Содержание

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

```cpp
namespace Dinic {
const int maxn = 10010;

struct Edge {
    int to, c, f;
} es[maxn*2];
int ne = 0;

int n;
vector<int> e[maxn];
int q[maxn], d[maxn], pos[maxn];
int S, T;

void addEdge(int u, int v, int c) {
    assert(c <= 1000000000);
    es[ne] = {v, c, 0};
    e[u].push_back(ne++);
    es[ne] = {u, 0, 0};
    e[v].push_back(ne++);
}

bool bfs() {
    forn(i, n) d[i] = maxn;
    d[S] = 0, q[0] = S;
    int lq = 0, rq = 1;
    while (lq != rq) {
        int v = q[lq++];
        for (int id: e[v]) if (es[id].f < es[id].c) {
            int to = es[id].to;
            if (d[to] == maxn)
                d[to] = d[v] + 1, q[rq++] = to;
        }
    }
    return d[T] != maxn;
}

int dfs(int v, int curf) {
    if (v == T || curf == 0) return curf;
    for (int &i = pos[v]; i < (int)e[v].size(); ++i) {
        int id = e[v][i];
        int to = es[id].to;
        if (es[id].f < es[id].c && d[v] + 1 == d[to]) {
            if (int ret = dfs(to, min(curf, es[id].c-es[id].f)))
            {
                es[id].f += ret;
                es[id^1].f -= ret;
                return ret;
            }
        }
    }
    return 0;
}

i64 dinic(int S, int T) {
    Dinic::S = S, Dinic::T = T;
    i64 res = 0;
    while (bfs()) {
        forn(i, n) pos[i] = 0;
        while (int f = dfs(S, 1e9)) {
            assert(f <= 1000000000);
            res += f;
        }
    }
    return res;
}

} // namespace Dinic

void test() {
    Dinic::n = 4;
    Dinic::addEdge(0, 1, 1);
    Dinic::addEdge(0, 2, 2);
    Dinic::addEdge(2, 1, 1);
    Dinic::addEdge(1, 3, 2);
    Dinic::addEdge(2, 3, 1);
    cout << Dinic::dinic(0, 3) << endl; // 3
}

/*
LR-поток находит не максимальный поток.
Добавим новый сток S' и исток T'. Заменим ребро (u, v, l, r)
LR-сети на ребра (u, T', l), (S', v, l), (u, v, r - l).
Добавим ребро (T, S, k). Ставим значение k=inf, пускаем поток.
Проверяем, что все ребра из S' насыщены (иначе ответ не
существует). Бинпоиском находим наименьшее k, что величина
потока не изменится. Это k - величина МИНИМАЛЬНОГО потока,
удовлетворяющего ограничениям. */
```

3   flows/globalcut.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
#define forn(i,n) for (int i = 0; i < int(n); ++i)
const int inf = 1e9 + 1e5;
#define all(x) (x).begin(), (x).end()

const int maxn = 505;
namespace StoerWagner {
int g[maxn][maxn];
int dist[maxn];
bool used[maxn];
int n;

void addEdge(int u, int v, int c) {
    g[u][v] += c;
    g[v][u] += c;
}

int run() {
    vector<int> vertices;
    forn (i, n)
        vertices.push_back(i);
    int mincut = inf;
    while (vertices.size() > 1) {
        int u = vertices[0];
        for (auto v: vertices) {
            used[v] = false;
            dist[v] = g[u][v];
        }
        used[u] = true;
        forn (ii, vertices.size() - 2) {
            for (auto v: vertices)
                if (!used[v])
                    if (used[u] || dist[v] > dist[u])
                        u = v;
            used[u] = true;
            for (auto v: vertices)
                if (!used[v])
                    dist[v] += g[u][v];
        }
        int t = -1;
        for (auto v: vertices)
            if (!used[v])
                t = v;
        assert(t != -1);
        mincut = min(mincut, dist[t]);
        vertices.erase(find(all(vertices), t));
        for (auto v: vertices)
            addEdge(u, v, g[v][t]);
    }
    return mincut;
}
} // namespace StoerWagner

int main() {
    StoerWagner::n = 4;
    StoerWagner::addEdge(0, 1, 5);
    StoerWagner::addEdge(2, 3, 5);
    StoerWagner::addEdge(1, 2, 4);
    cerr << StoerWagner::run() << '\n'; // 4
}
```

4   flows/hungary.cpp

```cpp
// left half is the smaller one
namespace Hungary {
const int maxn = 505;
int a[maxn][maxn];
int p[2][maxn];
int match[maxn];
bool used[maxn];
int from[maxn];
int mind[maxn];
int n, m;

int hungary(int v) {
    used[v] = true;
    int u = match[v];
    int best = -1;
    forn (i, m + 1) {
        if (used[i])
            continue;
        int nw = a[u][i] - p[0][u] - p[1][i];
        if (nw <= mind[i]) {
            mind[i] = nw;
            from[i] = v;
        }
        if (best == -1 || mind[best] > mind[i])
            best = i;
    }
    v = best;
    int delta = mind[best];
    forn (i, m + 1) {
        if (used[i]) {
            p[1][i] -= delta;
            p[0][match[i]] += delta;
        } else
            mind[i] -= delta;
    }
    if (match[v] == -1)
        return v;
    return hungary(v);
}

void check() {
    int edges = 0, res = 0;
    forn (i, m)
        if (match[i] != -1) {
            ++edges;
            assert(p[0][match[i]] + p[1][i] == a[match[i]][i]);
            res += a[match[i]][i];
        } else
            assert(p[1][i] == 0);
    assert(res == -p[1][m]);
    forn (i, n) forn (j, m)
        assert(p[0][i] + p[1][j] <= a[i][j]);
}

int run() {
    forn (i, n)
        p[0][i] = 0;
    forn (i, m + 1) {
        p[1][i] = 0;
        match[i] = -1;
    }
    forn (i, n) {
        match[m] = i;
        fill(used, used + m + 1, false);
        fill(mind, mind + m + 1, inf);
        fill(from, from + m + 1, -1);
        int v = hungary(m);
        while (v != m) {
            int w = from[v];
            match[v] = match[w];
            v = w;
        }
    }
    check();
    return -p[1][m];
}
} // namespace Hungary
```

5   flows/mincost.cpp

```
1 namespace MinCost {
2 const ll infc = 1e12;
3
4 struct Edge {
5     int to;
6     ll c, f, cost;
7
8     Edge(int to, ll c, ll cost): to(to), c(c), f(0), cost(cost)
9     { }
10 };
11
12 int N, S, T;
13 int totalFlow;
14 ll totalCost;
15 const int maxn = 505;
16 vector<Edge> edge;
17 vector<int> g[maxn];
18
19 void addEdge(int u, int v, ll c, ll cost) {
20     g[u].push_back(edge.size());
21     edge.emplace_back(v, c, cost);
22     g[v].push_back(edge.size());
23     edge.emplace_back(u, 0, -cost);
24 }
25
26 ll dist[maxn];
27 int fromEdge[maxn];
28
29 bool inQueue[maxn];
30 bool fordBellman() {
31     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
58 ll pot[maxn];
59 bool dikstra() {
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];
```

```
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         forn (iter, N) {
122             fill(d[iter + 1], d[iter + 1] + N, infc);
123             forn (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         forn (u, N) {
139             ld dmax = -1e18;
140             forn (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--][w] ^ 1].to;
153         }
154         int k2 = level[w];
155         ll delta = infc;
156         while (k2 > k) {
157             int e = dfrom[k2--][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--][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 }
```

6   flows/push_relabel.cpp

