

Contents

1	Reminder	1
1.1	Bug List	1
1.2	OwO	1
2	Basic	1
2.1	Vimrc	1
2.2	Runcpp.sh	1
2.3	PBDS	1
2.4	Random	1
3	Data Structure	1
3.1	BIT	1
3.2	DSU	2
3.3	Segment Tree	2
3.4	Treap	2
3.5	Persistent Treap	2
3.6	Li Chao Tree	3
3.7	Sparse Table	3
3.8	Time Segment Tree	3
4	Flow / Matching	4
4.1	Dinic	4
4.2	MCMF	4
4.3	KM	4
4.4	Hopcroft-Karp	5
4.5	Blossom	5
4.6	Weighted Blossom	6
4.7	Cover / Independent Set	7
5	Graph	7
5.1	Heavy-Light Decomposition	7
5.2	Centroid Decomposition	8
5.3	Bellman-Ford + SPFA	8
5.4	BCC - AP	9
5.5	BCC - Bridge	10
5.6	SCC - Tarjan	10
5.7	SCC - Kosaraju	11
5.8	Eulerian Path - Undir	11
5.9	Eulerian Path - Dir	11
5.10	Hamilton Path	11
5.11	Kth Shortest Path	12
5.12	System of Difference Constraints	13
6	String	13
6.1	Aho Corasick	13
6.2	KMP	13
6.3	Z Value	13
6.4	Manacher	14
6.5	Suffix Array	14
6.6	Minimum Rotation	14
6.7	Lyndon Factorization	14

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	15
7.5	Polygon Area	15
7.6	Convex Hull	15
7.7	Point In Convex	16
7.8	Point Segment Distance	16
7.9	Point in Polygon	16
7.10	Lower Concave Hull	16
7.11	Pick's Theorem	16
7.12	Vector In Polygon	16
7.13	Minkowski Sum	16
7.14	Rotating SweepLine	17
7.15	Half Plane Intersection	17
7.16	Minimum Enclosing Circle	18
7.17	Heart	18
7.18	Tangents	18
7.19	Point In Circle	18
7.20	Union of Circles	18
7.21	Union of Polygons	18
7.22	Delaunay Triangulation	18
7.23	Triangulation Voronoi	18
7.24	External Bisector	18
7.25	Intersection Area of Polygon and Circle	18
7.26	3D Point	18
7.27	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	19
8.5	Extend GCD	19
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

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

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

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

```
#!/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 PBDS

```
#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
```

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!

```

15 gp_hash_table<int, int> ht;
16 ht.find(element);
17 ht.insert({key, value});
18 ht.erase(element);
19
20 // priority queue
21 __gnu_pbds::priority_queue<int, less<int>> big_q;
22 // Big First
23 __gnu_pbds::priority_queue<int, greater<int>> small_q;
24 // Small First
25 q1.join(q2); // join

```

2.4 Random

```

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

```

3 Data Structure

3.1 BIT

```

1 struct BIT {
2     int n;
3     long long bit[N];
4
5     void init(int x, vector<long long> &a) {
6         n = x;
7         for (int i = 1, j; i <= n; i++) {
8             bit[i] += a[i - 1], j = i + (i & -i);
9             if (j <= n) bit[j] += bit[i];
10        }
11    }
12
13    void update(int x, long long dif) {
14        while (x <= n) bit[x] += dif, x += x & -x;
15    }
16
17    long long query(int l, int r) {
18        if (l != 1) return query(1, r) - query(1, l -
19            1);
20
21        long long ret = 0;
22        while (l <= r) ret += bit[r], r -= r & -r;
23        return ret;
24    }
25 } bm;

```

3.2 DSU

```

1 struct DSU {
2     int h[N], s[N];
3
4     void init(int n) { iota(h, h + n + 1, 0), fill(s, s
5         + n + 1, 1); }
6
7     int fh(int x) { return (h[x] == x ? x : h[x] = fh(h
8         [x])); }
9
10    bool mer(int x, int y) {
11        x = fh(x), y = fh(y);
12        if (x == y) return 0;
13        if (s[x] < s[y]) swap(x, y);
14        s[x] += s[y], s[y] = 0;
15        h[y] = x;
16        return 1;
17    }
18 } bm;

```

3.3 Segment Tree

