br.readLine();
            } catch (IOException e) {
                e.printStackTrace();
            }
            return str;
        }
    }
}
```

1.4.1 Java Issues

1. Random Shuffle before sorting: $Random\ rnd = new\ Random();\ rnd.nextInt();$
2. Use StringBuilder for large output

2 Reminder

1. 隊友的建議，要認真聽！通常隊友的建議都會突破你盲點
2. Read the problem statements carefully. Input and output specifications and constraints are crucial!
3. Estimate the **time complexity** and **memory complexity** carefully.
4. Time penalty is 20 minutes per WA, **don't rush**!
5. Sample test cases must all be tested and passed before every submission!
6. Test the corner cases, such as 0, 1, -1. Test all edge cases of the input specification.
7. Bus error: the code has $scanf, fgets$ but have nothing to read! Check if you have early termination but didn't handle it properly.
8. Binary search? 數學算式移項合併後查詢?
9. Two Pointer <–> Binary Search
10. Directed graph connectivity -> DFS. Undirected graph -> Union Find
11. Check connectivity of the graph if the problem statement doesn't say anything

3 Useful code

3.1 Grep Error and Warnings

```
g++ main.cpp 2>&1 | grep -E 'warning|error'
```

3.2 Leap year

```
year % 400 == 0 || (year % 4 == 0 && year % 100 != 0)
```

3.3 Fast Exponentiation $O(log(exp))$

```cpp
ll fast_pow(ll base, ll exp, ll mod)
{
    if (exp == 0)
        return 1LL;
    ll res = 1;
    while (exp > 0) {
        if (exp & 1) {
            res = ((res % mod) * (base % mod)) % mod;
        }
        exp >>= 1;
        base = (base * base) % mod;
    }
    return res;
}
```

3.4 GCD $O(log(a+b))$

注意負數的 case!

```cpp
ll gcd(ll a, ll b)
{
    return b == 0 ? a : gcd(b, a % b);
}
```

3.5 Extended Euclidean Algorithm

Bezout identity $ax + by = gcd(a, b)$, where $gcd(a, b)$ is the smallest positive integer that can be written as ax + by, and every integer of the form ax + by is a multiple of $gcd(a, b)$.

```cpp
ll ext_gcd(ll a, ll b, ll &x, ll &y)
{
    if (a == 0) {
        x = 0;
        y = 1;
        return b;
    }

    ll x1, y1;
    ll gcd = ext_gcd(b % a, a, x1, y1);

    x = y1 - (b / a) * x1;
    y = x1;

    return gcd;
}
```

3.6 Mod Inverse

Case 1 $gcd(a, m) = 1$: ax + my = gcd(a, m) = 1 (use ext_gcd)

Case 2 $m\ is\ prime$: $a^{m-2} \equiv a^{-1} mod\ m$ (use Fermat's little theorem)

3.7 Prime Generator

```
bool is_prime[N];
vector<ll> primes;
void init()
{
    fill(is_prime, is_prime + N, true);
    for (int i = 2; i < N; i++) {
        if (is_prime[i] == true) {
            primes.push_back(i);
            for (int j = i * i; j < N; j += i)
                is_prime[j] = false;
        }
    }
}
```

3.8 Binomial Coefficient

```
int binomialCoeff(int n, int k)
{
    int res = 1;

    if ( k > n - k ) // Since C(n, k) = C(n, n-k)
        k = n - k;

    for (int i = 0; i < k; ++i) // n...n-k / 1...k
    {
        res *= (n - i);
        res /= (i + 1);
    }

    return res;
}
```

3.9 STL quick reference

3.9.1 Map

```
map<T1, T2> m; // iterable
void clear();
void erase(T1 key);
it find(T1 key); // <key, val>
void insert(pair<T1, T2> P);
T2 &[](T1 key); // if key not in map, new key will be inserted with default val
it lower_bound(T1 key); // = m.end() if not found, *it = <key, val>
it upper_bound(T1 key); // = m.end() if not found, *it = <key, val>
```

3.9.2 Set

```
set<T> s; // iterable
void clear();
size_t count(T val); // number of val in set
void erase(T val);
it find(T val); // = s.end() if not found
void insert(T val);
it lower_bound(T val); // = s.end() if not found, *it = <key, val>
it upper_bound(T val); // = s.end() if not found, *it = <key, val>
```

3.9.3 Algorithm

