

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/chan.cpp	4
7	geometry/convex_hull.cpp	5
8	geometry/halfplanes.cpp	6
9	geometry/polygon.cpp	6
10	geometry/primitives.cpp	7
11	geometry/svg.cpp	8
12	graphs/2sat.cpp	8
13	graphs/directed_mst.cpp	9
14	graphs/edmonds_matching.cpp	10
15	graphs/euler_cycle.cpp	10
16	math/factor.cpp	11
17	math/fft.cpp	12
18	math/golden_search.cpp	12
19	math/numbers.tex	13
20	math/stuff.cpp	13
21	strings/automaton.cpp	14
22	strings/duval_manacher.cpp	14
23	strings/eertree.cpp	15
24	strings/suffix_array.cpp	15
25	strings/ukkonen.cpp	16
26	structures/convex_hull_trick.cpp	17
27	structures/heavy_light.cpp	17
28	structures/linkcut.cpp	18
29	structures/ordered_set.cpp	19
30	structures/treap.cpp	20
31	Сеточка	21
32	Сеточка	22

1 Strategy.txt

- 1 - Проверить руками сэмплы
- Подумать как дебагать после написания
- 2 - Выписать сложные формулы и все +-1
- 2 - Проверить имена файлов
- 3 - Прогнать сэмплы
- Переполнения int, переполнения long long
- 3 - Выход за границу массива: _GLIBCXX_DEBUG
- Переполнения по модулю: в
 - ↳ псевдо-онлайн-генераторе, в функциях-обертках
- 4 - Проверить мультитест на разных тестах
- 5 - Прогнать минимальный по каждому параметру тест
- 6 - Прогнать псевдо-максимальный тест(немного чисел,
 - ↳ но очень большие или очень маленькие)
- 6 - Представить что не зайдет и заранее написать
 - ↳ assert'ы, прогнать слегка модифицированные тесты
- 7 - cout.precision: в том числе в интерактивных
 - ↳ задачах
- 8 - Удалить debug-output, отсечения для тестов,
 - ↳ вернуть оригинальный тахп, удалить
 - ↳ _GLIBCXX_DEBUG
- 9 - Вердикт может врать
- 10 - Если много тестов(>3), дописать в конец каждого
 - ↳ теста ответ, чтобы не забыть
- 10 - (WA) Потестить не только ответ, но и содержимое
 - ↳ значимых массивов, переменных
- 11 - (WA) Изменить тест так, чтобы ответ не менялся:
 - ↳ поменять координаты местами, сжать/растянуть
 - ↳ координаты, поменять ROOT дерева
- 12 - (WA) Подвигать размер блока в корневой или
 - ↳ битсете
- 13 - (WA) Поставить assert'ы, возможно написать чекер
 - ↳ с assert'ом
- 14 - (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
- 15 - (TL) cin -> scanf -> getchar
- 15 - (TL) Упихать в кэш большие массивы, поменять
 - ↳ местами for'ы или измерения массива
- 16 - (RE) Проверить формулы на деление на 0, выход за
 - ↳ область определения(sqrt(-eps), acos(1 + eps))
- 17 - (WA) Проверить, что ответ влезает в int

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     forn(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 }
51 return 0;
52 }
53
54 i64 dinic(int S, int T) {
55     Dinic::S = S, Dinic::T = T;
56     i64 res = 0;
57     while (bfs()) {
58         forn(i, n) pos[i] = 0;
59         while (int f = dfs(S, 1e9)) {
60             assert(f <= 1000000000);
61             res += f;
62         }
63     }
64     return res;
65 }
66
67 } // namespace Dinic
68
69 void test() {
70     Dinic::n = 4;
71     Dinic::addEdge(0, 1, 1);
72     Dinic::addEdge(0, 2, 2);
73     Dinic::addEdge(2, 1, 1);
74     Dinic::addEdge(1, 3, 2);
75     Dinic::addEdge(2, 3, 1);
76     cout << Dinic::dinic(0, 3) << endl; // 3
77 }
78
79
80 /*
81 LR-поток находит не максимальный поток.
82 Добавим новый сток S' и исток T'. Заменяем ребро (u, v, l, r)
83 LR-сети на ребра (u, T', l), (S', v, l), (u, v, r - l).
84 Добавим ребро (T, S, k). Ставим значение k=inf, пускаем поток.
85 Проверяем, что все ребра из S' насыщены (иначе ответ не
86 существует). Бинариским находим наименьшее k, что величина
87 потока не изменится. Это k - величина МИНИМАЛЬНОГО потока,
88 удовлетворяющего ограничениям. */

```

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 #define all(x) (x).begin(), (x).end()
6
7 const int maxn = 505;
8 namespace StoerWagner {
9     int g[maxn][maxn];
10     int dist[maxn];
11     bool used[maxn];
12     int n;
13
14     void addEdge(int u, int v, int c) {
15         g[u][v] += c;
16         g[v][u] += c;
17     }
18
19     int run() {
20         vector<int> vertices;
21         forn(i, n)
22             vertices.push_back(i);
23         int mincut = inf;
24         while (vertices.size() > 1) {
25             int u = vertices[0];
26             for (auto v: vertices) {
27                 used[v] = false;
28                 dist[v] = g[u][v];
29             }
30             used[u] = true;
31             forn(ii, vertices.size() - 2) {
32                 for (auto v: vertices)
33                     if (!used[v])
34                         if (used[u] || dist[v] > dist[u])
35                             u = v;
36                 used[u] = true;
37                 for (auto v: vertices)
38                     if (!used[v])
39                         dist[v] += g[u][v];
40             }
41             int t = -1;
42             for (auto v: vertices)
43                 if (!used[v])
44                     t = v;
45             assert(t != -1);
46             mincut = min(mincut, dist[t]);
47             vertices.erase(find(all(vertices), t));
48             for (auto v: vertices)
49                 addEdge(u, v, g[v][t]);
50         }
51         return mincut;
52     }
53 } // namespace StoerWagner
54
55 int main() {
56     StoerWagner::n = 4;
57     StoerWagner::addEdge(0, 1, 5);
58     StoerWagner::addEdge(2, 3, 5);
59     StoerWagner::addEdge(1, 2, 4);
60     cerr << StoerWagner::run() << '\n'; // 4
61 }

```

4 flows/hungary.cpp

```

1// left half is the smaller one
2namespace Hungary {
3const int maxn = 505;
4int a[maxn][maxn];
5int p[2][maxn];
6int match[maxn];
7bool used[maxn];
8int from[maxn];
9int mind[maxn];
10int n, m;
11
12int 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
41void 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
55int 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} // namespace Hungary

