

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/basic3d.cpp	4
7	geometry/chan.cpp	5
8	geometry/halfplanes.cpp	6
9	geometry/nd_convex_hull.cpp	6
10	geometry/polygon.cpp	7
11	geometry/polygon_tangents.cpp	7
12	geometry/primitives.cpp	8
13	graphs/2sat.cpp	9
14	graphs/directed_mst.cpp	9
15	graphs/edmonds_matching.cpp	10
16	graphs/euler_cycle.cpp	10
17	graphs/kuhn.cpp	11
18	graphs/min_automaton.cpp	11
19	math/big.cpp	12
20	math/factor.cpp	13
21	math/golden_search_quad_eq.cpp	14
22	math/numbers.tex	14
23	math/quadratic_equation.cpp	15
24	math/simplex.cpp	15
25	math/stuff.cpp	16
26	strings/automaton.cpp	17
27	strings/duval_manacher.cpp	17
28	strings/eertree.cpp	18
29	strings/hashtables.cpp	18
30	strings/suffix_array.cpp	19
31	strings/ukkonen.cpp	19
32	structures/2d_segtree.cpp	20
33	structures/centroids.cpp	21
34	structures/fenwick.cpp	21
35	structures/heavy_light.cpp	22
36	structures/ordered_set.cpp	22
37	structures/splay.cpp	23
38	structures/treap.cpp	24
39	zzz_narfm/graph/dinic.cpp	25
40	zzz_narfm/graph/ford-falkerson.cpp	25

41	zzz_narfm/graph/lca.cpp	26
42	zzz_narfm/graph/mincost.cpp	27
43	zzz_narfm/misc/convex-hull.cpp	28
44	zzz_narfm/misc/gauss.cpp	28
45	zzz_narfm/strings/aho-corasick.cpp	29
46	zzz_narfm/strings/prefix_fun.cpp	29
47	zzz_narfm/strings/suffix_array.cpp	29
48	zzz_narfm/strings/z_function.cpp	29
49	zzz_narfm/structures/segtree_assign.cpp	30
50	zzz_narfm/structures/segtree_lazy.cpp	30
51	zzz_narfm/structures/segtree_persist.cpp	30
52	zzz_narfm/structures/sparse.cpp	30

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
 - ↪ RE: 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))
- (WA) Проверить, что ответ влезает в int

2 flows/dinic.cpp

```

1 namespace Dinic {
2 const int maxn = 100100;
3 struct Edge {
4     int to;
5     ll c, f;
6     Edge(int to, ll c): to(to), c(c), f(0) {}
7 };
8
9 vector<Edge> es;
10 vector<int> g[maxn];
11 int q[maxn], d[maxn], pos[maxn];
12 int N, S, T;
13
14 void addEdge(int u, int v, ll c) {
15     g[u].push_back(sz(es));
16     es.emplace_back(v, c);
17     g[v].push_back(sz(es));
18     es.emplace_back(u, 0);
19 }
20
21 bool bfs() {
22     fill(d, d + N, maxn);
23     d[S] = 0, q[0] = S;
24     int rq = 1;
25     forn (lq, rq) {
26         int u = q[lq];
27         for (int id: g[u]) {
28             if (es[id].c == es[id].f)
29                 continue;
30             int v = es[id].to;
31             if (d[v] == maxn) {
32                 d[v] = d[u] + 1;
33                 q[rq++] = v;
34             }
35         }
36     }
37     return d[T] != maxn;
38 }
39
40 ll dfs(int u, ll curf) {
41     if (u == T)
42         return curf;
43     ll ret = 0;
44     for (int &i = pos[u]; i < sz(g[u]); ++i) {
45         int id = g[u][i];
46         int v = es[id].to;
47         ll delta = min(curf, es[id].c - es[id].f);
48         if (delta == 0 || d[v] != d[u] + 1)
49             continue;
50         delta = dfs(v, delta);
51         curf -= delta;
52         ret += delta;
53         es[id].f += delta;
54         es[id ^ 1].f -= delta;
55         if (curf == 0)
56             return ret;
57     }
58     return ret;
59 }
60
61 ll dinic(int S, int T) {
62     Dinic::S = S, Dinic::T = T;
63     ll res = 0;
64     while (bfs()) {
65         fill(pos, pos + N, 0);
66         while (ll cur = dfs(S, infl))
67             res += cur;
68     }
69     return res;
70 }
71
72 } // namespace Dinic
73
74 void test() {
75     Dinic::N = 4;
76     Dinic::addEdge(0, 1, 1);
77     Dinic::addEdge(0, 2, 2);
78     Dinic::addEdge(2, 1, 1);
79     Dinic::addEdge(1, 3, 2);
80     Dinic::addEdge(2, 3, 1);
81     cout << Dinic::dinic(0, 3) << endl; // 3
82 }
83 /*
84 LR-поток находит не максимальный поток.
85 Добавим новый сток S' и исток T'. Заменяем ребро (u, v, l, r)
86 LR-сети на ребра (u, T', l), (S', v, l), (u, v, r - l).
87 Добавим ребро (T, S, k). Ставим значение k=inf, пускаем поток.
88 Проверяем, что все ребра из S' насыщены (иначе ответ не
89 существует). Бинарным поиском находим наименьшее k, что величина
90 потока не изменится. Это k - величина МИНИМАЛЬНОГО потока,
91 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/basic3d.cpp

```

92     }
93     return true;
94 }
95
96 bool push() {
97     //2 variants
98     //if (!fordBellman())
99     if (!dijkstra())
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 Plane {
2     pt v;
3     ld c;
4
5     Plane(pt a, pt b, pt c) {
6         v = ((b - a) % (c - a)).norm();
7         this->c = a * v;
8     }
9
10    ld dist(pt p) {
11        return p * v - c;
12    }
13};
14
15 pt projection(pt p, pt a, pt b) {
16     pt v = b - a;
17     if (ze(v.abs2())) {
18         //stub: bad line
19         return a;
20     }
21     return a + v * (((p - a) * v) / (v * v));
22 }
23
24 pair<pt, pt> planesIntersection(Plane a, Plane b) {
25     pt dir = a.v % b.v;
26     if (ze(dir.abs2())) {
27         //stub: parallel planes
28         return {pt{1e18, 1e18, 1e18}, pt{1e18, 1e18, 1e18}};
29     }
30     ld s = a.v * b.v;
31     pt v3 = b.v - a.v * s;
32     pt h = a.v * a.c + v3 * ((b.c - a.c * s) / (v3 * v3));
33     return {h, h + dir};
34 }
35
36 pair<pt, pt> commonPerpendicular(pt a, pt b, pt c, pt d) {
37     pt v = (b - a) % (d - c);
38     ld S = v.abs();
39     if (ze(S)) {
40         //stub: parallel lines
41         return {pt{1e18, 1e18, 1e18}, pt{1e18, 1e18, 1e18}};
42     }
43     v = v.norm();
44     pt sh = v * (v * c - v * a);
45     pt a2 = a + sh;
46     ld s1 = ((c - a2) % (d - a2)) * v;
47     pt p = a + (b - a) * (s1 / S);
48     return {p, p + sh};
49 }
50
51 /*
52 Absolute error test
53 testProjection: 1e1 -> -16.3
54 testProjection: 1e3 -> -14.1
55 testProjection: 1e4 -> -13.1
56 testProjection: 1e5 -> -12.3
57 testProjection: 1e6 -> -11.2
58 testPlanesIntersection: 1e1 -> -11.5
59 testPlanesIntersection: 1e3 -> -8.6
60 testPlanesIntersection: 1e4 -> -8.3
61 testPlanesIntersection: 1e5 -> -7.4
62 testPlanesIntersection: 1e6 -> -6.5
63 testCommonPerpendicular: 1e1 -> -13.5
64 testCommonPerpendicular: 1e3 -> -11.4
65 testCommonPerpendicular: 1e4 -> -10.5
66 testCommonPerpendicular: 1e5 -> -8.7
67 testCommonPerpendicular: 1e6 -> -8.6
68 */

```

7 geometry/chan.cpp

```

1mt19937 rr(111);
2ld rndEps() {
3    return (ld(rr()) / rr.max() - 0.5) * 1e-7;
4}
5
6typedef tuple<int, int, int> Face;
7const ld infc = 1e100;
8
9int n;
10pt p[maxn];
11
12namespace Chan {
13pt _p[maxn];
14
15ld turny(int p1, int p2, int p3) {
16    return (p[p2].x - p[p1].x) * (p[p3].y - p[p1].y) -
17           (p[p3].x - p[p1].x) * (p[p2].y - p[p1].y);
18}
19
20//replace y with z
21ld turnz(int p1, int p2, int p3) {
22    return (p[p2].x - p[p1].x) * (p[p3].z - p[p1].z) -
23           (p[p3].x - p[p1].x) * (p[p2].z - p[p1].z);
24}
25
26ld gett(int p1, int p2, int p3) {
27    if (p1 == -1 || p2 == -1 || p3 == -1)
28        return infc;
29    ld ty = turny(p1, p2, p3);
30    if (ty >= 0)
31        return infc;
32    else
33        return turnz(p1, p2, p3) / ty;
34}
35
36void act(int i) {
37    if (p[i].onHull) {
38        p[p[i].nx].pr = p[i].pr;
39        p[p[i].pr].nx = p[i].nx;
40    } else {
41        p[p[i].nx].pr = p[p[i].pr].nx = i;
42    }
43    p[i].onHull ^= 1;
44}
45
46ld updt(vector<int> &V) {
47    if (V.empty())
48        return infc;
49    int id = V.back();
50    if (p[id].onHull)
51        return gett(p[id].pr, p[id].nx, id);
52    else
53        return gett(p[id].pr, id, p[id].nx);
54}
55
56//builds lower hull
57vector<int> buildHull(int l, int r) {
58    if (l + 1 >= r) {
59        p[l].pr = p[l].nx = -1;
60        p[l].onHull = true;
61        return {};
62    }
63    int mid = (l + r) / 2;
64    auto L = buildHull(l, mid);
65    auto R = buildHull(mid, r);
66    reverse(all(L));
67    reverse(all(R));
68    int u = mid - 1, v = mid;
69    while (true) {
70        if (p[u].pr != -1 &&
71            (turny(p[u].pr, u, v) <= 0))
72            u = p[u].pr;
73        else if (p[v].nx != -1 &&
74            (turny(u, v, p[v].nx) <= 0))
75            v = p[v].nx;
76        else
77            break;
78    }
79
80    ld t[6];
81    t[0] = updt(L);
82    t[1] = updt(R);
83    vector<int> A;
84    while (true) {
85        t[2] = gett(p[u].pr, v, u);
86        t[3] = gett(u, p[u].nx, v);
87        t[4] = gett(u, p[v].pr, v);
88        t[5] = gett(u, p[v].nx, v);
89        ld nt = infc;
90        int type = -1;
91        for (i, 6)

```

