

[ExternalLocation=./]Consolas.ttf

Contents

1	Basic	1
1.1	vimrc	1
1.2	Default code	1
1.3	readchar	1
1.4	Black Magic	1
2	Graph	2
2.1	BCC Vertex*	2
2.2	Bridge*	2
2.3	Bipartite Matching*	2
2.4	2SAT (SCC)*	2
2.5	MinimumMeanCycle*	3
2.6	Maximum Clique Dyn*	3
2.7	Maximum Clique*	3
2.8	Minimum Steiner Tree*	3
2.9	Minimum Arborescence*	4
2.10	Vizing's theorem	4
2.11	Minimum Clique Cover*	4
2.12	NumberOfMaximalClique*	5
2.13	Dijkstra*	5
2.14	Kosaraju*	5
2.15	Simple Graph Matching*	5
2.16	Theory	6
3	Data Structure	6
3.1	Leftist Tree	6
3.2	Heavy light Decomposition	6
3.3	Centroid Decomposition*	6
3.4	Smart Pointer*	6
3.5	LiChaoST*	7
3.6	Link cut tree*	7
3.7	KDTree	8
3.8	Segment Tree with tag	8
4	Flow/Matching	8
4.1	Dinic	8
4.2	Kuhn Munkres	8
4.3	MincostMaxflow	9
4.4	Maximum Simple Graph Matching*	9
4.5	Minimum Weight Matching (Clique version)*	9
4.6	SW-mincut	10
4.7	BoundedFlow(Dinic*)	10
4.8	Flow Models	10
5	String	11
5.1	KMP	11
5.2	Z-value	11
5.3	Manacher*	11
5.4	SAIS*	11
5.5	Aho-Corasick Automatan	12
5.6	Smallest Rotation	12
5.7	De Bruijn sequence*	12
5.8	SAM	12
5.9	PalTree	13
5.10	cyclicLCS	13
5.11	Suffix Array	13
5.12	Suffix Array2	14
6	Math	14
6.1	exgcd*	14
6.2	floor and ceil	14
6.3	Miller Rabin*	14
6.4	Big number	14
6.5	Fraction	15
6.6	Simultaneous Equations	15
6.7	Pollard Rho	15
6.8	Simplex Algorithm	15
6.8.1	Construction	16
6.9	Schreier-Sims Algorithm*	16
6.10	chineseRemainder	16
6.11	QuadraticResidue	16
6.12	Discrete Log	17
6.13	PiCount	17
6.14	Primes	17
6.15	Theorem	17
6.15.1	Kirchhoff's Theorem	17
6.15.2	Tutte's Matrix	17
6.15.3	Cayley's Formula	17
6.15.4	Erdős-Gallai theorem	17
6.15.5	Gale-Ryser theorem	17
6.15.6	Fulkerson-Chen-Anstee theorem	17
6.15.7	Pick's theorem	17
6.16	Euclidean Algorithms	18
7	Polynomial	18
7.1	Fast Fourier Transform	18
7.2	Number Theory Transform	18
7.3	Fast Walsh Transform*	18
7.4	Newton's Method	18
8	Geometry	18
8.1	Default Code	18
8.2	Convex hull*	19
8.3	External bisector	19
8.4	Heart	19
8.5	Minimum Enclosing Circle*	19
8.6	Polar Angle Sort*	19
8.7	Intersection of two circles*	19
8.8	Intersection of polygon and circle	19

1 Basic

1.1 vimrc

```

1 set number
1 syntax on
1 set showmode
1 set tabstop=2
1 set shiftwidth=2
2 set expandtab
2 set cursorline
2 set autoindent
2 set breakindent
3 filetype indent on
3 set smartcase
3 inoremap {<CR> {<CR>}<Esc>ko
3 set incsearch
4 set hlsearch
4 set mouse=a
5
5 map <F9> :call CompileRunGcc(<CR>
5 func! CompileRunGcc()
5     exec "w"
5     exec "!g++ -Wall -Wshadow -g -fsanitize=address % -o %<.
6         out"
6 endfunc

```

1.2 Default code

```

7 #include <bits/stdc++.h>
8 using namespace std;
8 typedef long long ll;
8 typedef pair<int, int> pii;
8 typedef pair<ll, ll> pll;
9 #define debug(x) \
9     { cerr << #x << " = " << x << '\n'; }
9 #define X first
9 #define Y second
10 #define pb push_back
10 signed main() {
11     ios::sync_with_stdio(0);
11     cin.tie(0);
11     return 0;
11 }

```

1.3 readchar

```

12 inline char readchar() {
12     static const size_t bufsize = 65536;
12     static char buf[bufsize];
13     static char *p = buf, *end = buf;
13     if (p == end) end = buf + fread_unlocked(buf, 1, bufsize,
14         stdin), p = buf;
14     return *p++;
14 }

```

1.4 Black Magic

```

15 #include <bits/stdc++.h>
15
16 #include <ext/pb_ds/assoc_container.hpp> //rb_tree
16 #include <ext/pb_ds/priority_queue.hpp>
16 using namespace __gnu_pbds;
16 using namespace std;
17 #define ll long long
17 typedef __gnu_pbds::priority_queue<int> heap;
17 typedef tree<int, null_type, less<int>, rb_tree_tag,
17     tree_order_statistics_node_update> ordered_set;
17 typedef tree<int, null_type, less_equal<int>, rb_tree_tag,
17     tree_order_statistics_node_update> ordered_multiset;
17 int main() {
17     heap h1, h2;
17     h1.push(1), h1.push(3);
17     h2.push(2), h2.push(4);
17     h1.join(h2);
18     cout << h1.size() << h2.size() << h1.top() << endl; //
18     404
18     tree<ll, null_type, less<ll>, rb_tree_tag,
18         tree_order_statistics_node_update> st;
18     tree<ll, ll, less<ll>, rb_tree_tag,
18         tree_order_statistics_node_update> mp;
19     for (int x : {0, 2, 3, 4}) st.insert(x);
19     cout << *st.find_by_order(2) << st.order_of_key(1) <<
19         endl; // 31
19     // erase: st.erase(s.find_by_order(s.order_of_key(v)));
19 }
19 //__int128_t,__float128_t

```

2 Graph

2.1 BCC Vertex*

```
vector<int> G[N]; // 1-base
vector<int> nG[N], bcc[N];
int low[N], dfn[N], Time;
int bcc_id[N], bcc_cnt; // 1-base
bool is_cut[N]; // whether is av
bool cir[N];
int st[N], top;

void dfs(int u, int pa = -1) {
    int child = 0;
    low[u] = dfn[u] = ++Time;
    st[top++] = u;
    for (int v : G[u])
        if (!dfn[v]) {
            dfs(v, u), ++child;
            low[u] = min(low[u], low[v]);
            if (dfn[u] <= low[v]) {
                is_cut[u] = 1;
                bcc[++bcc_cnt].clear();
                int t;
                do {
                    bcc_id[t = st[--top]] = bcc_cnt;
                    bcc[bcc_cnt].push_back(t);
                } while (t != v);
                bcc_id[u] = bcc_cnt;
                bcc[bcc_cnt].pb(u);
            }
        } else if (dfn[v] < dfn[u] && v != pa)
            low[u] = min(low[u], dfn[v]);
    if (pa == -1 && child < 2) is_cut[u] = 0;
}

void bcc_init(int n) {
    Time = bcc_cnt = top = 0;
    for (int i = 1; i <= n; ++i) G[i].clear(), dfn[i] =
        bcc_id[i] = is_cut[i] = 0;
}

void bcc_solve(int n) {
    for (int i = 1; i <= n; ++i)
        if (!dfn[i]) dfs(i);
    // circle-square tree
    for (int i = 1; i <= n; ++i)
        if (is_cut[i]) bcc_id[i] = ++bcc_cnt, cir[bcc_cnt] = 1;
    for (int i = 1; i <= bcc_cnt && !cir[i]; ++i)
        for (int j : bcc[i])
            if (is_cut[j]) nG[i].pb(bcc_id[j]), nG[bcc_id[j]].pb(i);
}
}
```

2.2 Bridge*

```
int low[N], dfn[N], Time; // 1-base
vector<pii> G[N], edge;
vector<bool> is_bridge;

void init(int n) {
    Time = 0;
    for (int i = 1; i <= n; ++i) G[i].clear(), low[i] = dfn[i]
        ] = 0;
}

void add_edge(int a, int b) {
    G[a].pb(pii(b, SZ(edge))), G[b].pb(pii(a, SZ(edge)));
    edge.pb(pii(a, b));
}

void dfs(int u, int f) {
    dfn[u] = low[u] = ++Time;
    for (auto i : G[u])
        if (!dfn[i.X])
            dfs(i.X, i.Y), low[u] = min(low[u], low[i.X]);
        else if (i.Y != f)
            low[u] = min(low[u], dfn[i.X]);
    if (low[u] == dfn[u] && f != -1) is_bridge[f] = 1;
}

void solve(int n) {
    is_bridge.resize(SZ(edge));
    for (int i = 1; i <= n; ++i)
        if (!dfn[i]) dfs(i, -1);
}
}
```

2.3 Bipartite Matching*

```
// x nodes are numbered 1 to n, y nodes are numbered n+1 to
// n+m
// g[X].push_back[Y] / g[u].push_back(nx + v)
vector<int> g[200007];
int nx, ny, ma[200007], d[200007];
bool bfs() {
    int i, u, v, len;
    queue<int> Q;
    for (i = 1; i <= nx; i++) {
        if (ma[i] == 0) {
            d[i] = 0;
            Q.push(i);
        } else
            d[i] = INF;
    }
    d[0] = INF;
    while (!Q.empty()) {
        u = Q.front();
        Q.pop();
        if (u != 0) {
            len = g[u].size();
            for (i = 0; i < len; i++) {
                v = g[u][i];
                if (d[ma[v]] == INF) {
                    d[ma[v]] = d[u] + 1;
                    Q.push(ma[v]);
                }
            }
        }
    }
    return (d[0] != INF);
}

bool dfs(int u) {
    int i, v, len;
    if (u != 0) {
        len = g[u].size();
        for (i = 0; i < len; i++) {
            v = g[u][i];
            if (d[ma[v]] == d[u] + 1) {
                if (dfs(ma[v])) {
                    ma[v] = u;
                    ma[u] = v;
                    return true;
                }
            }
        }
        d[u] = INF;
        return false;
    }
    return true;
}

int hopcroft_karp() {
    int res = 0, i;
    while (bfs())
        for (i = 1; i <= nx; i++)
            if (ma[i] == 0 && dfs(i)) res++;
    return res;
}
}
```

2.4 2SAT (SCC)*

```
struct SAT { // 0-base
    int low[N], dfn[N], bln[N], n, Time, nScc;
    bool instack[N], istrue[N];
    stack<int> st;
    vector<int> G[N], SCC[N];
    void init(int _n) {
        n = _n; // assert(n * 2 <= N);
        for (int i = 0; i < n + n; ++i) G[i].clear();
    }
    void add_edge(int a, int b) { G[a].pb(b); }
    int rv(int a) {
        if (a > n) return a - n;
        return a + n;
    }
    void add_clause(int a, int b) { add_edge(rv(a), b),
        add_edge(rv(b), a); }
    void dfs(int u) {
        dfn[u] = low[u] = ++Time;
        instack[u] = 1, st.push(u);
        for (int i : G[u])
            if (!dfn[i])
                dfs(i), low[u] = min(low[i], low[u]);
            else if (instack[i] && dfn[i] < dfn[u])

```

```

        low[u] = min(low[u], dfn[i]);
    if (low[u] == dfn[u]) {
        int tmp;
        do {
            tmp = st.top(), st.pop();
            instack[tmp] = 0, bln[tmp] = nScc;
        } while (tmp != u);
        ++nScc;
    }
}
bool solve() {
    Time = nScc = 0;
    for (int i = 0; i < n + n; ++i) SCC[i].clear(), low[i]
        = dfn[i] = bln[i] = 0;
    for (int i = 0; i < n + n; ++i)
        if (!dfn[i]) dfs(i);
    for (int i = 0; i < n + n; ++i) SCC[bln[i]].pb(i);
    for (int i = 0; i < n; ++i) {
        if (bln[i] == bln[i + n]) return false;
        istrue[i] = bln[i] < bln[i + n];
        istrue[i + n] = !istrue[i];
    }
    return true;
}
};

```

2.5 MinimumMeanCycle*

```

ll road[N][N]; // input here
struct MinimumMeanCycle {
    ll dp[N + 5][N], n;
    pll solve() {
        ll a = -1, b = -1, L = n + 1;
        for (int i = 2; i <= L; ++i)
            for (int k = 0; k < n; ++k)
                for (int j = 0; j < n; ++j) dp[i][j] = min(dp[i - 1][k] + road[k][j], dp[i][j]);
        for (int i = 0; i < n; ++i) {
            if (dp[L][i] >= INF) continue;
            ll ta = 0, tb = 1;
            for (int j = 1; j < n; ++j)
                if (dp[j][i] < INF && ta * (L - j) < (dp[L][i] - dp[j][i]) * tb) ta = dp[L][i] - dp[j][i], tb = L - j;
            if (ta == 0) continue;
            if (a == -1 || a * tb > ta * b) a = ta, b = tb;
        }
        if (a != -1) {
            ll g = __gcd(a, b);
            return pll(a / g, b / g);
        }
        return pll(-1LL, -1LL);
    }
    void init(int _n) {
        n = _n;
        for (int i = 0; i < n; ++i)
            for (int j = 0; j < n; ++j) dp[i + 2][j] = INF;
    }
};

```

2.6 Maximum Clique Dyn*

```

const int N = 150;
struct MaxClique { // Maximum Clique
    bitset<N> a[N], cs[N];
    int ans, sol[N], q, cur[N], d[N], n;
    void init(int _n) {
        n = _n;
        for (int i = 0; i < n; ++i) a[i].reset();
    }
    void addEdge(int u, int v) { a[u][v] = a[v][u] = 1; }
    void csort(vector<int> &r, vector<int> &c) {
        int mx = 1, km = max(ans - q + 1, 1), t = 0, m = r.size();
        cs[1].reset(), cs[2].reset();
        for (int i = 0; i < m; ++i) {
            int p = r[i], k = 1;
            while ((cs[k] & a[p]).count()) k++;
            if (k > mx) mx++, cs[mx + 1].reset();
            cs[k][p] = 1;
            if (k < km) r[t++] = p;
        }
        c.resize(m);
        if (t) c[t - 1] = 0;
        for (int k = km; k <= mx; k++)
            for (int p = cs[k]._Find_first(); p < N; p = cs[k]._Find_next(p)) r[t] = p, c[t] = k, t++;
    }
};

```

```

}
void dfs(vector<int> &r, vector<int> &c, int l, bitset<N> mask) {
    while (!r.empty()) {
        int p = r.back();
        r.pop_back(), mask[p] = 0;
        if (q + c.back() <= ans) return;
        cur[q++] = p;
        vector<int> nr, nc;
        bitset<N> nmask = mask & a[p];
        for (int i : r)
            if (a[p][i]) nr.push_back(i);
        if (!nr.empty()) {
            if (l < 4) {
                for (int i : nr) d[i] = (a[i] & nmask).count();
                sort(nr.begin(), nr.end(), [&](int x, int y) {
                    return d[x] > d[y]; });
            }
            csort(nr, nc), dfs(nr, nc, l + 1, nmask);
        } else if (q > ans)
            ans = q, copy_n(cur, q, sol);
        c.pop_back(), q--;
    }
}
int solve(bitset<N> mask = bitset<N>(string(N, '1')) {
    // vertex mask
    vector<int> r, c;
    ans = q = 0;
    for (int i = 0; i < n; ++i)
        if (mask[i]) r.push_back(i);
    for (int i = 0; i < n; ++i) d[i] = (a[i] & mask).count();
    sort(r.begin(), r.end(), [&](int i, int j) { return d[i] > d[j]; });
    csort(r, c), dfs(r, c, 1, mask);
    return ans; // sol[0 ~ ans-1]
}
} graph;

```

2.7 Maximum Clique*

```

struct Maximum_Clique {
    typedef bitset<MAXN> bst;
    bst N[MAXN], empty;
    int p[MAXN], n, ans;
    void BronKerbosch2(bst R, bst P, bst X) {
        if (P == empty && X == empty) return ans = max(ans, (int)R.count()), void();
        bst tmp = P | X;
        int u;
        if ((R | P | X).count() <= ans) return;
        for (int uu = 0; uu < n; ++uu) {
            u = p[uu];
            if (tmp[u] == 1) break;
        }
        // if (double(clock())/CLOCKS_PER_SEC > .999)
        // return;
        bst now2 = P & ~N[u];
        for (int vv = 0; vv < n; ++vv) {
            int v = p[vv];
            if (now2[v] == 1) {
                R[v] = 1;
                BronKerbosch2(R, P & N[v], X & N[v]);
                R[v] = 0, P[v] = 0, X[v] = 1;
            }
        }
    }
    void init(int _n) {
        n = _n;
        for (int i = 0; i < n; ++i) N[i].reset();
    }
    void add_edge(int u, int v) { N[u][v] = N[v][u] = 1; }
    int solve() { // remember srand
        bst R, P, X;
        ans = 0, P.flip();
        for (int i = 0; i < n; ++i) p[i] = i;
        random_shuffle(p, p + n), BronKerbosch2(R, P, X);
        return ans;
    }
};

```

2.8 Minimum Steiner Tree*

```

// Minimum Steiner Tree
// O(V 3^T + V^2 2^T)
struct SteinerTree { // 0-base

```

```

static const int T = 10, N = 105, INF = 1e9;
int n, dst[N][N], dp[1 << T][N], tdst[N];
int vcost[N]; // the cost of vertexs
void init(int _n) {
    n = _n;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) dst[i][j] = INF;
        dst[i][i] = vcost[i] = 0;
    }
}
void add_edge(int ui, int vi, int wi) { dst[ui][vi] = min(
    dst[ui][vi], wi); }
void shortest_path() {
    for (int k = 0; k < n; ++k)
        for (int i = 0; i < n; ++i)
            for (int j = 0; j < n; ++j) dst[i][j] = min(dst[i][
                j], dst[i][k] + dst[k][j]);
}
int solve(const vector<int> &ter) {
    shortest_path();
    int t = SZ(ter);
    for (int i = 0; i < (1 << t); ++i)
        for (int j = 0; j < n; ++j) dp[i][j] = INF;
    for (int i = 0; i < n; ++i) dp[0][i] = vcost[i];
    for (int msk = 1; msk < (1 << t); ++msk) {
        if (!(msk & (msk - 1))) {
            int who = __lg(msk);
            for (int i = 0; i < n; ++i) dp[msk][i] = vcost[ter[
                who]] + dst[ter[who]][i];
        }
        for (int i = 0; i < n; ++i)
            for (int submsk = (msk - 1) & msk; submsk; submsk =
                (submsk - 1) & msk)
                dp[msk][i] = min(dp[msk][i], dp[submsk][i] + dp[
                    msk ^ submsk][i] - vcost[i]);
        for (int i = 0; i < n; ++i) {
            tdst[i] = INF;
            for (int j = 0; j < n; ++j) tdst[i] = min(tdst[i],
                dp[msk][j] + dst[j][i]);
        }
        for (int i = 0; i < n; ++i) dp[msk][i] = tdst[i];
    }
    int ans = INF;
    for (int i = 0; i < n; ++i) ans = min(ans, dp[(1 << t)
        - 1][i]);
    return ans;
}
};