```

1 struct segtree {
2     int n, seg[1 << 19];
3
4     void init(int x) {

```

```

5         n = 1 << (___lg(x) + 1);
6         for (int i = 1; i < 2 * n; i++)
7             seg[i] = inf;
8     }
9
10    void update(int x, int val) {
11        x += n;
12        seg[x] = val, x /= 2;
13        while (x)
14            seg[x] = min(seg[2 * x], seg[2 * x + 1]), x
15                /= 2;
16    }
17
18    int query(int l, int r) {
19        l += n, r += n;
20        int ret = inf;
21        while (l < r) {
22            if (l & 1)
23                ret = min(ret, seg[l++]);
24            if (r & 1)
25                ret = min(ret, seg[--r]);
26            l /= 2, r /= 2;
27        }
28        return ret;
29    }
30 } bm;

```

3.4 Treap

```

1 mt19937 rng(random_device{}());
2 struct Treap {
3     Treap *l, *r;
4     int val, num, pri;
5     Treap(int k) {
6         l = r = NULL;
7         val = k;
8         num = 1;
9         pri = rng();
10    }
11
12    int siz(Treap *now) { return now ? now->num : 0; }
13    void pull(Treap *&now) {
14        now->num = siz(now->l) + siz(now->r) + 1;
15    }
16    Treap *merge(Treap *a, Treap *b) {
17        if (!a || !b)
18            return a ? a : b;
19        else if (a->pri > b->pri) {
20            a->r = merge(a->r, b);
21            pull(a);
22            return a;
23        } else {
24            b->l = merge(a, b->l);
25            pull(b);
26            return b;
27        }
28    }
29    void split_size(Treap *rt, Treap *&a, Treap *&b, int
30        val) {
31        if (!rt) {
32            a = b = NULL;
33            return;
34        }
35        if (siz(rt->l) + 1 > val) {
36            b = rt;
37            split_size(rt->l, a, b->l, val);
38            pull(b);
39        } else {
40            a = rt;
41            split_size(rt->r, a->r, b, val - siz(a->l) - 1);
42            pull(a);
43        }
44    }
45    void split_val(Treap *rt, Treap *&a, Treap *&b, int val) {
46        if (!rt) {
47            a = b = NULL;
48            return;
49        }
50        if (rt->val <= val) {

```

```

50     a = rt;
51     split_val(rt->r, a->r, b, val);
52     pull(a);
53 } else {
54     b = rt;
55     split_val(rt->l, a, b->l, val);
56     pull(b);
57 }
58 }
59 void treap_dfs(Treap *now) {
60     if (!now) return;
61     treap_dfs(now->l);
62     cout << now->val << " ";
63     treap_dfs(now->r);
64 }

```

3.5 Persistent Treap

```

1 struct node {
2     node *l, *r;
3     char c;
4     int v, sz;
5     node(char x = '$') : c(x), v(mt()), sz(1) {
6         l = r = nullptr;
7     }
8     node(node* p) { *this = *p; }
9     void pull() {
10         sz = 1;
11         for (auto i : {l, r})
12             if (i) sz += i->sz;
13     }
14 } arr[maxn], *ptr = arr;
15 inline int size(node* p) { return p ? p->sz : 0; }
16 node* merge(node* a, node* b) {
17     if (!a || !b) return a ? b;
18     if (a->v < b->v) {
19         node* ret = new (ptr++) node(a);
20         ret->r = merge(ret->r, b), ret->pull();
21         return ret;
22     } else {
23         node* ret = new (ptr++) node(b);
24         ret->l = merge(a, ret->l), ret->pull();
25         return ret;
26     }
27 }
28 P<node*> split(node* p, int k) {
29     if (!p) return {nullptr, nullptr};
30     if (k >= size(p->l) + 1) {
31         auto [a, b] = split(p->r, k - size(p->l) - 1);
32         node* ret = new (ptr++) node(p);
33         ret->r = a, ret->pull();
34         return {ret, b};
35     } else {
36         auto [a, b] = split(p->l, k);
37         node* ret = new (ptr++) node(p);
38         ret->l = b, ret->pull();
39         return {a, ret};
40     }
41 }

```

3.6 Li Chao Tree

