

Содержание

1	Strategy.txt	1
2	flows/dinic.cpp	2
3	flows/globalcut.cpp	3
4	flows/hungary.cpp	3
5	flows/mincost.cpp	4
6	geometry/convex_hull.cpp	5
7	geometry/halfplanes.cpp	5
8	geometry/i.cpp	6
9	geometry/polygon.cpp	7
10	geometry/primitives.cpp	7
11	geometry/shortest-bridge/check.cpp	9
12	geometry/shortest-bridge/i.cpp	9
13	geometry/shortest-bridge/solutions/i2.cpp	12
14	geometry/svg.cpp	15
15	graphs/2sat.cpp	15
16	graphs/directed_mst.cpp	16
17	graphs/euler_cycle.cpp	17
18	math/fft_recursive.cpp	17
19	math/golden_search.cpp	18
20	math/numbers.txt	18
21	strings/automaton.cpp	19
22	strings/eertree.cpp	19
23	strings/suffix_array.cpp	20
24	strings/ukkonen.cpp	20
25	structures/convex_hull_trick.cpp	21
26	structures/heavy_light.cpp	22
27	structures/linkcut.cpp	23
28	structures/ordered_set.cpp	24
29	structures/treap.cpp	24

1 Strategy.txt

1	- Проверить руками сэмплы	1
	- Подумать как дебагать после написания	
2	- Выписать сложные формулы и все +-1	2
3	- Проверить имена файлов	3
	- Прогнать сэмплы	
	- Переполнения int, переполнения long long	
4	- Выход за границу массива: _GLIBCXX_DEBUG	4
	- Переполнения по модулю: в	
5	→ псевдо-онлайн-генераторе, в функциях-обертках	5
	- Проверить мультитест на разных тестах	
	- Прогнать минимальный по каждому параметру тест	
6	- Прогнать псевдо-максимальный тест(немного	6
	→ чисел, но очень большие или очень маленькие)	
7	- Представить что не зайдет и заранее написать	7
	→ assert'ы, прогнать слегка модифицированные	
7	→ тесты	7
	- cout.precision: в том числе в интерактивных	
9	→ задачах	9
	- Удалить debug-output, отсечения для тестов,	
9	→ вернуть оригинальный main, удалить	9
12	→ _GLIBCXX_DEBUG	12
15	- Вердикт может врать	15
	- Если много тестов(>3), дописать в конец каждого	
15	→ теста ответ, чтобы не забыть	15
	- (WA) Потестить не только ответ, но и содержимое	
16	→ значимых массивов, переменных	16
	- (WA) Изменить тест так, чтобы ответ не менялся:	
17	→ поменять координаты местами, сжать/растянуть	17
	→ координаты, поменять ROOT дерева	
17	- (WA) Подвигать размер блока в корневой или	17
18	→ битсете	18
	- (WA) Поставить assert'ы, возможно написать	
18	→ чекер с assert'ом	18
	- (WA) Проверить, что программа не печатает	
19	→ что-либо неожиданное, что должно попадать под	19
	PE: inf - 2, не лекс. мин. решение, одинаковые	
19	→ числа вместо разных, неправильное количество	19
	→ чисел, пустой ответ, перечитать output format	
20	- (TL) cin -> scanf -> getchar	20
	- (TL) Упихать в кэш большие массивы, поменять	
20	→ местами for'ы или измерения массива	20
	- (RE) Проверить формулы на деление на 0, выход	
21	→ за область определения(sqrt(-eps), acos(1 +	21
	→ eps))	
22	- (WA) Проверить, что ответ влезает в int	22

2 flows/dinic.cpp

```

1 namespace Dinic {
2 const int maxn = 10010;
3
4 struct Edge {
5     int to, c, f;
6 } es[maxn*2];
7 int ne = 0;
8
9 int n;
10 vector<int> e[maxn];
11 int q[maxn], d[maxn], pos[maxn];
12 int S, T;
13
14 void addEdge(int u, int v, int c) {
15     assert(c <= 1000000000);
16     es[ne] = {v, c, 0};
17     e[u].push_back(ne++);
18     es[ne] = {u, 0, 0};
19     e[v].push_back(ne++);
20 }
21
22 bool bfs() {
23     for(i, n) d[i] = maxn;
24     d[S] = 0, q[0] = S;
25     int lq = 0, rq = 1;
26     while (lq != rq) {
27         int v = q[lq++];
28         for (int id: e[v]) if (es[id].f < es[id].c) {
29             int to = es[id].to;
30             if (d[to] == maxn)
31                 d[to] = d[v] + 1, q[rq++] = to;
32         }
33     }
34     return d[T] != maxn;
35 }
36
37 int dfs(int v, int curf) {
38     if (v == T || curf == 0) return curf;
39     for (int &i = pos[v]; i < (int)e[v].size(); ++i) {
40         int id = e[v][i];
41         int to = es[id].to;
42         if (es[id].f < es[id].c && d[v] + 1 == d[to]) {
43             int ret = dfs(to, min(curf, es[id].c -
44 ↪ es[id].f));
45             es[id].f += ret;
46             es[id^1].f -= ret;
47             return ret;
48         }
49     }
50     return 0;
51 }
52
53 i64 dinic(int S, int T) {
54     Dinic::S = S, Dinic::T = T;
55     i64 res = 0;
56     while (bfs()) {
57         for(i, n) pos[i] = 0;
58         while (int f = dfs(S, 1e9)) {
59             assert(f <= 1000000000);
60             res += f;
61         }
62     }
63     return res;
64 }
65
66 } // namespace Dinic
67
68 void test() {
69     Dinic::n = 4;
70     Dinic::addEdge(0, 1, 1);
71     Dinic::addEdge(0, 2, 2);
72     Dinic::addEdge(2, 1, 1);
73     Dinic::addEdge(1, 3, 2);
74     Dinic::addEdge(2, 3, 1);
75     cout << Dinic::dinic(0, 3) << endl; // 3
76
77 }
78
79 /*
80 * LR-поток.
81 * LR-поток находит не максимальный поток.
82 * Добавим новый сток S' и исток T'. Заменяем ребро (u, v, l,
83 ↪ r) LR-сети
84 * на ребра (u, T', l), (S', v, l), (u, v, r - l).
85 * Добавим ребро (T, S, k). Ставим значение k=inf, пускаем
86 ↪ поток.
87 * Проверяем, что все ребра из S' насыщены (иначе ответ не
88 ↪ существует).

```

```

86 * Бинпоиском находим наименьшее k, что величина потока не
87 ↪ изменится.
87 * Это k - величина МИНИМАЛЬНОГО потока, удовлетворяющего
88 ↪ ограничениям.
88 */

```

3 flows/globalcut.cpp

```

1#include <bits/stdc++.h>
2using namespace std;
3#define forn(i,n) for (int i = 0; i < int(n); ++i)
4const int inf = 1e9 + 1e5;
5
6const int maxn = 505;
7namespace StoerWagner {
8    int g[maxn][maxn];
9    int dist[maxn];
10    bool used[maxn];
11    int n;
12
13    void addEdge(int u, int v, int c) {
14        g[u][v] += c;
15        g[v][u] += c;
16    }
17
18    int run() {
19        vector<int> vertices;
20        forn (i, n)
21            vertices.push_back(i);
22        int mincut = inf;
23        while (vertices.size() > 1) {
24            int u = vertices[0];
25            for (auto v: vertices) {
26                used[v] = false;
27                dist[v] = g[u][v];
28            }
29            used[u] = true;
30            forn (ii, vertices.size() - 2) {
31                for (auto v: vertices)
32                    if (!used[v])
33                        if (used[u] || dist[v] > dist[u])
34                            u = v;
35                used[u] = true;
36                for (auto v: vertices)
37                    if (!used[v])
38                        dist[v] += g[u][v];
39            }
40            int t = -1;
41            for (auto v: vertices)
42                if (!used[v])
43                    t = v;
44            assert(t != -1);
45            mincut = min(mincut, dist[t]);
46            vertices.erase(find(vertices.begin(),
↪ vertices.end(), t));
47            for (auto v: vertices)
48                addEdge(u, v, g[v][t]);
49        }
50        return mincut;
51    }
52};
53
54int main() {
55    StoerWagner::n = 4;
56    StoerWagner::addEdge(0, 1, 5);
57    StoerWagner::addEdge(2, 3, 5);
58    StoerWagner::addEdge(1, 2, 4);
59    cerr << StoerWagner::run() << '\n';
60}

```

4 flows/hungary.cpp

```

1// left half is the smaller one
2namespace Hungary {
3    const int maxn = 505;
4    int a[maxn][maxn];
5    int p[2][maxn];
6    int match[maxn];
7    bool used[maxn];
8    int from[maxn];
9    int mind[maxn];
10    int n, m;
11
12    int hungary(int v) {
13        used[v] = true;
14        int u = match[v];
15        int best = -1;
16        forn (i, m + 1) {
17            if (used[i])
18                continue;
19            int nw = a[u][i] - p[0][u] - p[1][i];
20            if (nw <= mind[i]) {
21                mind[i] = nw;
22                from[i] = v;
23            }
24            if (best == -1 || mind[best] > mind[i])
25                best = i;
26        }
27        v = best;
28        int delta = mind[best];
29        forn (i, m + 1) {
30            if (used[i]) {
31                p[1][i] -= delta;
32                p[0][match[i]] += delta;
33            } else
34                mind[i] -= delta;
35        }
36        if (match[v] == -1)
37            return v;
38        return hungary(v);
39    }
40
41    void check() {
42        int edges = 0, res = 0;
43        forn (i, m)
44            if (match[i] != -1) {
45                ++edges;
46                assert(p[0][match[i]] + p[1][i] ==
↪ a[match[i]][i]);
47                res += a[match[i]][i];
48            } else
49                assert(p[1][i] == 0);
50        assert(res == -p[1][m]);
51        forn (i, n) forn (j, m)
52            assert(p[0][i] + p[1][j] <= a[i][j]);
53    }
54
55    int run() {
56        forn (i, n)
57            p[0][i] = 0;
58        forn (i, m + 1) {
59            p[1][i] = 0;
60            match[i] = -1;
61        }
62        forn (i, n) {
63            match[m] = i;
64            fill(used, used + m + 1, false);
65            fill(mind, mind + m + 1, inf);
66            fill(from, from + m + 1, -1);
67            int v = hungary(m);
68            while (v != m) {
69                int w = from[v];
70                match[v] = match[w];
71                v = w;
72            }
73        }
74        check();
75        return -p[1][m];
76    }
77};

```

5 flows/mincost.cpp

```

1 namespace MinCost {
2     const ll infc = 1e12;
3
4     struct Edge {
5         int to;
6         ll c, f, cost;
7
8         Edge(int to, ll c, ll cost): to(to), c(c), f(0),
↪ cost(cost) {
9     }
10 };
11
12 int N, S, T;
13 int totalFlow;
14 ll totalCost;
15 const int maxn = 505;
16 vector<Edge> edge;
17 vector<int> g[maxn];
18
19 void addEdge(int u, int v, ll c, ll cost) {
20     g[u].push_back(edge.size());
21     edge.emplace_back(v, c, cost);
22     g[v].push_back(edge.size());
23     edge.emplace_back(u, 0, -cost);
24 }
25
26 ll dist[maxn];
27 int fromEdge[maxn];
28
29 bool inQueue[maxn];
30 bool fordBellman() {
31     for (i, N)
32         dist[i] = infc;
33     dist[S] = 0;
34     inQueue[S] = true;
35     vector<int> q;
36     q.push_back(S);
37     for (int ii = 0; ii < int(q.size()); ++ii) {
38         int u = q[ii];
39         inQueue[u] = false;
40         for (int e: g[u]) {
41             if (edge[e].f == edge[e].c)
42                 continue;
43             int v = edge[e].to;
44             ll nw = edge[e].cost + dist[u];
45             if (nw >= dist[v])
46                 continue;
47             dist[v] = nw;
48             fromEdge[v] = e;
49             if (!inQueue[v]) {
50                 inQueue[v] = true;
51                 q.push_back(v);
52             }
53         }
54     }
55     return dist[T] != infc;
56 }
57
58 ll pot[maxn];
59 bool dikstra() {
60     priority_queue<pair<ll, int>, vector<pair<ll, int>>,
↪ greater<pair<ll, int>>> q;
61     for (i, N)
62         dist[i] = infc;
63     dist[S] = 0;
64     q.emplace(dist[S], S);
65     while (!q.empty()) {
66         int u = q.top().second;
67         ll cdist = q.top().first;
68         q.pop();
69         if (cdist != dist[u])
70             continue;
71         for (int e: g[u]) {
72             int v = edge[e].to;
73             if (edge[e].c == edge[e].f)
74                 continue;
75             ll w = edge[e].cost + pot[u] - pot[v];
76             assert(w >= 0);
77             ll ndist = w + dist[u];
78             if (ndist >= dist[v])
79                 continue;
80             dist[v] = ndist;
81             fromEdge[v] = e;
82             q.emplace(dist[v], v);
83         }
84     }
85     if (dist[T] == infc)
86         return false;
87     for (i, N) {
88         if (dist[i] == infc)
89             continue;
90         pot[i] += dist[i];
91     }
92     return true;
93 }
94
95 bool push() {
96     //2 variants
97     //if (!fordBellman())
98     if (!dikstra())
99         return false;
100     ++totalFlow;
101     int u = T;
102     while (u != S) {
103         int e = fromEdge[u];
104         totalCost += edge[e].cost;
105         edge[e].f++;
106         edge[e ^ 1].f--;
107         u = edge[e ^ 1].to;
108     }
109     return true;
110 }
111
112 int main() {
113     MinCost::N = 3, MinCost::S = 1, MinCost::T = 2;
114     MinCost::addEdge(1, 0, 3, 5);
115     MinCost::addEdge(0, 2, 4, 6);
116     while (MinCost::push());
117     cout << MinCost::totalFlow << ' ' << MinCost::totalCost <<
↪ '\n'; //3 33
118 }
119 }