```

92            if (t[i] < nt)
93                nt = t[i], type = i;
94        if (nt >= infc)
95            break;
96
97        if (type == 0) {
98            act(L.back());
99            if (L.back() < u)
100                A.push_back(L.back());
101            L.pop_back();
102            t[0] = updt(L);
103        } else if (type == 1) {
104            act(R.back());
105            if (R.back() > v)
106                A.push_back(R.back());
107            R.pop_back();
108            t[1] = updt(R);
109        } else if (type == 2) {
110            A.push_back(u);
111            u = p[u].pr;
112        } else if (type == 3) {
113            A.push_back(u = p[u].nx);
114        } else if (type == 4) {
115            A.push_back(v = p[v].pr);
116        } else if (type == 5) {
117            A.push_back(v);
118            v = p[v].nx;
119        }
120    }
121    assert(L.empty() && R.empty());
122
123    p[u].nx = v, p[v].pr = u;
124    for (int i = u + 1; i < v; ++i)
125        p[i].onHull = false;
126    for (int i = sz(A) - 1; i >= 0; --i) {
127        int id = A[i];
128        if (id <= u || id >= v) {
129            if (u == id)
130                u = p[u].pr;
131            if (v == id)
132                v = p[v].nx;
133            act(id);
134        } else {
135            p[id].pr = u, p[id].nx = v;
136            act(id);
137            if (id >= mid)
138                v = id;
139            else
140                u = id;
141        }
142    }
143    return A;
144}
145
146//faces are oriented ccw if look from the outside
147vector<Face> getFaces() {
148    for (i, n) {
149        _p[i] = p[i];
150        p[i].x += rndEps();
151        p[i].y += rndEps();
152        p[i].z += rndEps();
153        p[i].id = i;
154    }
155    sort(p, p + n, [](const pt &a, const pt &b) {
156        return a.x < b.x;
157    });
158    vector<Face> faces;
159    for (q, 2) {
160        auto movie = buildHull(0, n);
161        for (int x: movie) {
162            int id = p[x].id;
163            int pid = p[p[x].pr].id;
164            int nid = p[p[x].nx].id;
165            if (!p[x].onHull)
166                faces.emplace_back(pid, id, nid);
167            else
168                faces.emplace_back(pid, nid, id);
169            act(x);
170        }
171        for (i, n) {
172            p[i].y *= -1;
173            p[i].z *= -1;
174        }
175    }
176    for (i, n)
177        p[i] = _p[i];
178    return faces;
179}
180
181} //namespace Chan

```

8 geometry/halfplanes.cpp

```

1ld det3x3(line a, line b, line c) {
2    return a.c * (b.v % c.v)
3        + b.c * (c.v % a.v)
4        + c.c * (a.v % b.v);
5}
6
7//check: bounding box is included
8vector<pt> halfplanesIntersection(vector<line> l) {
9    sort(all(l), cmpLine); //the strongest constraint is first
10   l.erase(unique(all(l), eqLine), l.end());
11   int n = sz(l);
12   vi st;
13   forn (iter, 2)
14       forn (i, n) {
15           while (sz(st) > 1) {
16               int j = st.back(), k = *next(st.rbegin());
17               if (l[k].v % l[i].v <= eps ||
18                   det3x3(l[k], l[j], l[i]) <= eps)
19                   break;
20               st.pop_back();
21           }
22           st.push_back(i);
23       }
24
25   vi pos(n, -1);
26   bool ok = false;
27   forn (i, sz(st)) {
28       int id = st[i];
29       if (pos[id] != -1) {
30           st = vi(st.begin() + pos[id], st.begin() + i);
31           ok = true;
32           break;
33       } else
34           pos[id] = i;
35   }
36   if (!ok)
37       return {};
38
39   vector<pt> res;
40   pt M{0, 0};
41   int k = sz(st);
42   forn (i, k) {
43       line l1 = l[st[i]], l2 = l[st[(i + 1) % k]];
44       res.push_back(linesIntersection(l1, l2));
45       M = M + res.back();
46   }
47   M = M * (1. / k);
48   for (int id: st)
49       if (l[id].signedDist(M) < -eps)
50           return {};
51   return res;
52}

```

9 geometry/nd_convex_hull.cpp

```

1const int DIM = 4;
2typedef array<ll, DIM> pt;
3pt operator-(const pt &a, const pt &b) {
4    pt res;
5    forn (i, DIM)
6        res[i] = a[i] - b[i];
7    return res;
8}
9typedef array<pt, DIM-1> Edge;
10typedef array<pt, DIM> Face;
11vector<Face> faces;
12
13ll det(pt *a) {
14    int p[DIM];
15    iota(p, p + DIM, 0);
16    ll res = 0;
17    do {
18        ll x = 1;
19        forn (i, DIM) {
20            forn (j, i)
21                if (p[j] > p[i])
22                    x *= -1;
23            x *= a[i][p[i]];
24        }
25        res += x;
26    } while (next_permutation(p, p + DIM));
27    return res;
28}
29
30ll V(Face f, pt pivot) {
31    pt p[DIM];
32    forn (i, DIM)
33        p[i] = f[i] - pivot;
34    return det(p);
35}
36
37void init(vector<pt> p) {
38    forn (i, DIM+1) {
39        Face a;
40        int q = 0;
41        forn (j, DIM+1)
42            if (j != i)
43                a[q++] = p[j];
44        ll v = V(a, p[i]);
45        assert(v != 0);
46        if (v < 0)
47            swap(a[0], a[1]);
48        faces.push_back(a);
49    }
50}
51
52void add(pt p) {
53    vector<Face> newf, bad;
54    for (auto f: faces) {
55        if (V(f, p) < 0)
56            bad.push_back(f);
57        else
58            newf.push_back(f);
59    }
60    if (bad.empty()) {
61        return;
62    }
63    faces = newf;
64    vector<pair<Edge, pt>> edges;
65    for (auto f: bad) {
66        sort(all(f));
67        forn (i, DIM) {
68            Edge e;
69            int q = 0;
70            forn (j, DIM)
71                if (i != j)
72                    e[q++] = f[j];
73            edges.emplace_back(e, f[i]);
74        }
75    }
76    sort(all(edges));
77    forn (i, sz(edges)) {
78        if (i + 1 < sz(edges) &&
79            edges[i + 1].first == edges[i].first) {
80            ++i;
81            continue;
82        }
83        Face f;
84        forn (j, DIM-1)
85            f[j] = edges[i].first[j];
86        f[DIM-1] = p;
87        if (V(f, edges[i].second) < 0)
88            swap(f[0], f[1]);
89        faces.push_back(f);
90    }
91}

```

10 geometry/polygon.cpp

```

1 bool 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 //check: p is oriented ccw
13 bool segmentInsidePolygon(pt a, pt b, pt *p, int n) {
14     if (!pointInsidePolygon((a + b) * .5, p, n))
15         return false;
16     if (ze((a - b).abs()))
17         return true;
18     forn (i, n) {
19         pt c = p[i];
20         if (ze((a - c) % (b - c)) &&
21             (a - c) * (b - c) < -eps) {
22             //point inside interval
23             pt pr = p[(i + n - 1) % n];
24             pt nx = p[(i + 1) % n];
25             if ((c - pr) % (nx - c) > eps)
26                 return false;
27             ld s1 = (pr - a) % (b - a);
28             ld s2 = (nx - a) % (b - a);
29             if ((s1 > eps || s2 > eps) &&
30                 (s1 < -eps || s2 < -eps))
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 (s1 >= -eps && s2 >= -eps)
38             continue;
39         if (s1 <= eps && s2 <= eps)
40             continue;
41
42         s1 = (c - a) % (b - a);
43         s2 = (d - a) % (b - a);
44         if (s1 >= -eps && s2 >= -eps)
45             continue;
46         if (s1 <= eps && s2 <= eps)
47             continue;
48
49         return false;
50     }
51     return true;
52 }

```

11 geometry/polygon_tangents.cpp

```

1 struct Cmp {
2     pt M, v0;
3
4     bool operator()(const pt &a, const pt &b) {
5         pt va{v0 * (a - M), v0 % (a - M)};
6         pt vb{v0 * (b - M), v0 % (b - M)};
7         return cmpAngle(va, vb);
8     }
9 };
10
11 struct Hull {
12     vector<pt> h;
13     int n;
14
15     void build() {
16         sort(all(h));
17         h.erase(unique(all(h)), h.end());
18         vector<pt> top, bot;
19         for (auto p: h) {
20             while (sz(bot) > 1 && (p - bot.back()) %
21                 (p - *next(bot.rbegin())) >= -eps)
22                 bot.pop_back();
23             bot.push_back(p);
24             while (sz(top) > 1 && (p - top.back()) %
25                 (p - *next(top.rbegin())) <= eps)
26                 top.pop_back();
27             top.push_back(p);
28         }
29         if (sz(top))
30             top.pop_back();
31         reverse(all(top));
32         if (sz(top))
33             top.pop_back();
34         h = bot;
35         h.insert(h.end(), all(top));
36         n = sz(h);
37     }
38
39     bool visSide(pt a, int i) {
40         return (h[(i + 1) % n] - a) % (h[i % n] - a) > eps;
41     }
42
43     bool vis(pt a, int i) {
44         return visSide(a, i) || visSide(a, i + n - 1);
45     }
46
47     bool isTangent(pt a, int i) {
48         return visSide(a, i) != visSide(a, i + n - 1);
49     }
50
51     int binSearch(int l, int r, pt a) {
52         //tricky binsearch; l < r not necessarily
53         while (abs(l - r) > 1) {
54             int c = (l + r) / 2;
55             if (vis(a, c))
56                 l = c;
57             else
58                 r = c;
59         }
60         assert(isTangent(a, l));
61         return l % n;
62     }
63
64     //check: n >= 3
65     pair<int, int> tangents(pt a) {
66         assert(n >= 3);
67         pt M = (h[0] + h[1] + h[2]) * (1. / 3);
68         if (a == M)
69             return {-1, -1};
70         Cmp cmp{M, h[0] - M};
71         //assert(is_sorted(all(h), cmp));
72         int pos = upper_bound(all(h), a, cmp) - h.begin();
73         pt L = h[(pos + n - 1) % n], R = h[pos % n];
74         if ((R - L) % (a - L) >= -eps)
75             return {-1, -1}; //point inside hull
76         int pos2 = upper_bound(all(h), M*2-a, cmp) - h.begin();
77         assert(pos % n != pos2 % n);
78         if (pos > pos2)
79             pos2 += n;
80         return {binSearch(pos, pos2, a),
81             binSearch(pos + n - 1, pos2 - 1, a)};
82     }
83 };

```

12 geometry/primitives.cpp