```

1 constexpr int maxn = 5e4 + 5;
2 struct line {
3     ld a, b;
4     ld operator()(ld x) { return a * x + b; }
5 } arr[(maxn + 1) << 2];
6 bool operator<(line a, line b) { return a.a < b.a; }
7 #define m ((l + r) >> 1)
8 void insert(line x, int i = 1, int l = 0, int r = maxn) {
9     if (r - l == 1) {
10         if (x(l) > arr[i](l))
11             arr[i] = x;
12         return;
13     }
14     line a = max(arr[i], x), b = min(arr[i], x);
15     if (a(m) > b(m))
16         arr[i] = a, insert(b, i << 1, l, m);
17     else

```

```

18         arr[i] = b, insert(a, i << 1 | 1, m, r);
19 }
20 ld query(int x, int i = 1, int l = 0, int r = maxn) {
21     if (x < l || r <= x) return -numeric_limits<ld>::
22         max();
23     if (r - l == 1) return arr[i](x);
24     return max({arr[i](x), query(x, i << 1, l, m),
25         query(x, i << 1 | 1, m, r)});
26 }
27 #undef m

```

3.7 Sparse Table

```

1 const int lgmx = 19;
2
3 int n, q;
4 int spt[lgmx][maxn];
5
6 void build() {
7     FOR(k, 1, lgmx, 1) {
8         for (int i = 0; i + (1 << k) - 1 < n; i++) {
9             spt[k][i] = min(spt[k - 1][i], spt[k - 1][i
10                 + (1 << (k - 1))]);
11         }
12     }
13 }
14
15 int query(int l, int r) {
16     int ln = len(l, r);
17     int lg = __lg(ln);
18     return min(spt[lg][l], spt[lg][r - (1 << lg) + 1]);
19 }

```

3.8 Time Segment Tree

```

1 constexpr int maxn = 1e5 + 5;
2 V<P<int>> arr[(maxn + 1) << 2];
3 V<int> dsu, sz;
4 V<tuple<int, int, int>> his;
5 int cnt, q;
6 int find(int x) {
7     return x == dsu[x] ? x : find(dsu[x]);
8 }
9 inline bool merge(int x, int y) {
10     int a = find(x), b = find(y);
11     if (a == b) return false;
12     if (sz[a] > sz[b]) swap(a, b);
13     his.emplace_back(a, b, sz[b]), dsu[a] = b, sz[b] +=
14         sz[a];
15     return true;
16 }
17 inline void undo() {
18     auto [a, b, s] = his.back();
19     his.pop_back();
20     dsu[a] = a, sz[b] = s;
21 }
22 #define m ((l + r) >> 1)
23 void insert(int ql, int qr, P<int> x, int i = 1, int l
24     = 0, int r = q) {
25     // debug(ql, qr, x); return;
26     if (qr <= l || r <= ql) return;
27     if (ql <= l && r <= qr) {
28         arr[i].push_back(x);
29         return;
30     }
31     if (qr <= m)
32         insert(ql, qr, x, i << 1, l, m);
33     else if (m <= ql)
34         insert(ql, qr, x, i << 1 | 1, m, r);
35     else {
36         insert(ql, qr, x, i << 1, l, m);
37         insert(ql, qr, x, i << 1 | 1, m, r);
38     }
39 }
40 void traversal(V<int>& ans, int i = 1, int l = 0, int r
41     = q) {
42     int opcnt = 0;
43     // debug(i, l, r);
44     for (auto [a, b] : arr[i])
45         if (merge(a, b))

```

