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 = 100100;
struct Edge {
    int to;
    ll c, f;
    Edge(int to, ll c): to(to), c(c), f(0) {}
};

vector<Edge> es;
vector<int> g[maxn];
int q[maxn], d[maxn], pos[maxn];
int N, S, T;

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

bool bfs() {
    fill(d, d + N, maxn);
    d[S] = 0, q[0] = S;
    int rq = 1;
    forn (lq, rq) {
        int u = q[lq];
        for (int id: g[u]) {
            if (es[id].c == es[id].f)
                continue;
            int v = es[id].to;
            if (d[v] == maxn) {
                d[v] = d[u] + 1;
                q[rq++] = v;
            }
        }
    }
    return d[T] != maxn;
}

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

ll dinic(int S, int T) {
    Dinic::S = S, Dinic::T = T;
    ll res = 0;
    while (bfs()) {
        fill(pos, pos + N, 0);
        while (ll cur = dfs(S, infl))
            res += cur;
    }
    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

```cpp
namespace MinCost {
const ll infc = 1e12;

struct Edge {
    int to;
    ll c, f, cost;

    Edge(int to, ll c, ll cost): to(to), c(c), f(0), cost(cost)
    {  }
};

int N, S, T;
int totalFlow;
ll totalCost;
const int maxn = 505;
vector<Edge> edge;
vector<int> g[maxn];

void addEdge(int u, int v, ll c, ll cost) {
    g[u].push_back(edge.size());
    edge.emplace_back(v, c, cost);
    g[v].push_back(edge.size());
    edge.emplace_back(u, 0, -cost);
}

ll dist[maxn];
int fromEdge[maxn];

bool inQueue[maxn];
bool fordBellman() {
    forn (i, N)
        dist[i] = infc;
    dist[S] = 0;
    inQueue[S] = true;
    vector<int> q;
    q.push_back(S);
    for (int ii = 0; ii < int(q.size()); ++ii) {
        int u = q[ii];
        inQueue[u] = false;
        for (int e: g[u]) {
            if (edge[e].f == edge[e].c)
                continue;
            int v = edge[e].to;
            ll nw = edge[e].cost + dist[u];
            if (nw >= dist[v])
                continue;
            dist[v] = nw;
            fromEdge[v] = e;
            if (!inQueue[v]) {
                inQueue[v] = true;
                q.push_back(v);
            }
        }
    }
    return dist[T] != infc;
}

ll pot[maxn];
bool dikstra() {
    typedef pair<ll, int> Pair;
    priority_queue<Pair, vector<Pair>, greater<Pair>> q;
    forn (i, N)
        dist[i] = infc;
    dist[S] = 0;
    q.emplace(dist[S], S);
    while (!q.empty()) {
        int u = q.top().second;
        ll cdist = q.top().first;
        q.pop();
        if (cdist != dist[u])
            continue;
        for (int e: g[u]) {
            int v = edge[e].to;
            if (edge[e].c == edge[e].f)
                continue;
            ll w = edge[e].cost + pot[u] - pot[v];
            assert(w >= 0);
            ll ndist = w + dist[u];
            if (ndist >= dist[v])
                continue;
            dist[v] = ndist;
            fromEdge[v] = e;
            q.emplace(dist[v], v);
        }
    }
    if (dist[T] == infc)
        return false;
    forn (i, N) {
        if (dist[i] == infc)
            continue;
        pot[i] += dist[i];
    }
    return true;
}

bool push() {
    //2 variants
    //if (!fordBellman())
    if (!dikstra())
        return false;
    ++totalFlow;
    int u = T;
    while (u != S) {
        int e = fromEdge[u];
        totalCost += edge[e].cost;
        edge[e].f++;
        edge[e ^ 1].f--;
        u = edge[e ^ 1].to;
    }
    return true;
}

//min-cost-circulation
ll d[maxn][maxn];
int dfrom[maxn][maxn];
int level[maxn];
void circulation() {
    while (true) {
        int q = 0;
        fill(d[0], d[0] + N, 0);
        forn (iter, N) {
            fill(d[iter + 1], d[iter + 1] + N, infc);
            forn (u, N)
                for (int e: g[u]) {
                    if (edge[e].c == edge[e].f)
                        continue;
                    int v = edge[e].to;
                    ll ndist = d[iter][u] + edge[e].cost;
                    if (ndist >= d[iter + 1][v])
                        continue;
                    d[iter + 1][v] = ndist;
                    dfrom[iter + 1][v] = e;
                }
            q ^= 1;
        }
        int w = -1;
        ld mindmax = 1e18;
        forn (u, N) {
            ld dmax = -1e18;
            forn (iter, N)
                dmax = max(dmax,
                    (d[N][u] - d[iter][u]) / ld(N - iter));
            if (mindmax > dmax)
                mindmax = dmax, w = u;
        }
        if (mindmax >= 0)
            break;
        fill(level, level + N, -1);
        int k = N;
        while (level[w] == -1) {
            level[w] = k;
            w = edge[dfrom[k--][w] ^ 1].to;
        }
        int k2 = level[w];
        ll delta = infc;
        while (k2 > k) {
            int e = dfrom[k2--][w];
            delta = min(delta, edge[e].c - edge[e].f);
            w = edge[e ^ 1].to;
        }
        k2 = level[w];
        while (k2 > k) {
            int e = dfrom[k2--][w];
            totalCost += edge[e].cost * delta;
            edge[e].f += delta;
            edge[e ^ 1].f -= delta;
            w = edge[e ^ 1].to;
        }
    }
}
} // namespace MinCost

int main() {
    MinCost::N = 3, MinCost::S = 1, MinCost::T = 2;
    MinCost::addEdge(1, 0, 3, 5);
    MinCost::addEdge(0, 2, 4, 6);
    while (MinCost::push());
    cout << MinCost::totalFlow << ' '
        << MinCost::totalCost << '\n'; //3 33
}
```