```
// return if i is smaller than j
comp = [&](const T &i, const T &j) -> bool;
vector<T> v;
bool any_of(v.begin(), v.end(), [&](const T &i) -> bool);
bool all_of(v.begin(), v.end(), [&](const T &i) -> bool);
void copy(inp.begin(), in.end(), out.begin());
int count(v.begin(), v.end(), int val); // number of val in v
it unique(v.begin(), v.end());           // it - v.begin() = size
// after calling, v[nth] will be n-th smallest elem in v
void nth_element(v.begin(), nth_it, bin_comp);
void merge(in1.begin(), in1.end(), in2.begin(), in2.end(), out.begin(), comp);
// include union, intersection, difference, symmetric_difference(xor)
void set_union(in1.begin(), in1.end(), in2.begin(), in2.end(), out.begin(), comp);
bool next_permutation(v.begin(), v.end());
// v1, v2 need sorted already, whether v1 includes v2
bool inclues(v1.begin(), v1.end(), v2.begin(), v2.end());
it find(v.begin(), v.end(), T val); // = v.end() if not found
it search(v1.begin(), v1.end(), v2.begin(), v2.end());
it lower_bound(v.begin(), v.end(), T val);
it upper_bound(v.begin(), v.end(), T val);
bool binary_search(v.begin(), v.end(), T val); // exist in v ?
void sort(v.begin(), v.end(), comp);
void stable_sort(v.begin(), v.end(), comp);
```

3.9.4 String

3.9.5 Priority Queue

```
bool cmp(ii a, ii b)
{
    if(a.first == b.first)
    return a.second > b.second;
    return b.first > a.first;
}

priority_queue< ii, vector<ii>, function<bool(ii, ii)> > pq(cmp);
```

4 Search

4.1 Binary Search

4.1.1 Find key

4.1.2 Upper / lower Bound

4.2 Ternary Search

4.3 折半完全列舉

4.4 Two-pointer 爬行法

5 Basic data structure

5.1 1D BIT

```
// BIT is 1-based
const int MAX_N = 20000; //這個記得改!
ll bit[MAX_N + 1];

int sum(int i) {
    int s = 0;
    while (i > 0) {
        s += bit[i];
        i -= (i & -i);
    }
    return s;
}

void add(int i, int x) {
    while (i <= MAX_N) {
        bit[i] += x;
        i += (i & -i);
    }
}
```

5.2 2D BIT

```
// BIT is 1-based
const int MAX_N = 20000, MAX_M = 20000; //這個記得改!
ll bit[MAX_N + 1][MAX_M + 1];

ll sum(int a, int b) {
    ll s = 0;
    for (int i = a; i > 0; i -= (i & -i))
        for (int j = b; j > 0; j -= (j & -j))
            s += bit[i][j];
    return s;
}

void add(int a, int b, ll x) {
    // MAX_N, MAX_M 須適時調整!
    for (int i = a; i <= MAX_N; i += (i & -i))
        for (int j = b; j <= MAX_M; j += (j & -j))
            bit[i][j] += x;
}
```

5.3 Union Find

```
#define N 20000 // 記得改
struct UFDS {
    int par[N];

    void init() {
        memset(par, -1, sizeof(par));
    }

    int root(int x) {
        return par[x] < 0 ? x : par[x] = root(par[x]);
    }

    void merge(int x, int y) {
        x = root(x);
        y = root(y);

        if (x != y) {
            if (par[x] > par[y])
                swap(x, y);
            par[x] += par[y];
            par[y] = x;
        }
    }
}
```

5.4 Segment Tree

5.5 Sparse Table

```
struct {
    int sp[MAX_LOG_N][MAX_N]; // MAX_LOG_N = ceil(lg(MAX_N))

    void build(int inp[], int n) {
        for (int j = 0; j < n; j++) {
            sp[0][j] = inp[j];
        }

        for (int i = 1; (1 << i) <= n; i++)
            for (int j = 0; j + (1 << i) <= n; j++)
                sp[i][j] =
                    min(sp[i - 1][j], sp[i - 1][j + (1 << (i - 1))]);
    }

    int query(int l, int r) { // [l, r)
        int k = floor(log2(r - l));

        return min(sp[k][l], sp[k][r - (1 << k)]);
    }
} sptb;
```

6 Dynamic Programming

7 Tree

7.1 LCA

8 Graph

8.1 Articulation point / edge

8.2 CC

8.2.1 BCC vertex

8.2.2 BCC edge

8.2.3 SCC

8.3 Shortest Path

8.3.1 Dijkatra

8.3.2 Dijkatra (next-to-shortest path)

8.3.3 SPFA

8.3.4 Bellman-Ford

8.3.5 Floyd-Warshall

8.4 Kruskal MST

8.5 Flow

8.5.1 Max Flow (Dinic)

8.5.2 Min-Cut

8.5.3 Min Cost Max Flow

8.5.4 Maximum Bipartite Graph

9 String

9.1 KMP