```

2.9 Minimum Arborescence*

```

struct zhu_liu { // O(VE)
    struct edge {
        int u, v;
        ll w;
    };
    vector<edge> E; // 0-base
    int pe[N], id[N], vis[N];
    ll in[N];
    void init() { E.clear(); }
    void add_edge(int u, int v, ll w) {
        if (u != v) E.pb(edge{u, v, w});
    }
    ll build(int root, int n) {
        ll ans = 0;
        for (;;) {
            fill_n(in, n, INF);
            for (int i = 0; i < SZ(E); ++i)
                if (E[i].u != E[i].v && E[i].w < in[E[i].v]) pe[E[i]
                    ].v = i, in[E[i].v] = E[i].w;
            for (int u = 0; u < n; ++u) // no solution
                if (u != root && in[u] == INF) return -INF;
            int cntnode = 0;
            fill_n(id, n, -1), fill_n(vis, n, -1);
            for (int u = 0; u < n; ++u) {
                if (u != root) ans += in[u];
                int v = u;
                while (vis[v] != u && !~id[v] && v != root) vis[v]
                    = u, v = E[pe[v]].u;
                if (v != root && !~id[v]) {
                    for (int x = E[pe[v]].u; x != v; x = E[pe[x]].u)
                        id[x] = cntnode;
                    id[v] = cntnode++;
                }
            }
            if (!cntnode) break; // no cycle
        }
    }
};

```

```

for (int u = 0; u < n; ++u)
    if (!~id[u]) id[u] = cntnode++;
for (int i = 0; i < SZ(E); ++i) {
    int v = E[i].v;
    E[i].u = id[E[i].u], E[i].v = id[E[i].v];
    if (E[i].u != E[i].v) E[i].w -= in[v];
}
n = cntnode, root = id[root];
}
return ans;
}
};

```

2.10 Vizing's theorem

```

namespace vizing { // returns edge coloring in adjacent
    // matrix G. 1 - based
    int C[kN][kN], G[kN][kN];
    void clear(int N) {
        for (int i = 0; i <= N; i++) {
            for (int j = 0; j <= N; j++) C[i][j] = G[i][j] = 0;
        }
    }
    void solve(vector<pair<int, int>> &E, int N, int M) {
        int X[kN] = {}, a;
        auto update = [&](int u) {
            for (X[u] = 1; C[u][X[u]]; X[u]++)
                ;
        };
        auto color = [&](int u, int v, int c) {
            int p = G[u][v];
            G[u][v] = G[v][u] = c;
            C[u][c] = v, C[v][c] = u;
            C[u][p] = C[v][p] = 0;
            if (p)
                X[u] = X[v] = p;
            else
                update(u), update(v);
            return p;
        };
        auto flip = [&](int u, int c1, int c2) {
            int p = C[u][c1];
            swap(C[u][c1], C[u][c2]);
            if (p) G[u][p] = G[p][u] = c2;
            if (!C[u][c1]) X[u] = c1;
            if (!C[u][c2]) X[u] = c2;
            return p;
        };
        for (int i = 1; i <= N; i++) X[i] = 1;
        for (int t = 0; t < E.size(); t++) {
            int u = E[t].first, v0 = E[t].second, v = v0, c0 = X[u]
                ], c = c0, d;
            vector<pair<int, int>> L;
            int vst[kN] = {};
            while (!G[u][v0]) {
                L.emplace_back(v, d = X[v]);
                if (!C[v][c])
                    for (a = (int)L.size() - 1; a >= 0; a--) c = color(
                        u, L[a].first, c);
                else if (!C[u][d])
                    for (a = (int)L.size() - 1; a >= 0; a--) color(u, L
                        [a].first, L[a].second);
                else if (vst[d])
                    break;
                else
                    vst[d] = 1, v = C[u][d];
            }
            if (!G[u][v0]) {
                for (; v; v = flip(v, c, d), swap(c, d))
                    ;
                if (C[u][c0]) {
                    for (a = (int)L.size() - 2; a >= 0 && L[a].second
                        != c; a--)
                        ;
                    for (; a >= 0; a--) color(u, L[a].first, L[a].
                        second);
                } else
                    t--;
            }
        }
    }
} // namespace vizing

```

2.11 Minimum Clique Cover*

```

struct Clique_Cover { // 0-base, O(n2n)

```

```

int co[1 << N], n, E[N];
int dp[1 << N];
void init(int _n) {
    n = _n, fill_n(dp, 1 << n, 0);
    fill_n(E, n, 0), fill_n(co, 1 << n, 0);
}
void add_edge(int u, int v) { E[u] |= 1 << v, E[v] |= 1 << u; }
int solve() {
    for (int i = 0; i < n; ++i) co[1 << i] = E[i] | (1 << i);
    co[0] = (1 << n) - 1;
    dp[0] = (n & 1) * 2 - 1;
    for (int i = 1; i < (1 << n); ++i) {
        int t = i & -i;
        dp[i] = -dp[i ^ t];
        co[i] = co[i ^ t] & co[t];
    }
    for (int i = 0; i < (1 << n); ++i) co[i] = (co[i] & i) == i;
    fwt(co, 1 << n);
    for (int ans = 1; ans < n; ++ans) {
        int sum = 0;
        for (int i = 0; i < (1 << n); ++i) sum += (dp[i] * co[i]);
        if (sum) return ans;
    }
    return n;
};

```

2.12 NumberofMaximalClique*

```

struct BronKerbosch { // 1-base
    int n, a[N], g[N][N];
    int S, all[N][N], some[N][N], none[N][N];
    void init(int _n) {
        n = _n;
        for (int i = 1; i <= n; ++i)
            for (int j = 1; j <= n; ++j) g[i][j] = 0;
    }
    void add_edge(int u, int v) { g[u][v] = g[v][u] = 1; }
    void dfs(int d, int an, int sn, int nn) {
        if (S > 1000) return; // pruning
        if (sn == 0 && nn == 0) ++S;
        int u = some[d][0];
        for (int i = 0; i < sn; ++i) {
            int v = some[d][i];
            if (g[u][v]) continue;
            int tsu = 0, tnn = 0;
            copy_n(all[d], an, all[d + 1]);
            all[d + 1][an] = v;
            for (int j = 0; j < sn; ++j)
                if (g[v][some[d][j]]) some[d + 1][tsu++] = some[d][j];
            for (int j = 0; j < nn; ++j)
                if (g[v][none[d][j]]) none[d + 1][tnn++] = none[d][j];
            dfs(d + 1, an + 1, tsu, tnn);
            some[d][i] = 0, none[d][nn++] = v;
        }
    }
    int solve() {
        iota(some[0], some[0] + n, 1);
        S = 0, dfs(0, 0, n, 0);
        return S;
    }
};

```

2.13 Dijkstra*

```

// luogu4779
vector<pii> edge[100020];
int dis[100020];
int vis[100020];
void dijkstra(int s) {
    memset(dis, 0x3f, sizeof(dis));
    dis[s] = 0;
    priority_queue<pii, vector<pii>, greater<pii>> pq;
    pq.emplace(0, s);
    while (pq.size()) {
        int now = pq.top().Y;
        pq.pop();
        if (vis[now]) continue;
        vis[now] = 1;
        for (pii e : edge[now]) {

```

```

            if (!vis[e.X] && dis[e.X] > dis[now] + e.Y) {
                dis[e.X] = dis[now] + e.Y;
                pq.emplace(dis[e.X], e.X);
            }
        }
    }
}

```

2.14 Kosaraju*

```

// find Strong Connected Component
vector<pii> edge[100020], redge[100020];
int vis[100020], scc[100020];
void dfs1(int x, vector<int> &stk) {
    vis[x] = 1;
    for (pii i : edge[x])
        if (!vis[i.X]) dfs1(i.X, stk);
    stk.emplace_back(x);
}
void dfs2(int x, int id) {
    scc[x] = id;
    for (pii i : redge[x])
        if (!scc[i.X]) dfs2(i.X, id);
}
void kosaraju() {
    int nscc = 0;
    vector<int> stk;
    for (int i = 1; i <= n; i++)
        if (!vis[i]) dfs1(i, stk);
    while (stk.size()) {
        if (!scc[stk.back()]) dfs2(stk.back(), ++nscc);
        stk.pop_back();
    }
}

```

2.15 Simple Graph Matching*

```

#define FOR(i, a, b) for (int i = a; i <= b; i++)
#define REP(u) for (int i = h[u], v; v = e[i].t, i; i = e[i].n)
const int N = 520, M = 2e5 + 1;
queue<int> q;
int n, m, tot, qwq, ans;
int h[N], lk[N], tag[N], fa[N], pre[N], dfn[N];
struct edge {
    int t, n;
} e[M];
void link(int x, int y) { lk[x] = y, lk[y] = x; }
void add_edge(int x, int y) {
    if (!lk[x] && !lk[y]) link(x, y), ans++;
    e[++tot] = (edge){y, h[x]}, h[x] = tot;
    e[++tot] = (edge){x, h[y]}, h[y] = tot;
}
void rev(int x) {
    if (x) rev(x[pre][lk]), link(x, pre[x]);
}
int find(int x) { return fa[x] == x ? x : fa[x] = find(fa[x]); }
int lca(int x, int y) {
    for (qwq++; x = x[lk][pre], swap(x, y))
        if (dfn[x = find(x)] == qwq)
            return x;
    else if (x)
        dfn[x] = qwq;
}
int shrink(int x, int y, int p) {
    for (; find(x) != p; x = pre[y]) {
        pre[x] = y, y = lk[x], fa[x] = fa[y] = p;
        if (tag[y] == 2) tag[y] = 1, q.push(y);
    }
}
int blossom(int u) {
    FOR(i, 1, n) tag[i] = pre[i] = 0, fa[i] = i;
    tag[u] = 1, q = queue<int>(), q.push(u);
    for (int p; !q.empty(); q.pop()) REP(u = q.front())
        if (tag[v] == 1)
            p = lca(u, v), shrink(u, v, p), shrink(v, u, p);
        else if (!tag[v]) {
            pre[v] = u, tag[v] = 2;
            if (!lk[v])
                return rev(v), 1;
            else
                tag[lk[v]] = 1, q.push(lk[v]);
        }
    return 0;
}

```

```
int main() {
    scanf("%d%d", &n, &m);
    for (int x, y; m--; add_edge(x, y)) scanf("%d%d", &x, &y);
    ;
    FOR(i, 1, n) ans += !lk[i] && blossom(i);
    cout << ans << '\n';
    for (int i = 1; i <= n; i++) cout << i << ' ' << lk[i] << '\n' return 0;
}
```

2.16 Theory

Maximum independent edge set = $|V| - |\text{Minimum edge cover}|$
 Maximum independent set = $|V| - |\text{Minimum vertex cover}|$

3 Data Structure

3.1 Leftist Tree

```
struct node {
    ll v, data, sz, sum;
    node *l, *r;
    node(ll k) : v(0), data(k), sz(1), l(0), r(0), sum(k) {}
};
ll sz(node *p) { return p ? p->sz : 0; }
ll V(node *p) { return p ? p->v : -1; }
ll sum(node *p) { return p ? p->sum : 0; }
node *merge(node *a, node *b) {
    if (!a || !b) return a ? a : b;
    if (a->data < b->data) swap(a, b);
    a->r = merge(a->r, b);
    if (V(a->r) > V(a->l)) swap(a->r, a->l);
    a->v = V(a->r) + 1, a->sz = sz(a->l) + sz(a->r) + 1;
    a->sum = sum(a->l) + sum(a->r) + a->data;
    return a;
}
void pop(node *&o) {
    node *tmp = o;
    o = merge(o->l, o->r);
    delete tmp;
}
```

3.2 Heavy light Decomposition

```
struct Heavy_light_Decomposition { // 1-base
    int n, ulink[10005], deep[10005], mxson[10005], w[10005],
        pa[10005];
    int t, pl[10005], data[10005], dt[10005], bln[10005],
        edge[10005], et;
    vector<pii> G[10005];
    void init(int _n) {
        n = _n, t = 0, et = 1;
        for (int i = 1; i <= n; ++i) G[i].clear(), mxson[i] = 0;
    }
    void add_edge(int a, int b, int w) { G[a].pb(pii(b, et)),
        G[b].pb(pii(a, et)), edge[et++] = w; }
    void dfs(int u, int f, int d) {
        w[u] = 1, pa[u] = f, deep[u] = d++;
        for (auto &i : G[u])
            if (i.X != f) {
                dfs(i.X, u, d), w[u] += w[i.X];
                if (w[mxson[u]] < w[i.X]) mxson[u] = i.X;
            } else
                bln[i.Y] = u, dt[u] = edge[i.Y];
    }
    void cut(int u, int link) {
        data[pl[u] = t++] = dt[u], ulink[u] = link;
        if (!mxson[u]) return;
        cut(mxson[u], link);
        for (auto i : G[u])
            if (i.X != pa[u] && i.X != mxson[u]) cut(i.X, i.X);
    }
    void build() { dfs(1, 1, 1), cut(1, 1), /*build*/; }
    int query(int a, int b) {
        int ta = ulink[a], tb = ulink[b], re = 0;
        while (ta != tb)
            if (deep[ta] < deep[tb])
                /*query*/, tb = ulink[b = pa[tb]];
            else /*query*/
                , ta = ulink[a = pa[ta]];
        if (a == b) return re;
        if (pl[a] > pl[b]) swap(a, b);
        /*query*/
        return re;
    }
};
```

3.3 Centroid Decomposition*

```
struct Cent_Dec { // 1-base
    vector<pll> G[N];
    pll info[N]; // store info. of itself
    pll upinfo[N]; // store info. of climbing up
    int n, pa[N], layer[N], sz[N], done[N];
    ll dis[_lg(N) + 1][N];
    void init(int _n) {
        n = _n, layer[0] = -1;
        fill_n(pa + 1, n, 0), fill_n(done + 1, n, 0);
        for (int i = 1; i <= n; ++i) G[i].clear();
    }
    void add_edge(int a, int b, int w) { G[a].pb(pll(b, w)),
        G[b].pb(pll(a, w)); }
    void get_cent(int u, int f, int &mx, int &c, int num) {
        int mxsz = 0;
        sz[u] = 1;
        for (pll e : G[u])
            if (!done[e.X] && e.X != f) {
                get_cent(e.X, u, mx, c, num);
                sz[u] += sz[e.X], mxsz = max(mxsz, sz[e.X]);
            }
        if (mx > max(mxsz, num - sz[u])) mx = max(mxsz, num - sz[u]), c = u;
    }
    void dfs(int u, int f, ll d, int org) {
        // if required, add self info or climbing info
        dis[layer[org]][u] = d;
        for (pll e : G[u])
            if (!done[e.X] && e.X != f) dfs(e.X, u, d + e.Y, org);
    }
    int cut(int u, int f, int num) {
        int mx = 1e9, c = 0, lc;
        get_cent(u, f, mx, c, num);
        done[c] = 1, pa[c] = f, layer[c] = layer[f] + 1;
        for (pll e : G[c])
            if (!done[e.X]) {
                if (sz[e.X] > sz[c])
                    lc = cut(e.X, c, num - sz[c]);
                else
                    lc = cut(e.X, c, sz[e.X]);
                upinfo[lc] = pll(), dfs(e.X, c, e.Y, c);
            }
        return done[c] = 0, c;
    }
    void build() { cut(1, 0, n); }
    void modify(int u) {
        for (int a = u, ly = layer[a]; a; a = pa[a], --ly) {
            info[a].X += dis[ly][u], ++info[a].Y;
            if (pa[a]) upinfo[a].X += dis[ly - 1][u], ++upinfo[a].Y;
        }
    }
    ll query(int u) {
        ll rt = 0;
        for (int a = u, ly = layer[a]; a; a = pa[a], --ly) {
            rt += info[a].X + info[a].Y * dis[ly][u];
            if (pa[a]) rt -= upinfo[a].X + upinfo[a].Y * dis[ly - 1][u];
        }
        return rt;
    }
};
```

3.4 Smart Pointer*

```
#ifndef REFERENCE_POINTER
#define REFERENCE_POINTER
template <typename T>
struct _RefCounter {
    T data;
    int ref;
    _RefCounter(const T &d = 0) : data(d), ref(0) {}
};
template <typename T>
struct reference_pointer {
    _RefCounter<T> *p;
    T *operator->() { return &p->data; }
    T &operator*() { return p->data; }
    operator _RefCounter<T> *() { return p; }
    reference_pointer &operator=(const reference_pointer &t) {
        if (p && !--p->ref) delete p;
        p = t.p;
    }
};
```



```

    p && ++p->ref;
    return *this;
}
reference_pointer(_RefCounter<T> *t = 0) : p(t) { p && ++
p->ref; }
reference_pointer(const reference_pointer &t) : p(t.p) {
p && ++p->ref; }
~reference_pointer() {
    if (p && !--p->ref) delete p;
}
};
template <typename T>
inline reference_pointer<T> new_reference(const T &nd) {
    return reference_pointer<T>(new _RefCounter<T>(nd));
}
#endif
// note:
reference_pointer<int> a;
a = new_reference(5);
a = new_reference<int>(5);
a = new_reference((int)5);
reference_pointer<int> b = a;

struct P {
    int a, b;
    P(int _a, int _b) : a(_a), b(_b) {}
} p(2, 3);
reference_pointer<P> a;
c = new_reference(P(1, 2));
c = new_reference<P>(P(1, 2));
c = new_reference(p);

```

3.5 LiChaoST*

```

struct LiChao_min {
    struct line {
        LL m, c;
        line(LL _m = 0, LL _c = 0) {
            m = _m;
            c = _c;
        }
        LL eval(LL x) { return m * x + c; }
    };
    struct node {
        node *l, *r;
        line f;
        node(line v) {
            f = v;
            l = r = NULL;
        }
    };
    typedef node *pnode;
    pnode root;
    int sz;
#define mid ((l + r) >> 1)
    void insert(line &v, int l, int r, pnode &nd) {
        if (!nd) {
            nd = new node(v);
            return;
        }
        LL trl = nd->f.eval(l), trr = nd->f.eval(r);
        LL vl = v.eval(l), vr = v.eval(r);
        if (trl <= vl && trr <= vr) return;
        if (trl > vl && trr > vr) {
            nd->f = v;
            return;
        }
        if (trl > vl) swap(nd->f, v);
        if (nd->f.eval(mid) < v.eval(mid))
            insert(v, mid + 1, r, nd->r);
        else
            swap(nd->f, v), insert(v, l, mid, nd->l);
    }
    LL query(int x, int l, int r, pnode &nd) {
        if (!nd) return LLONG_MAX;
        if (l == r) return nd->f.eval(x);
        if (mid >= x) return min(nd->f.eval(x), query(x, l, mid,
            nd->l));
        return min(nd->f.eval(x), query(x, mid + 1, r, nd->r));
    }
    /* -sz <= query_x <= sz */
    void init(int _sz) {
        sz = _sz + 1;
        root = NULL;
    }
    void add_line(LL m, LL c) {

```

```

        line v(m, c);
        insert(v, -sz, sz, root);
    }
    LL query(LL x) { return query(x, -sz, sz, root); }
};

