

Contents

1	Contest Setup	2
1.1	vimrc	2
1.2	bashrc	2
1.3	C++ template	2
1.4	Java template	3
1.4.1	Java Issues	3
2	Reminder	3
3	Useful code	3
3.1	Leap year	3
3.2	Fast Exponentiation $O(\log(exp))$	3
3.3	GCD $O(\log(a + b))$	3
3.4	Extended Euclidean Algorithm	3
3.5	Mod Inverse	3
3.6	Prime Generator	4
3.7	Binomial Coefficient	4
3.8	STL quick reference	4
3.8.1	Map	4
3.8.2	Set	4
3.8.3	Algorithm	4
3.8.4	String	4
3.8.5	Priority Queue	4
4	Search	5
4.1	Binary Search	5
4.1.1	Find key	5
4.1.2	Upper / lower Bound	5
4.2	Ternary Search	5
4.3	折半完全列舉	5
4.4	Two-pointer 爬行法	5
5	Basic data structure	5
5.1	1D BIT	5
5.2	2D BIT	5
5.3	Union Find	5
5.4	Segment Tree	5
5.5	Sparse Table	5
6	Dynamic Programming	6
7	Tree	6
7.1	LCA	6
8	Graph	6
8.1	Articulation point / edge	6
8.2	CC	6
8.2.1	BCC vertex	6
8.2.2	BCC edge	6
8.2.3	SCC	6
8.3	Shortest Path	6
8.3.1	Dijkstra	6
8.3.2	Dijkstra (next-to-shortest path)	6
8.3.3	SPFA	6
8.3.4	Bellman-Ford	6
8.3.5	Floyd-Warshall	6
8.4	Kruskal MST	6
8.5	Flow	6
8.5.1	Max Flow (Dinic)	6
8.5.2	Min-Cut	6
8.5.3	Min Cost Max Flow	6
8.5.4	Maximum Bipartite Graph	6
9	String	6
9.1	KMP	6
9.2	Z Algorithm	6
9.3	Trie	6
9.4	Suffix Array	6
10	Geometry	6
10.1	Template	6
10.1.1	Point / Line	10

10.1.2	Intersection	10
10.2	Half-plane intersection	10
10.3	Convex Hull	10

1 Contest Setup

1.1 vimrc

```

1 set number      " Show line numbers
2 set mouse=a     " Enable inaction via mouse
3 set showmatch   " Highlight matching brace
4 set cursorline  " Show underline
5 set cursorcolumn " highlight vertical column
6
7 filetype on "enable file detection
8 syntax on   "syntax highlight
9
10 set autoindent    " Auto-indent new lines
11 set shiftwidth=4  " Number of auto-indent spaces
12 set smartindent  " Enable smart-indent
13 set smarttab     " Enable smart-tabs
14 set softtabstop=4 " Number of spaces per Tab
15
16 " -----Optional-----
17
18 set undolevels=10000 " Number of undo levels
19 set scrolloff=5     " Auto scroll
20
21 set hlsearch      " Highlight all search results
22 set smartcase     " Enable smart-case search
23 set ignorecase   " Always case-insensitive
24 set incsearch     " Searches for strings incrementally
25
26 highlight Comment ctermfg=cyan
27 set showmode
28
29 set encoding=utf-8
30 set fileencoding=utf-8
31 set scriptencoding=utf-8

```

1.2 bashrc

```

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

```

1.3 C++ template

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 #define x first
6 #define y second
7
8 typedef long long int ll;
9 typedef pair<int, int> ii;
10
11 int main()
12 {

```

