

Содержание

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

1 Strategy.txt

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

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             if (int ret = dfs(to, min(curf, es[id].c - es[id].f))) {
44                 es[id].f += ret;
45                 es[id^1].f -= ret;
46                 return ret;
47             }
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 }

```

3 flows/globalcut.cpp

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define forn(i,n) for (int i = 0; i < int(n); ++i)
4 const int inf = 1e9 + 1e5;
5
6 const int maxn = 505;
7 namespace 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
54 int 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
2 namespace 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        for (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        for (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        for (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        for (i, n) for (j, m)
52            assert(p[0][i] + p[1][j] <= a[i][j]);
53    }
54
55    int run() {
56        for (i, n)
57            p[0][i] = 0;
58        for (i, m + 1) {
59            p[1][i] = 0;
60            match[i] = -1;
61        }
62        for (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    int N, S, T;
12    int totalFlow;
13    ll totalCost;
14    const int maxn = 505;
15    vector<Edge> edge;
16    vector<int> g[maxn];
17
18    void addEdge(int u, int v, ll c, ll cost) {
19        g[u].push_back(edge.size());
20        edge.emplace_back(v, c, cost);
21        g[v].push_back(edge.size());
22        edge.emplace_back(u, 0, -cost);
23    }
24
25    ll dist[maxn];
26    int fromEdge[maxn];
27
28    bool inQueue[maxn];
29    bool fordBellman() {
30        for (i, N)
31            dist[i] = infc;
32        dist[S] = 0;
33        inQueue[S] = true;
34        vector<int> q;
35        q.push_back(S);
36        for (int ii = 0; ii < int(q.size()); ++ii) {
37            int u = q[ii];
38            inQueue[u] = false;
39            for (int e: g[u]) {
40                if (edge[e].f == edge[e].c)
41                    continue;
42                int v = edge[e].to;
43                ll nw = edge[e].cost + dist[u];
44                if (nw >= dist[v])
45                    continue;
46                dist[v] = nw;
47                fromEdge[v] = e;
48                if (!inQueue[v]) {
49                    inQueue[v] = true;
50                    q.push_back(v);
51                }
52            }
53        }
54        return dist[T] != infc;
55    }
56
57    ll pot[maxn];
58    bool dikstra() {
59        priority_queue<pair<ll, int>, vector<pair<ll, int>>,
60            ⇨ 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    }
96
97    }

```

```

95 bool push() {
96     //2 variants
97     //if (!fordBellman())
98     if (!dijkstra())
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
113int main() {
114    MinCost::N = 3, MinCost::S = 1, MinCost::T = 2;
115    MinCost::addEdge(1, 0, 3, 5);
116    MinCost::addEdge(0, 2, 4, 6);
117    while (MinCost::push());
118    cout << MinCost::totalFlow << ' ' << MinCost::totalCost << '\n';
119}

```

6 geometry/convex_hull.cpp

```

1 #include <bits/stdc++.h>
2 using 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
8 bool 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
15 struct Hull {
16     vector<pt> top, bot;
17
18     void append(pt p) {
19         while (bot.size() > 1 && ge((p - bot.back()) % (bot.back() -
20 ↪ *next(bot.rbegin())), 0))
21             bot.pop_back();
22         bot.push_back(p);
23         while (top.size() > 1 && ge(0, (p - top.back()) % (top.back() -
24 ↪ *next(top.rbegin()))))
25             top.pop_back();
26         top.push_back(p);
27     }
28
29     void build(vector<pt> h) {
30         sort(h.begin(), h.end());
31         h.erase(unique(h.begin(), h.end()), h.end());
32         top.clear(), bot.clear();
33         for (pt p: h)
34             append(p);
35     }
36
37     pt kth(int k) {
38         if (k < sz(bot))
39             return bot[k];
40         else
41             return top[sz(top) - (k - sz(bot)) - 2];
42     }
43
44     pt mostDistant(pt dir) {
45         if (bot.empty()) {
46             //empty hull
47             return pt{1e18, 1e18};
48         }
49         if (bot.size() == 1)
50             return bot.back();
51         dir = dir.rot();
52         int n = sz(top) + sz(bot) - 2;
53         int L = -1, R = n;
54         while (L + 1 < R) {
55             int C = (L + R) / 2;
56             pt v = kth((C + 1) % n) - kth(C);
57             if (cmpAngle(dir, v)) //finds upper bound
58                 R = C;
59             else
60                 L = C;
61         }
62         return kth(R % n);
63 }