```

3.6 Link cut tree*

```

struct Splay { // xor-sum
    static Splay nil;
    Splay *ch[2], *f;
    int val, sum, rev, size;
    Splay(int _val = 0) : val(_val), sum(_val), rev(0), size
        (1) { f = ch[0] = ch[1] = &nil; }
    bool isr() { return f->ch[0] != this && f->ch[1] != this; }
    int dir() { return f->ch[0] == this ? 0 : 1; }
    void setCh(Splay *c, int d) {
        ch[d] = c;
        if (c != &nil) c->f = this;
        pull();
    }
    void push() {
        if (!rev) return;
        swap(ch[0], ch[1]);
        if (ch[0] != &nil) ch[0]->rev ^= 1;
        if (ch[1] != &nil) ch[1]->rev ^= 1;
        rev = 0;
    }
    void pull() {
        // take care of the nil!
        size = ch[0]->size + ch[1]->size + 1;
        sum = ch[0]->sum ^ ch[1]->sum ^ val;
        if (ch[0] != &nil) ch[0]->f = this;
        if (ch[1] != &nil) ch[1]->f = this;
    }
} Splay::nil;
Splay *nil = &Splay::nil;
void rotate(Splay *x) {
    Splay *p = x->f;
    int d = x->dir();
    if (!p->isr())
        p->f->setCh(x, p->dir());
    else
        x->f = p->f;
    p->setCh(x->ch[!d], d);
    x->setCh(p, !d);
    p->pull(), x->pull();
}
void splay(Splay *x) {
    vector<Splay *> splayVec;
    for (Splay *q = x;; q = q->f) {
        splayVec.pb(q);
        if (q->isr()) break;
    }
    reverse(ALL(splayVec));
    for (auto it : splayVec) it->push();
    while (!x->isr()) {
        if (x->f->isr())
            rotate(x);
        else if (x->dir() == x->f->dir())
            rotate(x->f), rotate(x);
        else
            rotate(x), rotate(x);
    }
}
Splay *access(Splay *x) {
    Splay *q = nil;
    for (; x != nil; x = x->f) splay(x), x->setCh(q, 1), q =
        x;
    return q;
}
void root_path(Splay *x) { access(x), splay(x); }
void chroot(Splay *x) {
    root_path(x), x->rev ^= 1;
    x->push(), x->pull();
}
void split(Splay *x, Splay *y) { chroot(x), root_path(y); }
void link(Splay *x, Splay *y) {
    root_path(x), chroot(y);
    x->setCh(y, 1);
}
void cut(Splay *x, Splay *y) {
    split(x, y);
    if (y->size != 5) return;
    y->push();

```

```

    y->ch[0] = y->ch[0]->f = nil;
}
Splay *get_root(Splay *x) {
    for (root_path(x); x->ch[0] != nil; x = x->ch[0]) x->push
        ();
    splay(x);
    return x;
}
bool conn(Splay *x, Splay *y) { return get_root(x) ==
    get_root(y); }
Splay *lca(Splay *x, Splay *y) {
    access(x), root_path(y);
    if (y->f == nil) return y;
    return y->f;
}
void change(Splay *x, int val) { splay(x), x->val = val, x
    ->pull(); }
int query(Splay *x, Splay *y) {
    split(x, y);
    return y->sum;
}
}

```

3.7 KDTree

```

namespace kdt {
int root, lc[maxn], rc[maxn], xl[maxn], xr[maxn], yl[maxn],
    yr[maxn];
point p[maxn];
int build(int l, int r, int dep = 0) {
    if (l == r) return -1;
    function<bool(const point &, const point &)> f = [dep](
        const point &a, const point &b) {
        if (dep & 1)
            return a.x < b.x;
        else
            return a.y < b.y;
        };
    int m = (l + r) >> 1;
    nth_element(p + l, p + m, p + r, f);
    xl[m] = xr[m] = p[m].x;
    yl[m] = yr[m] = p[m].y;
    lc[m] = build(l, m, dep + 1);
    if (~lc[m]) {
        xl[m] = min(xl[m], xl[lc[m]]);
        xr[m] = max(xr[m], xr[lc[m]]);
        yl[m] = min(yl[m], yl[lc[m]]);
        yr[m] = max(yr[m], yr[lc[m]]);
    }
    rc[m] = build(m + 1, r, dep + 1);
    if (~rc[m]) {
        xl[m] = min(xl[m], xl[rc[m]]);
        xr[m] = max(xr[m], xr[rc[m]]);
        yl[m] = min(yl[m], yl[rc[m]]);
        yr[m] = max(yr[m], yr[rc[m]]);
    }
    return m;
}
bool bound(const point &q, int o, long long d) {
    double ds = sqrt(d + 1.0);
    if (q.x < xl[o] - ds || q.x > xr[o] + ds || q.y < yl[o] -
        ds || q.y > yr[o] + ds) return false;
    return true;
}
long long dist(const point &a, const point &b) { return (a.
    x - b.x) * 111 * (a.x - b.x) + (a.y - b.y) * 111 * (a.y
    - b.y); }
void dfs(const point &q, long long &d, int o, int dep = 0)
    {
    if (!bound(q, o, d)) return;
    long long cd = dist(p[o], q);
    if (cd != 0) d = min(d, cd);
    if ((dep & 1) && q.x < p[o].x || !(dep & 1) && q.y < p[o]
        .y) {
        if (~lc[o]) dfs(q, d, lc[o], dep + 1);
        if (~rc[o]) dfs(q, d, rc[o], dep + 1);
    } else {
        if (~rc[o]) dfs(q, d, rc[o], dep + 1);
        if (~lc[o]) dfs(q, d, lc[o], dep + 1);
    }
}
}
void init(const vector<point> &v) {
    for (int i = 0; i < v.size(); ++i) p[i] = v[i];
    root = build(0, v.size());
}
long long nearest(const point &q) {
    long long res = 1e18;

```

```

    dfs(q, res, root);
    return res;
}
} // namespace kdt

```

3.8 Segment Tree with tag

```

void push(int id, int l, int r) {
    int mid = (l + r) >> 1;
    seg[id * 2] += tag[id] * (mid - l + 1);
    seg[id * 2 + 1] += tag[id] * (r - mid);
    tag[id * 2] += tag[id];
    tag[id * 2 + 1] += tag[id];
    tag[id] = 0;
}
void modify(int id, int l, int r, int ql, int qr, int val)
    {
    if (ql > r || qr < l) return;
    if (ql <= l && r <= qr) {
        seg[id] += val * (r - l + 1);
        tag[id] += val;
        return;
    }
    if (l == r) return;
    push(id, l, r);
    int mid = (l + r) >> 1;
    modify(id * 2, l, mid, ql, qr, val);
    modify(id * 2 + 1, mid + 1, r, ql, qr, val);
    seg[id] = seg[id * 2] + seg[id * 2 + 1];
}
int query(int id, int l, int r, int ql, int qr) {
    if (ql > r || qr < l) return 0;
    if (ql <= l && r <= qr) return seg[id];
    push(id, l, r);
    int mid = (l + r) >> 1;
    return query(id * 2, l, mid, ql, qr) + query(id * 2 + 1,
        mid + 1, r, ql, qr);
}

```

4 Flow/Matching

4.1 Dinic

```

struct MaxFlow { // 0-base
    struct edge {
        int to, cap, flow, rev;
    };
    vector<edge> G[MAXN];
    int s, t, dis[MAXN], cur[MAXN], n;
    int dfs(int u, int cap) {
        if (u == t || !cap) return cap;
        for (int &i = cur[u]; i < (int)G[u].size(); ++i) {
            edge &e = G[u][i];
            if (dis[e.to] == dis[u] + 1 && e.flow != e.cap) {
                int df = dfs(e.to, min(e.cap - e.flow, cap));
                if (df) {
                    e.flow += df;
                    G[e.to][e.rev].flow -= df;
                    return df;
                }
            }
        }
        dis[u] = -1;
        return 0;
    }
    bool bfs() {
        FILL(dis, -1);
        queue<int> q;
        q.push(s), dis[s] = 0;
        while (!q.empty()) {
            int tmp = q.front();
            q.pop();
            for (auto &u : G[tmp])
                if (!dis[u.to] && u.flow != u.cap) {
                    q.push(u.to);
                    dis[u.to] = dis[tmp] + 1;
                }
        }
        return dis[t] != -1;
    }
}
int maxflow(int _s, int _t) {
    s = _s, t = _t;
    int flow = 0, df;
    while (bfs()) {
        FILL(cur, 0);
        while (df = dfs(s, INF)) flow += df;
    }
}

```



```

    }
    return flow;
}
void init(int _n) {
    n = _n;
    for (int i = 0; i < n; ++i) G[i].clear();
}
void reset() {
    for (int i = 0; i < n; ++i)
        for (auto &j : G[i]) j.flow = 0;
}
void add_edge(int u, int v, int cap) {
    G[u].pb(edge{v, cap, 0, (int)G[v].size()});
    G[v].pb(edge{u, 0, 0, (int)G[u].size() - 1});
}
};

```

4.2 Kuhn Munkres

```

struct KM { // 0-base
    int w[MAXN][MAXN], hl[MAXN], hr[MAXN], slk[MAXN], n;
    int fl[MAXN], fr[MAXN], pre[MAXN], qu[MAXN], ql, qr;
    bool vl[MAXN], vr[MAXN];
    void init(int _n) {
        n = _n;
        for (int i = 0; i < n; ++i)
            for (int j = 0; j < n; ++j) w[i][j] = -INF;
    }
    void add_edge(int a, int b, int wei) { w[a][b] = wei; }
    bool Check(int x) {
        if (vl[x] = 1, ~fl[x]) return vr[qu[qr++]] = fl[x] = 1;
        while (~x) swap(x, fr[fl[x] = pre[x]]);
        return 0;
    }
    void Bfs(int s) {
        fill(slk, slk + n, INF);
        fill(vl, vl + n, 0), fill(vr, vr + n, 0);
        ql = qr = 0, qu[qr++] = s, vr[s] = 1;
        while (1) {
            int d;
            while (ql < qr)
                for (int x = 0, y = qu[ql++]; x < n; ++x)
                    if (!vl[x] && slk[x] >= (d = hl[x] + hr[y] - w[x][y]))
                        if (pre[x] = y, d)
                            slk[x] = d;
                        else if (!Check(x))
                            return;
            d = INF;
            for (int x = 0; x < n; ++x)
                if (!vl[x] && d > slk[x]) d = slk[x];
            for (int x = 0; x < n; ++x) {
                if (vl[x])
                    hl[x] += d;
                else
                    slk[x] -= d;
                if (vr[x]) hr[x] -= d;
            }
            for (int x = 0; x < n; ++x)
                if (!vl[x] && !slk[x] && !Check(x)) return;
        }
    }
    int Solve() {
        fill(fl, fl + n, -1), fill(fr, fr + n, -1), fill(hr, hr + n, 0);
        for (int i = 0; i < n; ++i) hl[i] = *max_element(w[i], w[i] + n);
        for (int i = 0; i < n; ++i) Bfs(i);
        int res = 0;
        for (int i = 0; i < n; ++i) res += w[i][fl[i]];
        return res;
    }
};

```

4.3 MincostMaxflow

```

struct MCMF { // 0-base
    struct edge {
        ll from, to, cap, flow, cost, rev;
    } *past[MAXN];
    vector<edge> G[MAXN];
    bitset<MAXN> inq;
    ll dis[MAXN], up[MAXN], s, t, mx, n;
    bool BellmanFord(ll &flow, ll &cost) {
        fill(dis, dis + n, INF);
        queue<ll> q;

```

```

        q.push(s), inq.reset(), inq[s] = 1;
        up[s] = mx - flow, past[s] = 0, dis[s] = 0;
        while (!q.empty()) {
            ll u = q.front();
            q.pop(), inq[u] = 0;
            if (!up[u]) continue;
            for (auto &e : G[u])
                if (e.flow != e.cap && dis[e.to] > dis[u] + e.cost)
                    {
                        dis[e.to] = dis[u] + e.cost, past[e.to] = &e;
                        up[e.to] = min(up[u], e.cap - e.flow);
                        if (!inq[e.to]) inq[e.to] = 1, q.push(e.to);
                    }
        }
        if (dis[t] == INF) return 0;
        flow += up[t], cost += up[t] * dis[t];
        for (ll i = t; past[i]; i = past[i]->from) {
            auto &e = *past[i];
            e.flow += up[t], G[e.to][e.rev].flow -= up[t];
        }
        return 1;
    }
    ll MinCostMaxFlow(ll _s, ll _t, ll &cost) {
        s = _s, t = _t, cost = 0;
        ll flow = 0;
        while (BellmanFord(flow, cost))
            ;
        return flow;
    }
    void init(ll _n, ll _mx) {
        n = _n, mx = _mx;
        for (int i = 0; i < n; ++i) G[i].clear();
    }
    void add_edge(ll a, ll b, ll cap, ll cost) {
        G[a].pb(edge{a, b, cap, 0, cost, G[b].size()});
        G[b].pb(edge{b, a, 0, 0, -cost, G[a].size() - 1});
    }
};

```

4.4 Maximum Simple Graph Matching*

```

struct GenMatch { // 1-base
    int V, pr[N];
    bool el[N][N], inq[N], inp[N], inb[N];
    int st, ed, nb, bk[N], djs[N], ans;
    void init(int _V) {
        V = _V;
        for (int i = 0; i <= V; ++i) {
            for (int j = 0; j <= V; ++j) el[i][j] = 0;
            pr[i] = bk[i] = djs[i] = 0;
            inq[i] = inp[i] = inb[i] = 0;
        }
    }
    void add_edge(int u, int v) { el[u][v] = el[v][u] = 1; }
    int lca(int u, int v) {
        fill_n(inp, V + 1, 0);
        while (1)
            if (u = djs[u], inp[u] = true, u == st)
                break;
            else
                u = bk[pr[u]];
        while (1)
            if (v = djs[v], inp[v])
                return v;
            else
                v = bk[pr[v]];
        return v;
    }
    void upd(int u) {
        for (int v; djs[u] != nb; ) {
            v = pr[u], inb[djs[u]] = inb[djs[v]] = true;
            u = bk[v];
            if (djs[u] != nb) bk[u] = v;
        }
    }
    void blo(int u, int v, queue<int> &qe) {
        nb = lca(u, v), fill_n(inb, V + 1, 0);
        upd(u), upd(v);
        if (djs[u] != nb) bk[u] = v;
        if (djs[v] != nb) bk[v] = u;
        for (int tu = 1; tu <= V; ++tu)
            if (inb[djs[tu]])
                if (djs[tu] = nb, !inq[tu]) qe.push(tu), inq[tu] = 1;
    }
    void flow() {

```

```

fill_n(inq + 1, V, 0), fill_n(bk + 1, V, 0);
iota(djs + 1, djs + V + 1, 1);
queue<int> qe;
qe.push(st), inq[st] = 1, ed = 0;
while (!qe.empty()) {
    int u = qe.front();
    qe.pop();
    for (int v = 1; v <= V; ++v)
        if (el[u][v] && djs[u] != djs[v] && pr[u] != v) {
            if ((v == st) || (pr[v] > 0 && bk[pr[v]] > 0))
                blo(u, v, qe);
            else if (!bk[v]) {
                if (bk[v] = u, pr[v] > 0) {
                    if (!inq[pr[v]]) qe.push(pr[v]);
                } else
                    return ed = v, void();
            }
        }
}
}
}
void aug() {
    for (int u = ed, v, w; u > 0; v = bk[u], w = pr[v], pr[v] = u, pr[u] = v, u = w; )
}
int solve() {
    fill_n(pr, V + 1, 0), ans = 0;
    for (int u = 1; u <= V; ++u)
        if (!pr[u])
            if (st = u, flow(), ed > 0) aug(), ++ans;
    return ans;
}
};

```

4.5 Minimum Weight Matching (Clique version)*

```

struct Graph { // 0-base (Perfect Match), n is even
    int n, match[N], onstk[N], stk[N], tp;
    ll edge[N][N], dis[N];
    void init(int _n) {
        n = _n, tp = 0;
        for (int i = 0; i < n; ++i) fill_n(edge[i], n, 0);
    }
    void add_edge(int u, int v, ll w) { edge[u][v] = edge[v][u] = w; }
    bool SPFA(int u) {
        stk[tp++] = u, onstk[u] = 1;
        for (int v = 0; v < n; ++v)
            if (!onstk[v] && match[u] != v) {
                int m = match[v];
                if (dis[m] > dis[u] - edge[v][m] + edge[u][v]) {
                    dis[m] = dis[u] - edge[v][m] + edge[u][v];
                    onstk[v] = 1, stk[tp++] = v;
                    if (onstk[m] || SPFA(m)) return 1;
                    --tp, onstk[v] = 0;
                }
            }
        onstk[u] = 0, --tp;
        return 0;
    }
    ll solve() { // find a match
        for (int i = 0; i < n; ++i) match[i] = i ^ 1;
        while (1) {
            int found = 0;
            fill_n(dis, n, 0);
            fill_n(onstk, n, 0);
            for (int i = 0; i < n; ++i)
                if (tp = 0, !onstk[i] && SPFA(i))
                    for (found = 1; tp >= 2; ) {
                        int u = stk[--tp];
                        int v = stk[--tp];
                        match[u] = v, match[v] = u;
                    }
            if (!found) break;
        }
        ll ret = 0;
        for (int i = 0; i < n; ++i) ret += edge[i][match[i]];
        return ret >> 1;
    }
};

```

4.6 SW-mincut

```

// global min cut
struct SW { // O(V^3)
    static const int MXN = 514;

```

```

    int n, vst[MXN], del[MXN];
    int edge[MXN][MXN], wei[MXN];
    void init(int _n) { n = _n, MEM(edge, 0), MEM(del, 0); }
    void addEdge(int u, int v, int w) { edge[u][v] += w, edge[v][u] += w; }
    void search(int &s, int &t) {
        MEM(vst, 0), MEM(wei, 0), s = t = -1;
        while (1) {
            int mx = -1, cur = 0;
            for (int i = 0; i < n; ++i)
                if (!del[i] && !vst[i] && mx < wei[i]) cur = i, mx = wei[i];
            if (mx == -1) break;
            vst[cur] = 1, s = t, t = cur;
            for (int i = 0; i < n; ++i)
                if (!vst[i] && !del[i]) wei[i] += edge[cur][i];
        }
    }
    int solve() {
        int res = INF;
        for (int i = 0, x, y; i < n - 1; ++i) {
            search(x, y), res = min(res, wei[y]), del[y] = 1;
            for (int j = 0; j < n; ++j) edge[x][j] = (edge[j][x] += edge[y][j]);
        }
        return res;
    }
};

```

4.7 BoundedFlow(Dinic*)

```

struct BoundedFlow { // 0-base
    struct edge {
        int to, cap, flow, rev;
    };
    vector<edge> G[N];
    int n, s, t, dis[N], cur[N], cnt[N];
    void init(int _n) {
        n = _n;
        for (int i = 0; i < n + 2; ++i) G[i].clear(), cnt[i] = 0;
    }
    void add_edge(int u, int v, int lcap, int rcap) {
        cnt[u] -= lcap, cnt[v] += lcap;
        G[u].pb(edge{v, rcap, lcap, SZ(G[v])});
        G[v].pb(edge{u, 0, 0, SZ(G[u]) - 1});
    }
    void add_edge(int u, int v, int cap) {
        G[u].pb(edge{v, cap, 0, SZ(G[v])});
        G[v].pb(edge{u, 0, 0, SZ(G[u]) - 1});
    }
    int dfs(int u, int cap) {
        if (u == t || !cap) return cap;
        for (int &i = cur[u]; i < SZ(G[u]); ++i) {
            edge &e = G[u][i];
            if (dis[e.to] == dis[u] + 1 && e.cap != e.flow) {
                int df = dfs(e.to, min(e.cap - e.flow, cap));
                if (df) {
                    e.flow += df, G[e.to][e.rev].flow -= df;
                    return df;
                }
            }
        }
        dis[u] = -1;
        return 0;
    }
    bool bfs() {
        fill_n(dis, n + 3, -1);
        queue<int> q;
        q.push(s), dis[s] = 0;
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            for (edge &e : G[u])
                if (!dis[e.to] && e.flow != e.cap) q.push(e.to), dis[e.to] = dis[u] + 1;
        }
        return dis[t] != -1;
    }
    int maxflow(int _s, int _t) {
        s = _s, t = _t;
        int flow = 0, df;
        while (bfs()) {
            fill_n(cur, n + 3, 0);
            while ((df = dfs(s, INF))) flow += df;
        }
    }
};

