Содержание

1 Strategy.txt 1
2 flows/dinic.cpp 2
3 flows/globalcut.cpp 2
4 flows/hungary.cpp 3
5 flows/mincost.cpp 3
6 flows/push_relabel.cpp 4
7 geometry/chan.cpp 5
8 geometry/convex_hull_trick.cpp 6
9 geometry/halfplanes.cpp 7
10 geometry/nd_convex_hull.cpp 7
11 geometry/planar_faces.cpp 8
12 geometry/polygon.cpp 8
13 geometry/polygon_tangents.cpp 9
14 geometry/primitives.cpp 9
15 geometry/svg.cpp 10
16 graphs/2sat.cpp 11
17 graphs/directed_mst.cpp 11
18 graphs/dominator_tree.cpp 12
19 graphs/edmonds_matching.cpp 13
20 graphs/euler_cycle.cpp 13
21 graphs/min_automaton.cpp 14
22 math/factor.cpp 14
23 math/fft.cpp 15
24 math/fft_inv.cpp 15
25 math/golden_search_quad_eq.cpp 16
26 math/numbers.tex 16
27 math/simplex.cpp 17
28 math/stuff.cpp 18
29 python/libs.py 18
30 strings/automaton.cpp 19
31 strings/duval_manacher.cpp 19
32 strings/eertree.cpp 20
33 strings/suffix_array.cpp 20
34 strings/ukkonen.cpp 21
35 structures/centroids.cpp 22
36 structures/heavy_light.cpp 22
37 structures/linkcut.cpp 23
38 structures/ordered_set.cpp 24
39 structures/treap.cpp 24

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

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

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

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

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

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

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

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

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

void discharge(int u) {
    while (e[u] > 0) {
        int &i = ptr[u];
        if (i == sz(g[u])) {
            i = 0;
            relabel(u);
            if (h[u] >= n)
                break;
            continue;
```

```
        } else {
            int id = g[u][i++];
            int v = edge[id].to;
            if (h[v] + 1 != h[u])
                continue;
            int delta = min(e[u], ll(edge[id].c - edge[id].f));
            push(id, delta);
        }
    }
}

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

} //PushRelabel
```

7 geometry/chan.cpp

```
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)
```

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

8 geometry/convex_hull_trick.cpp

```
1 struct Hull {
2     vector<pt> top, bot;
3
4     //check: add points in strictly increasing order
5     void append(pt p) {
6         while (sz(bot) > 1 && (p - bot.back()) %
7                     (p - *next(bot.rbegin())) >= -eps)
8             bot.pop_back();
9         bot.push_back(p);
10        while (sz(top) > 1 && (p - top.back()) %
11                    (p - *next(top.rbegin())) <= eps)
12            top.pop_back();
13        top.push_back(p);
14    }
15
16    pt mostDistant(pt dir) {
17        dir = dir.rot();
18        auto &v = dir.x < 0 ? top : bot;
19        int l = -1, r = sz(v) - 1;
20        while (l + 1 < r) {
21            int c = (l + r) / 2;
22            if (dir % (v[c + 1] - v[c]) > 0)
23                r = c;
24            else
25                l = c;
26        }
27        return v[r];
28    }
29 };
```

9 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;
}
```

10 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);
    }
}
```

11 geometry/planar_faces.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);
        }
    }
}
```

12 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;
}

//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;
}
```

13 geometry/polygon_tangents.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)};
    }
};
```

14 geometry/primitives.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;
}
```

15 geometry/svg.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;
```

16 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 - true, u*2+1 - false
    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;
            if (comp[i * 2] < comp[i * 2 + 1])
                res[i] = true;
            else
                res[i] = false;
        }
        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
}
```

17 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});
    }
}
```

18 graphs/dominator_tree.cpp

```
struct Dom {
    int n;
    vector<vi> e, re; // graph (on v), reverse graph (on id)
    vi id, p, sdom, dom, dsu, best;
    vector<vi> bucket;
    int dtime = 0;

    Dom() {}
    Dom(int n) : n(n), e(n), re(n), id(n, -1), p(n),
        sdom(n), dom(n), dsu(n), best(n), bucket(n)
    {  }

    void find(int v) {
        if (v != dsu[v]) {
            find(dsu[v]);
            if (sdom[best[dsu[v]]] <= sdom[best[v]]) {
                best[v] = best[dsu[v]];
            }
            dsu[v] = dsu[dsu[v]];
        }
    }

    void dfs1(int v) {
        id[v] = dtime++;
        for (int to: e[v]) {
            if (id[to] == -1) {
                dfs1(to);
                p[id[to]] = id[v];
            }
            re[id[to]].push_back(id[v]);
        }
    }

    void pre() {
        dfs1(0);
        iota(all(best), 0);
        iota(all(sdom), 0);
        iota(all(dsu), 0);
    }

    void run() {
        pre();
        for (int v = n-1; v >= 0; --v) {
            for (int w: bucket[v]) {
                find(w);
                dom[w] = best[w];
            }
            for (int u: re[v]) {
                find(u);
                sdom[v] = min(sdom[v], sdom[best[u]]);
            }
            if (v) {
                bucket[sdom[v]].pb(v);
                dsu[v] = p[v]; // unite(v, p[v])
            }
        }

        for (int v = 1; v < n; ++v) {
            if (dom[v] != sdom[v]) {
                dom[v] = dom[dom[v]];
            }
        }

        vi ndom(n), rev(n);
        forn(i, n) rev[id[i]] = i;
        forn(i, n) ndom[i] = rev[dom[id[i]]];
        dom = ndom;
    }
};
```

