# Contents

1 Reminder 1
1.1 Bug List 1
1.2 OwO 1

2 Basic 1
2.1 Vimrc 1
2.2 Runcpp.sh 2
2.3 Stress 2
2.4 PBDS 2
2.5 Random 2

3 Data Structure 2
3.1 BIT 2
3.2 DSU 2
3.3 Segment Tree 2
3.4 Treap 3
3.5 Persistent Treap 3
3.6 Li Chao Tree 3
3.7 Sparse Table 3
3.8 Time Segment Tree 4

4 Flow / Matching 4
4.1 Dinic 4
4.2 MCMF 5
4.3 KM 5
4.4 Hopcroft-Karp 6
4.5 Blossom 6
4.6 Weighted Blossom 6
4.7 Cover / Independent Set 8

5 Graph 8
5.1 Heavy-Light Decomposition 8
5.2 Centroid Decomposition 9
5.3 Bellman-Ford + SPFA 9
5.4 BCC - AP 10
5.5 BCC - Bridge 10
5.6 SCC - Tarjan 11
5.7 SCC - Kosaraju 11
5.8 Eulerian Path - Undir 12
5.9 Eulerian Path - Dir 12
5.10 Hamilton Path 12
5.11 Kth Shortest Path 12
5.12 System of Difference Constraints 13

6 String 13
6.1 Aho Corasick 13
6.2 KMP 14
6.3 Z Value 14
6.4 Manacher 14
6.5 Suffix Array 14
6.6 Minimum Rotation 15
6.7 Lyndon Factorization 15
6.8 Rolling Hash 15
6.9 Trie 15

7 Geometry 15
7.1 Basic Operations 15
7.2 SVG Writer 15
7.3 Sort by Angle 15
7.4 Line Intersection 16
7.5 Polygon Area 16
7.6 Convex Hull 16
7.7 Point In Convex 16
7.8 Point Segment Distance 16
7.9 Lower Concave Hull 16
7.10 Pick's Theorem 16
7.11 Vector In Polygon 17
7.12 Minkowski Sum 17
7.13 Rotating SweepLine 17
7.14 Half Plane Intersection 17
7.15 Minimum Enclosing Circle 18
7.16 Heart 18
7.17 Tangents 18
7.18 Point In Circle 18
7.19 Union of Circles 18
7.20 Union of Polygons 18
7.21 Delaunay Triangulation 18
7.22 Triangulation Vonoroi 18
7.23 External Bisector 18
7.24 Intersection Area of Polygon and Circle 18
7.25 3D Point 18
7.26 3D Convex Hull 18

8 Number Theory 18
8.1 FFT 18
8.2 Pollard's rho 19
8.3 Miller Rabin 19
8.4 Fast Power 20
8.5 Extend GCD 20
8.6 Mu + Phi 20
8.7 Other Formulas 20
8.8 Polynomial 20

9 Linear Algebra 22
9.1 Gaussian-Jordan Elimination 22
9.2 Determinant 22

10 Combinatorics 22
10.1 Catalan Number 22
10.2 Burnside's Lemma 22

11 Special Numbers 22
11.1 Fibonacci Series 22
11.2 Prime Numbers 22

# 1 Reminder

## 1.1 Bug List

- 沒開 long long
- 陣列戳出界／開不夠大/ 開太大本地 compile 噴怪 error
- 傳之前先確定選對檔案
- 寫好的函式忘記呼叫
- 變數打錯
- 0-base / 1-base
- 忘記初始化
- == 打成 =
- <= 打成 <+
- dp[i] 從 dp[i-1] 轉移時忘記特判 i > 0
- std::sort 比較運算子寫成 < 或是讓 = 的情況為 true
- 漏 case / 分 case 要好好想
- 線段樹改值懶標初始值不能設為 0
- DFS 的時候不小心覆寫到全域變數
- 浮點數誤差
- 多筆測資不能沒讀完直接 return
- 記得刪 cerr

## 1.2 OwO

- 可以構造複雜點的測資幫助思考
- 真的卡太久請跳題
- Enjoy The Contest!

# 2 Basic

## 2.1 Vimrc

```
set number relativenumber ai t_Co=256 tabstop=4
set mouse=a shiftwidth=4 encoding=utf8
set bs=2 ruler laststatus=2 cmdheight=2
set clipboard=unnamedplus showcmd autoread
set belloff=all
filetype indent on
"set guifont Hack:h16
":set guifont?

inoremap ( ()<Esc>i
inoremap " ""<Esc>i
inoremap [ []<Esc>i
inoremap ' ''<Esc>i
inoremap { {<CR>}<Esc>ko

vmap <C-c> "+y
inoremap <C-v> <Esc>p
nnoremap <C-v> p

nnoremap <tab> gt
nnoremap <S-tab> gT
inoremap <C-n> <Esc>:tabnew<CR>
nnoremap <C-n> :tabnew<CR>

inoremap <F9> <Esc>:w<CR>:!~/runcpp.sh %:p:t %:p:h<CR>
nnoremap <F9> :w<CR>:!~/runcpp.sh %:p:t %:p:h<CR>

syntax on
colorscheme desert
set filetype=cpp
set background=dark
hi Normal ctermfg=white ctermbg=black
```

## 2.2 Runcpp.sh

```bash
#! /bin/bash
clear
echo "Start compiling $1..."
echo
g++ -O2 -std=c++20 -Wall -Wextra -Wshadow $2/$1 -o $2/out
if [ "$?" -ne 0 ]
then
    exit 1
fi
echo
echo "Done compiling"
echo "=========================="
echo
echo "Input file:"
echo
cat $2/in.txt
echo
echo "=========================="
echo
declare startTime=`date +%s%N`
$2/out < $2/in.txt > $2/out.txt
declare endTime=`date +%s%N`
delta=`expr $endTime - $startTime`
delta=`expr $delta / 1000000`
cat $2/out.txt
echo
echo "time: $delta ms"
```

## 2.3 Stress

```bash
g++ gen.cpp -o gen.out
g++ ac.cpp -o ac.out
g++ wa.cpp -o wa.out
for ((i=0;;i++))
do
    echo "$i"
    ./gen.out > in.txt
    ./ac.out < in.txt > ac.txt
    ./wa.out < in.txt > wa.txt
    diff ac.txt wa.txt || break
done
```

## 2.4 PBDS

```cpp
#include <bits/extc++.h>
using namespace __gnu_pbds;

// map
tree<int, int, less<>, rb_tree_tag, tree_order_statistics_node_update> tr;
tr.order_of_key(element);
tr.find_by_order(rank);

// set
tree<int, null_type, less<>, rb_tree_tag, tree_order_statistics_node_update> tr;
tr.order_of_key(element);
tr.find_by_order(rank);

// hash table
gp_hash_table<int, int> ht;
ht.find(element);
ht.insert({key, value});
ht.erase(element);

// priority queue
__gnu_pbds::priority_queue<int, less<int>> big_q; // Big First
__gnu_pbds::priority_queue<int, greater<int>> small_q; // Small First
q1.join(q2); // join
```

## 2.5 Random

```cpp
mt19937 gen(chrono::steady_clock::now().time_since_epoch().count());
uniform_int_distribution<int> dis(1, 100);
cout << dis(gen) << endl;
shuffle(v.begin(), v.end(), gen);
```

# 3 Data Structure

## 3.1 BIT

```cpp
struct BIT {
    int n;
    long long bit[N];

    void init(int x, vector<long long> &a) {
        n = x;
        for (int i = 1, j; i <= n; i++) {
            bit[i] += a[i - 1], j = i + (i & -i);
            if (j <= n) bit[j] += bit[i];
        }
    }

    void update(int x, long long dif) {
        while (x <= n) bit[x] += dif, x += x & -x;
    }

    long long query(int l, int r) {
        if (l != 1) return query(1, r) - query(1, l - 1);

        long long ret = 0;
        while (l <= r) ret += bit[r], r -= r & -r;
        return ret;
    }
} bm;
```

## 3.2 DSU

```cpp
struct DSU {
    int h[N], s[N];

    void init(int n) { iota(h, h + n + 1, 0), fill(s, s + n + 1, 1); }

    int fh(int x) { return (h[x] == x ? x : h[x] = fh(h[x])); }

    bool mer(int x, int y) {
        x = fh(x), y = fh(y);
        if (x == y) return 0;
        if (s[x] < s[y]) swap(x, y);
        s[x] += s[y], s[y] = 0;
        h[y] = x;
        return 1;
    }
} bm;
```

## 3.3 Segment Tree

```cpp
struct segtree {
    int n, seg[1 << 19];

    void init(int x) {
        n = 1 << (__lg(x) + 1);
        for (int i = 1; i < 2 * n; i++)
            seg[i] = inf;
    }

    void update(int x, int val) {
        x += n;
        seg[x] = val, x /= 2;
        while (x)
            seg[x] = min(seg[2 * x], seg[2 * x + 1]), x /= 2;
    }

    int query(int l, int r) {
        l += n, r += n;
        int ret = inf;
        while (l < r) {
            if (l & 1)
                ret = min(ret, seg[l++]);
            if (r & 1)
                ret = min(ret, seg[--r]);
            l /= 2, r /= 2;
        }
        return ret;
    }
} bm;
```

## 3.4 Treap

```
mt19937 rng(random_device{}());
struct Treap {
    Treap *l, *r;
    int val, num, pri;
    Treap(int k) {
        l = r = NULL;
        val = k;
        num = 1;
        pri = rng();
    }
};
int siz(Treap *now) { return now ? now->num : 0; }
void pull(Treap *&now) {
    now->num = siz(now->l) + siz(now->r) + 1;
}
Treap *merge(Treap *a, Treap *b) {
    if (!a || !b)
        return a ? a : b;
    else if (a->pri > b->pri) {
        a->r = merge(a->r, b);
        pull(a);
        return a;
    } else {
        b->l = merge(a, b->l);
        pull(b);
        return b;
    }
}
void split_size(Treap *rt, Treap *&a, Treap *&b, int val) {
    if (!rt) {
        a = b = NULL;
        return;
    }
    if (siz(rt->l) + 1 > val) {
        b = rt;
        split_size(rt->l, a, b->l, val);
        pull(b);
    } else {
        a = rt;
        split_size(rt->r, a->r, b, val - siz(a->l) - 1);
        pull(a);
    }
}
void split_val(Treap *rt, Treap *&a, Treap *&b, int val) {
    if (!rt) {
        a = b = NULL;
        return;
    }
    if (rt->val <= val) {
        a = rt;
        split_val(rt->r, a->r, b, val);
        pull(a);
    } else {
        b = rt;
        split_val(rt->l, a, b->l, val);
        pull(b);
    }
}
void treap_dfs(Treap *now) {
    if (!now) return;
    treap_dfs(now->l);
    cout << now->val << " ";
    treap_dfs(now->r);
}
```

## 3.5 Persistent Treap

```
struct node {
    node *l, *r;
    char c;
    int v, sz;
    node(char x = '$') : c(x), v(mt()), sz(1) {
        l = r = nullptr;
    }
    node(node* p) { *this = *p; }
    void pull() {
        sz = 1;
        for (auto i : {l, r})
            if (i) sz += i->sz;
    }
} arr[maxn], *ptr = arr;
inline int size(node* p) { return p ? p->sz : 0; }
node* merge(node* a, node* b) {
    if (!a || !b) return a ?: b;
    if (a->v < b->v) {
        node* ret = new (ptr++) node(a);
        ret->r = merge(ret->r, b), ret->pull();
        return ret;
    } else {
        node* ret = new (ptr++) node(b);
        ret->l = merge(a, ret->l), ret->pull();
        return ret;
    }
}
P<node*> split(node* p, int k) {
    if (!p) return {nullptr, nullptr};
    if (k >= size(p->l) + 1) {
        auto [a, b] = split(p->r, k - size(p->l) - 1);
        node* ret = new (ptr++) node(p);
        ret->r = a, ret->pull();
        return {ret, b};
    } else {
        auto [a, b] = split(p->l, k);
        node* ret = new (ptr++) node(p);
        ret->l = b, ret->pull();
        return {a, ret};
    }
}
```

## 3.6 Li Chao Tree

```
constexpr int maxn = 5e4 + 5;
struct line {
    ld a, b;
    ld operator()(ld x) { return a * x + b; }
} arr[(maxn + 1) << 2];
bool operator<(line a, line b) { return a.a < b.a; }
#define m ((l + r) >> 1)
void insert(line x, int i = 1, int l = 0, int r = maxn) {
    if (r - l == 1) {
        if (x(l) > arr[i](l))
            arr[i] = x;
        return;
    }
    line a = max(arr[i], x), b = min(arr[i], x);
    if (a(m) > b(m))
        arr[i] = a, insert(b, i << 1, l, m);
    else
        arr[i] = b, insert(a, i << 1 | 1, m, r);
}
ld query(int x, int i = 1, int l = 0, int r = maxn) {
    if (x < l || r <= x) return -numeric_limits<ld>::max();
    if (r - l == 1) return arr[i](x);
    return max({arr[i](x), query(x, i << 1, l, m), query(x, i << 1 | 1, m, r)});
}
#undef m
```