```

5 flows/mincost.cpp

```

1namespace MinCost {
2const ll infc = 1e12;
3
4struct 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
12int N, S, T;
13int totalFlow;
14ll totalCost;
15const int maxn = 505;
16vector<Edge> edge;
17vector<int> g[maxn];
18
19void 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
26ll dist[maxn];
27int fromEdge[maxn];
28
29bool inQueue[maxn];
30bool fordBellman() {
31    forn (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
58ll pot[maxn];
59bool dijkstra() {
60    typedef pair<ll, int> Pair;
61    priority_queue<Pair, vector<Pair>, greater<Pair>> q;
62    forn (i, N)
63        dist[i] = infc;
64    dist[S] = 0;
65    q.emplace(dist[S], S);
66    while (!q.empty()) {
67        int u = q.top().second;
68        ll cdist = q.top().first;
69        q.pop();
70        if (cdist != dist[u])
71            continue;
72        for (int e: g[u]) {
73            int v = edge[e].to;
74            if (edge[e].c == edge[e].f)
75                continue;
76            ll w = edge[e].cost + pot[u] - pot[v];
77            assert(w >= 0);
78            ll ndist = w + dist[u];
79            if (ndist >= dist[v])
80                continue;
81            dist[v] = ndist;
82            fromEdge[v] = e;
83            q.emplace(dist[v], v);
84        }
85    }
86    if (dist[T] == infc)
87        return false;
88    forn (i, N) {
89        if (dist[i] == infc)
90            continue;
91        pot[i] += dist[i];

```

6 geometry/chan.cpp

```

92     }
93     return true;
94 }
95
96 bool push() {
97     //2 variants
98     //if (!fordBellman())
99     if (!dikstra())
100         return false;
101     ++totalFlow;
102     int u = T;
103     while (u != S) {
104         int e = fromEdge[u];
105         totalCost += edge[e].cost;
106         edge[e].f++;
107         edge[e ^ 1].f--;
108         u = edge[e ^ 1].to;
109     }
110     return true;
111 }
112
113 //min-cost-circulation
114 ll d[maxn][maxn];
115 int dfrom[maxn][maxn];
116 int level[maxn];
117 void circulation() {
118     while (true) {
119         int q = 0;
120         fill(d[0], d[0] + N, 0);
121         for (iter, N) {
122             fill(d[iter + 1], d[iter + 1] + N, infc);
123             for (u, N)
124                 for (int e: g[u]) {
125                     if (edge[e].c == edge[e].f)
126                         continue;
127                     int v = edge[e].to;
128                     ll ndist = d[iter][u] + edge[e].cost;
129                     if (ndist >= d[iter + 1][v])
130                         continue;
131                     d[iter + 1][v] = ndist;
132                     dfrom[iter + 1][v] = e;
133                 }
134             q ^= 1;
135         }
136         int w = -1;
137         ld mindmax = 1e18;
138         for (u, N) {
139             ld dmax = -1e18;
140             for (iter, N)
141                 dmax = max(dmax,
142                     (d[N][u] - d[iter][u]) / ld(N - iter));
143             if (mindmax > dmax)
144                 mindmax = dmax, w = u;
145         }
146         if (mindmax >= 0)
147             break;
148         fill(level, level + N, -1);
149         int k = N;
150         while (level[w] == -1) {
151             level[w] = k;
152             w = edge[dfrom[k-1][w] ^ 1].to;
153         }
154         int k2 = level[w];
155         ll delta = infc;
156         while (k2 > k) {
157             int e = dfrom[k2-1][w];
158             delta = min(delta, edge[e].c - edge[e].f);
159             w = edge[e ^ 1].to;
160         }
161         k2 = level[w];
162         while (k2 > k) {
163             int e = dfrom[k2-1][w];
164             totalCost += edge[e].cost * delta;
165             edge[e].f += delta;
166             edge[e ^ 1].f -= delta;
167             w = edge[e ^ 1].to;
168         }
169     }
170 }
171 // namespace MinCost
172
173 int main() {
174     MinCost::N = 3, MinCost::S = 1, MinCost::T = 2;
175     MinCost::addEdge(1, 0, 3, 5);
176     MinCost::addEdge(0, 2, 4, 6);
177     while (MinCost::push());
178     cout << MinCost::totalFlow << ' ',
179          << MinCost::totalCost << '\n'; //3 33
180 }

```

```

1 struct pt {
2     ld x, y, z;
3     ld ox, oy, oz;
4     int pr, nx;
5     bool inHull;
6
7     static pt *NIL;
8
9     pt() {}
10
11     pt(ld x, ld y, ld z): x(x), y(y), z(z) {}
12
13     void transform(bool rev) {
14         if (rev) {
15             x = ox, y = oy, z = oz;
16         } else {
17             ox = x, oy = y, oz = z;
18             x += rndEps(), y += rndEps(), z += rndEps();
19         }
20     }
21 };
22 namespace Chan {
23 int n;
24 pt p[maxn];
25
26 ld turn(int p1, int p2, int p3) {
27     assert(p1 != -1 && p2 != -1 && p3 != -1);
28     return (p[p2].x - p[p1].x) * (p[p3].y - p[p1].y) -
29         (p[p3].x - p[p1].x) * (p[p2].y - p[p1].y);
30 }
31
32 //replace y with z
33 ld turnz(int p1, int p2, int p3) {
34     assert(p1 != -1 && p2 != -1 && p3 != -1);
35     return (p[p2].x - p[p1].x) * (p[p3].z - p[p1].z) -
36         (p[p3].x - p[p1].x) * (p[p2].z - p[p1].z);
37 }
38
39 ld gett(int p1, int p2, int p3) {
40     return turnz(p1, p2, p3) / turn(p1, p2, p3);
41 }
42
43 void act(int i) {
44     if (p[i].inHull) {
45         p[p[i].nx].pr = p[i].pr;
46         p[p[i].pr].nx = p[i].nx;
47     } else {
48         p[p[i].nx].pr = p[p[i].pr].nx = i;
49     }
50     p[i].inHull ^= 1;
51 }
52
53 vector<int> buildHull(int l, int r, bool upper) {
54     if (l + 1 >= r) {
55         p[l].pr = p[l].nx = -1;
56         p[l].inHull = true;
57         return {};
58     }
59     int mid = (l + r) / 2;
60     auto L = buildHull(l, mid, upper);
61     auto R = buildHull(mid, r, upper);
62     reverse(L.begin(), L.end());
63     reverse(R.begin(), R.end());
64     int u = l, v = r - 1;
65     while (true) {
66         if (p[u].nx != -1 && ((turn(u, p[u].nx, v) > 0) ^ upper))
67             u = p[u].nx;
68         else if (p[v].pr != -1 && ((turn(u, p[v].pr, v) > 0) ^ upper))
69             v = p[v].pr;
70         else
71             break;
72     }
73
74     ld T = -1e100;
75     ld t[6];
76     vector<int> A;
77     while (true) {
78         for (i, 6)
79             t[i] = 1e100;
80         if (!L.empty()) {
81             int id = L.back();
82             t[0] = gett(p[id].pr, id, p[id].nx);
83         }
84         if (!R.empty()) {
85             int id = R.back();
86             t[1] = gett(p[id].pr, id, p[id].nx);
87         }
88         if (p[u].pr != -1)
89             t[2] = gett(p[u].pr, u, v);
90         if (p[u].nx != -1)
91             t[3] = gett(u, p[u].nx, v);