```

1 struct line {
2     pt v;
3     ld c; //  $v \cdot p = c$ 
4
5     //check:  $p_1 \neq p_2$ 
6     line(pt p1, pt p2) {
7         v = (p2 - p1).rot();
8         v = v * (1. / v.abs());
9         c = v * p1;
10    }
11
12    // Convert from  $ax + by + c = 0$ 
13
14    //check:  $a^2 + b^2 > 0$ 
15    line(ld a, ld b, ld _c): v(pt{a, b}), c(-_c) {
16        ld d = v.abs();
17        v = v * (1. / d);
18        c /= d;
19    }
20
21    //check:  $v.abs() == 1$ 
22    ld signedDist(pt p) {
23        return v * p - c;
24    }
25};
26
27//check:  $a \neq b$ 
28pt lineProjection(pt p, pt a, pt b) {
29    pt v = (b - a).rot();
30    ld s = (p - a) % (b - a);
31    return p + v * (s / v.abs2());
32}
33
34ld pointSegmentDist(pt p, pt a, pt b) {
35    if ((p - a) * (b - a) <= 0 || ze((b - a).abs()))
36        return (p - a).abs();
37    if ((p - b) * (a - b) <= 0)
38        return (p - b).abs();
39    return fabs1((p - a) % (p - b)) / (b - a).abs();
40}
41
42pt linesIntersection(line l1, line l2) {
43    ld d = l1.v.x * l2.v.y - l1.v.y * l2.v.x;
44    if (ze(d)) {
45        if (eq(l1.c, l2.c)) {
46            //stub: equal lines
47        } else {
48            //stub: empty intersection
49        }
50        return pt{1e18, 1e18};
51    }
52    ld dx = l1.c * l2.v.y - l1.v.y * l2.c;
53    ld dy = l1.v.x * l2.c - l1.c * l2.v.x;
54    return pt{dx / d, dy / d};
55}
56
57pt linesIntersection(pt a, pt b, pt c, pt d) {
58    ld s = (b - a) % (d - c);
59    if (ze(s)) {
60        //stub: parallel or equal lines
61        return pt{1e18, 1e18};
62    }
63    ld s1 = (c - a) % (d - a);
64    return a + (b - a) * (s1 / s);
65}
66
67bool pointInsideSegment(pt p, pt a, pt b) {
68    if (!ze((p - a) % (p - b)))
69        return false;
70    ld prod = (a - p) * (b - p);
71    return ze(prod) || prod < 0;
72    if (ze(prod)) {
73        //stub: coincides with segment end
74        return true;
75    }
76    return prod < 0;
77}
78
79bool checkSegmentIntersection(pt a, pt b, pt c, pt d) {
80    if (ze((a - b) % (c - d))) {
81        if (pointInsideSegment(a, c, d) ||
82            pointInsideSegment(b, c, d) ||
83            pointInsideSegment(c, a, b) ||
84            pointInsideSegment(d, a, b)) {
85            //stub: intersection of parallel segments
86            return true;
87        }
88        return false;
89    }
90    ld s1, s2;
91    forn (iter, 2) {
92        s1 = (c - a) % (b - a);
93        s2 = (d - a) % (b - a);
94        if (s1 > eps && s2 > eps)
95            return false;
96        if (s1 < -eps && s2 < -eps)
97            return false;
98        swap(a, c), swap(b, d);
99    }
100    return true;
101}
102
103vector<pt> lineCircleIntersection(line l, pt a, ld r) {
104    ld d = l.signedDist(a);
105    pt h = a - l.v * d;
106    if (eq(fabs1(d), r))
107        return {h};
108    else if (fabs1(d) > r)
109        return {};
110    pt w = l.v.rot() * Sqrt(sqr(r) - sqr(d));
111    return {h + w, h - w};
112}
113
114vector<pt> circlesIntersecton(pt a, ld r1, pt b, ld r2) {
115    ld d = (a - b).abs();
116    if (ze(d) && eq(r1, r2)) {
117        //stub: equal circles
118        return {};
119    }
120    // intersection is non-empty iff
121    // triangle with sides r1, r2, d exists
122    ld per = r1 + r2 + d;
123    ld mx = max(max(r1, r2), d);
124    int num = 2;
125    if (eq(mx * 2, per)) {
126        num = 1;
127    } else if (mx * 2 > per)
128        return {};
129    ld part = (sqr(r1) + sqr(d) - sqr(r2)) / ld(2 * d);
130    pt h = a + (b - a) * (part / d);
131    if (num == 1)
132        return {h};
133    ld dh = Sqrt(sqr(r1) - sqr(part));
134    pt w = ((b - a) * (dh / d)).rot();
135    return {h + w, h - w};
136}
137
138vector<pt> circleTangents(pt p, pt a, ld r) {
139    ld d = (p - a).abs();
140    if (eq(r, d))
141        return {p};
142    else if (r > d)
143        return {};
144    ld len = Sqrt(sqr(d) - sqr(r));
145    vector<pt> res;
146    pt vec = (a - p) * (len / sqr(d));
147    for (int sgn: {-1, 1})
148        res.push_back(p + vec.rotCw(pt{len, r * sgn}));
149    return res;
150}
151
152vector<line> circlesBitangents(pt a, ld r1, pt b, ld r2) {
153    ld d = (a - b).abs();
154    if (ze(d) && eq(r1, r2)) {
155        //stub: equal circles
156        return {};
157    }
158
159    vector<line> res;
160    for (int s1: {-1, 1})
161        for (int s2: {-1, 1}) {
162            // inner tangent iff s1 != s2
163            // treat radii as signed
164            ld r = s2 * r2 - s1 * r1;
165            if (eq(fabs1(r), d)) {
166                // incident tangents; need only one copy
167                if (s1 == 1)
168                    continue;
169            } else if (fabs1(r) > d)
170                continue;
171            ld len = Sqrt(sqr(d) - sqr(r));
172            line l(a, a + (b - a).rotCw(pt{len, r}));
173            l.c -= s1 * r1;
174            res.push_back(l);
175        }
176    return res;
177}

```


15 graphs/edmonds_matching.cpp

```

92         cycle[nc].push_back(u);
93         if (u == to) break;
94     }
95     st.push_back(nc);
96     b[nc] = 1;
97     ++nc;
98 } else {
99     st.push_back(to);
100 }
101 }
102 }
103 forn(i, n) assert(done[i]);
104 assert((int)edges.size() == n-1);
105 cout << res << endl;
106}
107
108void scan() {
109     int m;
110     scanf("%d%d", &n, &m);
111     forn(i, n) out[i].fi = new multiset<Edge>();
112     forn(i, m) {
113         int u, v, w;
114         scanf("%d%d%d", &u, &v, &w);
115         --u, --v;
116         out[v].fi->insert(Edge{v, u, i, w});
117     }
118}

```

```

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[pv] = v;
78                 v = ppv;
79             }
80         }
81     }
82     return res;
83}

```

16 graphs/euler_cycle.cpp

```

1 struct Edge {
2     int to, id;
3 };

```

```

4
5bool usedEdge[maxn];
6vector<Edge> g[maxn];
7int ptr[maxn];
8
9vector<int> cycle;
10void 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
22int edges = 0;
23void addEdge(int u, int v) {
24    g[u].push_back(Edge{v, edges++});
25    g[v].push_back(Edge{u, edges++});
26}

```

17 graphs/kuhn.cpp

```

1bool dfs(int v) {
2    if (vis[v]) return false;
3    vis[v] = true;
4    for (int i = 0; i < (int)e[v].size(); i++) {
5        if (mt[e[v][i]] == -1) {
6            mt[e[v][i]] = v;
7            return true;
8        }
9    }
10    for (int i = 0; i < (int)e[v].size(); i++) {
11        if (dfs(mt[e[v][i]])) {
12            mt[e[v][i]] = v;
13            return true;
14        }
15    }
16    return false;
17}
18
19...
20
21fill(pair, -1);
22for (int run = 1; run; ) {
23    run = 0, fill(used, 0);
24    forn(i, n)
25        if (pair[i] == -1 && dfs(i))
26            run = 1;
27}

```

18 graphs/min_automaton.cpp

```

1vi inc[maxn][A];
2int lst[maxn], pos[maxn], part[maxn];
3int lp[maxn], rp[maxn], nrp[maxn];
4int upd[maxn], used[maxn], inq[maxn];
5vector<int> q;
6int dtime;
7int np; // number of classes
8vector<int> toRefine[A];
9
10void doSwap(int x, int y) {
11    swap(lst[pos[x]], lst[pos[y]]);
12    swap(pos[x], pos[y]);
13}
14
15void refine(const vi& a) {
16    ++dtime;
17    vector<int> updated;
18    for (int x: a) {
19        if (used[x] == dtime) continue;
20        used[x] = dtime;
21
22        int p = part[x];
23        if (upd[p] != dtime) {
24            upd[p] = dtime;
25            nrp[p] = rp[p];
26            updated.pb(p);
27        }
28
29        doSwap(x, lst[nrp[p]-1]);
30        --nrp[p];
31    }
32
33    for (int p: updated) {
34        if (lp[p] == nrp[p]) continue;
35        lp[np] = nrp[p];
36        rp[np] = rp[p];
37        rp[p] = nrp[p];
38        for (int i = lp[np]; i < rp[np]; ++i) {
39            part[lst[i]] = np;
40        }
41
42        if (inq[p] || rp[np] - lp[np] < rp[p] - lp[p]) {
43            inq[np] = 1;
44            q.push_back(np);
45        } else {
46            inq[p] = 1;
47            q.push_back(p);
48        }
49
50        ++np;
51    }
52}
53
54void solve() {
55    forn(i, n) lst[i] = i;
56    sort(lst, lst+n, [](int i, int j) {
57        return col[i] < col[j];
58    });
59
60    forn(i, n) {
61        if (i && col[lst[i]] != col[lst[i-1]]) {
62            rp[np] = i;
63            lp[++np] = i;
64        }
65        part[lst[i]] = np;
66        pos[lst[i]] = i;
67    }
68    rp[np++] = n;
69
70    forn(i, np) {
71        inq[i] = 1;
72        q.push_back(i);
73    }
74
75    forn(i, q.size()) {
76        int p = q[i];
77        inq[p] = false;
78        forn(c, A) {
79            toRefine[c].clear();
80            for (int id = lp[p]; id < rp[p]; ++id) {
81                toRefine[c].insert(
82                    toRefine[c].end(), all(inc[lst[id]][c]));
83            }
84        }
85        forn(c, A) if (!toRefine[c].empty()) {
86            refine(toRefine[c]);
87        }
88    }
89
90    forn(i, n) printf("%d\n", part[i] + 1);
91}

```

19 math/big.cpp