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/convex_hull.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
#define forn(i, n) for (int i = 0; i < int(n); ++i)
#define sz(x) ((int) (x).size())

#include "primitives.cpp"

bool cmpAngle(const pt &a, const pt &b) {
    bool ar = a.right(), br = b.right();
    if (ar ^ br)
        return ar;
    return a % b > eps;
}

struct Hull {
    vector<pt> top, bot;

    void append(pt p) {
        while (bot.size() > 1 && (p - bot.back())
                % (bot.back() - *next(bot.rbegin())) >= -eps)
            bot.pop_back();
        bot.push_back(p);
        while (top.size() > 1 && (p - top.back())
                % (top.back() - *next(top.rbegin())) <= eps)
            top.pop_back();
        top.push_back(p);
    }

    void build(vector<pt> h) {
        sort(h.begin(), h.end());
        h.erase(unique(h.begin(), h.end()), h.end());
        top.clear(), bot.clear();
        for (pt p: h)
            append(p);
    }

    pt kth(int k) {
        if (k < sz(bot))
            return bot[k];
        else
            return top[sz(top) - (k - sz(bot)) - 2];
    }

    pt mostDistant(pt dir) {
        if (bot.empty()) {
            //empty hull
            return pt{1e18, 1e18};
        }
        if (bot.size() == 1)
            return bot.back();
        dir = dir.rot();
        int n = sz(top) + sz(bot) - 2;
        int L = -1, R = n;
        while (L + 1 < R) {
            int C = (L + R) / 2;
            pt v = kth((C + 1) % n) - kth(C);
            if (cmpAngle(dir, v)) //finds upper bound
                R = C;
            else
                L = C;
        }
        return kth(R % n);
    }
};
```

8 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;
}
```

9 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);
        }
    }
}
```

10 geometry/polygon.cpp

```
bool pointInsidePolygon(pt a, pt *p, int n) {
    double sumAng = 0;
    forn (i, n) {
        pt A = p[i], B = p[(i + 1) % n];
        if (pointInsideSegment(a, A, B))
            return true;
        sumAng += atan2((A - a) % (B - a), (A - a) * (B - a));
    }
    return fabs(sumAng) > 1;
}

//p must be oriented counterclockwise
bool segmentInsidePolygon(pt a, pt b, pt *p, int n) {
    if (!pointInsidePolygon((a + b) / 2, p, n))
        return false;
    if (a == b)
        return true;
    forn (i, n) {
        pt c = p[i];
        if (ze((a - c) % (b - c)) &&
                (a - c) * (b - c) < -eps) {
            //point on segment
            pt pr = p[(i + n - 1) % n];
            pt nx = p[(i + 1) % n];
            if ((c - pr) % (nx - c) > eps)
                return false;
            ld s1 = (pr - a) % (b - a);
            ld s2 = (nx - a) % (b - a);
            if ((s1 > eps || s2 > eps) &&
                    (s1 < -eps || s2 < -eps))
                return false;
        }
        //interval intersection
        pt d = p[(i + 1) % n];
        ld s1 = (a - c) % (d - c);
        ld s2 = (b - c) % (d - c);
        if (s1 >= -eps && s2 >= -eps)
            continue;
        if (s1 <= eps && s2 <= eps)
            continue;

        s1 = (c - a) % (b - a);
        s2 = (d - a) % (b - a);
        if (s1 >= -eps && s2 >= -eps)
            continue;
        if (s1 <= eps && s2 <= eps)
            continue;

        return false;
    }
    return true;
}
```

11 geometry/polygon_tangents.cpp

```
struct Hull {
    vector<pt> p, v;
    pt M;
    int n;

    void build() {
        sort(p.begin(), p.end());
        p.erase(unique(p.begin(), p.end()), p.end());
        vector<pt> top, bot;
        for (auto P: p) {
            while (sz(top) > 1 && (P - top.back()) %
                    (top.back() - *next(top.rbegin())) <= eps)
                top.pop_back();
            top.push_back(P);
            while (sz(bot) > 1 && (P - bot.back()) %
                    (bot.back() - *next(bot.rbegin())) >= -eps)
                bot.pop_back();
            bot.push_back(P);
        }
        if (sz(top))
            top.pop_back();
        reverse(top.begin(), top.end());
        if (sz(top))
            top.pop_back();
        p = bot;
        p.insert(p.end(), top.begin(), top.end());

        n = sz(p);
        M = accumulate(p.begin(), p.end(), pt{0, 0});
        M = M * (1. / n);
        v.resize(n);
        forn (i, n)
            v[i] = p[i] - M;
        int r = min_element(v.begin(), v.end(),
                cmpAngle) - v.begin();
        rotate(v.begin(), v.begin() + r, v.end());
        rotate(p.begin(), p.begin() + r, p.end());
        gassert(is_sorted(v.begin(), v.end(), cmpAngle));
    }