```
void fail()
{
    int len = strlen(pat);

    f[0] = 0;
    int j = 0;
    for (int i = 1; i < len; i++) {
        while (j != 0 && pat[i] != pat[j])
            j = f[j - 1];

        if (pat[i] == pat[j])
            j++;

        f[i] = j;
    }
}

int match()
{
    int res = 0;
    int j = 0, plen = strlen(pat), tlen = strlen(text);

    for (int i = 0; i < tlen; i++) {
        while (j != 0 && text[i] != pat[j])
            j = f[j - 1];

        if (text[i] == pat[j]) {
            if (j == plen - 1) { // find match
                res++;
                j = f[j];
            } else {
                j++;
            }
        }
    }

    return res;
}
```

9.2 Z Algorithm

9.3 Trie

```
#define N 600010
struct node {
    int child[26];
    bool ending;
} trie[N];

/*
root is 0
memset(trie, 0, sizeof(trie));
freeNode = 1;
*/
int freeNode;
void insert(string &str, int pos, int node)
{
    if (pos == (int)str.length()) {
        trie[node].ending = true;
    } else { // find which way to go
        int c = str[pos] - 'a';
        if (trie[node].child[c] == 0) // give a new node
            trie[node].child[c] = freeNode++;
        insert(str, pos + 1, trie[node].child[c]);
    }
}
```

9.4 Suffix Array

10 Geometry

10.1 Template

```cpp
// C++ routines for computational geometry.

#include <cassert>
#include <cmath>
#include <iostream>
#include <vector>

using namespace std;

double INF = 1e100;
double EPS = 1e-12;

struct PT {
    double x, y;
    PT() {}
    PT(double x, double 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*(double c) const
    {
        return PT(x * c, y * c);
    }
    PT operator/(double c) const
    {
        return PT(x / c, y / c);
    }
};

double dot(PT p, PT q)
{
    return p.x * q.x + p.y * q.y;
}
double dist2(PT p, PT q)
{
    return dot(p - q, p - q);
}
double 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, double t)
{
    return PT(p.x * cos(t) - p.y * sin(t), p.x * sin(t) + p.y * cos(t));
}

// project point c onto line through a and b
// assuming a != b
PT ProjectPointLine(PT a, PT b, PT 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
PT ProjectPointSegment(PT a, PT b, PT 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(PT a, PT b, PT 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(PT a, PT b, PT c, PT d)
{
    return fabs(cross(b - a, c - d)) < EPS;
}

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

// determine if line segment from a to b intersects with
// line segment from c to d
bool SegmentsIntersect(PT a, PT b, PT c, PT d)
{
    if (LinesCollinear(a, b, c, d)) {
        if (dist2(a, c) < EPS || dist2(a, d) < EPS || dist2(b, c) < EPS ||
            dist2(b, d) < EPS)
            return true;
        if (dot(c - a, c - b) > 0 && dot(d - a, d - b) > 0 && dot(c - b, d - b) > 0)
            return false;
        return true;
    }
    if (cross(d - a, b - a) * cross(c - a, b - a) > 0)
        return false;
    if (cross(a - c, d - c) * cross(b - c, d - c) > 0)
        return false;
    return true;
}

// compute intersection of line passing through a and b
// with line passing through c and d, assuming that unique
// intersection exists; for segment intersection, check if
// segments intersect first
PT ComputeLineIntersection(PT a, PT b, PT c, PT d)
{
    b = b - a;
    d = c - d;
    c = c - a;
    assert(dot(b, b) > EPS && dot(d, d) > EPS);
    return a + b * cross(c, d) / cross(b, d);
}

// compute center of circle given three points
PT ComputeCircleCenter(PT a, PT b, PT c)
{
    b = (a + b) / 2;
    c = (a + c) / 2;
    return ComputeLineIntersection(b, b + RotateCW90(a - b), c,
                                   c + RotateCW90(a - c));
}