```

13     return 0;
14 }

```

1.4 Java template

```

1 import java.io.*;
2 import java.util.*;
3
4 public class Main
5 {
6     public static void main(String[] args)
7     {
8         MyScanner sc = new MyScanner();
9         out = new PrintWriter(new BufferedOutputStream(System.out));
10        // Start writing your solution here.
11
12        // Stop writing your solution here.
13        out.close();
14    }
15
16    public static PrintWriter out;
17
18    public static class MyScanner
19    {
20        BufferedReader br;
21        StringTokenizer st;
22
23        public MyScanner()
24        {
25            br = new BufferedReader(new InputStreamReader(System.in));
26        }
27
28        boolean hasNext()
29        {
30            while (st == null || !st.hasMoreElements()) {
31                try {
32                    st = new StringTokenizer(br.readLine());
33                } catch (Exception e) {
34                    return false;
35                }
36            }
37            return true;
38        }
39
40        String next()
41        {
42            if (hasNext())
43                return st.nextToken();
44            return null;
45        }
46
47        int nextInt()
48        {
49            return Integer.parseInt(next());
50        }

```

```

51     long nextLong()
52     {
53         return Long.parseLong(next());
54     }
55
56     double nextDouble()
57     {
58         return Double.parseDouble(next());
59     }
60
61     String nextLine()
62     {
63         String str = "";
64         try {
65             str = br.readLine();
66         } catch (IOException e) {
67             e.printStackTrace();
68         }
69         return str;
70     }
71 }
72
73 }

```

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 \leftrightarrow Binary Search
10. Directed graph connectivity \rightarrow DFS. Undirected graph \rightarrow Union Find

3 Useful code

3.1 Leap year

```

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

```

3.2 Fast Exponentiation $O(\log(\exp))$

```

1 || ll fast_pow(ll base, ll exp, ll mod)
2 || {
3 ||     if (exp == 0)

```

```

4         return 1LL;
5         ll res = 1;
6         while (exp > 0) {
7             if (exp & 1) {
8                 res = ((res % mod) * (base % mod)) % mod;
9             }
10            exp >>= 1;
11            base = (base * base) % mod;
12        }
13        return res;
14    }

```

3.3 GCD $O(\log(a + b))$

注意負數的 case!

```

1 || ll gcd(ll a, ll b)
2 || {
3 ||     return b == 0 ? a : gcd(b, a % b);
4 || }

```

3.4 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 $\gcd(a, b)$.

```

1 || ll ext_gcd(ll a, ll b, ll &x, ll &y)
2 || {
3 ||     if (a == 0) {
4 ||         x = 0;
5 ||         y = 1;
6 ||         return b;
7 ||     }
8 ||
9 ||     ll x1, y1;
10 ||    ll gcd = ext_gcd(b % a, a, x1, y1);
11 ||
12 ||    x = y1 - (b / a) * x1;
13 ||    y = x1;
14 ||
15 ||    return gcd;
16 || }

```

3.5 Mod Inverse

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

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

3.6 Prime Generator

```

1 || bool is_prime[N];
2 || vector<ll> primes;
3 || void init()
4 || {
5 ||     fill(is_prime, is_prime + N, true);

```

```

6   for (int i = 2; i < N; i++) {
7       if (is_prime[i] == true) {
8           primes.push_back(i);
9           for (int j = i * i; j < N; j += i)
10              is_prime[j] = false;
11       }
12   }
13 }

```

3.7 Binomial Coefficient

```

1  int binomialCoeff(int n, int k)
2  {
3      int res = 1;
4
5      if ( k > n - k ) // Since C(n, k) = C(n, n-k)
6          k = n - k;
7
8      for (int i = 0; i < k; ++i) // n...n-k / 1...k
9      {
10         res *= (n - i);
11         res /= (i + 1);
12     }
13
14     return res;
15 }

```

3.8 STL quick reference

3.8.1 Map

```

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

```

3.8.2 Set

```

1  set<T> s; // iterable
2  void clear();
3  size_t count(T val); // number of val in set
4  void erase(T val);
5  it find(T val); // = s.end() if not found
6  void insert(T val);
7  it lower_bound(T val); // = s.end() if not found, *it = <key, val>
8  it upper_bound(T val); // = s.end() if not found, *it = <key, val>

```

3.8.3 Algorithm