```cpp
namespace PushRelabel {
const int maxn = 200500;

struct Edge {
    int to, c, f;
};
vector<Edge> edge;

int n;
vector<int> g[maxn];
ll e[maxn];
int h[maxn];
int onH[maxn];
int S, T;
int ptr[maxn];
int relabelTimer;

void addEdge(int u, int v, int c) {
    g[u].push_back(sz(edge));
    edge.push_back({v, c, 0});
    g[v].push_back(sz(edge));
    edge.push_back({u, 0, 0});
}

void push(int id, int delta) {
    int u = edge[id ^ 1].to;
    int v = edge[id].to;
    edge[id].f += delta;
    edge[id ^ 1].f -= delta;
    e[u] -= delta;
    e[v] += delta;
}

void gap(int ch) {
    forn (u, n) {
        if (h[u] > ch)
            h[u] = max(h[u], n);
    }
}

int o[maxn];
void globalRelabeling() {
    int oc = 0;
    forn (i, n) {
        h[i] = n;
        onH[i] = 0;
    }
    onH[0] = 1;
    h[T] = 0;
    o[oc++] = T;
    forn (ii, oc) {
        int u = o[ii];
        for (int id: g[u]) {
            if (edge[id ^ 1].c == edge[id ^ 1].f)
                continue;
            int v = edge[id].to;
            if (h[v] != n)
                continue;
            h[v] = h[u] + 1;
            onH[h[v]]++;
            o[oc++] = v;
        }
    }
}

void relabel(int u) {
    int oldh = h[u];
    int newh = inf;
    for (int id: g[u]) {
        if (edge[id].c == edge[id].f)
            continue;
        newh = min(newh, h[edge[id].to] + 1);
    }
    h[u] = newh;
    onH[oldh]--;
    onH[newh]++;
    if (onH[oldh] == 0)
        gap(oldh);
    if (++relabelTimer == n)
        globalRelabeling(), relabelTimer = 0;
}

void discharge(int u) {
    while (e[u] > 0) {
        int &i = ptr[u];
        if (i == sz(g[u])) {
            i = 0;
            relabel(u);
            if (h[u] >= n)
                break;
            continue;
        } else {
            int id = g[u][i++];
            int v = edge[id].to;
            if (h[v] + 1 != h[u])
                continue;
            int delta = min(e[u], ll(edge[id].c - edge[id].f));
            push(id, delta);
        }
    }
}

ll flow(int _S, int _T) {
    S = _S, T = _T;
    forn (i, n)
        ptr[i] = 0, e[i] = 0;
    for (int id: g[S]) {
        int delta = edge[id].c;
        push(id, delta);
    }
    globalRelabeling();
    bool ok = false;
    while (!ok) {
        ok = true;
        forn (u, n) {
            if (h[u] < n && u != T && e[u] > 0)
                discharge(u), ok = false;
        }
    }
    return e[T];
}

} //PushRelabel
```

7   geometry/chan.cpp

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  #define forn(i,n) for (int i = 0; i < int(n); ++i)
4  typedef long double ld;
5
6  const int maxn = 100100;
7  const ld eps = 1e-9;
8
9  mt19937 rr(111);
10 ld rndEps() {
11     return (ld(rr()) / rr.max() - 0.5) / 1e5;
12 }
13
14 bool gt(ld a, ld b) { return a - b > eps; }
15 bool lt(ld a, ld b) { return b - a > eps; }
16 bool eq(ld a, ld b) { return fabsl(a - b) < eps; }
17
18 struct pt {
19     ld x, y, z;
20     ld ox, oy, oz;
21     int pr, nx;
22     bool inHull;
23
24     static pt *NIL;
25
26     pt() {}
27
28     pt(ld x, ld y, ld z): x(x), y(y), z(z) {}
29
30     pt operator-(const pt &p) const {
31         return pt(x - p.x, y - p.y, z - p.z);
32     }
33
34     ld operator*(const pt &p) const {
35         return x * p.x + y * p.y + z * p.z;
36     }
37
38     pt operator%(const pt &p) const {
39         return pt(y * p.z - z * p.y,
40                   z * p.x - x * p.z,
41                   x * p.y - y * p.x);
42     }
43
44     bool operator==(const pt &a) {
45         return eq(x, a.x) && eq(y, a.y) && eq(z, a.z);
46     }
47
48     void transform(bool rev) {
49         if (rev) {
50             x = ox, y = oy, z = oz;
51         } else {
52             ox = x, oy = y, oz = z;
53             x += rndEps(), y += rndEps(), z += rndEps();
54         }
55     }
56 };
57
58 ostream &operator<<(ostream &out, pt &p) {
59     return out << p.x << ' ' << p.y << ' ' << p.z;
60 }
61
62 istream &operator>>(istream &in, pt &p) {
63     return in >> p.x >> p.y >> p.z;
64 }
65
66 typedef tuple<int, int, int> Facet;
67
68 namespace Chan {
69 int n;
70 pt p[maxn];
71
72 ld turn(int p1, int p2, int p3) {
73     assert(p1 != -1 && p2 != -1 && p3 != -1);
74     return (p[p2].x - p[p1].x) * (p[p3].y - p[p1].y) -
75            (p[p3].x - p[p1].x) * (p[p2].y - p[p1].y);
76 }
77
78 //replace y with z
79 ld turnz(int p1, int p2, int p3) {
80     assert(p1 != -1 && p2 != -1 && p3 != -1);
81     return (p[p2].x - p[p1].x) * (p[p3].z - p[p1].z) -
82            (p[p3].x - p[p1].x) * (p[p2].z - p[p1].z);
83 }
84
85 ld gett(int p1, int p2, int p3) {
86     return turnz(p1, p2, p3) / turn(p1, p2, p3);
87 }
88
89 void act(int i) {
90     if (p[i].inHull) {
91         p[p[i].nx].pr = p[i].pr;
92         p[p[i].pr].nx = p[i].nx;
93     } else {
94         p[p[i].nx].pr = p[p[i].pr].nx = i;
95     }
96     p[i].inHull ^= 1;
97 }
98
99 vector<int> buildHull(int l, int r, bool upper) {
100    if (l + 1 >= r) {
101        p[l].pr = p[l].nx = -1;
102        p[l].inHull = true;
103        return {};
104    }
105    int mid = (l + r) / 2;
106    auto L = buildHull(l, mid, upper);
107    auto R = buildHull(mid, r, upper);
108    reverse(L.begin(), L.end());
109    reverse(R.begin(), R.end());
110    int u = mid - 1, v = mid;
111    while (true) {
112        if (p[u].pr != -1 &&
113                ((turn(p[u].pr, u, v) < 0) ^ upper))
114            u = p[u].pr;
115        else if (p[v].nx != -1 &&
116                ((turn(u, v, p[v].nx) < 0) ^ upper))
117            v = p[v].nx;
118        else
119            break;
120    }
121
122    ld T = -1e100;
123    ld t[6];
124    vector<int> A;
125    while (true) {
126        forn (i, 6)
127            t[i] = 1e100;
128        if (!L.empty()) {
129            int id = L.back();
130            t[0] = gett(p[id].pr, id, p[id].nx);
131        }
132        if (!R.empty()) {
133            int id = R.back();
134            t[1] = gett(p[id].pr, id, p[id].nx);
135        }
136        if (p[u].pr != -1)
137            t[2] = gett(p[u].pr, u, v);
138        if (p[u].nx != -1)
139            t[3] = gett(u, p[u].nx, v);
140        if (p[v].pr != -1)
141            t[4] = gett(u, p[v].pr, v);
142        if (p[v].nx != -1)
143            t[5] = gett(u, v, p[v].nx);
144        ld nt = 1e100;
145        int type = -1;
146        forn (i, 6)
147            if ((t[i] - T >= 1e-15) && t[i] < nt)
148                nt = t[i], type = i;
149        if (type == -1)
150            break;
151
152        if (type == 0) {
153            act(L.back());
154            if (L.back() < u)
155                A.push_back(L.back());
156            L.pop_back();
157        } else if (type == 1) {
158            act(R.back());
159            if (R.back() > v)
160                A.push_back(R.back());
161            R.pop_back();
162        } else if (type == 2) {
163            A.push_back(u);
164            u = p[u].pr;
165        } else if (type == 3) {
166            u = p[u].nx;
167            A.push_back(u);
168        } else if (type == 4) {
169            v = p[v].pr;
170            A.push_back(v);
171        } else if (type == 5) {
172            A.push_back(v);
173            v = p[v].nx;
174        } else
175            assert(false);
176        T = nt;
177    }
178    assert(L.empty() && R.empty());
179
180    p[u].nx = v, p[v].pr = u;
181    for (int i = u + 1; i < v; ++i)
182        p[i].inHull = false;
183    for (int i = int(A.size()) - 1; i >= 0; --i) {
184        int id = A[i];
```

```
185        if (id <= u || id >= v) {
186            if (u == id)
187                u = p[u].pr;
188            if (v == id)
189                v = p[v].nx;
190            act(id);
191        } else {
192            p[id].pr = u, p[id].nx = v;
193            act(id);
194            if (id >= mid)
195                v = id;
196            else
197                u = id;
198        }
199    }
200
201    return A;
202}
203
204 //facets are oriented ccw if look from the outside
205 vector<Facet> getFacets() {
206    forn (i, n)
207        p[i].transform(false);
208    //WARNING: original order of points is changed
209    sort(p, p + n, [](const pt &a, const pt &b) {
210            return a.x < b.x;
211        });
212
213    vector<Facet> facets;
214    forn (q, 2) {
215        auto movie = buildHull(0, n, q);
216        for (auto x: movie) {
217            if (!p[x].inHull)
218                facets.emplace_back(p[x].pr, x, p[x].nx);
219            else
220                facets.emplace_back(p[x].pr, p[x].nx, x);
221            act(x);
222        }
223    }
224    forn (i, n)
225        p[i].transform(true);
226    return facets;
227}
228} //namespace Chan
229
230 int main() {
231    int n;
232    cin >> n;
233    Chan::n = n;
234    forn (i, n)
235        cin >> Chan::p[i];
236    auto facets = Chan::getFacets();
237    set<int> nodes;
238    for (auto f: facets) {
239        nodes.insert(get<0>(f));
240        nodes.insert(get<1>(f));
241        nodes.insert(get<2>(f));
242    }
243    assert(nodes.size() * 2 == facets.size() + 4);
244    ld V = 0, S = 0;
245    for (auto f: facets) {
246        pt v1 = Chan::p[get<1>(f)] - Chan::p[get<0>(f)];
247        pt v2 = Chan::p[get<2>(f)] - Chan::p[get<0>(f)];
248        pt v3 = Chan::p[get<0>(f)];
249        pt vv = v1 % v2;
250        forn (i, n) {
251            pt v4 = Chan::p[i] - Chan::p[get<0>(f)];
252            assert(v4 * vv < 0.1);
253        }
254        S += sqrtl(vv.x * vv.x + vv.y * vv.y + vv.z * vv.z) / 2;
255        V += vv * v3 / 6;
256    }
257    cout.precision(10);
258    cout << fixed;
259    cout << S << ' ' << V << '\n';
260}
```