```

7 geometry/convex_hull.cpp

```

92     if (p[v].pr != -1)
93         t[4] = gett(u, p[v].pr, v);
94     if (p[v].nx != -1)
95         t[5] = gett(u, v, p[v].nx);
96     ld nt = 1e100;
97     int type = -1;
98     forn (i, 6)
99         if (gt(t[i], T) && t[i] < nt)
100             nt = t[i], type = i;
101     if (type == -1)
102         break;
103
104     if (type == 0) {
105         act(L.back());
106         if (L.back() < u)
107             A.push_back(L.back());
108         L.pop_back();
109     } else if (type == 1) {
110         act(R.back());
111         if (R.back() > v)
112             A.push_back(R.back());
113         R.pop_back();
114     } else if (type == 2) {
115         A.push_back(u);
116         u = p[u].pr;
117     } else if (type == 3) {
118         u = p[u].nx;
119         A.push_back(u);
120     } else if (type == 4) {
121         v = p[v].pr;
122         A.push_back(v);
123     } else if (type == 5) {
124         A.push_back(v);
125         v = p[v].nx;
126     } else
127         assert(false);
128     T = nt;
129 }
130 assert(L.empty() && R.empty());
131
132 p[u].nx = v, p[v].pr = u;
133 for (int i = u + 1; i < v; ++i)
134     p[i].inHull = false;
135 for (int i = int(A.size()) - 1; i >= 0; --i) {
136     int id = A[i];
137     if (id <= u || id >= v) {
138         if (u == id)
139             u = p[u].pr;
140         if (v == id)
141             v = p[v].nx;
142         act(id);
143     } else {
144         p[id].pr = u, p[id].nx = v;
145         act(id);
146         if (id >= mid)
147             v = id;
148         else
149             u = id;
150     }
151 }
152 return A;
153 }
154
155 //facets are oriented ccw if look from the outside
156 vector<Facet> getFacets() {
157     forn (i, n)
158         p[i].transform(false);
159     //WARNING: original order of points is changed
160     sort(p, p + n, [](const pt &a, const pt &b) {
161         return a.x < b.x;
162     });
163     vector<Facet> facets;
164     forn (q, 2) {
165         auto movie = buildHull(0, n, q);
166         for (auto x: movie) {
167             if (!p[x].inHull)
168                 facets.emplace_back(p[x].pr, x, p[x].nx);
169             else
170                 facets.emplace_back(p[x].pr, p[x].nx, x);
171             act(x);
172         }
173     }
174     forn (i, n)
175         p[i].transform(true);
176     return facets;
177 }
178 } //namespace Chan

```

```

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())
20            % (bot.back() - *next(bot.rbegin())), 0))
21            bot.pop_back();
22        bot.push_back(p);
23        while (top.size() > 1 && ge(0, (p - top.back())
24            % (top.back() - *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    }
64};

```

8 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(),
15        [](const line &a, const line &b) {
16            bool ar = a.right(), br = b.right();
17            if (ar ^ br)
18                return ar;
19            ld prod = (pt{a.a, a.b} % pt{b.a, b.b});
20            if (!eq(prod, 0))
21                return prod > 0;
22            return a.c < b.c;
23        });
24    vector<line> lines2;
25    pt pr;
26    forn(i, lines.size()) {
27        pt cur{lines[i].a, lines[i].b};
28        if (i == 0 || cur != pr)
29            lines2.push_back(lines[i]);
30        pr = cur;
31    }
32    lines = lines2;
33    int n = lines.size();
34    forn(i, n)
35        lines[i].id = i;
36    vector<line> hull;
37    forn(i, 2 * n) {
38        line l = lines[i % n];
39        while ((int) hull.size() >= 2) {
40            ld D = det3x3(*next(hull.rbegin()), hull.back(), l);
41            if (ge(D, 0))
42                break;
43            hull.pop_back();
44        }
45        hull.push_back(l);
46    }
47    vector<int> firstTime(n, -1);
48    vector<line> v;
49    forn(i, hull.size()) {
50        int cid = hull[i].id;
51        if (firstTime[cid] == -1) {
52            firstTime[cid] = i;
53            continue;
54        }
55        forab(j, firstTime[cid], i)
56            v.push_back(hull[j]);
57        break;
58    }
59    n = v.size();
60    if (v.empty()) {
61        //empty intersection
62        return {};
63    }
64    v.push_back(v[0]);
65    vector<pt> res;
66    pt center{0, 0};
67    forn(i, n) {
68        res.push_back(linesIntersecion(v[i], v[i + 1]));
69        center = center + res.back();
70    }
71    center = center / n;
72    for (auto l: lines)
73        if (gt(0, l.signedDist(center))) {
74            //empty intersection
75            return {};
76        }
77    return res;
78}

```

9 geometry/polygon.cpp

```

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

```

10 geometry/primitives.cpp