## 3.7 Sparse Table

```
const int lgmx = 19;

int n, q;
int spt[lgmx][maxn];

void build() {
    FOR(k, 1, lgmx, 1) {
        for (int i = 0; i + (1 << k) - 1 < n; i++) {
            spt[k][i] = min(spt[k - 1][i], spt[k - 1][i + (1 << (k - 1))]);
        }
    }
}

int query(int l, int r) {
    int ln = len(l, r);
    int lg = __lg(ln);
    return min(spt[lg][l], spt[lg][r - (1 << lg) + 1]);
}
```

## 3.8 Time Segment Tree

```
constexpr int maxn = 1e5 + 5;
V<P<int>> arr[(maxn + 1) << 2];
V<int> dsu, sz;
V<tuple<int, int, int>> his;
int cnt, q;
int find(int x) {
    return x == dsu[x] ? x : find(dsu[x]);
};
inline bool merge(int x, int y) {
    int a = find(x), b = find(y);
    if (a == b) return false;
    if (sz[a] > sz[b]) swap(a, b);
    his.emplace_back(a, b, sz[b]), dsu[a] = b, sz[b] += sz[a];
    return true;
};
inline void undo() {
    auto [a, b, s] = his.back();
    his.pop_back();
    dsu[a] = a, sz[b] = s;
}
#define m ((l + r) >> 1)
void insert(int ql, int qr, P<int> x, int i = 1, int l = 0, int r = q) {
    // debug(ql, qr, x); return;
    if (qr <= l || r <= ql) return;
    if (ql <= l && r <= qr) {
        arr[i].push_back(x);
        return;
    }
    if (qr <= m)
        insert(ql, qr, x, i << 1, l, m);
    else if (m <= ql)
        insert(ql, qr, x, i << 1 | 1, m, r);
    else {
        insert(ql, qr, x, i << 1, l, m);
        insert(ql, qr, x, i << 1 | 1, m, r);
    }
}
void traversal(V<int>& ans, int i = 1, int l = 0, int r = q) {
    int opcnt = 0;
    // debug(i, l, r);
    for (auto [a, b] : arr[i])
        if (merge(a, b))
            opcnt++, cnt--;
    if (r - l == 1)
        ans[l] = cnt;
    else {
        traversal(ans, i << 1, l, m);
        traversal(ans, i << 1 | 1, m, r);
    }
    while (opcnt--)
        undo(), cnt++;
    arr[i].clear();
}
#undef m
inline void solve() {
    int n, m;
    cin >> n >> m >> q, q++;
    dsu.resize(cnt = n), sz.assign(n, 1);
    iota(dsu.begin(), dsu.end(), 0);
    // a, b, time, operation
    unordered_map<ll, V<int>> s;
    for (int i = 0; i < m; i++) {
        int a, b;
        cin >> a >> b;
        if (a > b) swap(a, b);
        s[((ll)a << 32) | b].emplace_back(0);
    }
    for (int i = 1; i < q; i++) {
        int op, a, b;
        cin >> op >> a >> b;
        if (a > b) swap(a, b);
        switch (op) {
            case 1:
                s[((ll)a << 32) | b].push_back(i);
                break;
            case 2:
                auto tmp = s[((ll)a << 32) | b].back();
                s[((ll)a << 32) | b].pop_back();
                insert(tmp, i, P<int>{a, b});
        }
    }
    for (auto [p, v] : s) {
        int a = p >> 32, b = p & -1;
        while (v.size()) {
            insert(v.back(), q, P<int>{a, b});
            v.pop_back();
        }
    }
    V<int> ans(q);
    traversal(ans);
    for (auto i : ans)
        cout << i << ' ';
    cout << endl;
}
```

# 4 Flow / Matching

## 4.1 Dinic

```
struct Dinic {
    int n, s, t, level[N], iter[N];
    struct edge {
        int to, cap, rev;
    };
    vector<edge> path[N];
    void init(int _n, int _s, int _t) {
        n = _n, s = _s, t = _t;
        FOR(i, 0, n + 1)
        path[i].clear();
    }
    void add(int a, int b, int c) {
        edge now;
        now.to = b, now.cap = c, now.rev = sz(path[b]);
        path[a].pb(now);
        now.to = a, now.cap = 0, now.rev = sz(path[a]) - 1;
        path[b].pb(now);
    }
    void bfs() {
        memset(level, -1, sizeof(level));
        level[s] = 0;
        queue<int> q;
        q.push(s);
        while (q.size()) {
            int now = q.front();
            q.pop();
            for (edge e : path[now]) {
                if (e.cap > 0 && level[e.to] == -1) {
                    level[e.to] = level[now] + 1;
                    q.push(e.to);
                }
            }
        }
    }
    int dfs(int now, int flow) {
        if (now == t) return flow;
        for (int &i = iter[now]; i < sz(path[now]); i++) {
            edge &e = path[now][i];
            if (e.cap > 0 && level[e.to] == level[now] + 1) {
                int res = dfs(e.to, min(flow, e.cap));
                if (res > 0) {
                    e.cap -= res;
                    path[e.to][e.rev].cap += res;
                    return res;
                }
            }
        }
        return 0;
    }
    int dinic() {
        int res = 0;
        while (true) {
            bfs();
            if (level[t] == -1) break;
            memset(iter, 0, sizeof(iter));
            int now = 0;
            while ((now = dfs(s, INF)) > 0) res += now;
        }
        return res;
    }
};
```

## 4.2 MCMF

```
struct MCMF {
    int n, s, t, par[N + 5], p_i[N + 5], dis[N + 5], vis[N + 5];
    struct edge {
        int to, cap, rev, cost;
    };
    vector<edge> path[N];
    void init(int _n, int _s, int _t) {
        n = _n, s = _s, t = _t;
        FOR(i, 0, 2 * n + 5)
        par[i] = p_i[i] = vis[i] = 0;
    }
    void add(int a, int b, int c, int d) {
        path[a].pb({b, c, sz(path[b]), d});
        path[b].pb({a, 0, sz(path[a]) - 1, -d});
    }
    void spfa() {
        FOR(i, 0, n * 2 + 5)
        dis[i] = INF,
        vis[i] = 0;
        dis[s] = 0;
        queue<int> q;
        q.push(s);
        while (!q.empty()) {
            int now = q.front();
            q.pop();
            vis[now] = 0;
            for (int i = 0; i < sz(path[now]); i++) {
                edge e = path[now][i];
                if (e.cap > 0 && dis[e.to] > dis[now] + e.cost) {
                    dis[e.to] = dis[now] + e.cost;
                    par[e.to] = now;
                    p_i[e.to] = i;
                    if (vis[e.to] == 0) {
                        vis[e.to] = 1;
                        q.push(e.to);
                    }
                }
            }
        }
    }
    pii flow() {
        int flow = 0, cost = 0;
        while (true) {
            spfa();
            if (dis[t] == INF)
                break;
            int mn = INF;
            for (int i = t; i != s; i = par[i])
                mn = min(mn, path[par[i]][p_i[i]].cap);
            flow += mn;
            cost += dis[t] * mn;
            for (int i = t; i != s; i = par[i]) {
                edge &now = path[par[i]][p_i[i]];
                now.cap -= mn;
                path[i][now.rev].cap += mn;
            }
        }
        return mp(flow, cost);
    }
};
```

## 4.3 KM

```
struct KM {
    int n, mx[1005], my[1005], pa[1005];
    int g[1005][1005], lx[1005], ly[1005], sy[1005];
    bool vx[1005], vy[1005];
    void init(int _n) {
        n = _n;
        FOR(i, 1, n + 1)
        fill(g[i], g[i] + 1 + n, 0);
    }
    void add(int a, int b, int c) { g[a][b] = c; }
    void augment(int y) {
        for (int x, z; y; y = z)
            x = pa[y], z = mx[x], my[y] = x, mx[x] = y;
    }
    void bfs(int st) {
        FOR(i, 1, n + 1)
        sy[i] = INF,
        vx[i] = vy[i] = 0;
        queue<int> q;
        q.push(st);
        for (;;) {
            while (!q.empty()) {
                int x = q.front();
                q.pop();
                vx[x] = 1;
                FOR(y, 1, n + 1)
                if (!vy[y]) {
                    int t = lx[x] + ly[y] - g[x][y];
                    if (t == 0) {
                        pa[y] = x;
                        if (!my[y]) {
                            augment(y);
                            return;
                        }
                        vy[y] = 1, q.push(my[y]);
                    } else if (sy[y] > t)
                        pa[y] = x, sy[y] = t;
                }
            }
            int cut = INF;
            FOR(y, 1, n + 1)
            if (!vy[y] && cut > sy[y]) cut = sy[y];
            FOR(j, 1, n + 1) {
                if (vx[j]) lx[j] -= cut;
                if (vy[j])
                    ly[j] += cut;
                else
                    sy[j] -= cut;
            }
            FOR(y, 1, n + 1) {
                if (!vy[y] && sy[y] == 0) {
                    if (!my[y]) {
                        augment(y);
                        return;
                    }
                    vy[y] = 1;
                    q.push(my[y]);
                }
            }
        }
    }
    int solve() {
        fill(mx, mx + n + 1, 0);
        fill(my, my + n + 1, 0);
        fill(ly, ly + n + 1, 0);
        fill(lx, lx + n + 1, 0);
        FOR(x, 1, n + 1)
        FOR(y, 1, n + 1)
        lx[x] = max(lx[x], g[x][y]);
        FOR(x, 1, n + 1)
        bfs(x);
        int ans = 0;
        FOR(y, 1, n + 1)
        ans += g[my[y]][y];
        return ans;
    }
};
```

## 4.4 Hopcroft-Karp

```
struct HopcroftKarp {
    // id: X = [1, nx], Y = [nx+1, nx+ny]
    int n, nx, ny, m, MXCNT;
    vector<vector<int> > g;
    vector<int> mx, my, dis, vis;
    void init(int nnx, int nny, int mm) {
        nx = nnx, ny = nny, m = mm;
        n = nx + ny + 1;
        g.clear();
        g.resize(n);
    }
    void add(int x, int y) {
        g[x].emplace_back(y);
        g[y].emplace_back(x);
    }
    bool dfs(int x) {
        vis[x] = true;
        Each(y, g[x]) {
            int px = my[y];
            if (px == -1 ||
                (dis[px] == dis[x] + 1 &&
                 !vis[px] && dfs(px))) {
                mx[x] = y;
                my[y] = x;
                return true;
            }
        }
        return false;
    }
    void get() {
        mx.clear();
        mx.resize(n, -1);
        my.clear();
        my.resize(n, -1);

        while (true) {
            queue<int> q;
            dis.clear();
            dis.resize(n, -1);
            for (int x = 1; x <= nx; x++) {
                if (mx[x] == -1) {
                    dis[x] = 0;
                    q.push(x);
                }
            }
            while (!q.empty()) {
                int x = q.front();
                q.pop();
                Each(y, g[x]) {
                    if (my[y] != -1 && dis[my[y]] == -1) {
                        dis[my[y]] = dis[x] + 1;
                        q.push(my[y]);
                    }
                }
            }

            bool brk = true;
            vis.clear();
            vis.resize(n, 0);
            for (int x = 1; x <= nx; x++)
                if (mx[x] == -1 && dfs(x))
                    brk = false;

            if (brk) break;
        }
        MXCNT = 0;
        for (int x = 1; x <= nx; x++)
            if (mx[x] != -1) MXCNT++;
    }
} hk;
```

## 4.5 Blossom

```
const int N=5e2+10;
struct Graph{
    int to[N],bro[N],head[N],e;
    int lnk[N],vis[N],stp,n;
    void init(int _n){
        stp=0;e=1;n=_n;
        FOR(i,0,n+1)head[i]=lnk[i]=vis[i]=0;
    }
    void add(int u,int v){
        to[e]=v,bro[e]=head[u],head[u]=e++;
        to[e]=u,bro[e]=head[v],head[v]=e++;
    }
    bool dfs(int x){
        vis[x]=stp;
        for(int i=head[x];i;i=bro[i])
        {
            int v=to[i];
            if(!lnk[v])
            {
                lnk[x]=v;lnk[v]=x;
                return true;
            }
            else if(vis[lnk[v]]<stp)
            {
                int w=lnk[v];
                lnk[x]=v,lnk[v]=x,lnk[w]=0;
                if(dfs(w))return true;
                lnk[w]=v,lnk[v]=w,lnk[x]=0;
            }
        }
        return false;
    }
    int solve(){
        int ans=0;
        FOR(i,1,n+1){
            if(!lnk[i]){
                stp++;
                ans+=dfs(i);
            }
        }
        return ans;
    }
    void print_matching(){
        FOR(i,1,n+1)
            if(i<graph.lnk[i])
                cout<<i<<" "<<graph.lnk[i]<<endl;
    }
};
```