8   geometry/convex_hull.cpp

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 #define forn(i, n) for (int i = 0; i < int(n); ++i)
4 #define sz(x) ((int) (x).size())
5
6 #include "primitives.cpp"
7
8 bool cmpAngle(const pt &a, const pt &b) {
9     bool ar = a.right(), br = b.right();
10    if (ar ^ br)
11        return ar;
12    return a % b > eps;
13 }
14
15 struct Hull {
16    vector<pt> top, bot;
17
18    void append(pt p) {
19        while (bot.size() > 1 && (p - bot.back())
20                % (bot.back() - *next(bot.rbegin())) >= -eps)
21            bot.pop_back();
22        bot.push_back(p);
23        while (top.size() > 1 && (p - top.back())
24                % (top.back() - *next(top.rbegin())) <= eps)
25            top.pop_back();
26        top.push_back(p);
27    }
28
29    void build(vector<pt> h) {
30        sort(h.begin(), h.end());
31        h.erase(unique(h.begin(), h.end()), h.end());
32        top.clear(), bot.clear();
33        for (pt p: h)
34            append(p);
35    }
36
37    pt kth(int k) {
38        if (k < sz(bot))
39            return bot[k];
40        else
41            return top[sz(top) - (k - sz(bot)) - 2];
42    }
43
44    pt mostDistant(pt dir) {
45        if (bot.empty()) {
46            //empty hull
47            return pt{1e18, 1e18};
48        }
49        if (bot.size() == 1)
50            return bot.back();
51        dir = dir.rot();
52        int n = sz(top) + sz(bot) - 2;
53        int L = -1, R = n;
54        while (L + 1 < R) {
55            int C = (L + R) / 2;
56            pt v = kth((C + 1) % n) - kth(C);
57            if (cmpAngle(dir, v)) //finds upper bound
58                R = C;
59            else
60                L = C;
61        }
62        return kth(R % n);
63    }
64 };
```

9   geometry/halfplanes.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
#define forn(i, n) for (int i = 0; i < int(n); ++i)
#define forab(i, a, b) for (int i = int(a); i < int(b); ++i)
#include "primitives.cpp"

ld det3x3(line &l1, line &l2, line &l3) {
    return l1.a * (l2.b * l3.c - l2.c * l3.b) +
           l1.b * (l2.c * l3.a - l2.a * l3.c) +
           l1.c * (l2.a * l3.b - l2.b * l3.a);
}

vector<pt> halfplanesIntersecion(vector<line> lines) {
    sort(lines.begin(), lines.end(),
        [](const line &a, const line &b) {
                bool ar = a.right(), br = b.right();
                if (ar ^ br)
                    return ar;
                ld prod = (pt{a.a, a.b} % pt{b.a, b.b});
                if (!ze(prod))
                    return prod > 0;
                return a.c < b.c;
            });
    vector<line> lines2;
    pt pr;
    forn (i, lines.size()) {
        pt cur{lines[i].a, lines[i].b};
        if (i == 0 || cur != pr)
            lines2.push_back(lines[i]);
        pr = cur;
    }
    lines = lines2;
    int n = lines.size();
    forn (i, n)
        lines[i].id = i;
    vector<line> hull;
    forn (i, 2 * n) {
        line l = lines[i % n];
        while ((int) hull.size() >= 2) {
            ld D = det3x3(*next(hull.rbegin()), hull.back(), l);
            if (D >= -eps)
                break;
            hull.pop_back();
        }
        hull.push_back(l);
    }
    vector<int> firstTime(n, -1);
    vector<line> v;
    forn (i, hull.size()) {
        int cid = hull[i].id;
        if (firstTime[cid] == -1) {
            firstTime[cid] = i;
            continue;
        }
        forab(j, firstTime[cid], i)
            v.push_back(hull[j]);
        break;
    }
    n = v.size();
    if (v.empty()) {
        //empty intersection
        return {};
    }
    v.push_back(v[0]);
    vector<pt> res;
    pt center{0, 0};
    forn (i, n) {
        res.push_back(linesIntersection(v[i], v[i + 1]));
        center = center + res.back();
    }
    center = center / n;
    for (auto l: lines)
        if (l.signedDist(center) < -eps) {
            //empty intersection
            return {};
        }
    return res;
}
```

10   geometry/planar_faces.cpp

```cpp
int m, n; // segs, points
pair<pt, pt> segs[maxn];
pt p[maxn], from, to;
map<pt, int> shr;
vi e[maxn]; // points adjacent to point
int getPoint(pt x) {
    if (shr.count(x)) return shr[x];
    p[n] = x;
    return shr[x] = n++;
}
// segIntersection: {bool, point}, true iff exactly one point
void genIntersections() {
    forn(i, m) {
        getPoint(segs[i].fi);
        getPoint(segs[i].se);
        forn(j, i) {
            auto t = segmentsIntersection(
                segs[i].fi, segs[i].se, segs[j].fi, segs[j].se);
            if (t.fi) getPoint(t.se);
        }
    }
}

void genGraph() {
    forn(i, m) {
        vi pts;
        forn(j, n) if (pointInsideSegment(
                    p[j], segs[i].fi, segs[i].se)) {
            pts.push_back(j);
        }
        sort(all(pts), [](int i, int j) {
            return p[i] < p[j]; });
        forn(j, pts.size() - 1) {
            int u = pts[j], v = pts[j+1];
            e[u].push_back(v);
            e[v].push_back(u);
        }
    }
    forn(i, n) {
        sort(all(e[i]), [i](int x, int y) {
            pt a = p[x] - p[i];
            pt b = p[y] - p[i];
            if (a.right() != b.right()) return a.right();
            return a % b > 0;
        });
    }
}

vector<pt> faces[maxn];
bool inner[maxn];
int nf;
map<pii, int> faceForEdge;
vi ef[maxn]; // graph on faces

void genFaces() {
    forn(i, n) for (int to: e[i]) {
        if (faceForEdge.count({i, to})) continue;
        int f = nf++;
        int v = i, u = to;
        do {
            faces[f].push_back(p[v]);
            faceForEdge[{v, u}] = f;
            auto it = lower_bound(all(e[u]), v,
                [u] (int x, int y) {
                    pt a = p[x] - p[u];
                    pt b = p[y] - p[u];
                    if (a.right()!=b.right()) return a.right();
                    return a % b > 0;
                });
            assert(*it == v);
            if (it == e[u].begin()) it = e[u].end();
            v = u;
            u = *--it;
        } while (v != i || u != to);
    }
    forn(i, nf) {
        ld s = 0;
        forn(j, faces[i].size()) {
            s += faces[i][j] % faces[i][(j+1)%faces[i].size()];
        }
        inner[i] = gt(s, 0);
    }
    forn(v, n) for (int to: e[v]) {
        int f1 = faceForEdge[{v, to}];
        int f2 = faceForEdge[{to, v}];
        if (f1 != f2) {
            ef[f1].push_back(f2);
            ef[f2].push_back(f1);
        }
    }
}
```

11   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 //p must be oriented counterclockwise
13 bool segmentInsidePolygon(pt a, pt b, pt *p, int n) {
14     if (!pointInsidePolygon((a + b) / 2, p, n))
15         return false;
16     if (a == b)
17         return true;
18     forn (i, n) {
19         pt c = p[i];
20         if (ze((a - c) % (b - c)) &&
21                 (a - c) * (b - c) < -eps) {
22             //point on segment
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 }
```

12   geometry/polygon_tangents.cpp

