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 geometry/basic3d.cpp

```
struct Plane {
    pt v;
    ld c;

    Plane(pt a, pt b, pt c) {
        v = ((b - a) % (c - a)).norm();
        this->c = a * v;
    }

    ld dist(pt p) {
        return p * v - c;
    }
};

pt projection(pt p, pt a, pt b) {
    pt v = b - a;
    if (ze(v.abs2())) {
        //stub: bad line
        return a;
    }
    return a + v * (((p - a) * v) / (v * v));
}

pair<pt, pt> planesIntersection(Plane a, Plane b) {
    pt dir = a.v % b.v;
    if (ze(dir.abs2())) {
        //stub: parallel planes
        return {pt{1e18, 1e18, 1e18}, pt{1e18, 1e18, 1e18}};
    }
    ld s = a.v * b.v;
    pt v3 = b.v - a.v * s;
    pt h = a.v * a.c + v3 * ((b.c - a.c * s) / (v3 * v3));
    return {h, h + dir};
}

pair<pt, pt> commonPerpendicular(pt a, pt b, pt c, pt d) {
    pt v = (b - a) % (d - c);
    ld S = v.abs();
    if (ze(S)) {
        //stub: parallel lines
        return {pt{1e18, 1e18, 1e18}, pt{1e18, 1e18, 1e18}};
    }
    v = v.norm();
    pt sh = v * (v * c - v * a);
    pt a2 = a + sh;
    ld s1 = ((c - a2) % (d - a2)) * v;
    pt p = a + (b - a) * (s1 / S);
    return {p, p + sh};
}

/*
Absolute error test
testProjection: 1e1 -> -16.3
testProjection: 1e3 -> -14.1
testProjection: 1e4 -> -13.1
testProjection: 1e5 -> -12.3
testProjection: 1e6 -> -11.2
testPlanesIntersection: 1e1 -> -11.5
testPlanesIntersection: 1e3 -> -8.6
testPlanesIntersection: 1e4 -> -8.3
testPlanesIntersection: 1e5 -> -7.4
testPlanesIntersection: 1e6 -> -6.5
testCommonPerpendicular: 1e1 -> -13.5
testCommonPerpendicular: 1e3 -> -11.4
testCommonPerpendicular: 1e4 -> -10.5
testCommonPerpendicular: 1e5 -> -8.7
testCommonPerpendicular: 1e6 -> -8.6
*/
```

7 geometry/chan.cpp

```
mt19937 rr(111);
ld rndEps() {
    return (ld(rr()) / rr.max() - 0.5) * 1e-7;
}

typedef tuple<int, int, int> Face;
const ld infc = 1e100;

int n;
pt p[maxn];

namespace Chan {
pt _p[maxn];

ld turny(int p1, int p2, int p3) {
    return (p[p2].x - p[p1].x) * (p[p3].y - p[p1].y) -
        (p[p3].x - p[p1].x) * (p[p2].y - p[p1].y);
}

//replace y with z
ld turnz(int p1, int p2, int p3) {
    return (p[p2].x - p[p1].x) * (p[p3].z - p[p1].z) -
        (p[p3].x - p[p1].x) * (p[p2].z - p[p1].z);
}

ld gett(int p1, int p2, int p3) {
    if (p1 == -1 || p2 == -1 || p3 == -1)
        return infc;
    ld ty = turny(p1, p2, p3);
    if (ty >= 0)
        return infc;
    else
        return turnz(p1, p2, p3) / ty;
}

void act(int i) {
    if (p[i].onHull) {
        p[p[i].nx].pr = p[i].pr;
        p[p[i].pr].nx = p[i].nx;
    } else {
        p[p[i].nx].pr = p[p[i].pr].nx = i;
    }
    p[i].onHull ^= 1;
}

ld updt(vector<int> &V) {
    if (V.empty())
        return infc;
    int id = V.back();
    if (p[id].onHull)
        return gett(p[id].pr, p[id].nx, id);
    else
        return gett(p[id].pr, id, p[id].nx);
}

//builds lower hull
vector<int> buildHull(int l, int r) {
    if (l + 1 >= r) {
        p[l].pr = p[l].nx = -1;
        p[l].onHull = true;
        return {};
    }
    int mid = (l + r) / 2;
    auto L = buildHull(l, mid);
    auto R = buildHull(mid, r);
    reverse(all(L));
    reverse(all(R));
    int u = mid - 1, v = mid;
    while (true) {
        if (p[u].pr != -1 &&
                (turny(p[u].pr, u, v) <= 0))
            u = p[u].pr;
        else if (p[v].nx != -1 &&
                (turny(u, v, p[v].nx) <= 0))
            v = p[v].nx;
        else
            break;
    }

    ld t[6];
    t[0] = updt(L);
    t[1] = updt(R);
    vector<int> A;
    while (true) {
        t[2] = gett(p[u].pr, v, u);
        t[3] = gett(u, p[u].nx, v);
        t[4] = gett(u, p[v].pr, v);
        t[5] = gett(u, p[v].nx, v);
        ld nt = infc;
        int type = -1;
        forn (i, 6)
            if (t[i] < nt)
                nt = t[i], type = i;
        if (nt >= infc)
            break;

        if (type == 0) {
            act(L.back());
            if (L.back() < u)
                A.push_back(L.back());
            L.pop_back();
            t[0] = updt(L);
        } else if (type == 1) {
            act(R.back());
            if (R.back() > v)
                A.push_back(R.back());
            R.pop_back();
            t[1] = updt(R);
        } else if (type == 2) {
            A.push_back(u);
            u = p[u].pr;
        } else if (type == 3) {
            A.push_back(u = p[u].nx);
        } else if (type == 4) {
            A.push_back(v = p[v].pr);
        } else if (type == 5) {
            A.push_back(v);
            v = p[v].nx;
        }
    }
    assert(L.empty() && R.empty());

    p[u].nx = v, p[v].pr = u;
    for (int i = u + 1; i < v; ++i)
        p[i].onHull = false;
    for (int i = sz(A) - 1; i >= 0; --i) {
        int id = A[i];
        if (id <= u || id >= v) {
            if (u == id)
                u = p[u].pr;
            if (v == id)
                v = p[v].nx;
            act(id);
        } else {
            p[id].pr = u, p[id].nx = v;
            act(id);
            if (id >= mid)
                v = id;
            else
                u = id;
        }
    }
    return A;
}

//faces are oriented ccw if look from the outside
vector<Face> getFaces() {
    forn (i, n) {
        _p[i] = p[i];
        p[i].x += rndEps();
        p[i].y += rndEps();
        p[i].z += rndEps();
        p[i].id = i;
    }
    sort(p, p + n, [](const pt &a, const pt &b) {
                return a.x < b.x;
            });
    vector<Face> faces;
    forn (q, 2) {
        auto movie = buildHull(0, n);
        for (int x: movie) {
            int id = p[x].id;
            int pid = p[p[x].pr].id;
            int nid = p[p[x].nx].id;
            if (!p[x].onHull)
                faces.emplace_back(pid, id, nid);
            else
                faces.emplace_back(pid, nid, id);
            act(x);
        }
        forn (i, n) {
            p[i].y *= -1;
            p[i].z *= -1;
        }
    }
    forn (i, n)
        p[i] = _p[i];
    return faces;
}

} //namespace Chan
```