```

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> halfplanesIntersecion(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(linesIntersecion(v[i], v[i + 1]));
68        center = center + res.back();
69    }
70    center = center / n;
71    for (auto l: lines)
72        if (lt(l.signedDist(center), 0)) {
73            //empty intersection
74            return {};
75        }
76    return res;
77}

```

8 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 fabs1(a - b) < eps; }
9bool le(ld a, ld b) { return b - a > -eps; }
10bool ge(ld a, ld b) { return a - b > -eps; }
11bool lt(ld a, ld b) { return b - a > eps; }
12bool gt(ld a, ld b) { return a - b > eps; }
13ld sqr(ld x) { return x * x; }
14
15#ifdef LOCAL
16#define gassert assert
17#else
18void gassert(bool) {}
19#endif
20
21struct pt {
22    ld x, y;
23
24    pt operator+(const pt &p) const { return pt{x + p.x, y + p.y}; }
25    pt operator-(const pt &p) const { return pt{x - p.x, y - p.y}; }
26    ld operator*(const pt &p) const { return x * p.x + y * p.y; }
27    ld operator%(const pt &p) const { return x * p.y - y * p.x; }
28
29    pt operator*(const ld &a) const { return pt{x * a, y * a}; }
30    pt operator/(const ld &a) const { gassert(!eq(a, 0)); return pt{x /
    ↪ a, y / a}; }
31    void operator*=(const ld &a) { x *= a, y *= a; }
32    void operator/=(const ld &a) { gassert(!eq(a, 0)); x /= a, y /= a;
    ↪ }
33
34    bool operator<(const pt &p) const {
35        if (eq(x, p.x)) return lt(y, p.y);
36        return x < p.x;
37    }
38
39    bool operator==(const pt &p) const { return eq(x, p.x) && eq(y,
    ↪ p.y); }
40    bool operator!=(const pt &p) const { return !(*this == p); }
41
42    bool right() const { return pt{0, 0} < *this; }
43
44    pt rot() { return pt{-y, x}; }
45    ld abs() const { return hypot1(x, y); }
46    ld abs2() const { return x * x + y * y; }
47};
48
49istream &operator>>(istream &in, pt &p) { return in >> p.x >> p.y; }
50ostream &operator<<(ostream &out, const pt &p) { return out << p.x <<
    ↪ ' ' << p.y; }
51
52//WARNING! do not forget to normalize vector (a,b)
53struct line {
54    ld a, b, c;
55    int id;
56
57    line(pt p1, pt p2) {
58        gassert(p1 != p2);
59        pt n = (p2 - p1).rot();
60        n /= n.abs();
61        a = n.x, b = n.y;
62        c = -(n * p1);
63    }
64
65    bool right() const {
66        return gt(a, 0) || (eq(a, 0) && gt(b, 0));
67    }
68
69    line(ld _a, ld _b, ld _c): a(_a), b(_b), c(_c) {
70        ld d = pt{a, b}.abs();
71        gassert(!eq(d, 0));
72        a /= d, b /= d, c /= d;
73    }
74
75    ld signedDist(pt p) {
76        return p * pt{a, b} + c;
77    }
78};
79
80ld pointSegmentDist(pt p, pt a, pt b) {
81    ld res = min((p - a).abs(), (p - b).abs());
82    if (a != b && ge((p - a) * (b - a), 0) && ge((p - b) * (a - b), 0))
83        res = min(res, fabs1((p - a) % (b - a)) / (b - a).abs());
84    return res;
85}
86
87pt linesIntersecion(line l1, line l2) {
88    ld D = l1.a * l2.b - l1.b * l2.a;
89    if (eq(D, 0)) {
90        if (eq(l1.c, l2.c)) {
91            //equal lines
92            5