```

43         opcnt++, cnt--;
44     if (r - 1 == 1)
45         ans[l] = cnt;
46     else {
47         traversal(ans, i << 1, l, m);
48         traversal(ans, i << 1 | 1, m, r);
49     }
50     while (opcnt--)
51         undo(), cnt++;
52     arr[i].clear();
53 }
54 #undef m
55 inline void solve() {
56     int n, m;
57     cin >> n >> m >> q, q++;
58     dsu.resize(cnt = n), sz.assign(n, 1);
59     iota(dsu.begin(), dsu.end(), 0);
60     // a, b, time, operation
61     unordered_map<ll, V<int>> s;
62     for (int i = 0; i < m; i++) {
63         int a, b;
64         cin >> a >> b;
65         if (a > b) swap(a, b);
66         s[(((ll)a << 32) | b).emplace_back(0);
67     }
68     for (int i = 1; i < q; i++) {
69         int op, a, b;
70         cin >> op >> a >> b;
71         if (a > b) swap(a, b);
72         switch (op) {
73             case 1:
74                 s[(((ll)a << 32) | b).push_back(i);
75                 break;
76             case 2:
77                 auto tmp = s[(((ll)a << 32) | b).back();
78                 s[(((ll)a << 32) | b).pop_back();
79                 insert(tmp, i, P<int>>{a, b});
80         }
81     }
82     for (auto [p, v] : s) {
83         int a = p >> 32, b = p & -1;
84         while (v.size()) {
85             insert(v.back(), q, P<int>>{a, b});
86             v.pop_back();
87         }
88     }
89     V<int> ans(q);
90     traversal(ans);
91     for (auto i : ans)
92         cout << i << ' ';
93     cout << endl;
94 }

```

4 Flow / Matching

4.1 Dinic

```

1 struct Dinic {
2     int n, s, t, level[N], iter[N];
3     struct edge {
4         int to, cap, rev;
5     };
6     vector<edge> path[N];
7     void init(int _n, int _s, int _t) {
8         n = _n, s = _s, t = _t;
9         FOR(i, 0, n + 1)
10             path[i].clear();
11     }
12     void add(int a, int b, int c) {
13         edge now;
14         now.to = b, now.cap = c, now.rev = sz(path[b]);
15         path[a].pb(now);
16         now.to = a, now.cap = 0, now.rev = sz(path[a]) - 1;
17         path[b].pb(now);
18     }
19     void bfs() {
20         memset(level, -1, sizeof(level));
21         level[s] = 0;
22         queue<int> q;

```

```

23         q.push(s);
24         while (q.size()) {
25             int now = q.front();
26             q.pop();
27             for (edge e : path[now]) {
28                 if (e.cap > 0 && level[e.to] == -1) {
29                     level[e.to] = level[now] + 1;
30                     q.push(e.to);
31                 }
32             }
33         }
34     }
35     int dfs(int now, int flow) {
36         if (now == t) return flow;
37         for (int &i = iter[now]; i < sz(path[now]); i++) {
38             edge &e = path[now][i];
39             if (e.cap > 0 && level[e.to] == level[now] + 1) {
40                 int res = dfs(e.to, min(flow, e.cap));
41                 if (res > 0) {
42                     e.cap -= res;
43                     path[e.to][e.rev].cap += res;
44                     return res;
45                 }
46             }
47         }
48         return 0;
49     }
50     int dinic() {
51         int res = 0;
52         while (true) {
53             bfs();
54             if (level[t] == -1) break;
55             memset(iter, 0, sizeof(iter));
56             int now = 0;
57             while ((now = dfs(s, INF)) > 0) res += now;
58         }
59         return res;
60     }
61 };

```

4.2 MCMF

```

1 struct MCMF {
2     int n, s, t, par[N + 5], p_i[N + 5], dis[N + 5],
3     vis[N + 5];
4     struct edge {
5         int to, cap, rev, cost;
6     };
7     vector<edge> path[N];
8     void init(int _n, int _s, int _t) {
9         n = _n, s = _s, t = _t;
10        FOR(i, 0, 2 * n + 5)
11            par[i] = p_i[i] = vis[i] = 0;
12    }
13    void add(int a, int b, int c, int d) {
14        path[a].pb({b, c, sz(path[b]), d});
15        path[b].pb({a, 0, sz(path[a]) - 1, -d});
16    }
17    void spfa() {
18        FOR(i, 0, n * 2 + 5)
19            dis[i] = INF,
20            vis[i] = 0;
21        dis[s] = 0;
22        queue<int> q;
23        q.push(s);
24        while (!q.empty()) {
25            int now = q.front();
26            q.pop();
27            vis[now] = 0;
28            for (int i = 0; i < sz(path[now]); i++) {
29                edge e = path[now][i];
30                if (e.cap > 0 && dis[e.to] > dis[now] + e.cost) {
31                    dis[e.to] = dis[now] + e.cost;
32                    par[e.to] = now;
33                    p_i[e.to] = i;
34                    if (vis[e.to] == 0) {
35                        vis[e.to] = 1;
36                        q.push(e.to);

```