```

1 struct bigint {
2     static const int LEN = 60;
3     static const int BIGMOD = 10000;
4     int s; // sign of big integer
5     int vl, v[LEN]; // vl is length of v array
6     bigint()
7     : s(1) {
8         vl = 0;
9     } // eg. bigint x;
10
11     bigint(long long a) { // eg. bigint x(a);
12         s = 1;
13         vl = 0;
14         if (a < 0) {
15             s = -1;
16             a = -a;
17         }
18         while (a) {
19             push_back(a % BIGMOD);
20             a /= BIGMOD;
21         }
22     }
23
24     bigint(string str) { // eg. bigint x(str);
25         s = 1;
26         vl = 0;
27         int stPos = 0, num = 0;
28         if (!str.empty() && str[0] == '-') {
29             stPos = 1;
30             s = -1;
31         }
32         for (int i = str.length() - 1, q = 1; i >= stPos; i--) {
33             num += (str[i] - '0') * q;
34             if ((q *= 10) >= BIGMOD) {
35                 push_back(num);
36                 num = 0;
37                 q = 1;
38             }
39         }
40         if (num)
41             push_back(num);
42     }
43     int len() const { return vl; }
44
45     bool empty() const { return len() == 0; }
46
47     void push_back(int x) { v[vl++] = x; }
48
49     void pop_back() { vl--; }
50
51     int back() const { return v[vl - 1]; }
52
53     void n() {
54         while (!empty() && !back())
55             pop_back();
56     }
57
58     void resize(int nl) {
59         vl = nl;
60         memset(v, 0, sizeof(int) * vl);
61     }
62
63     void print() const {
64         if (empty()) {
65             putchar('0');
66             return;
67         }
68         if (s == -1)
69             putchar('-');
70         printf("%d", back());
71         for (int i = len() - 2; i >= 0; i--)
72             printf("%.4d", v[i]);
73     }
74
75     friend std::ostream& operator<<(
76         std::ostream& out, const bigint& a) {
77         if (a.empty()) {
78             out << "0";
79             return out;
80         }
81         if (a.s == -1)
82             out << "-";
83         out << a.back();
84         for (int i = a.len() - 2; i >= 0; i--) {
85             char str[10];
86             snprintf(str, 5, "%.4d", a.v[i]);
87             out << str;
88         }
89         return out;
90     }
91
92     int compare(const bigint& b) const {
93         if (s != b.s)
94             return s - b.s;
95         if (s == -1)
96             return -(*this).compare(-b);
97         if (len() != b.len())
98             return len() - b.len(); // int
99         for (int i = len() - 1; i >= 0; i--)
100             if (v[i] != b.v[i])
101                 return v[i] - b.v[i];
102         return 0;
103     }
104
105     bool operator<(const bigint& b) const {
106         return compare(b) < 0;
107     }
108
109     bool operator<=(const bigint& b) const {
110         return compare(b) <= 0;
111     }
112
113     bool operator==(const bigint& b) const {
114         return compare(b) == 0;
115     }
116
117     bool operator!=(const bigint& b) const {
118         return compare(b) != 0;
119     }
120
121     bool operator>(const bigint& b) const {
122         return compare(b) > 0;
123     }
124
125     bool operator>=(const bigint& b) const {
126         return compare(b) >= 0;
127     }
128
129     bigint operator-() const {
130         bigint r = (*this);
131         r.s = -r.s;
132         return r;
133     }
134
135     bigint operator+(const bigint& b) const {
136         if (s == -1)
137             return -(*this) + (-b);
138         if (b.s == -1)
139             return (*this) - (-b);
140         bigint r;
141         int nl = max(len(), b.len());
142         r.resize(nl + 1);
143         for (int i = 0; i < nl; i++) {
144             if (i < len())
145                 r.v[i] += v[i];
146             if (i < b.len())
147                 r.v[i] += b.v[i];
148             if (r.v[i] >= BIGMOD) {
149                 r.v[i + 1] += r.v[i] / BIGMOD;
150                 r.v[i] %= BIGMOD;
151             }
152         }
153         r.n();
154         return r;
155     }
156
157     bigint operator-(const bigint& b) const {
158         if (s == -1)
159             return -(*this) - (-b);
160         if (b.s == -1)
161             return (*this) + (-b);
162         if ((*this) < b)
163             return -(b - (*this));
164         bigint r;
165         r.resize(len());
166         for (int i = 0; i < len(); i++) {
167             r.v[i] += v[i];
168             if (i < b.len())
169                 r.v[i] -= b.v[i];
170             if (r.v[i] < 0) {
171                 r.v[i] += BIGMOD;
172                 r.v[i + 1]--;
173             }
174         }
175         r.n();
176         return r;
177     }
178
179     bigint operator*(const bigint& b) {
180         bigint r;
181         r.resize(len() + b.len() + 1);
182         r.s = s * b.s;
183         for (int i = 0; i < len(); i++) {
184             for (int j = 0; j < b.len(); j++) {

```

20 math/factor.cpp

```

185         if (r.v[i + j] >= BIGMOD) {
186             r.v[i + j + 1] += r.v[i + j] / BIGMOD;
187             r.v[i + j] %= BIGMOD;
188         }
189     }
190 }
191 r.n();
192 return r;
193 }
194
195 bigint operator/(const bigint& b) {
196     bigint r;
197     r.resize(max(1, len() - b.len() + 1));
198     int oriS = s;
199     bigint b2 = b; // b2 = abs(b)
200     s = b2.s = r.s = 1;
201     for (int i = r.len() - 1; i >= 0; i--) {
202         int d = 0, u = BIGMOD - 1;
203         while (d < u) {
204             int m = (d + u + 1) >> 1;
205             r.v[i] = m;
206             if ((r * b2) > (*this))
207                 u = m - 1;
208             else
209                 d = m;
210         }
211         r.v[i] = d;
212     }
213     s = oriS;
214     r.s = s * b.s;
215     r.n();
216     return r;
217 }
218
219 bigint operator%(const bigint& b) {
220     return (*this) - (*this) / b * b;
221 }
222 };

```

```

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     // < 2^32: 2, 7, 61
40     // < 3e18: 2, 3, 5, 7, 11, 13, 17, 19, 23
41     // < 2^64: 2, 325, 9375, 28178, 450775, 9780504, 1795265022
42     for (ll a = 2; a < min<ll>(8, n); ++a)
43         if (!millerRabinTest(n, a))
44             return false;
45     return true;
46}
47
48//WARNING: p is not sorted
49void factorize(ll x, vector<ll> &p) {
50     if (x == 1)
51         return;
52     if (isPrime(x)) {
53         p.push_back(x);
54         return;
55     }
56     for (ll d: {2, 3, 5})
57         if (x % d == 0) {
58             p.push_back(d);
59             factorize(x / d, p);
60             return;
61         }
62     while (true) {
63         ll x1 = rr() % (x - 1) + 1;
64         ll x2 = (mul(x1, x1, x) + 1) % x;
65         int i1 = 1, i2 = 2;
66         while (true) {
67             ll c = (x1 + x - x2) % x;
68             if (c == 0)
69                 break;
70             ll g = gcd(c, x);
71             if (g > 1) {
72                 factorize(g, p);
73                 factorize(x / g, p);
74                 return;
75             }
76             if (i1 * 2 == i2) {
77                 i1 *= 2;
78                 x1 = x2;
79             }
80             ++i2;
81             x2 = (mul(x2, x2, x) + 1) % x;
82         }
83     }
84}

```

21 math/golden_search_quad_eq.cpp 22 math/numbers.tex

```

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}
33
34vector<ld> sqrRoots(ld a, ld b, ld c) {
35    ld d = b * b - 4 * a * c;
36    if (ze(d))
37        return {-b / (2 * a)};
38    if (d < 0)
39        return {};
40    d = sqrtl(d);
41    if (ze(b)) {
42        ld x1 = -d / (2 * a);
43        ld x2 = d / (2 * a);
44        if (x1 > x2)
45            swap(x1, x2);
46        return {x1, x2};
47    }
48    ld sgn = b > 0 ? 1 : -1;
49    ld x1 = (-b - sgn * d) / (2 * a);
50    ld x2 = c / (a * x1);
51    if (x1 > x2)
52        swap(x1, x2);
53    return {x1, x2};
54}

```

- 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:

$$\begin{array}{ll} 1, 2 : & 1 \\ 45 : & 1\,134\,903\,170 \\ 46 : & 1\,836\,311\,903 \text{ (max int)} \\ 47 : & 2\,971\,215\,073 \text{ (max unsigned)} \\ 91 : & 4\,660\,046\,610\,375\,530\,309 \\ 92 : & 7\,540\,113\,804\,746\,346\,429 \text{ (max i64)} \\ 93 : & 12\,200\,160\,415\,121\,876\,738 \text{ (max unsigned i64)} \end{array}$$

- 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

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

- 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

- 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

23 math/quadratic_equation.cpp

```

1vector<ld> sqrRoots(ld a, ld b, ld c) {
2    ld d = b * b - 4 * a * c;
3    if (ze(d))
4        return {-b / (2 * a)};
5    if (d < 0)
6        return {};
7    d = sqrtl(d);
8    if (ze(b)) {
9        ld x1 = -d / (2 * a);
10       ld x2 = d / (2 * a);
11       if (x1 > x2)
12           swap(x1, x2);
13       return {x1, x2};
14    }
15    ld sgn = b > 0 ? 1 : -1;
16    ld x1 = (-b - sgn * d) / (2 * a);
17    ld x2 = c / (a * x1);
18    if (x1 > x2)
19        swap(x1, x2);
20    return {x1, x2};
21}

```

24 math/simplex.cpp