```

6 geometry/convex_hull.cpp

```

1#include <bits/stdc++.h>
2using namespace std;
3#define forn(i, n) for (int i = 0; i < int(n); ++i)
4#define sz(x) ((int) (x).size())
5
6#include "primitives.cpp"
7
8bool cmpAngle(const pt &a, const pt &b) {
9    bool ar = a.right(), br = b.right();
10    if (ar ^ br)
11        return ar;
12    return gt(a % b, 0);
13}
14
15struct Hull {
16    vector<pt> top, bot;
17
18    void append(pt p) {
19        while (bot.size() > 1 && ge((p - bot.back()) %
    ↪ (bot.back() - *next(bot.rbegin()), 0))
20            bot.pop_back();
21        bot.push_back(p);
22        while (top.size() > 1 && ge(0, (p - top.back()) %
    ↪ (top.back() - *next(top.rbegin()))))
23            top.pop_back();
24        top.push_back(p);
25    }
26
27    void build(vector<pt> h) {
28        sort(h.begin(), h.end());
29        h.erase(unique(h.begin(), h.end()), h.end());
30        top.clear(), bot.clear();
31        for (pt p: h)
32            append(p);
33    }
34
35    pt kth(int k) {
36        if (k < sz(bot))
37            return bot[k];
38        else
39            return top[sz(top) - (k - sz(bot)) - 2];
40    }
41
42    pt mostDistant(pt dir) {
43        if (bot.empty()) {
44            //empty hull
45            return pt{1e18, 1e18};
46        }
47        if (bot.size() == 1)
48            return bot.back();
49        dir = dir.rot();
50        int n = sz(top) + sz(bot) - 2;
51        int L = -1, R = n;
52        while (L + 1 < R) {
53            int C = (L + R) / 2;
54            pt v = kth((C + 1) % n) - kth(C);
55            if (cmpAngle(dir, v)) //finds upper bound
56                R = C;
57            else
58                L = C;
59        }
60        return kth(R % n);
61    }
62};

```

7 geometry/halfplanes.cpp

```

1#include <bits/stdc++.h>
2using namespace std;
3#define forn(i, n) for (int i = 0; i < int(n); ++i)
4#define forab(i, a, b) for (int i = int(a); i < int(b); ++i)
5#include "primitives.cpp"
6
7ld det3x3(line &l1, line &l2, line &l3) {
8    return l1.a * (l2.b * l3.c - l2.c * l3.b) +
9        l1.b * (l2.c * l3.a - l2.a * l3.c) +
10        l1.c * (l2.a * l3.b - l2.b * l3.a);
11}
12
13vector<pt> halfplanesInterseccion(vector<line> lines) {
14    sort(lines.begin(), lines.end(), [](const line &a, const
    ↪ line &b) {
15        bool ar = a.right(), br = b.right();
16        if (ar ^ br)
17            return ar;
18        ld prod = (pt{a.a, a.b} % pt{b.a, b.b});
19        if (!eq(prod, 0))
20            return prod > 0;
21        return a.c < b.c;
22    });
23    vector<line> lines2;
24    pt pr;
25    forn (i, lines.size()) {
26        pt cur{lines[i].a, lines[i].b};
27        if (i == 0 || cur != pr)
28            lines2.push_back(lines[i]);
29        pr = cur;
30    }
31    lines = lines2;
32    int n = lines.size();
33    forn (i, n)
34        lines[i].id = i;
35    vector<line> hull;
36    forn (i, 2 * n) {
37        line l = lines[i % n];
38        while ((int) hull.size() >= 2) {
39            ld D = det3x3(*prev(prev(hull.end())),
    ↪ hull.back(), l);
40            if (ge(D, 0))
41                break;
42            hull.pop_back();
43        }
44        hull.push_back(l);
45    }
46    vector<int> firstTime(n, -1);
47    vector<line> v;
48    forn (i, hull.size()) {
49        int cid = hull[i].id;
50        if (firstTime[cid] == -1) {
51            firstTime[cid] = i;
52            continue;
53        }
54        forab(j, firstTime[cid], i)
55            v.push_back(hull[j]);
56        break;
57    }
58    n = v.size();
59    if (v.empty()) {
60        //empty intersection
61        return {};
62    }
63    v.push_back(v[0]);
64    vector<pt> res;
65    pt center{0, 0};
66    forn (i, n) {
67        res.push_back(linesIntersection(v[i], v[i + 1]));
68        center = center + res.back();
69    }
70    center = center / n;
71    for (auto l: lines)
72        if (gt(0, l.signedDist(center))) {
73            //empty intersection
74            return {};
75        }
76    return res;
77}

```

8 geometry/i.cpp

```

1#include <bits/stdc++.h>
2using namespace std;
3#define forn(i,n) for (int i = 0; i < int(n); ++i)
4
5#include "polygon.cpp"
6
7const int maxn = 33;
8vector<pt> side[2];
9vector<pt> poly[2];
10int s[2];
11ld dist[2][maxn];
12ld g[2][maxn][maxn];
13
14pt nearest(pt p, pt a, pt b) {
15    pt res = a;
16    if ((b - p).abs() < (a - p).abs())
17        res = b;
18    if (ge((p - a) * (b - a), 0) && ge((p - b) * (a - b), 0))
19        {
20            pt v = (b - a).rot();
21            v /= v.abs();
22        }
23}
24
25ld distance(pt p, int q) {
26    ld res = 1e18;
27    forn (i, side[q].size())
28        if (segmentInsidePolygon(p, side[q][i],
29            poly[q].data(), poly[q].size()))
30            res = min(res, dist[q][i] + (side[q][i] -
31                p).abs());
32    return res;
33}
34
35int main() {
36    cout.precision(10);
37    cout.setf(ios::fixed);
38    #ifdef LOCAL
39        assert(freopen("i.in", "r", stdin));
40    #endif
41    pt S, T;
42    cin >> S >> T;
43    forn (q, 2) {
44        int n;
45        cin >> n;
46        s[q] = n;
47        forn (i, n) {
48            pt x;
49            cin >> x;
50            side[q].push_back(x);
51        }
52    }
53    forn (q, 2) {
54        vector<pt> poly(side[q]);
55        if (q == 0) {
56            poly.push_back(pt{0, 1000});
57            poly.push_back(pt{0, 0});
58            side[q].push_back(S);
59        } else {
60            reverse(poly.begin(), poly.end());
61            poly.push_back(pt{1000, 0});
62            poly.push_back(pt{1000, 1000});
63            side[q].push_back(T);
64        }
65        ::poly[q] = poly;
66        int n = side[q].size();
67        forn (i, n)
68            forn (j, n) {
69                if (segmentInsidePolygon(side[q][i],
70                    side[q][j], poly.data(), poly.size()))
71                    g[q][i][j] = (side[q][i] -
72                        side[q][j]).abs();
73                else
74                    g[q][i][j] = 1e18;
75            }
76        forn (k, n)
77            forn (i, n)
78                forn (j, n)
79                    g[q][i][j] = min(g[q][i][j], g[q][i][k] +
80                        g[q][k][j]);
81        forn (i, n)
82            dist[q][i] = g[q][n - 1][i];
83    }
84    ld bridge = 1e18;
85    forn (i, s[0])
86        forn (j, s[1] - 1) {
87            ld cbridge = pointSegmentDist(side[0][i],
88                side[1][j], side[1][j + 1]);
89            if (!eq(cbridge, bridge))
90                continue;
91            //ans = min(ans, distance(
92                p, side[0][i], side[1][j], side[1][j + 1]));
93        }
94    //cout << bridge << '\n';
95}

```

9 geometry/polygon.cpp

```

1#include <bits/stdc++.h>
2using namespace std;
3#define forn(i,n) for (int i = 0; i < int(n); ++i)
4
5#include "primitives.cpp"
6
7bool pointInsidePolygon(pt a, pt *p, int n) {
8    double sumAng = 0;
9    forn (i, n) {
10        pt A = p[i], B = p[(i + 1) % n];
11        if (pointInsideSegment(a, A, B))
12            return true;
13        sumAng += atan2((A - a) % (B - a), (A - a) * (B - a));
14    }
15    return fabs(sumAng) > 1;
16}
17
18//p must be oriented counterclockwise
19bool segmentInsidePolygon(pt a, pt b, pt *p, int n) {
20    if (!pointInsidePolygon((a + b) / 2, p, n))
21        return false;
22    if (a == b)
23        return true;
24    forn (i, n) {
25        pt c = p[i];
26        if (eq((a - c) % (b - c), 0) && gt(0, (a - c) * (b -
27            ↪ c))) {
28            //point on segment
29            pt pr = p[(i + n - 1) % n];
30            pt nx = p[(i + 1) % n];
31            if (gt((c - pr) % (nx - c), 0))
32                return false;
33            ld s1 = (pr - a) % (b - a);
34            ld s2 = (nx - a) % (b - a);
35            if ((gt(s1, 0) || gt(s2, 0)) && (gt(0, s1) ||
36            ↪ gt(0, s2)))
37                return false;
38        }
39        //interval intersection
40        pt d = p[(i + 1) % n];
41        ld s1 = (a - c) % (d - c);
42        ld s2 = (b - c) % (d - c);
43        if (ge(s1, 0) && ge(s2, 0))
44            continue;
45        if (ge(0, s1) && ge(0, s2))
46            continue;
47        s1 = (c - a) % (b - a);
48        s2 = (d - a) % (b - a);
49        if (ge(s1, 0) && ge(s2, 0))
50            continue;
51        if (ge(0, s1) && ge(0, s2))
52            continue;
53        return false;
54    }
55    return true;
56}

