

## Содержание

1	Strategy.txt	1
2	flows/dinic.cpp	2
3	flows/hungary.cpp	2
4	flows/mincost.cpp	3
5	geometry/basic3d.cpp	4
6	geometry/chan.cpp	4
7	geometry/halfplanes.cpp	5
8	geometry/polygon.cpp	6
9	geometry/polygon_tangents.cpp	6
10	geometry/primitives.cpp	7
11	graphs/edmonds_matching.cpp	8
12	graphs/euler_cycle.cpp	8
13	graphs/kuhn.cpp	9
14	graphs/min_automaton.cpp	9
15	math/factor.cpp	10
16	math/golden_search_quad_eq.cpp	10
17	math/numbers.tex	11
18	math/stuff.cpp	11
19	strings/automaton.cpp	12
20	strings/eertree.cpp	12
21	strings/hashes.cpp	13
22	strings/suffix_array.cpp	13
23	strings/ukkonen.cpp	14
24	structures/fenwick.cpp	15
25	structures/heavy_light.cpp	15
26	structures/ordered_set.cpp	16
27	structures/treap.cpp	16

## 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 `maxn`, удалить `_GLIBCXX_DEBUG`
- Вердикт может врать
- Если много тестов(>3), дописать в конец каждого
  - ↪ теста ответ, чтобы не забыть
- (WA) Потестить не только ответ, но и содержимое
  - ↪ значимых массивов, переменных
- (WA) Изменить тест так, чтобы ответ не менялся:
  - ↪ поменять координаты местами, сжать/растянуть
  - ↪ координаты, поменять `ROOT` дерева
- (WA) Подвигать размер блока в корневой или
  - ↪ битсете
- (WA) Поставить `assert`'ы, возможно написать чекер
  - ↪ с `assert`'ом
- (WA) Проверить, что программа не печатает
  - ↪ что-либо неожиданное, что должно попадать под
  - ↪ PE: `inf - 2`, не лекс. мин. решение, одинаковые
  - ↪ числа вместо разных, неправильное количество
  - ↪ чисел, пустой ответ, перечитать `output format`
- (TL) `cin -> scanf -> getchar`
- (TL) Упихать в кэш большие массивы, поменять
  - ↪ местами `for`'ы или измерения массива
- (RE) Проверить формулы на деление на 0, выход за
  - ↪ область определения(`sqrt(-eps)`, `acos(1 + eps)`)
- (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/hungary.cpp

```

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