```

1namespace Simplex {
2
3ld D[maxm][maxn]; // [n+2][m+2]
4int B[maxm];
5int N[maxn];
6ld x[maxn];
7int n, m;
8
9// x >= 0, Ax <= b, c^T x -> max
10void init(int _n, int _m, ld A[][maxn], ld *b, ld *c) {
11    n = _n, m = _m;
12    forn (i, m)
13        forn (j, n)
14            D[i][j] = -A[i][j];
15    forn (i, m) {
16        D[i][n] = 1;
17        D[i][n + 1] = b[i];
18    }
19    forn (j, n) {
20        D[m][j] = c[j];
21        D[m + 1][j] = 0;
22    }
23    D[m][n + 1] = D[m][n] = D[m + 1][n + 1] = 0;
24    D[m + 1][n] = -1;
25    iota(B, B + m, n);
26    iota(N, N + n, 0);
27    N[n] = -1;
28}
29
30void pivot(int b, int nb) {
31    assert(D[b][nb] != 0);
32    ld q = 1. / -D[b][nb];
33    D[b][nb] = -1;
34    forn (i, n + 2)
35        D[b][i] *= q;
36    forn (i, m + 2) {
37        if (i == b)
38            continue;
39        ld coef = D[i][nb];
40        D[i][nb] = 0;
41        forn (j, n + 2)
42            D[i][j] += coef * D[b][j];
43    }
44    swap(B[b], N[nb]);
45}
46
47bool betterN(int f, int i, int j) {
48    if (eq(D[f][i], D[f][j]))
49        return N[i] < N[j];
50    return D[f][i] > D[f][j];
51}
52
53bool betterB(int nb, int i, int j) {
54    ld ai = D[i][n + 1] / D[i][nb];
55    ld aj = D[j][n + 1] / D[j][nb];
56    if (eq(ai, aj))
57        return B[i] < B[j];
58    return ai > aj;
59}
60
61bool simplex(int phase) {
62    int f = phase == 1 ? m : m + 1;
63    while (true) {
64        int nb = -1;
65        forn (i, n + 1) {
66            if (N[i] == -1 && phase == 1)
67                continue;
68            if (nb == -1 || betterN(f, i, nb))
69                nb = i;
70        }
71        if (D[f][nb] <= eps)
72            return phase == 1;
73        assert(nb != -1);
74
75        int b = -1;
76        forn (i, m) {
77            if (D[i][nb] >= -eps)
78                continue;
79            if (b == -1 || betterB(nb, i, b))
80                b = i;
81        }
82        if (b == -1)
83            return false;
84        pivot(b, nb);
85        if (N[nb] == -1 && phase == 2)
86            return true;
87    }
88}
89
90ld solve() {
91    int b = -1;

```

25 math/stuff.cpp

```

92     forn (i, m) {
93         if (b == -1 || D[i][n + 1] < D[b][n + 1])
94             b = i;
95     }
96     assert(b != -1);
97     if (D[b][n + 1] < -eps) {
98         pivot(b, n);
99         if (!simplex(2) || D[m + 1][n + 1] < -eps)
100             return -infl;
101     }
102     if (!simplex(1))
103         return infl;
104
105     forn (i, n)
106         x[i] = 0;
107     forn (i, m)
108         if (B[i] < n)
109             x[B[i]] = D[i][n + 1];
110
111     return D[m][n + 1];
112 }
113
114 } //Simplex

```

```

1const int M = 1e6;
2int phi[M];
3void calcPhi() {
4     for (int i = 1; i < M; ++i)
5         phi[i] = i;
6     for (int j = 1; j < M; ++j)
7         for (int i = 2 * j; i < M; i += j)
8             phi[i] -= phi[j];
9}
10int inv[M];
11void calcInv() {
12     inv[1] = 1;
13     for (int i = 2; i < M; ++i) {
14         inv[i] = mul(sub(0, mod / i), inv[mod % i]);
15         assert(mul(i, inv[i]) == 1);
16     }
17}
18int gcd(int a, int b, int &x, int &y) {
19     if (a == 0) {
20         x = 0, y = 1;
21         return b;
22     }
23     int x1, y1;
24     int g = gcd(b % a, a, x1, y1);
25     x = y1 - x1 * (b / a);
26     y = x1;
27     assert(a * x + b * y == g);
28     return g;
29}
30int crt(int mod1, int mod2, int rem1, int rem2) {
31     int r = (rem2 - (rem1 % mod2) + mod2) % mod2;
32     int x, y;
33     int g = gcd(mod1, mod2, x, y);
34     assert(r % g == 0);
35
36     x %= mod2;
37     if (x < 0)
38         x += mod2;
39
40     int ans = (x * (r / g)) % mod2;
41     ans = ans * mod1 + rem1;
42
43     assert(ans % mod1 == rem1);
44     assert(ans % mod2 == rem2);
45     return ans;
46}
47
48// primes to N
49const ll n = 1000000000000LL;
50const ll L = 1000000;
51int small[L+1];
52ll large[L+1];
53void calc_pi() {
54     for (int i = 1; i <= L; ++i) {
55         small[i] = i-1;
56         large[i] = n / i - 1;
57     }
58     for (ll p = 2; p <= L; ++p) {
59         if (small[p] == small[p-1]) continue;
60         int cntp = small[p-1];
61         ll p2 = p*p;
62         ll np = n / p;
63         for (int i = 1; i <= min(L, n / p2); ++i) {
64             ll x = np / i;
65             if (x <= L) {
66                 large[i] -= small[x] - cntp;
67             } else {
68                 large[i] -= large[p*i] - cntp;
69             }
70         }
71         for (int i = L; i >= p2; --i) {
72             small[i] -= small[i/p] - cntp;
73         }
74     }
75}
76ll pi(ll x) {
77     if (x > L) return small[n/x];
78     else return large[x];
79}
80
81int main() {
82     calcPhi();
83     assert(phi[30] == 1 * 2 * 4);
84     calcInv();
85     int x, y;
86     gcd(3, 5, x, y);
87     gcd(15, 10, x, y);
88     crt(15, 13, 2, 5);
89     crt(17, 3, 15, 2);
90     return 0;
91}

```


26 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}

```

27 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("ababcbab") << '\n'; // 5
75     test();
76}

```

28 strings/eertree.cpp

```

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

```

29 strings/hashe.cpp

```

1 #define forn(i, n) for (int i = 0; i < (int)(n); i++)
2 #define sz(a) (int)(a).size()
3
4 typedef long long ll;
5 typedef unsigned long long ull;
6
7 struct num {
8     static const int MA = 1e9 + 7, MB = 1e9 + 9;
9
10    int a, b;
11
12    num() {}
13    num(int x) : a(x), b(x) {}
14    num(int a, int b) : a(a), b(b) {}
15
16    num operator + (const num &x) const { return num((a + x.a) %
    ↪ MA, (b + x.b) % MB); }
17    num operator - (const num &x) const { return num((a + MA -
    ↪ x.a) % MA, (b + MB - x.b) % MB); }
18    num operator * (int x) const { return num(((ll)a * x) % MA,
    ↪ ((ll)b * x) % MB); }
19    num operator * (const num &x) const { return num(((ll)a *
    ↪ x.a) % MA, ((ll)b * x.b) % MB); }
20    bool operator == (const num &x) const { return a == x.a && b
    ↪ == x.b; }
21
22    explicit operator ll () const { return (ll)a * MB + b + 1; }
    ↪ // > 0
23 };
24
25 template <class hash_t>
26 struct StrComparator {
27     static const int P;
28     static vector<hash_t> deg;
29
30     int n;
31     const char *s;
32     hash_t *h;
33
34     StrComparator(int n, const char *s) : n(n), s(s) {
35         h = new hash_t[n + 1];
36         h[0] = 0;
37         forn(i, n)
38             h[i + 1] = h[i] * P + s[i];
39         deg.reserve(n);
40         while (sz(deg) <= n)
41             deg.push_back(*deg.rbegin() * P);
42     }
43
44     hash_t substr(int i, int len) const { return h[i + len] -
    ↪ h[i] * deg[len]; }
45
46     int lcp(int i, int j) {
47         int L = 0, R = n - max(i, j);
48         while (L < R) {
49             int M = (L + R + 1) / 2;
50             if (substr(i, M) == substr(j, M))
51                 L = M;
52             else
53                 R = M - 1;
54         }
55         return L;
56     }
57
58     int cmp(int a, int b) {
59         int LEN = n - max(a, b), L = lcp(a, b);
60         return L < LEN ? (int)s[a + L] - s[b + L] : b - a;
61     }
62
63     bool operator() (int i, int j) { return cmp(i, j) < 0; }
64 };
65 template <class hash_t> vector<hash_t>
    ↪ StrComparator<hash_t>::deg(1, hash_t(1));
66 template <class hash_t> const int StrComparator<hash_t>::P =
    ↪ max(239, rand());
67
68 // StrComparator<num> h(n, s);
69
70 /**
71  * Usage:
72  * StrComparator<num> h(length, s); // int length, char *s
73  * h.substr(0, 3) == h.substr(1, 3); // сравнение на равенство
    ↪ подстроки за O(1)
74  * h.cmp(2, 3); // сравнение на больше-меньше суффиксов за
    ↪ O(log n)
75  * int p[n]; forn(i, n) p[i] = i;
76  * sort(p, p + n, h); // сортировать суффиксы, суф.массив за
    ↪ O(n log^2 n)
77  */
78

```

30 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}

```

31 strings/ukkonen.cpp

```

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

```

32 structures/2d_segtree.cpp

```

92     bad = newBad;
93 }
94
95 void addCharOnPos(int id) {
96     Node *bad = 0;
97     while (!canGo(at(id))) {
98         if (cur->r != pos) {
99             splitNode();
100             fixLink(bad, cur);
101             bad = cur;
102         } else {
103             fixLink(bad, 0);
104         }
105         addLeaf(id);
106         goUp();
107     }
108     fixLink(bad, 0);
109     goDown(id, id + 1);
110 }
111
112 int cnt(Node *u, int ml) {
113     if (!u)
114         return 0;
115     int res = min(ml, u->r) - u->l;
116     forn (i, alpha)
117         res += cnt(u->to[i], ml);
118     return res;
119 }
120
121 void build(int l) {
122     init();
123     forn (i, l)
124         addCharOnPos(i);
125 }
126};