```
1  struct Hull {
2      vector<pt> p, v;
3      pt M;
4      int n;
5
6      void build() {
7          sort(p.begin(), p.end());
8          p.erase(unique(p.begin(), p.end()), p.end());
9          vector<pt> top, bot;
10         for (auto P: p) {
11             while (sz(top) > 1 && (P - top.back()) %
12                     (top.back() - *next(top.rbegin())) <= eps)
13                 top.pop_back();
14             top.push_back(P);
15             while (sz(bot) > 1 && (P - bot.back()) %
16                     (bot.back() - *next(bot.rbegin())) >= -eps)
17                 bot.pop_back();
18             bot.push_back(P);
19         }
20         if (sz(top))
21             top.pop_back();
22         reverse(top.begin(), top.end());
23         if (sz(top))
24             top.pop_back();
25         p = bot;
26         p.insert(p.end(), top.begin(), top.end());
27
28         n = sz(p);
29         M = accumulate(p.begin(), p.end(), pt{0, 0});
30         M = M * (1. / n);
31         v.resize(n);
32         forn (i, n)
33             v[i] = p[i] - M;
34         int r = min_element(v.begin(), v.end(),
35                 cmpAngle) - v.begin();
36         rotate(v.begin(), v.begin() + r, v.end());
37         rotate(p.begin(), p.begin() + r, p.end());
38         gassert(is_sorted(v.begin(), v.end(), cmpAngle));
39     }
40
41     bool visSide(pt a, int i) {
42         return (p[(i + 1) % n] - a) % (p[i % n] - a) > eps;
43     }
44
45     bool vis(pt a, int i) {
46         return visSide(a, i) || visSide(a, i + n - 1);
47     }
48
49     bool isTangent(pt a, int i) {
50         return visSide(a, i) != visSide(a, i + n - 1);
51     }
52
53     int binSearch(int l, int r, pt a) {
54         //tricky binsearch; l < r not necessarily
55         while (abs(l - r) > 1) {
56             int c = (l + r) / 2;
57             if (vis(a, c))
58                 l = c;
59             else
60                 r = c;
61         }
62         assert(isTangent(a, l));
63         return l % n;
64     }
65
66     pair<int, int> tangents(pt a) {
67         assert(n >= 3);
68         if (a == M)
69             return {-1, -1};
70         int pos = lower_bound(v.begin(), v.end(), a - M,
71                 cmpAngle) - v.begin();
72         pt L = p[(pos + n - 1) % n], R = p[pos % n];
73         if ((R - L) % (a - L) >= -eps)
74             return {-1, -1};
75         int pos2 = lower_bound(v.begin(), v.end(), M - a,
76                 cmpAngle) - v.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
84 };
```

13 geometry/primitives.cpp

```
1  //WARNING! do not forget to normalize vector (a,b)
2  struct line {
3      ld a, b, c;
4      int id;
5
6      line(pt p1, pt p2) {
7          gassert(p1 != p2);
8          pt n = (p2 - p1).rot();
9          n /= n.abs();
10         a = n.x, b = n.y;
11         c = -(n * p1);
12     }
13
14     bool right() const {
15         return a > eps || (ze(a) && b > eps);
16     }
17
18     line(ld _a, ld _b, ld _c): a(_a), b(_b), c(_c) {
19         ld d = pt{a, b}.abs();
20         gassert(!ze(d));
21         a /= d, b /= d, c /= d;
22     }
23
24     ld signedDist(pt p) {
25         return p * pt{a, b} + c;
26     }
27  };
28
29  ld pointSegmentDist(pt p, pt a, pt b) {
30      ld res = min((p - a).abs(), (p - b).abs());
31      if (a != b && (p - a) * (b - a) >= 0 &&
32              (p - b) * (a - b) >= 0)
33          res = min(res,
34              fabsl((p - a) % (b - a)) / (b - a).abs());
35      return res;
36  }
37
38  pt linesIntersection(line l1, line l2) {
39      ld D = l1.a * l2.b - l1.b * l2.a;
40      if (ze(D)) {
41          if (eq(l1.c, l2.c)) {
42              //equal lines
43          } else {
44              //no intersection
45          }
46      }
47      ld dx = -l1.c * l2.b + l1.b * l2.c;
48      ld dy = -l1.a * l2.c + l1.c * l2.a;
49      pt res{dx / D, dy / D};
50      //gassert(ze(l1.signedDist(res)));
51      //gassert(ze(l2.signedDist(res)));
52      return res;
53  }
54
55  bool pointInsideSegment(pt p, pt a, pt b) {
56      if (!ze((p - a) % (p - b)))
57          return false;
58      return (a - p) * (b - p) <= eps;
59  }
60
61  bool checkSegmentIntersection(pt a, pt b, pt c, pt d) {
62      if (ze((a - b) % (c - d))) {
63          if (pointInsideSegment(a, c, d) ||
64              pointInsideSegment(b, c, d) ||
65              pointInsideSegment(c, a, b) ||
66              pointInsideSegment(d, a, b)) {
67              //intersection of parallel segments
68              return true;
69          }
70          return false;
71      }
72
73      ld s1, s2;
74      forn (q, 2) {
75          s1 = (c - a) % (b - a);
76          s2 = (d - a) % (b - a);
77          if (s1 > eps && s2 > eps)
78              return false;
79          if (s1 < -eps && s2 < -eps)
80              return false;
81          swap(a, c), swap(b, d);
82      }
83
84      return true;
85  }
86
87  // WARNING! run checkSegmentIntersecion before and process
88  // parallel case manually
89  pt segmentsIntersection(pt a, pt b, pt c, pt d) {
90      ld S = (b - a) % (d - c);
91      ld s1 = (c - a) % (d - a);
```

```
92      return a + (b - a) / S * s1;
93  }
94
95  vector<pt> circlesIntersction(pt a, ld r1, pt b, ld r2) {
96      ld d2 = (a - b).abs2();
97      ld d = (a - b).abs();
98
99      if (a == b && eq(r1, r2)) {
100         //equal circles
101     }
102     if (d2 - sqr(r1 + r2) > eps || sqr(r1 - r2) - d2 > eps) {
103         //empty intersection
104         return {};
105     }
106     int num = 2;
107     if (eq(sqr(r1 + r2), d2) || eq(sqr(r1 - r2), d2))
108         num = 1;
109     ld cosa = (sqr(r1) + d2 - sqr(r2)) / ld(2 * r1 * d);
110     ld oh = cosa * r1;
111     pt h = a + ((b - a) / d * oh);
112     if (num == 1)
113         return {h};
114     ld hp = sqrtl(max(0.L, 1 - cosa * cosa)) * r1;
115
116     pt w = ((b - a) / d * hp).rot();
117     return {h + w, h - w};
118 }
119
120 //a is circle center, p is point
121 vector<pt> circleTangents(pt a, ld r, pt p) {
122     ld d2 = (a - p).abs2();
123     ld d = (a - p).abs();
124
125     if (sqr(r) - d2 > eps) {
126         //no tangents
127         return {};
128     }
129     if (eq(sqr(r), d2)) {
130         //point lies on circle - one tangent
131         return {p};
132     }
133
134     pt B = p - a;
135     pt H = B * sqr(r) / d2;
136     ld h = sqrtl(d2 - sqr(r)) * ld(r) / d;
137     pt w = (B / d * h).rot();
138     H = H + a;
139     return {H + w, H - w};
140 }
141
142 vector<pt> lineCircleIntersection(line l, pt a, ld r) {
143     ld d = l.signedDist(a);
144     if (fabsl(d) - r > eps)
145         return {};
146     pt h = a - pt{l.a, l.b} * d;
147     if (eq(fabsl(d), r))
148         return {h};
149     pt w(pt{l.a, l.b}.rot() * sqrtl(max<ld>(0, sqr(r)-sqr(d))));
150     return {h + w, h - w};
151 }
152
153 //modified magic from e-maxx
154 vector<line> commonTangents(pt a, ld r1, pt b, ld r2) {
155     if (a == b && eq(r1, r2)) {
156         //equal circles
157         return {};
158     }
159     vector<line> res;
160     pt c = b - a;
161     ld z = c.abs2();
162     for (int i = -1; i <= 1; i += 2)
163         for (int j = -1; j <= 1; j += 2) {
164             ld r = r2 * j - r1 * i;
165             ld d = z - sqr(r);
166             if (d < -eps)
167                 continue;
168             d = sqrtl(max<ld>(0, d));
169             pt magic = pt{r, d} / z;
170             line l(magic * c, magic % c, r1 * i);
171             l.c -= pt{l.a, l.b} * a;
172             res.push_back(l);
173         }
174     return res;
175 }
```

14   geometry/svg.cpp