    bool visSide(pt a, int i) {
        return (p[(i + 1) % n] - a) % (p[i % n] - a) > eps;
    }

    bool vis(pt a, int i) {
        return visSide(a, i) || visSide(a, i + n - 1);
    }

    bool isTangent(pt a, int i) {
        return visSide(a, i) != visSide(a, i + n - 1);
    }

    int binSearch(int l, int r, pt a) {
        //tricky binsearch; l < r not necessarily
        while (abs(l - r) > 1) {
            int c = (l + r) / 2;
            if (vis(a, c))
                l = c;
            else
                r = c;
        }
        assert(isTangent(a, l));
        return l % n;
    }

    pair<int, int> tangents(pt a) {
        assert(n >= 3);
        if (a == M)
            return {-1, -1};
        int pos = lower_bound(v.begin(), v.end(), a - M,
                cmpAngle) - v.begin();
        pt L = p[(pos + n - 1) % n], R = p[pos % n];
        if ((R - L) % (a - L) >= -eps)
            return {-1, -1};
        int pos2 = lower_bound(v.begin(), v.end(), M - a,
                cmpAngle) - v.begin();
        assert(pos % n != pos2 % n);
        if (pos > pos2)
            pos2 += n;
        return {binSearch(pos, pos2, a),
                binSearch(pos + n - 1, pos2 - 1, a)};
    }

};
```

12 geometry/primitives.cpp

```cpp
//WARNING! do not forget to normalize vector (a,b)
struct line {
    ld a, b, c;
    int id;

    line(pt p1, pt p2) {
        gassert(p1 != p2);
        pt n = (p2 - p1).rot();
        n /= n.abs();
        a = n.x, b = n.y;
        c = -(n * p1);
    }

    bool right() const {
        return a > eps || (ze(a) && b > eps);
    }

    line(ld _a, ld _b, ld _c): a(_a), b(_b), c(_c) {
        ld d = pt{a, b}.abs();
        gassert(!ze(d));
        a /= d, b /= d, c /= d;
    }

    ld signedDist(pt p) {
        return p * pt{a, b} + c;
    }
};

ld pointSegmentDist(pt p, pt a, pt b) {
    ld res = min((p - a).abs(), (p - b).abs());
    if (a != b && (p - a) * (b - a) >= 0 &&
            (p - b) * (a - b) >= 0)
        res = min(res,
            fabsl((p - a) % (b - a)) / (b - a).abs());
    return res;
}

pt linesIntersection(line l1, line l2) {
    ld D = l1.a * l2.b - l1.b * l2.a;
    if (ze(D)) {
        if (eq(l1.c, l2.c)) {
            //equal lines
        } else {
            //no intersection
        }
    }
    ld dx = -l1.c * l2.b + l1.b * l2.c;
    ld dy = -l1.a * l2.c + l1.c * l2.a;
    pt res{dx / D, dy / D};
    //gassert(ze(l1.signedDist(res)));
    //gassert(ze(l2.signedDist(res)));
    return res;
}

bool pointInsideSegment(pt p, pt a, pt b) {
    if (!ze((p - a) % (p - b)))
        return false;
    return (a - p) * (b - p) <= eps;
}

bool checkSegmentIntersection(pt a, pt b, pt c, pt d) {
    if (ze((a - b) % (c - d))) {
        if (pointInsideSegment(a, c, d) ||
            pointInsideSegment(b, c, d) ||
            pointInsideSegment(c, a, b) ||
            pointInsideSegment(d, a, b)) {
            //intersection of parallel segments
            return true;
        }
        return false;
    }

    ld s1, s2;
    forn (q, 2) {
        s1 = (c - a) % (b - a);
        s2 = (d - a) % (b - a);
        if (s1 > eps && s2 > eps)
            return false;
        if (s1 < -eps && s2 < -eps)
            return false;
        swap(a, c), swap(b, d);
    }

    return true;
}

// WARNING! run checkSegmentIntersecion before and process
// parallel case manually
pt segmentsIntersection(pt a, pt b, pt c, pt d) {
    ld S = (b - a) % (d - c);
    ld s1 = (c - a) % (d - a);
    return a + (b - a) / S * s1;
}

vector<pt> circlesIntersction(pt a, ld r1, pt b, ld r2) {
    ld d2 = (a - b).abs2();
    ld d = (a - b).abs();

    if (a == b && eq(r1, r2)) {
        //equal circles
    }
    if (d2 - sqr(r1 + r2) > eps || sqr(r1 - r2) - d2 > eps) {
        //empty intersection
        return {};
    }
    int num = 2;
    if (eq(sqr(r1 + r2), d2) || eq(sqr(r1 - r2), d2))
        num = 1;
    ld cosa = (sqr(r1) + d2 - sqr(r2)) / ld(2 * r1 * d);
    ld oh = cosa * r1;
    pt h = a + ((b - a) / d * oh);
    if (num == 1)
        return {h};
    ld hp = sqrtl(max(0.L, 1 - cosa * cosa)) * r1;