## 4.6 Weighted Blossom

```
struct WeightGraph {  // 1-based
    static const int inf = INT_MAX;
    static const int maxn = 514;
    struct edge {
        int u, v, w;
        edge() {}
        edge(int u, int v, int w) : u(u), v(v), w(w) {}
    };
    int n, n_x;
    edge g[maxn * 2][maxn * 2];
    int lab[maxn * 2];
    int match[maxn * 2], slack[maxn * 2], st[maxn * 2], pa[maxn * 2];
    int flo_from[maxn * 2][maxn + 1], S[maxn * 2], vis[maxn * 2];
    vector<int> flo[maxn * 2];
    queue<int> q;
    int e_delta(const edge &e) { return lab[e.u] + lab[e.v] - g[e.u][e.v].w * 2; }
    void update_slack(int u, int x) {
        if (!slack[x] || e_delta(g[u][x]) < e_delta(g[slack[x]][x])) slack[x] = u;
    }
    void set_slack(int x) {
        slack[x] = 0;
        for (int u = 1; u <= n; ++u)
            if (g[u][x].w > 0 && st[u] != x && S[st[u]] == 0)
```

```
                update_slack(u, x);
    }
    void q_push(int x) {
        if (x <= n)
            q.push(x);
        else
            for (size_t i = 0; i < flo[x].size(); i++) q_push(flo[x][i]);
    }
    void set_st(int x, int b) {
        st[x] = b;
        if (x > n)
            for (size_t i = 0; i < flo[x].size(); ++i) set_st(flo[x][i], b);
    }
    int get_pr(int b, int xr) {
        int pr = find(flo[b].begin(), flo[b].end(), xr) - flo[b].begin();
        if (pr % 2 == 1) {
            reverse(flo[b].begin() + 1, flo[b].end());
            return (int)flo[b].size() - pr;
        }
        return pr;
    }
    void set_match(int u, int v) {
        match[u] = g[u][v].v;
        if (u <= n) return;
        edge e = g[u][v];
        int xr = flo_from[u][e.u], pr = get_pr(u, xr);
        for (int i = 0; i < pr; ++i) set_match(flo[u][i], flo[u][i ^ 1]);
        set_match(xr, v);
        rotate(flo[u].begin(), flo[u].begin() + pr, flo[u].end());
    }
    void augment(int u, int v) {
        for (;;) {
            int xnv = st[match[u]];
            set_match(u, v);
            if (!xnv) return;
            set_match(xnv, st[pa[xnv]]);
            u = st[pa[xnv]], v = xnv;
        }
    }
    int get_lca(int u, int v) {
        static int t = 0;
        for (++t; u || v; swap(u, v)) {
            if (u == 0) continue;
            if (vis[u] == t) return u;
            vis[u] = t;
            u = st[match[u]];
            if (u) u = st[pa[u]];
        }
        return 0;
    }
    void add_blossom(int u, int lca, int v) {
        int b = n + 1;
        while (b <= n_x && st[b]) ++b;
        if (b > n_x) ++n_x;
        lab[b] = 0, S[b] = 0;
        match[b] = match[lca];
        flo[b].clear();
        flo[b].push_back(lca);
        for (int x = u, y; x != lca; x = st[pa[y]])
            flo[b].push_back(x), flo[b].push_back(y = st[match[x]]), q_push(y);
        reverse(flo[b].begin() + 1, flo[b].end());
        for (int x = v, y; x != lca; x = st[pa[y]])
            flo[b].push_back(x), flo[b].push_back(y = st[match[x]]), q_push(y);
        set_st(b, b);
        for (int x = 1; x <= n_x; ++x) g[b][x].w = g[x][b].w = 0;
        for (int x = 1; x <= n; ++x) flo_from[b][x] = 0;
        for (size_t i = 0; i < flo[b].size(); ++i) {
            int xs = flo[b][i];
            for (int x = 1; x <= n_x; ++x)
                if (g[b][x].w == 0 || e_delta(g[xs][x]) < e_delta(g[b][x]))
                    g[b][x] = g[xs][x], g[x][b] = g[x][xs];
            for (int x = 1; x <= n; ++x)
                if (flo_from[xs][x]) flo_from[b][x] = xs;
        }
        set_slack(b);
    }
    void expand_blossom(int b) {
        for (size_t i = 0; i < flo[b].size(); ++i)
            set_st(flo[b][i], flo[b][i]);
        int xr = flo_from[b][g[b][pa[b]].u], pr = get_pr(b, xr);
        for (int i = 0; i < pr; i += 2) {
            int xs = flo[b][i], xns = flo[b][i + 1];
            pa[xs] = g[xns][xs].u;
            S[xs] = 1, S[xns] = 0;
            slack[xs] = 0, set_slack(xns);
            q_push(xns);
        }
        S[xr] = 1, pa[xr] = pa[b];
        for (size_t i = pr + 1; i < flo[b].size(); ++i) {
            int xs = flo[b][i];
            S[xs] = -1, set_slack(xs);
        }
        st[b] = 0;
    }
    bool on_found_edge(const edge &e) {
        int u = st[e.u], v = st[e.v];
        if (S[v] == -1) {
            pa[v] = e.u, S[v] = 1;
            int nu = st[match[v]];
            slack[v] = slack[nu] = 0;
            S[nu] = 0, q_push(nu);
        } else if (S[v] == 0) {
            int lca = get_lca(u, v);
            if (!lca)
                return augment(u, v), augment(v, u), true;
            else
                add_blossom(u, lca, v);
        }
        return false;
    }
    bool matching() {
        memset(S + 1, -1, sizeof(int) * n_x);
        memset(slack + 1, 0, sizeof(int) * n_x);
        q = queue<int>();
        for (int x = 1; x <= n_x; ++x)
            if (st[x] == x && !match[x]) pa[x] = 0, S[x] = 0, q_push(x);
        if (q.empty()) return false;
        for (;;) {
            while (q.size()) {
                int u = q.front();
                q.pop();
                if (S[st[u]] == 1) continue;
                for (int v = 1; v <= n; ++v)
                    if (g[u][v].w > 0 && st[u] != st[v]) {
                        if (e_delta(g[u][v]) == 0) {
                            if (on_found_edge(g[u][v])) return true;
                        } else
                            update_slack(u, st[v]);
                    }
            }
            int d = inf;
            for (int b = n + 1; b <= n_x; ++b)
                if (st[b] == b && S[b] == 1) d = min(d, lab[b] / 2);
            for (int x = 1; x <= n_x; ++x)
                if (st[x] == x && slack[x]) {
                    if (S[x] == -1)
                        d = min(d, e_delta(g[slack[x]][x]));
                    else if (S[x] == 0)
                        d = min(d, e_delta(g[slack[x]][x]) / 2);
                }
            for (int u = 1; u <= n; ++u) {
                if (S[st[u]] == 0) {
                    if (lab[u] <= d) return 0;
                    lab[u] -= d;
                } else if (S[st[u]] == 1)
                    lab[u] += d;
            }
            for (int b = n + 1; b <= n_x; ++b)
                if (st[b] == b) {
                    if (S[st[b]] == 0)
                        lab[b] += d * 2;
                    else if (S[st[b]] == 1)
                        lab[b] -= d * 2;
                }
            q = queue<int>();
            for (int x = 1; x <= n_x; ++x)
                if (st[x] == x && slack[x] && st[slack[x]] != x && e_delta(g[slack[x]][x]) == 0)
                    if (on_found_edge(g[slack[x]][x])) return true;
            for (int b = n + 1; b <= n_x; ++b)
                if (st[b] == b && S[b] == 1 && lab[b] == 0) expand_blossom(b);
        }
        return false;
    }
    pair<long long, int> solve() {
        memset(match + 1, 0, sizeof(int) * n);
        n_x = n;
        int n_matches = 0;
        long long tot_weight = 0;
        for (int u = 0; u <= n; ++u) st[u] = u, flo[u].clear();
        int w_max = 0;
        for (int u = 1; u <= n; ++u)
            for (int v = 1; v <= n; ++v) {
                flo_from[u][v] = (u == v ? u : 0);
                w_max = max(w_max, g[u][v].w);
            }
        for (int u = 1; u <= n; ++u) lab[u] = w_max;
        while (matching()) ++n_matches;
        for (int u = 1; u <= n; ++u)
            if (match[u] && match[u] < u)
                tot_weight += g[u][match[u]].w;
        return make_pair(tot_weight, n_matches);
    }
    void add_edge(int ui, int vi, int wi) { g[ui][vi].w = g[vi][ui].w = wi; }
    void init(int _n) {
        n = _n;
        for (int u = 1; u <= n; ++u)
            for (int v = 1; v <= n; ++v)
                g[u][v] = edge(u, v, 0);
    }
};
```

## 4.7 Cover / Independent Set

```
V(E) Cover: choose some V(E) to cover all E(V)
V(E) Independ: set of V(E) not adj to each other

M = Max Matching
Cv = Min V Cover
Ce = Min E Cover
Iv = Max V Ind
Ie = Max E Ind (equiv to M)

M = Cv (Konig Theorem)
Iv = V \ Cv
Ce = V - M

Construct Cv:
1. Run Dinic
2. Find s-t min cut
3. Cv = {X in T} + {Y in S}
```

# 5 Graph

## 5.1 Heavy-Light Decomposition

```
const int N = 2e5 + 5;
int n, dfn[N], son[N], top[N], num[N], dep[N], p[N];
vector<int> path[N];
struct node {
    int mx, sum;
} seg[N << 2];
void update(int x, int l, int r, int qx, int val) {
    if (l == r) {
        seg[x].mx = seg[x].sum = val;
        return;
    }
    int mid = (l + r) >> 1;
    if (qx <= mid)update(x << 1, l, mid, qx, val);
    else update(x << 1 | 1, mid + 1, r, qx, val);
    seg[x].mx = max(seg[x << 1].mx, seg[x << 1 | 1].mx);
    seg[x].sum = seg[x << 1].sum + seg[x << 1 | 1].sum;
}
int big(int x, int l, int r, int ql, int qr) {
    if (ql <= l && r <= qr) return seg[x].mx;
    int mid = (l + r) >> 1;
    int res = -INF;
    if (ql <= mid) res = max(res, big(x << 1, l, mid, ql, qr));
    if (mid < qr) res = max(res, big(x << 1 | 1, mid + 1, r, ql, qr));
    return res;
}
int ask(int x, int l, int r, int ql, int qr) {
    if (ql <= l && r <= qr) return seg[x].sum;
    int mid = (l + r) >> 1;
    int res = 0;
    if (ql <= mid) res += ask(x << 1, l, mid, ql, qr);
    if (mid < qr) res += ask(x << 1 | 1, mid + 1, r, ql, qr);
    return res;
}
void dfs1(int now) {
    son[now] = -1;
    num[now] = 1;
    for (auto i : path[now]) {
        if (!dep[i]) {
            dep[i] = dep[now] + 1;
            p[i] = now;
            dfs1(i);
            num[now] += num[i];
            if (son[now] == -1 || num[i] > num[son[now]]) son[now] = i;
        }
    }
}
int cnt;
void dfs2(int now, int t) {
    top[now] = t;
    cnt++;
    dfn[now] = cnt;
    if (son[now] == -1) return;
    dfs2(son[now], t);
    for (auto i : path[now])
        if (i != p[now] && i != son[now])dfs2(i, i);
}
int path_big(int x, int y) {
    int res = -INF;
    while (top[x] != top[y]) {
        if (dep[top[x]] < dep[top[y]]) swap(x, y);
        res = max(res, big(1, 1, n, dfn[top[x]], dfn[x]));
        x = p[top[x]];
    }
    if (dfn[x] > dfn[y]) swap(x, y);
    res = max(res, big(1, 1, n, dfn[x], dfn[y]));
    return res;
}
int path_sum(int x, int y) {
    int res = 0;
    while (top[x] != top[y]) {
        if (dep[top[x]] < dep[top[y]]) swap(x, y);
        res += ask(1, 1, n, dfn[top[x]], dfn[x]);
        x = p[top[x]];
    }
    if (dfn[x] > dfn[y]) swap(x, y);
    res += ask(1, 1, n, dfn[x], dfn[y]);
    return res;
```

```cpp
}
void buildTree() {
    FOR(i, 0, n - 1) {
        int a, b;
        cin >> a >> b;
        path[a].pb(b);
        path[b].pb(a);
    }
}
void buildHLD(int root) {
    dep[root] = 1;
    dfs1(root);
    dfs2(root, root);
    FOR(i, 1, n + 1) {
        int now;
        cin >> now;
        update(1, 1, n, dfn[i], now);
    }
}
```

## 5.2 Centroid Decomposition

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 5;
vector<int> a[N];
int sz[N], lv[N];
bool used[N];
int f_sz(int x, int p) {
    sz[x] = 1;
    for (int i : a[x])
        if (i != p && !used[i])
            sz[x] += f_sz(i, x);
    return sz[x];
}
int f_cen(int x, int p, int total) {
    for (int i : a[x]) {
        if (i != p && !used[i] && 2 * sz[i] > total)
            return f_cen(i, x, total);
    }
    return x;
}
void cd(int x, int p) {
    int total = f_sz(x, p);
    int cen = f_cen(x, p, total);
    lv[cen] = lv[p] + 1;
    used[cen] = 1;
    // cout << "cd: " << x << " " << p << " " << cen << "\n";
    for (int i : a[cen]) {
        if (!used[i])
            cd(i, cen);
    }
}
int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    int n;
    cin >> n;
    for (int i = 0, x, y; i < n - 1; i++) {
        cin >> x >> y;
        a[x].push_back(y);
        a[y].push_back(x);
    }
    cd(1, 0);
    for (int i = 1; i <= n; i++)
        cout << (char)('A' + lv[i] - 1) << " ";
    cout << "\n";
}
```

## 5.3 Bellman-Ford + SPFA

```cpp
int n, m;

// Graph
vector<vector<pair<int, ll> > > g;
vector<ll> dis;
vector<bool> negCycle;