```

1//WARNING! do not forget to normalize vector (a,b)
2struct line {
3    ld a, b, c;
4    int id;
5
6    line(pt p1, pt p2) {
7        gassert(p1 != p2);
8        pt n = (p2 - p1).rot();
9        n /= n.abs();
10       a = n.x, b = n.y;
11       c = -(n * p1);
12    }
13
14    bool right() const {
15        return gt(a, 0) || (eq(a, 0) && gt(b, 0));
16    }
17
18    line(ld _a, ld _b, ld _c): a(_a), b(_b), c(_c) {
19        ld d = pt{a, b}.abs();
20        gassert(!eq(d, 0));
21        a /= d, b /= d, c /= d;
22    }
23
24    ld signedDist(pt p) {
25        return p * pt{a, b} + c;
26    }
27};
28
29ld pointSegmentDist(pt p, pt a, pt b) {
30    ld res = min((p - a).abs(), (p - b).abs());
31    if (a != b && ge((p - a) * (b - a), 0) &&
32        ge((p - b) * (a - b), 0))
33        res = min(res,
34            fabsl((p - a) % (b - a)) / (b - a).abs());
35    return res;
36}
37
38pt linesIntersection(line l1, line l2) {
39    ld D = l1.a * l2.b - l1.b * l2.a;
40    if (eq(D, 0)) {
41        if (eq(l1.c, l2.c)) {
42            //equal lines
43        } else {
44            //no intersection
45        }
46    }
47    ld dx = -l1.c * l2.b + l1.b * l2.c;
48    ld dy = -l1.a * l2.c + l1.c * l2.a;
49    pt res{dx / D, dy / D};
50    //gassert(eq(l1.signedDist(res), 0));
51    //gassert(eq(l2.signedDist(res), 0));
52    return res;
53}
54
55bool pointInsideSegment(pt p, pt a, pt b) {
56    if (!eq((p - a) % (b - a), 0))
57        return false;
58    return ge(0, (a - p) * (b - p));
59}
60
61bool checkSegmentIntersection(pt a, pt b, pt c, pt d) {
62    if (eq((a - b) % (c - d), 0)) {
63        if (pointInsideSegment(a, c, d) ||
64            pointInsideSegment(b, c, d) ||
65            pointInsideSegment(c, a, b) ||
66            pointInsideSegment(d, a, b)) {
67            //intersection of parallel segments
68            return true;
69        }
70        return false;
71    }
72
73    ld s1, s2;
74
75    s1 = (c - a) % (b - a);
76    s2 = (d - a) % (b - a);
77    if (gt(s1, 0) && gt(s2, 0))
78        return false;
79    if (gt(0, s1) && gt(0, s2))
80        return false;
81
82    swap(a, c), swap(b, d);
83
84    s1 = (c - a) % (b - a);
85    s2 = (d - a) % (b - a);
86    if (gt(s1, 0) && gt(s2, 0))
87        return false;
88    if (gt(0, s1) && gt(0, s2))
89        return false;
90
91    return true;
92}
93
94// WARNING! run checkSegmentIntersection before and process
95// parallel case manually
96pt segmentsIntersection(pt a, pt b, pt c, pt d) {
97    ld S = (b - a) % (d - c);
98    ld s1 = (c - a) % (d - a);
99    return a + (b - a) / S * s1;
100}
101
102vector<pt> circlesIntersection(pt a, ld r1, pt b, ld r2) {
103    ld d2 = (a - b).abs2();
104    ld d = (a - b).abs();
105
106    if (a == b && eq(r1, r2)) {
107        //equal circles
108    }
109    if (gt(d2, sqr(r1 + r2)) || gt(sqr(r1 - r2), d2)) {
110        //empty intersection
111        return {};
112    }
113    int num = 2;
114    if (eq(sqr(r1 + r2), d2) || eq(sqr(r1 - r2), d2))
115        num = 1;
116    ld cosa = (sqr(r1) + d2 - sqr(r2)) / ld(2 * r1 * d);
117    ld oh = cosa * r1;
118    pt h = a + ((b - a) / d * oh);
119    if (num == 1)
120        return {h};
121    ld hp = sqrtl(max(0.L, 1 - cosa * cosa)) * r1;
122
123    pt w = ((b - a) / d * hp).rot();
124    return {h + w, h - w};
125}
126
127//a is circle center, p is point
128vector<pt> circleTangents(pt a, ld r, pt p) {
129    ld d2 = (a - p).abs2();
130    ld d = (a - p).abs();
131
132    if (gt(sqr(r), d2)) {
133        //no tangents
134        return {};
135    }
136    if (eq(sqr(r), d2)) {
137        //point lies on circle - one tangent
138        return {p};
139    }
140
141    pt B = p - a;
142    pt H = B * sqr(r) / d2;
143    ld h = sqrtl(d2 - sqr(r)) * ld(r) / d;
144    pt w = (B / d * h).rot();
145    H = H + a;
146    return {H + w, H - w};
147}
148
149vector<pt> lineCircleIntersection(line l, pt a, ld r) {
150    ld d = l.signedDist(a);
151    if (gt(fabsl(d), r))
152        return {};
153    pt h = a - pt{l.a, l.b} * d;
154    if (eq(fabsl(d), r))
155        return {h};
156    pt w{pt{l.a, l.b}.rot() * sqrtl(max<ld>(0, sqr(r)-sqr(d)))};
157    return {h + w, h - w};
158}
159
160//modified magic from e-mazw
161vector<line> commonTangents(pt a, ld r1, pt b, ld r2) {
162    if (a == b && eq(r1, r2)) {
163        //equal circles
164        return {};
165    }
166    vector<line> res;
167    pt c = b - a;
168    ld z = c.abs2();
169    for (int i = -1; i <= 1; i += 2)
170        for (int j = -1; j <= 1; j += 2) {
171            ld r = r2 * j - r1 * i;
172            ld d = z - sqr(r);
173            if (gt(0, d))
174                continue;
175            d = sqrtl(max<ld>(0, d));
176            pt magic = pt{r, d} / z;
177            line l(magic * c, magic % c, r1 * i);
178            l.c -= pt{l.a, l.b} * a;
179            res.push_back(l);
180        }
181    return res;
182}

```

11 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'
8             ↪ viewBox='-1000 -1000 2000 2000'>\n");
9     }
10
11     void line(pt a, pt b) {
12         a = a * sc, b = b * sc;
13         fprintf(out, "<line x1='%Lf' y1='%Lf' x2='%Lf' y2='%Lf'
14             ↪ stroke='black'/>\n", a.x, -a.y, b.x, -b.y);
15     }
16
17     void circle(pt a, ld r = -1, string col = "red") {
18         r = (r == -1 ? 10 : sc * r);
19         a = a * sc;
20         fprintf(out, "<circle cx='%Lf' cy='%Lf' r='%Lf'
21             ↪ fill='%s'/>\n", a.x, -a.y, r, col.c_str());
22     }
23
24     void text(pt a, string s) {
25         a = a * sc;
26         fprintf(out, "<text x='%Lf' y='%Lf'
27             ↪ font-size='10px'>s</text>\n", a.x, -a.y,
28             ↪ s.c_str());
29     }
30
31     void close() {
32         fprintf(out, "</svg>\n");
33         fclose(out);
34         out = 0;
35     }
36
37     ~SVG() {
38         if (out)
39             close();
40     }
41 }
42
43 SVG svg;

```

12 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 (int 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 (int 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';
86     //1 0
87 }

```


13 graphs/directed_mst.cpp

```

1// WARNING: this code wasn't submitted anywhere
2
3namespace TwoChinese {
4
5struct 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};
11typedef vector<vector<Edge>> Graph;
12
13const int maxn = 2050;
14
15// global, for supplementary algorithms
16int b[maxn];
17int tin[maxn], tup[maxn];
18int dtime; // counter for tin, tout
19vector<int> st;
20int nc; // number of strongly connected components
21int q[maxn];
22
23int answer;
24
25void 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
52vector<Edge> bfs(
53    const Graph& e, const vi& init, const vi& 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
78vector<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 comp-s 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
143void 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}

```

14 graphs/edmonds_matching.cpp