// determine if point is in a possibly non-convex polygon (by William
// Randolph Franklin); returns 1 for strictly interior points, 0 for
// strictly exterior points, and 0 or 1 for the remaining points.
// Note that it is possible to convert this into an *exact* test using
// integer arithmetic by taking care of the division appropriately
// (making sure to deal with signs properly) and then by writing exact
// tests for checking point on polygon boundary
bool PointInPolygon(const vector<PT> &p, PT q)
{
    bool c = 0;
    for (int i = 0; i < p.size(); i++) {
        int j = (i + 1) % p.size();
        if ((p[i].y <= q.y && q.y < p[j].y || p[j].y <= q.y && q.y < p[i].y) &&
            q.x < p[i].x + (p[j].x - p[i].x) * (q.y - p[i].y) / (p[j].y - p[i].y))
            c = !c;
    }
    return c;
}

// determine if point is on the boundary of a polygon
bool PointOnPolygon(const vector<PT> &p, PT q)
{
    for (int i = 0; i < p.size(); i++)
        if (dist2(ProjectPointSegment(p[i], p[(i + 1) % p.size()], q), q) < EPS)
            return true;
    return false;
}

// compute intersection of line through points a and b with
// circle centered at c with radius r > 0
vector<PT> CircleLineIntersection(PT a, PT b, PT c, double r)
{
    vector<PT> ret;
    b = b - a;
    a = a - c;
    double A = dot(b, b);
    double B = dot(a, b);
    double C = dot(a, a) - r * r;
    double D = B * B - A * C;
    if (D < -EPS)
        return ret;
    ret.push_back(c + a + b * (-B + sqrt(D + EPS)) / A);
    if (D > EPS)
        ret.push_back(c + a + b * (-B - sqrt(D)) / A);
    return ret;
}

// compute intersection of circle centered at a with radius r
// with circle centered at b with radius R
vector<PT> CircleCircleIntersection(PT a, PT b, double r, double R)
{
    vector<PT> ret;
    double d = sqrt(dist2(a, b));
    if (d > r + R || d + min(r, R) < max(r, R))
        return ret;
    double x = (d * d - R * R + r * r) / (2 * d);
    double y = sqrt(r * r - x * x);
    PT v = (b - a) / d;
    ret.push_back(a + v * x + RotateCCW90(v) * y);
    if (y > 0)
        ret.push_back(a + v * x - RotateCCW90(v) * y);
    return ret;
}

// 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".
double ComputeSignedArea(const vector<PT> &p)
{
    double 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;
}

double ComputeArea(const vector<PT> &p)
{
    return fabs(ComputeSignedArea(p));
}

PT ComputeCentroid(const vector<PT> &p)
{
    PT c(0, 0);
    double 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;
}

// tests whether or not a given polygon (in CW or CCW order) is simple
bool IsSimple(const vector<PT> &p)
{
    for (int i = 0; i < p.size(); i++) {
        for (int k = i + 1; k < p.size(); k++) {
            int j = (i + 1) % p.size();
            int l = (k + 1) % p.size();
            if (i == l || j == k)
                continue;
            if (SegmentsIntersect(p[i], p[j], p[k], p[l]))
                return false;
        }
    }
    return true;
}

int main()
{

    // expected: (-5,2)
    cerr << RotateCCW90(PT(2, 5)) << endl;

    // expected: (5,-2)
    cerr << RotateCW90(PT(2, 5)) << endl;

    // expected: (-5,2)
    cerr << RotateCCW(PT(2, 5), M_PI / 2) << endl;

    // expected: (5,2)
    cerr << ProjectPointLine(PT(-5, -2), PT(10, 4), PT(3, 7)) << endl;

    // expected: (5,2) (7.5,3) (2.5,1)
    cerr << ProjectPointSegment(PT(-5, -2), PT(10, 4), PT(3, 7)) << " "
         << ProjectPointSegment(PT(7.5, 3), PT(10, 4), PT(3, 7)) << " "
         << ProjectPointSegment(PT(-5, -2), PT(2.5, 1), PT(3, 7)) <<
    endl;

    // expected: 6.78903
    cerr << DistancePointPlane(4, -4, 3, 2, -2, 5, -8) << endl;

    // expected: 1 0 1
    cerr << LinesParallel(PT(1, 1), PT(3, 5), PT(2, 1), PT(4, 5)) << "
    "
         << LinesParallel(PT(1, 1), PT(3, 5), PT(2, 0), PT(4, 5)) << "
    "
         << LinesParallel(PT(1, 1), PT(3, 5), PT(5, 9), PT(7, 13)) <<
    endl;

    // expected: 0 0 1
    cerr << LinesCollinear(PT(1, 1), PT(3, 5), PT(2, 1), PT(4, 5)) << "
     "
         << LinesCollinear(PT(1, 1), PT(3, 5), PT(2, 0), PT(4, 5)) << "
     "
         << LinesCollinear(PT(1, 1), PT(3, 5), PT(5, 9), PT(7, 13)) <<
    endl;