```cpp
struct SVG {
    FILE *out;
    ld sc = 50;

    void open() {
        out = fopen("image.svg", "w");
        fprintf(out, "<svg xmlns='http://www.w3.org/2000/svg'
        ↪    viewBox='-1000 -1000 2000 2000'>\n");
    }

    void line(pt a, pt b) {
        a = a * sc, b = b * sc;
        fprintf(out, "<line x1='%Lf' y1='%Lf' x2='%Lf' y2='%Lf'
        ↪    stroke='black'/>\n", a.x, -a.y, b.x, -b.y);
    }

    void circle(pt a, ld r = -1, string col = "red") {
        r = (r == -1 ? 10 : sc * r);
        a = a * sc;
        fprintf(out, "<circle cx='%Lf' cy='%Lf' r='%Lf'
        ↪    fill='%s'/>\n", a.x, -a.y, r, col.c_str());
    }

    void text(pt a, string s) {
        a = a * sc;
        fprintf(out, "<text x='%Lf' y='%Lf'
        ↪    font-size='10px'>%s</text>\n", a.x, -a.y,
        ↪    s.c_str());
    }

    void close() {
        fprintf(out, "</svg>\n");
        fclose(out);
        out = 0;
    }

    ~SVG() {
        if (out)
            close();
    }
} svg;
```

15   graphs/2sat.cpp

```cpp
const int maxn = 200100; //2 x number of variables

namespace TwoSAT {
    int n; //number of variables
    bool used[maxn];
    vector<int> g[maxn];
    vector<int> gr[maxn];
    int comp[maxn];
    int res[maxn];

    void addEdge(int u, int v) { //u or v
        g[u].push_back(v ^ 1);
        g[v].push_back(u ^ 1);
        gr[u ^ 1].push_back(v);
        gr[v ^ 1].push_back(u);
    }

    vector<int> ord;
    void dfs1(int u) {
        used[u] = true;
        for (int v: g[u]) {
            if (used[v])
                continue;
            dfs1(v);
        }
        ord.push_back(u);
    }

    int COL = 0;
    void dfs2(int u) {
        used[u] = true;
        comp[u] = COL;
        for (int v: gr[u]) {
            if (used[v])
                continue;
            dfs2(v);
        }
    }

    void mark(int u) {
        res[u / 2] = u % 2;
        used[u] = true;
        for (int v: g[u]) {
            if (used[v])
                continue;
            mark(v);
        }
    }

    bool run() {
        fill(res, res + 2 * n, -1);
        fill(used, used + 2 * n, false);
        forn (i, 2 * n)
            if (!used[i])
                dfs1(i);
        reverse(ord.begin(), ord.end());
        assert((int) ord.size() == (2 * n));
        fill(used, used + 2 * n, false);
        for (int u: ord) if (!used[u]) {
            dfs2(u);
            ++COL;
        }
        forn (i, n)
            if (comp[i * 2] == comp[i * 2 + 1])
                return false;

        reverse(ord.begin(), ord.end());
        fill(used, used + 2 * n, false);
        for (int u: ord) {
            if (res[u / 2] != -1) {
                continue;
            }
            mark(u);
        }
        return true;
    }
};

int main() {
    TwoSAT::n = 2;
    TwoSAT::addEdge(0, 2); //x or y
    TwoSAT::addEdge(0, 3); //x or !y
    TwoSAT::addEdge(3, 3); //!y or !y
    assert(TwoSAT::run());
    cout << TwoSAT::res[0] << ' ' << TwoSAT::res[1] << '\n';
    //1 0
}
```

16   graphs/directed_mst.cpp

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

17   graphs/edmonds_matching.cpp

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

18   graphs/euler_cycle.cpp

```cpp
1  struct Edge {
2      int to, id;
3  };
4
5  bool usedEdge[maxm];
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 }
```

19   math/factor.cpp

```cpp
//WARNING: only mod <= 1e18
ll mul(ll a, ll b, ll mod) {
    ll res = a * b - (ll(ld(a) * ld(b) / ld(mod)) * mod);
    while (res < 0)
        res += mod;
    while (res >= mod)
        res -= mod;
    return res;
}

bool millerRabinTest(ll n, ll a) {
    if (gcd(n, a) > 1)
        return false;
    ll x = n - 1;
    int l = 0;
    while (x % 2 == 0) {
        x /= 2;
        ++l;
    }
    ll c = binpow(a, x, n);
    for (int i = 0; i < l; ++i) {
        ll nx = mul(c, c, n);
        if (nx == 1) {
            if (c != 1 && c != n - 1)
                return false;
            else
                return true;
        }
        c = nx;
    }
    return c == 1;
}

bool isPrime(ll n) {
    if (n == 1)
        return false;
    if (n % 2 == 0)
        return n == 2;
    for (ll a = 2; a < min<ll>(8, n); ++a)
        if (!millerRabinTest(n, a))
            return false;
    return true;
}

//WARNING: p is not sorted
void factorize(ll x, vector<ll> &p) {
    if (x == 1)
        return;
    if (isPrime(x)) {
        p.push_back(x);
        return;
    }
    for (ll d: {2, 3, 5})
        if (x % d == 0) {
            p.push_back(d);
            factorize(x / d, p);
            return;
        }
    while (true) {
        ll x1 = rr() % (x - 1) + 1;
        ll x2 = (mul(x1, x1, x) + 1) % x;
        int i1 = 1, i2 = 2;
        while (true) {
            ll c = (x1 + x - x2) % x;
            if (c == 0)
                break;
            ll g = gcd(c, x);
            if (g > 1) {
                factorize(g, p);
                factorize(x / g, p);
                return;
            }
            if (i1 * 2 == i2) {
                i1 *= 2;
                x1 = x2;
            }
            ++i2;
            x2 = (mul(x2, x2, x) + 1) % x;
        }
    }
}
```

20   math/fft.cpp

```cpp
const int maxlg = 20;

vector<base> ang[maxlg + 5];

void init_fft() {
    int n = 1 << maxlg;
    ld e = acosl(-1) * 2 / n;
    ang[maxlg].resize(n);
    forn(i, n) {
        ang[maxlg][i] = { cos(e * i), sin(e * i) };
    }

    for (int k = maxlg - 1; k >= 0; --k) {
        ang[k].resize(1 << k);
        forn(i, 1<<k) {
            ang[k][i] = ang[k+1][i*2];
        }
    }
}

void fft_rec(base *a, int lg, bool rev) {
    if (lg == 0) {
        return;
    }
    int len = 1 << (lg - 1);
    fft_rec(a, lg-1, rev);
    fft_rec(a+len, lg-1, rev);

    forn(i, len) {
        base w = ang[lg][i];
        if (rev) {
            w.im *= -1;
        }
        base u = a[i];
        base v = a[i+len] * w;
        a[i] = u + v;
        a[i+len] = u - v;
    }
}

//n must be power of 2
void fft(base *a, int n, bool rev) {
    int lg = 0;
    while ((1<<lg) != n) {
        ++lg;
    }
    int j = 0, bit;
    for (int i = 1; i < n; ++i) {
        for (bit = n >> 1; bit & j; bit >>= 1)
            j ^= bit;
        j ^= bit;
        if (i < j) swap(a[i], a[j]);
    }
    fft_rec(a, lg, rev);
    if (rev) forn(i, n) {
        a[i] = a[i] * (1.0 / n);
    }
}

const int maxn = 1050000;

int n;
base a[maxn];
base b[maxn];

void test() {
    int n = 8;
    init_fft();
    base a[8] = {1,3,5,2,4,6,7,1};
    base b[16];
    fft(b, 16, 0);
    fft(a, n, 0);
    forn(i, n) cout << a[i].re << " "; cout << endl;
    forn(i, n) cout << a[i].im << " "; cout << endl;
    // 29 -5.82843 -7 -0.171573 5 -0.171573 -7 -5.82843
    // 0 -3.41421 6 0.585786 0 -0.585786 -6 3.41421
}
```

21   math/fft_inv.cpp