```

1 int n;
2 vi e[maxn];
3 int mt[maxn], p[maxn], base[maxn], b[maxn], blos[maxn];
4 int q[maxn];
5 int blca[maxn]; // used for lca
6
7 int lca(int u, int v) {
8     forn(i, n) blca[i] = 0;
9     while (true) {
10         u = base[u];
11         blca[u] = 1;
12         if (mt[u] == -1) break;
13         u = p[mt[u]];
14     }
15     while (!blca[base[v]]) {
16         v = p[mt[base[v]]];
17     }
18     return base[v];
19 }
20
21 void mark_path(int v, int b, int ch) {
22     while (base[v] != b) {
23         blos[base[v]] = blos[base[mt[v]]] = 1;
24         p[v] = ch;
25         ch = mt[v];
26         v = p[mt[v]];
27     }
28 }
29
30 int find_path(int root) {
31     forn(i, n) {
32         base[i] = i;
33         p[i] = -1;
34         b[i] = 0;
35     }
36
37     b[root] = 1;
38     q[0] = root;
39     int lq = 0, rq = 1;
40     while (lq != rq) {
41         int v = q[lq++];
42         for (int to: e[v]) {
43             if (base[v] == base[to] || mt[v] == to) continue;
44             if (to == root || (mt[to] != -1 && p[mt[to]] != -1)) {
45                 int curbase = lca(v, to);
46                 forn(i, n) blos[i] = 0;
47                 mark_path(v, curbase, to);
48                 mark_path(to, curbase, v);
49                 forn(i, n) if (blos[base[i]]) {
50                     base[i] = curbase;
51                     if (!b[i]) b[i] = 1, q[rq++] = i;
52                 }
53             } else if (p[to] == -1) {
54                 p[to] = v;
55                 if (mt[to] == -1) {
56                     return to;
57                 }
58                 to = mt[to];
59                 b[to] = 1;
60                 q[rq++] = to;
61             }
62         }
63     }
64 }
65 return -1;
66 }
67
68 int matching() {
69     forn(i, n) mt[i] = -1;
70     int res = 0;
71     forn(i, n) if (mt[i] == -1) {
72         int v = find_path(i);
73         if (v != -1) {
74             ++res;
75             while (v != -1) {
76                 int pv = p[v], ppv = mt[p[v]];
77                 mt[v] = pv, mt[ppv] = v;
78                 v = ppv;
79             }
80         }
81     }
82     return res;
83 }

```

15 graphs/euler_cycle.cpp

```

1 struct Edge {
2     int to, id;
3 };
4
5 bool usedEdge[maxn];
6 vector<Edge> g[maxn];
7 int ptr[maxn];
8
9 vector<int> cycle;
10 void eulerCycle(int u) {
11     while (ptr[u] < sz(g[u]) && usedEdge[g[u][ptr[u]].id])
12         ++ptr[u];
13     if (ptr[u] == sz(g[u]))
14         return;
15     const Edge &e = g[u][ptr[u]];
16     usedEdge[e.id] = true;
17     eulerCycle(e.to);
18     cycle.push_back(e.id);
19     eulerCycle(u);
20 }
21
22 int edges = 0;
23 void addEdge(int u, int v) {
24     g[u].push_back(Edge{v, edges});
25     g[v].push_back(Edge{u, edges++});
26 }

```

16 math/factor.cpp

```

1//WARNING: only mod <= 1e18
211 mul(ll a, ll b, ll mod) {
3    ll res = a * b - (ll(ld(a) * ld(b) / ld(mod)) * mod);
4    while (res < 0)
5        res += mod;
6    while (res >= mod)
7        res -= mod;
8    return res;
9}
10
11bool millerRabinTest(ll n, ll a) {
12    if (gcd(n, a) > 1)
13        return false;
14    ll x = n - 1;
15    int l = 0;
16    while (x % 2 == 0) {
17        x /= 2;
18        ++l;
19    }
20    ll c = binpow(a, x, n);
21    for (int i = 0; i < l; ++i) {
22        ll nx = mul(c, c, n);
23        if (nx == 1) {
24            if (c != 1 && c != n - 1)
25                return false;
26            else
27                return true;
28        }
29        c = nx;
30    }
31    return c == 1;
32}
33
34bool isPrime(ll n) {
35    if (n == 1)
36        return false;
37    if (n % 2 == 0)
38        return n == 2;
39    for (ll a = 2; a < min<ll>(8, n); ++a)
40        if (!millerRabinTest(n, a))
41            return false;
42    return true;
43}
44
45//WARNING: p is not sorted
46void factorize(ll x, vector<ll> &p) {
47    if (x == 1)
48        return;
49    if (isPrime(x)) {
50        p.push_back(x);
51        return;
52    }
53    for (ll d: {2, 3, 5})
54        if (x % d == 0) {
55            p.push_back(d);
56            factorize(x / d, p);
57            return;
58        }
59    while (true) {
60        ll x1 = rr() % (x - 1) + 1;
61        ll x2 = (mul(x1, x1, x) + 1) % x;
62        int i1 = 1, i2 = 2;
63        while (true) {
64            ll c = (x1 + x - x2) % x;
65            if (c == 0)
66                break;
67            ll g = gcd(c, x);
68            if (g > 1) {
69                factorize(g, p);
70                factorize(x / g, p);
71                return;
72            }
73            if (i1 * 2 == i2) {
74                i1 *= 2;
75                x1 = x2;
76            }
77            ++i2;
78            x2 = (mul(x2, x2, x) + 1) % x;
79        }
80    }
81}
82
83bool isPrimeSlow(int x) {
84    for (int i = 2; i * i <= x; ++i)
85        if (x % i == 0)
86            return false;
87    return x != 1;
88}
89
90void test() {
91    for (i, 100000) {
92        if (i == 0)
93            continue;
94        assert(isPrime(i) == isPrimeSlow(i));
95        vector<ll> p;
96        factorize(i, p);
97        ll prod = 1;
98        for (ll x: p) {
99            assert(x > 1);
100            assert(isPrimeSlow(x));
101            prod *= x;
102        }
103        assert(prod == i);
104    }
105}

```

17 math/fft.cpp

```

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

```

18 math/golden_search.cpp