```

1  // return if i is smaller than j
2  comp = [&](const T &i, const T &j) -> bool;
3  vector<T> v;
4  bool any_of(v.begin(), v.end(), [&](const T &i) -> bool);
5  bool all_of(v.begin(), v.end(), [&](const T &i) -> bool);
6  void copy(inp.begin(), inp.end(), out.begin());
7  int count(v.begin(), v.end(), int val); // number of val in v
8  it unique(v.begin(), v.end()); // it - v.begin() = size
9  // after calling, v[nth] will be n-th smallest elem in v
10 void nth_element(v.begin(), nth_it, bin_comp);
11 void merge(in1.begin(), in1.end(), in2.begin(), in2.end(), out.begin(),
   comp);
12 // include union, intersection, difference, symmetric_difference(xor)
13 void set_union(in1.begin(), in1.end(), in2.begin(), in2.end(), out.
   begin(), comp);
14 bool next_permutation(v.begin(), v.end());
15 // v1, v2 need sorted already, whether v1 includes v2
16 bool inclues(v1.begin(), v1.end(), v2.begin(), v2.end());
17 it find(v.begin(), v.end(), T val); // = v.end() if not found
18 it search(v1.begin(), v1.end(), v2.begin(), v2.end());
19 it lower_bound(v.begin(), v.end(), T val);
20 it upper_bound(v.begin(), v.end(), T val);
21 bool binary_search(v.begin(), v.end(), T val); // exist in v ?
22 void sort(v.begin(), v.end(), comp);
23 void stable_sort(v.begin(), v.end(), comp);

```

3.8.4 String

3.8.5 Priority Queue

```

1  bool cmp(ii a, ii b)
2  {
3      if(a.first == b.first)
4          return a.second > b.second;
5      return b.first > a.first;
6  }
7
8  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

```

1 // BIT is 1-based
2 const int MAX_N = 20000; //這個記得改!
3 ll bit[MAX_N + 1];
4
5 int sum(int i) {
6     int s = 0;
7     while (i > 0) {
8         s += bit[i];
9         i -= (i & -i);
10    }
11    return s;
12 }
13
14 void add(int i, int x) {
15     while (i <= MAX_N) {
16         bit[i] += x;
17         i += (i & -i);
18    }
19 }
```

5.2 2D BIT

```

1 // BIT is 1-based
2 const int MAX_N = 20000, MAX_M = 20000; //這個記得改!
3 ll bit[MAX_N + 1][MAX_M + 1];
4
5 ll sum(int a, int b) {
6     ll s = 0;
7     for (int i = a; i > 0; i -= (i & -i))
8         for (int j = b; j > 0; j -= (j & -j))
9             s += bit[i][j];
10    return s;
11 }
12
13 void add(int a, int b, ll x) {
14     // MAX_N, MAX_M 須適時調整!
15     for (int i = a; i <= MAX_N; i += (i & -i))
```

```

16         for (int j = b; j <= MAX_M; j += (j & -j))
17             bit[i][j] += x;
18     }
```

5.3 Union Find

```

1 #define N 20000 // 記得改
2 struct UFDS {
3     int par[N];
4
5     void init() {
6         memset(par, -1, sizeof(par));
7     }
8
9     int root(int x) {
10        return par[x] < 0 ? x : par[x] = root(par[x]);
11    }
12
13    void merge(int x, int y) {
14        x = root(x);
15        y = root(y);
16
17        if (x != y) {
18            if (par[x] > par[y])
19                swap(x, y);
20            par[x] += par[y];
21            par[y] = x;
22        }
23    }
24 }
```

5.4 Segment Tree

5.5 Sparse Table

```

1 struct {
2     int sp[MAX_LOG_N][MAX_N]; // MAX_LOG_N = ceil(lg(MAX_N))
3
4     void build(int inp[], int n) {
5         for (int j = 0; j < n; j++) {
6             sp[0][j] = inp[j];
7         }
8
9         for (int i = 1; (1 << i) <= n; i++)
10            for (int j = 0; j + (1 << i) <= n; j++)
11                sp[i][j] =
12                    min(sp[i - 1][j], sp[i - 1][j + (1 << (i - 1))]);
13    }
14
15    int query(int l, int r) { // [l, r)
16        int k = floor(log2(r - l));
17
18        return min(sp[k][l], sp[k][r - (1 << k)]);
19    }
20 } 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 Dijkstra

8.3.2 Dijkstra (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

9.2 Z Algorithm

9.3 Trie

9.4 Suffix Array

10 Geometry

10.1 Template