8 geometry/halfplanes.cpp

```
ld det3x3(line a, line b, line c) {
    return a.c * (b.v % c.v)
         + b.c * (c.v % a.v)
         + c.c * (a.v % b.v);
}

//check: bounding box is included
vector<pt> halfplanesIntersection(vector<line> l) {
    sort(all(l), cmpLine); //the strongest constraint is first
    l.erase(unique(all(l), eqLine), l.end());
    int n = sz(l);
    vi st;
    forn (iter, 2)
        forn (i, n) {
            while (sz(st) > 1) {
                int j = st.back(), k = *next(st.rbegin());
                if (l[k].v % l[i].v <= eps ||
                        det3x3(l[k], l[j], l[i]) <= eps)
                    break;
                st.pop_back();
            }
            st.push_back(i);
        }

    vi pos(n, -1);
    bool ok = false;
    forn (i, sz(st)) {
        int id = st[i];
        if (pos[id] != -1) {
            st = vi(st.begin() + pos[id], st.begin() + i);
            ok = true;
            break;
        } else
            pos[id] = i;
    }
    if (!ok)
        return {};

    vector<pt> res;
    pt M{0, 0};
    int k = sz(st);
    forn (i, k) {
        line l1 = l[st[i]], l2 = l[st[(i + 1) % k]];
        res.push_back(linesIntersection(l1, l2));
        M = M + res.back();
    }
    M = M * (1. / k);
    for (int id: st)
        if (l[id].signedDist(M) < -eps)
            return {};
    return res;
}
```

9 geometry/nd_convex_hull.cpp

```
const int DIM = 4;
typedef array<ll, DIM> pt;
pt operator-(const pt &a, const pt &b) {
    pt res;
    forn (i, DIM)
        res[i] = a[i] - b[i];
    return res;
}
typedef array<pt, DIM-1> Edge;
typedef array<pt, DIM> Face;
vector<Face> faces;

ll det(pt *a) {
    int p[DIM];
    iota(p, p + DIM, 0);
    ll res = 0;
    do {
        ll x = 1;
        forn (i, DIM) {
            forn (j, i)
                if (p[j] > p[i])
                    x *= -1;
            x *= a[i][p[i]];
        }
        res += x;
    } while (next_permutation(p, p + DIM));
    return res;
}

ll V(Face f, pt pivot) {
    pt p[DIM];
    forn (i, DIM)
        p[i] = f[i] - pivot;
    return det(p);
}

void init(vector<pt> p) {
    forn (i, DIM+1) {
        Face a;
        int q = 0;
        forn (j, DIM+1)
            if (j != i)
                a[q++] = p[j];
        ll v = V(a, p[i]);
        assert(v != 0);
        if (v < 0)
            swap(a[0], a[1]);
        faces.push_back(a);
    }
}

void add(pt p) {
    vector<Face> newf, bad;
    for (auto f: faces) {
        if (V(f, p) < 0)
            bad.push_back(f);
        else
            newf.push_back(f);
    }
    if (bad.empty()) {
        return;
    }
    faces = newf;
    vector<pair<Edge, pt>> edges;
    for (auto f: bad) {
        sort(all(f));
        forn (i, DIM) {
            Edge e;
            int q = 0;
            forn (j, DIM)
                if (i != j)
                    e[q++] = f[j];
            edges.emplace_back(e, f[i]);
        }
    }
    sort(all(edges));
    forn (i, sz(edges)) {
        if (i + 1 < sz(edges) &&
                edges[i + 1].first == edges[i].first) {
            ++i;
            continue;
        }
        Face f;
        forn (j, DIM-1)
            f[j] = edges[i].first[j];
        f[DIM-1] = p;
        if (V(f, edges[i].second) < 0)
            swap(f[0], f[1]);
        faces.push_back(f);
    }
}
```

10 geometry/polygon.cpp

```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;
}

//check: p is oriented ccw
bool segmentInsidePolygon(pt a, pt b, pt *p, int n) {
    if (!pointInsidePolygon((a + b) * .5, p, n))
        return false;
    if (ze((a - b).abs()))
        return true;
    forn (i, n) {
        pt c = p[i];
        if (ze((a - c) % (b - c)) &&
                (a - c) * (b - c) < -eps) {
            //point inside interval
            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

```cpp
struct Cmp {
    pt M, v0;

    bool operator()(const pt &a, const pt &b) {
        pt va{v0 * (a - M), v0 % (a - M)};
        pt vb{v0 * (b - M), v0 % (b - M)};
        return cmpAngle(va, vb);
    }
};

struct Hull {
    vector<pt> h;
    int n;

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

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

    //check: n >= 3
    pair<int, int> tangents(pt a) {
        assert(n >= 3);
        pt M = (h[0] + h[1] + h[2]) * (1. / 3);
        if (a == M)
            return {-1, -1};
        Cmp cmp{M, h[0] - M};
        //assert(is_sorted(all(h), cmp));
        int pos = upper_bound(all(h), a, cmp) - h.begin();
        pt L = h[(pos + n - 1) % n], R = h[pos % n];
        if ((R - L) % (a - L) >= -eps)
            return {-1, -1}; //point inside hull
        int pos2 = upper_bound(all(h), M*2-a, cmp) - h.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
struct line {
    pt v;
    ld c; // v * p = c

    //check: p1 != p2
    line(pt p1, pt p2) {
        v = (p2 - p1).rot();
        v = v * (1. / v.abs());
        c = v * p1;
    }

    // Convert from ax + by + c = 0

    //check: a^2+b^2 > 0
    line(ld a, ld b, ld _c): v(pt{a, b}), c(-_c) {
        ld d = v.abs();
        v = v * (1. / d);
        c /= d;
    }

    //check: v.abs() == 1
    ld signedDist(pt p) {
        return v * p - c;
    }
};

//check: a != b
pt lineProjection(pt p, pt a, pt b) {
    pt v = (b - a).rot();
    ld s = (p - a) % (b - a);
    return p + v * (s / v.abs2());
}

ld pointSegmentDist(pt p, pt a, pt b) {
    if ((p - a) * (b - a) <= 0 || ze((b - a).abs()))
        return (p - a).abs();
    if ((p - b) * (a - b) <= 0)
        return (p - b).abs();
    return fabsl((p - a) % (p - b)) / (b - a).abs();
}

pt linesIntersection(line l1, line l2) {
    ld d = l1.v.x * l2.v.y - l1.v.y * l2.v.x;
    if (ze(d)) {
        if (eq(l1.c, l2.c)) {
            //stub: equal lines
        } else {
            //stub: empty intersection
        }
        return pt{1e18, 1e18};
    }
    ld dx = l1.c * l2.v.y - l1.v.y * l2.c;
    ld dy = l1.v.x * l2.c - l1.c * l2.v.x;
    return pt{dx / d, dy / d};
}

pt linesIntersection(pt a, pt b, pt c, pt d) {
    ld s = (b - a) % (d - c);
    if (ze(s)) {
        //stub: parallel or equal lines
        return pt{1e18, 1e18};
    }
    ld s1 = (c - a) % (d - a);
    return a + (b - a) * (s1 / s);
}

bool pointInsideSegment(pt p, pt a, pt b) {
    if (!ze((p - a) % (p - b)))
        return false;
    ld prod = (a - p) * (b - p);
    return ze(prod) || prod < 0;
    if (ze(prod)) {
        //stub: coincides with segment end
        return true;
    }
    return prod < 0;
}