    // expected: 1 1 1 0
    cerr << SegmentsIntersect(PT(0, 0), PT(2, 4), PT(3, 1), PT(-1, 3))
    << " "
         << SegmentsIntersect(PT(0, 0), PT(2, 4), PT(4, 3), PT(0, 5))
    << " "
         << SegmentsIntersect(PT(0, 0), PT(2, 4), PT(2, -1), PT(-2, 1))
     << " "
         << SegmentsIntersect(PT(0, 0), PT(2, 4), PT(5, 5), PT(1, 7))
    << endl;

    // expected: (1,2)
    cerr << ComputeLineIntersection(PT(0, 0), PT(2, 4), PT(3, 1), PT
    (-1, 3))
         << endl;

    // expected: (1,1)
    cerr << ComputeCircleCenter(PT(-3, 4), PT(6, 1), PT(4, 5)) << endl;

    vector<PT> v;
    v.push_back(PT(0, 0));
    v.push_back(PT(5, 0));
    v.push_back(PT(5, 5));
    v.push_back(PT(0, 5));

    // expected: 1 1 1 0 0
    cerr << PointInPolygon(v, PT(2, 2)) << " " << PointInPolygon(v, PT(2, 0))
         << " " << PointInPolygon(v, PT(0, 2)) << " "
         << PointInPolygon(v, PT(5, 2)) << " " << PointInPolygon(v, PT(2, 5))
         << endl;

    // expected: 0 1 1 1 1
    cerr << PointOnPolygon(v, PT(2, 2)) << " " << PointOnPolygon(v, PT(2, 0))
         << " " << PointOnPolygon(v, PT(0, 2)) << " "
         << PointOnPolygon(v, PT(5, 2)) << " " << PointOnPolygon(v, PT(2, 5))
         << endl;

    // expected: (1,6)
    //           (5,4) (4,5)
    //           blank line
    //           (4,5) (5,4)
    //           blank line
    //           (4,5) (5,4)
    vector<PT> u = CircleLineIntersection(PT(0, 6), PT(2, 6), PT(1, 1), 5);
    for (int i = 0; i < u.size(); i++)
        cerr << u[i] << " ";
    cerr << endl;
    u = CircleLineIntersection(PT(0, 9), PT(9, 0), PT(1, 1), 5);
    for (int i = 0; i < u.size(); i++)
        cerr << u[i] << " ";
    cerr << endl;
    u = CircleCircleIntersection(PT(1, 1), PT(10, 10), 5, 5);
    for (int i = 0; i < u.size(); i++)
        cerr << u[i] << " ";
    cerr << endl;
    u = CircleCircleIntersection(PT(1, 1), PT(8, 8), 5, 5);
    for (int i = 0; i < u.size(); i++)
        cerr << u[i] << " ";
    cerr << endl;
    u = CircleCircleIntersection(PT(1, 1), PT(4.5, 4.5), 10, sqrt(2.0) / 2.0);
    for (int i = 0; i < u.size(); i++)
        cerr << u[i] << " ";
    cerr << endl;
    u = CircleCircleIntersection(PT(1, 1), PT(4.5, 4.5), 5, sqrt(2.0) / 2.0);
    for (int i = 0; i < u.size(); i++)
        cerr << u[i] << " ";
    cerr << endl;

    // area should be 5.0
    // centroid should be (1.1666666, 1.166666)
    PT pa[] = {PT(0, 0), PT(5, 0), PT(1, 1), PT(0, 5)};
    vector<PT> p(pa, pa + 4);
    PT c = ComputeCentroid(p);
    cerr << "Area: " << ComputeArea(p) << endl;
    cerr << "Centroid: " << c << endl;

    return 0;
}
```

========

```
#define x first
#define y second
typedef pair <double , double > pt;
struct line {
double a, b, c;
// coefficients in general form, compare up to constant factor
}
pt operator-(pt u, pt v) { return pt(u.x-v.x, u.y-v.y); }
pt operator+(pt u, pt v) { return pt(u.x+v.x, u.y+v.y); }
pt operator*(pt u, double d) { return pt(u.x*d, u.y*d); }
double operator*(pt u, pt v) { return u.x*v.x + u.y*v.y; } // dot product double operator!(pt p) { return sqrt(p*p); } // norm
double operator^(pt u, pt v) { return u.x*v.y - u.y*v.x; } // cross product
```

10.1.1 Point / Line

10.1.2 Intersection

10.2 Half-plane intersection

10.3 Convex Hull