```

10 geometry/primitives.cpp

```

1#include <bits/stdc++.h>
2#define forn(i, n) for (int i = 0; i < int(n); ++i)
3using namespace std;
4typedef long double ld;
5
6const ld eps = 1e-9;
7
8bool eq(ld a, ld b) { return fabs(a - b) < eps; }
9bool ge(ld a, ld b) { return a - b > -eps; }
10bool gt(ld a, ld b) { return a - b > eps; }
11ld sqr(ld x) { return x * x; }
12
13#ifdef LOCAL
14#define gassert assert
15#else
16void gassert(bool) {}
17#endif
18
19struct pt {
20    ld x, y;
21
22    pt operator+(const pt &p) const { return pt{x + p.x, y +
23    ↪ p.y}; }
24    pt operator-(const pt &p) const { return pt{x - p.x, y -
25    ↪ p.y}; }
26    ld operator*(const pt &p) const { return x * p.x + y *
27    ↪ p.y; }
28    ld operator%(const pt &p) const { return x * p.y - y *
29    ↪ p.x; }
30
31    pt operator*(const ld &a) const { return pt{x * a, y * a};
32    ↪ }
33    pt operator/(const ld &a) const { gassert(!eq(a, 0));
34    ↪ return pt{x / a, y / a}; }
35    void operator*=(const ld &a) { x *= a, y *= a; }
36    void operator/=(const ld &a) { gassert(!eq(a, 0)); x /= a,
37    ↪ y /= a; }
38
39    bool operator<(const pt &p) const {
40        if (eq(x, p.x)) return gt(p.y, y);
41        return x < p.x;
42    }
43
44    bool operator==(const pt &p) const { return eq(x, p.x) &&
45    ↪ eq(y, p.y); }
46    bool operator!=(const pt &p) const { return !(*this == p);
47    ↪ }
48
49    bool right() const { return pt{0, 0} < *this; }
50
51    pt rot() { return pt{-y, x}; }
52    ld abs() const { return hypotl(x, y); }
53    ld abs2() const { return x * x + y * y; }
54};
55
56istream &operator>>(istream &in, pt &p) { return in >> p.x >>
57    ↪ p.y; }
58ostream &operator<<(ostream &out, const pt &p) { return out <<
59    ↪ p.x << ' ' << p.y; }
60
61//WARNING! do not forget to normalize vector (a,b)
62struct line {
63    ld a, b, c;
64    int id;
65
66    line(pt p1, pt p2) {
67        gassert(p1 != p2);
68        pt n = (p2 - p1).rot();
69        n /= n.abs();
70        a = n.x, b = n.y;
71        c = -(n * p1);
72    }
73
74    bool right() const {
75        return gt(a, 0) || (eq(a, 0) && gt(b, 0));
76    }
77
78    line(ld _a, ld _b, ld _c): a(_a), b(_b), c(_c) {
79        ld d = pt{a, b}.abs();
80        gassert(!eq(d, 0));
81        a /= d, b /= d, c /= d;
82    }
83
84    ld signedDist(pt p) {
85        return p * pt{a, b} + c;
86    }
87};
88
89ld pointSegmentDist(pt p, pt a, pt b) {
90    ld res = min((p - a).abs(), (p - b).abs());

```



```

80     if (a != b && ge((p - a) * (b - a), 0) && ge((p - b) * (a - b), 0))
81         res = min(res, fabs1((p - a) % (b - a)) / (b - a).abs());
82     return res;
83 }
84
85 pt linesIntersection(line l1, line l2) {
86     ld D = l1.a * l2.b - l1.b * l2.a;
87     if (eq(D, 0)) {
88         if (eq(l1.c, l2.c)) {
89             //equal lines
90         } else {
91             //no intersection
92         }
93     }
94     ld dx = -l1.c * l2.b + l1.b * l2.c;
95     ld dy = -l1.a * l2.c + l1.c * l2.a;
96     pt res{dx / D, dy / D};
97     //gassert(eq(l1.signedDist(res), 0));
98     //gassert(eq(l2.signedDist(res), 0));
99     return res;
100 }
101
102 bool pointInsideSegment(pt p, pt a, pt b) {
103     if (!eq((p - a) % (p - b), 0))
104         return false;
105     return ge(0, (a - p) * (b - p));
106 }
107
108 bool checkSegmentIntersection(pt a, pt b, pt c, pt d) {
109     if (eq((a - b) % (c - d), 0)) {
110         if (pointInsideSegment(a, c, d) ||
111             pointInsideSegment(b, c, d) ||
112             pointInsideSegment(c, a, b) ||
113             pointInsideSegment(d, a, b)) {
114             //intersection of parallel segments
115             return true;
116         }
117         return false;
118     }
119     ld s1, s2;
120     s1 = (c - a) % (b - a);
121     s2 = (d - a) % (b - a);
122     if (gt(s1, 0) && gt(s2, 0))
123         return false;
124     if (gt(0, s1) && gt(0, s2))
125         return false;
126     swap(a, c), swap(b, d);
127     s1 = (c - a) % (b - a);
128     s2 = (d - a) % (b - a);
129     if (gt(s1, 0) && gt(s2, 0))
130         return false;
131     if (gt(0, s1) && gt(0, s2))
132         return false;
133     return true;
134 }
135
136 //WARNING! run checkSegmentIntersection before and process
137 //    parallel case manually
138
139 pt segmentsIntersection(pt a, pt b, pt c, pt d) {
140     ld S = (b - a) % (d - c);
141     ld s1 = (c - a) % (d - a);
142     return a + (b - a) / S * s1;
143 }
144
145 vector<pt> circlesIntersection(pt a, ld r1, pt b, ld r2) {
146     ld d2 = (a - b).abs2();
147     ld d = (a - b).abs();
148
149     if (a == b && eq(r1, r2)) {
150         //equal circles
151     }
152     if (gt(d2, sqr(r1 + r2)) || gt(sqr(r1 - r2), d2)) {
153         //empty intersection
154         return {};
155     }
156     int num = 2;
157     if (eq(sqr(r1 + r2), d2) || eq(sqr(r1 - r2), d2))
158         num = 1;
159     ld cosa = (sqr(r1) + d2 - sqr(r2)) / ld(2 * r1 * d);
160     ld oh = cosa * r1;
161     pt h = a + ((b - a) / d * oh);
162     if (num == 1)
163         return {h};
164     ld hp = sqrt1(max(0.L, 1 - cosa * cosa)) * r1;
165     pt w = ((b - a) / d * hp).rot();
166     return {h + w, h - w};
167 }
168
169 vector<pt> circleTangents(pt a, ld r, pt p) {
170     ld d2 = (a - p).abs2();
171     ld d = (a - p).abs();
172     if (gt(sqr(r), d2)) {
173         //no tangents
174         return {};
175     }
176     if (eq(sqr(r), d2)) {
177         //point lies on circle - one tangent
178         return {p};
179     }
180     pt B = p - a;
181     pt H = B * sqr(r) / d2;
182     ld h = sqrt1(d2 - sqr(r)) * ld(r) / d;
183     pt w = (B / d * h).rot();
184     H = H + a;
185     return {H + w, H - w};
186 }
187
188 vector<pt> lineCircleIntersection(line l, pt a, ld r) {
189     ld d = l.signedDist(a);
190     if (gt(fabs1(d), r))
191         return {};
192     pt h = a - pt{l.a, l.b} * d;
193     if (eq(fabs1(d), r))
194         return {h};
195     pt w = pt{l.a, l.b}.rot() * sqrt1(max<ld>(0, sqr(r) -
196         sqr(d)));
197     return {h + w, h - w};
198 }
199
200 //modified magic from e-maxx
201 vector<line> commonTangents(pt a, ld r1, pt b, ld r2) {
202     if (a == b && eq(r1, r2)) {
203         //equal circles
204         return {};
205     }
206     vector<line> res;
207     pt c = b - a;
208     ld z = c.abs2();
209     for (int i = -1; i <= 1; i += 2)
210         for (int j = -1; j <= 1; j += 2) {
211             ld r = r2 * j - r1 * i;
212             ld d = z - sqr(r);
213             if (gt(0, d))
214                 continue;
215             d = sqrt1(max<ld>(0, d));
216             pt magic = pt{r, d} / z;
217             line l(magic * c, magic % c, r1 * i);
218             l.c -= pt{l.a, l.b} * a;
219             res.push_back(l);
220         }
221     return res;
222 }

```


11 geometry/shortest-bridge/check.cpp

```

1#include "testlib.h"
2#include <cmath>
3
4using namespace std;
5
6const double EPS = 1E-8;
7
8int main(int argc, char * argv[])
9{
10    setName("compare two sequences of doubles, max absolute or
    ↪ relative error = %.7lf", EPS);
11    registerTestlibCmd(argc, argv);
12
13    int n = 0;
14    double j, p;
15
16    while (!ans.seekEof())
17    {
18        n++;
19        j = ans.readDouble();
20        p = ouf.readDouble();
21
22        if (isnan(p))
23            quitf(_wa, "%d%s number: NAN
    ↪ found", n, englishEnding(n).c_str());
24        if (isnan(j))
25            quitf(_fail, "%d%s number: NAN
    ↪ found", n, englishEnding(n).c_str());
26        if (isinf(p))
27            quitf(_wa, "%d%s number: INFINITY
    ↪ found", n, englishEnding(n).c_str());
28        if (isinf(j))
29            quitf(_fail, "%d%s number: INFINITY
    ↪ found", n, englishEnding(n).c_str());
30
31        if (!doubleCompare(j, p, EPS))
32        {
33            quitf(_wa, "%d%s numbers differ - expected:
    ↪ '%.7lf', found: '%.7lf', error = '%.7lf'",
34                n, englishEnding(n).c_str(), j, p,
    ↪ doubleDelta(j, p));
35        }
36    }
37
38    if (n == 1)
39        quitf(_ok, "found '%.7lf', expected '%.7lf', error
    ↪ '%.7lf'", p, j, doubleDelta(j, p));
40
41    quitf(_ok, "%d numbers", n);
42}

```

12 geometry/shortest-bridge/i.cpp

```

1#include <cstdio>
2#include <algorithm>
3#include <vector>
4#include <queue>
5#include <stack>
6#include <set>
7#include <map>
8#include <cmath>
9#include <cstdlib>
10#include <string>
11#include <cstring>
12using namespace std;
13const long double EPS = 1e-13;
14const long double INF = 1e+10;
15const long double PI = acos(-1);
16int
17sig(long double r)
18{
19    return (r < -EPS) ? -1 : (r > EPS) ? +1 : 0;
20}
21
22long double
23ABS(long double a)
24{
25    return max(a, -a);
26}
27
28struct Pt
29{
30    long double x, y;
31    Pt()
32    {
33    }
34    Pt(long double x, long double y):x(x), y(y)
35    {
36    }
37    Pt operator+ (const Pt & a) const
38    {
39        return Pt(x + a.x, y + a.y);
40    }
41    Pt operator- (const Pt & a) const
42    {
43        return Pt(x - a.x, y - a.y);
44    }
45    Pt operator* (const Pt & a) const
46    {
47        return Pt(x * a.x - y * a.y, x * a.y + y * a.x);
48    }
49    Pt operator/ (const Pt & a) const
50    {
51        return Pt(-x, -y);
52    }
53    Pt operator* (const long double &k) const
54    {
55        return Pt(x * k, y * k);
56    }
57    Pt operator/ (const long double &k) const
58    {
59        return Pt(x / k, y / k);
60    }
61    long double ABS() const
62    {
63        return sqrt(x * x + y * y);
64    }
65    long double arg() const
66    {
67        return atan2(y, x);
68    }
69    long double dot(const Pt & a) const
70    {
71        return x * a.x + y * a.y;
72    }
73    long double det(const Pt & a) const
74    {
75        return x * a.y - y * a.x;
76    }
77    bool operator==(const Pt & a) const
78    {
79        return (a - *this).ABS() < EPS;
80    }
81};
82long double
83tri(const Pt & a, const Pt & b, const Pt & c)
84{
85    return (b - a).det(c - a);
86}
87
88int
89ISP(Pt a, Pt b, Pt c)
90{