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)) {
            //stub: intersection of parallel segments
            return true;
        }
        return false;
    }
    ld s1, s2;
    forn (iter, 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;
}

vector<pt> lineCircleIntersection(line l, pt a, ld r) {
    ld d = l.signedDist(a);
    pt h = a - l.v * d;
    if (eq(fabsl(d), r))
        return {h};
    else if (fabsl(d) > r)
        return {};
    pt w = l.v.rot() * Sqrt(sqr(r) - sqr(d));
    return {h + w, h - w};
}

vector<pt> circlesIntersction(pt a, ld r1, pt b, ld r2) {
    ld d = (a - b).abs();
    if (ze(d) && eq(r1, r2)) {
        //stub: equal circles
        return {};
    }
    //  intersection is non-empty iff
    //  triangle with sides r1, r2, d exists
    ld per = r1 + r2 + d;
    ld mx = max(max(r1, r2), d);
    int num = 2;
    if (eq(mx * 2, per)) {
        num = 1;
    } else if (mx * 2 > per)
        return {};
    ld part = (sqr(r1) + sqr(d) - sqr(r2)) / ld(2 * d);
    pt h = a + (b - a) * (part / d);
    if (num == 1)
        return {h};
    ld dh = Sqrt(sqr(r1) - sqr(part));
    pt w = ((b - a) * (dh / d)).rot();
    return {h + w, h - w};
}

vector<pt> circleTangents(pt p, pt a, ld r) {
    ld d = (p - a).abs();
    if (eq(r, d))
        return {p};
    else if (r > d)
        return {};
    ld len = Sqrt(sqr(d) - sqr(r));
    vector<pt> res;
    pt vec = (a - p) * (len / sqr(d));
    for (int sgn: {-1, 1})
        res.push_back(p + vec.rotCw(pt{len, r * sgn}));
    return res;
}

vector<line> circlesBitangents(pt a, ld r1, pt b, ld r2) {
    ld d = (a - b).abs();
    if (ze(d) && eq(r1, r2)) {
        //stub: equal circles
        return {};
    }

    vector<line> res;
    for (int s1: {-1, 1})
        for (int s2: {-1, 1}) {
            //  inner tangent iff s1 != s2
            //  treat radii as signed
            ld r = s2 * r2 - s1 * r1;
            if (eq(fabsl(r), d)) {
                //  incident tangents; need only one copy
                if (s1 == 1)
                    continue;
            } else if (fabsl(r) > d)
                continue;
            ld len = Sqrt(sqr(d) - sqr(r));
            line l(a, a + (b - a).rotCw(pt{len, r}));
            l.c -= s1 * r1;
            res.push_back(l);
        }
    return res;
}
```

13 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];
    vector<int> ord;

    //u*2 - false, u*2+1 - true
    void addEdge(int u, int v) { //u or v
        g[u ^ 1].push_back(v);
        g[v ^ 1].push_back(u);
        gr[u].push_back(v ^ 1);
        gr[v].push_back(u ^ 1);
    }

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

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

    bool run() {
        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);
        int cc = 0;
        for (int u: ord)
            if (!used[u])
                dfs2(u, cc++);
        forn (i, n) {
            if (comp[i * 2] == comp[i * 2 + 1])
                return false;
            res[i] = comp[i * 2] < comp[i * 2 + 1];
        }
        return true;
    }

    void clear() {
        ord.clear();
        forn (i, 2 * n) {
            g[i].clear();
            gr[i].clear();
            comp[i] = -1;
        }
    }
};

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

14 graphs/directed_mst.cpp

```cpp
struct Edge {
    int v, to, id, w;
    bool operator<(const Edge& other) const {
        return w < other.w;
    }
};
typedef pair<multiset<Edge>*, int> Set; // real value: x-Set.se
Set merge(Set a, Set b) {
    if (a.fi == NULL) return b;
    if (b.fi->size() > a.fi->size()) swap(a, b);
    for (Edge e: *b.fi) {
        a.fi->insert(Edge{e.v, e.to, e.id, e.w - b.se + a.se});
    }
    return a;
}
Edge take(Set& set) {
    auto e = *set.fi->begin();
    set.fi->erase(set.fi->begin());
    assert(e.w >= set.se);
    e.w -= set.se;
    set.se += e.w;
    return e;
}

const int maxn = 200500; // must be >= n*2

int n;
int p[maxn];
int get(int x) { return x == p[x] ? x : (p[x] = get(p[x])); }

Set out[maxn]; // outgoing edges from v, endpoints swapped
int b[maxn], top[maxn], done[maxn];
int nc;
int root;
vector<int> edges;
vi cycle[maxn];
vi st;
i64 res;
Edge in[maxn];

void restore(Edge e) {
    edges.push_back(e.id);
    int v = e.v;
    int prev = v;
    while (v != -1) {
        done[v] = true;
        if (v >= n) {
            for (int x: cycle[v]) {
                if (x != prev) {
                    top[x] = -1;
                    restore(in[x]);
                }
            }
        }
        prev = v;
        v = top[v];
    }
}

void solve() {
    forn(i, n*2) p[i] = i, top[i] = -1;
    nc = n;
    root = 0;
    done[root] = true;
    forn(start, n) if (!b[start]) {
        st = {start};
        b[start] = 1;
        while (!done[st[0]]) {
            int v = st.back();
            b[v] = 1;
            if (done[v]) {
                assert(st.size() >= 2);
                st.pop_back();
                assert(!done[st.back()]);
                restore(in[st.back()]);
                assert(done[st.back()]);
                continue;
            }
            assert(!out[v].fi->empty());
            auto e = take(out[v]);
            in[v] = e;
            res += e.w;
            int to = get(e.to);
            if (to == v) continue;
            if (b[to] && !done[to]) {
                while (true) {
                    int u = st.back();
                    st.pop_back();
                    top[u] = nc;
                    p[get(u)] = nc;
                    out[nc] = merge(out[nc], out[u]);
```

```
                    cycle[nc].push_back(u);
                    if (u == to) break;
                }
                st.push_back(nc);
                b[nc] = 1;
                ++nc;
            } else {
                st.push_back(to);
            }
        }
    }
    forn(i, n) assert(done[i]);
    assert((int)edges.size() == n-1);
    cout << res << endl;
}

void scan() {
    int m;
    scanf("%d%d", &n, &m);
    forn(i, n) out[i].fi = new multiset<Edge>();
    forn(i, m) {
        int u, v, w;
        scanf("%d%d%d", &u, &v, &w);
        --u, --v;
        out[v].fi->insert(Edge{v, u, i, w});
    }
}
```

15 graphs/edmonds_matching.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;
}
```

16 graphs/euler_cycle.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++});
}
```

17 graphs/kuhn.cpp