    pt w = ((b - a) / d * hp).rot();
    return {h + w, h - w};
}

//a is circle center, p is point
vector<pt> circleTangents(pt a, ld r, pt p) {
    ld d2 = (a - p).abs2();
    ld d = (a - p).abs();

    if (sqr(r) - d2 > eps) {
        //no tangents
        return {};
    }
    if (eq(sqr(r), d2)) {
        //point lies on circle - one tangent
        return {p};
    }

    pt B = p - a;
    pt H = B * sqr(r) / d2;
    ld h = sqrtl(d2 - sqr(r)) * ld(r) / d;
    pt w = (B / d * h).rot();
    H = H + a;
    return {H + w, H - w};
}

vector<pt> lineCircleIntersection(line l, pt a, ld r) {
    ld d = l.signedDist(a);
    if (fabsl(d) - r > eps)
        return {};
    pt h = a - pt{l.a, l.b} * d;
    if (eq(fabsl(d), r))
        return {h};
    pt w(pt{l.a, l.b}.rot() * sqrtl(max<ld>(0, sqr(r)-sqr(d))));
    return {h + w, h - w};
}

//modified magic from e-maxx
vector<line> commonTangents(pt a, ld r1, pt b, ld r2) {
    if (a == b && eq(r1, r2)) {
        //equal circles
        return {};
    }
    vector<line> res;
    pt c = b - a;
    ld z = c.abs2();
    for (int i = -1; i <= 1; i += 2)
        for (int j = -1; j <= 1; j += 2) {
            ld r = r2 * j - r1 * i;
            ld d = z - sqr(r);
            if (d < -eps)
                continue;
            d = sqrtl(max<ld>(0, d));
            pt magic = pt{r, d} / z;
            line l(magic * c, magic % c, r1 * i);
            l.c -= pt{l.a, l.b} * a;
            res.push_back(l);
        }
    return res;
}
```

13 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;
```

14 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
}
```

15 graphs/directed_mst.cpp

```cpp
// WARNING: this code wasn't submitted anywhere

namespace TwoChinese {

struct Edge {
    int to, w, id;
    bool operator<(const Edge& other) const {
        return to < other.to || (to == other.to && w < other.w);
    }
};
typedef vector<vector<Edge>> Graph;

const int maxn = 2050;

// global, for supplementary algorithms
int b[maxn];
int tin[maxn], tup[maxn];
int dtime; // counter for tin, tout
vector<int> st;
int nc; // number of strongly connected components
int q[maxn];

int answer;

void tarjan(int v, const Graph& e, vector<int>& comp) {
    b[v] = 1;
    st.push_back(v);
    tin[v] = tup[v] = dtime++;

    for (Edge t: e[v]) if (t.w == 0) {
        int to = t.to;
        if (b[to] == 0) {
            tarjan(to, e, comp);
            tup[v] = min(tup[v], tup[to]);
        } else if (b[to] == 1) {
            tup[v] = min(tup[v], tin[to]);
        }
    }

    if (tin[v] == tup[v]) {
        while (true) {
            int t = st.back();
            st.pop_back();
            comp[t] = nc;
            b[t] = 2;
            if (t == v) break;
        }
        ++nc;
    }
}

vector<Edge> bfs(
    const Graph& e,const vi& init, const vi& comp)
{
    int n = e.size();
    forn(i, n) b[i] = 0;
    int lq = 0, rq = 0;
    for (int v: init) b[v] = 1, q[rq++] = v;

    vector<Edge> result;

    while (lq != rq) {
        int v = q[lq++];
        for (Edge t: e[v]) if (t.w == 0) {
            int to = t.to;
            if (b[to]) continue;
            if (!comp.empty() && comp[v] != comp[to]) continue;
            b[to] = 1;
            q[rq++] = to;
            result.push_back(t);
        }
    }

    return result;
}

// warning: check that each vertex is reachable from root
vector<Edge> run(Graph e, int root) {
    int n = e.size();

    // find minimum incoming weight for each vertex
    vector<int> minw(n, inf);
    forn(v, n) for (Edge t: e[v]) {
        minw[t.to] = min(minw[t.to], t.w);
    }
    forn(v, n) for (Edge &t: e[v]) if (t.to != root) {
        t.w -= minw[t.to];
    }
    forn(i, n) if (i != root) answer += minw[i];

    // check if each vertex is reachable from root by zero edges
    vector<Edge> firstResult = bfs(e, {root}, {});
    if ((int)firstResult.size() + 1 == n) {
        return firstResult;
    }

    // find stongly connected comp-s and build compressed graph
    vector<int> comp(n);
    forn(i, n) b[i] = 0;
    nc = 0;
    dtime = 0;
    forn(i, n) if (!b[i]) tarjan(i, e, comp);

    // multiple edges may be removed here if needed
    Graph ne(nc);
    forn(v, n) for (Edge t: e[v]) {
        if (comp[v] != comp[t.to]) {
            ne[comp[v]].push_back({comp[t.to], t.w, t.id});
        }
    }
    int oldnc = nc;

    // run recursively on compressed graph
    vector<Edge> subres = run(ne, comp[root]);