```

```

92     } else {
93         //no intersection
94     }
95 }
96 ld dx = -l1.c * l2.b + l1.b * l2.c;
97 ld dy = -l1.a * l2.c + l1.c * l2.a;
98 pt res{dx / D, dy / D};
99 //gassert(eq(l1.signedDist(res), 0));
100 //gassert(eq(l2.signedDist(res), 0));
101 return res;
102}
103
104 bool pointInsideSegment(pt p, pt a, pt b) {
105     if (!eq((p - a) % (b - a), 0))
106         return false;
107     return le((a - p) * (b - p), 0);
108}
109
110 bool checkSegmentIntersection(pt a, pt b, pt c, pt d) {
111     if (eq((a - b) % (c - d), 0)) {
112         if (pointInsideSegment(a, c, d) || pointInsideSegment(b, c, d)
113             || pointInsideSegment(c, a, b) || pointInsideSegment(d, a,
114             b)) {
115             //intersection of parallel segments
116             return true;
117         }
118         return false;
119     }
120     ld s1, s2;
121     s1 = (c - a) % (b - a);
122     s2 = (d - a) % (b - a);
123     if (gt(s1, 0) && gt(s2, 0))
124         return false;
125     if (lt(s1, 0) && lt(s2, 0))
126         return false;
127     swap(a, c), swap(b, d);
128     s1 = (c - a) % (b - a);
129     s2 = (d - a) % (b - a);
130     if (gt(s1, 0) && gt(s2, 0))
131         return false;
132     if (lt(s1, 0) && lt(s2, 0))
133         return false;
134     return true;
135 }
136
137 //WARNING! run checkSegmentIntersection before and process parallel case
138 ↳ manually
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     if (a == b && eq(r1, r2)) {
149         //equal circles
150     }
151     if (lt(sqr(r1 + r2), d2) || gt(sqr(r1 - r2), d2)) {
152         //empty intersection
153         return {};
154     }
155     int num = 2;
156     if (eq(sqr(r1 + r2), d2) || eq(sqr(r1 - r2), d2))
157         num = 1;
158     ld cosa = (sqr(r1) + d2 - sqr(r2)) / ld(2 * r1 * d);
159     ld oh = cosa * r1;
160     pt h = a + ((b - a) / d * oh);
161     if (num == 1)
162         return {h};
163     ld hp = sqrtl(max(0.L, 1 - cosa * cosa)) * r1;
164     pt w = ((b - a) / d * hp).rot();
165     return {h + w, h - w};
166 }
167
168 //a is circle center, p is point
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 }
181
182 pt B = p - a;
183 pt H = B * sqr(r) / d2;
184 ld h = sqrtl(d2 - sqr(r)) * ld(r) / d;
185 pt w = (B / d * h).rot();
186 H = H + a;
187 return {H + w, H - w};
188 }
189
190 vector<pt> lineCircleIntersection(line l, pt a, ld r) {
191     ld d = l.signedDist(a);
192     if (gt(fabs1(d), r))
193         return {};
194     pt h = a - pt{l.a, l.b} * d;
195     if (eq(fabs1(d), r))
196         return {h};
197     pt w = pt{l.a, l.b}.rot() * sqrtl(max<ld>(0, sqr(r) - sqr(d)));
198     return {h + w, h - w};
199 }
200
201 //modified magic from e-mazz
202 vector<line> commonTangents(pt a, ld r1, pt b, ld r2) {
203     if (a == b && eq(r1, r2)) {
204         //equal circles
205         return {};
206     }
207     vector<line> res;
208     pt c = b - a;
209     ld z = c.abs2();
210     for (int i = -1; i <= 1; i += 2)
211         for (int j = -1; j <= 1; j += 2) {
212             ld r = r2 * j - r1 * i;
213             ld d = z - sqr(r);
214             if (lt(d, 0))
215                 continue;
216             d = sqrtl(max<ld>(0, d));
217             pt magic = pt{r, d} / z;
218             line l(magic * c, magic % c, r1 * i);
219             l.c -= pt{l.a, l.b} * a;
220             res.push_back(l);
221         }
222     return res;
223 }