```

```

    return flow;
}
bool solve() {
    int sum = 0;
    for (int i = 0; i < n; ++i)
        if (cnt[i] > 0)
            add_edge(n + 1, i, cnt[i]), sum += cnt[i];
        else if (cnt[i] < 0)
            add_edge(i, n + 2, -cnt[i]);
    if (sum != maxflow(n + 1, n + 2)) sum = -1;
    for (int i = 0; i < n; ++i)
        if (cnt[i] > 0)
            G[n + 1].pop_back(), G[i].pop_back();
        else if (cnt[i] < 0)
            G[i].pop_back(), G[n + 2].pop_back();
    return sum != -1;
}
int solve(int _s, int _t) {
    add_edge(_t, _s, INF);
    if (!solve()) return -1; // invalid flow
    int x = G[_t].back().flow;
    return G[_t].pop_back(), G[_s].pop_back(), x;
}
};

```

4.8 Flow Models

•Maximum/Minimum flow with lower bound / Circulation problem

1. Construct super source S and sink T .
2. For each edge (x, y, l, u) , connect $x \rightarrow y$ with capacity $u - l$.
3. For each vertex v , denote by $in(v)$ the difference between the sum of incoming lower bounds and the sum of outgoing lower bounds.
4. If $in(v) > 0$, connect $S \rightarrow v$ with capacity $in(v)$, otherwise, connect $v \rightarrow T$ with capacity $-in(v)$.
- To maximize, connect $t \rightarrow s$ with capacity ∞ (skip this in circulation problem), and let f be the maximum flow from S to T . If $f \neq \sum_{v \in V, in(v) > 0} in(v)$, there's no solution. Otherwise, the maximum flow from s to t is the answer.
- To minimize, let f be the maximum flow from S to T . Connect $t \rightarrow s$ with capacity ∞ and let the flow from S to T be f' . If $f + f' \neq \sum_{v \in V, in(v) > 0} in(v)$, there's no solution. Otherwise, f' is the answer.
5. The solution of each edge e is $l_e + f_e$, where f_e corresponds to the flow of edge e on the graph.

•Construct minimum vertex cover from maximum matching M on bipartite graph (X, Y)

1. Redirect every edge: $y \rightarrow x$ if $(x, y) \in M$, $x \rightarrow y$ otherwise.
2. DFS from unmatched vertices in X .
3. $x \in X$ is chosen iff x is unvisited.
4. $y \in Y$ is chosen iff y is visited.

•Minimum cost cyclic flow

1. Construct super source S and sink T .
2. For each edge (x, y, c) , connect $x \rightarrow y$ with $(cost, cap) = (c, 1)$ if $c > 0$, otherwise connect $y \rightarrow x$ with $(cost, cap) = (-c, 1)$.
3. For each edge with $c < 0$, sum these cost as K , then increase $d(y)$ by 1, decrease $d(x)$ by 1.
4. For each vertex v with $d(v) > 0$, connect $S \rightarrow v$ with $(cost, cap) = (0, d(v))$.
5. For each vertex v with $d(v) < 0$, connect $v \rightarrow T$ with $(cost, cap) = (0, -d(v))$.
6. Flow from S to T , the answer is the cost of the flow $C + K$.

•Maximum density induced subgraph

1. Binary search on answer, suppose we're checking answer T .
2. Construct a max flow model, let K be the sum of all weights.
3. Connect source $s \rightarrow v$, $v \in G$ with capacity K .
4. For each edge (u, v, w) in G , connect $u \rightarrow v$ and $v \rightarrow u$ with capacity w .
5. For $v \in G$, connect it with sink $v \rightarrow t$ with capacity $K + 2T - (\sum_{e \in E(v)} w(e)) - 2w(v)$.
6. T is a valid answer if the maximum flow $f < K|V|$.

•Minimum weight edge cover

1. For each $v \in V$ create a copy v' , and connect $u' \rightarrow v'$ with weight $w(u, v)$.
2. Connect $v \rightarrow v'$ with weight $2\mu(v)$, where $\mu(v)$ is the cost of the cheapest edge incident to v .
3. Find the minimum weight perfect matching on G' .

•Project selection problem

1. If $p_v > 0$, create edge (s, v) with capacity p_v ; otherwise, create edge (v, t) with capacity $-p_v$.
2. Create edge (u, v) with capacity w with w being the cost of choosing u without choosing v .
3. The mincut is equivalent to the maximum profit of a subset of projects.

•0/1 quadratic programming

$$\sum_x c_x x + \sum_y c_y \bar{y} + \sum_{xy} c_{xy} x \bar{y} + \sum_{xyx'y'} c_{xyx'y'} (x \bar{y} + x' \bar{y}')$$

can be minimized by the mincut of the following graph:

1. Create edge (x, t) with capacity c_x and create edge (s, y) with capacity c_y .
2. Create edge (x, y) with capacity c_{xy} .
3. Create edge (x, y) and edge (x', y') with capacity $c_{xyx'y'}$.

5 String

5.1 KMP

```

int F[MAXN];
vector<int> match(string A, string B) {
    vector<int> ans;
    F[0] = -1, F[1] = 0;
    for (int i = 1, j = 0; i < SZ(B); F[++i] = ++j) {
        if (B[i] == B[j]) F[i] = F[j]; // optimize
        while (j != -1 && B[i] != B[j]) j = F[j];
    }
    for (int i = 0, j = 0; i < SZ(A); ++i) {
        while (j != -1 && A[i] != B[j]) j = F[j];
        if (++j == SZ(B)) ans.pb(i + 1 - j), j = F[j];
    }
    return ans;
}

```

5.2 Z-value

```

const int MAXn = 1e5 + 5;
int z[MAXN];
void make_z(string s) {
    int l = 0, r = 0;
    for (int i = 1; i < s.size(); ++i) {
        for (z[i] = max(0, min(r - i + 1, z[i - l])); i + z[i] < s.size() && s[i + z[i]] == s[z[i]]; z[i]++);
        if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
    }
}

```

5.3 Manacher*

```

int z[MAXN];
int Manacher(string tmp) {
    string s = "&";
    int l = 0, r = 0, x, ans;
    for (char c : tmp) s.pb(c), s.pb('%');
    ans = 0, x = 0;
    for (int i = 1; i < SZ(s); ++i) {
        z[i] = r > i ? min(z[2 * l - i], r - i) : 1;
        while (s[i + z[i]] == s[i - z[i]]) ++z[i];
        if (z[i] + i > r) r = z[i] + i, l = i;
    }
    for (int i = 1; i < SZ(s); ++i)
        if (s[i] == '%') x = max(x, z[i]);
    ans = x / 2 * 2, x = 0;
    for (int i = 1; i < SZ(s); ++i)
        if (s[i] != '%') x = max(x, z[i]);
    return max(ans, (x - 1) / 2 * 2 + 1);
}

```

5.4 SAIS*

```

class SAIS {
public:
    int *SA, *H;
    // zero based, string content MUST > 0
    // result height H[i] is LCP(SA[i - 1], SA[i])
    // string, length, |sigma|
    void build(int *s, int n, int m = 128) {
        copy_n(s, n, _s);
        _h[0] = _s[n++] = 0;
        sais(_s, _sa, _p, _q, _t, _c, n, m);
        mkhei(n);
        SA = _sa + 1;
        H = _h + 1;
    }

private:
    bool _t[N * 2];
    int _s[N * 2], _c[N * 2], x[N], _p[N], _q[N * 2], r[N],
        _sa[N * 2], _h[N];
    void mkhei(int n) {
        for (int i = 0; i < n; ++i) r[_sa[i]] = i;
        for (int i = 0; i < n; ++i)
            if (r[i]) {
                int ans = i > 0 ? max(_h[r[i] - 1], 0) : 0;
                while (_s[i + ans] == _s[_sa[r[i] - 1] + ans]) ans++;
                _h[r[i]] = ans;
            }
    }
    void sais(int *s, int *sa, int *p, int *q, bool *t, int *c, int n, int z) {

```

```

bool uniq = t[n - 1] = 1, neq;
int nn = 0, nmzx = -1, *nsa = sa + n, *ns = s + n, lst
    = -1;

#define MAGIC(XD)
    \
fill_n(sa, n, 0);
    \
copy_n(c, z, x);
    \
XD;
    \
copy_n(c, z - 1, x + 1);
    \
for (int i = 0; i < n; i++)
    \
    if (sa[i] && !t[sa[i] - 1]) sa[x[s[sa[i] - 1]]++] = sa[
        i] - 1; \
copy_n(c, z, x);
    \
for (int i = n - 1; i >= 0; i--)
    \
    if (sa[i] && t[sa[i] - 1]) sa[--x[s[sa[i] - 1]]] = sa[i
        ] - 1;

fill_n(c, z, 0);
for (int i = 0; i < n; i++) uniq &= ++c[s[i]] < 2;
partial_sum(c, c + z, c);
if (uniq) {
    for (int i = 0; i < n; i++) sa[--c[s[i]]] = i;
    return;
}
for (int i = n - 2; i >= 0; i--) t[i] = (s[i] == s[i +
    1] ? t[i + 1] : s[i] < s[i + 1]);
MAGIC(for (int i = 1; i <= n - 1; i++) if (t[i] && !t[i
    - 1]) sa[--x[s[i]]] = p[q[i] = nn++] = i);
for (int i = 0; i < n; i++)
    if (sa[i] && t[sa[i]] && !t[sa[i] - 1]) {
        neq = (lst < 0) || !equal(s + lst, s + lst + p[q[sa
            [i]] + 1] - sa[i], s + sa[i]);
        ns[q[lst = sa[i]]] = nmzx += neq;
    }
sais(ns, nsa, p + nn, q + n, t + n, c + z, nn, nmzx +
    1);
MAGIC(for (int i = nn - 1; i >= 0; i--) sa[--x[s[p[nsa[
    i]]]] = p[nsa[i]]);
}
} sa;

```

5.5 Aho-Corasick Automatan

```

const int len = 400000, sigma = 26;
struct AC_Automatan {
    int nx[len][sigma], fl[len], cnt[len], pri[len], top;
    int newnode() {
        fill(nx[top], nx[top] + sigma, -1);
        return top++;
    }
    void init() { top = 1, newnode(); }
    int input(string &s) { // return the end_node of string
        int X = 1;
        for (char c : s) {
            if (!~nx[X][c - 'a']) nx[X][c - 'a'] = newnode();
            X = nx[X][c - 'a'];
        }
        return X;
    }
    void make_fl() {
        queue<int> q;
        q.push(1), fl[1] = 0;
        for (int t = 0; !q.empty(); ) {
            int R = q.front();
            q.pop(), pri[t++] = R;
            for (int i = 0; i < sigma; ++i)
                if (~nx[R][i]) {
                    int X = nx[R][i], Z = fl[R];
                    for (; Z && !~nx[Z][i]; Z = fl[Z]);
                    fl[X] = Z ? nx[Z][i] : 1, q.push(X);
                }
        }
    }
    void get_v(string &s) {
        int X = 1;
        fill(cnt, cnt + top, 0);
        for (char c : s) {

```

```

while (X && !~nx[X][c - 'a']) X = fl[X];
X = X ? nx[X][c - 'a'] : 1, ++cnt[X];
}
for (int i = top - 2; i > 0; --i) cnt[fl[pri[i]]] +=
    cnt[pri[i]];
}
};

```

5.6 Smallest Rotation

```

string mcp(string s) {
    int n = s.size(), i = 0, j = 1;
    s += s;
    while (i < n && j < n) {
        int k = 0;
        while (k < n && s[i + k] == s[j + k]) ++k;
        if (s[i + k] <= s[j + k])
            j += k + 1;
        else
            i += k + 1;
        if (i == j) ++j;
    }
    int ans = i < n ? i : j;
    return s.substr(ans, n);
}

```

5.7 De Bruijn sequence*

```

constexpr int MAXC = 10, MAXN = 1e5 + 10;
struct DBSeq {
    int C, N, K, L, buf[MAXC * MAXN]; // K <= C^N
    void dfs(int *out, int t, int p, int &ptr) {
        if (ptr >= L) return;
        if (t > N) {
            if (N % p) return;
            for (int i = 1; i <= p && ptr < L; ++i) out[ptr++] =
                buf[i];
        } else {
            buf[t] = buf[t - p], dfs(out, t + 1, p, ptr);
            for (int j = buf[t - p] + 1; j < C; ++j) buf[t] = j,
                dfs(out, t + 1, t, ptr);
        }
    }
    void solve(int _c, int _n, int _k, int *out) {
        int p = 0;
        C = _c, N = _n, K = _k, L = N + K - 1;
        dfs(out, 1, 1, p);
        if (p < L) fill(out + p, out + L, 0);
    }
} dbs;

```

5.8 SAM

```

const int MAXM = 1000010;
struct SAM {
    int tot, root, lst, mom[MAXM], mx[MAXM];
    int acc[MAXM], nxt[MAXM][33];
    int newNode() {
        int res = ++tot;
        fill(nxt[res], nxt[res] + 33, 0);
        mom[res] = mx[res] = acc[res] = 0;
        return res;
    }
    void init() {
        tot = 0;
        root = newNode();
        mom[root] = 0, mx[root] = 0;
        lst = root;
    }
    void push(int c) {
        int p = lst;
        int np = newNode();
        mx[np] = mx[p] + 1;
        for (; p && nxt[p][c] == 0; p = mom[p]) nxt[p][c] = np;
        if (p == 0)
            mom[np] = root;
        else {
            int q = nxt[p][c];
            if (mx[p] + 1 == mx[q])
                mom[np] = q;
            else {
                int nq = newNode();
                mx[nq] = mx[p] + 1;
                for (int i = 0; i < 33; i++) nxt[nq][i] = nxt[q][i];
                mom[nq] = mom[q];

```

```

    mom[q] = nq;
    mom[np] = nq;
    for (; p && nxt[p][c] == q; p = mom[p]) nxt[p][c] =
        nq;
}
}
lst = np;
}
void push(char *str) {
    for (int i = 0; str[i]; i++) push(str[i] - 'a' + 1);
}
} sam;

```

5.9 PalTree

```

struct palindromic_tree { // Check by APIO 2014
    // palindrome
    struct node {
        int next[26], fail, len;
        int cnt, num; // cnt: appear times, num: number of
        // pal. suf.
        node(int l = 0) : fail(0), len(l), cnt(0), num(0) {
            for (int i = 0; i < 26; ++i) next[i] = 0;
        }
    };
    vector<node> St;
    vector<char> s;
    int last, n;
    palindromic_tree() : St(2), last(1), n(0) { St[0].fail =
        1, St[1].len = -1, s.pb(-1); }
    inline void clear() {
        St.clear(), s.clear(), last = 1, n = 0;
        St.pb(0), St.pb(-1);
        St[0].fail = 1, s.pb(-1);
    }
    inline int get_fail(int x) {
        while (s[n - St[x].len - 1] != s[n]) x = St[x].fail;
        return x;
    }
    inline void add(int c) {
        s.push_back(c - 'a'), ++n;
        int cur = get_fail(last);
        if (!St[cur].next[c]) {
            int now = SZ(St);
            St.pb(St[cur].len + 2);
            St[now].fail = St[get_fail(St[cur].fail)].next[c];
            St[cur].next[c] = now;
            St[now].num = St[St[now].fail].num + 1;
        }
        last = St[cur].next[c], ++St[last].cnt;
    }
    inline void count() { // counting cnt
        auto i = St.rbegin();
        for (; i != St.rend(); ++i) {
            St[i->fail].cnt += i->cnt;
        }
    }
    inline int size() { // The number of diff. pal.
        return SZ(St) - 2;
    }
};

```

5.10 cyclicLCS

```

#define L 0
#define LU 1
#define U 2
const int mov[3][2] = {0, -1, -1, -1, -1, 0};
int al, bl;
char a[MAXL * 2], b[MAXL * 2]; // 0-indexed
int dp[MAXL * 2][MAXL];
char pred[MAXL * 2][MAXL];
inline int lcs_length(int r) {
    int i = r + al, j = bl, l = 0;
    while (i > r) {
        char dir = pred[i][j];
        if (dir == LU) l++;
        i += mov[dir][0];
        j += mov[dir][1];
    }
    return l;
}
inline void reroot(int r) { // r = new base row
    int i = r, j = 1;
    while (j <= bl && pred[i][j] != LU) j++;
    if (j > bl) return;
}

```

```

pred[i][j] = L;
while (i < 2 * al && j <= bl) {
    if (pred[i + 1][j] == U) {
        i++;
        pred[i][j] = L;
    } else if (j < bl && pred[i + 1][j + 1] == LU) {
        i++;
        j++;
        pred[i][j] = L;
    } else {
        j++;
    }
}
}
}
int cyclic_lcs() {
    // a, b, al, bl should be properly filled
    // note: a WILL be altered in process
    // -- concatenated after itself
    char tmp[MAXL];
    if (al > bl) {
        swap(al, bl);
        strcpy(tmp, a);
        strcpy(a, b);
        strcpy(b, tmp);
    }
    strcpy(tmp, a);
    strcat(a, tmp);
    // basic lcs
    for (int i = 0; i <= 2 * al; i++) {
        dp[i][0] = 0;
        pred[i][0] = U;
    }
    for (int j = 0; j <= bl; j++) {
        dp[0][j] = 0;
        pred[0][j] = L;
    }
    for (int i = 1; i <= 2 * al; i++) {
        for (int j = 1; j <= bl; j++) {
            if (a[i - 1] == b[j - 1])
                dp[i][j] = dp[i - 1][j - 1] + 1;
            else
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
            if (dp[i][j - 1] == dp[i][j])
                pred[i][j] = L;
            else if (a[i - 1] == b[j - 1])
                pred[i][j] = LU;
            else
                pred[i][j] = U;
        }
    }
    // do cyclic lcs
    int clcs = 0;
    for (int i = 0; i < al; i++) {
        clcs = max(clcs, lcs_length(i));
        reroot(i + 1);
    }
    // recover a
    a[al] = '\0';
    return clcs;
}