```

1 ld f(ld x) {
2     return 5 * x * x + 100 * x + 1; //-10 is minimum
3 }
4
5 ld 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
30 int main() {
31     std::cout << goldenSearch(-100, 100) << '\n';
32 }

```

19 math/numbers.tex

- Simpson and Gauss numerical integration:

$$\int_a^b f(x)dx = (b-a)/6 \cdot (f(a) + 4(f(a+b)/2) + f(b))$$

$$\int_{-1}^1 x_{1,3} = \pm\sqrt{0.6}, x_2 = 0; a_{1,3} = 5/9, a_2 = 8/9$$

- Large primes: $10^{18} + 3, +31, +3111, 10^9 + 21, +33$

- FFT modules:

$$\begin{array}{lll} 1\ 107\ 296\ 257 & 2^{25} \cdot 3 \cdot 11 + 1 & 10 \\ 1\ 161\ 822\ 209 & 2^{22} \cdot 277 + 1 & 3 \\ 1\ 261\ 007\ 895\ 663\ 738\ 881 & 2^{55} \cdot 5 \cdot 7 + 1 & 6 \text{ (check)} \end{array}$$

- Fibonacci numbers:

$$\begin{array}{ll} 1, 2 : & 1 \\ 45 : & 1134903170 \\ 46 : & 1836311903 \text{ (max int)} \\ 47 : & 2971215073 \text{ (max unsigned)} \\ 91 : & 4660046610375530309 \\ 92 : & 7540113804746346429 \text{ (max i64)} \\ 93 : & 12200160415121876738 \text{ (max unsigned i64)} \end{array}$$

- Powers of two

$$\begin{array}{l} 2^{31} = 2\ 147\ 483\ 648 = 2.1 \cdot 10^9 \\ 2^{32} = 4\ 294\ 967\ 296 = 4.2 \cdot 10^9 \\ 2^{63} = 9\ 223\ 372\ 036\ 854\ 775\ 808 = 9.2 \cdot 10^{18} \\ 2^{64} = 18\ 446\ 744\ 073\ 709\ 551\ 616 = 1.8 \cdot 10^{19} \end{array}$$

- Highly composite numbers

$$\begin{array}{l} - \leq 1000: d(840) = 32, \leq 10^4: d(9\ 240) = 64 \\ - \leq 10^5: d(83\ 160) = 128, \leq 10^6: d(720\ 720) = 240 \\ - \leq 10^7: d(8\ 648\ 640) = 448, \leq 10^8: d(91\ 891\ 800) = 768 \\ - \leq 10^9: d(931\ 170\ 240) = 1344 \\ - \leq 10^{11}: d(97\ 772\ 875\ 200) = 4032 \\ - \leq 10^{12}: d(963\ 761\ 198\ 400) = 6720 \\ - \leq 10^{15}: d(866\ 421\ 317\ 361\ 600) = 26880 \\ - \leq 10^{18}: d(897\ 612\ 484\ 786\ 617\ 600) = 103680 \end{array}$$

- Misc

$$\begin{array}{l} - \text{Расстояние между точками по сфере: } L = R \cdot \arccos(\cos \theta_1 \cdot \cos \theta_2 + \sin \theta_1 \cdot \sin \theta_2 \cdot \cos(\varphi_1 - \varphi_2)), \\ \text{где } \theta \text{ — широты (от } -\frac{\pi}{2} \text{ до } \frac{\pi}{2}), \varphi \text{ — долготы (от } -\pi \text{ до } \pi). \\ - \text{Объём шарового сегмента: } V = \pi h^2(R - \frac{1}{3}h), \text{ где } h \text{ — высота от вершины сектора до секущей плоскости} \\ - \text{Площадь поверхности шарового сегмента: } S = 2\pi Rh, \text{ где } h \text{ — высота.} \end{array}$$

$$\begin{array}{l} \bullet \text{ Bell numbers: } 0:1, 1:1, 2:2, 3:5, 4:15, 5:52, \\ 6:203, 7:877, 8:4140, 9:21147, 10:115975, \\ 11:678570, 12:4213597, 13:27644437, 14:190899322, \\ 15:1382958545, 16:10480142147, 17:82864869804, \\ 18:682076806159, 19:5832742205057, 20:51724158235372, \\ 21:474869816156751, 22:4506715738447323, \\ 23:44152005855084346 \end{array}$$

$$\begin{array}{l} \bullet \text{ Catalan numbers: } 0:1, 1:1, 2:2, 3:5, 4:14, 5:42, 6:132, 7:429, \\ 8:1430, 9:4862, 10:16796, 11:58786, 12:208012, 13:742900, \\ 14:2674440, 15:9694845, 16:35357670, 17:129644790, \\ 18:477638700, 19:1767263190, 20:6564120420, \\ 21:24466267020, 22:91482563640, 23:343059613650, \\ 24:1289904147324, 25:4861946401452 \end{array}$$

20 math/stuff.cpp

```

1const int M = 1e6;
2
3int phi[M];
4
5void calcPhi() {
6    for (int i = 1; i < M; ++i)
7        phi[i] = i;
8    for (int j = 1; j < M; ++j)
9        for (int i = 2 * j; i < M; i += j)
10            phi[i] -= phi[j];
11}
12int inv[M];
13
14void calcInv() {
15    inv[1] = 1;
16    for (int i = 2; i < M; ++i) {
17        inv[i] = mul(sub(0, mod / i), inv[mod % i]);
18        assert(mul(i, inv[i]) == 1);
19    }
20}
21
22int gcd(int a, int b, int &x, int &y) {
23    if (a == 0) {
24        x = 0, y = 1;
25        return b;
26    }
27    int x1, y1;
28    int g = gcd(b % a, a, x1, y1);
29    x = y1 - x1 * (b / a);
30    y = x1;
31    assert(a * x + b * y == g);
32    return g;
33}
34
35int crt(int mod1, int mod2, int rem1, int rem2) {
36    int r = (rem2 - (rem1 % mod2) + mod2) % mod2;
37    int x, y;
38    int g = gcd(mod1, mod2, x, y);
39    assert(r % g == 0);
40
41    x %= mod2;
42    if (x < 0)
43        x += mod2;
44
45    int ans = (x * (r / g)) % mod2;
46    ans = ans * mod1 + rem1;
47
48    assert(ans % mod1 == rem1);
49    assert(ans % mod2 == rem2);
50    return ans;
51}
52
53int main() {
54    calcPhi();
55    assert(phi[30] == 1 * 2 * 4);
56    calcInv();
57    int x, y;
58    gcd(3, 5, x, y);
59    gcd(15, 10, x, y);
60    crt(15, 13, 2, 5);
61    crt(17, 3, 15, 2);
62    return 0;
63}

```

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/duval_manacher.cpp