```

## 4 flows/mincost.cpp

```

1 namespace MinCost {
2 const ll infc = 1e12;
3
4 struct Edge {
5     int to;
6     ll c, f, cost;
7
8     Edge(int to, ll c, ll cost): to(to), c(c), f(0), cost(cost) {
9         {}
10};
11
12 int N, S, T;
13 int totalFlow;
14 ll totalCost;
15 const int maxn = 505;
16 vector<Edge> edge;
17 vector<int> g[maxn];
18
19 void addEdge(int u, int v, ll c, ll cost) {
20     g[u].push_back(edge.size());
21     edge.emplace_back(v, c, cost);
22     g[v].push_back(edge.size());
23     edge.emplace_back(u, 0, -cost);
24}
25
26 ll dist[maxn];
27 int fromEdge[maxn];
28
29 bool inQueue[maxn];
30 bool fordBellman() {
31     for (i, N)
32         dist[i] = infc;
33     dist[S] = 0;
34     inQueue[S] = true;
35     vector<int> q;
36     q.push_back(S);
37     for (int ii = 0; ii < int(q.size()); ++ii) {
38         int u = q[ii];
39         inQueue[u] = false;
40         for (int e: g[u]) {
41             if (edge[e].f == edge[e].c)
42                 continue;
43             int v = edge[e].to;
44             ll nw = edge[e].cost + dist[u];
45             if (nw >= dist[v])
46                 continue;
47             dist[v] = nw;
48             fromEdge[v] = e;
49             if (!inQueue[v]) {
50                 inQueue[v] = true;
51                 q.push_back(v);
52             }
53         }
54     }
55     return dist[T] != infc;
56}
57
58 ll pot[maxn];
59 bool dikstra() {
60     typedef pair<ll, int> Pair;
61     priority_queue<Pair, vector<Pair>, greater<Pair>> q;
62     for (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     for (i, N) {
89         if (dist[i] == infc)
90             continue;
91         pot[i] += dist[i];
92     }
93     return true;
94}
95
96 bool push() {
97     //2 variants
98     //if (!fordBellman())
99     if (!dikstra())
100         return false;
101     ++totalFlow;
102     int u = T;
103     while (u != S) {
104         int e = fromEdge[u];
105         totalCost += edge[e].cost;
106         edge[e].f++;
107         edge[e ^ 1].f--;
108         u = edge[e ^ 1].to;
109     }
110     return true;
111}
112
113 //min-cost-circulation
114 ll d[maxn][maxn];
115 int dfrom[maxn][maxn];
116 int level[maxn];
117 void circulation() {
118     while (true) {
119         int q = 0;
120         fill(d[0], d[0] + N, 0);
121         for (iter, N) {
122             fill(d[iter + 1], d[iter + 1] + N, infc);
123             for (u, N)
124                 for (int e: g[u]) {
125                     if (edge[e].c == edge[e].f)
126                         continue;
127                     int v = edge[e].to;
128                     ll ndist = d[iter][u] + edge[e].cost;
129                     if (ndist >= d[iter + 1][v])
130                         continue;
131                     d[iter + 1][v] = ndist;
132                     dfrom[iter + 1][v] = e;
133                 }
134             q ^= 1;
135         }
136         int w = -1;
137         ld mindmax = 1e18;
138         for (u, N) {
139             ld dmax = -1e18;
140             for (iter, N)
141                 dmax = max(dmax,
142                     (d[N][u] - d[iter][u]) / ld(N - iter));
143             if (mindmax > dmax)
144                 mindmax = dmax, w = u;
145         }
146         if (mindmax >= 0)
147             break;
148         fill(level, level + N, -1);
149         int k = N;
150         while (level[w] == -1) {
151             level[w] = k;
152             w = edge[dfrom[k-1]][w ^ 1].to;
153         }
154         int k2 = level[w];
155         ll delta = infc;
156         while (k2 > k) {
157             int e = dfrom[k2-1][w];
158             delta = min(delta, edge[e].c - edge[e].f);
159             w = edge[e ^ 1].to;
160         }
161         k2 = level[w];
162         while (k2 > k) {
163             int e = dfrom[k2-1][w];
164             totalCost += edge[e].cost * delta;
165             edge[e].f += delta;
166             edge[e ^ 1].f -= delta;
167             w = edge[e ^ 1].to;
168         }
169     }
170}
171 // namespace MinCost
172
173 int main() {
174     MinCost::N = 3, MinCost::S = 1, MinCost::T = 2;
175     MinCost::addEdge(1, 0, 3, 5);
176     MinCost::addEdge(0, 2, 4, 6);
177     while (MinCost::push());
178     cout << MinCost::totalFlow << ' '
179         << MinCost::totalCost << '\n'; //3 33
180}

```

## 5 geometry/basic3d.cpp

```

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 */

```

## 6 geometry/chan.cpp

```

1 mt19937 rr(111);
2 ld rndEps() {
3     return (ld(rr()) / rr.max() - 0.5) * 1e-7;
4 }
5
6 typedef tuple<int, int, int> Face;
7 const ld infc = 1e100;
8
9 int n;
10 pt p[maxn];
11
12 namespace Chan {
13 pt _p[maxn];
14
15 ld 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
21 ld 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
26 ld 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
36 void 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
46 ld 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
57 vector<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)

```

## 7 geometry/halfplanes.cpp

```

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
147 vector<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

```

```

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    for (iter, 2)
14        for (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    for (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    for (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}

```

## 8 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 }

```

## 9 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 };

```

## 10 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}

```

## 11 graphs/edmonds\_matching.cpp

```

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

```

## 12 graphs/euler\_cycle.cpp

```

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

```



## 13 graphs/kuhn.cpp

```

1 bool 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
21 fill(pair, -1);
22 for (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}

```

## 14 graphs/min\_automaton.cpp

```

1 vi inc[maxn][A];
2 int lst[maxn], pos[maxn], part[maxn];
3 int lp[maxn], rp[maxn], nrp[maxn];
4 int upd[maxn], used[maxn], inq[maxn];
5 vector<int> q;
6 int dtime;
7 int np; // number of classes
8 vector<int> toRefine[A];
9
10 void doSwap(int x, int y) {
11     swap(lst[pos[x]], lst[pos[y]]);
12     swap(pos[x], pos[y]);
13 }
14
15 void 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
54 void 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 }

```

## 15 math/factor.cpp

```

1//WARNING: only mod <= 1e18
2ll 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}

```

## 16 math/golden\_search\_quad\_eq.cpp