```

```

91 int s = sig ((b - a).det (c - a));
92 if (s)
93     return s;
94 if (sig ((b - a).dot (c - a)) < 0)
95     return -2;
96 if (sig ((a - b).dot (c - b)) < 0)
97     return 2;
98 return 0;
99}
100
101int
102iLL (Pt a, Pt b, Pt c, Pt d)
103{
104    if (sig ((b - a).det (d - c)))
105        return 1;
106    if (sig ((b - a).det (c - a)))
107        return 0;
108    return -1;
109}
110
111Pt
112pLL (Pt a, Pt b, Pt c, Pt d)
113{
114    b = b - a;
115    d = d - c;
116    return a + b * (c - a).det (d) / b.det (d);
117}
118
119Pt
120hLP (Pt a, Pt b, Pt c)
121{
122    return pLL (a, b, c, c + (b - a) * Pt (0, 1));
123}
124
125bool
126iSS (Pt a, Pt b, Pt c, Pt d)
127{
128    return (iSP (a, b, c) * iSP (a, b, d) <= 0
129            && iSP (c, d, a) * iSP (c, d, b) <= 0);
130}
131
132long double
133dLP (Pt a, Pt b, Pt c)
134{
135    return ABS (tri (a, b, c)) / (b - a).ABS ();
136}
137
138long double
139dLL (Pt a, Pt b, Pt c, Pt d)
140{
141    return iLL (a, b, c, d) ? 0 : dLP (a, b, c);
142}
143
144long double
145dSP (Pt a, Pt b, Pt c)
146{
147    if (sig ((b - a).dot (c - a)) <= 0)
148        return (c - a).ABS ();
149    if (sig ((a - b).dot (c - b)) <= 0)
150        return (c - b).ABS ();
151    return ABS (tri (a, b, c)) / (b - a).ABS ();
152}
153
154long double
155dSS (Pt a, Pt b, Pt c, Pt d)
156{
157    return iSS (a, b, c, d) ? 0 : min (min (dSP (a, b, c), dSP
158    ↪ (a, b, d)),
159    ↪ (c, d, b));
160}
161
162int s_a[50], s_b[50], s_ab[50];
163
164int
165sAP (Pt a, Pt b, Pt c)
166{
167    return sig (a.det (c)) - sig (b.det (c)) - sig (a.det (b));
168}
169
170bool
171GSstrict (int n, Pt p[], Pt a, Pt b)
172{
173    int i;
174    p[n] = p[0];
175    p[n + 1] = p[1];
176    for (i = 0; i <= n; i++)
177    {
178        s_a[i] = sig (tri (p[i], p[i + 1], a));
179        s_b[i] = sig (tri (p[i], p[i + 1], b));
180        s_ab[i] = sig (tri (a, b, p[i]));
181    }
182    for (i = 0; i < n; i++)
183    {
184        if (s_a[i] * s_b[i] < 0 && s_ab[i] * s_ab[i + 1] < 0)
185            return 1;
186    }
187    for (i = 0; i < n; i++)
188    {
189        if (s_a[i] == 0 && s_b[i] > 0
190            && sig ((a - p[i]).dot (a - p[i + 1])) < 0)
191            return 1;
192        if (s_b[i] == 0 && s_a[i] > 0
193            && sig ((b - p[i]).dot (b - p[i + 1])) < 0)
194            return 1;
195    }
196    for (i = 0; i < n; i++)
197    {
198        if (s_ab[i + 1] == 0 && sig ((p[i + 1] - a).dot (p[i + 1]
199    ↪ - b)) <= 0)
200        {
201            if (!(p[i + 1] == a)
202                && sAP (p[i + 2] - p[i + 1], p[i] - p[i + 1], a -
203    ↪ p[i + 1]) > 0)
204                return 1;
205            if (!(p[i + 1] == b)
206                && sAP (p[i + 2] - p[i + 1], p[i] - p[i + 1], b -
207    ↪ p[i + 1]) > 0)
208                return 1;
209        }
210    }
211    return 0;
212}
213
214int wn;
215Pt wa[50];
216Pt W[50];
217Pt E[50];
218int ws, es;
219long double ijk[1100];
220int v[1100];
221Pt var[1100];
222long double nowbest = 999999999.9;
223long double
224calc (Pt s, Pt t, int side)
225{
226    for (int i = 0; i < 100; i++)
227    {
228        v[i] = 0;
229        ijk[i] = 999999999.9;
230    }
231    int n;
232    if (side == 0)
233        n = ws + 2;
234    else
235        n = es + 2;
236    if (side == 0)
237    {
238        for (int i = 0; i < ws; i++)
239            var[i] = W[i];
240        var[ws] = s;
241        var[ws + 1] = t;
242    }
243    else
244    {
245        for (int i = 0; i < es; i++)
246            var[i] = E[i];
247        var[es] = s;
248        var[es + 1] = t;
249    }
250    ijk[n - 2] = 0;
251    for (int i = 0; i < n; i++)
252    {
253        long double tmp = 999999999.9;
254        int at = 0;
255        for (int j = 0; j < n; j++)
256        {
257            if (v[j])
258                continue;
259            if (tmp > ijk[j])
260            {
261                tmp = ijk[j];
262                at = j;
263            }
264        }
265        v[at] = 1;
266        //printf("%d: %f\n", at, ijk[at]);
267        for (int j = 0; j < n; j++)
268        {
269            if (v[j])
270                continue;
271            if (ijk[j] < ijk[at] + (var[j] - var[at]).ABS ())
272                continue;
273            if ((var[j] - var[at]).ABS () < EPS)
274            {
275                ijk[j] = ijk[at];
276                continue;
277            }
278        }
279    }
280}

```

```

270     }
271     if (iGSstrict (wn, wa, var[j], var[at]))
272         continue;
273     //printf("ok\n");
274     ijk[j] = ijk[at] + (var[j] - var[at]).ABS ();
275 }
276 }
277 //printf("(%f, %f) -> (%f, %f):
278 //f\n", s.x, s.y, t.x, t.y, ijk[n-1]);
279 return ijk[n - 1];
280 }
281 int
282 main ()
283 {
284     long double sx, sy, tx, ty;
285     scanf ("%Lf%Lf%Lf%Lf", &sx, &sy, &tx, &ty);
286     Pt s = Pt (sx, sy);
287     Pt t = Pt (tx, ty);
288     int a, b;
289     scanf ("%d", &a);
290     for (int i = 0; i < a; i++)
291     {
292         long double x, y;
293         scanf ("%Lf%Lf", &x, &y);
294         W[i] = Pt (x, y);
295     }
296     scanf ("%d", &b);
297     for (int i = 0; i < b; i++)
298     {
299         long double x, y;
300         scanf ("%Lf%Lf", &x, &y);
301         E[i] = Pt (x, y);
302     }
303     ws = a;
304     es = b;
305     long double r1 = 999999999;
306     vector < pair < Pt, Pt > > best;
307     vector < pair < pair < Pt, Pt >, pair < Pt, Pt > > bli;
308     for (int i = 0; i < a - 1; i++)
309     for (int j = 0; j < b - 1; j++)
310     {
311         long double t = dSS (W[i], W[i + 1], E[j], E[j + 1]);
312         if (t < r1 - EPS)
313         {
314             r1 = t;
315         }
316     }
317     for (int i = 0; i < a - 1; i++)
318     for (int j = 0; j < b - 1; j++)
319     {
320         long double t = dSS (W[i], W[i + 1], E[j], E[j + 1]);
321         if (t > r1 + EPS)
322             continue;
323         if (iLL (W[i], W[i + 1], E[j], E[j + 1]) == 1
324             || dLL (W[i], W[i + 1], E[j], E[j + 1]) < r1 -
325 EPS)
326         {
327             if ((W[i] - E[j]).ABS () < r1 + EPS)
328                 best.push_back (make_pair (W[i], E[j]));
329             if ((W[i] - E[j + 1]).ABS () < r1 + EPS)
330                 best.push_back (make_pair (W[i], E[j + 1]));
331             if ((W[i + 1] - E[j]).ABS () < r1 + EPS)
332                 best.push_back (make_pair (W[i + 1], E[j]));
333             if ((W[i + 1] - E[j + 1]).ABS () < r1 + EPS)
334                 best.push_back (make_pair (W[i + 1], E[j + 1]));
335             if ((W[i] - hLP (E[j], E[j + 1], W[i])).ABS () <
336 r1 + EPS
337             && iSP (E[j], hLP (E[j], E[j + 1], W[i]), E[j
338 + 1]) == 2)
339                 best.push_back (make_pair (W[i], hLP (E[j], E[j + 1], W[i] +
340 1))));
341             if ((E[j] - hLP (W[i], W[i + 1], E[j])).ABS () <
342 r1 + EPS
343             && iSP (W[i], hLP (W[i], W[i + 1], E[j]), W[i
344 + 1]) == 2)
345                 best.push_back (make_pair (hLP (W[i], W[i + 1],
346 E[j]), E[j]));
347             if ((E[j + 1] - hLP (W[i], W[i + 1], E[j +
348 1])).ABS () < r1 + EPS
349             && iSP (W[i], hLP (W[i], W[i + 1], E[j + 1]),
350 W[i + 1]) == 2)
351                 best.push_back (make_pair

```

```

352 (hLP (W[i], W[i + 1], E[j + 1]),
353 E[j + 1]));
354 }
355 else if (iLL (W[i], W[i + 1], E[j], E[j + 1]) == 0)
356 {
357     vector < Pt > tmp;
358     tmp.push_back (hLP (W[i], W[i + 1], E[j]));
359     tmp.push_back (hLP (W[i], W[i + 1], E[j + 1]));
360     tmp.push_back (W[i]);
361     tmp.push_back (W[i + 1]);
362     for (int k = 0; k < 4; k++)
363     {
364         for (int l = 0; l < 3; l++)
365         {
366             if (tmp[l].x * 11451 + tmp[l].y * 810 >
367 tmp[l + 1].x * 11451 + tmp[l + 1].y *
368 810)
369                 swap (tmp[l], tmp[l + 1]);
370         }
371     }
372     bli.push_back (make_pair
373 (make_pair (tmp[l], tmp[l + 1]),
374 make_pair (hLP (E[j], E[j + 1],
375 tmp[l]),
376 hLP (E[j], E[j + 1],
377 tmp[l + 1]))));
378 }
379 for (int i = 0; i < b; i++)
380 wa[wn++] = E[i];
381 wa[wn++] = Pt (500, 100000000);
382 for (int i = a - 1; i >= 0; i--)
383 wa[wn++] = W[i];
384 wa[wn++] = Pt (500, -100000000);
385 long double r2 = 999999999;
386 for (int i = 0; i < best.size (); i++)
387 {
388     //if((best[i].first-best[i].second).ABS()>r1+EPS)while(1);
389     long double tmp =
390 calc (s, best[i].first, 0) + calc (t, best[i].second,
391 1);
392     r2 = min (r2, tmp);
393 }
394 for (int i = 0; i < bli.size (); i++)
395 {
396     long double left = 0;
397     long double right = 1;
398     //
399     if((bli[i].first.first*m1+bli[i].first.second*(1.0-left)-bli[i].se
400 //
401 if((bli[i].first.first*m1+bli[i].first.second*(1.0-left)-bli[i].se
402 for (int j = 0; j < 100; j++)
403 {
404     long double m1 = (left * 2 + right) / 3;
405     long double m2 = (left + right * 2) / 3;
406     long double t1, t2;
407     //
408     if((bli[i].first.first*m1+bli[i].first.second*(1.0-m1)-bli[i].se
409 //
410 if((bli[i].first.first*m2+bli[i].first.second*(1.0-m2)-bli[i].se
411 t1 =
412 calc (s,
413 bli[i].first.first * m1 +
414 bli[i].first.second * (1.0 - m1),
415 0) + calc (t,
416 bli[i].second.first * m1 +
417 bli[i].second.second * (1.0 -
418 m1), 1);
419     t2 =
420 calc (s,
421 bli[i].first.first * m2 +
422 bli[i].first.second * (1.0 - m2),
423 0) + calc (t,
424 bli[i].second.first * m2 +
425 bli[i].second.second * (1.0 -
426 m2), 1);
427     if (t1 > t2)
428         left = m1;
429     else
430         right = m2;
431     r2 = min (r2, min (t1, t2));
432 }
433 }
434 printf ("%f\n", r1, r1 + r2);
435 }