```

```

1// 2-D Segment tree
2#define N 1000
3#define M 1000
4int t[4 * N][4 * M] = { 0 }; // tree
5int a[N][M] = { 0 }; // n*m matrix
6int n, m;
7void buildy(int idx, int lx, int rx, int idy = 1, int ly = 0,
8            int ry = m - 1) // id starts from 1
9{ // [ly,ry]
10    if (ly == ry) {
11        if (lx == rx) {
12            t[idx][idy] = a[lx][ly];
13        } else {
14            t[idx][idy] = t[2 * idx][idy] + t[2 * idx + 1][idy];
15        }
16    } else {
17        int my = ly + (ry - ly) / 2;
18        buildy(idx, lx, rx, 2 * idy, ly, my);
19        buildy(idx, lx, rx, 2 * idy + 1, my + 1, ry);
20        t[idx][idy] = t[idx][2 * idy] + t[idx][2 * idy + 1];
21    }
22}
23
24void buildx(
25    int idx = 1, int lx = 0, int rx = n - 1) // id starts from 1
26{ // call buildx(), [l,r]
27    if (lx != rx) {
28        int mx = lx + (rx - lx) / 2;
29        buildx(2 * idx, lx, mx); // [l,mx], left child
30        buildx(
31            2 * idx + 1, mx + 1, rx); // [mx+1,r], right child
32    }
33    buildy(idx, lx, rx, 1, 0, m - 1);
34}
35
36ll queryy(int idx, int idy, int tly, int tryy, int ly, int ry) {
37    if (ly > ry || ly > tryy || ry < tly) // no overlap
38    {
39        return 0;
40    }
41    if (ly <= tly && tryy <= ry) // complete overlap
42    {
43        return t[idx][idy];
44    }
45    int tmy = tly + (tryy - tly) / 2;
46    return queryy(idx, 2 * idy, tly, tmy, ly, min(tmy, ry))
47        + queryy(idx, 2 * idy + 1, tmy + 1, tryy,
48            max(ly, tmy + 1), ry);
49}
50
51ll queryx(int lx, int rx, int ly, int ry, int idx = 1,
52          int tlx = 0,
53          int trx = n - 1) // query in rectangle (lx,ly) to (rx,ry)
54{ // call queryx(x1,x2,y1,y2) where
55    // (x1,y1)=upper-left,(x2,y2)=lower-right coord.
56    if (lx > rx || lx > trx || rx < tlx) // no overlap
57    {
58        return 0;
59    }
60    if (lx <= tlx && trx <= rx) // complete overlap
61    {
62        return queryy(idx, 1, 0, m - 1, ly, ry);
63    }
64    int tmx = tlx + (trx - tlx) / 2;
65    return queryx(lx, min(rx, tmx), ly, ry, 2 * idx, tlx, tmx)
66        + queryx(max(lx, tmx + 1), rx, ly, ry, 2 * idx + 1,
67            tmx + 1, trx);
68}
69
70}
71
72void updatey(int x, int y, int value, int idx, int lx, int rx,
73            int idy = 1, int ly = 0, int ry = m - 1) {
74    if (ly == ry) {
75        if (lx == rx) {
76            t[idx][idy] = value;
77        } else {
78            t[idx][idy] = t[2 * idx][idy] + t[2 * idx + 1][idy];
79        }
80    } else {
81        int my = ly + (ry - ly) / 2;
82        if (y <= my) {
83            updatey(x, y, value, idx, lx, rx, 2 * idy, ly, my);
84        } else {
85            updatey(x, y, value, idx, lx, rx, 2 * idy + 1,
86                my + 1, ry);
87        }
88    }
89    t[idx][idy] = t[idx][2 * idy] + t[idx][2 * idy + 1];
90}
91}

```

33 structures/centroids.cpp

```

92
93 void updatex(int x, int y, int value, int idx = 1, int lx = 0,
94             int rx = n - 1) // call updatex(x,y,value) if a[x][y]
95                             // changes to value.
96 {
97     if (lx != rx) {
98         int mx = lx + (rx - lx) / 2;
99         if (x <= mx) {
100             updatex(x, y, value, 2 * idx, lx, mx);
101         } else {
102             updatex(x, y, value, 2 * idx + 1, mx + 1, rx);
103         }
104     }
105
106     updatey(x, y, value, idx, lx, rx, 1, 0, m - 1);
107 }
108 // assign values to n( no. of rows ), m( no. of columns ), array
109 // a! And then call build!

1 const int maxn = 100100;
2 const int LG = 18; // 2*maxn <= 2^LG
3
4 vector<int> g[LG][maxn];
5 int rt[LG][maxn];
6 int from[LG][maxn];
7
8 namespace Cenroids {
9
10 int D;
11 int cnt[maxn];
12 int CENTER, BEST;
13
14 void pre(int u, int prev = -1) {
15     cnt[u] = 1;
16     for (int v: g[D][u]) {
17         if (v == prev)
18             continue;
19         pre(v, u);
20         cnt[u] += cnt[v];
21     }
22 }
23
24 void findCenter(int u, int prev = -1, int up = 0) {
25     int worst = up;
26     for (int v: g[D][u]) {
27         if (v == prev)
28             continue;
29         findCenter(v, u, up + cnt[u] - cnt[v]);
30         worst = max(worst, cnt[v]);
31     }
32     if (worst < BEST) {
33         CENTER = u;
34         BEST = worst;
35     }
36 }
37
38 void markAll(int u, int prev = -1, int subtree = -1) {
39     rt[D][u] = CENTER;
40     from[D][u] = subtree;
41     for (int v: g[D][u]) {
42         if (v == prev)
43             continue;
44         g[D + 1][u].push_back(v);
45         g[D + 1][v].push_back(u);
46         if (subtree == -1)
47             markAll(v, u, v);
48         else
49             markAll(v, u, subtree);
50     }
51 }
52
53 void decompose(int u, int depth = 0) {
54     D = depth;
55     pre(u);
56     CENTER = -1, BEST = 1e9;
57     findCenter(u);
58     assert(CENTER != -1);
59     u = CENTER;
60     markAll(u);
61     D = depth + 1;
62     for (int v: g[D][u]) {
63         auto it = find(g[D][v].begin(), g[D][v].end(), u);
64         assert(it != g[D][v].end());
65         g[D][v].erase(it);
66     }
67     for (int v: g[D][u])
68         decompose(v, depth + 1);
69 }
70
71 };

```

34 structures/fenwick.cpp

```

1 //BEGIN ALGO
2 struct Fenwick {
3     int *t;
4     int n;
5
6     Fenwick(int *a, int len): n(len) {
7         t = new int[n];
8         memset(t, 0, sizeof(int) * n);
9         for (int i = 0; i < n; ++i) {
10             inc(i, a[i]);
11         }
12     }
13
14     ~Fenwick() {
15         delete[] t;
16     }
17 };

```

```

16     }
17
18     void inc(int l, int delta) {
19         for (int i = l; i < n; i = (i | (i + 1))) {
20             t[i] += delta;
21         }
22     }
23
24     int sum(int r) {
25         int result = 0;
26         for (int i = r; i >= 0; i = (i & (i + 1)) - 1) {
27             result += t[i];
28         }
29         return result;
30     }
31
32     int sum(int l, int r) {
33         return sum(r) - sum(l - 1);
34     }
35 };
36
37 //END ALGO

```

35 structures/heavy_light.cpp

```

1 int n;
2 vi e[maxn];
3
4 namespace HLD {
5 int p[maxn], s[maxn], h[maxn], root[maxn];
6 Rmq rmq[maxn];
7
8 void dfs1(int v, int anc) {
9     s[v] = 1;
10    if (anc != -1) e[v].erase(find(all(e[v]), anc));
11    for (int to: e[v]) {
12        p[to] = v;
13        h[to] = h[v] + 1;
14        dfs1(to, v);
15        s[v] += s[to];
16    }
17 }
18
19 void dfs2(int v, int rt) {
20     root[v] = rt;
21     if (e[v].empty()) {
22         rmq[rt] = Rmq(h[v] - h[rt] + 1);
23         return;
24     }
25     int mxv = e[v][0];
26     for (int to: e[v]) {
27         if (s[to] > s[mxv]) mxv = to;
28     }
29     for (int to: e[v]) {
30         dfs2(to, to == mxv ? rt : to);
31     }
32 }
33
34 int get(int u, int v) {
35     int res = 0;
36     int t;
37     while (root[u] != root[v]) {
38         if (h[root[u]] > h[root[v]]) {
39             t = rmq[root[u]].get(0, h[u] - h[root[u]] + 1);
40             u = p[root[u]];
41         } else {
42             t = rmq[root[v]].get(0, h[v] - h[root[v]] + 1);
43             v = p[root[v]];
44         }
45         res = max(res, t);
46     }
47     int r = root[u];
48     if (h[u] > h[v]) {
49         t = rmq[r].get(h[v] - h[r], h[u] - h[r] + 1);
50     } else {
51         t = rmq[r].get(h[u] - h[r], h[v] - h[r] + 1);
52     }
53     return max(res, t);
54 }
55
56 void put(int v, int x) {
57     rmq[root[v]].put(h[v] - h[root[v]], x);
58 }
59
60 void init() {
61     const int ROOT = 0;
62     h[0] = 0;
63     dfs1(ROOT, -1);
64     dfs2(ROOT, ROOT);
65 }
66 } // namespace HLD

```

36 structures/ordered_set.cpp

```

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     __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 }

```

37 structures/splay.cpp

```

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

```

```

92     }
93     splay(t);
94     return t;
95 }
96
97 node *insert(node *t, int x) {
98     return insert(t, new node(x));
99 }

```

38 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     for (i = 1000000; i--;) {
68         int x = rand();
69         t = fast_insert(t, x);
70     }
71 }

```


39 zzz_narfm/graph/dinic.cpp

```

1#define next botvinnik
2const int maxn = 505, maxm = 20005;
3int to[maxm], next[maxm], c[maxm], f[maxm];
4int esz = 0;
5int head[maxn];
6
7void addEdge(int a, int b, int cap) {
8    to[esz] = b; c[esz] = cap; f[esz] = 0; next[esz] = head[a];
9    head[a] = esz++;
10   to[esz] = a; c[esz] = 0; f[esz] = 0; next[esz] = head[b];
11   head[b] = esz++;
12}
13
14int n, m;
15int source, sink;
16int bound = 1;
17int ptr[maxn], dist[maxn];
18int dfs(int v, int maxf) {
19    if (!maxf)
20        return 0;
21    if (v == sink)
22        return maxf;
23    int& e = ptr[v];
24    for (; e != -1; e = next[e]) {
25        if (dist[to[e]] != dist[v] + 1 || c[e] - f[e] < bound)
26            continue;
27        int flow = dfs(to[e], min(maxf, c[e] - f[e]));
28        if (flow) {
29            f[e] += flow;
30            f[e ^ 1] -= flow;
31            return flow;
32        }
33    }
34    return 0;
35}
36int q[maxn];
37bool bfs() {
38    memset(dist, -1, sizeof dist);
39    dist[source] = 0;
40    int ql = 0, qr = 0;
41    q[qr++] = source;
42    while (ql < qr) {
43        int v = q[ql++];
44        for (int e = head[v]; e != -1; e = next[e]) {
45            if (dist[to[e]] == -1 && c[e] - f[e] >= bound) {
46                dist[to[e]] = dist[v] + 1;
47                q[qr++] = to[e];
48            }
49        }
50    }
51    return dist[sink] != -1;
52}
53long long maxFlow() {
54    long long ret = 0;
55    while (bfs()) {
56        for (int i = 0; i < n; ++i)
57            ptr[i] = head[i];
58        while (int add = dfs(source, 1 << 30))
59            ret += add;
60    }
61    return ret;
62}

```

40 zzz_narfm/graph/ford-falkerson.cpp