```

9 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;
```

10 graphs/2sat.cpp

```
1 const int maxn = 200100; //2 x 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        forn (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        forn (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 }
```

11 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 < other.w);
9     }
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 vector<int>& comp)
54 {
55     int n = e.size();
56     forn(i, n) b[i] = 0;
57     int lq = 0, rq = 0;
58     for (int v: init) b[v] = 1, q[rq++] = v;
59
60     vector<Edge> result;
61
62     while (lq != rq) {
63         int v = q[lq++];
64         for (Edge t: e[v]) if (t.w == 0) {
65             int to = t.to;
66             if (b[to]) continue;
67             if (!comp.empty() && comp[v] != comp[to]) continue;
68             b[to] = 1;
69             q[rq++] = to;
70             result.push_back(t);
71         }
72     }
73
74     return result;
75 }
76
77 // warning: check that each vertex is reachable from root
78 vector<Edge> run(Graph e, int root) {
79     int n = e.size();
80
81     // find minimum incoming weight for each vertex
82     vector<int> minw(n, inf);
83     forn(v, n) for (Edge t: e[v]) {
84         minw[t.to] = min(minw[t.to], t.w);
85     }
86     forn(v, n) for (Edge &t: e[v]) if (t.to != root) {
87         t.w -= minw[t.to];
88     }
89     forn(i, n) if (i != root) answer += minw[i];
90
91     // check if each vertex is reachable from root by zero edges
92     vector<Edge> firstResult = bfs(e, {root}, {});
93     if ((int)firstResult.size() + 1 == n) {
94         return firstResult;
95     }
96
97     // find strongly connected components and build compressed graph
98     vector<int> comp(n);
99     forn(i, n) b[i] = 0;
100     nc = 0;
101     dtime = 0;
102     forn(i, n) if (!b[i]) tarjan(i, e, comp);
103
104     // multiple edges may be removed here if needed
105     Graph ne(nc);
106     forn(v, n) for (Edge t: e[v]) {
107         if (comp[v] != comp[t.to]) {
108             ne[comp[v]].push_back({comp[t.to], t.w, t.id});
109         }
110     }
111
112     // run recursively on compressed graph
113     vector<Edge> subres = run(ne, comp[root]);
114
115     // find incoming edge id for each component, init queue
116     // if there is an edge (u, v) between different components
117     // than v is added to queue
118     vector<int> incomingId(nc);
119     for (Edge e: subres) {
120         incomingId[e.to] = e.id;
121     }
122
123     vector<Edge> result;
124     vector<int> init;
125     init.push_back(root);
126     forn(v, n) for (Edge t: e[v]) {
127         if (incomingId[comp[t.to]] == t.id) {
128             result.push_back(t);
129             init.push_back(t.to);
130         }
131     }
132
133     // run bfs to add edges inside components and return answer
134     vector<Edge> innerEdges = bfs(e, init, comp);
135     result.insert(result.end(), all(innerEdges));
136
137     assert((int)result.size() + 1 == n);
138     return result;
139 }
140
141 // namespace TwoChinese
142
143 void test () {
144     auto res = TwoChinese::run({
145         {{1,5,0},{2,5,1}},
146         {{3,1,2}},
147         {{1,2,3},{4,1,4}},
148         {{1,1,5},{4,2,6}},
149         {{2,1,7}}},
150         0);
151     cout << TwoChinese::answer << endl;
152     for (auto e: res) cout << e.id << " ";
153     cout << endl;
154     // 9    0 6 2 7
155 }