```
bool dfs(int v) {
    if (vis[v]) return false;
    vis[v] = true;
    for (int i = 0; i < (int)e[v].size(); i++) {
        if (mt[e[v][i]] == -1) {
            mt[e[v][i]] = v;
            return true;
        }
    }
    for (int i = 0; i < (int)e[v].size(); i++) {
        if (dfs(mt[e[v][i]])) {
            mt[e[v][i]] = v;
            return true;
        }
    }
    return false;
}

...

fill(pair, -1);
for (int run = 1; run; ) {
  run = 0, fill(used, 0);
  forn(i, n)
    if (pair[i] == -1 && dfs(i))
      run = 1;
}
```

18 graphs/min_automaton.cpp

```
vi inc[maxn][A];
int lst[maxn], pos[maxn], part[maxn];
int lp[maxn], rp[maxn], nrp[maxn];
int upd[maxn], used[maxn], inq[maxn];
vector<int> q;
int dtime;
int np; // number of classes
vector<int> toRefine[A];

void doSwap(int x, int y) {
    swap(lst[pos[x]], lst[pos[y]]);
    swap(pos[x], pos[y]);
}

void refine(const vi& a) {
    ++dtime;
    vector<int> updated;
    for (int x: a) {
        if (used[x] == dtime) continue;
        used[x] = dtime;

        int p = part[x];
        if (upd[p] != dtime) {
            upd[p] = dtime;
            nrp[p] = rp[p];
            updated.pb(p);
        }

        doSwap(x, lst[nrp[p]-1]);
        --nrp[p];
    }

    for (int p: updated) {
        if (lp[p] == nrp[p]) continue;
        lp[np] = nrp[p];
        rp[np] = rp[p];
        rp[p] = nrp[p];
        for (int i = lp[np]; i < rp[np]; ++i) {
            part[lst[i]] = np;
        }

        if (inq[p] || rp[np] - lp[np] < rp[p] - lp[p]) {
            inq[np] = 1;
            q.push_back(np);
        } else {
            inq[p] = 1;
            q.push_back(p);
        }

        ++np;
    }
}

void solve() {
    forn(i, n) lst[i] = i;
    sort(lst, lst+n, [](int i, int j) {
        return col[i] < col[j];
    });

    forn(i, n) {
        if (i && col[lst[i]] != col[lst[i-1]]) {
            rp[np] = i;
            lp[++np] = i;
        }
        part[lst[i]] = np;
        pos[lst[i]] = i;
    }
    rp[np++] = n;

    forn(i, np) {
        inq[i] = 1;
        q.push_back(i);
    }

    forn(i, q.size()) {
        int p = q[i];
        inq[p] = false;
        forn(c, A) {
            toRefine[c].clear();
            for (int id = lp[p]; id < rp[p]; ++id) {
                toRefine[c].insert(
                    toRefine[c].end(), all(inc[lst[id]][c]));
            }
        }
        forn(c, A) if (!toRefine[c].empty()) {
            refine(toRefine[c]);
        }
    }

    forn(i, n) printf("%d\n", part[i] + 1);
}
```

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;
    // < 2^32: 2, 7, 61
    // < 3e18: 2, 3, 5, 7, 11, 13, 17, 19, 23
    // < 2^64: 2, 325, 9375, 28178, 450775, 9780504, 1795265022
    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/golden_search_quad_eq.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';
}

vector<ld> sqrRoots(ld a, ld b, ld c) {
    ld d = b * b - 4 * a * c;
    if (ze(d))
        return {-b / (2 * a)};
    if (d < 0)
        return {};
    d = sqrtl(d);
    if (ze(b)) {
        ld x1 = -d / (2 * a);
        ld x2 = d / (2 * a);
        if (x1 > x2)
            swap(x1, x2);
        return {x1, x2};
    }
    ld sgn = b > 0 ? 1 : -1;
    ld x1 = (-b - sgn * d) / (2 * a);
    ld x2 = c / (a * x1);
    if (x1 > x2)
        swap(x1, x2);
    return {x1, x2};
}
```

21 math/numbers.tex

- Simpson and Gauss numerical integration:
  $\int_a^b f(x)\mathrm{dx} = (b-a)/6 \cdot (f(a) + 4(f(a+b)/2) + f(b))$
  $\int_{-1}^{1}$, $x_{1,3} = \pm\sqrt{0.6}$, $x_2 = 0$; $a_{1,3} = 5/9$, $a_2 = 8/9$
- Large primes: $10^{18} + 3, +31, +3111$, $10^9 + 21, +33$
- FFT modules:

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.
  - Интеграл дуги: $y(x) = \sqrt{r^2 - x^2}$, $\int y(x)dx = \frac{1}{2}(xy + r^2 \arctan\frac{x}{y}) + C$
- 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

22 math/quadratic_equation.cpp

```
vector<ld> sqrRoots(ld a, ld b, ld c) {
    ld d = b * b - 4 * a * c;
    if (ze(d))
        return {-b / (2 * a)};
    if (d < 0)
        return {};
    d = sqrtl(d);
    if (ze(b)) {
        ld x1 = -d / (2 * a);
        ld x2 = d / (2 * a);
        if (x1 > x2)
            swap(x1, x2);
        return {x1, x2};
    }
    ld sgn = b > 0 ? 1 : -1;
    ld x1 = (-b - sgn * d) / (2 * a);
    ld x2 = c / (a * x1);
    if (x1 > x2)
        swap(x1, x2);
    return {x1, x2};
}
```

23 math/simplex.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

```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

```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;
}
```

26 strings/duval_manacher.cpp

```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
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/hashes.cpp

```cpp
#define forn(i, n) for (int i = 0; i < (int)(n); i++)
#define sz(a) (int)(a).size()

typedef long long ll;
typedef unsigned long long ull;

struct num {
  static const int MA = 1e9 + 7, MB = 1e9 + 9;

  int a, b;

  num() { }
  num( int x ) : a(x), b(x) { }
  num( int a, int b ) : a(a), b(b) { }

  num operator + ( const num &x ) const { return num((a + x.a) % MA, (b + x.b) % MB); }
  num operator - ( const num &x ) const { return num((a + MA - x.a) % MA, (b + MB - x.b) % MB); }
  num operator * ( int x ) const { return num(((ll)a * x) % MA, ((ll)b * x) % MB); }
  num operator * ( const num &x ) const { return num(((ll)a * x.a) % MA, ((ll)b * x.b) % MB); }
  bool operator == ( const num &x ) const { return a == x.a && b == x.b; }

  explicit operator ll () const { return (ll)a * MB + b + 1; } // > 0
};

template <class hash_t>
struct StrComparator {
  static const int P;
  static vector<hash_t> deg;

  int n;
  const char *s;
  hash_t *h;

  StrComparator( int n, const char *s ) : n(n), s(s) {
    h = new hash_t[n + 1];
    h[0] = 0;
    forn(i, n)
      h[i + 1] = h[i] * P + s[i];
    deg.reserve(n);
    while (sz(deg) <= n)
      deg.push_back(*deg.rbegin() * P);
  }

  hash_t substr( int i, int len ) const { return h[i + len] - h[i] * deg[len]; }

  int lcp( int i, int j ) {
    int L = 0, R = n - max(i, j);
    while (L < R) {
      int M = (L + R + 1) / 2;
      if (substr(i, M) == substr(j, M))
        L = M;
      else
        R = M - 1;
    }
    return L;
  }

  int cmp( int a, int b ) {
    int LEN = n - max(a, b), L = lcp(a, b);
    return L < LEN ? (int)s[a + L] - s[b + L] : b - a;
  }

  bool operator() ( int i, int j ) { return cmp(i, j) < 0; }
};
template <class hash_t> vector <hash_t> StrComparator<hash_t>::deg(1, hash_t(1));
template <class hash_t> const int StrComparator<hash_t>::P = max(239, rand());

//   StrComparator<num> h(n, s);

/**
 * Usage:
 *   StrComparator<num> h(length, s); // int length, char *s
 *   h.substr(0, 3) == h.substr(1, 3); // сравнение на равенство подстрок за O(1)
 *   h.cmp(2, 3); // сравнение на больше-меньше суффиксов за O(logn)
 *
 *   int p[n]; forn(i, n) p[i] = i;
 *   sort(p, p + n, h); // сортировать суффиксы, суф.массив за O(nlog^2n)
 */
```