```

1struct Edge {
2    int to, f, c;
3    Edge() {}
4    Edge(int to, int f, int c) : to(to), f(f), c(c) {}
5};
6
7const int maxn;
8
9vector<Edge> edgelist;
10vector<vector<int>> edge;
11int p[maxn], used[maxn];
12int source, sink;
13int bound, q = 0;
14
15int dfs(int v, int w) {
16    p[v] = w;
17    used[v] = q;
18    if (v == sink)
19        return 1 << 30;
20    for (size_t i = 0; i < edge[v].size(); ++i) {
21        int e = edge[v][i];
22        int u = edgelist[e].to;
23        if (used[u] != q
24            && edgelist[e].c - edgelist[e].f >= bound) {
25            int d = dfs(u, e);
26            if (d)
27                return min(d, edgelist[e].c - edgelist[e].f);
28        }
29    }
30    return 0;
31}
32
33foreach (a->b)
34    edge[a - 1].push_back(edgelist.size());
35edgelist.push_back(Edge(b - 1, 0, c));
36
37int ans = 0;
38p[source] = -1;
39memset(used, -1, sizeof used);
40q = -1;
41for (bound = 1 << 30; bound > 0;) {
42    ++q;
43    int flow = dfs(source, -1);
44    if (!flow) {
45        bound >>= 1;
46        continue;
47    }
48    ans += flow;
49    for (int cur = p[sink]; cur != -1;
50         cur = p[edgelist[cur ^ 1].to]) {
51        edgelist[cur].f += flow;
52        edgelist[cur ^ 1].f -= flow;
53    }
54}
55
56// Another try:
57
58nt to[maxe], cap[maxe], nxt[maxe];
59int ecnt = 0;
60int head[maxv];
61
62inline void addEdge(int a, int b, int c) {
63    nxt[ecnt] = head[a]; to[ecnt] = b; cap[ecnt] = c;
64    head[a] = ecnt++;
65    nxt[ecnt] = head[b]; to[ecnt] = a; cap[ecnt] = 0;
66    head[b] = ecnt++;
67}
68
69int source, sink;
70int used[maxv];
71int qused = 1;
72
73int dfs(int v, int maxf) {
74    if (v == sink || !maxf)
75        return maxf;
76    used[v] = qused;
77    for (int e = head[v]; e != -1; e = nxt[e]) {
78        int u = to[e], c = min(cap[e], maxf);
79        if (used[u] == qused || !c)
80            continue;
81        int f = dfs(u, c);
82        if (f) {
83            cap[e] -= f;
84            cap[e ^ 1] += f;
85            return f;
86        }
87    }
88    return 0;
89}

```

```

90
91 int maxFlow() {
92     memset(used, 0, sizeof used);
93     int ret = 0, d;
94     while (d = dfs(source, 1 << 30)) {
95         ret += d;
96         ++qused;
97     }
98     return ret;
99 }

```

41 zzz_narfm/graph/lca.cpp

```

1 // минимум на пути через lca на двоичных подъёмах
2
3 inline bool ancestor(int a, int b) {
4     return in[b] >= in[a] && in[b] <= out[a];
5 }
6
7 int climb(int to, int v) {
8     int ans = 1 << 30;
9     for (int i = maxk - 1; v != to;)
10         if (ancestor(to, p[i][v])) {
11             ans = min(ans, w[i][v]);
12             v = p[i][v];
13         } else
14             --i;
15     return ans;
16 }
17
18 int getans(int a, int b) {
19     if (ancestor(a, b))
20         return climb(a, b); // lca=a
21     if (ancestor(b, a))
22         return climb(b, a); // lca=b
23     int lca = a;
24     for (int i = maxk - 1; !ancestor(p[0][lca], b);)
25         if (ancestor(p[i][lca], b))
26             --i;
27         else
28             lca = p[i][lca];
29     lca = p[0][lca];
30     return min(climb(lca, a), climb(lca, b));
31 }
32
33 // через RMQ
34
35 pair<int, int> euler[maxn * 2];
36 int fst[maxn];
37 int psz = 0;
38 int lca(int a, int b) {
39     if (fst[a] > fst[b])
40         swap(a, b);
41     pair<int, int> m = getMin(fst[a], fst[b]);
42     return m.second;
43 }
44
45 vector<int> edge[maxn];
46 void dfs(int v, int h) {
47     fst[v] = psz;
48     euler[psz++] = make_pair(h, v);
49     for (size_t i = 0; i < edge[v].size(); ++i) {
50         dfs(edge[v][i], h + 1);
51         euler[psz++] = make_pair(h, v);
52     }
53 }

```

42 zzz_narfm/graph/mincost.cpp

```

1 const int maxn = 205, maxm = 10005, inf = 1 << 30;
2 #define next youSuddenlyVomit
3
4 int from[maxn], to[maxn], c[maxn], f[maxn], cost[maxn],
5   next[maxn], id[maxn];
6 int head[maxn];
7 int esz = 0;
8
9 void addEdge(int a, int b, int cst, int i) {
10     from[esz] = a; to[esz] = b; c[esz] = 1; f[esz] = 0;
11     cost[esz] = cst; next[esz] = head[a]; id[esz] = i;
12     head[a] = esz++;
13
14     from[esz] = b; to[esz] = a; c[esz] = 0; f[esz] = 0;
15     cost[esz] = -cst; next[esz] = head[b]; id[esz] = i;
16     head[b] = esz++;
17 }
18
19 int n, m;
20 int source, sink;
21
22 int range[maxn], p[maxn];
23
24 vector<int> ansPath[105];
25 vector<int> path;
26
27 int main() {
28     freopen("brides.in", "r", stdin);
29     freopen("brides.out", "w", stdout);
30
31     int k;
32     cin >> n >> m >> k;
33     for (int i = 0; i < m; ++i) {
34         int a, b, c;
35         cin >> a >> b >> c;
36         addEdge(a - 1, b - 1, c, i);
37         addEdge(b - 1, a - 1, c, i);
38     }
39     source = 0;
40     sink = n - 1;
41
42     for (int brother = 0; brother < k; ++brother) {
43         for (int i = 0; i < n; ++i)
44             range[i] = inf;
45         range[source] = 0;
46         bool need = true;
47         while (need) {
48             need = false;
49             for (int e = 0; e < esz; ++e)
50                 if (f[e] < c[e] && range[from[e]] != inf
51                     && range[to[e]]
52                     > range[from[e]] + cost[e]) {
53                     need = true;
54                     range[to[e]] = range[from[e]] + cost[e];
55                     p[to[e]] = e;
56                 }
57         }
58
59         // минимост. 2.
60
61         if (range[sink] == inf) {
62             cout << -1 << endl;
63             return 0;
64         }
65
66         int flow = inf;
67         for (int u = sink; u != source; u = from[p[u]])
68             flow = min(flow, c[p[u]] - f[p[u]]);
69         for (int u = sink; u != source; u = from[p[u]]) {
70             f[p[u]] += flow;
71             f[p[u] ^ 1] -= flow;
72         }
73     }
74
75     int ansCost = 0;
76     for (int brother = 0; brother < k; ++brother) {
77         int u = sink;
78         while (u != source) {
79             for (int e = 0; e < esz; ++e)
80                 if (to[e] == u && f[e] > 0) {
81                     ansCost += cost[e];
82                     f[e]--;
83                     f[e ^ 1]++;
84                     ansPath[brother].push_back(id[e]);
85                     u = from[e];
86                     break;
87                 }
88         }
89         reverse(
90             ansPath[brother].begin(), ansPath[brother].end());
91     }
92
93     cout.precision(10);
94     cout << fixed << double(ansCost) / k << endl;
95     for (int i = 0; i < k; ++i) {
96         cout << ansPath[i].size() << ' ';
97         for (size_t j = 0; j < ansPath[i].size(); ++j)
98             cout << ansPath[i][j] + 1 << ' ';
99         cout << endl;
100     }
101
102     return 0;
103 }

```

43 zzz_narfm/misc/convex-hull.cpp

```

1 typedef long long coord;
2 struct point {
3     coord x, y;
4     point() {}
5     point(coord x, coord y)
6         : x(x)
7         , y(y) {}
8     point(point a, point b)
9         : x(b.x - a.x)
10        , y(b.y - a.y) {}
11};
12 inline coord operator*(point a, point b) {
13     return a.x * b.x + a.y * b.y;
14 }
15 inline coord operator%(point a, point b) {
16     return a.x * b.y - a.y * b.x;
17 }
18 inline coord operator==(point a, point b) {
19     // Warning: consider using epsilon!
20     return a.x == b.x && a.y == b.y;
21 }
22
23 inline bool as_pair(const point& a, const point& b) {
24     // Warning: consider using epsilon!
25     return (a.x == b.x ? a.y < b.y : a.x < b.x);
26 }
27 struct by_angle {
28     by_angle(const point& corner)
29         : corner(corner) {}
30     inline bool operator()(const point& a, const point& b) {
31         point ca(corner, a);
32         point cb(corner, b);
33         // Warning: consider using epsilon!
34         return ca % cb > 0
35            || (ca % cb == 0
36                && point(a, corner) * point(a, b) < 0);
37     }
38     point corner;
39 };
40
41 vector<point> hull(vector<point> p) {
42     sort(p.begin(), p.end(), as_pair);
43     p.erase(unique(p.begin(), p.end(), p.end()), p.end());
44     sort(p.begin() + 1, p.end(), by_angle(p[0]));
45
46     vector<point> ret;
47     int sz = 0;
48     for (size_t i = 0; i < p.size(); ++i) {
49         // Warning: consider using epsilon!
50         while (sz > 1
51             && point(ret[sz - 2], ret[sz - 1])
52                % point(ret[sz - 1], p[i])
53                <= 0) {
54             ret.pop_back();
55             --sz;
56         }
57         ret.push_back(p[i]);
58         ++sz;
59     }
60
61     return ret;
62 }

```

44 zzz_narfm/misc/gauss.cpp

```

1 int gauss(vector<vector<ld>> v, vector<ld>& ret) {
2     int n = v.size();
3     int m = n;
4     vector<int> p(m), dist(m, 0);
5     for (int i = 0; i < m; ++i)
6         p[i] = i;
7     for (int row = 0, col = 0; row < n && col < m; ++col) {
8         int sr = row, sc = col;
9         for (int i = row; i < n; ++i)
10             for (int j = col; j < m; ++j) {
11                 if (abs(v[i][j]) > abs(v[sr][sc])) {
12                     sr = i;
13                     sc = j;
14                 }
15             }
16         if (abs(v[sr][sc]) < eps)
17             break;
18         swap(v[row], v[sr]);
19         for (int i = 0; i < n; ++i)
20             swap(v[i][col], v[i][sc]);
21         swap(p[col], p[sc]);
22         dist[col] = 1;
23         for (int i = 0; i < n; ++i)
24             if (i != row) {
25                 ld c = v[i][col] / v[row][col];
26                 for (int j = col; j <= m; ++j)
27                     v[i][j] -= v[row][j] * c;
28             }
29         ++row;
30     }
31     ret.assign(m, 0);
32     for (int i = 0; i < m; ++i)
33         if (dist[p[i]])
34             ret[i] = v[p[i]][m] / v[p[i]][p[i]];
35     for (int i = 0; i < m; ++i) {
36         ld sum = 0;
37         for (int j = 0; j < m; ++j)
38             sum += ret[j] * v[i][p[j]];
39         if (abs(sum - v[i][m]) > eps)
40             return 0;
41     }
42     for (int i = 0; i < m; ++i)
43         if (!dist[i])
44             return -1;
45     return 1;
46 }

```

45 zzz_narfm/strings/ahocorasick.cpp