// SPFA
vector<int> rlx;
queue<int> q;
vector<bool> inq;
vector<int> pa;
void SPFA(vector<int>& src) {
    dis.assign(n + 1, LINF);
    negCycle.assign(n + 1, false);
    rlx.assign(n + 1, 0);
    while (!q.empty()) q.pop();
    inq.assign(n + 1, false);
    pa.assign(n + 1, -1);

    for (auto& s : src) {
        dis[s] = 0;
        q.push(s);
        inq[s] = true;
    }

    while (!q.empty()) {
        int u = q.front();
        q.pop();
        inq[u] = false;
        if (rlx[u] >= n) {
            negCycle[u] = true;
        } else
            for (auto& e : g[u]) {
                int v = e.first;
                ll w = e.second;
                if (dis[v] > dis[u] + w) {
                    dis[v] = dis[u] + w;
                    rlx[v] = rlx[u] + 1;
                    pa[v] = u;
                    if (!inq[v]) {
                        q.push(v);
                        inq[v] = true;
                    }
                }
            }
    }
}

// Bellman-Ford
queue<int> q;
vector<int> pa;
void BellmanFord(vector<int>& src) {
    dis.assign(n + 1, LINF);
    negCycle.assign(n + 1, false);
    pa.assign(n + 1, -1);

    for (auto& s : src) dis[s] = 0;

    for (int rlx = 1; rlx <= n; rlx++) {
        for (int u = 1; u <= n; u++) {
            if (dis[u] == LINF) continue;  // Important !!
            for (auto& e : g[u]) {
                int v = e.first;
                ll w = e.second;
                if (dis[v] > dis[u] + w) {
                    dis[v] = dis[u] + w;
                    pa[v] = u;
                    if (rlx == n) negCycle[v] = true;
                }
            }
        }
    }
}

// Negative Cycle Detection
void NegCycleDetect() {
    /* No Neg Cycle: NO
    Exist Any Neg Cycle:
    YES
    v0 v1 v2 ... vk v0 */

    vector<int> src;
    for (int i = 1; i <= n; i++)
        src.emplace_back(i);

    SPFA(src);
    // BellmanFord(src);
    int ptr = -1;
    for (int i = 1; i <= n; i++)
        if (negCycle[i]) {
            ptr = i;
            break;
        }

    if (ptr == -1) {
        return cout << "NO" << endl, void();
    }

    cout << "YES\n";
    vector<int> ans;
    vector<bool> vis(n + 1, false);

    while (true) {
        ans.emplace_back(ptr);
        if (vis[ptr]) break;
        vis[ptr] = true;
        ptr = pa[ptr];
    }
    reverse(ans.begin(), ans.end());

    vis.assign(n + 1, false);
    for (auto& x : ans) {
        cout << x << ' ';
        if (vis[x]) break;
        vis[x] = true;
    }
    cout << endl;
}

// Distance Calculation
void calcDis(int s) {
    vector<int> src;
    src.emplace_back(s);
    SPFA(src);
    // BellmanFord(src);

    while (!q.empty()) q.pop();
    for (int i = 1; i <= n; i++)
        if (negCycle[i]) q.push(i);

    while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (auto& e : g[u]) {
            int v = e.first;
            if (!negCycle[v]) {
                q.push(v);
                negCycle[v] = true;
            }
        }
    }
}
```

## 5.4 BCC - AP

```
int n, m;
int low[maxn], dfn[maxn], instp;
vector<int> E, g[maxn];
bitset<maxn> isap;
bitset<maxm> vis;
stack<int> stk;
int bccnt;
vector<int> bcc[maxn];
inline void popout(int u) {
    bccnt++;
    bcc[bccnt].emplace_back(u);
    while (!stk.empty()) {
        int v = stk.top();
        if (u == v) break;
        stk.pop();
        bcc[bccnt].emplace_back(v);
    }
}
void dfs(int u, bool rt = 0) {
    stk.push(u);
    low[u] = dfn[u] = ++instp;
    int kid = 0;
    Each(e, g[u]) {
        if (vis[e]) continue;
        vis[e] = true;
        int v = E[e] ^ u;
        if (!dfn[v]) {
            // tree edge
            kid++;
            dfs(v);
            low[u] = min(low[u], low[v]);
            if (!rt && low[v] >= dfn[u]) {
                // bcc found: u is ap
                isap[u] = true;
                popout(u);
            }
        } else {
            // back edge
            low[u] = min(low[u], dfn[v]);
        }
    }
    // special case: root
    if (rt) {
        if (kid > 1) isap[u] = true;
        popout(u);
    }
}
void init() {
    cin >> n >> m;
    fill(low, low + maxn, INF);
    REP(i, m) {
        int u, v;
        cin >> u >> v;
        g[u].emplace_back(i);
        g[v].emplace_back(i);
        E.emplace_back(u ^ v);
    }
}
void solve() {
    FOR(i, 1, n + 1, 1) {
        if (!dfn[i]) dfs(i, true);
    }
    vector<int> ans;
    int cnt = 0;
    FOR(i, 1, n + 1, 1) {
        if (isap[i]) cnt++, ans.emplace_back(i);
    }
    cout << cnt << endl;
    Each(i, ans) cout << i << ' ';
    cout << endl;
}
```

## 5.5 BCC - Bridge

```
int n, m;
vector<int> g[maxn], E;
int low[maxn], dfn[maxn], instp;
int bccnt, bccid[maxn];
stack<int> stk;
bitset<maxm> vis, isbrg;
void init() {
    cin >> n >> m;
    REP(i, m) {
        int u, v;
        cin >> u >> v;
        E.emplace_back(u ^ v);
        g[u].emplace_back(i);
        g[v].emplace_back(i);
    }
    fill(low, low + maxn, INF);
}
void popout(int u) {
    bccnt++;
    while (!stk.empty()) {
        int v = stk.top();
        if (v == u) break;
        stk.pop();
        bccid[v] = bccnt;
    }
}
void dfs(int u) {
    stk.push(u);
    low[u] = dfn[u] = ++instp;

    Each(e, g[u]) {
        if (vis[e]) continue;
        vis[e] = true;

        int v = E[e] ^ u;
        if (dfn[v]) {
            // back edge
            low[u] = min(low[u], dfn[v]);
        } else {
            // tree edge
            dfs(v);
            low[u] = min(low[u], low[v]);
            if (low[v] == dfn[v]) {
                isbrg[e] = true;
                popout(u);
            }
        }
    }
}
void solve() {
    FOR(i, 1, n + 1, 1) {
        if (!dfn[i]) dfs(i);
    }
    vector<pii> ans;
    vis.reset();
    FOR(u, 1, n + 1, 1) {
        Each(e, g[u]) {
            if (!isbrg[e] || vis[e]) continue;
            vis[e] = true;
            int v = E[e] ^ u;
            ans.emplace_back(mp(u, v));
        }
    }
    cout << (int)ans.size() << endl;
    Each(e, ans) cout << e.F << ' ' << e.S << endl;
}
```

## 5.6 SCC - Tarjan

```
// 2-SAT
vector<int> E, g[maxn];  // 1~n, n+1~2n
int low[maxn], in[maxn], instp;
int sccnt, sccid[maxn];
stack<int> stk;
bitset<maxn> ins, vis;
int n, m;
void init() {
    cin >> m >> n;
    E.clear();
    fill(g, g + maxn, vector<int>());
    fill(low, low + maxn, INF);
    memset(in, 0, sizeof(in));
    instp = 1;
    sccnt = 0;
    memset(sccid, 0, sizeof(sccid));
    ins.reset();
    vis.reset();
}
inline int no(int u) {
    return (u > n ? u - n : u + n);
}
int ecnt = 0;
inline void clause(int u, int v) {
    E.eb(no(u) ^ v);
    g[no(u)].eb(ecnt++);
    E.eb(no(v) ^ u);
    g[no(v)].eb(ecnt++);
}
void dfs(int u) {
    in[u] = instp++;
    low[u] = in[u];
    stk.push(u);
    ins[u] = true;

    Each(e, g[u]) {
        if (vis[e]) continue;
        vis[e] = true;

        int v = E[e] ^ u;
        if (ins[v])
            low[u] = min(low[u], in[v]);
        else if (!in[v]) {
            dfs(v);
            low[u] = min(low[u], low[v]);
        }
    }
    if (low[u] == in[u]) {
        sccnt++;
        while (!stk.empty()) {
            int v = stk.top();
            stk.pop();
            ins[v] = false;
            sccid[v] = sccnt;
            if (u == v) break;
        }
    }
}
int main() {
    init();
    REP(i, m) {
        char su, sv;
        int u, v;
        cin >> su >> u >> sv >> v;
        if (su == '-') u = no(u);
        if (sv == '-') v = no(v);
        clause(u, v);
    }
    FOR(i, 1, 2 * n + 1, 1) {
        if (!in[i]) dfs(i);
    }
    FOR(u, 1, n + 1, 1) {
        int du = no(u);
        if (sccid[u] == sccid[du]) {
            return cout << "IMPOSSIBLE\n", 0;
        }
    }
    FOR(u, 1, n + 1, 1) {
        int du = no(u);
        cout << (sccid[u] < sccid[du] ? '+' : '-') << ' ';
    }
    cout << endl;
}
```

## 5.7 SCC - Kosaraju

```
const int N = 1e5 + 10;
vector<int> ed[N], ed_b[N];  // 反邊
vector<int> SCC(N);          // 最後SCC的分組
bitset<N> vis;
int SCC_cnt;
int n, m;
vector<int> pre;  // 後序遍歷

void dfs(int x) {
    vis[x] = 1;
    for (int i : ed[x]) {
        if (vis[i]) continue;
        dfs(i);
    }
    pre.push_back(x);
}

void dfs2(int x) {
    vis[x] = 1;
    SCC[x] = SCC_cnt;
    for (int i : ed_b[x]) {
        if (vis[i]) continue;
        dfs2(i);
    }
}

void kosaraju() {
    for (int i = 1; i <= n; i++) {
        if (!vis[i]) {
            dfs(i);
        }
    }
    SCC_cnt = 0;
    vis = 0;
    for (int i = n - 1; i >= 0; i--) {
        if (!vis[pre[i]]) {
            SCC_cnt++;
            dfs2(pre[i]);
        }
    }
}
```

## 5.8 Eulerian Path - Undir

```
// from 1 to n
#define gg return cout << "IMPOSSIBLE\n", void();

int n, m;
vector<int> g[maxn];
bitset<maxn> inodd;

void init() {
    cin >> n >> m;
    inodd.reset();
    for (int i = 0; i < m; i++) {
        int u, v;
        cin >> u >> v;
        inodd[u] = inodd[u] ^ true;
        inodd[v] = inodd[v] ^ true;
        g[u].emplace_back(v);
        g[v].emplace_back(u);
    }
}
stack<int> stk;
void dfs(int u) {
    while (!g[u].empty()) {
        int v = g[u].back();
        g[u].pop_back();
        dfs(v);
    }
    stk.push(u);
}
```

## 5.9 Eulerian Path - Dir

```
// from node 1 to node n
#define gg return cout << "IMPOSSIBLE\n", 0

int n, m;
vector<int> g[maxn];
stack<int> stk;
int in[maxn], out[maxn];

void init() {
    cin >> n >> m;
    for (int i = 0; i < m; i++) {
        int u, v;
        cin >> u >> v;
        g[u].emplace_back(v);
        out[u]++, in[v]++;
    }
    for (int i = 1; i <= n; i++) {
        if (i == 1 && out[i] - in[i] != 1) gg;
        if (i == n && in[i] - out[i] != 1) gg;
        if (i != 1 && i != n && in[i] != out[i]) gg;
    }
}
void dfs(int u) {
    while (!g[u].empty()) {
        int v = g[u].back();
        g[u].pop_back();
        dfs(v);
    }
    stk.push(u);
}
void solve() {
    dfs(1) for (int i = 1; i <= n; i++) if ((int)g[i].size()) gg;
    while (!stk.empty()) {
        int u = stk.top();
        stk.pop();
        cout << u << ' ';
    }
}
```

## 5.10 Hamilton Path

```
// top down DP
// Be Aware Of Multiple Edges
int n, m;
ll dp[maxn][1<<maxn];
int adj[maxn][maxn];

void init() {
    cin >> n >> m;
    fill(dp[0], dp[maxn-1]+(1<<maxn), -1);
}

void DP(int i, int msk) {
    if (dp[i][msk] != -1) return;
    dp[i][msk] = 0;
    REP(j, n) if (j != i && (msk & (1<<j)) && adj[j][i]) {
        int sub = msk ^ (1<<i);
        if (dp[j][sub] == -1) DP(j, sub);
        dp[i][msk] += dp[j][sub] * adj[j][i];
        if (dp[i][msk] >= MOD) dp[i][msk] %= MOD;
    }
}


int main() {
    WiwiHorz
    init();

    REP(i, m) {
        int u, v;
        cin >> u >> v;
        if (u == v) continue;
        adj[--u][--v]++;
    }

    dp[0][1] = 1;
    FOR(i, 1, n, 1) {
        dp[i][1] = 0;
        dp[i][1|(1<<i)] = adj[0][i];
    }
    FOR(msk, 1, (1<<n), 1) {
        if (msk == 1) continue;
        dp[0][msk] = 0;
    }