29 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;
    }
}
```

30 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] - 'a';
    }

    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

```
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, BEST;

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 < BEST) {
        CENTER = u;
        BEST = worst;
    }
}

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, BEST = 1e9;
    findCenter(u);
    assert(CENTER != -1);
    u = CENTER;
    markAll(u);
    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/fenwick.cpp

```
//BEGIN ALGO
struct Fenwick {
    int *t;
    int n;

    Fenwick(int *a, int len): n(len) {
        t = new int[n];
        memset(t, 0, sizeof(int) * n);
        for (int i = 0; i < n; ++i) {
            inc(i, a[i]);
        }
    }

    ~Fenwick() {
        delete[] t;
    }

    void inc(int l, int delta) {
        for (int i = l; i < n; i = (i | (i + 1))) {
            t[i] += delta;
        }
    }

    int sum(int r) {
        int result = 0;
        for (int i = r; i >= 0; i = (i & (i + 1)) - 1) {
            result += t[i];
        }
        return result;
    }

    int sum(int l, int r) {
        return sum(r) - sum(l - 1);
    }
};

//END ALGO
```

33 structures/heavy_light.cpp

```
int n;
vi e[maxn];

namespace HLD {
int p[maxn], s[maxn], h[maxn], root[maxn];
Rmq rmq[maxn];

void dfs1(int v, int anc) {
    s[v] = 1;
    if (anc != -1) e[v].erase(find(all(e[v]), anc));
    for (int to: e[v]) {
        p[to] = v;
        h[to] = h[v] + 1;
        dfs1(to, v);
        s[v] += s[to];
    }
}

void dfs2(int v, int rt) {
    root[v] = rt;
    if (e[v].empty()) {
        rmq[rt] = Rmq(h[v] - h[rt] + 1);
        return;
    }
    int mxv = e[v][0];
    for (int to: e[v]) {
        if (s[to] > s[mxv]) mxv = to;
    }
    for (int to: e[v]) {
        dfs2(to, to == mxv ? rt : to);
    }
}

int get(int u, int v) {
    int res = 0;
    int t;
    while (root[u] != root[v]) {
        if (h[root[u]] > h[root[v]]) {
            t = rmq[root[u]].get(0, h[u] - h[root[u]] + 1);
            u = p[root[u]];
        } else {
            t = rmq[root[v]].get(0, h[v] - h[root[v]] + 1);
            v = p[root[v]];
        }
        res = max(res, t);
    }
    int r = root[u];
    if (h[u] > h[v]) {
        t = rmq[r].get(h[v] - h[r], h[u] - h[r] + 1);
    } else {
        t = rmq[r].get(h[u] - h[r], h[v] - h[r] + 1);
    }
    return max(res, t);
}

void put(int v, int x) {
    rmq[root[v]].put(h[v] - h[root[v]], x);
}

void init() {
    const int ROOT = 0;
    h[0] = 0;
    dfs1(ROOT, -1);
    dfs2(ROOT, ROOT);
}
} // namespace HLD
```

34 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
}
```

35 structures/splay.cpp

```
struct node;
void updson(node* p, node* v, node* was);

struct node {
    int val;
    node *l, *r, *p;
    node() {}
    node(int val) : val(val), l(r=p=NULL) {}

    bool isRoot() const { return !p; }
    bool isRight() const { return p && p->r == this; }
    bool isLeft() const { return p && p->l == this; }
    void setLeft(node* t) {
        if (t) t->p = this;
        l = t;
    }
    void setRight(node *t) {
        if (t) t->p = this;
        r = t;
    }
};

void updson(node *p, node *v, node *was) {
    if (p) {
        if (p->l == was) p->l = v;
        else p->r = v;
    }
    if (v) v->p = p;
}

void rightRotate(node *v) {
    assert(v && v->l);
    node *u = v->l;
    node *p = v->p;
    v->setLeft(u->r);
    u->setRight(v);
    updson(p, u, v);
}

void leftRotate(node *v) {
    assert(v && v->r);
    node *u = v->r;
    node *p = v->p;
    v->setRight(u->l);
    u->setLeft(v);
    updson(p, u, v);
}

void splay(node *v) {
    while (v->p) {
        if (!v->p->p) {
            if (v->isLeft()) rightRotate(v->p);
            else leftRotate(v->p);
        } else if (v->isLeft() && v->p->isLeft()) {
            rightRotate(v->p->p);
            rightRotate(v->p);
        } else if (v->isRight() && v->p->isRight()) {
            leftRotate(v->p->p);
            leftRotate(v->p);
        } else if (v->isLeft()) {
            rightRotate(v->p);
            leftRotate(v->p);
        } else {
            leftRotate(v->p);
            rightRotate(v->p);
        }
    }
    v->p = NULL;
}

node *insert(node *t, node *n) {
    if (!t) return n;
    int x = n->val;
    while (true) {
        if (x < t->val) {
            if (t->l) {
                t = t->l;
            } else {
                t->setLeft(n);
                t = t->l;
                break;
            }
        } else {
            if (t->r) {
                t = t->r;
            } else {
                t->setRight(n);
                t = t->r;
                break;
            }
        }
    }
    splay(t);
    return t;
}

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

36 structures/treap.cpp

```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);
    }
}
```

37 zzz_narfm/graph/dinic.cpp

```cpp
#define next botvinnik
const int maxn = 505, maxm = 20005;
int to[maxm], next[maxm], c[maxm], f[maxm];
int esz = 0;
int head[maxn];

void addEdge(int a, int b, int cap) {
    to[esz] = b; c[esz] = cap; f[esz] = 0; next[esz] = head[a];
    head[a] = esz++;
    to[esz] = a; c[esz] = 0; f[esz] = 0; next[esz] = head[b];
    head[b] = esz++;
}

int n, m;
int source, sink;
int bound = 1;
int ptr[maxn], dist[maxn];
int dfs(int v, int maxf) {
    if (!maxf)
        return 0;
    if (v == sink)
        return maxf;
    int& e = ptr[v];
    for (; e != -1; e = next[e]) {
        if (dist[to[e]] != dist[v] + 1 || c[e] - f[e] < bound)
            continue;
        int flow = dfs(to[e], min(maxf, c[e] - f[e]));
        if (flow) {
            f[e] += flow;
            f[e ^ 1] -= flow;
            return flow;
        }
    }
    return 0;
}
int q[maxn];
bool bfs() {
    memset(dist, -1, sizeof dist);
    dist[source] = 0;
    int ql = 0, qr = 0;
    q[qr++] = source;
    while (ql < qr) {
        int v = q[ql++];
        for (int e = head[v]; e != -1; e = next[e]) {
            if (dist[to[e]] == -1 && c[e] - f[e] >= bound) {
                dist[to[e]] = dist[v] + 1;
                q[qr++] = to[e];
            }
        }
    }
    return dist[sink] != -1;
}
long long maxFlow() {
    long long ret = 0;
    while (bfs()) {
        for (int i = 0; i < n; ++i)
            ptr[i] = head[i];
        while (int add = dfs(source, 1 << 30))
            ret += add;
    }
    return ret;
}
```