    // find incoming edge id for each component, init queue
    // if there is an edge (u, v) between different components
    // than v is added to queue
    nc = oldnc;
    vector<int> incomingId(nc);
    for (Edge e: subres) {
        incomingId[e.to] = e.id;
    }

    vector<Edge> result;
    vector<int> init;
    init.push_back(root);
    forn(v, n) for (Edge t: e[v]) {
        if (incomingId[comp[t.to]] == t.id) {
            result.push_back(t);
            init.push_back(t.to);
        }
    }

    // run bfs to add edges inside components and return answer
    vector<Edge> innerEdges = bfs(e, init, comp);
    result.insert(result.end(), all(innerEdges));

    assert((int)result.size() + 1 == n);
    return result;
}

} // namespace TwoChinese

void test () {
    auto res = TwoChinese::run({
        {{1,5,0},{2,5,1}},
        {{3,1,2}},
        {{1,2,3},{4,1,4}},
        {{1,1,5},{4,2,6}},
        {{2,1,7}}},
        0);
    cout << TwoChinese::answer << endl;
    for (auto e: res) cout << e.id << " ";
    cout << endl;
    // 9     0 6 2 7
}
```

16 graphs/edmonds_matching.cpp

```cpp
int n;
vi e[maxn];
int mt[maxn], p[maxn], base[maxn], b[maxn], blos[maxn];
int q[maxn];
int blca[maxn]; // used for lca

int lca(int u, int v) {
    forn(i, n) blca[i] = 0;
    while (true) {
        u = base[u];
        blca[u] = 1;
        if (mt[u] == -1) break;
        u = p[mt[u]];
    }
    while (!blca[base[v]]) {
        v = p[mt[base[v]]];
    }
    return base[v];
}

void mark_path(int v, int b, int ch) {
    while (base[v] != b) {
        blos[base[v]] = blos[base[mt[v]]] = 1;
        p[v] = ch;
        ch = mt[v];
        v = p[mt[v]];
    }
}

int find_path(int root) {
    forn(i, n) {
        base[i] = i;
        p[i] = -1;
        b[i] = 0;
    }

    b[root] = 1;
    q[0] = root;
    int lq = 0, rq = 1;
    while (lq != rq) {
        int v = q[lq++];
        for (int to: e[v]) {
            if (base[v] == base[to] || mt[v] == to) continue;
            if (to==root || (mt[to] != -1 && p[mt[to]] != -1)) {
                int curbase = lca(v, to);
                forn(i, n) blos[i] = 0;
                mark_path(v, curbase, to);
                mark_path(to, curbase, v);
                forn(i, n) if (blos[base[i]]) {
                    base[i] = curbase;
                    if (!b[i]) b[i] = 1, q[rq++] = i;
                }
            } else if (p[to] == -1) {
                p[to] = v;
                if (mt[to] == -1) {
                    return to;
                }
                to = mt[to];
                b[to] = 1;
                q[rq++] = to;

            }
        }
    }
    return -1;
}

int matching() {
    forn(i, n) mt[i] = -1;
    int res = 0;
    forn(i, n) if (mt[i] == -1) {
        int v = find_path(i);
        if (v != -1) {
            ++res;
            while (v != -1) {
                int pv = p[v], ppv = mt[p[v]];
                mt[v] = pv, mt[pv] = v;
                v = ppv;
            }
        }
    }
    return res;
}
```

17 graphs/euler_cycle.cpp

```cpp
struct Edge {
    int to, id;
};

bool usedEdge[maxm];
vector<Edge> g[maxn];
int ptr[maxn];

vector<int> cycle;
void eulerCycle(int u) {
    while (ptr[u] < sz(g[u]) && usedEdge[g[u][ptr[u]].id])
        ++ptr[u];
    if (ptr[u] == sz(g[u]))
        return;
    const Edge &e = g[u][ptr[u]];
    usedEdge[e.id] = true;
    eulerCycle(e.to);
    cycle.push_back(e.id);
    eulerCycle(u);
}

int edges = 0;
void addEdge(int u, int v) {
    g[u].push_back(Edge{v, edges});
    g[v].push_back(Edge{u, edges++});
}
```

18 math/factor.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;
        }
    }
}
```

19 math/fft.cpp

```
const int LG = 20;
typedef complex<ld> base;

vector<base> ang[LG + 5];

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

    for (int k = LG - 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 inv) {
    if (lg == 0)
        return;
    int hlen = 1 << (lg - 1);
    fft_rec(a, lg-1, inv);
    fft_rec(a + hlen, lg-1, inv);

    forn (i, hlen) {
        base w = ang[lg][i];
        if (inv)
            w = conj(w);
        base u = a[i];
        base v = a[i + hlen] * w;
        a[i] = u + v;
        a[i + hlen] = u - v;
    }
}

void fft(base *a, int lg, bool inv) {
    int n = 1 << 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, inv);
    if (inv) {
        forn(i, n)
            a[i] /= n;
    }
}

void test() {
    int lg = 3;
    int n = 1 << lg;
    init_fft();
    base a[] = {1,3,5,2,4,6,7,1};
    fft(a, lg, 0);
    forn(i, n)
        cout << a[i].real() << " ";
    cout << '\n';
    forn(i, n)
        cout << a[i].imag() << " ";
    cout << '\n';
    // 29 -5.82843 -7 -0.171573 5 -0.171573 -7 -5.82843
    // 0 -3.41421 6 0.585786 0 -0.585786 -6 3.41421
}
```

20 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;
}
```

21 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';
}
```