```cpp
vector <int> mul(vector <int> a, vector <int> b,
        bool carry = true) {
    int n = sz(a);
    if (carry) {
        a.resize(n * 2);
        b.resize(n * 2);
    }
    fft(a.data(), a.size(), false);
    fft(b.data(), b.size(), false);
    for (int i = 0; i < sz(a); ++i)
        a[i] = mul(a[i], b[i]);
    fft(a.data(), a.size(), true);
    a.resize(n);
    return a;
}

vector <int> inv(vector <int> v) {
    int n = 1;
    while (n < sz(v))
        n <<= 1;
    v.resize(n, 0);
    vector <int> res(1, binpow(v[0], mod - 2));
    for (int k = 1; k < n; k <<= 1) {
        vector <int> A(k * 2, 0);
        copy(v.begin(), v.begin() + k, A.begin());
        vector <int> C = res;
        C.resize(k * 2, 0);
        A = mul(A, C, false);
        for (int i = 0; i < 2 * k; ++i)
            A[i] = sub(0, A[i]);
        A[0] = sum(A[0], 1);
        for (int i = 0; i < k; ++i)
            assert(A[i] == 0);
        copy(A.begin() + k, A.end(), A.begin());
        A.resize(k);
        vector <int> B(k);
        copy(v.begin() + k, v.begin() + 2 * k, B.begin());
        C.resize(k);
        B = mul(B, C);
        for (int i = 0; i < k; ++i)
            A[i] = sub(A[i], B[i]);
        A = mul(A, C);
        res.resize(k * 2);
        copy(A.begin(), A.end(), res.begin() + k);
    }
    return res;
}
```

22   math/golden_search.cpp

```cpp
ld f(ld x) {
    return 5 * x * x + 100 * x + 1; //-10 is minimum
}

ld goldenSearch(ld l, ld r) {
    ld phi = (1 + sqrtl(5)) / 2;
    ld resphi = 2 - phi;
    ld x1 = l + resphi * (r - l);
    ld x2 = r - resphi * (r - l);
    ld f1 = f(x1);
    ld f2 = f(x2);
    forn (iter, 60) {
        if (f1 < f2) {
            r = x2;
            x2 = x1;
            f2 = f1;
            x1 = l + resphi * (r - l);
            f1 = f(x1);
        } else {
            l = x1;
            x1 = x2;
            f1 = f2;
            x2 = r - resphi * (r - l);
            f2 = f(x2);
        }
    }
    return (x1 + x2) / 2;
}

int main() {
    std::cout << goldenSearch(-100, 100) << '\n';
}
```

23　math/numbers.tex

- Simpson and Gauss numerical integration:

$\int_a^b f(x)\mathrm{d}x = (b - a)/6 \cdot (f(a) + 4(f(a + b)/2) + f(b))$

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

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

- FFT modules:

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 (check)

- Fibonacci numbers:

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

- Powers of two

$2^{31} = 2\,147\,483\,648 = 2.1 \cdot 10^9$

$2^{32} = 4\,294\,967\,296 = 4.2 \cdot 10^9$

$2^{63} = 9\,223\,372\,036\,854\,775\,808 = 9.2 \cdot 10^{18}$

$2^{64} = 18\,446\,744\,073\,709\,551\,616 = 1.8 \cdot 10^{19}$

- Highly composite numbers

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

- Misc

    - Расстояние между точками по сфере: $L = R \cdot \arccos(\cos\theta_1 \cdot \cos\theta_2 + \sin\theta_1 \cdot \sin\theta_2 \cdot \cos(\varphi_1 - \varphi_2))$, где θ — широты (от $-\frac{\pi}{2}$ до $\frac{\pi}{2}$), φ — долготы (от $-\pi$ до π).

    - Объём шарового сегмента: $V = \pi h^2(R - \frac{1}{3}h)$, где h — высота от вершины сектора до секущей плоскости

    - Площадь поверхности шарового сегмента: $S = 2\pi R h$, где h — высота.

- 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

24　math/simplex.cpp

```cpp
namespace Simplex {

ld D[maxm][maxn]; // [n+2][m+2]
int B[maxm];
int N[maxn];
ld x[maxn];
int n, m;

//x >= 0, Ax <= b, c^Tx -> max
void init(int _n, int _m, ld A[][maxn], ld *b, ld *c) {
    n = _n, m = _m;
    forn (i, m)
        forn (j, n)
            D[i][j] = -A[i][j];
    forn (i, m) {
        D[i][n] = 1;
        D[i][n + 1] = b[i];
    }
    forn (j, n) {
        D[m][j] = c[j];
        D[m + 1][j] = 0;
    }
    D[m][n + 1] = D[m][n] = D[m + 1][n + 1] = 0;
    D[m + 1][n] = -1;
    iota(B, B + m, n);
    iota(N, N + n, 0);
    N[n] = -1;
}

void pivot(int b, int nb) {
    assert(D[b][nb] != 0);
    ld q = 1. / -D[b][nb];
    D[b][nb] = -1;
    forn (i, n + 2)
        D[b][i] *= q;
    forn (i, m + 2) {
        if (i == b)
            continue;
        ld coef = D[i][nb];
        D[i][nb] = 0;
        forn (j, n + 2)
            D[i][j] += coef * D[b][j];
    }
    swap(B[b], N[nb]);
}

bool betterN(int f, int i, int j) {
    if (eq(D[f][i], D[f][j]))
        return N[i] < N[j];
    return D[f][i] > D[f][j];
}

bool betterB(int nb, int i, int j) {
    ld ai = D[i][n + 1] / D[i][nb];
    ld aj = D[j][n + 1] / D[j][nb];
    if (eq(ai, aj))
        return B[i] < B[j];
    return ai > aj;
}

bool simplex(int phase) {
    int f = phase == 1 ? m : m + 1;
    while (true) {
        int nb = -1;
        forn (i, n + 1) {
            if (N[i] == -1 && phase == 1)
                continue;
            if (nb == -1 || betterN(f, i, nb))
                nb = i;
        }
        if (D[f][nb] <= eps)
            return phase == 1;
        assert(nb != -1);

        int b = -1;
        forn (i, m) {
            if (D[i][nb] >= -eps)
                continue;
            if (b == -1 || betterB(nb, i, b))
                b = i;
        }
        if (b == -1)
            return false;
        pivot(b, nb);
        if (N[nb] == -1 && phase == 2)
            return true;
    }
}

ld solve() {
    int b = -1;
```

```
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
```

25   math/stuff.cpp

```
1  const int M = 1e6;
2  int phi[M];
3  void 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  }
10 int inv[M];
11 void 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 }
18 int 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 }
30 int 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
49 const ll n = 1000000000000LL;
50 const ll L = 1000000;
51 int small[L+1];
52 ll large[L+1];
53 void 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 }
76 ll pi(ll x) {
77     if (x > L) return small[n/x];
78     else return large[x];
79 }
80
81 int 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

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

27   strings/duval_manacher.cpp

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

28   strings/eertree.cpp

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

29   strings/suffix_array.cpp

```
 1 string s;
 2 int n;
 3 int sa[maxn], new_sa[maxn], cls[maxn], new_cls[maxn],
 4     cnt[maxn], lcp[maxn];
 5 int n_cls;
 6
 7 void build() {
 8     n_cls = 256;
 9     forn(i, n) {
10         sa[i] = i;
11         cls[i] = s[i];
12     }
13     for (int d = 0; d < n; d = d ? d*2 : 1) {
14
15         forn(i, n) new_sa[i] = (sa[i] - d + n) % n;
16         forn(i, n_cls) cnt[i] = 0;
17         forn(i, n) ++cnt[cls[i]];
18         forn(i, n_cls) cnt[i+1] += cnt[i];
19         for (int i = n-1; i >= 0; --i)
20             sa[--cnt[cls[new_sa[i]]]] = new_sa[i];
21
22         n_cls = 0;
23         forn(i, n) {
24             if (i && (cls[sa[i]] != cls[sa[i-1]] ||
25                     cls[(sa[i]+d)%n] != cls[(sa[i-1]+d)%n])) {
26                 ++n_cls;
27             }
28             new_cls[sa[i]] = n_cls;
29         }
30         ++n_cls;
31         forn(i, n) cls[i] = new_cls[i];
32     }
33
34     // cls is also a inv 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
47 int main() {
48     cin >> s;
49     s += '$';
50     n = s.length();
51     build();
52     forn(i, n) {
53         cout << s.substr(sa[i]) << endl;
54         cout << lcp[i] << endl;
55     }
56 }
```