```

5.11 Suffix Array

```

#define FILL(x, y) memset(x, y, sizeof(x))
struct suffix_array {
    int box[100007], tp[100007], m;
    bool not_equ(int a, int b, int k, int n) { return ra[a]
        != ra[b] || a + k >= n || b + k >= n || ra[a + k] !=
        ra[b + k]; }
    void radix(int *key, int *it, int *ot, int n) {
        fill_n(box, m, 0);
        for (int i = 0; i < n; ++i) ++box[key[i]];
        partial_sum(box, box + m, box);
        for (int i = n - 1; i >= 0; --i) ot[--box[key[it[i]]]]
            = it[i];
    }
    void make_sa(string s, int n) {
        int k = 1;
        for (int i = 0; i < n; ++i) ra[i] = s[i];
        do {
            iota(tp, tp + k, n - k), iota(sa + k, sa + n, 0);
            radix(ra + k, sa + k, tp + k, n - k);
            radix(ra, tp, sa, n);
            tp[sa[0]] = 0, m = 1;
            for (int i = 1; i < n; ++i) {
                m += not_equ(sa[i], sa[i - 1], k, n);
            }
        } while (m < n);
    }
}

```

```

        tp[sa[i]] = m - 1;
    }
    copy_n(tp, n, ra);
    k *= 2;
} while (k < n && m != n);
}
void make_he(string s, int n) {
    for (int j = 0, k = 0; j < n; ++j) {
        if (ra[j])
            for (; s[j + k] == s[sa[ra[j] - 1] + k]; ++k)
                he[ra[j]] = k, k = max(0, k - 1);
    }
}
int sa[100007], ra[100007], he[100007];
void build(string s) {
    FILL(sa, 0), FILL(ra, 0), FILL(he, 0);
    FILL(box, 0), FILL(tp, 0), m = 256;
    make_sa(s, s.size());
    make_he(s, s.size());
}
};
main() {
    string s;
    cin >> s;
    suffix_array saa;
    saa.build(s);
    for (int i = 0; i <= s.length(); i++) {
        cout << i << " " << saa.sa[i] << " " << saa.ra[i] << "
            " << saa.he[i] << endl;
    }
}

```

5.12 Suffix Array2

```

// array c is eventually equal to the position of the
// suffixes in the suffix
// array don't add another '$' to the string
int sa[MAXN], c[MAXN], sa_new[MAXN], c_new[MAXN], cnt[MAXN]
    , pos[MAXN], lcp[MAXN];
pair<char, int> P[MAXN];
void calc_suffix_array(string s) {
    s += '$';
    int n = s.size();
    for (int i = 0; i < n; i++) P[i] = {s[i], i};
    sort(P, P + n);
    for (int i = 0; i < n; i++) sa[i] = P[i].second;
    c[sa[0]] = 0;
    for (int i = 1; i < n; i++) c[sa[i]] = c[sa[i - 1]] + (P[i].first > P[i - 1].first ? 1 : 0);
    int k = 1;
    while (k < n) {
        for (int i = 0; i < n; i++) sa[i] = (sa[i] - k + n) % n;
        for (int i = 0; i < n; i++) cnt[i] = 0;
        for (int i = 0; i < n; i++) cnt[c[i]]++;
        pos[0] = cnt[0] - 1;
        for (int i = 1; i < n; i++) pos[i] = pos[i - 1] + cnt[i];
        for (int i = n - 1; i >= 0; i--) sa_new[pos[c[sa[i]]]--] = sa[i];
        for (int i = 0; i < n; i++) sa[i] = sa_new[i];
        c_new[sa[0]] = 0;
        for (int i = 1; i < n; i++) {
            c_new[sa[i]] = c_new[sa[i - 1]];
            pair<int, int> prev = {c[sa[i - 1]], c[(sa[i - 1] + k) % n]};
            pair<int, int> now = {c[sa[i]], c[(sa[i] + k) % n]};
            if (now > prev) c_new[sa[i]]++;
        }
        for (int i = 0; i < n; i++) c[i] = c_new[i];
        k *= 2;
    }
}
void calc_lcp_array(string s) {
    int n = s.size(), k = 0;
    for (int i = 0; i < n; i++) {
        int j = sa[c[i] - 1];
        while (i + k < n && j + k < n && s[i + k] == s[j + k])
            k++;
        lcp[c[i] - 1] = k;
        k = max(k - 1, 0);
    }
}

```

6 Math

6.1 exgcd*

```

pll exgcd(ll a, ll b) {
    if (b == 0)
        return pll(1, 0);
    else {
        ll p = a / b;
        pll q = exgcd(b, a % b);
        return pll(q.Y, q.X - q.Y * p);
    }
}

```

6.2 floor and ceil

```

int floor(int a, int b) { return a / b - (a % b && a < 0 ^ b < 0); }
int ceil(int a, int b) { return a / b + (a % b && a < 0 ^ b > 0); }

```

6.3 Miller Rabin*

```

// n < 4,759,123,141      3 : 2, 7, 61
// n < 1,122,004,669,633  4 : 2, 13, 23, 1662803
// n < 3,474,749,660,383  6 : pimes <= 13
// n < 2^64              7 :
// 2, 325, 9375, 28178, 450775, 9780504, 1795265022
bool Miller_Rabin(ll a, ll n) {
    if ((a = a % n) == 0) return 1;
    if ((n & 1) ^ 1) return n == 2;
    ll tmp = (n - 1) / ((n - 1) & (1 - n));
    ll t = __lg(((n - 1) & (1 - n))), x = 1;
    for (; tmp; tmp >>= 1, a = mul(a, a, n))
        if (tmp & 1) x = mul(x, a, n);
    if (x == 1 || x == n - 1) return 1;
    while (--t)
        if ((x = mul(x, x, n)) == n - 1) return 1;
    return 0;
}

```

6.4 Big number

```

template <typename T>
inline string to_string(const T &x) {
    stringstream ss;
    return ss << x, ss.str();
}
struct bigN : vector<ll> {
    const static int base = 1e9, width = log10(base);
    bool negative;
    bigN(const_iterator a, const_iterator b) : vector<ll>(a, b) {}
    bigN(string s) {
        if (s.empty()) return;
        if (s[0] == '-')
            negative = 1, s = s.substr(1);
        else
            negative = 0;
        for (int i = int(s.size()) - 1; i >= 0; i -= width) {
            ll t = 0;
            for (int j = max(0, i - width + 1); j <= i; ++j) t =
                t * 10 + s[j] - '0';
            push_back(t);
        }
        trim();
    }
    template <typename T>
    bigN(const T &x) : bigN(to_string(x)) {}
    bigN() : negative(0) {}
    void trim() {
        while (size() && !back()) pop_back();
        if (empty()) negative = 0;
    }
    void carry(int _base = base) {
        for (size_t i = 0; i < size(); ++i) {
            if (at(i) >= 0 && at(i) < _base) continue;
            if (i + 1u == size()) push_back(0);
            int r = at(i) % _base;
            if (r < 0) r += _base;
            at(i + 1) += (at(i) - r) / _base, at(i) = r;
        }
    }
    int abscmp(const bigN &b) const {
        if (size() > b.size()) return 1;
        if (size() < b.size()) return -1;
    }
}

```



```

    for (int i = int(size()) - 1; i >= 0; --i) {
        if (at(i) > b[i]) return 1;
        if (at(i) < b[i]) return -1;
    }
    return 0;
}
int cmp(const bigN &b) const {
    if (negative != b.negative) return negative ? -1 : 1;
    return negative ? -abscmp(b) : abscmp(b);
}
bool operator<(const bigN &b) const { return cmp(b) < 0; }
bool operator>(const bigN &b) const { return cmp(b) > 0; }
bool operator<=(const bigN &b) const { return cmp(b) <= 0; }
bool operator>=(const bigN &b) const { return cmp(b) >= 0; }
bool operator==(const bigN &b) const { return !cmp(b); }
bool operator!=(const bigN &b) const { return cmp(b) != 0; }
bigN abs() const {
    bigN res = *this;
    return res.negative = 0, res;
}
bigN operator-() const {
    bigN res = *this;
    return res.negative = !negative, res.trim(), res;
}
bigN operator+(const bigN &b) const {
    if (negative) return -(*this) + (-b);
    if (b.negative) return *this - (-b);
    bigN res = *this;
    if (b.size() > size()) res.resize(b.size());
    for (size_t i = 0; i < b.size(); ++i) res[i] += b[i];
    return res.carry(), res.trim(), res;
}
bigN operator-(const bigN &b) const {
    if (negative) return -(*this) - (-b);
    if (b.negative) return *this + (-b);
    if (abscmp(b) < 0) return -(b - (*this));
    bigN res = *this;
    if (b.size() > size()) res.resize(b.size());
    for (size_t i = 0; i < b.size(); ++i) res[i] -= b[i];
    return res.carry(), res.trim(), res;
}
bigN operator*(const bigN &b) const {
    bigN res;
    res.negative = negative != b.negative;
    res.resize(size() + b.size());
    for (size_t i = 0; i < size(); ++i)
        for (size_t j = 0; j < b.size(); ++j)
            if ((res[i + j] += at(i) * b[j]) >= base) {
                res[i + j + 1] += res[i + j] / base;
                res[i + j] %= base;
            }
    return res.trim(), res;
}
bigN operator/(const bigN &b) const {
    int norm = base / (b.back() + 1);
    bigN x = abs() * norm;
    bigN y = b.abs() * norm;
    bigN q, r;
    q.resize(x.size());
    for (int i = int(x.size()) - 1; i >= 0; --i) {
        r = r * base + x[i];
        int s1 = r.size() <= y.size() ? 0 : r[y.size()];
        int s2 = r.size() < y.size() ? 0 : r[y.size() - 1];
        int d = (ll(base) * s1 + s2) / y.back();
        r = r - y * d;
        while (r.negative) r = r + y, --d;
        q[i] = d;
    }
    q.negative = negative != b.negative;
    return q.trim(), q;
}
bigN operator%(const bigN &b) const { return *this - (*this / b) * b; }
friend istream &operator>>(istream &ss, bigN &b) {
    string s;
    return ss >> s, b = s, ss;
}
friend ostream &operator<<(ostream &ss, const bigN &b) {
    if (b.negative) ss << '-';
    ss << (b.empty() ? 0 : b.back());
    for (int i = int(b.size()) - 2; i >= 0; --i) ss << setw

```

```

        (width) << setfill('0') << b[i];
    }
    return ss;
}
template <typename T>
operator T() {
    stringstream ss;
    ss << *this;
    T res;
    return ss >> res, res;
}
};

```

6.5 Fraction

```

struct fraction {
    // n / d
    ll n, d;
    fraction(const ll &n = 0, const ll &d = 1) : n(_n), d(
        _d) {
        ll t = __gcd(n, d);
        n /= t, d /= t;
        if (d < 0) n = -n, d = -d;
    }
    fraction operator-() const { return fraction(-n, d); }
    fraction operator+(const fraction &b) const { return
        fraction(n * b.d + b.n * d, d * b.d); }
    fraction operator-(const fraction &b) const { return
        fraction(n * b.d - b.n * d, d * b.d); }
    fraction operator*(const fraction &b) const { return
        fraction(n * b.n, d * b.d); }
    fraction operator/(const fraction &b) const { return
        fraction(n * b.d, d * b.n); }
};

```

6.6 Simultaneous Equations

```

struct matrix { // m variables, n equations
    int n, m;
    fraction M[MAXN][MAXN + 1], sol[MAXN];
    int solve() { // -1: inconsistent, >= 0: rank
        for (int i = 0; i < n; ++i) {
            int piv = 0;
            while (piv < m && !M[i][piv].n) ++piv;
            if (piv == m) continue;
            for (int j = 0; j < n; ++j) {
                if (i == j) continue;
                fraction tmp = -M[j][piv] / M[i][piv];
                for (int k = 0; k <= m; ++k) M[j][k] = tmp * M[i][k]
                    + M[j][k];
            }
        }
        int rank = 0;
        for (int i = 0; i < n; ++i) {
            int piv = 0;
            while (piv < m && !M[i][piv].n) ++piv;
            if (piv == m && M[i][m].n)
                return -1;
            else if (piv < m)
                ++rank, sol[piv] = M[i][m] / M[i][piv];
        }
        return rank;
    }
};

```

6.7 Pollard Rho

```

// n is prime -> miller_rabin
// qpow need int128
ll Pollard_Rho(ll x) {
    ll s = 0, t = 0;
    ll c = (ll)rand() % (x - 1) + 1;
    int step = 0, goal = 1;
    ll val = 1;
    for (goal = 1; goal * 2, s = t, val = 1) {
        for (step = 1; step <= goal; ++step) {
            t = (qpow(t, 2, x) + c) % x;
            val = (__int128)val * abs(t - s) % x;
            if ((step % 127) == 0) {
                ll d = __gcd(val, x);
                if (d > 1) return d;
            }
        }
        ll d = __gcd(val, x);
        if (d > 1) return d;
    }
}

```

6.8 Simplex Algorithm

```

const int MAXN = 111;
const int MAXM = 111;
const double eps = 1E-10;
double a[MAXN][MAXM], b[MAXN], c[MAXM], d[MAXN][MAXM];
double x[MAXM];
int ix[MAXN + MAXM]; // !!! array all indexed from 0
// max{cx} subject to {Ax<=b,x>=0}
// n: constraints, m: vars !!!
// x[] is the optimal solution vector
// usage :
// value = simplex(a, b, c, N, M);
double simplex(double a[MAXN][MAXM], double b[MAXN], double
c[MAXM], int n, int m) {
    ++m;
    int r = n, s = m - 1;
    memset(d, 0, sizeof(d));
    for (int i = 0; i < n + m; ++i) ix[i] = i;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m - 1; ++j) d[i][j] = -a[i][j];
        d[i][m - 1] = 1;
        d[i][m] = b[i];
        if (d[r][m] > d[i][m]) r = i;
    }
    for (int j = 0; j < m - 1; ++j) d[n][j] = c[j];
    d[n + 1][m - 1] = -1;
    for (double dd;;) {
        if (r < n) {
            int t = ix[s];
            ix[s] = ix[r + m];
            ix[r + m] = t;
            d[r][s] = 1.0 / d[r][s];
            for (int j = 0; j <= m; ++j)
                if (j != s) d[r][j] *= -d[r][s];
            for (int i = 0; i <= n + 1; ++i)
                if (i != r) {
                    for (int j = 0; j <= m; ++j)
                        if (j != s) d[i][j] += d[r][j] * d[i][s];
                    d[i][s] *= d[r][s];
                }
        }
        r = -1;
        s = -1;
        for (int j = 0; j < m; ++j)
            if (s < 0 || ix[s] > ix[j]) {
                if (d[n + 1][j] > eps || (d[n + 1][j] > -eps && d[n
][j] > eps)) s = j;
            }
        if (s < 0) break;
        for (int i = 0; i < n; ++i)
            if (d[i][s] < -eps) {
                if (r < 0 || (dd = d[r][m] / d[r][s] - d[i][m] / d[
i][s]) < -eps || (dd < eps && ix[r + m] > ix[i
+ m])) r = i;
            }
        if (r < 0) return -1; // not bounded
    }
    if (d[n + 1][m] < -eps) return -1; // not executable
    double ans = 0;
    for (int i = 0; i < m; i++) x[i] = 0;
    for (int i = m; i < n + m; ++i) { // the missing
        enumerated x[i] = 0
        if (ix[i] < m - 1) {
            ans += d[i - m][m] * c[ix[i]];
            x[ix[i]] = d[i - m][m];
        }
    }
    return ans;
}

```

6.8.1 Construction

Standard form: maximize $c^T x$ subject to $Ax \leq b$ and $x \geq 0$.

Dual LP: minimize $b^T y$ subject to $A^T y \geq c$ and $y \geq 0$.

\bar{x} and \bar{y} are optimal if and only if for all $i \in [1, n]$, either $\bar{x}_i = 0$ or $\sum_{j=1}^m A_{ij} \bar{y}_j = c_i$ holds and for all $i \in [1, m]$ either $\bar{y}_i = 0$ or $\sum_{j=1}^n A_{ij} \bar{x}_j = b_j$ holds.

1. In case of minimization, let $c'_i = -c_i$
2. $\sum_{1 \leq i \leq n} A_{ij} x_i \geq b_j \rightarrow \sum_{1 \leq i \leq n} -A_{ij} x_i \leq -b_j$
3. $\sum_{1 \leq i \leq n} A_{ij} x_i = b_j$
 - $\bullet \sum_{1 \leq i \leq n} A_{ij} x_i \leq b_j$
 - $\bullet \sum_{1 \leq i \leq n} A_{ij} x_i \geq b_j$
4. If x_i has no lower bound, replace x_i with $x_i - x'_i$

6.9 Schreier-Sims Algorithm*

```

namespace schreier {
int n;
vector<vector<vector<int>>> bkts, binv;
vector<vector<int>> lk;
vector<int> operator*(const vector<int> &a, const vector<
int> &b) {
    vector<int> res(SZ(a));
    for (int i = 0; i < SZ(a); ++i) res[i] = b[a[i]];
    return res;
}
vector<int> inv(const vector<int> &a) {
    vector<int> res(SZ(a));
    for (int i = 0; i < SZ(a); ++i) res[a[i]] = i;
    return res;
}
int filter(const vector<int> &g, bool add = true) {
    n = SZ(bkts);
    vector<int> p = g;
    for (int i = 0; i < n; ++i) {
        assert(p[i] >= 0 && p[i] < SZ(lk[i]));
        if (lk[i][p[i]] == -1) {
            if (add) {
                bkts[i].pb(p);
                binv[i].pb(inv(p));
                lk[i][p[i]] = SZ(bkts[i]) - 1;
            }
            return i;
        }
        p = p * binv[i][lk[i][p[i]]];
    }
    return -1;
}
bool inside(const vector<int> &g) { return filter(g, false)
== -1; }
void solve(const vector<vector<int>> &gen, int _n) {
    n = _n;
    bkts.clear(), bkts.resize(n);
    binv.clear(), binv.resize(n);
    lk.clear(), lk.resize(n);
    vector<int> iden(n);
    iota(iden.begin(), iden.end(), 0);
    for (int i = 0; i < n; ++i) {
        lk[i].resize(n, -1);
        bkts[i].pb(iden);
        binv[i].pb(iden);
        lk[i][i] = 0;
    }
    for (int i = 0; i < SZ(gen); ++i) filter(gen[i]);
    queue<pair<pii, pii>> upd;
    for (int i = 0; i < n; ++i)
        for (int j = i; j < n; ++j)
            for (int k = 0; k < SZ(bkts[i]); ++k)
                for (int l = 0; l < SZ(bkts[j]); ++l) upd.emplace(
                    pii(i, k), pii(j, l));
    while (!upd.empty()) {
        auto a = upd.front().X;
        auto b = upd.front().Y;
        upd.pop();
        int res = filter(bkts[a.X][a.Y] * bkts[b.X][b.Y]);
        if (res == -1) continue;
        pii pr = pii(res, SZ(bkts[res]) - 1);
        for (int i = 0; i < n; ++i)
            for (int j = 0; j < SZ(bkts[i]); ++j) {
                if (i <= res) upd.emplace(pii(i, j), pr);
                if (res <= i) upd.emplace(pr, pii(i, j));
            }
    }
}
long long size() {
    long long res = 1;
    for (int i = 0; i < n; ++i) res = res * SZ(bkts[i]);
    return res;
}
} // namespace schreier

```

6.10 chineseRemainder

```

LL solve(LL x1, LL m1, LL x2, LL m2) {
    LL g = __gcd(m1, m2);
    if ((x2 - x1) % g) return -1; // no sol
    m1 /= g;
    m2 /= g;
    pair<LL, LL> p = gcd(m1, m2);
    LL lcm = m1 * m2 * g;

```

```
LL res = p.first * (x2 - x1) * m1 + x1;
return (res % lcm + lcm) % lcm;
}
```