```

1const int triesize, alph;
2struct node {
3    int p, pch;
4    int link, term, upterm;
5    int next[alph], go[alph];
6};
7node t[triesize];
8int tsz = 0;
9
10int mkNode(int p, int pch) {
11    t[tsz].p = p;
12    t[tsz].pch = pch;
13    t[tsz].link = t[tsz].upterm = -1;
14    memset(t[tsz].next, -1, sizeof t[tsz].next);
15    memset(t[tsz].go, -1, sizeof t[tsz].go);
16    t[tsz].term = 0;
17    return tsz++;
18}
19
20void addWord(string s) {
21    int v = 0;
22    for (size_t i = 0; i < s.size(); ++i) {
23        int c = s[i] - '0';
24        if (t[v].next[c] == -1)
25            t[v].next[c] = mkNode(v, c);
26        v = t[v].next[c];
27    }
28    t[v].term = 1;
29}
30
31int q[triesize];
32void bfs() {
33    int ql = 0, qr = 0;
34    q[qr++] = 0;
35    t[0].link = 0;
36    t[0].upterm = 0;
37    for (int i = 0; i < alph; ++i)
38        t[0].go[i] = max(t[0].next[i], 0);
39    while (ql < qr) {
40        int v = q[ql++];
41        for (int i = 0; i < alph; ++i) {
42            int u = t[v].next[i];
43            if (u == -1)
44                continue;
45            t[u].link = (v ? t[t[v].link].go[i] : 0);
46            t[u].upterm
47                = (t[t[u].link].upterm || t[u].term ? 1 : 0);
48            for (int j = 0; j < alph; ++j)
49                t[u].go[j]
50                    = (t[u].next[j] == -1 ? t[t[u].link].go[j]
51                     : t[u].next[j]);
52            q[qr++] = u;
53        }
54    }
55}

```

46 zzz_narfm/strings/prefix_fun.cpp

```

1vector<int> pFunc(string s) {
2    int n = s.size();
3    vector<int> ret(n);
4    ret[0] = 0;
5    for (int i = 1; i < n; ++i) {
6        int t = ret[i - 1];
7        while (t && s[t] != s[i])
8            t = ret[t - 1];
9        if (s[t] == s[i])
10            t++;
11        ret[i] = t;
12    }
13    return ret;
14}

```

47 zzz_narfm/strings/suffix_array.cpp

```

1void buildSuffixArray(int* src, int n, int* p) {
2    static int s[maxn], scale[maxn], cnt[maxn], color[maxn],
3        start[maxn], pp[maxn], cc[maxn];
4    memcpy(s, src, sizeof(int) * n);
5    memcpy(scale, src, sizeof(int) * n);
6    sort(scale, scale + n);
7
8    int csz = int(unique(scale, scale + n) - scale);
9    for (int i = 0; i < n; ++i)
10        s[i] = int(
11            lower_bound(scale, scale + csz, s[i]) - scale + 1);
12    s[n++] = 0;
13    csz++;
14
15    memset(cnt, 0, sizeof cnt);
16    for (int i = 0; i < n; ++i)
17        cnt[s[i]]++;
18    start[0] = 0;
19    for (int i = 1; i < csz; ++i)
20        start[i] = start[i - 1] + cnt[i - 1];
21    for (int i = 0; i < n; ++i)
22        p[start[s[i]]++] = i;
23    color[p[0]] = 0;
24    for (int i = 1; i < n; ++i)
25        color[p[i]] = color[p[i - 1]]
26            + (s[p[i - 1]] == s[p[i]] ? 0 : 1);
27
28    for (int k = 1; k < n; k <= 1) {
29        memset(cnt, 0, sizeof(int) * n);
30        for (int i = 0; i < n; ++i)
31            cnt[color[i]]++;
32        start[0] = 0;
33        for (int i = 1; i < n; ++i)
34            start[i] = start[i - 1] + cnt[i - 1];
35
36        for (int i = 0; i < n; ++i)
37            p[i] = (p[i] - k + n) % n;
38        for (int i = 0; i < n; ++i)
39            pp[start[color[p[i]]]++] = p[i];
40        memcpy(p, pp, sizeof(int) * n);
41        cc[p[0]] = 0;
42        for (int i = 1; i < n; ++i)
43            cc[p[i]] = cc[p[i - 1]]
44                + (color[p[i]] == color[p[i - 1]]
45                  && color[(p[i] + k) % n]
46                    == color[(p[i - 1] + k) % n]
47                    ? 0
48                    : 1);
49        memcpy(color, cc, sizeof(int) * n);
50    }
51    for (int i = 0; i + 1 < n; ++i)
52        p[i] = p[i + 1];
53}
54
55void buildLcp(int* s, int* sa, int n, int* lcp) {
56    static int p[maxn];
57    for (int i = 0; i < n; ++i)
58        p[sa[i]] = i;
59    for (int i = 0; i < n; ++i) {
60        if (p[i] + 1 == n)
61            continue;
62        int j = (i ? max(0, lcp[p[i - 1]] - 1) : 0);
63        while (sa[p[i]] + j < n && sa[p[i] + 1] + j < n
64              && s[sa[p[i]] + j] == s[sa[p[i] + 1] + j])
65            ++j;
66        lcp[p[i]] = j;
67    }
68}

```

48 zzz_narfm/strings/z_function.cpp

```

1int z[maxn];
2void getZ(const string& s) {
3    int n = s.size();
4    int l, r;
5    l = r = 0;
6    for (int i = 1; i < n; ++i) {
7        z[i] = 0;
8        if (i < r)
9            z[i] = min(r - i, z[i - 1]);
10        while (i + z[i] < n && s[i + z[i]] == s[z[i]])
11            z[i]++;
12        if (i + z[i] > r) {
13            l = i;
14            r = i + z[i];
15        }
16    }
17    z[0] = n;
18}

```

49 zzz_narfm/structures/segtree_assign.cpp

```

1struct node {
2    long long fill, sum;
3    int flag;
4};
5
6const int hfsz, treesz = hfsz << 1;
7
8node tree[treesz];
9
10void push(int i) {
11    if (i >= hfsz - 1 || !tree[i].flag)
12        return;
13    int left = i * 2 + 1, right = left + 1;
14    tree[left].flag = tree[right].flag = 1;
15    tree[left].fill = tree[right].fill = tree[i].fill;
16    tree[left].sum = tree[right].sum = tree[i].sum / 2;
17    tree[i].flag = 0;
18}
19
20void change(
21    int i, int l, int r, int tl, int tr, long long val) {
22    if (r < tl || l > tr)
23        return;
24    if (l >= tl && r <= tr) {
25        tree[i].flag = 1;
26        tree[i].fill = val;
27        tree[i].sum = val * (r - l + 1);
28        return;
29    }
30    push(i);
31    int m = (l + r) / 2, left = i * 2 + 1, right = left + 1;
32    change(left, l, m, tl, tr, val);
33    change(right, m + 1, r, tl, tr, val);
34    tree[i].sum = tree[left].sum + tree[right].sum;
35}
36long long get(int i, int l, int r, int tl, int tr) {
37    if (r < tl || l > tr)
38        return 0;
39    if (l >= tl && r <= tr)
40        return tree[i].sum;
41    push(i);
42    int m = (l + r) / 2, left = i * 2 + 1, right = left + 1;
43    return get(left, l, m, tl, tr)
44        + get(right, m + 1, r, tl, tr);
45}

```

50 zzz_narfm/structures/segtree_lazy.cpp

```

1#define left morkva
2#define right svekolka
3long long sum[treesz];
4int left[treesz], right[treesz];
5int tsz = 0;
6void change(int& t, int l, int r, int at, long long value) {
7    if (l > at || r < at)
8        return;
9    if (t == -1)
10        t = tsz++;
11    if (l == r)
12        return void(sum[t] = value);
13    int m = (l + r) / 2;
14    change(left[t], l, m, at, value);
15    change(right[t], m + 1, r, at, value);
16    sum[t] = 0;
17    if (left[t] != -1)
18        sum[t] += sum[left[t]];
19    if (right[t] != -1)
20        sum[t] += sum[right[t]];
21}
22long long get(int t, int l, int r, int tl, int tr) {
23    if (l > tr || r < tl || t == -1)
24        return 0;
25    if (l >= tl && r <= tr)
26        return sum[t];
27    int m = (l + r) / 2;
28    return get(left[t], l, m, tl, tr)
29        + get(right[t], m + 1, r, tl, tr);
30}

```

51 zzz_narfm/structures/segtree_persist.cpp

```

1const int hfsz;
2struct node {
3    int value;
4    node *l, *r;
5    node()
6        : l(0)

```

```

7        : r(0) {}
8    node(int value)
9        : value(value)
10       , l(0)
11       , r(0) {}
12    node(int value, node* l, node* r)
13        : value(value)
14        , l(l)
15        , r(r) {}
16};
17node* setValue(node* v, int l, int r, int at, int value) {
18    if (l > at || r < at)
19        return v;
20    if (l == r)
21        return new node(value);
22    int m = (l + r) / 2;
23    node* left = setValue(v ? v->l : 0, l, m, at, value);
24    node* right = setValue(v ? v->r : 0, m + 1, r, at, value);
25    return new node(
26        (left ? left->value : 0) + (right ? right->value : 0),
27        left, right);
28}
29int getsum(node* v, int l, int r, int tl, int tr) {
30    if (l > tr || r < tl || !v)
31        return 0;
32    if (l >= tl && r <= tr)
33        return v->value;
34    int m = (l + r) / 2;
35    return getsum(v->l, l, m, tl, tr)
36        + getsum(v->r, m + 1, r, tl, tr);
37}

```

52 zzz_narfm/structures/sparse.cpp

```

1struct Sparse {
2    static const int logn = 18;
3    T st[logn][maxn];
4    int log2[maxn];
5    Sparse() {}
6    Sparse(T* src, int n) { build(src, n); }
7    void build(T* src, int n) {
8        log2[1] = 0;
9        for (int i = 2; i <= n; ++i)
10            log2[i] = log2[i - 1]
11                + ((2 << log2[i - 1]) < i ? 1 : 0);
12        memcpy(st[0], src, sizeof(T) * n);
13        for (int i = 1; i < logn; ++i)
14            for (int j = 0; j < n; ++j)
15                st[i][j]
16                    = getmin(j, min(j + (1 << i) - 1, n - 1));
17    }
18    inline int getmin(int l, int r) {
19        int p = log2[r - l + 1];
20        return min(st[p][l], st[p][r - (1 << p) + 1]);
21    }
22};

```