```

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

```

## 17 math/numbers.tex

- Simpson and Gauss numerical integration:

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

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

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

- FFT modules:

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

- Fibonacci numbers:

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

- 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

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

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

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

## 18 math/stuff.cpp

```

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}

```

## 19 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}

```

## 20 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 }

```

## 21 strings/hashes.cpp

```

1#define forn(i, n) for (int i = 0; i < (int)(n); i++)
2#define sz(a) (int)(a).size()
3
4typedef long long ll;
5typedef unsigned long long ull;
6
7struct 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) %
17   ↪ MA, (b + x.b) % MB); }
18   num operator - ( const num &x ) const { return num((a + MA -
19   ↪ x.a) % MA, (b + MB - x.b) % MB); }
20   num operator * ( int x ) const { return num(((ll)a * x) % MA,
21   ↪ ((ll)b * x) % MB); }
22   num operator * ( const num &x ) const { return num(((ll)a *
23   ↪ x.a) % MA, ((ll)b * x.b) % MB); }
24   bool operator == ( const num &x ) const { return a == x.a && b
25   ↪ == x.b; }
26
27   explicit operator ll () const { return (ll)a * MB + b + 1; }
28   ↪ // > 0
29};
30
31template <class hash_t>
32struct StrComparator {
33    static const int P;
34    static vector<hash_t> deg;
35
36    int n;
37    const char *s;
38    hash_t *h;
39
40    StrComparator( int n, const char *s ) : n(n), s(s) {
41        h = new hash_t[n + 1];
42        h[0] = 0;
43        forn(i, n)
44            h[i + 1] = h[i] * P + s[i];
45        deg.reserve(n);
46        while (sz(deg) <= n)
47            deg.push_back(*deg.rbegin() * P);
48    }
49
50    hash_t substr( int i, int len ) const { return h[i + len] -
51    ↪ h[i] * deg[len]; }
52
53    int lcp( int i, int j ) {
54        int L = 0, R = n - max(i, j);
55        while (L < R) {
56            int M = (L + R + 1) / 2;
57            if (substr(i, M) == substr(j, M))
58                L = M;
59            else
60                R = M - 1;
61        }
62        return L;
63    }
64
65    int cmp( int a, int b ) {
66        int LEN = n - max(a, b), L = lcp(a, b);
67        return L < LEN ? (int)s[a + L] - s[b + L] : b - a;
68    }
69
70    bool operator() ( int i, int j ) { return cmp(i, j) < 0; }
71};
72
73template <class hash_t> vector <hash_t>
74    ↪ StrComparator<hash_t>::deg(1, hash_t(1));
75
76template <class hash_t> const int StrComparator<hash_t>::P =
77    ↪ max(239, rand());
78
79// StrComparator<num> h(n, s);
80
81/**
82 * Usage:
83 * StrComparator<num> h(length, s); // int length, char *s
84 * h.substr(0, 3) == h.substr(1, 3); // сравнение на равенство
85 ↪ подстрок за O(1)
86 * h.cmp(2, 3); // сравнение на больше-меньше суффиксов за
87 ↪ O(logn)
88 *
89 * int p[n]; forn(i, n) p[i] = i;
90 * sort(p, p + n, h); // сортировать суффиксы, суф.массив за
91 ↪ O(n log^2 n)
92 */

```

## 22 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}

```

## 23 strings/ukkonen.cpp

```

1 string s;
2 const int alpha = 26;
3
4 namespace 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;
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};

```

## 24 structures/fenwick.cpp

```

1//BEGIN ALGO
2struct 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
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

```

## 25 structures/heavy\_light.cpp

```

1int n;
2vi e[maxn];
3
4namespace 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

```

## 26 structures/ordered\_set.cpp

```

1#include <ext/pb_ds/assoc_container.hpp>
2#include <ext/pb_ds/tree_policy.hpp>
3
4typedef __gnu_pbds::tree<int, __gnu_pbds::null_type,
5    std::less<int>,
6    __gnu_pbds::rb_tree_tag,
7    __gnu_pbds::tree_order_statistics_node_update> oset;
8
9#include <iostream>
10
11int main() {
12    oset X;
13    X.insert(1);
14    X.insert(2);
15    X.insert(4);
16    X.insert(8);
17    X.insert(16);
18
19    std::cout << *X.find_by_order(1) << std::endl; // 2
20    std::cout << *X.find_by_order(2) << std::endl; // 4
21    std::cout << *X.find_by_order(4) << std::endl; // 16
22    std::cout << std::boolalpha <<
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}

```

## 27 structures/treap.cpp

```

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

```