6.11 QuadraticResidue

```
int Jacobi(int a, int m) {
    int s = 1;
    for (; m > 1;) {
        a %= m;
        if (a == 0) return 0;
        const int r = __builtin_ctz(a);
        if ((r & 1) && ((m + 2) & 4)) s = -s;
        a >>= r;
        if (a & m & 2) s = -s;
        swap(a, m);
    }
    return s;
}

int QuadraticResidue(int a, int p) {
    if (p == 2) return a & 1;
    const int jc = Jacobi(a, p);
    if (jc == 0) return 0;
    if (jc == -1) return -1;
    int b, d;
    for (;;) {
        b = rand() % p;
        d = (1LL * b * b + p - a) % p;
        if (Jacobi(d, p) == -1) break;
    }
    int f0 = b, f1 = 1, g0 = 1, g1 = 0, tmp;
    for (int e = (1LL + p) >> 1; e; e >>= 1) {
        if (e & 1) {
            tmp = (1LL * g0 * f0 + 1LL * d * (1LL * g1 * f1 % p)) % p;
            g1 = (1LL * g0 * f1 + 1LL * g1 * f0) % p;
            g0 = tmp;
        }
        tmp = (1LL * f0 * f0 + 1LL * d * (1LL * f1 * f1 % p)) % p;
        f1 = (2LL * f0 * f1) % p;
        f0 = tmp;
    }
    return g0;
}
```

6.12 Discrete Log

```
int DiscreteLog(int s, int x, int y, int m) {
    constexpr int kStep = 32000;
    unordered_map<int, int> p;
    int b = 1;
    for (int i = 0; i < kStep; ++i) {
        p[y] = i;
        y = 1LL * y * x % m;
        b = 1LL * b * x % m;
    }
    for (int i = 0; i < m + 10; i += kStep) {
        s = 1LL * s * b % m;
        if (p.find(s) != p.end()) return i + kStep - p[s];
    }
    return -1;
}

int DiscreteLog(int x, int y, int m) {
    if (m == 1) return 0;
    int s = 1;
    for (int i = 0; i < 100; ++i) {
        if (s == y) return i;
        s = 1LL * s * x % m;
    }
    if (s == y) return 100;
    int p = 100 + DiscreteLog(s, x, y, m);
    if (fpow(x, p, m) != y) return -1;
    return p;
}
```

6.13 PiCount

```
int64_t PrimeCount(int64_t n) {
    if (n <= 1) return 0;
    const int v = sqrt(n);
    vector<int> smalls(v + 1);
    for (int i = 2; i <= v; ++i) smalls[i] = (i + 1) / 2;
    int s = (v + 1) / 2;
    vector<int> roughs(s);
```

```
for (int i = 0; i < s; ++i) roughs[i] = 2 * i + 1;
vector<int64_t> larges(s);
for (int i = 0; i < s; ++i) larges[i] = (n / (2 * i + 1) + 1) / 2;
vector<bool> skip(v + 1);
int pc = 0;
for (int p = 3; p <= v; ++p) {
    if (smalls[p] > smalls[p - 1]) {
        int q = p * p;
        pc++;
        if (1LL * q * q > n) break;
        skip[p] = true;
        for (int i = q; i <= v; i += 2 * p) skip[i] = true;
        int ns = 0;
        for (int k = 0; k < s; ++k) {
            int i = roughs[k];
            if (skip[i]) continue;
            int64_t d = 1LL * i * p;
            larges[ns] = larges[k] - (d <= v ? larges[smalls[d] - pc] : smalls[n / d]) + pc;
            roughs[ns++] = i;
        }
        s = ns;
        for (int j = v / p; j >= p; --j) {
            int c = smalls[j] - pc;
            for (int i = j * p, e = min(i + p, v + 1); i < e; ++i) smalls[i] -= c;
        }
    }
    for (int k = 1; k < s; ++k) {
        const int64_t m = n / roughs[k];
        int64_t s = larges[k] - (pc + k - 1);
        for (int l = 1; l < k; ++l) {
            int p = roughs[l];
            if (1LL * p * p > m) break;
            s -= smalls[m / p] - (pc + l - 1);
        }
        larges[0] -= s;
    }
    return larges[0];
}
```

6.14 Primes

```
/*
12721 13331 14341 75577 123457 222557 556679 999983
1097774749 1076767633
100102021 999997771 1001010013 1000512343 987654361
999991231 999888733
98789101 987777733 999991921 1010101333 1010102101
10000000000039
100000000000037 2305843009213693951 4611686018427387847
9223372036854775783
18446744073709551557
*/
```

6.15 Theorem

6.15.1 Kirchhoff's Theorem

Denote L be a $n \times n$ matrix as the Laplacian matrix of graph G , where $L_{ii} = d(i)$, $L_{ij} = -c$ where c is the number of edge (i, j) in G .

- The number of undirected spanning in G is $|\det(\tilde{L}_{11})|$.
- The number of directed spanning tree rooted at r in G is $|\det(\tilde{L}_{rr})|$.

6.15.2 Tutte's Matrix

Let D be a $n \times n$ matrix, where $d_{ij} = x_{ij}$ (x_{ij} is chosen uniformly at random) if $i < j$ and $(i, j) \in E$, otherwise $d_{ij} = -d_{ji}$. $\frac{\text{rank}(D)}{2}$ is the maximum matching on G .

6.15.3 Cayley's Formula

- Given a degree sequence d_1, d_2, \dots, d_n for each labeled vertices, there are $\frac{(n-2)!}{(d_1-1)!(d_2-1)!\dots(d_n-1)!}$ spanning trees.
- Let $T_{n,k}$ be the number of labeled forests on n vertices with k components, such that vertex $1, 2, \dots, k$ belong to different components. Then $T_{n,k} = kn^{n-k-1}$.

6.15.4 Erdős–Gallai theorem

A sequence of nonnegative integers $d_1 \geq \dots \geq d_n$ can be represented as the degree sequence of a finite simple graph on n vertices if and only if

$d_1 + \dots + d_n$ is even and $\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min(d_i, k)$ holds for every $1 \leq k \leq n$.

6.15.5 Gale–Ryser theorem

A pair of sequences of nonnegative integers $a_1 \geq \dots \geq a_n$ and b_1, \dots, b_n is bigraphic if and only if $\sum_{i=1}^n a_i = \sum_{i=1}^n b_i$ and $\sum_{i=1}^k a_i \leq \sum_{i=1}^k \min(b_i, k)$ holds for every $1 \leq k \leq n$.

6.15.6 Fulkerson–Chen–Anstee theorem

A sequence $(a_1, b_1), \dots, (a_n, b_n)$ of nonnegative integer pairs with $a_1 \geq \dots \geq a_n$ is digraphic if and only if $\sum_{i=1}^n a_i = \sum_{i=1}^n b_i$ and $\sum_{i=1}^k a_i \leq \sum_{i=1}^k \min(b_i, k-1) + \sum_{i=k+1}^n \min(b_i, k)$ holds for every $1 \leq k \leq n$.

6.15.7 Pick's theorem

給定頂點座標均是整點（或正方形格子點）的簡單多邊形，皮克定理說明了其面積 A 和內部格點數目 i 、邊上格點數目 b 的關係： $A = i + \frac{b}{2} - 1$

6.16 Euclidean Algorithms

- $m = \lfloor \frac{an+b}{c} \rfloor$
- Time complexity: $O(\log n)$

$$f(a, b, c, n) = \sum_{i=0}^n \lfloor \frac{ai+b}{c} \rfloor$$
$$= \begin{cases} \lfloor \frac{a}{c} \rfloor \cdot \frac{n(n+1)}{2} + \lfloor \frac{b}{c} \rfloor \cdot (n+1) \\ + f(a \bmod c, b \bmod c, c, n), & a \geq c \vee b \geq c \\ 0, & n < 0 \vee a = 0 \\ nm - f(c, c-b-1, a, m-1), & \text{otherwise} \end{cases}$$
$$g(a, b, c, n) = \sum_{i=0}^n i \lfloor \frac{ai+b}{c} \rfloor$$
$$= \begin{cases} \lfloor \frac{a}{c} \rfloor \cdot \frac{n(n+1)(2n+1)}{6} + \lfloor \frac{b}{c} \rfloor \cdot \frac{n(n+1)}{2} \\ + g(a \bmod c, b \bmod c, c, n), & a \geq c \vee b \geq c \\ 0, & n < 0 \vee a = 0 \\ \frac{1}{2} \cdot (n(n+1)m - f(c, c-b-1, a, m-1)) \\ - h(c, c-b-1, a, m-1), & \text{otherwise} \end{cases}$$
$$h(a, b, c, n) = \sum_{i=0}^n \lfloor \frac{ai+b}{c} \rfloor^2$$
$$= \begin{cases} \lfloor \frac{a}{c} \rfloor^2 \cdot \frac{n(n+1)(2n+1)}{6} + \lfloor \frac{b}{c} \rfloor^2 \cdot (n+1) \\ + 2 \lfloor \frac{a}{c} \rfloor \cdot \lfloor \frac{b}{c} \rfloor \cdot n(n+1) \\ + h(a \bmod c, b \bmod c, c, n) \\ + 2 \lfloor \frac{a}{c} \rfloor \cdot g(a \bmod c, b \bmod c, c, n) \\ + 2 \lfloor \frac{b}{c} \rfloor \cdot f(a \bmod c, b \bmod c, c, n), & a \geq c \vee b \geq c \\ 0, & n < 0 \vee a = 0 \\ nm(m+1) - 2g(c, c-b-1, a, m-1) \\ - 2f(c, c-b-1, a, m-1) - f(a, b, c, n), & \text{otherwise} \end{cases}$$

7 Polynomial

7.1 Fast Fourier Transform

```
template <int MAXN>
struct FFT {
    using val_t = complex<double>;
    const double PI = acos(-1);
    val_t w[MAXN];
    FFT() {
        for (int i = 0; i < MAXN; ++i) {
            double arg = 2 * PI * i / MAXN;
            w[i] = val_t(cos(arg), sin(arg));
        }
    }
    void bitrev(val_t *a, int n); // see NTT
    void trans(val_t *a, int n, bool inv = false); // see NTT;
    // remember to replace LL with val_t
};
```

7.2 Number Theory Transform

```
//(2^16)+1, 65537, 3
// 7*17*(2^23)+1, 998244353, 3
// 1255*(2^20)+1, 1315962881, 3
// 51*(2^25)+1, 1711276033, 29
template <int MAXN, LL P, LL RT> // MAXN must be 2^k
struct NTT {
    LL w[MAXN];
    LL mpow(LL a, LL n);
    LL minv(LL a) { return mpow(a, P - 2); }
    NTT() {
        LL dw = mpow(RT, (P - 1) / MAXN);
        w[0] = 1;
        for (int i = 1; i < MAXN; ++i) w[i] = w[i - 1] * dw % P;
    }
};
```

```
}
void bitrev(LL *a, int n) {
    int i = 0;
    for (int j = 1; j < n - 1; ++j) {
        for (int k = n >> 1; (i ^ k) < k; k >>= 1)
            if (j < i) swap(a[i], a[j]);
    }
}
void operator()(LL *a, int n, bool inv = false) { // 0
    if (a[0] < P)
        bitrev(a, n);
    for (int L = 2; L <= n; L <= 1) {
        int dx = MAXN / L, dl = L >> 1;
        for (int i = 0; i < n; i += L) {
            for (int j = i, x = 0; j < i + dl; ++j, x += dx) {
                LL tmp = a[j + dl] * w[x] % P;
                if ((a[j + dl] = a[j] - tmp) < 0) a[j + dl] += P;
                if ((a[j] += tmp) >= P) a[j] -= P;
            }
        }
    }
    if (inv) {
        reverse(a + 1, a + n);
        LL invn = minv(n);
        for (int i = 0; i < n; ++i) a[i] = a[i] * invn % P;
    }
};
```

7.3 Fast Walsh Transform*

```
/* x: a[j], y: a[j + (L >> 1)]
or: (y += x * op), and: (x += y * op)
xor: (x, y = (x + y) * op, (x - y) * op)
inop: or, and, xor = -1, -1, 1/2 */
void fwt(int *a, int n, int op) { // or
    for (int L = 2; L <= n; L <= 1)
        for (int i = 0; i < n; i += L)
            for (int j = i; j < i + (L >> 1); ++j) a[j + (L >> 1)] += a[j] * op;
}
const int N = 21;
int f[N][1 << N], g[N][1 << N], h[N][1 << N], ct[1 << N];
void subset_convolution(int *a, int *b, int *c, int L) {
    // c_k = \sum_{i+j=k, i&j=0} a_i * b_j
    int n = 1 << L;
    for (int i = 1; i < n; ++i) ct[i] = ct[i & (i - 1)] + 1;
    for (int i = 0; i < n; ++i) f[ct[i]][i] = a[i], g[ct[i]][i] = b[i];
    for (int i = 0; i <= L; ++i) fwt(f[i], n, 1), fwt(g[i], n, 1);
    for (int i = 0; i <= L; ++i)
        for (int j = 0; j <= i; ++j)
            for (int x = 0; x < n; ++x) h[i][x] += f[j][x] * g[i - j][x];
    for (int i = 0; i <= L; ++i) fwt(h[i], n, -1);
    for (int i = 0; i < n; ++i) c[i] = h[ct[i]][i];
}
```

7.4 Newton's Method

Given $F(x)$ where

$$F(x) = \sum_{i=0}^{\infty} \alpha_i (x - \beta)^i$$

for β being some constant. Polynomial P such that $F(P) = 0$ can be found iteratively. Denote by Q_k the polynomial such that $F(Q_k) = 0 \pmod{x^{2^k}}$, then

$$Q_{k+1} = Q_k - \frac{F(Q_k)}{F'(Q_k)} \pmod{x^{2^{k+1}}}$$

8 Geometry

8.1 Default Code

```
typedef pair<double, double> pdd;
typedef pair<pdd, pdd> Line;
struct Cir {
    pdd O;
    double R;
};
const double eps = 1e-8;
pdd operator+(const pdd &a, const pdd &b) { return pdd(a.X + b.X, a.Y + b.Y); }
```

```

pdd operator-(const pdd &a, const pdd &b) { return pdd(a.X
- b.X, a.Y - b.Y); }
pdd operator*(const pdd &a, const double &b) { return pdd(a
.X * b, a.Y * b); }
pdd operator/(const pdd &a, const double &b) { return pdd(a
.X / b, a.Y / b); }
double dot(const pdd &a, const pdd &b) { return a.X * b.X +
a.Y * b.Y; }
double cross(const pdd &a, const pdd &b) { return a.X * b.Y
- a.Y * b.X; }
double abs2(const pdd &a) { return dot(a, a); }
double abs(const pdd &a) { return sqrt(dot(a, a)); }
int sign(const double &a) { return fabs(a) < eps ? 0 : a >
0 ? 1 : -1; }
int ori(const pdd &a, const pdd &b, const pdd &c) { return
sign(cross(b - a, c - a)); }
bool collinearity(const pdd &p1, const pdd &p2, const pdd &
p3) { return fabs(cross(p1 - p3, p2 - p3)) < eps; }
bool btw(const pdd &p1, const pdd &p2, const pdd &p3) {
if (!collinearity(p1, p2, p3)) return 0;
return dot(p1 - p3, p2 - p3) < eps;
}
bool seg_intersect(const pdd &p1, const pdd &p2, const pdd
&p3, const pdd &p4) {
int a123 = ori(p1, p2, p3);
int a124 = ori(p1, p2, p4);
int a341 = ori(p3, p4, p1);
int a342 = ori(p3, p4, p2);
if (a123 == 0 && a124 == 0) return btw(p1, p2, p3) || btw
(p1, p2, p4) || btw(p3, p4, p1) || btw(p3, p4, p2);
return a123 * a124 <= 0 && a341 * a342 <= 0;
}
pdd intersect(const pdd &p1, const pdd &p2, const pdd &p3,
const pdd &p4) {
double a123 = cross(p2 - p1, p3 - p1);
double a124 = cross(p2 - p1, p4 - p1);
return (p4 * a123 - p3 * a124) / (a123 - a124);
}
pdd perp(const pdd &p1) { return pdd(-p1.Y, p1.X); }
pdd foot(const pdd &p1, const pdd &p2, const pdd &p3) {
return intersect(p1, p2, p3, p3 + perp(p2 - p1)); }

```

8.2 Convex hull*

```

void hull(vector<p11> &dots) {
sort(dots.begin(), dots.end());
vector<p11> ans(1, dots[0]);
for (int ct = 0; ct < 2; ++ct, reverse(ALL(dots)))
for (int i = 1, t = SZ(ans); i < SZ(dots); ans.pb(dots[
i++]))
while (SZ(ans) > t && ori(ans[SZ(ans) - 2], ans.back
(), dots[i]) <= 0) ans.pop_back();
ans.pop_back(), ans.swap(dots);
}

```

8.3 External bisector

```

pdd external_bisector(pdd p1, pdd p2, pdd p3) { // 213
pdd L1 = p2 - p1, L2 = p3 - p1;
L2 = L2 * abs(L1) / abs(L2);
return L1 + L2;
}

```

8.4 Heart

```

pdd excenter(pdd p0, pdd p1, pdd p2, double &radius) {
p1 = p1 - p0, p2 = p2 - p0;
double x1 = p1.X, y1 = p1.Y, x2 = p2.X, y2 = p2.Y;
double m = 2. * (x1 * y2 - y1 * x2);
center.X = (x1 * x1 * y2 - x2 * x2 * y1 + y1 * y2 * (y1 -
y2)) / m;
center.Y = (x1 * x2 * (x2 - x1) - y1 * y1 * x2 + x1 * y2
* y2) / m;
return radius = abs(center), center + p0;
}
pdd incenter(pdd p1, pdd p2, pdd p3, double &radius) {
double a = abs(p2 - p1), b = abs(p3 - p1), c = abs(p3 -
p2);
double s = (a + b + c) / 2, area = sqrt(s * (s - a) * (s
- b) * (s - c));
pdd L1 = external_bisector(p1, p2, p3), L2 =
external_bisector(p2, p1, p3);
return radius = area / s, intersect(p1, p1 + L1, p2, p2 +
L2),
}

```

```

pdd escenter(pdd p1, pdd p2, pdd p3) { // 213
pdd L1 = external_bisector(p1, p2, p3), L2 =
external_bisector(p2, p2 + p2 - p1, p3);
return intersect(p1, p1 + L1, p2, p2 + L2);
}

```

```

pdd barycenter(pdd p1, pdd p2, pdd p3) { return (p1 + p2 +
p3) / 3; }

```

```

pdd orthocenter(pdd p1, pdd p2, pdd p3) {
pdd L1 = p3 - p2, L2 = p3 - p1;
swap(L1.X, L1.Y), L1.X *= -1;
swap(L2.X, L2.Y), L2.X *= -1;
return intersect(p1, p1 + L1, p2, p2 + L2);
}

```

8.5 Minimum Enclosing Circle*

```

pdd Minimum_Enclosing_Circle(vector<pdd> dots, double &r) {
pdd cent;
random_shuffle(ALL(dots));
cent = dots[0], r = 0;
for (int i = 1; i < SZ(dots); ++i)
if (abs(dots[i] - cent) > r) {
cent = dots[i], r = 0;
for (int j = 0; j < i; ++j)
if (abs(dots[j] - cent) > r) {
cent = (dots[i] + dots[j]) / 2;
r = abs(dots[i] - cent);
for (int k = 0; k < j; ++k)
if (abs(dots[k] - cent) > r) cent = excenter(
dots[i], dots[j], dots[k], r);
}
}
return cent;
}