```


12 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() && usedEdge[g[u][ptr[u]].id])
18        ++ptr[u];
19    if (ptr[u] == (int) g[u].size())
20        return;
21    const Edge &e = g[u][ptr[u]];
22    usedEdge[e.id] = true;
23    eulerCycle(e.to);
24    cycle.push_back(e.id);
25    eulerCycle(u);
26}
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}

```

13 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}

```

14 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 + sqrtl(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}

```

15 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

16 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}

```

17 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}

```

18 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) % n])) {
26                 ++n_cls;
27             }
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 cyclic
35     // (i.e. a position of i-th lexicographical suffix)
36     int val = 0;
37     forn(i, n) {
38         if (val) --val;
39         if (cls[i] == n-1) continue;
40         int j = sa[cls[i] + 1];
41         while (i + val != n && j + val != n && s[i+val] == s[j+val])
42             ++val;
43         lcp[cls[i]] = val;
44     }
45 }
46
47 int main() {
48     cin >> s;
49     s += '$';
50     n = s.length();
51     build();
52     forn(i, n) {
53         cout << s.substr(sa[i]) << endl;
54         cout << lcp[i] << endl;
55     }
56 }

```

19 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 = blank;
31         cur = root;
32         pos = 0;
33     }
34
35     int at(int id) {
36         return s[id];
37     }
38
39     void goDown(int l, int r) {
40         if (l >= r)
41             return;
42         if (pos == cur->r) {
43             int c = at(l);
44             assert(cur->to[c]);
45             cur = cur->to[c];
46             pos = min(cur->r, cur->l + 1);
47             ++l;
48         } else {
49             int delta = min(r - l, cur->r - pos);
50             l += delta;
51             pos += delta;
52         }
53         goDown(l, r);
54     }
55
56     void goUp() {
57         if (pos == cur->r && cur->lnk) {
58             cur = cur->lnk;
59             pos = cur->r;
60             return;
61         }
62         int l = cur->l, r = pos;
63         cur = cur->par->lnk;
64         pos = cur->r;
65         goDown(l, r);
66     }
67
68     void setParent(Node *a, Node *b) {
69         assert(a);
70         a->par = b;
71         if (b)
72             b->to[at(a->l)] = a;
73     }
74
75     void addLeaf(int id) {
76         Node *x = new Node(id, inf);
77         setParent(x, cur);
78     }
79
80     void splitNode() {
81         assert(pos != cur->r);
82         Node *mid = new Node(cur->l, pos);
83         setParent(mid, cur->par);
84         cur->l = pos;
85         setParent(cur, mid);
86         cur = mid;
87     }
88
89     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

```

```

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     for (i, alpha)
123         res += cnt(u->to[i], ml);
124     return res;
125 }
126
127 void build(int l) {
128     init();
129     for (i, l)
130         addCharOnPos(i);
131 }
132 };
133
134 int main() {
135     cin >> s;
136     SuffixTree::build(s.size());
137 }

```

```

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 -inf;
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 = -inf;
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     for (ii, 10000)
78         as.push_back(rand() % int(1e8));
79     sort(as.begin(), as.end());
80     for (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 }

```

21 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,
39    ↪ cr));
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; d >= 0; --d)
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 };