38 zzz_narfm/graph/ford-falkerson.cpp

```
struct Edge {
    int to, f, c;
    Edge() {}
    Edge(int to, int f, int c) : to(to), f(f), c(c) {}
};

const int maxn;

vector<Edge> edgelist;
vector<vector<int>> edge;
int p[maxn], used[maxn];
int source, sink;
int bound, q = 0;

int dfs(int v, int w) {
    p[v] = w;
    used[v] = q;
    if (v == sink)
        return 1 << 30;
    for (size_t i = 0; i < edge[v].size(); ++i) {
        int e = edge[v][i];
        int u = edgelist[e].to;
        if (used[u] != q
            && edgelist[e].c - edgelist[e].f >= bound) {
            int d = dfs(u, e);
            if (d)
                return min(d, edgelist[e].c - edgelist[e].f);
        }
    }
    return 0;
}

foreach (a->b)
    edge[a - 1].push_back(edgelist.size());
edgelist.push_back(Edge(b - 1, 0, c));

int ans = 0;
p[source] = -1;
memset(used, -1, sizeof used);
q = -1;
for (bound = 1 << 30; bound > 0;) {
    ++q;
    int flow = dfs(source, -1);
    if (!flow) {
        bound >>= 1;
        continue;
    }
    ans += flow;
    for (int cur = p[sink]; cur != -1;
         cur = p[edgelist[cur ^ 1].to]) {
        edgelist[cur].f += flow;
        edgelist[cur ^ 1].f -= flow;
    }
}

// Another try:

nt to[maxe], cap[maxe], nxt[maxe];
int ecnt = 0;
int head[maxv];

inline void addEdge(int a, int b, int c) {
    nxt[ecnt] = head[a]; to[ecnt] = b; cap[ecnt] = c;
    head[a] = ecnt++;
    nxt[ecnt] = head[b]; to[ecnt] = a; cap[ecnt] = 0;
    head[b] = ecnt++;
}

int source, sink;
int used[maxv];
int qused = 1;

int dfs(int v, int maxf) {
    if (v == sink || !maxf)
        return maxf;
    used[v] = qused;
    for (int e = head[v]; e != -1; e = nxt[e]) {
        int u = to[e], c = min(cap[e], maxf);
        if (used[u] == qused || !c)
            continue;
        int f = dfs(u, c);
        if (f) {
            cap[e] -= f;
            cap[e ^ 1] += f;
            return f;
        }
    }
    return 0;
}

int maxFlow() {
    memset(used, 0, sizeof used);
    int ret = 0, d;
    while (d = dfs(source, 1 << 30)) {
        ret += d;
        ++qused;
    }
    return ret;
}
```

39 zzz_narfm/graph/lca-rmq.cpp

```
pair<int, int> euler[maxn * 2];
int fst[maxn];
int psz = 0;
int lca(int a, int b) {
    if (fst[a] > fst[b])
        swap(a, b);
    pair<int, int> m = getMin(fst[a], fst[b]);
    return m.second;
}

vector<int> edge[maxn];
void dfs(int v, int h) {
    fst[v] = psz;
    euler[psz++] = make_pair(h, v);
    for (size_t i = 0; i < edge[v].size(); ++i) {
        dfs(edge[v][i], h + 1);
        euler[psz++] = make_pair(h, v);
    }
}
```

40 zzz_narfm/graph/lca.cpp

```
// минимум на пути через lca на двоичных подъёмах

inline bool ancestor(int a, int b) {
    return in[b] >= in[a] && in[b] <= out[a];
}

int climb(int to, int v) {
    int ans = 1 << 30;
    for (int i = maxk - 1; v != to;)
        if (ancestor(to, p[i][v])) {
            ans = min(ans, w[i][v]);
            v = p[i][v];
        } else
            --i;
    return ans;
}

int getans(int a, int b) {
    if (ancestor(a, b))
        return climb(a, b); // lca=a
    if (ancestor(b, a))
        return climb(b, a); // lca=b
    int lca = a;
    for (int i = maxk - 1; !ancestor(p[0][lca], b);)
        if (ancestor(p[i][lca], b))
            --i;
        else
            lca = p[i][lca];
    lca = p[0][lca];
    return min(climb(lca, a), climb(lca, b));
}
```

41 zzz_narfm/graph/mincost.cpp

```cpp
const int maxn = 205, maxm = 10005, inf = 1 << 30;
#define next youSuddenlyVomit

int from[maxm], to[maxm], c[maxm], f[maxm], cost[maxm],
    next[maxm], id[maxm];
int head[maxn];
int esz = 0;

void addEdge(int a, int b, int cst, int i) {
    from[esz] = a; to[esz] = b; c[esz] = 1; f[esz] = 0;
    cost[esz] = cst; next[esz] = head[a]; id[esz] = i;
    head[a] = esz++;

    from[esz] = b; to[esz] = a; c[esz] = 0; f[esz] = 0;
    cost[esz] = -cst; next[esz] = head[b]; id[esz] = i;
    head[b] = esz++;
}

int n, m;
int source, sink;

int range[maxn], p[maxn];

vector<int> ansPath[105];
vector<int> path;

int main() {
    freopen("brides.in", "r", stdin);
    freopen("brides.out", "w", stdout);

    int k;
    cin >> n >> m >> k;
    for (int i = 0; i < m; ++i) {
        int a, b, c;
        cin >> a >> b >> c;
        addEdge(a - 1, b - 1, c, i);
        addEdge(b - 1, a - 1, c, i);
    }
    source = 0;
    sink = n - 1;

    for (int brother = 0; brother < k; ++brother) {
        for (int i = 0; i < n; ++i)
            range[i] = inf;
        range[source] = 0;
        bool need = true;
        while (need) {
            need = false;
            for (int e = 0; e < esz; ++e)
                if (f[e] < c[e] && range[from[e]] != inf
                    && range[to[e]]
                        > range[from[e]] + cost[e]) {
                    need = true;
                    range[to[e]] = range[from[e]] + cost[e];
                    p[to[e]] = e;
                }
        }

        // минкост. 2.

        if (range[sink] == inf) {
            cout << -1 << endl;
            return 0;
        }

        int flow = inf;
        for (int u = sink; u != source; u = from[p[u]])
            flow = min(flow, c[p[u]] - f[p[u]]);
        for (int u = sink; u != source; u = from[p[u]]) {
            f[p[u]] += flow;
            f[p[u] ^ 1] -= flow;
        }
    }

    int ansCost = 0;
    for (int brother = 0; brother < k; ++brother) {
        int u = sink;
        while (u != source) {
            for (int e = 0; e < esz; ++e)
                if (to[e] == u && f[e] > 0) {
                    ansCost += cost[e];
                    f[e]--;
                    f[e ^ 1]++;
                    ansPath[brother].push_back(id[e]);
                    u = from[e];
                    break;
                }
        }
        reverse(
            ansPath[brother].begin(), ansPath[brother].end());
    }

    cout.precision(10);
    cout << fixed << double(ansCost) / k << endl;
    for (int i = 0; i < k; ++i) {
        cout << ansPath[i].size() << ' ';
        for (size_t j = 0; j < ansPath[i].size(); ++j)
            cout << ansPath[i][j] + 1 << ' ';
        cout << endl;
    }

    return 0;
}
```