22 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: $S = 2\pi Rh$, 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

23 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;
    forn (i, m) {
        if (b == -1 || D[i][n + 1] < D[b][n + 1])
            b = i;
    }
    assert(b != -1);
    if (D[b][n + 1] < -eps) {
        pivot(b, n);
        if (!simplex(2) || D[m + 1][n + 1] < -eps)
            return -infl;
    }
    if (!simplex(1))
        return infl;

    forn (i, n)
        x[i] = 0;
    forn (i, m)
        if (B[i] < n)
            x[B[i]] = D[i][n + 1];

    return D[m][n + 1];
}

} //Simplex
```

24 math/stuff.cpp

```
const int M = 1e6;
int phi[M];
void calcPhi() {
    for (int i = 1; i < M; ++i)
        phi[i] = i;
    for (int j = 1; j < M; ++j)
        for (int i = 2 * j; i < M; i += j)
            phi[i] -= phi[j];
}
int inv[M];
void calcInv() {
    inv[1] = 1;
    for (int i = 2; i < M; ++i) {
        inv[i] = mul(sub(0, mod / i), inv[mod % i]);
        assert(mul(i, inv[i]) == 1);
    }
}
int gcd(int a, int b, int &x, int &y) {
    if (a == 0) {
        x = 0, y = 1;
        return b;
    }
    int x1, y1;
    int g = gcd(b % a, a, x1, y1);
    x = y1 - x1 * (b / a);
    y = x1;
    assert(a * x + b * y == g);
    return g;
}
int crt(int mod1, int mod2, int rem1, int rem2) {
    int r = (rem2 - (rem1 % mod2) + mod2) % mod2;
    int x, y;
    int g = gcd(mod1, mod2, x, y);
    assert(r % g == 0);

    x %= mod2;
    if (x < 0)
        x += mod2;

    int ans = (x * (r / g)) % mod2;
    ans = ans * mod1 + rem1;

    assert(ans % mod1 == rem1);
    assert(ans % mod2 == rem2);
    return ans;
}

// primes to N
const ll n = 1000000000000LL;
const ll L = 1000000;
int small[L+1];
ll large[L+1];
void calc_pi() {
    for (int i = 1; i <= L; ++i) {
        small[i] = i-1;
        large[i] = n / i - 1;
    }
    for (ll p = 2; p <= L; ++p) {
        if (small[p] == small[p-1]) continue;
        int cntp = small[p-1];
        ll p2 = p*p;
        ll np = n / p;
        for (int i = 1; i <= min(L, n / p2); ++i) {
            ll x = np / i;
            if (x <= L) {
                large[i] -= small[x] - cntp;
            } else {
                large[i] -= large[p*i] - cntp;
            }
        }
        for (int i = L; i >= p2; --i) {
            small[i] -= small[i/p] - cntp;
        }
    }
}
ll pi(ll x) {
    if (x > L) return small[n/x];
    else return large[x];
}

int main() {
    calcPhi();
    assert(phi[30] == 1 * 2 * 4);
    calcInv();
    int x, y;
    gcd(3, 5, x, y);
    gcd(15, 10, x, y);
    crt(15, 13, 2, 5);
    crt(17, 3, 15, 2);
    return 0;
}
```

25 strings/automaton.cpp

```
int t[maxn][26], lnk[maxn], len[maxn];
int sz;
int last;

void init() {
    sz = 3;
    last = 1;
    forn(i, 26) t[2][i] = 1;
    len[2] = -1;
    lnk[1] = 2;
}

void addchar(int c) {
    int nlast = sz++;
    len[nlast] = len[last] + 1;
    int p = last;
    for (; !t[p][c]; p = lnk[p]) {
        t[p][c] = nlast;
    }
    int q = t[p][c];
    if (len[p] + 1 == len[q]) {
        lnk[nlast] = q;
    } else {
        int clone = sz++;
        len[clone] = len[p] + 1;
        lnk[clone] = lnk[q];
        lnk[q] = lnk[nlast] = clone;
        forn(i, 26) t[clone][i] = t[q][i];
        for (; t[p][c] == q; p = lnk[p]) {
            t[p][c] = clone;
        }
    }
    last = nlast;
}

bool check(const string& s) {
    int v = 1;
    for (int c: s) {
        c -= 'a';
        if (!t[v][c]) return false;
        v = t[v][c];
    }
    return true;
}

int main() {
    string s;
    cin >> s;
    init();
    for (int i: s) {
        addchar(i-'a');
    }
    forn(i, s.length()) {
        assert(check(s.substr(i)));
    }
    cout << sz << endl;
    return 0;
}
```

26 strings/duval_manacher.cpp

```
/*
  Строка простая, если строго меньше всех суффиксов <=>
  наименьший циклический сдвиг - первый.
  Декомпозиция Линдона - разбиение s на w1, w2, ... wk -
  простые строки такие, что w1 >= w2 >= ... wk.
*/
int duval(string s) {
    s += s; //remove this to find Lyndon decomposition of s
    int n = s.size();
    int i = 0;
    int ans = 0;
    //while (i < n) { //for Lyndon decomposition
    while (i < n / 2) {
        ans = i;
        int j = i + 1, k = i;
        while (j < n && s[k] <= s[j]) {
            if (s[k] < s[j])
                k = i;
            else
                ++k;
            ++j;
        }
        while (i <= k) {
            //s.substr(i, j - k) -
            //next prime string of Lyndon decomposition
            i += j - k;
        }
    }
    return ans;
}