```

1 /*
2  Строка простая, если строго меньше всех суффиксов <=>
3  наименьший циклический сдвиг - первый.
4  Декомпозиция Линдона - разбиение s на w1, w2, ... wk -
5  простые строки такие, что w1 >= w2 >= ... wk.
6 */
7 int duval(string s) {
8     s += s; //remove this to find Lyndon decomposition of s
9     int n = s.size();
10    int i = 0;
11    int ans = 0;
12    //while (i < n) { //for Lyndon decomposition
13    while (i < n / 2) {
14        ans = i;
15        int j = i + 1, k = i;
16        while (j < n && s[k] <= s[j]) {
17            if (s[k] < s[j])
18                k = i;
19            else
20                ++k;
21            ++j;
22        }
23        while (i <= k) {
24            //s.substr(i, j - k) -
25            //next prime string of Lyndon decomposition
26            i += j - k;
27        }
28    }
29    return ans;
30 }
31
32 //actual odd length is (odd[i] * 2 - 1)
33 //actual even length is (even[i] * 2)
34 void manacher(const string& s, vi &odd, vi &even) {
35     int n = s.size();
36     odd.resize(n);
37     int c = -1, r = -1;
38     forn(i, n) {
39         int k = (r <= i ? 0 : min(odd[2 * c - i], r - i));
40         while (i + k < n && i - k >= 0 && s[i + k] == s[i - k])
41             ++k;
42         odd[i] = k;
43         if (i + k > r)
44             r = i + k, c = i;
45     }
46     c = -1, r = -1;
47     even.resize(n - 1);
48     forn(i, n - 1) {
49         int k = (r <= i ? 0 : min(even[2 * c - i], r - i));
50         while (i + k + 1 < n && i - k >= 0 &&
51             s[i + k + 1] == s[i - k])
52             ++k;
53         even[i] = k;
54         if (i + k > r)
55             c = i, r = i + k;
56     }
57 }
58
59 void test() {
60     vector<int> odd, even;
61     string s = "aaaabbaaaaa";
62     manacher(s, odd, even);
63     for (int x: even)
64         cerr << x << ' ';
65     cerr << '\n';
66     for (int x: odd)
67         cerr << x << ' ';
68     cerr << '\n';
69     // 1 2 1 0 5 0 1 2 2 1
70     // 1 2 2 1 1 1 1 2 3 2 1
71 }
72
73 int main() {
74     cout << duval("ababcabab") << '\n'; // 5
75     test();
76 }

```

23 strings/eertree.cpp

```

1#include <bits/stdc++.h>
2using namespace std;
3const int maxn = 5000100;
4const int inf = 1e9 + 1e5;
5
6char buf[maxn];
7char *s = buf + 1;
8int to[maxn][2];
9int suff[maxn];
10int len[maxn];
11int sz;
12int last;
13
14const int odd = 1;
15const int even = 2;
16const int blank = 3;
17
18inline void go(int &u, int pos) {
19    while (u != blank && s[pos - len[u] - 1] != s[pos])
20        u = suff[u];
21}
22
23void 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
37void 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
46void build() {
47    init();
48    scanf("%s", s);
49    for (int i = 0; s[i]; ++i)
50        add_char(i);
51}

```

24 strings/suffix_array.cpp

```

1string s;
2int n;
3int sa[maxn], new_sa[maxn], cls[maxn], new_cls[maxn],
4    cnt[maxn], lcp[maxn];
5int n_cls;
6
7void 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 perm 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
47int 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}

```

25 strings/ukkonen.cpp

```

1#include <bits/stdc++.h>
2using namespace std;
3#define sz(x) ((int) (x).size())
4#define forn(i,n) for (int i = 0; i < int(n); ++i)
5const int inf = int(1e9) + int(1e5);
6
7string s;
8const int alpha = 26;
9
10namespace 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
93        return at(pos) == c;
94    }
95
96    void fixLink(Node *&bad, Node *newBad) {
97        if (bad)
98            bad->lnk = cur;
99        bad = newBad;
100    }
101
102    void addCharOnPos(int id) {
103        Node *bad = 0;
104        while (!canGo(at(id))) {
105            if (cur->r != pos) {
106                splitNode();
107                fixLink(bad, cur);
108                bad = cur;
109            } else {
110                fixLink(bad, 0);
111            }
112            addLeaf(id);
113            goUp();
114        }
115        fixLink(bad, 0);
116        goDown(id, id + 1);
117    }
118
119    int cnt(Node *u, int ml) {
120        if (!u)
121            return 0;
122        int res = min(ml, u->r) - u->l;
123        forn (i, alpha)
124            res += cnt(u->to[i], ml);
125        return res;
126    }
127
128    void build(int l) {
129        init();
130        forn (i, l)
131            addCharOnPos(i);
132    };
133
134    int main() {
135        cin >> s;
136        SuffixTree::build(s.size());
137    }

```


26 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*/
8struct 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 -inf1;
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
58struct 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 = -inf1;
67      for (auto p: v)
68          best = max(best, p.first * x + p.second);
69      return best;
70  }
71};
72
73int 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}

```

27 structures/heavy_light.cpp

```

1const int maxn = 100500;
2const int maxd = 17;
3
4vector<int> g[maxn];
5
6struct Tree {
7  vector<int> t;
8  int base;
9
10  Tree(): base(0) {}
11
12  Tree(int n) {
13      base = 1;
14      while (base < n)
15          base *= 2;
16      t = vector<int>(base * 2, 0);
17  }
18
19  void put(int v, int delta) {
20      assert(v < base);
21      v += base;
22      t[v] += delta;
23      while (v > 1) {
24          v /= 2;
25          t[v] = max(t[v * 2], t[v * 2 + 1]);
26      }
27  }
28
29  //Careful here: cr = 2 * maxn
30  int get(int l, int r, int v=1, int cl=0, int cr = 2*maxn) {
31      cr = min(cr, base);
32      if (l <= cl && cr <= r)
33          return t[v];
34      if (r <= cl || cr <= l)
35          return 0;
36      int cc = (cl + cr) / 2;
37      return max(get(l, r, v * 2, cl, cc),
38                get(l, r, v * 2 + 1, cc, cr));
39  }
40
41};
42
43namespace HLD {
44  int h[maxn];
45  int timer;
46  int in[maxn], out[maxn], cnt[maxn];
47  int p[maxd][maxn];
48  int vroot[maxn];
49  int vpos[maxn];
50  int ROOT;
51  Tree tree[maxn];
52
53  void dfs1(int u, int prev) {
54      p[0][u] = prev;
55      in[u] = timer++;
56      cnt[u] = 1;
57      for (int v: g[u]) {
58          if (v == prev)
59              continue;
60          h[v] = h[u] + 1;
61          dfs1(v, u);
62          cnt[u] += cnt[v];
63      }
64      out[u] = timer;
65  }
66
67  int dfs2(int u, int prev) {
68      int to = -1;
69      for (int v: g[u]) {
70          if (v == prev)
71              continue;
72          if (to == -1 || cnt[v] > cnt[to])
73              to = v;
74      }
75      int len = 1;
76      for (int v: g[u]) {
77          if (v == prev)
78              continue;
79          if (to == v) {
80              vpos[v] = vpos[u] + 1;
81              vroot[v] = vroot[u];
82              len += dfs2(v, u);
83          }
84          else {
85              vroot[v] = v;
86              vpos[v] = 0;
87              dfs2(v, u);
88          }
89      }
90      if (vroot[u] == u)
91          tree[u] = Tree(len);