```

13 geometry/shortest-bridge/solutions/i2.cpp

```

1#include <cstdio>
2#include <algorithm>
3#include <vector>
4#include <queue>
5#include <stack>
6#include <set>
7#include <map>
8#include <cmath>
9#include <cstdlib>
10#include <string>
11#include <cstring>
12using namespace std;
13const long double EPS = 1e-13;
14const long double INF = 1e+10;
15const long double PI = acos(-1);
16int
17sig (long double r)
18{
19    return (r < -EPS) ? -1 : (r > EPS) ? +1 : 0;
20}
21
22long double
23ABS (long double a)
24{
25    return max (a, -a);
26}
27
28struct Pt
29{
30    long double x, y;
31    Pt ()
32    {
33    }
34    Pt (long double x, long double y):x (x), y (y)
35    {
36    }
37    Pt operator+ (const Pt & a) const
38    {
39        return Pt (x + a.x, y + a.y);
40    }
41    Pt operator- (const Pt & a) const
42    {
43        return Pt (x - a.x, y - a.y);
44    }
45    Pt operator* (const Pt & a) const
46    {
47        return Pt (x * a.x - y * a.y, x * a.y + y * a.x);
48    }
49    Pt operator- () const
50    {
51        return Pt (-x, -y);
52    }
53    Pt operator* (const long double &k) const
54    {
55        return Pt (x * k, y * k);
56    }
57    Pt operator/ (const long double &k) const
58    {
59        return Pt (x / k, y / k);
60    }
61    long double ABS () const
62    {
63        return sqrt (x * x + y * y);
64    }
65    long double arg () const
66    {
67        return atan2 (y, x);
68    }
69    long double dot (const Pt & a) const
70    {
71        return x * a.x + y * a.y;
72    }
73    long double det (const Pt & a) const
74    {
75        return x * a.y - y * a.x;
76    }
77    bool operator== (const Pt & a) const
78    {
79        return (a - *this).ABS () < EPS;
80    }
81};
82long double
83tri (const Pt & a, const Pt & b, const Pt & c)
84{
85    return (b - a).det (c - a);
86}
87
88int
89iSP (Pt a, Pt b, Pt c)
90{
91    int s = sig ((b - a).det (c - a));
92    if (s)
93        return s;
94    if (sig ((b - a).dot (c - a)) < 0)
95        return -2;
96    if (sig ((a - b).dot (c - b)) < 0)
97        return 2;
98    return 0;
99}
100
101int
102iLL (Pt a, Pt b, Pt c, Pt d)
103{
104    if (sig ((b - a).det (d - c)))
105        return 1;
106    if (sig ((b - a).det (c - a)))
107        return 0;
108    return -1;
109}
110
111Pt
112pLL (Pt a, Pt b, Pt c, Pt d)
113{
114    b = b - a;
115    d = d - c;
116    return a + b * (c - a).det (d) / b.det (d);
117}
118
119Pt
120hLP (Pt a, Pt b, Pt c)
121{
122    return pLL (a, b, c, c + (b - a) * Pt (0, 1));
123}
124
125bool
126iSS (Pt a, Pt b, Pt c, Pt d)
127{
128    return (iSP (a, b, c) * iSP (a, b, d) <= 0
129            && iSP (c, d, a) * iSP (c, d, b) <= 0);
130}
131
132long double
133dLP (Pt a, Pt b, Pt c)
134{
135    return ABS (tri (a, b, c)) / (b - a).ABS ();
136}
137
138long double
139dLL (Pt a, Pt b, Pt c, Pt d)
140{
141    return iLL (a, b, c, d) ? 0 : dLP (a, b, c);
142}
143
144long double
145dSP (Pt a, Pt b, Pt c)
146{
147    if (sig ((b - a).dot (c - a)) <= 0)
148        return (c - a).ABS ();
149    if (sig ((a - b).dot (c - b)) <= 0)
150        return (c - b).ABS ();
151    return ABS (tri (a, b, c)) / (b - a).ABS ();
152}
153
154long double
155dSS (Pt a, Pt b, Pt c, Pt d)
156{
157    return iSS (a, b, c, d) ? 0 : min (min (dSP (a, b, c), dSP
158        ↪ (a, b, d)),
159        min (dSP (c, d, a), dSP
160        ↪ (c, d, b)));
161}
162
163int
164s_a[50], s_b[50], s_ab[50];
165
166int
167sAP (Pt a, Pt b, Pt c)
168{
169    return sig (a.det (c)) - sig (b.det (c)) - sig (a.det (b));
170}
171
172bool
173iGSstrict (int n, Pt p[], Pt a, Pt b)
174{
175    int i;
176    p[n] = p[0];
177    p[n + 1] = p[1];
178    for (i = 0; i <= n; i++)
179    {
180        s_a[i] = sig (tri (p[i], p[i + 1], a));
181        s_b[i] = sig (tri (p[i], p[i + 1], b));
182        s_ab[i] = sig (tri (a, b, p[i]));
183    }
184}

```

```

179     }
180     for (i = 0; i < n; i++)
181     {
182         if (s_a[i] * s_b[i] < 0 && s_ab[i] * s_ab[i + 1] < 0)
183             return 1;
184     }
185     for (i = 0; i < n; i++)
186     {
187         if (s_a[i] == 0 && s_b[i] > 0
188             && sig ((a - p[i]).dot (a - p[i + 1])) < 0)
189             return 1;
190         if (s_b[i] == 0 && s_a[i] > 0
191             && sig ((b - p[i]).dot (b - p[i + 1])) < 0)
192             return 1;
193     }
194     for (i = 0; i < n; i++)
195         if (s_ab[i + 1] == 0 && sig ((p[i + 1] - a).dot (p[i + 1]
196         ↪ - b)) <= 0)
197         {
198             if (!(p[i + 1] == a)
199                 && sAP (p[i + 2] - p[i + 1], p[i] - p[i + 1], a -
200             ↪ p[i + 1]) > 0)
201                 return 1;
202             if (!(p[i + 1] == b)
203                 && sAP (p[i + 2] - p[i + 1], p[i] - p[i + 1], b -
204             ↪ p[i + 1]) > 0)
205                 return 1;
206         }
207     }
208     return 0;
209 }
210
211 int wn;
212 Pt wa[50];
213 Pt W[50];
214 Pt E[50];
215 int ws, es;
216 long double ijk[1100];
217 int v[1100];
218 Pt var[1100];
219 long double nowbest = 999999999.9;
220 long double
221 calc (Pt s, Pt t, int side)
222 {
223     for (int i = 0; i < 100; i++)
224     {
225         v[i] = 0;
226         ijk[i] = 999999999.9;
227     }
228     int n;
229     if (side == 0)
230         n = ws + 2;
231     else
232         n = es + 2;
233     if (side == 0)
234     {
235         for (int i = 0; i < ws; i++)
236             var[i] = W[i];
237         var[ws] = s;
238         var[ws + 1] = t;
239     }
240     else
241     {
242         for (int i = 0; i < es; i++)
243             var[i] = E[i];
244         var[es] = s;
245         var[es + 1] = t;
246     }
247     ijk[n - 2] = 0;
248     for (int i = 0; i < n; i++)
249     {
250         long double tmp = 999999999.9;
251         int at = 0;
252         for (int j = 0; j < n; j++)
253         {
254             if (v[j])
255                 continue;
256             if (tmp > ijk[j])
257             {
258                 tmp = ijk[j];
259                 at = j;
260             }
261         }
262         v[at] = 1;
263         //printf("%d: %f\n", at, ijk[at]);
264         for (int j = 0; j < n; j++)
265         {
266             if (v[j])
267                 continue;
268             if (ijk[j] < ijk[at] + (var[j] - var[at]).ABS ())
269                 continue;
270             if ((var[j] - var[at]).ABS () < EPS)
271                 {
272                     ijk[j] = ijk[at];
273                     continue;
274                 }
275             if (iGSstrict (wn, wa, var[j], var[at]))
276                 continue;
277             //printf("ok\n");
278             ijk[j] = ijk[at] + (var[j] - var[at]).ABS ();
279         }
280     }
281     //printf("%f, %f) -> (%f, %f):
282     ↪ %f\n", s.x, s.y, t.x, t.y, ijk[n - 1]);
283     return ijk[n - 1];
284 }
285
286 int
287 main ()
288 {
289     long double sx, sy, tx, ty;
290     scanf ("%Lf%Lf%Lf%Lf", &sx, &sy, &tx, &ty);
291     Pt s = Pt (sx, sy);
292     Pt t = Pt (tx, ty);
293     int a, b;
294     scanf ("%d", &a);
295     for (int i = 0; i < a; i++)
296     {
297         long double x, y;
298         scanf ("%Lf%Lf", &x, &y);
299         W[i] = Pt (x, y);
300     }
301     scanf ("%d", &b);
302     for (int i = 0; i < b; i++)
303     {
304         long double x, y;
305         scanf ("%Lf%Lf", &x, &y);
306         E[i] = Pt (x, y);
307     }
308     ws = a;
309     es = b;
310     long double r1 = 999999999.9;
311     vector < pair < Pt, Pt > > best;
312     vector < pair < pair < Pt, Pt >, pair < Pt, Pt > > > bli;
313     for (int i = 0; i < a - 1; i++)
314     for (int j = 0; j < b - 1; j++)
315     {
316         long double t = dSS (W[i], W[i + 1], E[j], E[j + 1]);
317         if (t < r1 - EPS)
318         {
319             r1 = t;
320         }
321     }
322     for (int i = 0; i < a - 1; i++)
323     for (int j = 0; j < b - 1; j++)
324     {
325         long double t = dSS (W[i], W[i + 1], E[j], E[j + 1]);
326         if (t > r1 + EPS)
327             continue;
328         if (iLL (W[i], W[i + 1], E[j], E[j + 1]) == 1
329             || dLL (W[i], W[i + 1], E[j], E[j + 1]) < r1 -
330         ↪ EPS)
331         {
332             if ((W[i] - E[j]).ABS () < r1 + EPS)
333                 best.push_back (make_pair (W[i], E[j]));
334             if ((W[i] - E[j + 1]).ABS () < r1 + EPS)
335                 best.push_back (make_pair (W[i], E[j + 1]));
336             if ((W[i + 1] - E[j]).ABS () < r1 + EPS)
337                 best.push_back (make_pair (W[i + 1], E[j]));
338             if ((W[i + 1] - E[j + 1]).ABS () < r1 + EPS)
339                 best.push_back (make_pair (W[i + 1], E[j + 1]));
340             if ((W[i] - hLP (E[j], E[j + 1], W[i])).ABS () <
341             ↪ r1 + EPS
342             && iSP (E[j], hLP (E[j], E[j + 1], W[i]), E[j
343             ↪ + 1]) == 2)
344                 best.push_back (make_pair (W[i], hLP (E[j], E[j
345             ↪ + 1], W[i])));
346             if ((W[i + 1] - hLP (E[j], E[j + 1], W[i + 1])).ABS () <
347             ↪ r1 + EPS
348             && iSP (E[j], hLP (E[j], E[j + 1], W[i + 1]),
349             ↪ E[j + 1]) == 2)
350                 best.
351                 push_back (make_pair
352                     (W[i + 1], hLP (E[j], E[j + 1], W[i
353             ↪ + 1])));
354             if ((E[j] - hLP (W[i], W[i + 1], E[j])).ABS () <
355             ↪ r1 + EPS
356             && iSP (W[i], hLP (W[i], W[i + 1], E[j]), W[i
357             ↪ + 1]) == 2)
358                 best.push_back (make_pair (hLP (W[i], W[i + 1],
359             ↪ E[j]), E[j]));
360             if ((E[j + 1] - hLP (W[i], W[i + 1], E[j + 1])).ABS () <
361             ↪ r1 + EPS