```

8.6 Polar Angle Sort*

```

pdd center; // sort base
int Quadrant(pdd a) {
if (a.X > 0 && a.Y >= 0) return 1;
if (a.X <= 0 && a.Y > 0) return 2;
if (a.X < 0 && a.Y <= 0) return 3;
if (a.X >= 0 && a.Y < 0) return 4;
}
bool cmp(p11 a, p11 b) {
a = a - center, b = b - center;
if (Quadrant(a) != Quadrant(b)) return Quadrant(a) <
Quadrant(b);
if (cross(b, a) == 0) return abs2(a) < abs2(b);
return cross(a, b) > 0;
}
bool cmp(pdd a, pdd b) {
a = a - center, b = b - center;
if (fabs(atan2(a.Y, a.X) - atan2(b.Y, b.X)) > eps) return
atan2(a.Y, a.X) < atan2(b.Y, b.X);
return abs(a) < abs(b);
}

```

8.7 Intersection of two circles*

```

bool CCinter(Cir &a, Cir &b, pdd &p1, pdd &p2) {
pdd o1 = a.o, o2 = b.o;
double r1 = a.R, r2 = b.R, d2 = abs2(o1 - o2), d = sqrt(
d2);
if (d < max(r1, r2) - min(r1, r2) || d > r1 + r2) return
0;
pdd u = (o1 + o2) * 0.5 + (o1 - o2) * ((r2 * r2 - r1 * r1
) / (2 * d2));
double A = sqrt((r1 + r2 + d) * (r1 - r2 + d) * (r1 + r2
- d) * (-r1 + r2 + d));
pdd v = pdd(o1.Y - o2.Y, -o1.X + o2.X) * A / (2 * d2);
p1 = u + v, p2 = u - v;
return 1;
}

```

8.8 Intersection of polygon and circle

```

// Divides into multiple triangle, and sum up
// test by HDU2892
const double PI = acos(-1);
double _area(pdd pa, pdd pb, double r) {
if (abs(pa) < abs(pb)) swap(pa, pb);
if (abs(pb) < eps) return 0;

```



```

    return ata[i] < ata[j];
});
vector<Line> fin;
for (int i = 0; i < sz; ++i)
    if (!i || fabs(ata[ord[i]] - ata[ord[i - 1]]) > eps)
        fin.pb(lines[ord[i]]);
deque<Line> dq;
for (int i = 0; i < SZ(fin); i++) {
    while (SZ(dq) >= 2 && !isin(fin[i], dq[SZ(dq) - 2], dq.
        back())) dq.pop_back();
    while (SZ(dq) >= 2 && !isin(fin[i], dq[0], dq[1])) dq.
        pop_front();
    dq.push_back(fin[i]);
}
while (SZ(dq) >= 3 && !isin(dq[0], dq[SZ(dq) - 2], dq.
    back())) dq.pop_back();
while (SZ(dq) >= 3 && !isin(dq.back(), dq[0], dq[1])) dq.
    pop_front();
vector<Line> res(ALL(dq));
return res;
}

```

8.12 CircleCover*

```

const int N = 1021;
struct CircleCover {
    int C;
    Cir c[N];
    bool g[N][N], overlap[N][N];
    // Area[i] : area covered by at least i circles
    double Area[N];
    void init(int _C) { C = _C; }
    struct Teve {
        pdd p;
        double ang;
        int add;
        Teve() {}
        Teve(pdd _a, double _b, int _c) : p(_a), ang(_b), add(_c) {}
        bool operator<(const Teve &a) const { return ang < a.ang; }
    } eve[N * 2];
    // strict: x = 0, otherwise x = -1
    bool disjunct(Cir &a, Cir &b, int x) { return sign(abs(a.O - b.O) - a.R - b.R) > x; }
    bool contain(Cir &a, Cir &b, int x) { return sign(a.R - b.R - abs(a.O - b.O)) > x; }
    bool contain(int i, int j) {
        /* c[j] is non-strictly in c[i]. */
        return (sign(c[i].R - c[j].R) > 0 || (sign(c[i].R - c[j].R) == 0 && i < j)) && contain(c[i], c[j], -1);
    }
    void solve() {
        fill_n(Area, C + 2, 0);
        for (int i = 0; i < C; ++i)
            for (int j = 0; j < C; ++j) overlap[i][j] = contain(i, j);
        for (int i = 0; i < C; ++i)
            for (int j = 0; j < C; ++j) g[i][j] = !(overlap[i][j] || overlap[j][i] || disjunct(c[i], c[j], -1));
        for (int i = 0; i < C; ++i) {
            int E = 0, cnt = 1;
            for (int j = 0; j < C; ++j)
                if (j != i && overlap[j][i]) ++cnt;
            for (int j = 0; j < C; ++j)
                if (i != j && g[i][j]) {
                    pdd aa, bb;
                    CCinter(c[i], c[j], aa, bb);
                    double A = atan2(aa.Y - c[i].O.Y, aa.X - c[i].O.X);
                    double B = atan2(bb.Y - c[i].O.Y, bb.X - c[i].O.X);
                    eve[E++] = Teve(bb, B, 1), eve[E++] = Teve(aa, A, -1);
                    if (B > A) ++cnt;
                }
            if (E == 0)
                Area[cnt] += pi * c[i].R * c[i].R;
            else {
                sort(eve, eve + E);
                eve[E] = eve[0];
                for (int j = 0; j < E; ++j) {
                    cnt += eve[j].add;
                    Area[cnt] += cross(eve[j].p, eve[j + 1].p) * .5;
                    double theta = eve[j + 1].ang - eve[j].ang;
                    if (theta < 0) theta += 2. * pi;
                }
            }
        }
    }
};

```

8.12 CircleCover*

```

const int N = 1021;
struct CircleCover {
    int C;
    Cir c[N];
    bool g[N][N], overlap[N][N];
    // Area[i] : area covered by at least i circles
    double Area[N];
    void init(int _C) { C = _C; }
    struct Teve {
        pdd p;
        double ang;
        int add;
        Teve() {}
        Teve(pdd _a, double _b, int _c) : p(_a), ang(_b), add(_c) {}
        bool operator<(const Teve &a) const { return ang < a.ang; }
    } eve[N * 2];
    // strict: x = 0, otherwise x = -1
    bool disjunct(Cir &a, Cir &b, int x) { return sign(abs(a.O - b.O) - a.R - b.R) > x; }
    bool contain(Cir &a, Cir &b, int x) { return sign(a.R - b.R - abs(a.O - b.O)) > x; }
    bool contain(int i, int j) {
        /* c[j] is non-strictly in c[i]. */
        return (sign(c[i].R - c[j].R) > 0 || (sign(c[i].R - c[j].R) == 0 && i < j)) && contain(c[i], c[j], -1);
    }
    void solve() {
        fill_n(Area, C + 2, 0);
        for (int i = 0; i < C; ++i)
            for (int j = 0; j < C; ++j) overlap[i][j] = contain(i, j);
        for (int i = 0; i < C; ++i)
            for (int j = 0; j < C; ++j) g[i][j] = !(overlap[i][j] || overlap[j][i] || disjunct(c[i], c[j], -1));
        for (int i = 0; i < C; ++i) {
            int E = 0, cnt = 1;
            for (int j = 0; j < C; ++j)
                if (j != i && overlap[j][i]) ++cnt;
            for (int j = 0; j < C; ++j)
                if (i != j && g[i][j]) {
                    pdd aa, bb;
                    CCinter(c[i], c[j], aa, bb);
                    double A = atan2(aa.Y - c[i].O.Y, aa.X - c[i].O.X);
                    double B = atan2(bb.Y - c[i].O.Y, bb.X - c[i].O.X);
                    eve[E++] = Teve(bb, B, 1), eve[E++] = Teve(aa, A, -1);
                    if (B > A) ++cnt;
                }
            if (E == 0)
                Area[cnt] += pi * c[i].R * c[i].R;
            else {
                sort(eve, eve + E);
                eve[E] = eve[0];
                for (int j = 0; j < E; ++j) {
                    cnt += eve[j].add;
                    Area[cnt] += cross(eve[j].p, eve[j + 1].p) * .5;
                    double theta = eve[j + 1].ang - eve[j].ang;
                    if (theta < 0) theta += 2. * pi;
                }
            }
        }
    }
};

```

8.12 CircleCover*

```

const int N = 1021;
struct CircleCover {
    int C;
    Cir c[N];
    bool g[N][N], overlap[N][N];
    // Area[i] : area covered by at least i circles
    double Area[N];
    void init(int _C) { C = _C; }
    struct Teve {
        pdd p;
        double ang;
        int add;
        Teve() {}
        Teve(pdd _a, double _b, int _c) : p(_a), ang(_b), add(_c) {}
        bool operator<(const Teve &a) const { return ang < a.ang; }
    } eve[N * 2];
    // strict: x = 0, otherwise x = -1
    bool disjunct(Cir &a, Cir &b, int x) { return sign(abs(a.O - b.O) - a.R - b.R) > x; }
    bool contain(Cir &a, Cir &b, int x) { return sign(a.R - b.R - abs(a.O - b.O)) > x; }
    bool contain(int i, int j) {
        /* c[j] is non-strictly in c[i]. */
        return (sign(c[i].R - c[j].R) > 0 || (sign(c[i].R - c[j].R) == 0 && i < j)) && contain(c[i], c[j], -1);
    }
    void solve() {
        fill_n(Area, C + 2, 0);
        for (int i = 0; i < C; ++i)
            for (int j = 0; j < C; ++j) overlap[i][j] = contain(i, j);
        for (int i = 0; i < C; ++i)
            for (int j = 0; j < C; ++j) g[i][j] = !(overlap[i][j] || overlap[j][i] || disjunct(c[i], c[j], -1));
        for (int i = 0; i < C; ++i) {
            int E = 0, cnt = 1;
            for (int j = 0; j < C; ++j)
                if (j != i && overlap[j][i]) ++cnt;
            for (int j = 0; j < C; ++j)
                if (i != j && g[i][j]) {
                    pdd aa, bb;
                    CCinter(c[i], c[j], aa, bb);
                    double A = atan2(aa.Y - c[i].O.Y, aa.X - c[i].O.X);
                    double B = atan2(bb.Y - c[i].O.Y, bb.X - c[i].O.X);
                    eve[E++] = Teve(bb, B, 1), eve[E++] = Teve(aa, A, -1);
                    if (B > A) ++cnt;
                }
            if (E == 0)
                Area[cnt] += pi * c[i].R * c[i].R;
            else {
                sort(eve, eve + E);
                eve[E] = eve[0];
                for (int j = 0; j < E; ++j) {
                    cnt += eve[j].add;
                    Area[cnt] += cross(eve[j].p, eve[j + 1].p) * .5;
                    double theta = eve[j + 1].ang - eve[j].ang;
                    if (theta < 0) theta += 2. * pi;
                }
            }
        }
    }
};

```



```

        Area[cnt] += (theta - sin(theta)) * c[i].R * c[i]
        ].R * .5;
    }
}
};

```

8.13 3Dpoint*

```

struct Point {
    double x, y, z;
    Point(double _x = 0, double _y = 0, double _z = 0) : x(_x), y(_y), z(_z) {}
    Point(pdd p) { x = p.X, y = p.Y, z = abs2(p); }
};

Point operator-(const Point &p1, const Point &p2) { return Point(p1.x - p2.x, p1.y - p2.y, p1.z - p2.z); }
Point cross(const Point &p1, const Point &p2) { return Point(p1.y * p2.z - p1.z * p2.y, p1.z * p2.x - p1.x * p2.z, p1.x * p2.y - p1.y * p2.x); }
double dot(const Point &p1, const Point &p2) { return p1.x * p2.x + p1.y * p2.y + p1.z * p2.z; }
double abs(const Point &a) { return sqrt(dot(a, a)); }
Point cross3(const Point &a, const Point &b, const Point &c) { return cross(b - a, c - a); }
double area(Point a, Point b, Point c) { return abs(cross3(a, b, c)); }
double volume(Point a, Point b, Point c, Point d) { return dot(cross3(a, b, c), d - a); }
pdd proj(Point a, Point b, Point c, Point u) {
    // proj. u to the plane of a, b, and c
    Point e1 = b - a;
    Point e2 = c - a;
    e1 = e1 / abs(e1);
    e2 = e2 - e1 * dot(e2, e1);
    e2 = e2 / abs(e2);
    Point p = u - a;
    return pdd(dot(p, e1), dot(p, e2));
}

```

8.14 Convexhull3D*

```

struct CH3D {
    struct face {
        int a, b, c;
        bool ok;
    } F[8 * N];
    double dblcmp(Point &p, face &f) { return dot(cross3(P[f.a], P[f.b], P[f.c]), p - P[f.a]); }
    int g[N][N], num, n;
    Point P[N];
    void deal(int p, int a, int b) {
        int f = g[a][b];
        face add;
        if (F[f].ok) {
            if (dblcmp(P[p], F[f]) > eps)
                dfs(p, f);
            else
                add.a = b, add.b = a, add.c = p, add.ok = 1, g[p][b] = g[a][p] = g[b][a] = num, F[num++] = add;
        }
    }
    void dfs(int p, int now) {
        F[now].ok = 0;
        deal(p, F[now].b, F[now].a), deal(p, F[now].c, F[now].b), deal(p, F[now].a, F[now].c);
    }
    bool same(int s, int t) {
        Point &a = P[F[s].a];
        Point &b = P[F[s].b];
        Point &c = P[F[s].c];
        return fabs(volume(a, b, c, P[F[t].a])) < eps && fabs(volume(a, b, c, P[F[t].b])) < eps && fabs(volume(a, b, c, P[F[t].c])) < eps;
    }
    void init(int _n) { n = _n, num = 0; }
    void solve() {
        face add;
        num = 0;
        if (n < 4) return;
        if ([&]() {
            for (int i = 1; i < n; ++i)
                if (abs(P[0] - P[i]) > eps) return swap(P[1], P[i]), 0;
            return 1;
        }) ||
        [&]() {
            for (int i = 2; i < n; ++i)
                if (abs(cross3(P[i], P[0], P[1])) > eps) return swap(P[2], P[i]), 0;
            return 1;
        }) ||
        [&]() {
            for (int i = 3; i < n; ++i)
                if (fabs(dot(cross(P[0] - P[1], P[1] - P[2]), P[0] - P[i])) > eps) return swap(P[3], P[i]), 0;
            return 1;
        }) ||
        [&]() {
            for (int i = 0; i < 4; ++i) {
                add.a = (i + 1) % 4, add.b = (i + 2) % 4, add.c = (i + 3) % 4, add.ok = true;
                if (dblcmp(P[i], add) > 0) swap(add.b, add.c);
                g[add.a][add.b] = g[add.b][add.c] = g[add.c][add.a] = num;
                F[num++] = add;
            }
            for (int i = 4; i < n; ++i)
                for (int j = 0; j < num; ++j)
                    if (F[j].ok && dblcmp(P[i], F[j]) > eps) {
                        dfs(i, j);
                        break;
                    }
            for (int tmp = num, i = (num = 0); i < tmp; ++i)
                if (F[i].ok) F[num++] = F[i];
        }
        double get_area() {
            double res = 0.0;
            if (n == 3) return abs(cross3(P[0], P[1], P[2])) / 2.0;
            for (int i = 0; i < num; ++i) res += area(P[F[i].a], P[F[i].b], P[F[i].c]);
            return res / 2.0;
        }
        double get_volume() {
            double res = 0.0;
            for (int i = 0; i < num; ++i) res += volume(Point(0, 0, 0), P[F[i].a], P[F[i].b], P[F[i].c]);
            return fabs(res / 6.0);
        }
    }
    int triangle() { return num; }
    int polygon() {
        int res = 0;
        for (int i = 0, flag = 1; i < num; ++i, res += flag, flag = 1)
            for (int j = 0; j < i && flag; ++j) flag &= !same(i, j);
        return res;
    }
    Point getcent() {
        Point ans(0, 0, 0), temp = P[F[0].a];
        double v = 0.0, t2;
        for (int i = 0; i < num; ++i)
            if (F[i].ok == true) {
                Point p1 = P[F[i].a], p2 = P[F[i].b], p3 = P[F[i].c];
                t2 = volume(temp, p1, p2, p3) / 6.0;
                if (t2 > 0)
                    ans.x += (p1.x + p2.x + p3.x + temp.x) * t2, ans.y += (p1.y + p2.y + p3.y + temp.y) * t2, ans.z += (p1.z + p2.z + p3.z + temp.z) * t2, v += t2;
            }
        ans.x /= (4 * v), ans.y /= (4 * v), ans.z /= (4 * v);
        return ans;
    }
    double pointmindis(Point p) {
        double rt = 999999999;
        for (int i = 0; i < num; ++i)
            if (F[i].ok == true) {
                Point p1 = P[F[i].a], p2 = P[F[i].b], p3 = P[F[i].c];
                double a = (p2.y - p1.y) * (p3.z - p1.z) - (p2.z - p1.z) * (p3.y - p1.y);
                double b = (p2.z - p1.z) * (p3.x - p1.x) - (p2.x - p1.x) * (p3.z - p1.z);
                double c = (p2.x - p1.x) * (p3.y - p1.y) - (p2.y - p1.y) * (p3.x - p1.x);
                double d = 0 - (a * p1.x + b * p1.y + c * p1.z);
                double temp = fabs(a * p.x + b * p.y + c * p.z + d) / sqrt(a * a + b * b + c * c);
            }
    }
}

```

```

}() ||
[&]() {
    for (int i = 2; i < n; ++i)
        if (abs(cross3(P[i], P[0], P[1])) > eps) return swap(P[2], P[i]), 0;
    return 1;
}() ||
[&]() {
    for (int i = 3; i < n; ++i)
        if (fabs(dot(cross(P[0] - P[1], P[1] - P[2]), P[0] - P[i])) > eps) return swap(P[3], P[i]), 0;
    return 1;
}() ||
[&]() {
    for (int i = 0; i < 4; ++i) {
        add.a = (i + 1) % 4, add.b = (i + 2) % 4, add.c = (i + 3) % 4, add.ok = true;
        if (dblcmp(P[i], add) > 0) swap(add.b, add.c);
        g[add.a][add.b] = g[add.b][add.c] = g[add.c][add.a] = num;
        F[num++] = add;
    }
    for (int i = 4; i < n; ++i)
        for (int j = 0; j < num; ++j)
            if (F[j].ok && dblcmp(P[i], F[j]) > eps) {
                dfs(i, j);
                break;
            }
    for (int tmp = num, i = (num = 0); i < tmp; ++i)
        if (F[i].ok) F[num++] = F[i];
}
double get_area() {
    double res = 0.0;
    if (n == 3) return abs(cross3(P[0], P[1], P[2])) / 2.0;
    for (int i = 0; i < num; ++i) res += area(P[F[i].a], P[F[i].b], P[F[i].c]);
    return res / 2.0;
}
double get_volume() {
    double res = 0.0;
    for (int i = 0; i < num; ++i) res += volume(Point(0, 0, 0), P[F[i].a], P[F[i].b], P[F[i].c]);
    return fabs(res / 6.0);
}
int triangle() { return num; }
int polygon() {
    int res = 0;
    for (int i = 0, flag = 1; i < num; ++i, res += flag, flag = 1)
        for (int j = 0; j < i && flag; ++j) flag &= !same(i, j);
    return res;
}
Point getcent() {
    Point ans(0, 0, 0), temp = P[F[0].a];
    double v = 0.0, t2;
    for (int i = 0; i < num; ++i)
        if (F[i].ok == true) {
            Point p1 = P[F[i].a], p2 = P[F[i].b], p3 = P[F[i].c];
            t2 = volume(temp, p1, p2, p3) / 6.0;
            if (t2 > 0)
                ans.x += (p1.x + p2.x + p3.x + temp.x) * t2, ans.y += (p1.y + p2.y + p3.y + temp.y) * t2, ans.z += (p1.z + p2.z + p3.z + temp.z) * t2, v += t2;
        }
    ans.x /= (4 * v), ans.y /= (4 * v), ans.z /= (4 * v);
    return ans;
}
double pointmindis(Point p) {
    double rt = 999999999;
    for (int i = 0; i < num; ++i)
        if (F[i].ok == true) {
            Point p1 = P[F[i].a], p2 = P[F[i].b], p3 = P[F[i].c];
            double a = (p2.y - p1.y) * (p3.z - p1.z) - (p2.z - p1.z) * (p3.y - p1.y);
            double b = (p2.z - p1.z) * (p3.x - p1.x) - (p2.x - p1.x) * (p3.z - p1.z);
            double c = (p2.x - p1.x) * (p3.y - p1.y) - (p2.y - p1.y) * (p3.x - p1.x);
            double d = 0 - (a * p1.x + b * p1.y + c * p1.z);
            double temp = fabs(a * p.x + b * p.y + c * p.z + d) / sqrt(a * a + b * b + c * c);
        }
    return rt;
}