```

1 // C++ routines for computational geometry.
2
3 #include <cassert>
4 #include <cmath>
5 #include <iostream>
6 #include <vector>
7
8 using namespace std;
9
10 double INF = 1e100;
11 double EPS = 1e-12;
12
13 struct PT {
14     double x, y;
15     PT() {}
16     PT(double x, double y) : x(x), y(y) {}
17     PT(const PT &p) : x(p.x), y(p.y) {}
18     PT operator+(const PT &p) const
19     {
20         return PT(x + p.x, y + p.y);
21     }
22     PT operator-(const PT &p) const
23     {
24         return PT(x - p.x, y - p.y);
25     }
26     PT operator*(double c) const
27     {
28         return PT(x * c, y * c);
29     }
30     PT operator/(double c) const
31     {
32         return PT(x / c, y / c);
33     }
34 };
35
36 double dot(PT p, PT q)
37 {
38     return p.x * q.x + p.y * q.y;
39 }
40 double dist2(PT p, PT q)
41 {
42     return dot(p - q, p - q);
43 }
44 double cross(PT p, PT q)
45 {
46     return p.x * q.y - p.y * q.x;
47 }
48 ostream &operator<<(ostream &os, const PT &p)
49 {
50     os << "(" << p.x << ", " << p.y << ")";
51 }
52
53 // rotate a point CCW or CW around the origin
54 PT RotateCCW90(PT p)
55 {
56     return PT(-p.y, p.x);

```

```

57 }
58 PT RotateCW90(PT p)
59 {
60     return PT(p.y, -p.x);
61 }
62 PT RotateCCW(PT p, double t)
63 {
64     return PT(p.x * cos(t) - p.y * sin(t), p.x * sin(t) + p.y * cos(t));
65 }
66 // project point c onto line through a and b
67 // assuming a != b
68 PT ProjectPointLine(PT a, PT b, PT c)
69 {
70     return a + (b - a) * dot(c - a, b - a) / dot(b - a, b - a);
71 }
72 // project point c onto line segment through a and b
73 PT ProjectPointSegment(PT a, PT b, PT c)
74 {
75     double r = dot(b - a, b - a);
76     if (fabs(r) < EPS)
77         return a;
78     r = dot(c - a, b - a) / r;
79     if (r < 0)
80         return a;
81     if (r > 1)
82         return b;
83     return a + (b - a) * r;
84 }
85 // compute distance from c to segment between a and b
86 double DistancePointSegment(PT a, PT b, PT c)
87 {
88     return sqrt(dist2(c, ProjectPointSegment(a, b, c)));
89 }
90 // compute distance between point (x,y,z) and plane ax+by+cz=d
91 double DistancePointPlane(double x, double y, double z, double a,
92     double b,
93     double c, double d)
94 {
95     return fabs(a * x + b * y + c * z - d) / sqrt(a * a + b * b + c * c);
96 }
97 // determine if lines from a to b and c to d are parallel or collinear
98 bool LinesParallel(PT a, PT b, PT c, PT d)
99 {
100     return fabs(cross(b - a, c - d)) < EPS;
101 }
102 bool LinesCollinear(PT a, PT b, PT c, PT d)
103 {

```