19 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;
}
```

20 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++});
}
```

21 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);
}
```

22 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;
    // < 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;
        }
    }
}
```

23 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
}
```

24 math/fft_inv.cpp

```
const int M = 1 << LG;
//check: a[0] not zero, lg < LG
//check: to is of length (1 << (lg + 1))
base c[M], d[M], e[M];
void fft_inv(base *a, base *to, int lg) {
    base r0 = base(1) / a[0];
    for (int i = 0; i < (1 << lg); ++i)
        a[i] *= r0;
    fill(to, to + (1 << lg), 0);
    to[0] = 1;

    for (int i = 1; i <= lg; ++i) {
        int n = 1 << i;
        int n2 = 1 << (i + 1);
        int hn = 1 << (i - 1);

        fill(c, c + n2, 0);
        fill(d, d + n2, 0);
        fill(e, e + n2, 0);

        copy(a, a + n, c);
        fft(c, i + 1, false);

        copy(to, to + hn, d);
        fft(d, i + 1, false);

        for (int i = 0; i < n2; ++i)
            e[i] = c[i] * d[i];
        fft(e, i + 1, true);

        //cerr << "i = " << i << endl;
        //assert(abs(e[0] - base(1)) < 1e-9);
        //for (int i = 1; i < hn; ++i)
            //assert(abs(e[i]) < 1e-9);

        for (int i = 0; i < hn; ++i) {
            e[i] = -e[i + hn];
            e[i + hn] = 0;
        }
        for (int i = n; i < n2; ++i)
            e[i] = 0;

        fft(e, i, false);
        for (int i = 0; i < n; ++i)
            e[i] *= d[2 * i];
        fft(e, i, true);
        for (int i = 0; i < hn; ++i)
            to[i + hn] = e[i];

    }
    for (int i = 0; i < (1 << lg); ++i)
        to[i] *= r0;
}
```

25 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};
}
```

26 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.
  - 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

27 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
```

28 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;
}
```

29 python/libs.py

```python
""" Decimal """
from decimal import Decimal as D, getcontext, FloatOperation
from decimal import ROUND_DOWN
getcontext().prec = 50
getcontext().traps[FloatOperation] = True
#getcontext().rounding = ROUND_05UP

a = D('3.14') # Correct
#a = D(3.14)  # Wrong!
print(a ** D('0.123'))
print(a % D('1.5'))
print(D(2).sqrt())
print(D(1).exp())
print(D('10').ln())
print(D('10').log10())
print(D('3.15').quantize(D('0.1'))) # 3.2
print(D('3.15').quantize(D('0.1'), rounding=ROUND_DOWN)) # 3.1
print()

def pi():
    """Compute Pi to the current precision. """
    getcontext().prec += 2 # extra digits for intermediate steps
    three = D(3)      # substitute "three=3.0" for regular floats
    lasts, t, s, n, na, d, da = 0, three, 3, 1, 0, 0, 24
    while s != lasts:
        lasts = s
        n, na = n+na, na+8
        d, da = d+da, da+32
        t = (t * n) / d
        s += t
    getcontext().prec -= 2
    return +s               # unary plus applies the new precision

def cos(x):
    """Return the cosine of x as measured in radians.
    The Taylor series approximation works best for a small x.
    For larger values, first compute x = x % (2 * pi). """
    getcontext().prec += 2
    i, lasts, s, fact, num, sign = 0, 0, 1, 1, 1, 1
    # sin: i, lasts, s, fact, num, sign = 1, 0, x, 1, x, 1
    while s != lasts:
        lasts = s
        i += 2
        fact *= i * (i-1)
        num *= x * x
        sign *= -1
        s += num / fact * sign
    getcontext().prec -= 2
    return +s

""" Fraction """

from fractions import Fraction as F
from math import pi
print(F(16, -10)) # -8/5
print(F(123)) # 123
print(F('-.125')) # -1/8
print(F(pi).limit_denominator(30)) # 22/7
print(F(1, 2) ** 31) # 1/2147483648
```

30 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;
}

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;
}
```

31 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();
}
```

32 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);
}
```

33 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;
    }
}
```

34 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);
    }
};
```

35 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, 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);
}

};
```

36 structures/heavy_light.cpp

```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
```

37 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;
}

}
```

38 structures/ordered_set.cpp

```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
}
```

39 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);
    }
}
```