```

```

347         && iSP (W[i], hLP (W[i], W[i + 1], E[j + 1]), 424 printf ("%12Lf %12Lf\n", r1, r1 + r2);
↪ W[i + 1]) == 2) 425}
348         best.
349         push_back (make_pair
350                     (hLP (W[i], W[i + 1], E[j + 1]),
↪ E[j + 1]));
351     }
352     else if (iLL (W[i], W[i + 1], E[j], E[j + 1]) == 0)
353     {
354         vector < Pt > tmp;
355         tmp.push_back (hLP (W[i], W[i + 1], E[j]));
356         tmp.push_back (hLP (W[i], W[i + 1], E[j + 1]));
357         tmp.push_back (W[i]);
358         tmp.push_back (W[i + 1]);
359         for (int k = 0; k < 4; k++)
360         {
361             for (int l = 0; l < 3; l++)
362             {
363                 if (tmp[l].x * 11451 + tmp[l].y * 810 >
364                     tmp[l + 1].x * 11451 + tmp[l + 1].y *
↪ 810)
365                     swap (tmp[l], tmp[l + 1]);
366             }
367         }
368         bli.
369         push_back (make_pair
370                     (make_pair (tmp[1], tmp[2]),
371                         make_pair (hLP (E[j], E[j + 1],
↪ tmp[1]),
372                             hLP (E[j], E[j + 1],
↪ tmp[2]))));
373     }
374 }
375
376 for (int i = 0; i < b; i++)
377     wa[wn++] = E[i];
378 wa[wn++] = Pt (500, 100000000);
379 for (int i = a - 1; i >= 0; i--)
380     wa[wn++] = W[i];
381 wa[wn++] = Pt (500, -100000000);
382 long double r2 = 999999999;
383 for (int i = 0; i < best.size (); i++)
384 {
385     ↪ //if((best[i].first-best[i].second).ABS()>r1+EPS)while(1);
386     long double tmp =
387         calc (s, best[i].first, 0) + calc (t, best[i].second,
↪ 1);
388     r2 = min (r2, tmp);
389 }
390 for (int i = 0; i < bli.size (); i++)
391 {
392     long double left = 0;
393     long double right = 1;
394     ↪ //
395     ↪ if((bli[i].first.first*m1+bli[i].first.second*(1.0-left)-bli[i].second.first*m1+bli[i].second.second*(1.0-left)).ABS()>r1+EPS)con
396     ↪ //
397     ↪ if((bli[i].first.first*m1+bli[i].first.second*(1.0-left)-bli[i].second.first*m1+bli[i].second.second*(1.0-left)).ABS()>r1+EPS)con
398     {
399         long double m1 = (left * 2 + right) / 3;
400         long double m2 = (left + right * 2) / 3;
401         long double t1, t2;
402         ↪ //
403         ↪ if((bli[i].first.first*m1+bli[i].first.second*(1.0-m1)-bli[i].second.first*m1+bli[i].second.second*(1.0-m1)).ABS()>r1+EPS)t1=9999
404         ↪ //
405         ↪ if((bli[i].first.first*m2+bli[i].first.second*(1.0-m2)-bli[i].second.first*m2+bli[i].second.second*(1.0-m2)).ABS()>r1+EPS)t2=9999
406         t1 =
407         calc (s,
408             bli[i].first.first * m1 +
↪ bli[i].first.second * (1.0 - m1),
409             0) + calc (t,
410                 bli[i].second.first * m1 +
411                 bli[i].second.second * (1.0 -
↪ m1), 1);
412         t2 =
413         calc (s,
414             bli[i].first.first * m2 +
↪ bli[i].first.second * (1.0 - m2),
415             0) + calc (t,
416                 bli[i].second.first * m2 +
417                 bli[i].second.second * (1.0 -
↪ m2), 1);
418         if (t1 > t2)
419             left = m1;
420         else
421             right = m2;
422         r2 = min (r2, min (t1, t2));
423     }

```


14 geometry/svg.cpp

```

1 struct SVG {
2     FILE *out;
3     ld sc = 50;
4
5     void open() {
6         out = fopen("image.svg", "w");
7         fprintf(out, "<svg xmlns='http://www.w3.org/2000/svg'
→   viewBox='-1000 -1000 2000 2000'>\n");
8     }
9
10    void line(pt a, pt b) {
11        a = a * sc, b = b * sc;
12        fprintf(out, "<line x1='%Lf' y1='%Lf' x2='%Lf'
→   y2='%Lf' stroke='black'/>\n", a.x, -a.y, b.x, -b.y);
13    }
14
15    void circle(pt a, ld r = -1, string col = "red") {
16        r = (r == -1 ? 10 : sc * r);
17        a = a * sc;
18        fprintf(out, "<circle cx='%Lf' cy='%Lf' r='%Lf'
→   fill='%s'/>\n", a.x, -a.y, r, col.c_str());
19    }
20
21    void text(pt a, string s) {
22        a = a * sc;
23        fprintf(out, "<text x='%Lf' y='%Lf'
→   font-size='10px'>%s</text>\n", a.x, -a.y, s.c_str());
24    }
25
26    void close() {
27        fprintf(out, "</svg>\n");
28        fclose(out);
29    }
30
31    ~SVG() {
32        if (out)
33            close();
34    }
35} svg;

```

15 graphs/2sat.cpp

```

1 const int maxn = 200100; //2 * number of variables
2
3 namespace TwoSAT {
4     int n; //number of variables
5     bool used[maxn];
6     vector<int> g[maxn];
7     vector<int> gr[maxn];
8     int comp[maxn];
9     int res[maxn];
10
11    void addEdge(int u, int v) { //u or v
12        g[u].push_back(v ^ 1);
13        g[v].push_back(u ^ 1);
14        gr[u ^ 1].push_back(v);
15        gr[v ^ 1].push_back(u);
16    }
17
18    vector<int> ord;
19    void dfs1(int u) {
20        used[u] = true;
21        for (int v: g[u]) {
22            if (used[v])
23                continue;
24            dfs1(v);
25        }
26        ord.push_back(u);
27    }
28
29    int COL = 0;
30    void dfs2(int u) {
31        used[u] = true;
32        comp[u] = COL;
33        for (int v: gr[u]) {
34            if (used[v])
35                continue;
36            dfs2(v);
37        }
38    }
39
40    void mark(int u) {
41        res[u / 2] = u % 2;
42        used[u] = true;
43        for (int v: g[u]) {
44            if (used[v])
45                continue;
46            mark(v);
47        }
48    }
49
50    bool run() {
51        fill(res, res + 2 * n, -1);
52        fill(used, used + 2 * n, false);
53        for (i, 2 * n)
54            if (!used[i])
55                dfs1(i);
56        reverse(ord.begin(), ord.end());
57        assert((int) ord.size() == (2 * n));
58        fill(used, used + 2 * n, false);
59        for (int u: ord) if (!used[u]) {
60            dfs2(u);
61            ++COL;
62        }
63        for (i, n)
64            if (comp[i * 2] == comp[i * 2 + 1])
65                return false;
66
67        reverse(ord.begin(), ord.end());
68        fill(used, used + 2 * n, false);
69        for (int u: ord) {
70            if (res[u / 2] != -1) {
71                continue;
72            }
73            mark(u);
74        }
75        return true;
76    }
77};
78
79 int main() {
80     TwoSAT::n = 2;
81     TwoSAT::addEdge(0, 2); //x or y
82     TwoSAT::addEdge(0, 3); //x or !y
83     TwoSAT::addEdge(3, 3); //!y or !y
84     assert(TwoSAT::run());
85     cout << TwoSAT::res[0] << ' ' << TwoSAT::res[1] << '\n';
→   //1 0
86}

```


16 graphs/directed_mst.cpp

```

1 // WARNING: this code wasn't submitted anywhere
2
3 namespace TwoChinese {
4
5 struct Edge {
6     int to, w, id;
7     bool operator<(const Edge& other) const {
8         return to < other.to || (to == other.to && w <
9             other.w);
10 };
11 typedef vector<vector<Edge>> Graph;
12
13 const int maxn = 2050;
14
15 // global, for supplementary algorithms
16 int b[maxn];
17 int tin[maxn], tup[maxn];
18 int dtime; // counter for tin, tout
19 vector<int> st;
20 int nc; // number of strongly connected components
21 int q[maxn];
22
23 int answer;
24
25 void tarjan(int v, const Graph& e, vector<int>& comp) {
26     b[v] = 1;
27     st.push_back(v);
28     tin[v] = tup[v] = dtime++;
29
30     for (Edge t: e[v]) if (t.w == 0) {
31         int to = t.to;
32         if (b[to] == 0) {
33             tarjan(to, e, comp);
34             tup[v] = min(tup[v], tup[to]);
35         } else if (b[to] == 1) {
36             tup[v] = min(tup[v], tin[to]);
37         }
38     }
39
40     if (tin[v] == tup[v]) {
41         while (true) {
42             int t = st.back();
43             st.pop_back();
44             comp[t] = nc;
45             b[t] = 2;
46             if (t == v) break;
47         }
48         ++nc;
49     }
50 }
51
52 vector<Edge> bfs(
53     const Graph& e, const vector<int>& init, const
54     vector<int>& comp)
55 {
56     int n = e.size();
57     for (i, n) b[i] = 0;
58     int lq = 0, rq = 0;
59     for (int v: init) b[v] = 1, q[rq++] = v;
60
61     vector<Edge> result;
62
63     while (lq != rq) {
64         int v = q[lq++];
65         for (Edge t: e[v]) if (t.w == 0) {
66             int to = t.to;
67             if (b[to]) continue;
68             if (!comp.empty() && comp[v] != comp[to])
69                 continue;
70             b[to] = 1;
71             q[rq++] = to;
72             result.push_back(t);
73         }
74     }
75
76     return result;
77 }
78
79 // warning: check that each vertex is reachable from root
80 vector<Edge> run(Graph e, int root) {
81     int n = e.size();
82
83     // find minimum incoming weight for each vertex
84     vector<int> minw(n, inf);
85     for (v, n) for (Edge t: e[v]) {
86         minw[t.to] = min(minw[t.to], t.w);
87     }
88     for (v, n) for (Edge &t: e[v]) if (t.to != root) {
89         t.w -= minw[t.to];
90     }
91
92     // check if each vertex is reachable from root by zero
93     // edges
94     vector<Edge> firstResult = bfs(e, {root}, {});
95     if ((int)firstResult.size() + 1 == n) {
96         return firstResult;
97     }
98
99     // find strongly connected components and build compressed
100    graph
101    vector<int> comp(n);
102    for (i, n) b[i] = 0;
103    nc = 0;
104    dtime = 0;
105    for (i, n) if (!b[i]) tarjan(i, e, comp);
106
107    // multiple edges may be removed here if needed
108    Graph ne(nc);
109    for (v, n) for (Edge t: e[v]) {
110        if (comp[v] != comp[t.to]) {
111            ne[comp[v]].push_back({comp[t.to], t.w, t.id});
112        }
113    }
114
115    // run recursively on compressed graph
116    vector<Edge> subres = run(ne, comp[root]);
117
118    // find incoming edge id for each component, init queue
119    // if there is an edge (u, v) between different components
120    // than v is added to queue
121    vector<int> incomingId(nc);
122    for (Edge e: subres) {
123        incomingId[e.to] = e.id;
124    }
125
126    vector<Edge> result;
127    vector<int> init;
128    init.push_back(root);
129    for (v, n) for (Edge t: e[v]) {
130        if (incomingId[comp[t.to]] == t.id) {
131            result.push_back(t);
132            init.push_back(t.to);
133        }
134    }
135
136    // run bfs to add edges inside components and return
137    answer
138    vector<Edge> innerEdges = bfs(e, init, comp);
139    result.insert(result.end(), all(innerEdges));
140
141    assert((int)result.size() + 1 == n);
142    return result;
143 }
144
145 // namespace TwoChinese
146
147 void test () {
148     auto res = TwoChinese::run({
149         {{1,5,0},{2,5,1}},
150         {{3,1,2}},
151         {{1,2,3},{4,1,4}},
152         {{1,1,5},{4,2,6}},
153         {{2,1,7}}},
154         0);
155     cout << TwoChinese::answer << endl;
156     for (auto e: res) cout << e.id << " ";
157     cout << endl;
158     // 9    0 6 2 7
159 }

```

17 graphs/euler_cycle.cpp

```

1#include <bits/stdc++.h>
2using namespace std;
3
4const int maxn = 100100;
5const int maxm = 100100;
6
7struct Edge {
8    int to, id;
9};
10
11bool usedEdge[maxn];
12vector<Edge> g[maxn];
13int ptr[maxn];
14
15vector<int> cycle;
16void eulerCycle(int u) {
17    while (ptr[u] < (int) g[u].size() &&
18        ↪ usedEdge[g[u][ptr[u]].id])
19        ++ptr[u];
20    if (ptr[u] == (int) g[u].size())
21        return;
22    const Edge &e = g[u][ptr[u]];
23    usedEdge[e.id] = true;
24    eulerCycle(e.to);
25    cycle.push_back(e.id);
26    eulerCycle(u);
27}
28int edges = 0;
29void addEdge(int u, int v) {
30    g[u].push_back(Edge{v, edges});
31    g[v].push_back(Edge{u, edges++});
32}
33
34int main() {
35}