//actual odd length is (odd[i] * 2 - 1)
//actual even length is (even[i] * 2)
void manacher(const string &s, vi &odd, vi &even) {
    int n = s.size();
    odd.resize(n);
    int c = -1, r = -1;
    forn (i, n) {
        int k = (r <= i ? 0 : min(odd[2 * c - i], r - i));
        while (i + k < n && i - k >= 0 && s[i + k] == s[i - k])
            ++k;
        odd[i] = k;
        if (i + k > r)
            r = i + k, c = i;
    }
    c = -1, r = -1;
    even.resize(n - 1);
    forn (i, n - 1) {
        int k = (r <= i ? 0 : min(even[2 * c - i], r - i));
        while (i + k + 1 < n && i - k >= 0 &&
                s[i + k + 1] == s[i - k])
            ++k;
        even[i] = k;
        if (i + k > r)
            c = i, r = i + k;
    }
}

void test() {
    vector<int> odd, even;
    string s = "aaaabbaaaaa";
    manacher(s, odd, even);
    for (int x: even)
        cerr << x << ' ';
    cerr << '\n';
    for (int x: odd)
        cerr << x << ' ';
    cerr << '\n';
    // 1 2 1 0 5 0 1 2 2 1
    // 1 2 2 1 1 1 1 2 3 2 1
}

int main() {
    cout << duval("ababcabab") << '\n'; // 5
    test();
}
```

27 strings/eertree.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
const int maxn = 5000100;
const int inf = 1e9 + 1e5;

char buf[maxn];
char *s = buf + 1;
int to[maxn][2];
int suff[maxn];
int len[maxn];
int sz;
int last;

const int odd = 1;
const int even = 2;
const int blank = 3;

inline void go(int &u, int pos) {
    while (u != blank && s[pos - len[u] - 1] != s[pos])
        u = suff[u];
}

void add_char(int pos) {
    go(last, pos);
    int u = suff[last];
    go(u, pos);
    int c = s[pos] - 'a';
    if (!to[last][c]) {
        to[last][c] = sz++;
        len[sz - 1] = len[last] + 2;
        assert(to[u][c]);
        suff[sz - 1] = to[u][c];
    }
    last = to[last][c];
}

void init() {
    sz = 4;
    to[blank][0] = to[blank][1] = even;
    len[blank] = suff[blank] = inf;
    len[even] = 0, suff[even] = odd;
    len[odd] = -1, suff[odd] = blank;
    last = 2;
}

void build() {
    init();
    scanf("%s", s);
    for (int i = 0; s[i]; ++i)
        add_char(i);
}
```

28 strings/suffix_array.cpp

```cpp
string s;
int n;
int sa[maxn], new_sa[maxn], cls[maxn], new_cls[maxn],
        cnt[maxn], lcp[maxn];
int n_cls;

void build() {
    n_cls = 256;
    forn(i, n) {
        sa[i] = i;
        cls[i] = s[i];
    }
    for (int d = 0; d < n; d = d ? d*2 : 1) {

        forn(i, n) new_sa[i] = (sa[i] - d + n) % n;
        forn(i, n_cls) cnt[i] = 0;
        forn(i, n) ++cnt[cls[i]];
        forn(i, n_cls) cnt[i+1] += cnt[i];
        for (int i = n-1; i >= 0; --i)
            sa[--cnt[cls[new_sa[i]]]] = new_sa[i];

        n_cls = 0;
        forn(i, n) {
            if (i && (cls[sa[i]] != cls[sa[i-1]] ||
                    cls[(sa[i]+d)%n] != cls[(sa[i-1]+d)%n])) {
                ++n_cls;
            }
            new_cls[sa[i]] = n_cls;
        }
        ++n_cls;
        forn(i, n) cls[i] = new_cls[i];
    }

    // cls is also a inv perm of sa if a string is not cyclic
    // (i.e. a position of i-th lexicographical suffix)
    int val = 0;
    forn(i, n) {
        if (val) --val;
        if (cls[i] == n-1) continue;
        int j = sa[cls[i] + 1];
        while (i+val != n && j+val != n && s[i+val] == s[j+val])
            ++val;
        lcp[cls[i]] = val;
    }
}

int main() {
    cin >> s;
    s += '$';
    n = s.length();
    build();
    forn(i, n) {
        cout << s.substr(sa[i]) << endl;
        cout << lcp[i] << endl;
    }
}
```

29 strings/ukkonen.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);
    }
};
```

30 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);
}

};
```

31 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);
        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);
    }
};
```

32 structures/linkcut.cpp

```
namespace LinkCut {

typedef struct _node {
    _node *l, *r, *p, *pp;
    int size; bool rev;
    _node();

    explicit _node(nullptr_t) {
        l = r = p = pp = this;
        size = rev = 0;
    }

    void push() {
        if (rev) {
            l->rev ^= 1; r->rev ^= 1;
            rev = 0; swap(l,r);
        }
    }

    void update();
}* node;

node None = new _node(nullptr);
node v2n[maxn];