```

104     return LinesParallel(a, b, c, d) && fabs(cross(a - b, a - c)) < EPS
105         &&
106             fabs(cross(c - d, c - a)) < EPS;
107 }
108 // determine if line segment from a to b intersects with
109 // line segment from c to d
110 bool SegmentsIntersect(PT a, PT b, PT c, PT d)
111 {
112     if (LinesCollinear(a, b, c, d)) {
113         if (dist2(a, c) < EPS || dist2(a, d) < EPS || dist2(b, c) < EPS
114             ||
115                 dist2(b, d) < EPS)
116             return true;
117         if (dot(c - a, c - b) > 0 && dot(d - a, d - b) > 0 && dot(c - b,
118             d - b) > 0)
119             return false;
120         return true;
121     }
122     if (cross(d - a, b - a) * cross(c - a, b - a) > 0)
123         return false;
124     if (cross(a - c, d - c) * cross(b - c, d - c) > 0)
125         return false;
126     return true;
127 }
128 // compute intersection of line passing through a and b
129 // with line passing through c and d, assuming that unique
130 // intersection exists; for segment intersection, check if
131 // segments intersect first
132 PT ComputeLineIntersection(PT a, PT b, PT c, PT d)
133 {
134     b = b - a;
135     d = c - d;
136     c = c - a;
137     assert(dot(b, b) > EPS && dot(d, d) > EPS);
138     return a + b * cross(c, d) / cross(b, d);
139 }
140 // compute center of circle given three points
141 PT ComputeCircleCenter(PT a, PT b, PT c)
142 {
143     b = (a + b) / 2;
144     c = (a + c) / 2;
145     return ComputeLineIntersection(b, b + RotateCW90(a - b), c,
146         c + RotateCW90(a - c));
147 }
148 // determine if point is in a possibly non-convex polygon (by William
149 // Randolph Franklin); returns 1 for strictly interior points, 0 for
150 // strictly exterior points, and 0 or 1 for the remaining points.
151 // Note that it is possible to convert this into an *exact* test using
152 // integer arithmetic by taking care of the division appropriately
153 // (making sure to deal with signs properly) and then by writing exact
154 // tests for checking point on polygon boundary
155 bool PointInPolygon(const vector<PT> &p, PT q)

```

```

162 {
163     bool c = 0;
164     for (int i = 0; i < p.size(); i++) {
165         int j = (i + 1) % p.size();
166         if ((p[i].y <= q.y && q.y < p[j].y || p[j].y <= q.y && q.y < p[
167             i].y) &&
168             q.x < p[i].x + (p[j].x - p[i].x) * (q.y - p[i].y) / (p[j].y
169                 - p[i].y))
170             c = !c;
171     }
172     return c;
173 }
174 // determine if point is on the boundary of a polygon
175 bool PointOnPolygon(const vector<PT> &p, PT q)
176 {
177     for (int i = 0; i < p.size(); i++)
178         if (dist2(ProjectPointSegment(p[i], p[(i + 1) % p.size()], q),
179             q) < EPS)
180             return true;
181     return false;
182 }
183 // compute intersection of line through points a and b with
184 // circle centered at c with radius r > 0
185 vector<PT> CircleLineIntersection(PT a, PT b, PT c, double r)
186 {
187     vector<PT> ret;
188     b = b - a;
189     a = a - c;
190     double A = dot(b, b);
191     double B = dot(a, b);
192     double C = dot(a, a) - r * r;
193     double D = B * B - A * C;
194     if (D < -EPS)
195         return ret;
196     ret.push_back(c + a + b * (-B + sqrt(D + EPS)) / A);
197     if (D > EPS)
198         ret.push_back(c + a + b * (-B - sqrt(D)) / A);
199     return ret;
200 }
201 // compute intersection of circle centered at a with radius r
202 // with circle centered at b with radius R
203 vector<PT> CircleCircleIntersection(PT a, PT b, double r, double R)
204 {
205     vector<PT> ret;
206     double d = sqrt(dist2(a, b));
207     if (d > r + R || d + min(r, R) < max(r, R))
208         return ret;
209     double x = (d * d - R * R + r * r) / (2 * d);
210     double y = sqrt(r * r - x * x);
211     PT v = (b - a) / d;
212     ret.push_back(a + v * x + RotateCCW90(v) * y);
213     if (y > 0)
214         ret.push_back(a + v * x - RotateCCW90(v) * y);

```