```

18 math/fft_recursive.cpp

```

1const int sz = 1<<20;
2
3int revb[sz];
4vector<base> ang[21];
5
6void init(int n) {
7    int lg = 0;
8    while ((1<<lg) != n) {
9        ++lg;
10    }
11    forn(i, n) {
12        revb[i] = (revb[i>>1]>>1)^((i&1)<<(lg-1));
13    }
14
15    ld e = M_PI * 2 / n;
16    ang[lg].resize(n);
17    forn(i, n) {
18        ang[lg][i] = { cos(e * i), sin(e * i) };
19    }
20
21    for (int k = lg - 1; k >= 0; --k) {
22        ang[k].resize(1 << k);
23        forn(i, 1<<k) {
24            ang[k][i] = ang[k+1][i*2];
25        }
26    }
27}
28
29void fft_rec(base *a, int lg, bool rev) {
30    if (lg == 0) {
31        return;
32    }
33    int len = 1 << (lg - 1);
34    fft_rec(a, lg-1, rev);
35    fft_rec(a+len, lg-1, rev);
36
37    forn(i, len) {
38        base w = ang[lg][i];
39        if (rev) w.im *= -1;
40        base u = a[i];
41        base v = a[i+len] * w;
42        a[i] = u + v;
43        a[i+len] = u - v;
44    }
45}
46
47void fft(base *a, int n, bool rev) {
48    forn(i, n) {
49        int j = revb[i];
50        if (i < j) swap(a[i], a[j]);
51    }
52    int lg = 0;
53    while ((1<<lg) != n) {
54        ++lg;
55    }
56    fft_rec(a, lg, rev);
57    if (rev) forn(i, n) {
58        a[i] = a[i] * (1.0 / n);
59    }
60}
61
62const int maxn = 1050000;
63
64int n;
65base a[maxn];
66base b[maxn];
67
68void test() {
69    int n = 8;
70    init(n);
71    base a[8] = {1,3,5,2,4,6,7,1};
72    fft(a, n, 0);
73    forn(i, n) cout << a[i].re << " "; cout << endl;
74    forn(i, n) cout << a[i].im << " "; cout << endl;
75    // 29 -5.82843 -7 -0.171573 5 -0.171573 -7 -5.82843
76    // 0 -3.41421 6 0.585786 0 -0.585786 -6 3.41421
77}

```

19 math/golden_search.cpp

```

1ld f(ld x) {
2    return 5 * x * x + 100 * x + 1; // -10 is minimum
3}
4
5ld goldenSearch(ld l, ld r) {
6    ld phi = (1 + sqrt(5)) / 2;
7    ld resphi = 2 - phi;
8    ld x1 = l + resphi * (r - l);
9    ld x2 = r - resphi * (r - l);
10   ld f1 = f(x1);
11   ld f2 = f(x2);
12   forn (iter, 60) {
13       if (f1 < f2) {
14           r = x2;
15           x2 = x1;
16           f2 = f1;
17           x1 = l + resphi * (r - l);
18           f1 = f(x1);
19       } else {
20           l = x1;
21           x1 = x2;
22           f1 = f2;
23           x2 = r - resphi * (r - l);
24           f2 = f(x2);
25       }
26   }
27   return (x1 + x2) / 2;
28}
29
30int main() {
31    std::cout << goldenSearch(-100, 100) << '\n';
32}

```

20 math/numbers.txt

Simpson's numerical integration:
integral from a to b $f(x) dx =$
 $(b - a) / 6 * (f(a) + 4 * f((a + b) / 2) + f(b))$

Gauss 5-th order numerical integration:
integral from -1 to 1
 $x_1, x_3 = \pm \sqrt{0.6}$, $x_2 = 0$
 $a_1, a_3 = 5/9$, $a_2 = 8/9$

large primes: $10^{18} + 3$, $+31$, $+3111$

fft modules for 2^{**20} :
7340033 13631489 26214401 28311553 70254593
976224257 (largest less than 10^{**9})

fibonacci numbers:
1, 2: 1
45: 1134903170
46: 1836311903 (max int)
47: 2971215073 (max unsigned)
91: 4660046610375530309
92: 7540113804746346429 (max i64)
93: 12200160415121876738 (max unsigned i64)

$2^{**31} = 2147483648 = 2.1e9$
 $2^{**32} = 4294967296 = 4.2e9$
 $2^{**63} = 9223372036854775808 = 9.2e18$
 $2^{**64} = 18446744073709551616 = 1.8e19$

highly composite: todo

21 strings/automaton.cpp

```

1 int t[maxn][26], lnk[maxn], len[maxn];
2 int sz;
3 int last;
4
5 void init() {
6     sz = 3;
7     last = 1;
8     forn(i, 26) t[2][i] = 1;
9     len[2] = -1;
10    lnk[1] = 2;
11}
12
13 void addchar(int c) {
14     int nlast = sz++;
15     len[nlast] = len[last] + 1;
16     int p = last;
17     for (; !t[p][c]; p = lnk[p]) {
18         t[p][c] = nlast;
19     }
20     int q = t[p][c];
21     if (len[p] + 1 == len[q]) {
22         lnk[nlast] = q;
23     } else {
24         int clone = sz++;
25         len[clone] = len[p] + 1;
26         lnk[clone] = lnk[q];
27         lnk[q] = lnk[nlast] = clone;
28         forn(i, 26) t[clone][i] = t[q][i];
29         for (; t[p][c] == q; p = lnk[p]) {
30             t[p][c] = clone;
31         }
32     }
33     last = nlast;
34}
35
36 bool check(const string& s) {
37     int v = 1;
38     for (int c: s) {
39         c -= 'a';
40         if (!t[v][c]) return false;
41         v = t[v][c];
42     }
43     return true;
44}
45
46 int main() {
47     string s;
48     cin >> s;
49     init();
50     for (int i: s) {
51         addchar(i - 'a');
52     }
53     forn(i, s.length()) {
54         assert(check(s.substr(i)));
55     }
56     cout << sz << endl;
57     return 0;
58}

```

22 strings/eertree.cpp

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int maxn = 5000100;
4 const int inf = 1e9 + 1e5;
5
6 char buf[maxn];
7 char *s = buf + 1;
8 int to[maxn][2];
9 int suff[maxn];
10 int len[maxn];
11 int sz;
12 int last;
13
14 const int odd = 1;
15 const int even = 2;
16 const int blank = 3;
17
18 inline void go(int &u, int pos) {
19     while (u != blank && s[pos - len[u] - 1] != s[pos])
20         u = suff[u];
21 }
22
23 void add_char(int pos) {
24     go(last, pos);
25     int u = suff[last];
26     go(u, pos);
27     int c = s[pos] - 'a';
28     if (!to[last][c]) {
29         to[last][c] = sz++;
30         len[sz - 1] = len[last] + 2;
31         assert(to[u][c]);
32         suff[sz - 1] = to[u][c];
33     }
34     last = to[last][c];
35}
36
37 void init() {
38     sz = 4;
39     to[blank][0] = to[blank][1] = even;
40     len[blank] = suff[blank] = inf;
41     len[even] = 0, suff[even] = odd;
42     len[odd] = -1, suff[odd] = blank;
43     last = 2;
44}
45
46 void build() {
47     init();
48     scanf("%s", s);
49     for (int i = 0; s[i]; ++i)
50         add_char(i);
51}

```

23 strings/suffix_array.cpp

```

1 string s;
2 int n;
3 int sa[maxn], new_sa[maxn], cls[maxn], new_cls[maxn],
4     cnt[maxn], lcp[maxn];
5 int n_cls;
6
7 void build() {
8     n_cls = 256;
9     forn(i, n) {
10         sa[i] = i;
11         cls[i] = s[i];
12     }
13     for (int d = 0; d < n; d = d ? d*2 : 1) {
14
15         forn(i, n) new_sa[i] = (sa[i] - d + n) % n;
16         forn(i, n_cls) cnt[i] = 0;
17         forn(i, n) ++cnt[cls[i]];
18         forn(i, n_cls) cnt[i+1] += cnt[i];
19         for (int i = n-1; i >= 0; --i)
20             sa[--cnt[cls[new_sa[i]]]] = new_sa[i];
21
22         n_cls = 0;
23         forn(i, n) {
24             if (i && (cls[sa[i]] != cls[sa[i-1]] ||
25                 ↪ cls[(sa[i] + d) % n] != cls[(sa[i-1] + d)
26                 ↪ % n])) {
27                 ++n_cls;
28                 new_cls[sa[i]] = n_cls;
29             }
30             ++n_cls;
31             forn(i, n) cls[i] = new_cls[i];
32         }
33
34         // cls is also a inv permutation of sa if a string is not
35         ↪ cyclic
36         // (i.e. a position of i-th lexicographical suffix)
37         int val = 0;
38         forn(i, n) {
39             if (val) --val;
40             if (cls[i] == n-1) continue;
41             int j = sa[cls[i] + 1];
42             while (i + val != n && j + val != n && s[i+val] ==
43             ↪ s[j+val])
44                 ++val;
45             lcp[cls[i]] = val;
46         }
47     }
48
49 int main() {
50     cin >> s;
51     s += '$';
52     n = s.length();
53     build();
54     forn(i, n) {
55         cout << s.substr(sa[i]) << endl;
56         cout << lcp[i] << endl;
57     }
58 }

```

24 strings/ukkonen.cpp

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define sz(x) ((int) (x).size())
4 #define forn(i,n) for (int i = 0; i < int(n); ++i)
5 const int inf = int(1e9) + int(1e5);
6
7 string s;
8 const int alpha = 26;
9
10 namespace SuffixTree {
11     struct Node {
12         Node *to[alpha];
13         Node *lnk, *par;
14         int l, r;
15
16         Node(int l, int r): l(l), r(r) {
17             memset(to, 0, sizeof(to));
18             lnk = par = 0;
19         }
20     };
21
22     Node *root, *blank, *cur;
23     int pos;
24
25     void init() {
26         root = new Node(0, 0);
27         blank = new Node(0, 0);
28         forn(i, alpha)
29             blank->to[i] = root;
30         root->lnk = root->par = blank->lnk = blank->par =
31         ↪ blank;
32         cur = root;
33         pos = 0;
34     }
35
36     int at(int id) {
37         return s[id];
38     }
39
40     void goDown(int l, int r) {
41         if (l >= r)
42             return;
43         if (pos == cur->r) {
44             int c = at(l);
45             assert(cur->to[c]);
46             cur = cur->to[c];
47             pos = min(cur->r, cur->l + 1);
48             ++l;
49         } else {
50             int delta = min(r - l, cur->r - pos);
51             l += delta;
52             pos += delta;
53         }
54         goDown(l, r);
55     }
56
57     void goUp() {
58         if (pos == cur->r && cur->lnk) {
59             cur = cur->lnk;
60             pos = cur->r;
61             return;
62         }
63         int l = cur->l, r = pos;
64         cur = cur->par->lnk;
65         pos = cur->r;
66         goDown(l, r);
67     }
68
69     void setParent(Node *a, Node *b) {
70         assert(a);
71         a->par = b;
72         if (b)
73             b->to[at(a->l)] = a;
74     }
75
76     void addLeaf(int id) {
77         Node *x = new Node(id, inf);
78         setParent(x, cur);
79     }
80
81     void splitNode() {
82         assert(pos != cur->r);
83         Node *mid = new Node(cur->l, pos);
84         setParent(mid, cur->par);
85         cur->l = pos;
86         setParent(cur, mid);
87         cur = mid;
88     }
89
90     bool canGo(int c) {

```

```

90     if (pos == cur->r)
91         return cur->to[c];
92     return at(pos) == c;
93 }
94
95 void fixLink(Node *&bad, Node *newBad) {
96     if (bad)
97         bad->lnk = cur;
98     bad = newBad;
99 }
100
101 void addCharOnPos(int id) {
102     Node *bad = 0;
103     while (!canGo(at(id))) {
104         if (cur->r != pos) {
105             splitNode();
106             fixLink(bad, cur);
107             bad = cur;
108         } else {
109             fixLink(bad, 0);
110         }
111         addLeaf(id);
112         goUp();
113     }
114     fixLink(bad, 0);
115     goDown(id, id + 1);
116 }
117
118 int cnt(Node *u, int ml) {
119     if (!u)
120         return 0;
121     int res = min(ml, u->r) - u->l;
122     forn (i, alpha)
123         res += cnt(u->to[i], ml);
124     return res;
125 }
126
127 void build(int l) {
128     init();
129     forn (i, l)
130         addCharOnPos(i);
131 }
132 };
133
134 int main() {
135     cin >> s;
136     SuffixTree::build(s.size());
137 }