```

22 structures/linkcut.cpp

```

1#include <bits/stdc++.h>
2using namespace std;
3#define forn(i, n) for (int i = 0; i < int(n); ++i)
4
5const int maxn = 110000;
6
7//BEGIN CODE
8namespace LinkCut {
9
10typedef struct _node {
11    _node *l, *r, *p, *pp;
12    int size; bool rev;
13    _node();
14
15    explicit _node(nullptr_t) {
16        l = r = p = pp = this;
17        size = rev = 0;
18    }
19
20    void push() {
21        if (rev) {
22            l->rev ^= 1; r->rev ^= 1;
23            rev = 0; swap(l, r);
24        }
25    }
26
27    void update();
28}* node;
29
30node None = new _node(nullptr);
31node v2n[maxn];
32
33_node::_node() {
34    l = r = p = pp = None;
35    size = 1; rev = false;
36}
37
38void _node::update() {
39    size = (this != None) + l->size + r->size;
40    l->p = r->p = this;
41}
42
43void rotate(node v) {
44    assert(v != None && v->p != None);
45    assert(!v->rev);
46    assert(!v->p->rev);
47    node u = v->p;
48    if (v == u->l)
49        u->l = v->r, v->r = u;
50    else
51        u->r = v->l, v->l = u;
52    swap(u->p, v->p);
53    swap(v->pp, u->pp);
54    if (v->p != None) {
55        assert(v->p->l == u || v->p->r == u);
56        if (v->p->r == u)
57            v->p->r = v;
58        else
59            v->p->l = v;
60    }
61    u->update();
62    v->update();
63}
64
65void bigRotate(node v) {
66    assert(v->p != None);
67    v->p->p->push();
68    v->p->push();
69    v->push();
70    if (v->p->p != None) {
71        if ((v->p->l == v) ^ (v->p->p->r == v->p))
72            rotate(v->p);
73        else
74            rotate(v);
75    }
76    rotate(v);
77}
78
79inline void splay(node v) {
80    while (v->p != None)
81        bigRotate(v);
82}
83
84inline void splitAfter(node v) {
85    v->push();
86    splay(v);
87    v->r->p = None;
88    v->r->pp = v;
89    v->r = None;
90    v->update();
91}
92
93void expose(int x) {
94    node v = v2n[x];
95    splitAfter(v);
96
97    while (v->pp != None) {
98        assert(v->p == None);
99        splitAfter(v->pp);
100        assert(v->pp->r == None);
101        assert(v->pp->p == None);
102        assert(!v->pp->rev);
103        v->pp->r = v;
104        v->pp->update();
105        v = v->pp;
106        v->r->pp = None;
107    }
108    assert(v->p == None);
109    splay(v2n[x]);
110}
111
112inline void makeRoot(int x) {
113    expose(x);
114    assert(v2n[x]->p == None);
115    assert(v2n[x]->pp == None);
116    assert(v2n[x]->r == None);
117    v2n[x]->rev ^= 1;
118}
119
120inline void link(int x, int y) {
121    makeRoot(x);
122    v2n[x]->pp = v2n[y];
123}
124
125inline void cut(int x, int y) {
126    expose(x);
127    splay(v2n[y]);
128    if (v2n[y]->pp != v2n[x]) {
129        swap(x, y);
130        expose(x);
131        splay(v2n[y]);
132        assert(v2n[y]->pp == v2n[x]);
133    }
134    v2n[y]->pp = None;
135}
136
137inline int get(int x, int y) {
138    if (x == y)
139        return 0;
140    makeRoot(x);
141    expose(y);
142    expose(x);
143    splay(v2n[y]);
144    if (v2n[y]->pp != v2n[x])
145        return -1;
146    return v2n[y]->size;
147}
148
149//END CODE
150
151LinkCut::_node mem[maxn];
152
153int main() {
154    int n, m;
155    scanf("%d %d", &n, &m);
156
157    forn (i, n)
158        LinkCut::v2n[i] = &mem[i];
159
160    forn (i, m) {
161        int a, b;
162        if (scanf(" link %d %d", &a, &b) == 2)
163            LinkCut::link(a - 1, b - 1);
164        else if (scanf(" cut %d %d", &a, &b) == 2)
165            LinkCut::cut(a - 1, b - 1);
166        else if (scanf(" get %d %d", &a, &b) == 2)
167            printf("%d\n", LinkCut::get(a - 1, b - 1));
168        else
169            assert(false);
170    }
171    return 0;
172}