```

215     return ret;
216 }
217 // This code computes the area or centroid of a (possibly nonconvex)
218 // polygon, assuming that the coordinates are listed in a clockwise or
219 // counterclockwise fashion. Note that the centroid is often known as
220 // the "center of gravity" or "center of mass".
221 double ComputeSignedArea(const vector<PT> &p)
222 {
223     double area = 0;
224     for (int i = 0; i < p.size(); i++) {
225         int j = (i + 1) % p.size();
226         area += p[i].x * p[j].y - p[j].x * p[i].y;
227     }
228     return area / 2.0;
229 }
230 double ComputeArea(const vector<PT> &p)
231 {
232     return fabs(ComputeSignedArea(p));
233 }
234 PT ComputeCentroid(const vector<PT> &p)
235 {
236     PT c(0, 0);
237     double scale = 6.0 * ComputeSignedArea(p);
238     for (int i = 0; i < p.size(); i++) {
239         int j = (i + 1) % p.size();
240         c = c + (p[i] + p[j]) * (p[i].x * p[j].y - p[j].x * p[i].y);
241     }
242     return c / scale;
243 }
244 // tests whether or not a given polygon (in CW or CCW order) is simple
245 bool IsSimple(const vector<PT> &p)
246 {
247     for (int i = 0; i < p.size(); i++) {
248         for (int k = i + 1; k < p.size(); k++) {
249             int j = (i + 1) % p.size();
250             int l = (k + 1) % p.size();
251             if (i == l || j == k)
252                 continue;
253             if (SegmentsIntersect(p[i], p[j], p[k], p[l]))
254                 return false;
255         }
256     }
257     return true;
258 }
259 int main()
260 {
261     // expected: (-5,2)
262     cerr << RotateCCW90(PT(2, 5)) << endl;
263     // expected: (5,-2)

```



```

271 cerr << RotateCW90(PT(2, 5)) << endl;
272
273 // expected: (-5,2)
274 cerr << RotateCCW(PT(2, 5), M_PI / 2) << endl;
275
276 // expected: (5,2)
277 cerr << ProjectPointLine(PT(-5, -2), PT(10, 4), PT(3, 7)) << endl;
278
279 // expected: (5,2) (7.5,3) (2.5,1)
280 cerr << ProjectPointSegment(PT(-5, -2), PT(10, 4), PT(3, 7)) << " "
281 << ProjectPointSegment(PT(7.5, 3), PT(10, 4), PT(3, 7)) << " "
282 << ProjectPointSegment(PT(-5, -2), PT(2.5, 1), PT(3, 7)) <<
endl;
283
284 // expected: 6.78903
285 cerr << DistancePointPlane(4, -4, 3, 2, -2, 5, -8) << endl;
286
287 // expected: 1 0 1
288 cerr << LinesParallel(PT(1, 1), PT(3, 5), PT(2, 1), PT(4, 5)) << "
"
289 << LinesParallel(PT(1, 1), PT(3, 5), PT(2, 0), PT(4, 5)) << "
"
290 << LinesParallel(PT(1, 1), PT(3, 5), PT(5, 9), PT(7, 13)) <<
endl;
291
292 // expected: 0 0 1
293 cerr << LinesCollinear(PT(1, 1), PT(3, 5), PT(2, 1), PT(4, 5)) << "
"
294 << LinesCollinear(PT(1, 1), PT(3, 5), PT(2, 0), PT(4, 5)) << "
"
295 << LinesCollinear(PT(1, 1), PT(3, 5), PT(5, 9), PT(7, 13)) <<
endl;
296
297 // expected: 1 1 1 0
298 cerr << SegmentsIntersect(PT(0, 0), PT(2, 4), PT(3, 1), PT(-1, 3))
<< " "
299 << SegmentsIntersect(PT(0, 0), PT(2, 4), PT(4, 3), PT(0, 5))
<< " "
300 << SegmentsIntersect(PT(0, 0), PT(2, 4), PT(2, -1), PT(-2, 1))
<< " "
301 << SegmentsIntersect(PT(0, 0), PT(2, 4), PT(5, 5), PT(1, 7))
<< endl;
302
303 // expected: (1,2)
304 cerr << ComputeLineIntersection(PT(0, 0), PT(2, 4), PT(3, 1), PT
(-1, 3))
305 << endl;
306
307 // expected: (1,1)
308 cerr << ComputeCircleCenter(PT(-3, 4), PT(6, 1), PT(4, 5)) << endl;
309
310 vector<PT> v;
311 v.push_back(PT(0, 0));
312 v.push_back(PT(5, 0));
313 v.push_back(PT(5, 5));
314 v.push_back(PT(0, 5));

```