_node::_node(){
    l = r = p = pp = None;
    size = 1; rev = false;
}

void _node::update() {
    size = (this != None) + l->size + r->size;
    l->p = r->p = this;
}

void rotate(node v) {
    assert(v != None && v->p != None);
    assert(!v->rev);
    assert(!v->p->rev);
    node u = v->p;
    if (v == u->l)
        u->l = v->r, v->r = u;
    else
        u->r = v->l, v->l = u;
    swap(u->p,v->p);
    swap(v->pp,u->pp);
    if (v->p != None) {
        assert(v->p->l == u || v->p->r == u);
        if (v->p->r == u)
            v->p->r = v;
        else
            v->p->l = v;
    }
    u->update();
    v->update();
}

void bigRotate(node v) {
    assert(v->p != None);
    v->p->p->push();
    v->p->push();
    v->push();
    if (v->p->p != None) {
        if ((v->p->l == v) ^ (v->p->p->r == v->p))
            rotate(v->p);
        else
            rotate(v);
    }
    rotate(v);
}

inline void splay(node v) {
    while (v->p != None)
        bigRotate(v);
}

inline void splitAfter(node v) {
    v->push();
    splay(v);
    v->r->p = None;
    v->r->pp = v;
    v->r = None;
    v->update();
}

void expose(int x) {
    node v = v2n[x];
    splitAfter(v);
    while (v->pp != None) {
        assert(v->p == None);
        splitAfter(v->pp);
        assert(v->pp->r == None);
        assert(v->pp->p == None);
        assert(!v->pp->rev);
        v->pp->r = v;
        v->pp->update();
        v = v->pp;
        v->r->pp = None;
    }
    assert(v->p == None);
    splay(v2n[x]);
}

inline void makeRoot(int x) {
    expose(x);
    assert(v2n[x]->p == None);
    assert(v2n[x]->pp == None);
    assert(v2n[x]->r == None);
    v2n[x]->rev ^= 1;
}

inline void link(int x, int y) {
    makeRoot(x);
    v2n[x]->pp = v2n[y];
}

inline void cut(int x, int y) {
    expose(x);
    splay(v2n[y]);
    if (v2n[y]->pp != v2n[x]) {
        swap(x,y);
        expose(x);
        splay(v2n[y]);
        assert(v2n[y]->pp == v2n[x]);
    }
    v2n[y]->pp = None;
}

inline int get(int x, int y) {
    if (x == y)
        return 0;
    makeRoot(x);
    expose(y);
    expose(x);
    splay(v2n[y]);
    if (v2n[y]->pp != v2n[x])
        return -1;
    return v2n[y]->size;
}

}
```

33 structures/ordered_set.cpp

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

typedef __gnu_pbds::tree<int, __gnu_pbds::null_type,
        std::less<int>,
        __gnu_pbds::rb_tree_tag,
        __gnu_pbds::tree_order_statistics_node_update> oset;

#include <iostream>

int main() {
    oset X;
    X.insert(1);
    X.insert(2);
    X.insert(4);
    X.insert(8);
    X.insert(16);

    std::cout << *X.find_by_order(1) << std::endl; // 2
    std::cout << *X.find_by_order(2) << std::endl; // 4
    std::cout << *X.find_by_order(4) << std::endl; // 16
    std::cout << std::boolalpha <<
        (end(X)==X.find_by_order(6)) << std::endl; // true

    std::cout << X.order_of_key(-5) << std::endl;  // 0
    std::cout << X.order_of_key(1) << std::endl;   // 0
    std::cout << X.order_of_key(3) << std::endl;   // 2
    std::cout << X.order_of_key(4) << std::endl;   // 2
    std::cout << X.order_of_key(400) << std::endl; // 5
}
```

34 structures/treap.cpp

```
struct node {
    int x, y;
    node *l, *r;
    node(int x) : x(x), y(rand()), l(r=NULL) {}
};

void split(node *t, node *&l, node *&r, int x) {
    if (!t) return (void)(l=r=NULL);
    if (x <= t->x) {
        split(t->l, l, t->l, x), r = t;
    } else {
        split(t->r, t->r, r, x), l = t;
    }
}

node *merge(node *l, node *r) {
    if (!l) return r;
    if (!r) return l;
    if (l->y > r->y) {
        l->r = merge(l->r, r);
        return l;
    } else {
        r->l = merge(l, r->l);
        return r;
    }
}

node *insert(node *t, node *n) {
    node *l, *r;
    split(t, l, r, n->x);
    return merge(l, merge(n, r));
}

node *insert(node *t, int x) {
    return insert(t, new node(x));
}

node *fast_insert(node *t, node *n) {
    if (!t) return n;
    node *root = t;
    while (true) {
        if (n->x < t->x) {
            if (!t->l || t->l->y < n->y) {
                split(t->l, n->l, n->r, n->x), t->l = n;
                break;
            } else {
                t = t->l;
            }
        } else {
            if (!t->r || t->r->y < n->y) {
                split(t->r, n->l, n->r, n->x), t->r = n;
                break;
            } else {
                t = t->r;
            }
        }
    }
    return root;
}

node *fast_insert(node *t, int x) {
    return fast_insert(t, new node(x));
}

int main() {
    node *t = NULL;
    forn(i, 1000000) {
        int x = rand();
        t = fast_insert(t, x);
    }
}
```

35 Сеточка

36 Сеточка