42 zzz_narfm/misc/convex-hull.cpp

```cpp
typedef long long coord;
struct point {
    coord x, y;
    point() {}
    point(coord x, coord y)
        : x(x)
        , y(y) {}
    point(point a, point b)
        : x(b.x - a.x)
        , y(b.y - a.y) {}
};
inline coord operator*(point a, point b) {
    return a.x * b.x + a.y * b.y;
}
inline coord operator%(point a, point b) {
    return a.x * b.y - a.y * b.x;
}
inline coord operator==(point a, point b) {
    // Warning: consider using epsilon!
    return a.x == b.x && a.y == b.y;
}

inline bool as_pair(const point& a, const point& b) {
    // Warning: consider using epsilon!
    return (a.x == b.x ? a.y < b.y : a.x < b.x);
}
struct by_angle {
    by_angle(const point& corner)
        : corner(corner) {}
    inline bool operator()(const point& a, const point& b) {
        point ca(corner, a);
        point cb(corner, b);
        // Warning: consider using epsilon!
        return ca % cb > 0
            || (ca % cb == 0
                    && point(a, corner) * point(a, b) < 0);
    }
    point corner;
};

vector<point> hull(vector<point> p) {
    sort(p.begin(), p.end(), as_pair);
    p.erase(unique(p.begin(), p.end()), p.end());
    sort(p.begin() + 1, p.end(), by_angle(p[0]));

    vector<point> ret;
    int sz = 0;
    for (size_t i = 0; i < p.size(); ++i) {
        // Warning: consider using epsilon!
        while (sz > 1
            && point(ret[sz - 2], ret[sz - 1])
                    % point(ret[sz - 1], p[i])
                <= 0) {
            ret.pop_back();
            --sz;
        }
        ret.push_back(p[i]);
        ++sz;
    }

    return ret;
}
```

43 zzz_narfm/misc/gauss.cpp

```cpp
int gauss(vector<vector<ld>> v, vector<ld>& ret) {
    int n = v.size();
    int m = n;
    vector<int> p(m), dist(m, 0);
    for (int i = 0; i < m; ++i)
        p[i] = i;
    for (int row = 0, col = 0; row < n && col < m; ++col) {
        int sr = row, sc = col;
        for (int i = row; i < n; ++i)
            for (int j = col; j < m; ++j) {
                if (abs(v[i][j]) > abs(v[sr][sc])) {
                    sr = i;
                    sc = j;
                }
            }
        if (abs(v[sr][sc]) < eps)
            break;
        swap(v[row], v[sr]);
        for (int i = 0; i < n; ++i)
            swap(v[i][col], v[i][sc]);
        swap(p[col], p[sc]);
        dist[col] = 1;
        for (int i = 0; i < n; ++i)
            if (i != row) {
                ld c = v[i][col] / v[row][col];
                for (int j = col; j <= m; ++j)
                    v[i][j] -= v[row][j] * c;
            }
        ++row;
    }
    ret.assign(m, 0);
    for (int i = 0; i < m; ++i)
        if (dist[p[i]])
            ret[i] = v[p[i]][m] / v[p[i]][p[i]];
    for (int i = 0; i < m; ++i) {
        ld sum = 0;
        for (int j = 0; j < m; ++j)
            sum += ret[j] * v[i][p[j]];
        if (abs(sum - v[i][m]) > eps)
            return 0;
    }
    for (int i = 0; i < m; ++i)
        if (!dist[i])
            return -1;
    return 1;
}
```

44 zzz_narfm/strings/aho-corasick.cpp

```cpp
const int triesize, alph;
struct node {
    int p, pch;
    int link, term, upterm;
    int next[alph], go[alph];
};
node t[triesize];
int tsz = 0;

int mkNode(int p, int pch) {
    t[tsz].p = p;
    t[tsz].pch = pch;
    t[tsz].link = t[tsz].upterm = -1;
    memset(t[tsz].next, -1, sizeof t[tsz].next);
    memset(t[tsz].go, -1, sizeof t[tsz].go);
    t[tsz].term = 0;
    return tsz++;
}

void addWord(string s) {
    int v = 0;
    for (size_t i = 0; i < s.size(); ++i) {
        int c = s[i] - '0';
        if (t[v].next[c] == -1)
            t[v].next[c] = mkNode(v, c);
        v = t[v].next[c];
    }
    t[v].term = 1;
}

int q[triesize];
void bfs() {
    int ql = 0, qr = 0;
    q[qr++] = 0;
    t[0].link = 0;
    t[0].upterm = 0;
    for (int i = 0; i < alph; ++i)
        t[0].go[i] = max(t[0].next[i], 0);
    while (ql < qr) {
        int v = q[ql++];
        for (int i = 0; i < alph; ++i) {
            int u = t[v].next[i];
            if (u == -1)
                continue;
            t[u].link = (v ? t[t[v].link].go[i] : 0);
            t[u].upterm
                = (t[t[u].link].upterm || t[u].term ? 1 : 0);
            for (int j = 0; j < alph; ++j)
                t[u].go[j]
                    = (t[u].next[j] == -1 ? t[t[u].link].go[j]
                                          : t[u].next[j]);
            q[qr++] = u;
        }
    }
}
```

45 zzz_narfm/strings/prefix_fun.cpp

```cpp
vector<int> pFunc(string s) {
    int n = s.size();
    vector<int> ret(n);
    ret[0] = 0;
    for (int i = 1; i < n; ++i) {
        int t = ret[i - 1];
        while (t && s[t] != s[i])
            t = ret[t - 1];
        if (s[t] == s[i])
            t++;
        ret[i] = t;
    }
    return ret;
}
```