```

25 structures/convex_hull_trick.cpp

```

1 /*
2  WARNING!!!
3  - finds maximum of  $A*x+B$ 
4  - double check max coords for int/long long overflow
5  - set min x query in put function
6  - add lines with non-descending A coefficient
7 */
8 struct FastHull {
9     int a[maxn];
10    ll b[maxn];
11    ll p[maxn];
12    int c;
13
14    FastHull(): c(0) {}
15
16    ll get(int x) {
17        if (c == 0)
18            return -infl;
19        int pos = upper_bound(p, p + c, x) - p - 1;
20        assert(pos >= 0);
21        return (ll) a[pos] * x + b[pos];
22    }
23
24    ll divideCeil(ll p, ll q) {
25        assert(q > 0);
26        if (p >= 0)
27            return (p + q - 1) / q;
28        return -((-p) / q);
29    }
30
31    void put(int A, ll B) {
32        while (c > 0) {
33            if (a[c - 1] == A && b[c - 1] >= B)
34                return;
35            ll pt = p[c - 1];
36            if (a[c - 1] * pt + b[c - 1] < A * pt + B) {
37                --c;
38                continue;
39            }
40            ll q = A - a[c - 1];
41            ll np = divideCeil(b[c - 1] - B, q);
42            p[c] = np;
43            a[c] = A;
44            b[c] = B;
45            ++c;
46            return;
47        }
48        if (c == 0) {
49            a[c] = A, b[c] = B;
50            p[c] = -1e9; //min x query
51            ++c;
52            return;
53        }
54    }
55 };
56 };
57
58 struct SlowHull {
59     vector<pair<int, ll>> v;
60
61     void put(int a, ll b) {
62         v.emplace_back(a, b);
63     }
64
65     ll get(ll x) {
66         ll best = -infl;
67         for (auto p: v)
68             best = max(best, p.first * x + p.second);
69         return best;
70     }
71 };
72
73 int main() {
74     FastHull hull1;
75     SlowHull hull2;
76     vector<int> as;
77     forn (ii, 10000)
78         as.push_back(rand() % int(1e8));
79     sort(as.begin(), as.end());
80     forn (ii, 10000) {
81         int b = rand() % int(1e8);
82         hull1.put(as[ii], b);
83         hull2.put(as[ii], b);
84         int x = rand() % int(2e8 + 1) - int(1e8);
85         assert(hull1.get(x) == hull2.get(x));
86     }
87 }

```

26 structures/heavy_light.cpp

```

1 const int maxn = 100500;
2 const int maxd = 17;
3
4 vector<int> g[maxn];
5
6 struct Tree {
7     vector<int> t;
8     int base;
9
10    Tree(): base(0) {
11    }
12
13    Tree(int n) {
14        base = 1;
15        while (base < n)
16            base *= 2;
17        t = vector<int>(base * 2, 0);
18    }
19
20    void put(int v, int delta) {
21        assert(v < base);
22        v += base;
23        t[v] += delta;
24        while (v > 1) {
25            v /= 2;
26            t[v] = max(t[v * 2], t[v * 2 + 1]);
27        }
28    }
29
30    //Careful here: cr = 2 * maxn
31    int get(int l, int r, int v = 1, int cl = 0, int cr = 2 *
↪ maxn) {
32        cr = min(cr, base);
33        if (l <= cl && cr <= r)
34            return t[v];
35        if (r <= cl || cr <= 1)
36            return 0;
37        int cc = (cl + cr) / 2;
38        return max(get(l, r, v * 2, cl, cc), get(l, r, v * 2 +
↪ 1, cc, cr));
39    }
40};
41
42 namespace HLD {
43     int h[maxn];
44     int timer;
45     int in[maxn], out[maxn], cnt[maxn];
46     int p[maxd][maxn];
47     int vroot[maxn];
48     int vpos[maxn];
49     int ROOT;
50     Tree tree[maxn];
51
52     void dfs1(int u, int prev) {
53         p[0][u] = prev;
54         in[u] = timer++;
55         cnt[u] = 1;
56         for (int v: g[u]) {
57             if (v == prev)
58                 continue;
59             h[v] = h[u] + 1;
60             dfs1(v, u);
61             cnt[u] += cnt[v];
62         }
63         out[u] = timer;
64     }
65
66     int dfs2(int u, int prev) {
67         int to = -1;
68         for (int v: g[u]) {
69             if (v == prev)
70                 continue;
71             if (to == -1 || cnt[v] > cnt[to])
72                 to = v;
73         }
74         int len = 1;
75         for (int v: g[u]) {
76             if (v == prev)
77                 continue;
78             if (to == v) {
79                 vpos[v] = vpos[u] + 1;
80                 vroot[v] = vroot[u];
81                 len += dfs2(v, u);
82             }
83             else {
84                 vroot[v] = v;
85                 vpos[v] = 0;
86                 dfs2(v, u);
87             }
88         }

```

```

89         if (vroot[u] == u)
90             tree[u] = Tree(len);
91         return len;
92     }
93
94     void init(int n) {
95         timer = 0;
96         h[ROOT] = 0;
97         dfs1(ROOT, ROOT);
98         forn (d, maxd - 1)
99             forn (i, n)
100                 p[d + 1][i] = p[d][p[d][i]];
101         vroot[ROOT] = ROOT;
102         vpos[ROOT] = 0;
103         dfs2(ROOT, ROOT);
104         //WARNING: init all trees
105     }
106
107     bool isPrev(int u, int v) {
108         return in[u] <= in[v] && out[v] <= out[u];
109     }
110
111     int lca(int u, int v) {
112         for (int d = maxd - 1; d >= 0; --d)
113             if (!isPrev(p[d][u], v))
114                 u = p[d][u];
115             if (!isPrev(u, v))
116                 u = p[0][u];
117         return u;
118     }
119
120     //for each v: h[v] >= toh
121     int getv(int u, int toh) {
122         int res = 0;
123         while (h[u] >= toh) {
124             int rt = vroot[u];
125             int l = max(0, toh - h[rt]), r = vpos[u] + 1;
126             res = max(res, tree[rt].get(l, r));
127             if (rt == ROOT)
128                 break;
129             u = p[0][rt];
130         }
131         return res;
132     }
133
134     int get(int u, int v) {
135         int w = lca(u, v);
136         return max(getv(u, h[w]), getv(v, h[w] + 1));
137     }
138
139     void put(int u, int val) {
140         int rt = vroot[u];
141         int pos = vpos[u];
142         tree[rt].put(pos, val);
143     }
144};

```


27 structures/linkcut.cpp

```

1 namespace LinkCut {
2
3 typedef struct _node {
4     _node *l, *r, *p, *pp;
5     int size; bool rev;
6     _node();
7
8     explicit _node(nullptr_t) {
9         l = r = p = pp = this;
10        size = rev = 0;
11    }
12
13    void push() {
14        if (rev) {
15            l->rev ^= 1; r->rev ^= 1;
16            rev = 0; swap(l, r);
17        }
18    }
19
20    void update();
21}* node;
22
23 node None = new _node(nullptr);
24 node v2n[maxn];
25
26 _node::_node() {
27     l = r = p = pp = None;
28     size = 1; rev = false;
29 }
30
31 void _node::update() {
32     size = (this != None) + l->size + r->size;
33     l->p = r->p = this;
34 }
35
36 void rotate(node v) {
37     assert(v != None && v->p != None);
38     assert(!v->rev);
39     assert(!v->p->rev);
40     node u = v->p;
41     if (v == u->l)
42         u->l = v->r, v->r = u;
43     else
44         u->r = v->l, v->l = u;
45     swap(u->p, v->p);
46     swap(v->pp, u->pp);
47     if (v->p != None) {
48         assert(v->p->l == u || v->p->r == u);
49         if (v->p->r == u)
50             v->p->r = v;
51         else
52             v->p->l = v;
53     }
54     u->update();
55     v->update();
56 }
57
58 void bigRotate(node v) {
59     assert(v->p != None);
60     v->p->p->push();
61     v->p->push();
62     v->push();
63     if (v->p->p != None) {
64         if ((v->p->l == v) ^ (v->p->p->r == v->p))
65             rotate(v->p);
66         else
67             rotate(v);
68     }
69     rotate(v);
70 }
71
72 inline void splay(node v) {
73     while (v->p != None)
74         bigRotate(v);
75 }
76
77 inline void splitAfter(node v) {
78     v->push();
79     splay(v);
80     v->r->p = None;
81     v->r->pp = v;
82     v->r = None;
83     v->update();
84 }
85
86 void expose(int x) {
87     node v = v2n[x];
88     splitAfter(v);
89     while (v->pp != None) {
90         assert(v->p == None);
91         splitAfter(v->pp);
92         assert(v->pp->r == None);
93         assert(v->pp->p == None);
94         assert(!v->pp->rev);
95         v->pp->r = v;
96         v->pp->update();
97         v = v->pp;
98         v->r->pp = None;
99     }
100    assert(v->p == None);
101    splay(v2n[x]);
102 }
103
104 inline void makeRoot(int x) {
105     expose(x);
106     assert(v2n[x]->p == None);
107     assert(v2n[x]->pp == None);
108     assert(v2n[x]->r == None);
109     v2n[x]->rev ^= 1;
110 }
111
112 inline void link(int x, int y) {
113     makeRoot(x);
114     v2n[x]->pp = v2n[y];
115 }
116
117 inline void cut(int x, int y) {
118     expose(x);
119     splay(v2n[y]);
120     if (v2n[y]->pp != v2n[x]) {
121         swap(x, y);
122         expose(x);
123         splay(v2n[y]);
124         assert(v2n[y]->pp == v2n[x]);
125     }
126     v2n[y]->pp = None;
127 }
128
129 inline int get(int x, int y) {
130     if (x == y)
131         return 0;
132     makeRoot(x);
133     expose(y);
134     expose(x);
135     splay(v2n[y]);
136     if (v2n[y]->pp != v2n[x])
137         return -1;
138     return v2n[y]->size;
139 }
140
141 }

```

28 structures/ordered_set.cpp

```

1#include <ext/pb_ds/assoc_container.hpp>
2#include <ext/pb_ds/tree_policy.hpp>
3
4typedef __gnu_pbds::tree<int, __gnu_pbds::null_type,
5    std::less<int>,
6    t__gnu_pbds::rb_tree_tag,
7    __gnu_pbds::tree_order_statistics_node_update> oset;
8
9#include <iostream>
10
11int main() {
12    oset X;
13    X.insert(1);
14    X.insert(2);
15    X.insert(4);
16    X.insert(8);
17    X.insert(16);
18
19    std::cout << *X.find_by_order(1) << std::endl; // 2
20    std::cout << *X.find_by_order(2) << std::endl; // 4
21    std::cout << *X.find_by_order(4) << std::endl; // 16
22    std::cout << std::boolalpha <<
    → (end(X)==X.find_by_order(6)) << std::endl; // true
23
24    std::cout << X.order_of_key(-5) << std::endl; // 0
25    std::cout << X.order_of_key(1) << std::endl; // 0
26    std::cout << X.order_of_key(3) << std::endl; // 2
27    std::cout << X.order_of_key(4) << std::endl; // 2
28    std::cout << X.order_of_key(400) << std::endl; // 5
29}

```

29 structures/treap.cpp

```

1struct node {
2    int x, y;
3    node *l, *r;
4    node(int x) : x(x), y(rand()), l(r=NULL) {}
5};
6
7void split(node *t, node *&l, node *&r, int x) {
8    if (!t) return (void)(l=r=NULL);
9    if (x <= t->x) {
10        split(t->l, l, t->l, x), r = t;
11    } else {
12        split(t->r, t->r, r, x), l = t;
13    }
14}
15
16node *merge(node *l, node *r) {
17    if (!l) return r;
18    if (!r) return l;
19    if (l->y > r->y) {
20        l->r = merge(l->r, r);
21        return l;
22    } else {
23        r->l = merge(l, r->l);
24        return r;
25    }
26}
27
28node *insert(node *t, node *n) {
29    node *l, *r;
30    split(t, l, r, n->x);
31    return merge(l, merge(n, r));
32}
33
34node *insert(node *t, int x) {
35    return insert(t, new node(x));
36}
37
38node *fast_insert(node *t, node *n) {
39    if (!t) return n;
40    node *root = t;
41    while (true) {
42        if (n->x < t->x) {
43            if (!t->l || t->l->y < n->y) {
44                split(t->l, n->l, n->r, n->x), t->l = n;
45                break;
46            } else {
47                t = t->l;
48            }
49        } else {
50            if (!t->r || t->r->y < n->y) {
51                split(t->r, n->l, n->r, n->x), t->r = n;
52                break;
53            } else {
54                t = t->r;
55            }
56        }
57    }
58    return root;
59}
60
61node *fast_insert(node *t, int x) {
62    return fast_insert(t, new node(x));
63}
64
65int main() {
66    node *t = NULL;
67    forn(i, 1000000) {
68        int x = rand();
69        t = fast_insert(t, x);
70    }
71}

```