    DP(n-1, (1<<n)-1);
    cout << dp[n-1][(1<<n)-1] << endl;

    return 0;
}
```

## 5.11 Kth Shortest Path

```
// time: O(|E| \lg |E|+|V| \lg |V|+K)
// memory: O(|E| \lg |E|+|V|)
struct KSP{ // 1-base
  struct nd{
    int u,v; ll d;
    nd(int ui=0,int vi=0,ll di=INF){ u=ui; v=vi; d=di; }
  };
  struct heap{ nd* edge; int dep; heap* chd[4]; };
  static int cmp(heap* a,heap* b)
  { return a->edge->d > b->edge->d; }
  struct node{
    int v; ll d; heap* H; nd* E;
    node(){}
    node(ll _d,int _v,nd* _E){ d =_d; v=_v; E=_E; }
    node(heap* _H,ll _d){ H=_H; d=_d; }
    friend bool operator<(node a,node b)
    { return a.d>b.d; }
  };
  int n,k,s,t,dst[N]; nd *nxt[N];
  vector<nd*> g[N],rg[N]; heap *nullNd,*head[N];
  void init(int _n,int _k,int _s,int _t){
    n=_n; k=_k; s=_s; t=_t;
    for(int i=1;i<=n;i++){
      g[i].clear(); rg[i].clear();
      nxt[i]=NULL; head[i]=NULL; dst[i]=-1;
    }
  }
  void addEdge(int ui,int vi,ll di){
    nd* e=new nd(ui,vi,di);
    g[ui].push_back(e); rg[vi].push_back(e);
  }
  queue<int> dfsQ;
  void dijkstra(){
    while(dfsQ.size()) dfsQ.pop();
    priority_queue<node> Q; Q.push(node(0,t,NULL));
    while (!Q.empty()){
      node p=Q.top(); Q.pop(); if(dst[p.v]!=-1)continue;
      dst[p.v]=p.d; nxt[p.v]=p.E; dfsQ.push(p.v);
      for(auto e:rg[p.v]) Q.push(node(p.d+e->d,e->u,e));
    }
  }
  heap* merge(heap* curNd,heap* newNd){
    if(curNd==nullNd) return newNd;
    heap* root=new heap;memcpy(root,curNd,sizeof(heap));
    if(newNd->edge->d<curNd->edge->d){
      root->edge=newNd->edge;
      root->chd[2]=newNd->chd[2];
      root->chd[3]=newNd->chd[3];
      newNd->edge=curNd->edge;
      newNd->chd[2]=curNd->chd[2];
      newNd->chd[3]=curNd->chd[3];
    }
    if(root->chd[0]->dep<root->chd[1]->dep)
      root->chd[0]=merge(root->chd[0],newNd);
    else root->chd[1]=merge(root->chd[1],newNd);
    root->dep=max(root->chd[0]->dep,
              root->chd[1]->dep)+1;
    return root;
  }
  vector<heap*> V;
  void build(){
    nullNd=new heap; nullNd->dep=0; nullNd->edge=new nd;
    fill(nullNd->chd,nullNd->chd+4,nullNd);
    while(not dfsQ.empty()){
      int u=dfsQ.front(); dfsQ.pop();
      if(!nxt[u]) head[u]=nullNd;
      else head[u]=head[nxt[u]->v];
      V.clear();
      for(auto&& e:g[u]){
        int v=e->v;
        if(dst[v]==-1) continue;
        e->d+=dst[v]-dst[u];
        if(nxt[u]!=e){
          heap* p=new heap;fill(p->chd,p->chd+4,nullNd);
          p->dep=1; p->edge=e; V.push_back(p);
        }
      }
      if(V.empty()) continue;
      make_heap(V.begin(),V.end(),cmp);
#define L(X) ((X<<1)+1)
#define R(X) ((X<<1)+2)
      for(size_t i=0;i<V.size();i++){
        if(L(i)<V.size()) V[i]->chd[2]=V[L(i)];
        else V[i]->chd[2]=nullNd;
        if(R(i)<V.size()) V[i]->chd[3]=V[R(i)];
        else V[i]->chd[3]=nullNd;
      }
      head[u]=merge(head[u],V.front());
    }
  }
  vector<ll> ans;
  void first_K(){
    ans.clear(); priority_queue<node> Q;
    if(dst[s]==-1) return;
    ans.push_back(dst[s]);
    if(head[s]!=nullNd)
      Q.push(node(head[s],dst[s]+head[s]->edge->d));
    for(int _=1;_<k and not Q.empty();_++){
      node p=Q.top(),q; Q.pop(); ans.push_back(p.d);
      if(head[p.H->edge->v]!=nullNd){
        q.H=head[p.H->edge->v]; q.d=p.d+q.H->edge->d;
        Q.push(q);
      }
      for(int i=0;i<4;i++)
        if(p.H->chd[i]!=nullNd){
          q.H=p.H->chd[i];
          q.d=p.d-p.H->edge->d+p.H->chd[i]->edge->d;
          Q.push(q);
  } }   }
  void solve(){ // ans[i] stores the i-th shortest path
    dijkstra(); build();
    first_K(); // ans.size() might less than k
  }
} solver;
```

## 5.12 System of Difference Constraints

```
vector<vector<pair<int, ll>>> G;
void add(int u, int v, ll w) {
    G[u].emplace_back(make_pair(v, w));
}
```

- $x_u - x_v \leq c \Rightarrow$ `add(v, u, c)`
- $x_u - x_v \geq c \Rightarrow$ `add(u, v, -c)`
- $x_u - x_v = c \Rightarrow$ `add(v, u, c), add(u, v -c)`
- $x_u \geq c \Rightarrow$ add super vertex $x_0 = 0$, then $x_u - x_0 \geq c \Rightarrow$ `add(u, 0, -c)`
- Don't for get non-negative constraints for every variable if specified implicitly.
- Interval sum $\Rightarrow$ Use prefix sum to transform into differential constraints. Don't for get $S_{i+1} - S_i \geq 0$ if $x_i$ needs to be non-negative.
- $\frac{x_u}{x_v} \leq c \Rightarrow \log x_u - \log x_v \leq \log c$

# 6 String

## 6.1 Aho Corasick

```
struct ACautomata {
    struct Node {
        int cnt;
        Node *go[26], *fail, *dic;
        Node() {
            cnt = 0;
            fail = 0;
            dic = 0;
            memset(go, 0, sizeof(go));
        }
    } pool[1048576], *root;
    int nMem;
    Node *new_Node() {
        pool[nMem] = Node();
        return &pool[nMem++];
    }
    void init() {
        nMem = 0;
        root = new_Node();
    }
    void add(const string &str) { insert(root, str, 0); }
    void insert(Node *cur, const string &str, int pos) {
        for (int i = pos; i < str.size(); i++) {
            if (!cur->go[str[i] - 'a'])
                cur->go[str[i] - 'a'] = new_Node();
            cur = cur->go[str[i] - 'a'];
        }
        cur->cnt++;
    }
    void make_fail() {
        queue<Node *> que;
        que.push(root);
        while (!que.empty()) {
            Node *fr = que.front();
            que.pop();
            for (int i = 0; i < 26; i++) {
                if (fr->go[i]) {
                    Node *ptr = fr->fail;
                    while (ptr && !ptr->go[i]) ptr = ptr->fail;
                    fr->go[i]->fail = ptr = (ptr ? ptr->go[i] : root);
                    fr->go[i]->dic = (ptr->cnt ? ptr : ptr->dic);
                    que.push(fr->go[i]);
                }
            }
        }
    }
} AC;
```

## 6.2 KMP

```
vector<int> f;
void buildFailFunction(string &s) {
    f.resize(s.size(), -1);
    for (int i = 1; i < s.size(); i++) {
        int now = f[i - 1];
        while (now != -1 and s[now + 1] != s[i]) now = f[now];
        if (s[now + 1] == s[i]) f[i] = now + 1;
    }
}

void KMPmatching(string &a, string &b) {
    for (int i = 0, now = -1; i < a.size(); i++) {
        while (a[i] != b[now + 1] and now != -1) now = f[now];
        if (a[i] == b[now + 1]) now++;
        if (now + 1 == b.size()) {
            cout << "found a match start at position " << i - now << endl;
            now = f[now];
        }
    }
}
```

## 6.3 Z Value

```
string is, it, s;
int n;
vector<int> z;
void init() {
    cin >> is >> it;
    s = it + '0' + is;
    n = (int)s.size();
    z.resize(n, 0);
}
void solve() {
    int ans = 0;
    z[0] = n;
    for (int i = 1, l = 0, r = 0; i < n; i++) {
        if (i <= r) z[i] = min(z[i - l], r - i + 1);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]]) z[i]++;
        if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
        if (z[i] == (int)it.size()) ans++;
    }
    cout << ans << endl;
}
```

## 6.4 Manacher

```
int n;
string S, s;
vector<int> m;
void manacher() {
    s.clear();
    s.resize(2 * n + 1, '.');
    for (int i = 0, j = 1; i < n; i++, j += 2) s[j] = S[i];
    m.clear();
    m.resize(2 * n + 1, 0);
    // m[i] := max k such that s[i-k, i+k] is palindrome
    int mx = 0, mxk = 0;
    for (int i = 1; i < 2 * n + 1; i++) {
        if (mx - (i - mx) >= 0) m[i] = min(m[mx - (i - mx)], mx + mxk - i);
        while (0 <= i - m[i] - 1 && i + m[i] + 1 < 2 * n + 1 &&
               s[i - m[i] - 1] == s[i + m[i] + 1]) m[i]++;
        if (i + m[i] > mx + mxk) mx = i, mxk = m[i];
    }
}
void init() {
    cin >> S;
    n = (int)S.size();
}
void solve() {
    manacher();
    int mx = 0, ptr = 0;
    for (int i = 0; i < 2 * n + 1; i++)
        if (mx < m[i]) {
            mx = m[i];
            ptr = i;
        }
    for (int i = ptr - mx; i <= ptr + mx; i++)
        if (s[i] != '.') cout << s[i];
    cout << endl;
}
```

## 6.5 Suffix Array

```
#define F first
#define S second
struct SuffixArray {  // don't forget s += "$";
    int n;
    string s;
    vector<int> suf, lcp, rk;
    vector<int> cnt, pos;
    vector<pair<pii, int> > buc[2];
    void init(string _s) {
        s = _s;
        n = (int)s.size();
        // resize(n): suf, rk, cnt, pos, lcp, buc[0~1]
    }
    void radix_sort() {
        for (int t : {0, 1}) {
            fill(cnt.begin(), cnt.end(), 0);
            for (auto& i : buc[t]) cnt[(t ? i.F.F : i.F.S)]++;
            for (int i = 0; i < n; i++)
                pos[i] = (!i ? 0 : pos[i - 1] + cnt[i - 1]);
            for (auto& i : buc[t])
                buc[t ^ 1][pos[(t ? i.F.F : i.F.S)]++] = i;
        }
    }
    bool fill_suf() {
        bool end = true;
        for (int i = 0; i < n; i++) suf[i] = buc[0][i].S;
        rk[suf[0]] = 0;
        for (int i = 1; i < n; i++) {
            int dif = (buc[0][i].F != buc[0][i - 1].F);
            end &= dif;
            rk[suf[i]] = rk[suf[i - 1]] + dif;
        }
        return end;
    }
    void sa() {
        for (int i = 0; i < n; i++)
            buc[0][i] = make_pair(make_pair(s[i], s[i]), i);
        sort(buc[0].begin(), buc[0].end());
        if (fill_suf()) return;
        for (int k = 0; (1 << k) < n; k++) {
            for (int i = 0; i < n; i++)
```

```
                buc[0][i] = make_pair(make_pair(rk[i], rk[(i + (1 << k)) % n]), i);
            radix_sort();
            if (fill_suf()) return;
        }
    }
    void LCP() {
        int k = 0;
        for (int i = 0; i < n - 1; i++) {
            if (rk[i] == 0) continue;
            int pi = rk[i];
            int j = suf[pi - 1];
            while (i + k < n && j + k < n && s[i + k] == s[j + k]) k++;
            lcp[pi] = k;
            k = max(k - 1, 0);
        }
    }
};
SuffixArray suffixarray;
```

## 6.6 Minimum Rotation

```
// rotate(begin(s), begin(s)+minRotation(s), end(s))
int minRotation(string s) {
    int a = 0, n = s.size();
    s += s;
    for (int b = 0; b < n; b++)
        for (int k = 0; k < n; k++) {
            if (a + k == b || s[a + k] < s[b + k]) {
                b += max(0, k - 1);
                break;
            }
            if (s[a + k] > s[b + k]) {
                a = b;
                break;
            }
        }
    return a;
}
```

## 6.7 Lyndon Factorization

```
vector<string> duval(string const& s) {
    int n = s.size();
    int i = 0;
    vector<string> factorization;
    while (i < n) {
        int j = i + 1, k = i;
        while (j < n && s[k] <= s[j]) {
            if (s[k] < s[j])
                k = i;
            else
                k++;
            j++;
        }
        while (i <= k) {
            factorization.push_back(s.substr(i, j - k));
            i += j - k;
        }
    }
    return factorization;  // O(n)
}
```

## 6.8 Rolling Hash