30   strings/ukkonen.cpp

```cpp
string s;
const int alpha = 26;

namespace SuffixTree {
    struct Node {
        Node *to[alpha];
        Node *lnk, *par;
        int l, r;

        Node(int l, int r): l(l), r(r) {
            memset(to, 0, sizeof(to));
            lnk = par = 0;
        }
    };

    Node *root, *blank, *cur;
    int pos;

    void init() {
        root = new Node(0, 0);
        blank = new Node(0, 0);
        forn (i, alpha)
            blank->to[i] = root;
        root->lnk = root->par = blank->lnk = blank->par = blank;
        cur = root;
        pos = 0;
    }

    int at(int id) {
        return s[id];
    }

    void goDown(int l, int r) {
        if (l >= r)
            return;
        if (pos == cur->r) {
            int c = at(l);
            assert(cur->to[c]);
            cur = cur->to[c];
            pos = min(cur->r, cur->l + 1);
            ++l;
        } else {
            int delta = min(r - l, cur->r - pos);
            l += delta;
            pos += delta;
        }
        goDown(l, r);
    }

    void goUp() {
        if (pos == cur->r && cur->lnk) {
            cur = cur->lnk;
            pos = cur->r;
            return;
        }
        int l = cur->l, r = pos;
        cur = cur->par->lnk;
        pos = cur->r;
        goDown(l, r);
    }

    void setParent(Node *a, Node *b) {
        assert(a);
        a->par = b;
        if (b)
            b->to[at(a->l)] = a;
    }

    void addLeaf(int id) {
        Node *x = new Node(id, inf);
        setParent(x, cur);
    }

    void splitNode() {
        assert(pos != cur->r);
        Node *mid = new Node(cur->l, pos);
        setParent(mid, cur->par);
        cur->l = pos;
        setParent(cur, mid);
        cur = mid;
    }

    bool canGo(int c) {
        if (pos == cur->r)
            return cur->to[c];
        return at(pos) == c;
    }

    void fixLink(Node *&bad, Node *newBad) {
        if (bad)
            bad->lnk = cur;
        bad = newBad;
    }

    void addCharOnPos(int id) {
        Node *bad = 0;
        while (!canGo(at(id))) {
            if (cur->r != pos) {
                splitNode();
                fixLink(bad, cur);
                bad = cur;
            } else {
                fixLink(bad, 0);
            }
            addLeaf(id);
            goUp();
        }
        fixLink(bad, 0);
        goDown(id, id + 1);
    }

    int cnt(Node *u, int ml) {
        if (!u)
            return 0;
        int res = min(ml, u->r) - u->l;
        forn (i, alpha)
            res += cnt(u->to[i], ml);
        return res;
    }

    void build(int l) {
        init();
        forn (i, l)
            addCharOnPos(i);
    }
};
```

31   structures/centroids.cpp

```cpp
const int maxn = 100100;
const int LG = 18; //2*maxn <= 2^LG

vector<int> g[LG][maxn];
int rt[LG][maxn];
int from[LG][maxn];

namespace Cenroids {

int D;
int cnt[maxn];
int CENTER, BOUND;

void pre(int u, int prev = -1) {
    cnt[u] = 1;
    for (int v: g[D][u]) {
        if (v == prev)
            continue;
        pre(v, u);
        cnt[u] += cnt[v];
    }
}

void findCenter(int u, int prev = -1, int up = 0) {
    int worst = up;
    for (int v: g[D][u]) {
        if (v == prev)
            continue;
        findCenter(v, u, up + cnt[u] - cnt[v]);
        worst = max(worst, cnt[v]);
    }
    if (worst <= BOUND)
        CENTER = u;
}

void markAll(int u, int prev = -1, int subtree = -1) {
    rt[D][u] = CENTER;
    from[D][u] = subtree;
    for (int v: g[D][u]) {
        if (v == prev)
            continue;
        g[D + 1][u].push_back(v);
        g[D + 1][v].push_back(u);
        if (subtree == -1)
            markAll(v, u, v);
        else
            markAll(v, u, subtree);
    }
}

void decompose(int u, int depth = 0) {
    D = depth;
    pre(u);
    CENTER = -1, BOUND = cnt[u] / 2;
    findCenter(u);
    assert(CENTER != -1);
    markAll(u);
    u = CENTER;
    D = depth + 1;
    for (int v: g[D][u]) {
        auto it = find(g[D][v].begin(), g[D][v].end(), u);
        assert(it != g[D][v].end());
        g[D][v].erase(it);
    }
    for (int v: g[D][u])
        decompose(v, depth + 1);
}
};
```

32   structures/convex_hull_trick.cpp

```cpp
/*
    WARNING!!!
    - finds maximum of A*x+B
    - double check max coords for int/long long overflow
    - set min x query in put function
    - add lines with non-descending A coefficient
*/
struct FastHull {
    int a[maxn];
    ll b[maxn];
    ll p[maxn];
    int c;

    FastHull(): c(0) {}

    ll get(int x) {
        if (c == 0)
            return -infl;
        int pos = upper_bound(p, p + c, x) - p - 1;
        assert(pos >= 0);
        return (ll) a[pos] * x + b[pos];
    }

    ll divideCeil(ll p, ll q) {
        assert(q > 0);
        if (p >= 0)
            return (p + q - 1) / q;
        return -((-p) / q);
    }

    void put(int A, ll B) {
        while (c > 0) {
            if (a[c - 1] == A && b[c - 1] >= B)
                return;
            ll pt = p[c - 1];
            if (a[c - 1] * pt + b[c - 1] < A * pt + B) {
                --c;
                continue;
            }
            ll q = A - a[c - 1];
            ll np = divideCeil(b[c - 1] - B, q);
            p[c] = np;
            a[c] = A;
            b[c] = B;
            ++c;
            return;
        }
        if (c == 0) {
            a[c] = A, b[c] = B;
            p[c] = -1e9; //min x query
            ++c;
            return;
        }
    }
};

struct SlowHull {
    vector<pair<int, ll>> v;

    void put(int a, ll b) {
        v.emplace_back(a, b);
    }

    ll get(ll x) {
        ll best = -infl;
        for (auto p: v)
            best = max(best, p.first * x + p.second);
        return best;
    }
};

int main() {
    FastHull hull1;
    SlowHull hull2;
    vector<int> as;
    forn (ii, 10000)
        as.push_back(rand() % int(1e8));
    sort(as.begin(), as.end());
    forn (ii, 10000) {
        int b = rand() % int(1e8);
        hull1.put(as[ii], b);
        hull2.put(as[ii], b);
        int x = rand() % int(2e8 + 1) - int(1e8);
        assert(hull1.get(x) == hull2.get(x));
    }
}
```

33 structures/heavy_light.cpp

```cpp
const int maxn = 100500;
const int maxd = 17;

vector<int> g[maxn];

struct Tree {
    vector<int> t;
    int base;

    Tree(): base(0) {
    }

    Tree(int n) {
        base = 1;
        while (base < n)
            base *= 2;
        t = vector<int>(base * 2, 0);
    }

    void put(int v, int delta) {
        assert(v < base);
        v += base;
        t[v] += delta;
        while (v > 1) {
            v /= 2;
            t[v] = max(t[v * 2], t[v * 2 + 1]);
        }
    }

    //Careful here: cr = 2 * maxn
    int get(int l, int r, int v=1, int cl=0, int cr = 2*maxn) {
        cr = min(cr, base);
        if (l <= cl && cr <= r)
            return t[v];
        if (r <= cl || cr <= l)
            return 0;
        int cc = (cl + cr) / 2;
        return max(get(l, r, v * 2, cl, cc),
            get(l, r, v * 2 + 1, cc, cr));
    }
};

namespace HLD {
    int h[maxn];
    int timer;
    int in[maxn], out[maxn], cnt[maxn];
    int p[maxd][maxn];
    int vroot[maxn];
    int vpos[maxn];
    int ROOT;
    Tree tree[maxn];

    void dfs1(int u, int prev) {
        p[0][u] = prev;
        in[u] = timer++;
        cnt[u] = 1;
        for (int v: g[u]) {
            if (v == prev)
                continue;
            h[v] = h[u] + 1;
            dfs1(v, u);
            cnt[u] += cnt[v];
        }
        out[u] = timer;
    }

    int dfs2(int u, int prev) {
        int to = -1;
        for (int v: g[u]) {
            if (v == prev)
                continue;
            if (to == -1 || cnt[v] > cnt[to])
                to = v;
        }
        int len = 1;
        for (int v: g[u]) {
            if (v == prev)
                continue;
            if (to == v) {
                vpos[v] = vpos[u] + 1;
                vroot[v] = vroot[u];
                len += dfs2(v, u);
            }
            else {
                vroot[v] = v;
                vpos[v] = 0;
                dfs2(v, u);
            }
        }
        if (vroot[u] == u)
            tree[u] = Tree(len);
```

```cpp
        return len;
    }

    void init(int n) {
        timer = 0;
        h[ROOT] = 0;
        dfs1(ROOT, ROOT);
        forn (d, maxd - 1)
            forn (i, n)
                p[d + 1][i] = p[d][p[d][i]];
        vroot[ROOT] = ROOT;
        vpos[ROOT] = 0;
        dfs2(ROOT, ROOT);
        //WARNING: init all trees
    }

    bool isPrev(int u, int v) {
        return in[u] <= in[v] && out[v] <= out[u];
    }

    int lca(int u, int v) {
        for (int d = maxd - 1; d >= 0; --d)
            if (!isPrev(p[d][u], v))
                u = p[d][u];
        if (!isPrev(u, v))
            u = p[0][u];
        return u;
    }

    //for each v: h[v] >= toh
    int getv(int u, int toh) {
        int res = 0;
        while (h[u] >= toh) {
            int rt = vroot[u];
            int l = max(0, toh - h[rt]), r = vpos[u] + 1;
            res = max(res, tree[rt].get(l, r));
            if (rt == ROOT)
                break;
            u = p[0][rt];
        }
        return res;
    }

    int get(int u, int v) {
        int w = lca(u, v);
        return max(getv(u, h[w]), getv(v, h[w] + 1));
    }

    void put(int u, int val) {
        int rt = vroot[u];
        int pos = vpos[u];
        tree[rt].put(pos, val);
    }
};
```