```

```

        rt = min(rt, temp);
    }
    return rt;
}
};

```

8.15 Tangent line of two circles

```

vector<Line> go(const Cir &c1, const Cir &c2, int sign1) {
    // sign1 = 1 for outer tang, -1 for inner tang
    vector<Line> ret;
    double d_sq = norm2(c1.O - c2.O);
    if (d_sq < eps) return ret;
    double d = sqrt(d_sq);
    Pt v = (c2.O - c1.O) / d;
    double c = (c1.R - sign1 * c2.R) / d;
    if (c * c > 1) return ret;
    double h = sqrt(max(0.0, 1.0 - c * c));
    for (int sign2 = 1; sign2 >= -1; sign2 -= 2) {
        Pt n = {v.X * c - sign2 * h * v.Y, v.Y * c + sign2 * h * v.X};
        Pt p1 = c1.O + n * c1.R;
        Pt p2 = c2.O + n * (c2.R * sign1);
        if (fabs(p1.X - p2.X) < eps and fabs(p1.Y - p2.Y) < eps)
            p2 = p1 + perp(c2.O - c1.O);
        ret.push_back({p1, p2});
    }
    return ret;
}

```

8.16 minMaxEnclosingRectangle

```

pdd solve(vector<pll> &dots) {
    vector<pll> hull;
    const double INF = 1e18, qi = acos(-1) / 2 * 3;
    cv.dots = dots;
    hull = cv.hull();
    double Max = 0, Min = INF, deg;
    ll n = hull.size();
    hull.pb(hull[0]);
    for (int i = 0, u = 1, r = 1, l; i < n; ++i) {
        pll nw = hull[i + 1] - hull[i];
        while (cross(nw, hull[u + 1] - hull[i]) > cross(nw, hull[u] - hull[i])) u = (u + 1) % n;
        while (dot(nw, hull[r + 1] - hull[i]) > dot(nw, hull[r] - hull[i])) r = (r + 1) % n;
        if (!i) l = (r + 1) % n;
        while (dot(nw, hull[l + 1] - hull[i]) < dot(nw, hull[l] - hull[i])) l = (l + 1) % n;
        Min = min(Min, (double)(dot(nw, hull[r] - hull[i]) - dot(nw, hull[l] - hull[i])) * cross(nw, hull[u] - hull[i]) / abs2(nw));
        deg = acos((double)dot(hull[r] - hull[l], hull[u] - hull[i]) / abs(hull[r] - hull[l]) / abs(hull[u] - hull[i]));
        deg = (qi - deg) / 2;
        Max = max(Max, (double)abs(hull[r] - hull[l]) * abs(hull[u] - hull[i]) * sin(deg) * sin(deg));
    }
    return pdd(Min, Max);
}

```

8.17 minDistOfTwoConvex

```

// p, q is convex
double TwoConvexHullMinDist(Point P[], Point Q[], int n, int m) {
    int YMinP = 0, YMaxQ = 0;
    double tmp, ans = 999999999;
    for (i = 0; i < n; ++i)
        if (P[i].y < P[YMinP].y) YMinP = i;
    for (i = 0; i < m; ++i)
        if (Q[i].y > Q[YMaxQ].y) YMaxQ = i;
    P[n] = P[0], Q[m] = Q[0];
    for (int i = 0; i < n; ++i) {
        while (tmp = Cross(Q[YMaxQ + 1] - P[YMinP + 1], P[YMinP] - P[YMinP + 1]) > Cross(Q[YMaxQ] - P[YMinP + 1], P[YMinP] - P[YMinP + 1]))
            YMaxQ = (YMaxQ + 1) % m;
        if (tmp < 0)
            ans = min(ans, PointToSegDist(P[YMinP], P[YMinP + 1], Q[YMaxQ]));
        else
            ans = min(ans, TwoSegMinDist(P[YMinP], P[YMinP + 1], Q[YMaxQ], Q[YMaxQ + 1]));
        YMinP = (YMinP + 1) % n;
    }
}

```

```

}
return ans;
}

```

8.18 Minkowski Sum*

```

vector<pll> Minkowski(vector<pll> A, vector<pll> B) {
    hull(A), hull(B);
    vector<pll> C(1, A[0] + B[0]), s1, s2;
    for (int i = 0; i < SZ(A); ++i) s1.pb(A[(i + 1) % SZ(A)] - A[i]);
    for (int i = 0; i < SZ(B); ++i) s2.pb(B[(i + 1) % SZ(B)] - B[i]);
    for (int p1 = 0, p2 = 0; p1 < SZ(A) || p2 < SZ(B);)
        if (p2 >= SZ(B) || (p1 < SZ(A) && cross(s1[p1], s2[p2]) >= 0))
            C.pb(C.back() + s1[p1++]);
        else
            C.pb(C.back() + s2[p2++]);
    return hull(C), C;
}

```

8.19 RotatingSweepLine

```

void rotatingSweepLine(vector<pii> &ps) {
    int n = SZ(ps);
    vector<int> id(n), pos(n);
    vector<pii> line(n * (n - 1) / 2);
    int m = 0;
    for (int i = 0; i < n; ++i)
        for (int j = i + 1; j < n; ++j) line[m++] = pii(i, j);
    sort(ALL(line), [&](const pii &a, const pii &b) -> bool {
        if (ps[a.X].X == ps[a.Y].X) return 0;
        if (ps[b.X].X == ps[b.Y].X) return 1;
        return (double)(ps[a.X].Y - ps[a.Y].Y) / (ps[a.X].X - ps[a.Y].X) < (double)(ps[b.X].Y - ps[b.Y].Y) / (ps[b.X].X - ps[b.Y].X);
    });
    iota(id, id + n, 0);
    sort(ALL(id), [&](const int &a, const int &b) { return ps[a.X] < ps[b.X]; });
    for (int i = 0; i < n; ++i) pos[id[i]] = i;
    for (int i = 0; i < m; ++i) {
        auto l = line[i];
        // meow
        tie(pos[l.X], pos[l.Y], id[pos[l.X]], id[pos[l.Y]]) = make_tuple(pos[l.Y], pos[l.X], l.Y, l.X);
    }
}

```

9 Else

9.1 Mo's Alogrithm(With modification)

```

struct QUERY { // BLOCK=N^{2/3}
    int L, R, id, LBid, RBid, T;
    QUERY(int l, int r, int id, int lb, int rb, int t) : L(l), R(r), id(id), LBid(lb), RBid(rb), T(t) {}
    bool operator<(const QUERY &b) const {
        if (LBid != b.LBid) return LBid < b.LBid;
        if (RBid != b.RBid) return RBid < b.RBid;
        return T < b.T;
    }
};
vector<QUERY> query;
int cur_ans, arr[MAXN], ans[MAXN];
void addTime(int L, int R, int T) {}
void subTime(int L, int R, int T) {}
void add(int x) {}
void sub(int x) {}
void solve() {
    sort(ALL(query));
    int L = 0, R = 0, T = -1;
    for (auto q : query) {
        while (T < q.T) addTime(L, R, ++T);
        while (T > q.T) subTime(L, R, T--);
        while (R < q.R) add(arr[++R]);
        while (L > q.L) add(arr[--L]);
        while (R > q.R) sub(arr[R--]);
        while (L < q.L) sub(arr[L++]);
        ans[q.id] = cur_ans;
    }
}

```

9.2 Mo's Alogrithm On Tree

```

const int MAXN = 40005;
vector<int> G[MAXN]; // 1-base
int n, B, arr[MAXN], ans[100005], cur_ans;
int in[MAXN], out[MAXN], dfn[MAXN * 2], dft;
int deep[MAXN], sp[__lg(MAXN * 2) + 1][MAXN * 2], bln[MAXN], spt;
bitset<MAXN> inset;
struct QUERY {
    int L, R, Lid, id, lca;
    QUERY(int l, int r, int _id) : L(l), R(r), lca(0), id(_id) {}
    bool operator<(const QUERY &b) {
        if (Lid != b.Lid) return Lid < b.Lid;
        return R < b.R;
    }
};
vector<QUERY> query;
void dfs(int u, int f, int d) {
    deep[u] = d, sp[0][spt] = u, bln[u] = spt++;
    dfn[dfn] = u, in[u] = dft++;
    for (int v : G[u])
        if (v != f) dfs(v, u, d + 1), sp[0][spt] = u, bln[u] = spt++;
    dfn[dfn] = u, out[u] = dft++;
}
int lca(int u, int v) {
    if (bln[u] > bln[v]) swap(u, v);
    int t = __lg(bln[v] - bln[u] + 1);
    int a = sp[t][bln[u]], b = sp[t][bln[v] - (1 << t) + 1];
    if (deep[a] < deep[b]) return a;
    return b;
}
void sub(int x) {}
void add(int x) {}
void flip(int x) {
    if (inset[x])
        sub(arr[x]);
    else
        add(arr[x]);
    inset[x] = ~inset[x];
}
void solve() {
    B = sqrt(2 * n), dft = spt = cur_ans = 0, dfs(1, 1, 0);
    for (int i = 1, x = 2; x < 2 * n; ++i, x <= 1)
        for (int j = 0; j + x <= 2 * n; ++j)
            if (deep[sp[i - 1][j]] < deep[sp[i - 1][j + x / 2]])
                sp[i][j] = sp[i - 1][j];
            else
                sp[i][j] = sp[i - 1][j + x / 2];
    for (auto &q : query) {
        int c = lca(q.L, q.R);
        if (c == q.L || c == q.R)
            q.L = out[c == q.L ? q.R : q.L], q.R = out[c];
        else if (out[q.L] < in[q.R])
            q.lca = c, q.L = out[q.L], q.R = in[q.R];
        else
            q.lca = c, c = in[q.L], q.L = out[q.R], q.R = c;
        q.Lid = q.L / B;
    }
    sort(ALL(query));
    int L = 0, R = -1;
    for (auto q : query) {
        while (R < q.R) flip(dfn[++R]);
        while (L > q.L) flip(dfn[--L]);
        while (R > q.R) flip(dfn[R--]);
        while (L < q.L) flip(dfn[L++]);
        if (q.lca) add(arr[q.lca]);
        ans[q.id] = cur_ans;
        if (q.lca) sub(arr[q.lca]);
    }
}

```

9.3 DynamicConvexTrick*

```

// only works for integer coordinates!!
struct Line {
    mutable ll a, b, p;
    bool operator<(const Line &rhs) const { return a < rhs.a; }
    bool operator<(ll x) const { return p < x; }
};
struct DynamicHull : multiset<Line, less<>> {
    static const ll kInf = 1e18;
    ll Div(ll a, ll b) { return a / b - ((a ^ b) < 0 && a % b); }
    bool isect(iterator x, iterator y) {

```

```

        if (y == end()) {
            x->p = kInf;
            return 0;
        }
        if (x->a == y->a)
            x->p = x->b > y->b ? kInf : -kInf;
        else
            x->p = Div(y->b - x->b, x->a - y->a);
        return x->p >= y->p;
    }
    void addline(ll a, ll b) {
        auto z = insert({a, b, 0}), y = z++, x = y;
        while (isect(y, z)) z = erase(z);
        if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
        while ((y = x) != begin() && (--x)->p >= y->p) isect(x, erase(y));
    }
    ll query(ll x) {
        auto l = *lower_bound(x);
        return l.a * x + l.b;
    }
};

```

9.4 DLX*

```

#define TRAV(i, link, start) for (int i = link[start]; i != start; i = link[i])
template<bool A, bool B = !A> // A: Exact
struct DLX {
    int lt[NN], rg[NN], up[NN], dn[NN], cl[NN], rw[NN], bt[NN], s[NN], head, sz, ans;
    int columns;
    bool vis[NN];
    void remove(int c) {
        if (A) lt[rg[c]] = lt[c], rg[lt[c]] = rg[c];
        TRAV(i, dn, c) {
            if (A) {
                TRAV(j, rg, i)
                up[dn[j]] = up[j], dn[up[j]] = dn[j], --s[cl[j]];
            } else {
                lt[rg[i]] = lt[i], rg[lt[i]] = rg[i];
            }
        }
    }
    void restore(int c) {
        TRAV(i, up, c) {
            if (A) {
                TRAV(j, lt, i)
                ++s[cl[j]], up[dn[j]] = j, dn[up[j]] = j;
            } else {
                lt[rg[i]] = rg[lt[i]] = i;
            }
        }
        if (A) lt[rg[c]] = c, rg[lt[c]] = c;
    }
    void init(int c) {
        columns = c;
        for (int i = 0; i < c; ++i) {
            up[i] = dn[i] = bt[i] = i;
            lt[i] = i == 0 ? c : i - 1;
            rg[i] = i == c - 1 ? c : i + 1;
            s[i] = 0;
        }
        rg[c] = 0, lt[c] = c - 1;
        up[c] = dn[c] = -1;
        head = c, sz = c + 1;
    }
    void insert(int r, const vector<int> &col) {
        if (col.empty()) return;
        int f = sz;
        for (int i = 0; i < (int)col.size(); ++i) {
            int c = col[i], v = sz++;
            dn[bt[c]] = v;
            up[v] = bt[c], bt[c] = v;
            rg[v] = (i + 1 == (int)col.size()) ? f : v + 1;
            rw[v] = r, cl[v] = c;
            ++s[c];
            if (i > 0) lt[v] = v - 1;
        }
        lt[f] = sz - 1;
    }
    int h() {
        int ret = 0;
        memset(vis, 0, sizeof(bool) * sz);
        TRAV(x, rg, head) {

```

```

    if (vis[x]) continue;
    vis[x] = true, ++ret;
    TRAV(i, dn, x) TRAV(j, rg, i) vis[c1[j]] = true;
}
return ret;
}
void dfs(int dep) {
    if (dep + (A ? 0 : h()) >= ans) return;
    if (rg[head] == head) return ans = dep, void();
    if (dn[rg[head]] == rg[head]) return;
    int w = rg[head];
    TRAV(x, rg, head) if (s[x] < s[w]) w = x;
    if (A) remove(w);
    TRAV(i, dn, w) {
        if (B) remove(i);
        TRAV(j, rg, i) remove(A ? c1[j] : j);
        dfs(dep + 1);
        TRAV(j, lt, i) restore(A ? c1[j] : j);
        if (B) restore(i);
    }
    if (A) restore(w);
}
int solve() {
    for (int i = 0; i < columns; ++i) dn[bt[i]] = i, up[i] = bt[i];
    ans = 1e9, dfs(0);
    return ans;
}
};

```

```

    else
        t[id++] = p[j++];
}
for (int i = 1; i <= r; i++) p[i] = t[i];
vector<pair<double, double>> v;
for (int i = 1; i <= r; i++)
    if (abs(p[i].first - x) < d) v.push_back(p[i]);
for (int i = 0; i < v.size(); i++) {
    for (int j = i + 1; j < v.size(); j++) {
        if (v[j].second - v[i].second >= d) break;
        d = min(d, sqrt((v[i].first - v[j].first) * (v[i].first - v[j].first) + (v[i].second - v[j].second) * (v[i].second - v[j].second)));
    }
}
return d;
}
main() {
    sort(p + 1, p + n + 1);
    solve(1, n);
}

```

9.5 Matroid Intersection

Start from $S = \emptyset$. In each iteration, let

- $Y_1 = \{x \notin S \mid S \cup \{x\} \in I_1\}$
- $Y_2 = \{x \notin S \mid S \cup \{x\} \in I_2\}$

If there exists $x \in Y_1 \cap Y_2$, insert x into S . Otherwise for each $x \in S, y \notin S$, create edges

- $x \rightarrow y$ if $S - \{x\} \cup \{y\} \in I_1$.
- $y \rightarrow x$ if $S - \{x\} \cup \{y\} \in I_2$.

Find a shortest path (with BFS) starting from a vertex in Y_1 and ending at a vertex in Y_2 which doesn't pass through any other vertices in Y_2 , and alternate the path. The size of S will be incremented by 1 in each iteration. For the weighted case, assign weight $w(x)$ to vertex x if $x \in S$ and $-w(x)$ if $x \notin S$. Find the path with the minimum number of edges among all minimum length paths and alternate it.

9.6 AdaptiveSimpson

```

using F_t = function<double(double)>;
pdd simpson(const F_t &f, double l, double r, double fl,
    double fr, double fm = nan("")) {
    if (isnan(fm)) fm = f((l + r) / 2);
    return {fm, (r - l) / 6 * (fl + 4 * fm + fr)};
}
double simpson_ada(const F_t &f, double l, double r, double fl, double fm, double fr, double eps) {
    double m = (l + r) / 2, s = simpson(f, l, r, fl, fr, fm).second;
    auto [flm, sl] = simpson(f, l, m, fl, fm);
    auto [fmr, sr] = simpson(f, m, r, fm, fr);
    double delta = sl + sr - s;
    if (abs(delta) <= 15 * eps) return sl + sr + delta / 15;
    return simpson_ada(f, l, m, fl, flm, fm, eps / 2) +
        simpson_ada(f, m, r, fm, fmr, fr, eps / 2);
}
double simpson_ada(const F_t &f, double l, double r) {
    return simpson_ada(f, l, r, f(l), f((l + r) / 2), f(r),
        1e-9 / 7122);
}
double simpson_ada2(const F_t &f, double l, double r) {
    double h = (r - l) / 7122, s = 0;
    for (int i = 0; i < 7122; ++i, l += h) s += simpson_ada(f,
        l, l + h);
    return s;
}

```

9.7 Closest Pair

```

pair<double, double> p[50007], t[50007];
double solve(int l, int r) {
    if (l == r) return INF;
    int mid = (l + r) >> 1;
    double x = p[mid].first;
    double d = min(solve(l, mid), solve(mid + 1, r));
    int i = l, j = mid + 1, id = l;
    while (i <= mid || j <= r) {
        if (i <= mid && (j > r || p[i].second < p[j].second))
            t[id++] = p[i++];
    }
}

```