```
const ll C = 27;
inline int id(char c) { return c - 'a' + 1; }
struct RollingHash {
    string s;
    int n;
    ll mod;
    vector<ll> Cexp, hs;
    RollingHash(string& _s, ll _mod) : s(_s), n((int)_s.size()), mod(_mod) {
        Cexp.assign(n, 0);
        hs.assign(n, 0);
        Cexp[0] = 1;
        for (int i = 1; i < n; i++) {
            Cexp[i] = Cexp[i - 1] * C;
            if (Cexp[i] >= mod) Cexp[i] %= mod;
        }
        hs[0] = id(s[0]);
        for (int i = 1; i < n; i++) {
            hs[i] = hs[i - 1] * C + id(s[i]);
            if (hs[i] >= mod) hs[i] %= mod;
        }
    }
    inline ll query(int l, int r) {
        ll res = hs[r] - (l ? hs[l - 1] * Cexp[r - l + 1] : 0);
        res = (res % mod + mod) % mod;
        return res;
    }
};
```

## 6.9 Trie

```
pii a[N][26];

void build(string &s) {
    static int idx = 0;
    int n = s.size();
    for (int i = 0, v = 0; i < n; i++) {
        pii &now = a[v][s[i] - 'a'];
        if (now.first != -1)
            v = now.first;
        else
            v = now.first = ++idx;
        if (i == n - 1)
            now.second++;
    }
}
```

# 7 Geometry

## 7.1 Basic Operations

```
typedef long long T;
// typedef long double T;
const long double eps = 1e-8;
short sgn(T x) {
    if (abs(x) < eps) return 0;
    return x < 0 ? -1 : 1;
}
struct Pt {
    T x, y;
    Pt(T _x = 0, T _y = 0) : x(_x), y(_y) {}
    Pt operator+(Pt a) { return Pt(x + a.x, y + a.y); }
    Pt operator-(Pt a) { return Pt(x - a.x, y - a.y); }
    Pt operator*(T a) { return Pt(x * a, y * a); }
    Pt operator/(T a) { return Pt(x / a, y / a); }
    T operator*(Pt a) { return x * a.x + y * a.y; }
    T operator^(Pt a) { return x * a.y - y * a.x; }
    bool operator<(Pt a) { return x < a.x || (x == a.x && y < a.y); }
    // return sgn(x-a.x) < 0 || (sgn(x-a.x) == 0 && sgn(y-a.y) < 0); }
    bool operator==(Pt a) { return sgn(x - a.x) == 0 && sgn(y - a.y) == 0; }
};
Pt mv(Pt a, Pt b) { return b - a; }
T len2(Pt a) { return a * a; }
T dis2(Pt a, Pt b) { return len2(b - a); }
short ori(Pt a, Pt b) { return ((a ^ b) > 0) - ((a ^ b) < 0); }
bool onseg(Pt p, Pt l1, Pt l2) {
    Pt a = mv(p, l1), b = mv(p, l2);
    return ((a ^ b) == 0) && ((a * b) <= 0);
}
```

## 7.2 SVG Writer

## 7.3 Sort by Angle

```
int ud(Pt a) {  // up or down half plane
```

```cpp
    if (a.y > 0) return 0;
    if (a.y < 0) return 1;
    return (a.x >= 0 ? 0 : 1);
}
sort(pts.begin(), pts.end(), [&](const Pt& a, const Pt& b) {
    if (ud(a) != ud(b)) return ud(a) < ud(b);
    return (a ^ b) > 0;
});
```

## 7.4 Line Intersection

```cpp
bool line_intersect_check(Pt p1, Pt p2, Pt q1, Pt q2) {
    if (onseg(q1, p1, p2) || onseg(p2, q1, q2) || onseg(q1, p1, p2) || onseg(q2, p1, p2)) return true;
    Pt p = mv(p1, p2), q = mv(q1, q2);
    return (ori(p, mv(p1, q1)) * ori(p, mv(p1, q2)) < 0 && ori(q, mv(q1, p1)) * ori(q, mv(q1, p2)) < 0);
}
// long double
Pt line_intersect(Pt a1, Pt a2, Pt b1, Pt b2) {
    Pt da = mv(a1, a2), db = mv(b1, b2);
    T det = da ^ db;
    if (sgn(det) == 0) {  // parallel
        // return Pt(NAN, NAN);
    }
    T t = ((b1 - a1) ^ db) / det;
    return a1 + da * t;
}
```

## 7.5 Polygon Area

```cpp
// 2 * area
T dbPoly_area(vector<Pt>& e) {
    ll res = 0;
    int sz = e.size();
    for (int i = 0; i < sz; i++) {
        res += e[i] ^ e[(i + 1) % sz];
    }
    return abs(res);
}
```

## 7.6 Convex Hull

```cpp
vector<Pt> convexHull(vector<Pt> pts) {
    vector<Pt> hull;
    sort(pts.begin(), pts.end());
    for (int i = 0; i < 2; i++) {
        int b = hull.size();
        for (auto ei : pts) {
            while (hull.size() - b >= 2 && ori(mv(hull[hull.size() - 2], hull.back()), mv(hull[hull.size() - 2], ei)) == -1) {
                hull.pop_back();
            }
            hull.emplace_back(ei);
        }
        hull.pop_back();
        reverse(pts.begin(), pts.end());
    }
    return hull;
}
```

## 7.7 Point In Convex

```cpp
bool point_in_convex(const vector<Pt> &C, Pt p, bool strict = true) {
    // only works when no three point are collinear
    int n = C.size();
    int a = 1, b = n - 1, r = !strict;
    if (n == 0) return false;
    if (n < 3) return r && onseg(p, C[0], C.back());
    if (ori(mv(C[0], C[a]), mv(C[0], C[b])) > 0) swap(a, b);
    if (ori(mv(C[0], C[a]), mv(C[0], p)) >= r || ori(mv(C[0], C[b]), mv(C[0], p)) <= -r) return false;
    while (abs(a - b) > 1) {
        int c = (a + b) / 2;
        if (ori(mv(C[0], C[c]), mv(C[0], p)) > 0) b = c;
        else a = c;
    }
    return ori(mv(C[a], C[b]), mv(C[a], p)) < r;
}
```

## 7.8 Point Segment Distance

```cpp
double point_segment_dist(Pt q0, Pt q1, Pt p) {
    if (q0 == q1) {
        double dx = double(p.x - q0.x);
        double dy = double(p.y - q0.y);
        return sqrt(dx * dx + dy * dy);
    }
    T d1 = (q1 - q0) * (p - q0);
    T d2 = (q0 - q1) * (p - q1);
    if (d1 >= 0 && d2 >= 0) {
        double area = fabs(double((q1 - q0) ^ (p - q0)));
        double base = sqrt(double(dis2(q0, q1)));
        return area / base;
    }
    double dx0 = double(p.x - q0.x), dy0 = double(p.y - q0.y);
    double dx1 = double(p.x - q1.x), dy1 = double(p.y - q1.y);
    return min(sqrt(dx0 * dx0 + dy0 * dy0), sqrt(dx1 * dx1 + dy1 * dy1));
}
```

## 7.9 Lower Concave Hull

```cpp
struct Line {
  mutable ll m, b, p;
  bool operator<(const Line& o) const { return m < o.m; }
  bool operator<(ll x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> {
  // (for doubles, use inf = 1/.0, div(a,b) = a/b)
  const ll inf = LLONG_MAX;
  ll div(ll a, ll b) { // floored division
    return a / b - ((a ^ b) < 0 && a % b); }
  bool isect(iterator x, iterator y) {
    if (y == end()) { x->p = inf; return false; }
    if (x->m == y->m) x->p = x->b > y->b ? inf : -inf;
    else x->p = div(y->b - x->b, x->m - y->m);
    return x->p >= y->p;
  }
  void add(ll m, ll b) {
    auto z = insert({m, 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) {
    assert(!empty());
    auto l = *lower_bound(x);
    return l.m * x + l.b;
  }
};
```

## 7.10 Pick's Theorem

Consider a polygon which vertices are all lattice points.

Let $i$ = number of points inside the polygon.

Let $b$ = number of points on the boundary of the polygon.

Then we have the following formula:

$$Area = i + \frac{b}{2} - 1$$

## 7.11 Vector In Polygon

## 7.12 Minkowski Sum

```
/* convex hull Minkowski Sum*/
#define INF 1000000000000000LL
int pos(const Pt& tp) {
    if (tp.Y == 0) return tp.X > 0 ? 0 : 1;
    return tp.Y > 0 ? 0 : 1;
}
#define N 300030
Pt pt[N], qt[N], rt[N];
LL Lx, Rx;
int dn, un;
inline bool cmp(Pt a, Pt b) {
    int pa = pos(a), pb = pos(b);
    if (pa == pb) return (a ^ b) > 0;
    return pa < pb;
}
int minkowskiSum(int n, int m) {
    int i, j, r, p, q, fi, fj;
    for (i = 1, p = 0; i < n; i++) {
        if (pt[i].Y < pt[p].Y ||
            (pt[i].Y == pt[p].Y && pt[i].X < pt[p].X)) p = i;
    }
    for (i = 1, q = 0; i < m; i++) {
        if (qt[i].Y < qt[q].Y ||
            (qt[i].Y == qt[q].Y && qt[i].X < qt[q].X)) q = i;
    }
    rt[0] = pt[p] + qt[q];
    r = 1;
    i = p;
    j = q;
    fi = fj = 0;
    while (1) {
        if ((fj && j == q) ||
            ((!fi || i != p) &&
             cmp(pt[(p + 1) % n] - pt[p], qt[(q + 1) % m] - qt[q]))) {
            rt[r] = rt[r - 1] + pt[(p + 1) % n] - pt[p];
            p = (p + 1) % n;
            fi = 1;
        } else {
            rt[r] = rt[r - 1] + qt[(q + 1) % m] - qt[q];
            q = (q + 1) % m;
            fj = 1;
        }
        if (r <= 1 || ((rt[r] - rt[r - 1]) ^ (rt[r - 1] - rt[r - 2])) != 0) r++;
        else rt[r - 1] = rt[r];
        if (i == p && j == q) break;
    }
    return r - 1;
}
void initInConvex(int n) {
    int i, p, q;
    LL Ly, Ry;
    Lx = INF;
    Rx = -INF;
    for (i = 0; i < n; i++) {
        if (pt[i].X < Lx) Lx = pt[i].X;
        if (pt[i].X > Rx) Rx = pt[i].X;
    }
    Ly = Ry = INF;
    for (i = 0; i < n; i++) {
        if (pt[i].X == Lx && pt[i].Y < Ly) {
            Ly = pt[i].Y;
            p = i;
        }
        if (pt[i].X == Rx && pt[i].Y < Ry) {
            Ry = pt[i].Y;
            q = i;
        }
    }
    for (dn = 0, i = p; i != q; i = (i + 1) % n)
        qt[dn++] = pt[i];
    qt[dn] = pt[q];
    Ly = Ry = -INF;
    for (i = 0; i < n; i++) {
        if (pt[i].X == Lx && pt[i].Y > Ly) {
            Ly = pt[i].Y;
            p = i;
        }
        if (pt[i].X == Rx && pt[i].Y > Ry) {
            Ry = pt[i].Y;
            q = i;
        }
    }
    for (un = 0, i = p; i != q; i = (i + n - 1) % n)
        rt[un++] = pt[i];
    rt[un] = pt[q];
}
inline int inConvex(Pt p) {
    int L, R, M;
    if (p.X < Lx || p.X > Rx) return 0;
    L = 0;
    R = dn;
    while (L < R - 1) {
        M = (L + R) / 2;
        if (p.X < qt[M].X) R = M;
        else L = M;
    }
    if (tri(qt[L], qt[R], p) < 0) return 0;
    L = 0;
    R = un;
    while (L < R - 1) {
        M = (L + R) / 2;
        if (p.X < rt[M].X) R = M;
        else L = M;
    }
    if (tri(rt[L], rt[R], p) > 0) return 0;
    return 1;
}
int main() {
    int n, m, i;
    Pt p;
    scanf("%d", &n);
    for (i = 0; i < n; i++) scanf("%lld%lld", &pt[i].X, &pt[i].Y);
    scanf("%d", &m);
    for (i = 0; i < m; i++) scanf("%lld%lld", &qt[i].X, &qt[i].Y);
    n = minkowskiSum(n, m);
    for (i = 0; i < n; i++) pt[i] = rt[i];
    scanf("%d", &m);
    for (i = 0; i < m; i++) scanf("%lld%lld", &qt[i].X, &qt[i].Y);
    n = minkowskiSum(n, m);
    for (i = 0; i < n; i++) pt[i] = rt[i];
    initInConvex(n);
    scanf("%d", &m);
    for (i = 0; i < m; i++) {
        scanf("%lld %lld", &p.X, &p.Y);
        p.X *= 3;
        p.Y *= 3;
        puts(inConvex(p) ? "YES" : "NO");
    }
}
```

## 7.13 Rotating SweepLine

## 7.14 Half Plane Intersection

```
const long double eps = 1e-9, inf = 1e9;
struct Point {
    long double x, y;
    explicit Point(long double x = 0, long double y = 0) : x(x), y(y) {}
    friend Point operator+(const Point& p, const Point& q) {
        return Point(p.x + q.x, p.y + q.y);
    }
    friend Point operator-(const Point& p, const Point& q) {
        return Point(p.x - q.x, p.y - q.y);
    }
    friend Point operator*(const Point& p, const long double& k) {
        return Point(p.x * k, p.y * k);
```