```

23 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, std::less<int>,
5    __gnu_pbds::rb_tree_tag,
6    __gnu_pbds::tree_order_statistics_node_update> oset;
7
8#include <iostream>
9
10int main() {
11    oset X;
12    X.insert(1);
13    X.insert(2);
14    X.insert(4);
15    X.insert(8);
16    X.insert(16);
17
18    std::cout << *X.find_by_order(1) << std::endl; // 2
19    std::cout << *X.find_by_order(2) << std::endl; // 4
20    std::cout << *X.find_by_order(4) << std::endl; // 16
21    std::cout << std::boolalpha << (end(X)==X.find_by_order(6)) <<
22    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}

```

24 structures/splay.cpp

```

1#include <bits/stdc++.h>
2using namespace std;
3#define form(i, n) for (int i = 0; i < (int)(n); ++i)
4
5const int maxn = 100500;
6
7struct node;
8void updson(node* p, node* v, node* was);
9
10struct node {
11    int val;
12    node *l, *r, *p;
13    node() {}
14    node(int val) : val(val), l(r=NULL) {}
15
16    bool isRoot() const { return !p; }
17    bool isRight() const { return p && p->r == this; }
18    bool isLeft() const { return p && p->l == this; }
19    void setLeft(node* t) {
20        if (t) t->p = this;
21        l = t;
22    }
23    void setRight(node *t) {
24        if (t) t->p = this;
25        r = t;
26    }
27};
28
29void updson(node *p, node *v, node *was) {
30    if (p) {
31        if (p->l == was) p->l = v;
32        else p->r = v;
33    }
34    if (v) v->p = p;
35}
36
37void rightRotate(node *v) {
38    assert(v && v->l);
39    node *u = v->l;
40    node *p = v->p;
41    v->setLeft(u->r);
42    u->setRight(v);
43    updson(p, u, v);
44}
45
46void leftRotate(node *v) {
47    assert(v && v->r);
48    node *u = v->r;
49    node *p = v->p;
50    v->setRight(u->l);
51    u->setLeft(v);
52    updson(p, u, v);
53}
54
55void splay(node *v) {
56    while (v->p) {
57        if (!v->p->p) {
58            if (v->isLeft()) rightRotate(v->p);
59            else leftRotate(v->p);
60        } else if (v->isLeft() && v->p->isLeft()) {
61            rightRotate(v->p->p);
62            rightRotate(v->p);
63        } else if (v->isRight() && v->p->isRight()) {
64            leftRotate(v->p->p);
65            leftRotate(v->p);
66        } else if (v->isLeft()) {
67            rightRotate(v->p);
68            leftRotate(v->p);
69        } else {
70            leftRotate(v->p);
71            rightRotate(v->p);
72        }
73    }
74    v->p = NULL;
75}
76
77node *insert(node *t, node *n) {
78    if (!t) return n;
79    int x = n->val;
80    while (true) {
81        if (x < t->val) {
82            if (t->l) {
83                t = t->l;
84            } else {
85                t->setLeft(n);
86                t = t->l;
87                break;
88            }
89        } else {
90            if (t->r) {
91                t = t->r;
92            } else {
93                t->setRight(n);
94                t = t->r;
95                break;
96            }
97        }
98    }
99}

```



```

96     }
97 }
98 }
99 splay(t);
100 return t;
101}
102
103node *insert(node *t, int x) {
104     return insert(t, new node(x));
105}
106
107int main() {
108     node *t = NULL;
109     forn(i, 1000000) {
110         int x = rand();
111         t = insert(t, x);
112     }
113     return 0;
114}

```

25 structures/treap.cpp

```

1 struct node {
2     int x, y;
3     node *l, *r;
4     node(int x) : x(x), y(rand()), l(r=NULL) {}
5 };
6
7 void 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
16 node *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
28 node *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
34 node *insert(node *t, int x) {
35     return insert(t, new node(x));
36 }
37
38 node *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
61 node *fast_insert(node *t, int x) {
62     return fast_insert(t, new node(x));
63 }
64
65 int main() {
66     node *t = NULL;
67     forn(i, 1000000) {
68         int x = rand();
69         t = fast_insert(t, x);
70     }
71 }

```