46 zzz_narfm/strings/suffix_array.cpp

```cpp
void buildSuffixArray(int* src, int n, int* p) {
    static int s[maxn], scale[maxn], cnt[maxn], color[maxn],
        start[maxn], pp[maxn], cc[maxn];
    memcpy(s, src, sizeof(int) * n);
    memcpy(scale, src, sizeof(int) * n);
    sort(scale, scale + n);

    int csz = int(unique(scale, scale + n) - scale);
    for (int i = 0; i < n; ++i)
        s[i] = int(
            lower_bound(scale, scale + csz, s[i]) - scale + 1);
    s[n++] = 0;
    csz++;

    memset(cnt, 0, sizeof cnt);
    for (int i = 0; i < n; ++i)
        cnt[s[i]]++;
    start[0] = 0;
    for (int i = 1; i < csz; ++i)
        start[i] = start[i - 1] + cnt[i - 1];
    for (int i = 0; i < n; ++i)
        p[start[s[i]]++] = i;
    color[p[0]] = 0;
    for (int i = 1; i < n; ++i)
        color[p[i]] = color[p[i - 1]]
            + (s[p[i - 1]] == s[p[i]] ? 0 : 1);

    for (int k = 1; k < n; k <<= 1) {
        memset(cnt, 0, sizeof(int) * n);
        for (int i = 0; i < n; ++i)
            cnt[color[i]]++;
        start[0] = 0;
        for (int i = 1; i < n; ++i)
            start[i] = start[i - 1] + cnt[i - 1];

        for (int i = 0; i < n; ++i)
            p[i] = (p[i] - k + n) % n;
        for (int i = 0; i < n; ++i)
            pp[start[color[p[i]]]++] = p[i];
        memcpy(p, pp, sizeof(int) * n);
        cc[p[0]] = 0;
        for (int i = 1; i < n; ++i)
            cc[p[i]] = cc[p[i - 1]]
                + (color[p[i]] == color[p[i - 1]]
                    && color[(p[i] + k) % n]
                        == color[(p[i - 1] + k) % n]
                ? 0
                : 1);
        memcpy(color, cc, sizeof(int) * n);
    }
    for (int i = 0; i + 1 < n; ++i)
        p[i] = p[i + 1];
}

void buildLcp(int* s, int* sa, int n, int* lcp) {
    static int p[maxn];
    for (int i = 0; i < n; ++i)
        p[sa[i]] = i;
    for (int i = 0; i < n; ++i) {
        if (p[i] + 1 == n)
            continue;
        int j = (i ? max(0, lcp[p[i - 1]] - 1) : 0);
        while (sa[p[i]] + j < n && sa[p[i] + 1] + j < n
            && s[sa[p[i]] + j] == s[sa[p[i] + 1] + j])
            ++j;
        lcp[p[i]] = j;
    }
}
```

47 zzz_narfm/strings/z_function.cpp

```cpp
int z[maxn];
void getZ(const string& s) {
    int n = s.size();
    int l, r;
    l = r = 0;
    for (int i = 1; i < n; ++i) {
        z[i] = 0;
        if (i < r)
            z[i] = min(r - i, z[i - l]);
        while (i + z[i] < n && s[i + z[i]] == s[z[i]])
            z[i]++;
        if (i + z[i] > r) {
            l = i;
            r = i + z[i];
        }
    }
    z[0] = n;
}
```

48 zzz_narfm/structures/segtree_assign.cpp

```
struct node {
    long long fill, sum;
    int flag;
};

const int hfsize, treesize = hfsize << 1;

node tree[treesize];

void push(int i) {
    if (i >= hfsize - 1 || !tree[i].flag)
        return;
    int left = i * 2 + 1, right = left + 1;
    tree[left].flag = tree[right].flag = 1;
    tree[left].fill = tree[right].fill = tree[i].fill;
    tree[left].sum = tree[right].sum = tree[i].sum / 2;
    tree[i].flag = 0;
}

void change(
    int i, int l, int r, int tl, int tr, long long val) {
    if (r < tl || l > tr)
        return;
    if (l >= tl && r <= tr) {
        tree[i].flag = 1;
        tree[i].fill = val;
        tree[i].sum = val * (r - l + 1);
        return;
    }
    push(i);
    int m = (l + r) / 2, left = i * 2 + 1, right = left + 1;
    change(left, l, m, tl, tr, val);
    change(right, m + 1, r, tl, tr, val);
    tree[i].sum = tree[left].sum + tree[right].sum;
}
long long get(int i, int l, int r, int tl, int tr) {
    if (r < tl || l > tr)
        return 0;
    if (l >= tl && r <= tr)
        return tree[i].sum;
    push(i);
    int m = (l + r) / 2, left = i * 2 + 1, right = left + 1;
    return get(left, l, m, tl, tr)
        + get(right, m + 1, r, tl, tr);
}
```

49 zzz_narfm/structures/segtree_lazy.cpp

```
#define left morkva
#define right svekolka
long long sum[treesize];
int left[treesize], right[treesize];
int tsz = 0;
void change(int& t, int l, int r, int at, long long value) {
    if (l > at || r < at)
        return;
    if (t == -1)
        t = tsz++;
    if (l == r)
        return void(sum[t] = value);
    int m = (l + r) / 2;
    change(left[t], l, m, at, value);
    change(right[t], m + 1, r, at, value);
    sum[t] = 0;
    if (left[t] != -1)
        sum[t] += sum[left[t]];
    if (right[t] != -1)
        sum[t] += sum[right[t]];
}
long long get(int t, int l, int r, int tl, int tr) {
    if (l > tr || r < tl || t == -1)
        return 0;
    if (l >= tl && r <= tr)
        return sum[t];
    int m = (l + r) / 2;
    return get(left[t], l, m, tl, tr)
        + get(right[t], m + 1, r, tl, tr);
}
```

50 zzz_narfm/structures/segtree_persist.cpp

```
const int hfsize;
struct node {
    int value;
    node *l, *r;
    node()
        : l(0)
        , r(0) {}
    node(int value)
        : value(value)
        , l(0)
        , r(0) {}
    node(int value, node* l, node* r)
        : value(value)
        , l(l)
        , r(r) {}
};
node* setValue(node* v, int l, int r, int at, int value) {
    if (l > at || r < at)
        return v;
    if (l == r)
        return new node(value);
    int m = (l + r) / 2;
    node* left = setValue(v ? v->l : 0, l, m, at, value);
    node* right = setValue(v ? v->r : 0, m + 1, r, at, value);
    return new node(
        (left ? left->value : 0) + (right ? right->value : 0),
        left, right);
}
int getsum(node* v, int l, int r, int tl, int tr) {
    if (l > tr || r < tl || !v)
        return 0;
    if (l >= tl && r <= tr)
        return v->value;
    int m = (l + r) / 2;
    return getsum(v->l, l, m, tl, tr)
        + getsum(v->r, m + 1, r, tl, tr);
}
```

51 zzz_narfm/structures/sparse.cpp

```
struct Sparse {
    static const int logn = 18;
    T st[logn][maxn];
    int log2[maxn];
    Sparse() {}
    Sparse(T* src, int n) { build(src, n); }
    void build(T* src, int n) {
        log2[1] = 0;
        for (int i = 2; i <= n; ++i)
            log2[i] = log2[i - 1]
                + ((2 << log2[i - 1]) < i ? 1 : 0);
        memcpy(st[0], src, sizeof(T) * n);
        for (int i = 1; i < logn; ++i)
            for (int j = 0; j < n; ++j)
                st[i][j]
                    = getmin(j, min(j + (1 << i) - 1, n - 1));
    }
    inline int getmin(int l, int r) {
        int p = log2[r - l + 1];
        return min(st[p][l], st[p][r - (1 << p) + 1]);
    }
};
```