34   structures/linkcut.cpp

```
1 namespace LinkCut {
2
3 typedef struct _node {
4     _node *l, *r, *p, *pp;
5     int size; bool rev;
6     _node();
7
8     explicit _node(nullptr_t) {
9         l = r = p = pp = this;
10        size = rev = 0;
11    }
12
13    void push() {
14        if (rev) {
15            l->rev ^= 1; r->rev ^= 1;
16            rev = 0; swap(l,r);
17        }
18    }
19
20    void update();
21 }* node;
22
23 node None = new _node(nullptr);
24 node v2n[maxn];
25
26 _node::_node(){
27     l = r = p = pp = None;
28     size = 1; rev = false;
29 }
30
31 void _node::update() {
32     size = (this != None) + l->size + r->size;
33     l->p = r->p = this;
34 }
35
36 void rotate(node v) {
37     assert(v != None && v->p != None);
38     assert(!v->rev);
39     assert(!v->p->rev);
40     node u = v->p;
41     if (v == u->l)
42         u->l = v->r, v->r = u;
43     else
44         u->r = v->l, v->l = u;
45     swap(u->p,v->p);
46     swap(v->pp,u->pp);
47     if (v->p != None) {
48         assert(v->p->l == u || v->p->r == u);
49         if (v->p->r == u)
50             v->p->r = v;
51         else
52             v->p->l = v;
53     }
54     u->update();
55     v->update();
56 }
57
58 void bigRotate(node v) {
59     assert(v->p != None);
60     v->p->p->push();
61     v->p->push();
62     v->push();
63     if (v->p->p != None) {
64         if ((v->p->l == v) ^ (v->p->p->r == v->p))
65             rotate(v->p);
66         else
67             rotate(v);
68    }
69    rotate(v);
70 }
71
72 inline void splay(node v) {
73     while (v->p != None)
74         bigRotate(v);
75 }
76
77 inline void splitAfter(node v) {
78     v->push();
79     splay(v);
80     v->r->p = None;
81     v->r->pp = v;
82     v->r = None;
83     v->update();
84 }
85
86 void expose(int x) {
87     node v = v2n[x];
88     splitAfter(v);
89     while (v->pp != None) {
90         assert(v->p == None);
91         splitAfter(v->pp);
```

```
92         assert(v->pp->r == None);
93         assert(v->pp->p == None);
94         assert(!v->pp->rev);
95         v->pp->r = v;
96         v->pp->update();
97         v = v->pp;
98         v->r->pp = None;
99     }
100    assert(v->p == None);
101    splay(v2n[x]);
102 }
103
104 inline void makeRoot(int x) {
105     expose(x);
106     assert(v2n[x]->p == None);
107     assert(v2n[x]->pp == None);
108     assert(v2n[x]->r == None);
109     v2n[x]->rev ^= 1;
110 }
111
112 inline void link(int x, int y) {
113     makeRoot(x);
114     v2n[x]->pp = v2n[y];
115 }
116
117 inline void cut(int x, int y) {
118     expose(x);
119     splay(v2n[y]);
120     if (v2n[y]->pp != v2n[x]) {
121         swap(x,y);
122         expose(x);
123         splay(v2n[y]);
124         assert(v2n[y]->pp == v2n[x]);
125     }
126     v2n[y]->pp = None;
127 }
128
129 inline int get(int x, int y) {
130     if (x == y)
131         return 0;
132     makeRoot(x);
133     expose(y);
134     expose(x);
135     splay(v2n[y]);
136     if (v2n[y]->pp != v2n[x])
137         return -1;
138     return v2n[y]->size;
139 }
140
141 }
```

35   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 }
```
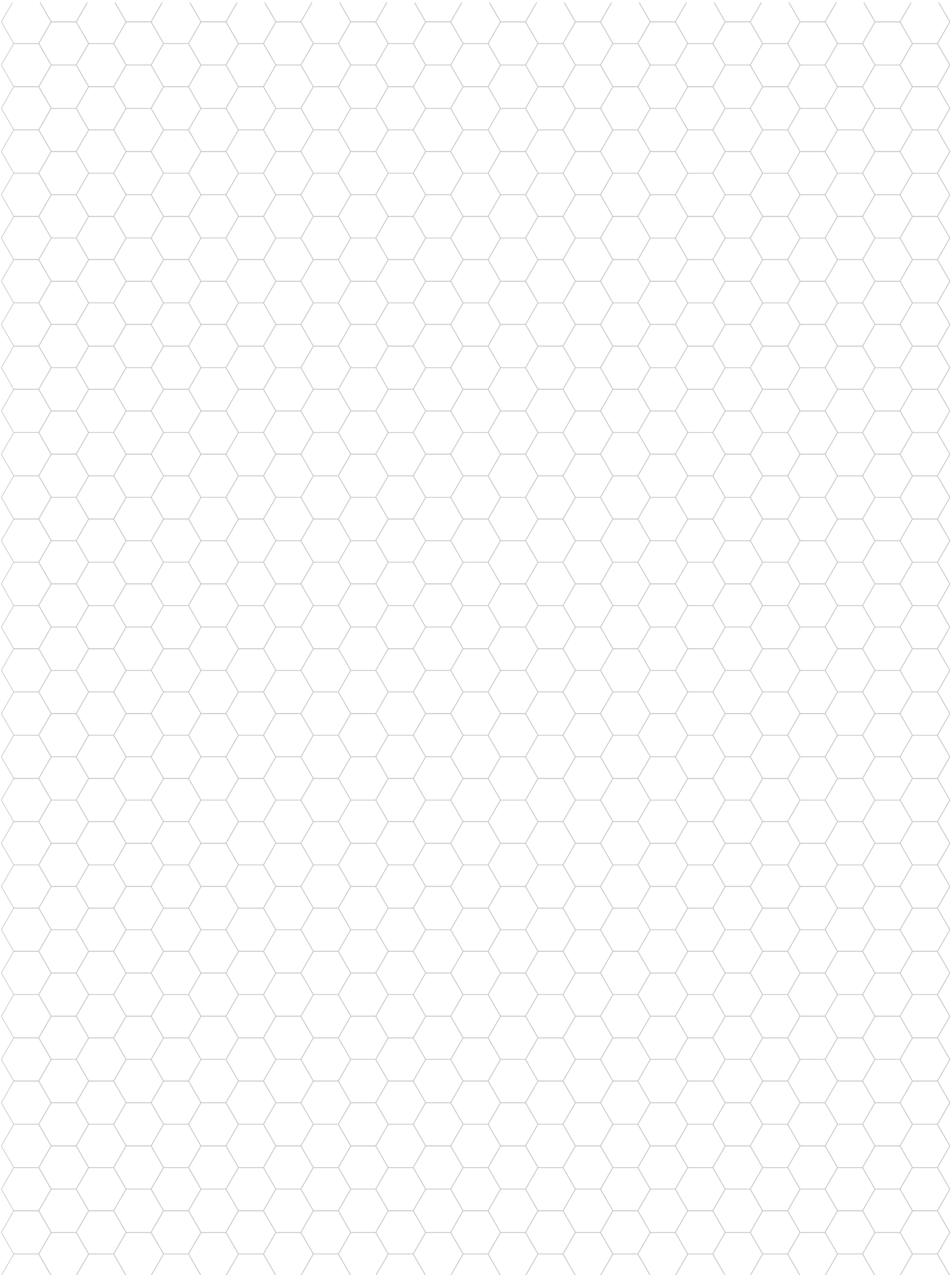
36   structures/treap.cpp

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

37 Сеточка

38 Сеточка