```

315 // expected: 1 1 1 0 0
316 cerr << PointInPolygon(v, PT(2, 2)) << " " << PointInPolygon(v, PT
(2, 0))
317 << " " << PointInPolygon(v, PT(0, 2)) << " "
318 << PointInPolygon(v, PT(5, 2)) << " " << PointInPolygon(v, PT
(2, 5))
319 << endl;
320
321 // expected: 0 1 1 1 1
322 cerr << PointOnPolygon(v, PT(2, 2)) << " " << PointOnPolygon(v, PT
(2, 0))
323 << " " << PointOnPolygon(v, PT(0, 2)) << " "
324 << PointOnPolygon(v, PT(5, 2)) << " " << PointOnPolygon(v, PT
(2, 5))
325 << endl;
326
327 // expected: (1,6)
328 // (5,4) (4,5)
329 // blank line
330 // (4,5) (5,4)
331 // blank line
332 // (4,5) (5,4)
333 vector<PT> u = CircleLineIntersection(PT(0, 6), PT(2, 6), PT(1, 1),
5);
334 for (int i = 0; i < u.size(); i++)
335 cerr << u[i] << " ";
336 cerr << endl;
337 u = CircleLineIntersection(PT(0, 9), PT(9, 0), PT(1, 1), 5);
338 for (int i = 0; i < u.size(); i++)
339 cerr << u[i] << " ";
340 cerr << endl;
341 u = CircleCircleIntersection(PT(1, 1), PT(10, 10), 5, 5);
342 for (int i = 0; i < u.size(); i++)
343 cerr << u[i] << " ";
344 cerr << endl;
345 u = CircleCircleIntersection(PT(1, 1), PT(8, 8), 5, 5);
346 for (int i = 0; i < u.size(); i++)
347 cerr << u[i] << " ";
348 cerr << endl;
349 u = CircleCircleIntersection(PT(1, 1), PT(4.5, 4.5), 10, sqrt(2.0)
/ 2.0);
350 for (int i = 0; i < u.size(); i++)
351 cerr << u[i] << " ";
352 cerr << endl;
353 u = CircleCircleIntersection(PT(1, 1), PT(4.5, 4.5), 5, sqrt(2.0) /
2.0);
354 for (int i = 0; i < u.size(); i++)
355 cerr << u[i] << " ";
356 cerr << endl;
357
358 // area should be 5.0
359 // centroid should be (1.1666666, 1.1666666)
360 PT pa[] = {PT(0, 0), PT(5, 0), PT(1, 1), PT(0, 5)};
361 vector<PT> p(pa, pa + 4);
362 PT c = ComputeCentroid(p);
363

```

```

364 |     cerr << "Area: " << ComputeArea(p) << endl;
365 |     cerr << "Centroid: " << c << endl;
366 |
367 |     return 0;
368 | }

```

=====

```

1 | #define x first
2 | #define y second
3 | typedef pair <double , double > pt;
4 | struct line {
5 |     double a, b, c;
6 |     // coefficients in general form, compare up to constant factor
7 | }

```

```

8 | pt operator-(pt u, pt v) { return pt(u.x-v.x, u.y-v.y); }
9 | pt operator+(pt u, pt v) { return pt(u.x+v.x, u.y+v.y); }
10 | pt operator*(pt u, double d) { return pt(u.x*d, u.y*d); }
11 | 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
12 | 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