```cpp
    }
    friend long double dot(const Point& p, const Point& q) {
        return p.x * q.x + p.y * q.y;
    }
    friend long double cross(const Point& p, const Point& q) {
        return p.x * q.y - p.y * q.x;
    }
};
struct Halfplane {
    Point p, pq;
    long double angle;
    Halfplane() {}
    Halfplane(const Point& a, const Point& b) : p(a), pq(b - a) {
        angle = atan2l(pq.y, pq.x);
    }
    bool out(const Point& r) {
        return cross(pq, r - p) < -eps;
    }
    bool operator<(const Halfplane& e) const {
        return angle < e.angle;
    }
    friend Point inter(const Halfplane& s, const Halfplane& t) {
        long double alpha = cross((t.p - s.p), t.pq) / cross(s.pq, t.pq);
        return s.p + (s.pq * alpha);
    }
};
vector<Point> hp_intersect(vector<Halfplane>& H) {
    Point box[4] = {// Bounding box in CCW order
                    Point(inf, inf),
                    Point(-inf, inf),
                    Point(-inf, -inf),
                    Point(inf, -inf)};
    for (int i = 0; i < 4; i++) {  // Add bounding box half-planes.
        Halfplane aux(box[i], box[(i + 1) % 4]);
        H.push_back(aux);
    }
    sort(H.begin(), H.end());
    deque<Halfplane> dq;
    int len = 0;
    for (int i = 0; i < int(H.size()); i++) {
        while (len > 1 && H[i].out(inter(dq[len - 1], dq[len - 2]))) {
            dq.pop_back();
            --len;
        }
        while (len > 1 && H[i].out(inter(dq[0], dq[1]))) {
            dq.pop_front();
            --len;
        }
        if (len > 0 && fabsl(cross(H[i].pq, dq[len - 1].pq)) < eps) {
            if (dot(H[i].pq, dq[len - 1].pq) < 0.0)
                return vector<Point>();
            if (H[i].out(dq[len - 1].p)) {
                dq.pop_back();
                --len;
            } else
                continue;
        }
        dq.push_back(H[i]);
        ++len;
    }
    while (len > 2 && dq[0].out(inter(dq[len - 1], dq[len - 2]))) {
        dq.pop_back();
        --len;
    }
    while (len > 2 && dq[len - 1].out(inter(dq[0], dq[1]))) {
        dq.pop_front();
        --len;
    }
    if (len < 3) return vector<Point>();
    vector<Point> ret(len);
    for (int i = 0; i + 1 < len; i++) {
        ret[i] = inter(dq[i], dq[i + 1]);
    }
    ret.back() = inter(dq[len - 1], dq[0]);
    return ret;
}
```

## 7.15 Minimum Enclosing Circle

```cpp
Pt circumcenter(Pt A, Pt B, Pt C) {
    // a1(x-A.x) + b1(y-A.y) = c1
    // a2(x-A.x) + b2(y-A.y) = c2
    // solve using Cramer's rule
    T a1 = B.x - A.x, b1 = B.y - A.y, c1 = dis2(A, B) / 2.0;
    T a2 = C.x - A.x, b2 = C.y - A.y, c2 = dis2(A, C) / 2.0;
    T D = Pt(a1, b1) ^ Pt(a2, b2);
    T Dx = Pt(c1, b1) ^ Pt(c2, b2);
    T Dy = Pt(a1, c1) ^ Pt(a2, c2);
    if (D == 0) return Pt(-INF, -INF);
    return A + Pt(Dx / D, Dy / D);
}
Pt center;
T r2;
void minEncloseCircle() {
    mt19937 gen(chrono::steady_clock::now().time_since_epoch().count());
    shuffle(ALL(E), gen);
    center = E[0], r2 = 0;

    for (int i = 0; i < n; i++) {
        if (dis2(center, E[i]) <= r2) continue;
        center = E[i], r2 = 0;
        for (int j = 0; j < i; j++) {
            if (dis2(center, E[j]) <= r2) continue;
            center = (E[i] + E[j]) / 2.0;
            r2 = dis2(center, E[i]);
            for (int k = 0; k < j; k++) {
                if (dis2(center, E[k]) <= r2) continue;
                center = circumcenter(E[i], E[j], E[k]);
                r2 = dis2(center, E[i]);
            }
        }
    }
}
```

## 7.16 Heart

## 7.17 Tangents

## 7.18 Point In Circle

## 7.19 Union of Circles

## 7.20 Union of Polygons

## 7.21 Delaunay Triangulation

## 7.22 Triangulation Vonoroi

## 7.23 External Bisector

## 7.24 Intersection Area of Polygon and Circle

## 7.25 3D Point

## 7.26 3D Convex Hull

# 8 Number Theory

## 8.1 FFT

```cpp
typedef complex<double> cp;

const double pi = acos(-1);
const int NN = 131072;

struct FastFourierTransform{
    /*
        Iterative Fast Fourier Transform
        How this works? Look at this
        0th recursion 0(000)   1(001)   2(010)   3(011)   4(100)   5(101)   6(110)   7(111)
        1th recursion 0(000)   2(010)   4(100)   6(110) | 1(011)   3(011)   5(101)   7(111)
        2th recursion 0(000)   4(100) | 2(010)   6(110) | 1(011)   5(101) | 3(011)   7(111)
        3th recursion 0(000) | 4(100) | 2(010) | 6(110) | 1(011) | 5(101) | 3(011) | 7(111)
        All the bits are reversed => We can save the reverse of the numbers in an array!
    */
    int n, rev[NN];
    cp omega[NN], iomega[NN];
    void init(int n_){
        n = n_;
        for(int i = 0;i < n_;i++){
            //Calculate the nth roots of unity
            omega[i] = cp(cos(2*pi*i/n_),sin(2*pi*i/n_));
            iomega[i] = conj(omega[i]);
        }
        int k = __lg(n_);
        for(int i = 0;i < n_;i++){
            int t = 0;
            for(int j = 0;j < k;j++){
                if(i & (1<<j)) t |= (1<<(k-j-1));
            }
            rev[i] = t;
        }
    }

    void transform(vector<cp> &a, cp* xomega){
        for(int i = 0;i < n;i++)
            if(i < rev[i]) swap(a[i],a[rev[i]]);
        for(int len = 2; len <= n; len <<= 1){
            int mid = len >> 1;
            int r = n/len;
            for(int j = 0;j < n;j += len)
                for(int i = 0;i < mid;i++){
                    cp tmp = xomega[r*i] * a[j+mid+i];
                    a[j+mid+i] = a[j+i] - tmp;
                    a[j+i] = a[j+i] + tmp;
                }
        }
    }

    void fft(vector<cp> &a){ transform(a,omega); }
    void ifft(vector<cp> &a){ transform(a,iomega); for(int i = 0;i < n;i++) a[i] /= n;}
} FFT;


const int MAXN = 262144;
// (must be 2^k)
// 262144, 524288, 1048576, 2097152, 4194304
// before any usage, run pre_fft() first
typedef long double ld;
typedef complex<ld> cplx; //real() ,imag()
const ld PI = acosl(-1);
const cplx I(0, 1);
cplx omega[MAXN+1];
void pre_fft(){
    for(int i=0; i<=MAXN; i++) {
        omega[i] = exp(i * 2 * PI / MAXN * I);
    }
}
// n must be 2^k
void fft(int n, cplx a[], bool inv=false){
    int basic = MAXN / n;
    int theta = basic;
    for (int m = n; m >= 2; m >>= 1) {
        int mh = m >> 1;
        for (int i = 0; i < mh; i++) {
            cplx w = omega[inv ? MAXN - (i * theta % MAXN) : i * theta % MAXN];
            for (int j = i; j < n; j += m) {
                int k = j + mh;
                cplx x = a[j] - a[k];
                a[j] += a[k];
                a[k] = w * x;
            }
        }
        theta = (theta * 2) % MAXN;
    }
    int i = 0;
    for (int j = 1; j < n - 1; j++) {
        for (int k = n >> 1; k > (i ^= k); k >>= 1);
        if (j < i) swap(a[i], a[j]);
    }
    if(inv) {
        for (i = 0; i < n; i++) a[i] /= n;
    }
}
cplx arr[MAXN + 1];
inline void mul(int _n,long long a[],int _m,long long b[],long long ans[]){
    int n=1, sum = _n + _m - 1;
    while(n < sum) n <<= 1;
    for(int i = 0; i < n; i++) {
        double x= (i < _n ? a[i] : 0), y=(i < _m ? b[i] : 0);
        arr[i] = complex<double>(x + y, x - y);
    }
    fft(n, arr);
    for(int i = 0; i < n; i++) arr[i]=arr[i]*arr[i];
    fft(n,arr,true);
    for(int i=0;i<sum;i++) ans[i]=(long long int)(arr[i].real() / 4 + 0.5);
}

long long a[MAXN];
long long b[MAXN];
long long ans[MAXN];
int a_length;
int b_length;
```

## 8.2 Pollard’s rho

```
ll add(ll x, ll y, ll p) {
    return (x + y) % p;
}
ll qMul(ll x, ll y, ll mod) {
    ll ret = x * y - (ll)((long double)x / mod * y) * mod;
    return ret < 0 ? ret + mod : ret;
}
ll f(ll x, ll mod) { return add(qMul(x, x, mod), 1, mod); }
ll pollard_rho(ll n) {
    if (!(n & 1)) return 2;
    while (true) {
        ll y = 2, x = rand() % (n - 1) + 1, res = 1;
        for (int sz = 2; res == 1; sz *= 2) {
            for (int i = 0; i < sz && res <= 1; i++) {
                x = f(x, n);
                res = __gcd(llabs(x - y), n);
            }
            y = x;
        }
        if (res != 0 && res != n) return res;
    }
}
vector<ll> ret;
void fact(ll x) {
    if (miller_rabin(x)) {
        ret.push_back(x);
        return;
    }
    ll f = pollard_rho(x);
    fact(f);
    fact(x / f);
}
```

## 8.3 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 :  pirmes <= 13
// n < 2^64                       7 :
// 2, 325, 9375, 28178, 450775, 9780504, 1795265022
bool witness(ll a,ll n,ll u,int t){
    if(!(a%=n)) return 0;
    ll x=mypow(a,u,n);
    for(int i=0;i<t;i++) {
```

```cpp
        ll nx=mul(x,x,n);
        if(nx==1&&x!=1&&x!=n-1) return 1;
        x=nx;
    }
    return x!=1;
}
bool miller_rabin(ll n,int s=100) {
    // iterate s times of witness on n
    // return 1 if prime, 0 otherwise
    if(n<2) return 0;
    if(!(n&1)) return n == 2;
    ll u=n-1; int t=0;
    while(!(u&1)) u>>=1, t++;
    while(s--){
        ll a=randll()%(n-1)+1;
        if(witness(a,n,u,t)) return 0;
    }
    return 1;
}
```

## 8.4 Fast Power

Note: $a^n \equiv a^{(n\ mod\ (p-1))} (mod\ p)$

## 8.5 Extend GCD

```cpp
ll GCD;
pll extgcd(ll a, ll b) {
    if (b == 0) {
        GCD = a;
        return pll{1, 0};
    }
    pll ans = extgcd(b, a % b);
    return pll{ans.S, ans.F - a / b * ans.S};
}
pll bezout(ll a, ll b, ll c) {
    bool negx = (a < 0), negy = (b < 0);
    pll ans = extgcd(abs(a), abs(b));
    if (c % GCD != 0) return pll{-LLINF, -LLINF};
    return pll{ans.F * c / GCD * (negx ? -1 : 1),
               ans.S * c / GCD * (negy ? -1 : 1)};
}
ll inv(ll a, ll p) {
    if (p == 1) return -1;
    pll ans = bezout(a % p, -p, 1);
    if (ans == pll{-LLINF, -LLINF}) return -1;
    return (ans.F % p + p) % p;
}
```

## 8.6 Mu + Phi

```cpp
const int maxn = 1e6 + 5;
ll f[maxn];
vector<int> lpf, prime;
void build() {
lpf.clear(); lpf.resize(maxn, 1);
prime.clear();
f[1] = ...;  /* mu[1] = 1, phi[1] = 1 */
for (int i = 2; i < maxn; i++) {
    if (lpf[i] == 1) {
        lpf[i] = i; prime.emplace_back(i);
        f[i] = ...;  /* mu[i] = 1, phi[i] = i-1 */
    }
    for (auto& j : prime) {
        if (i*j >= maxn) break;
        lpf[i*j] = j;
        if (i % j == 0) f[i*j] = ...;  /* 0, phi[i]*j
            */
        else f[i*j] = ...;  /* -mu[i], phi[i]*phi[j] */
        if (j >= lpf[i]) break;
} } }
```

## 8.7 Other Formulas

- Inversion:
  $aa^{-1} \equiv 1 \pmod{m}$. $a^{-1}$ exists iff gcd$(a, m) = 1$.

- Linear inversion:
  $a^{-1} \equiv (m - \lfloor \frac{m}{a} \rfloor) \times (m \bmod a)^{-1} \pmod{m}$

- Fermat's little theorem:
  $a^p \equiv a \pmod{p}$ if $p$ is prime.

- Euler function:
  $\phi(n) = n \prod_{p|n} \frac{p-1}{p}$

- Euler theorem:
  $a^{\phi(n)} \equiv 1 \pmod{n}$ if gcd$(a, n) = 1$.