```

28 structures/linkcut.cpp

```

92     return len;
93 }
94
95 void init(int n) {
96     timer = 0;
97     h[ROOT] = 0;
98     dfs1(ROOT, ROOT);
99     forn (d, maxd - 1)
100         forn (i, n)
101             p[d + 1][i] = p[d][p[d][i]];
102     vroot[ROOT] = ROOT;
103     vpos[ROOT] = 0;
104     dfs2(ROOT, ROOT);
105     //WARNING: init all trees
106 }
107
108 bool isPrev(int u, int v) {
109     return in[u] <= in[v] && out[v] <= out[u];
110 }
111
112 int lca(int u, int v) {
113     for (int d = maxd - 1; d >= 0; --d)
114         if (!isPrev(p[d][u], v))
115             u = p[d][u];
116     if (!isPrev(u, v))
117         u = p[0][u];
118     return u;
119 }
120
121 //for each v: h[u] >= toh
122 int getv(int u, int toh) {
123     int res = 0;
124     while (h[u] >= toh) {
125         int rt = vroot[u];
126         int l = max(0, toh - h[rt]), r = vpos[u] + 1;
127         res = max(res, tree[rt].get(l, r));
128         if (rt == ROOT)
129             break;
130         u = p[0][rt];
131     }
132     return res;
133 }
134
135 int get(int u, int v) {
136     int w = lca(u, v);
137     return max(getv(u, h[w]), getv(v, h[w] + 1));
138 }
139
140 void put(int u, int val) {
141     int rt = vroot[u];
142     int pos = vpos[u];
143     tree[rt].put(pos, val);
144 }
145};

```

```

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);

```

29 structures/ordered_set.cpp

```

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
104inline 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
112inline void link(int x, int y) {
113    makeRoot(x);
114    v2n[x]->pp = v2n[y];
115}
116
117inline 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
129inline 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}

```

```

1 #include <ext/pb_ds/assoc_container.hpp>
2 #include <ext/pb_ds/tree_policy.hpp>
3
4 typedef __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
11 int 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 <<
23         (end(X)==X.find_by_order(6)) << std::endl; // true
24
25     std::cout << X.order_of_key(-5) << std::endl; // 0
26     std::cout << X.order_of_key(1) << std::endl; // 0
27     std::cout << X.order_of_key(3) << std::endl; // 2
28     std::cout << X.order_of_key(4) << std::endl; // 2
29     std::cout << X.order_of_key(400) << std::endl; // 5
30 }

```

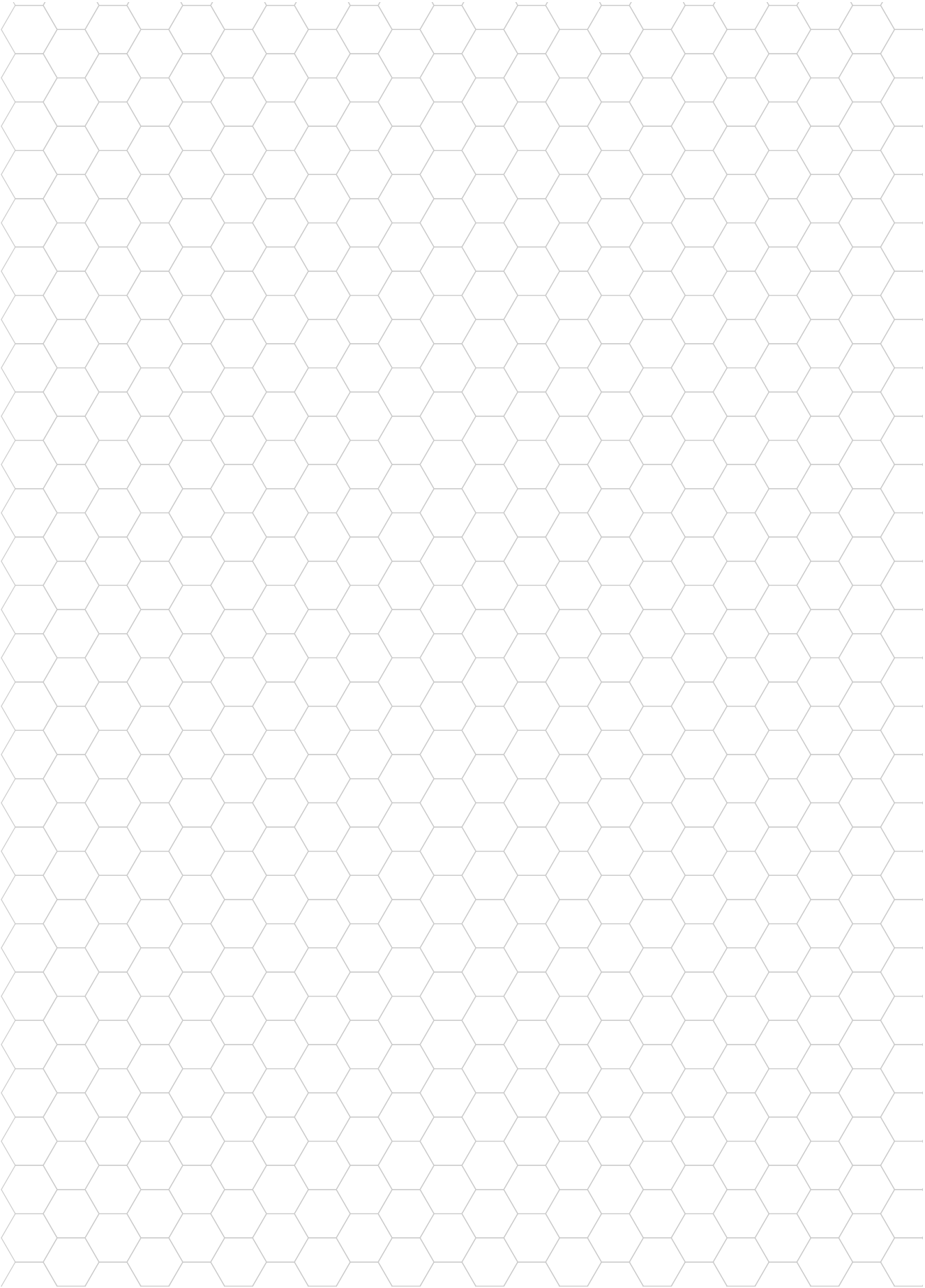
30 structures/treap.cpp

```

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 }

```

31 Сеточка



32 Сеточка