- Extended Euclidean algorithm:
  $ax + by =$ gcd$(a, b) =$ gcd$(b, a \bmod b) =$ gcd$(b, a - \lfloor \frac{a}{b} \rfloor b) = bx_1 + (a - \lfloor \frac{a}{b} \rfloor b)y_1 = ay_1 + b(x_1 - \lfloor \frac{a}{b} \rfloor y_1)$

- Divisor function:
  $\sigma_x(n) = \sum_{d|n} d^x$. $n = \prod_{i=1}^r p_i^{a_i}$.
  $\sigma_x(n) = \prod_{i=1}^r \frac{p_i^{(a_i+1)x} - 1}{p_i^x - 1}$ if $x \neq 0$. $\sigma_0(n) = \prod_{i=1}^r (a_i + 1)$.

- Chinese remainder theorem (Coprime Moduli):
  $x \equiv a_i \pmod{m_i}$.
  $M = \prod m_i$. $M_i = M/m_i$. $t_i = M_i^{-1}$.
  $x = kM + \sum a_i t_i M_i$, $k \in \mathbb{Z}$.

- Chinese remainder theorem:
  $x \equiv a_1 \pmod{m_1}, x \equiv a_2 \pmod{m_2} \Rightarrow x = m_1 p + a_1 = m_2 q + a_2 \Rightarrow m_1 p - m_2 q = a_2 - a_1$
  Solve for $(p, q)$ using ExtGCD.
  $x \equiv m_1 p + a_1 \equiv m_2 q + a_2 \pmod{lcm(m_1, m_2)}$

- Avoiding Overflow: $ca \mod cb = c(a \mod b)$

- Dirichlet Convolution: $(f * g)(n) = \sum_{d|n} f(n)g(n/d)$

- Important Multiplicative Functions + Proterties:
  1. $\epsilon(n) = [n = 1]$
  2. $1(n) = 1$
  3. $id(n) = n$
  4. $\mu(n) = 0$ if $n$ has squared prime factor
  5. $\mu(n) = (-1)^k$ if $n = p_1 p_2 \cdots p_k$
  6. $\epsilon = \mu * 1$
  7. $\phi = \mu * id$
  8. $[n = 1] = \sum_{d|n} \mu(d)$
  9. $[gcd = 1] = \sum_{d|gcd} \mu(d)$

- Möbius inversion: $f = g * 1 \Leftrightarrow g = f * \mu$

## 8.8 Polynomial

```cpp
const int maxk = 20;
const int maxn = 1<<maxk;
const ll LINF = 1e18;

/* P = r*2^k + 1
P                   r   k   g
998244353           119 23  3
1004535809          479 21  3

P                   r   k   g
3                   1   1   2
5                   1   2   2
17                  1   4   3
97                  3   5   5
193                 3   6   5
257                 1   8   3
7681                15  9   17
12289               3   12  11
40961               5   13  3
65537               1   16  3
786433              3   18  10
5767169             11  19  3
7340033             7   20  3
23068673            11  21  3
104857601           25  22  3
```

```
167772161           5   25  3
469762049           7   26  3
1004535809          479 21  3
2013265921          15  27  31
2281701377          17  27  3
3221225473          3   30  5
75161927681         35  31  3
77309411329         9   33  7
206158430209        3   36  22
2061584302081       15  37  7
2748779069441       5   39  3
6597069766657       3   41  5
39582418599937      9   42  5
79164837199873      9   43  5
263882790666241     15  44  7
1231453023109121    35  45  3
1337006139375617    19  46  3
3799912185593857    27  47  5
4222124650659841    15  48  19
7881299347898369    7   50  6
31525197391593473   7   52  3
180143985094819841  5   55  6
1945555039024054273 27  56  5
4179340454199820289 29  57  3
9097271247288401921 505 54  6 */

const int g = 3;
const ll MOD = 998244353;

ll pw(ll a, ll n) { /* fast pow */ }

#define siz(x) (int)x.size()

template<typename T>
vector<T>& operator+=(vector<T>& a, const vector<T>& b) {
    if (siz(a) < siz(b)) a.resize(siz(b));
    for (int i = 0; i < min(siz(a), siz(b)); i++) {
        a[i] += b[i];
        a[i] -= a[i] >= MOD ? MOD : 0;
    }
    return a;
}

template<typename T>
vector<T>& operator-=(vector<T>& a, const vector<T>& b) {
    if (siz(a) < siz(b)) a.resize(siz(b));
    for (int i = 0; i < min(siz(a), siz(b)); i++) {
        a[i] -= b[i];
        a[i] += a[i] < 0 ? MOD : 0;
    }
    return a;
}

template<typename T>
vector<T> operator-(const vector<T>& a) {
    vector<T> ret(siz(a));
    for (int i = 0; i < siz(a); i++) {
        ret[i] = -a[i] < 0 ? -a[i] + MOD : -a[i];
    }
    return ret;
}

vector<ll> X, iX;
vector<int> rev;

void init_ntt() {
    X.clear(); X.resize(maxn, 1);  // x1 = g^((p-1)/n)
    iX.clear(); iX.resize(maxn, 1);

    ll u = pw(g, (MOD-1)/maxn);
    ll iu = pw(u, MOD-2);

    for (int i = 1; i < maxn; i++) {
        X[i] = X[i-1] * u;
        iX[i] = iX[i-1] * iu;
        if (X[i] >= MOD) X[i] %= MOD;
        if (iX[i] >= MOD) iX[i] %= MOD;
    }

    rev.clear(); rev.resize(maxn, 0);
    for (int i = 1, hb = -1; i < maxn; i++) {
        if (!(i & (i-1))) hb++;
        rev[i] = rev[i ^ (1<<hb)] | (1<<(maxk-hb-1));
} }

template<typename T>
void NTT(vector<T>& a, bool inv=false) {

    int _n = (int)a.size();
    int k = __lg(_n) + ((1<<__lg(_n)) != _n);
    int n = 1<<k;
    a.resize(n, 0);

    short shift = maxk-k;
    for (int i = 0; i < n; i++)
        if (i > (rev[i]>>shift))
            swap(a[i], a[rev[i]>>shift]);

    for (int len = 2, half = 1, div = maxn>>1; len <= n; len<<=1, half<<=1, div>>=1) {
        for (int i = 0; i < n; i += len) {
            for (int j = 0; j < half; j++) {
                T u = a[i+j];
                T v = a[i+j+half] * (inv ? iX[j*div] : X[j*div]) % MOD;
                a[i+j] = (u+v >= MOD ? u+v-MOD : u+v);
                a[i+j+half] = (u-v < 0 ? u-v+MOD : u-v);
    } } }

    if (inv) {
        T dn = pw(n, MOD-2);
        for (auto& x : a) {
            x *= dn;
            if (x >= MOD) x %= MOD;
} } }

template<typename T>
inline void resize(vector<T>& a) {
    int cnt = (int)a.size();
    for (; cnt > 0; cnt--) if (a[cnt-1]) break;
    a.resize(max(cnt, 1));
}

template<typename T>
vector<T>& operator*=(vector<T>& a, vector<T> b) {
    int na = (int)a.size();
    int nb = (int)b.size();
    a.resize(na + nb - 1, 0);
    b.resize(na + nb - 1, 0);

    NTT(a); NTT(b);
    for (int i = 0; i < (int)a.size(); i++) {
        a[i] *= b[i];
        if (a[i] >= MOD) a[i] %= MOD;
    }
    NTT(a, true);

    resize(a);
    return a;
}

template<typename T>
void inv(vector<T>& ia, int N) {
    vector<T> _a(move(ia));
    ia.resize(1, pw(_a[0], MOD-2));
    vector<T> a(1, -_a[0] + (-_a[0] < 0 ? MOD : 0));

    for (int n = 1; n < N; n<<=1) {
        // n -> 2*n
        // ia' = ia(2-a*ia);

        for (int i = n; i < min(siz(_a), (n<<1)); i++)
            a.emplace_back(-_a[i] + (-_a[i] < 0 ? MOD : 0));

        vector<T> tmp = ia;
        ia *= a;
        ia.resize(n<<1);
        ia[0] = ia[0] + 2 >= MOD ? ia[0] + 2 - MOD : ia[0] + 2;
        ia *= tmp;
        ia.resize(n<<1);
    }
    ia.resize(N);
}

template<typename T>
void mod(vector<T>& a, vector<T>& b) {
    int n = (int)a.size()-1, m = (int)b.size()-1;
    if (n < m) return;

    vector<T> ra = a, rb = b;
    reverse(ra.begin(), ra.end()); ra.resize(min(n+1, n-m+1));
    reverse(rb.begin(), rb.end()); rb.resize(min(m+1, n-m+1));

    inv(rb, n-m+1);

    vector<T> q = move(ra);
    q *= rb;
    q.resize(n-m+1);
    reverse(q.begin(), q.end());

    q *= b;
    a -= q;
    resize(a);
}

/* Kitamasa Method (Fast Linear Recurrence):
Find a[K] (Given a[j] = c[0]a[j-N] + ... + c[N-1]a[j-1])
Let B(x) = x^N - c[N-1]x^(N-1) - ... - c[1]x^1 - c[0]
Let R(x) = x^K mod B(x)   (get x^K using fast pow and use poly mod to get R(x))
Let r[i] = the coefficient of x^i in R(x)
=> a[K] = a[0]r[0] + a[1]r[1] + ... + a[N-1]r[N-1] */
```

# 9 Linear Algebra

## 9.1 Gaussian-Jordan Elimination

```
int n;
vector<vector<ll>> v;
void gauss(vector<vector<ll>>& v) {
    int r = 0;
    for (int i = 0; i < n; i++) {
        bool ok = false;
        for (int j = r; j < n; j++) {
            if (v[j][i] == 0) continue;
            swap(v[j], v[r]);
            ok = true;
            break;
        }
        if (!ok) continue;
        ll div = inv(v[r][i]);
        for (int j = 0; j < n + 1; j++) {
            v[r][j] *= div;
            if (v[r][j] >= MOD) v[r][j] %= MOD;
        }
        for (int j = 0; j < n; j++) {
            if (j == r) continue;
            ll t = v[j][i];
            for (int k = 0; k < n + 1; k++) {
                v[j][k] -= v[r][k] * t % MOD;
                if (v[j][k] < 0) v[j][k] += MOD;
            }
        }
        r++;
    }
}
```

## 9.2 Determinant

1. Use GJ Elimination, if there's any row consists of only 0, then det = 0, otherwise det = product of diagonal elements.
2. Properties of det:
   - Transpose: Unchanged
   - Row Operation 1 - Swap 2 rows: $-det$
   - Row Operation 2 - $k\overrightarrow{r_i}$: $k \times det$
   - Row Operation 3 - $k\overrightarrow{r_i}$ add to $\overrightarrow{r_j}$: Unchaged

# 10 Combinatorics

## 10.1 Catalan Number

$$C_0 = 1, C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i}, C_n = C_n^{2n} - C_{n-1}^{2n}$$

| | | | | |
|---|---|---|---|---|
| 0 | 1 | 1 | 2 | 5 |
| 4 | 14 | 42 | 132 | 429 |
| 8 | 1430 | 4862 | 16796 | 58786 |
| 12 | 208012 | 742900 | 2674440 | 9694845 |

## 10.2 Burnside's Lemma

Let $X$ be the original set.
Let $G$ be the group of operations acting on $X$.
Let $X^g$ be the set of $x$ not affected by $g$.
Let $X/G$ be the set of orbits.
Then the following equation holds:

$$|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|$$

# 11 Special Numbers

## 11.1 Fibonacci Series

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 2 | 3 |
| 5 | 5 | 8 | 13 | 21 |
| 9 | 34 | 55 | 89 | 144 |
| 13 | 233 | 377 | 610 | 987 |
| 17 | 1597 | 2584 | 4181 | 6765 |
| 21 | 10946 | 17711 | 28657 | 46368 |
| 25 | 75025 | 121393 | 196418 | 317811 |
| 29 | 514229 | 832040 | 1346269 | 2178309 |
| 33 | 3524578 | 5702887 | 9227465 | 14930352 |

$f(45) \approx 10^9, f(88) \approx 10^{18}$

## 11.2 Prime Numbers

- First 50 prime numbers:

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 5 | 7 | 11 |
| 6 | 13 | 17 | 19 | 23 | 29 |
| 11 | 31 | 37 | 41 | 43 | 47 |
| 16 | 53 | 59 | 61 | 67 | 71 |
| 21 | 73 | 79 | 83 | 89 | 97 |
| 26 | 101 | 103 | 107 | 109 | 113 |
| 31 | 127 | 131 | 137 | 139 | 149 |
| 36 | 151 | 157 | 163 | 167 | 173 |
| 41 | 179 | 181 | 191 | 193 | 197 |
| 46 | 199 | 211 | 223 | 227 | 229 |

- Very large prime numbers:

| | | |
|---|---|---|
| 1000001333 | 1000500889 | 2500001909 |
| 2000000659 | 900004151 | 850001359 |

- $\pi(n) \equiv$ Number of primes $\le n \approx n/((\ln n) - 1)$
  $\pi(100) = 25, \pi(200) = 46$
  $\pi(500) = 95, \pi(1000) = 168$

$\pi(2000) = 303, \pi(4000) = 550$
$\pi(10^4) = 1229, \pi(10^5) = 9592$
$\pi(10^6) = 78498, \pi(10^7) = 664579$