```

36         }
37     }
38 }
39 }
40 }
41 pii flow() {
42     int flow = 0, cost = 0;
43     while (true) {
44         spfa();
45         if (dis[t] == INF)
46             break;
47         int mn = INF;
48         for (int i = t; i != s; i = par[i])
49             mn = min(mn, path[par[i]][p_i[i]].cap);
50         flow += mn;
51         cost += dis[t] * mn;
52         for (int i = t; i != s; i = par[i]) {
53             edge &now = path[par[i]][p_i[i]];
54             now.cap -= mn;
55             path[i][now.rev].cap += mn;
56         }
57     }
58     return mp(flow, cost);
59 }
60 };

```

```

54         return;
55     }
56     vy[y] = 1;
57     q.push(my[y]);
58 }
59 }
60 }
61 }
62 int solve() {
63     fill(mx, mx + n + 1, 0);
64     fill(my, my + n + 1, 0);
65     fill(ly, ly + n + 1, 0);
66     fill(lx, lx + n + 1, 0);
67     FOR(x, 1, n + 1)
68         FOR(y, 1, n + 1)
69             lx[x] = max(lx[x], g[x][y]);
70     FOR(x, 1, n + 1)
71         bfs(x);
72     int ans = 0;
73     FOR(y, 1, n + 1)
74         ans += g[my[y]][y];
75     return ans;
76 }
77 };

```

4.3 KM

```

1 struct KM {
2     int n, mx[1005], my[1005], pa[1005];
3     int g[1005][1005], lx[1005], ly[1005], sy[1005];
4     bool vx[1005], vy[1005];
5     void init(int _n) {
6         n = _n;
7         FOR(i, 1, n + 1)
8             fill(g[i], g[i] + 1 + n, 0);
9     }
10    void add(int a, int b, int c) { g[a][b] = c; }
11    void augment(int y) {
12        for (int x, z; y; y = z)
13            x = pa[y], z = mx[x], my[y] = x, mx[x] = y;
14    }
15    void bfs(int st) {
16        FOR(i, 1, n + 1)
17            sy[i] = INF;
18        vx[st] = vy[st] = 0;
19        queue<int> q;
20        q.push(st);
21        for (;;) {
22            while (!q.empty()) {
23                int x = q.front();
24                q.pop();
25                vx[x] = 1;
26                FOR(y, 1, n + 1)
27                    if (!vy[y]) {
28                        int t = lx[x] + ly[y] - g[x][y];
29                        if (t == 0) {
30                            pa[y] = x;
31                            if (!my[y]) {
32                                augment(y);
33                                return;
34                            }
35                            vy[y] = 1, q.push(my[y]);
36                        } else if (sy[y] > t)
37                            pa[y] = x, sy[y] = t;
38                    }
39            }
40            int cut = INF;
41            FOR(y, 1, n + 1)
42                if (!vy[y] && cut > sy[y]) cut = sy[y];
43            FOR(j, 1, n + 1) {
44                if (vx[j]) lx[j] -= cut;
45                if (vy[j])
46                    ly[j] += cut;
47                else
48                    sy[j] -= cut;
49            }
50            FOR(y, 1, n + 1) {
51                if (!vy[y] && sy[y] == 0) {
52                    if (!my[y]) {
53                        augment(y);

```

4.4 Hopcroft-Karp

```

1 struct HopcroftKarp {
2     // id: X = [1, nx], Y = [nx+1, nx+ny]
3     int n, nx, ny, m, MXCNT;
4     vector<vector<int>> g;
5     vector<int> mx, my, dis, vis;
6     void init(int nnx, int nny, int mm) {
7         nx = nnx, ny = nny, m = mm;
8         n = nx + ny + 1;
9         g.clear();
10        g.resize(n);
11    }
12    void add(int x, int y) {
13        g[x].emplace_back(y);
14        g[y].emplace_back(x);
15    }
16    bool dfs(int x) {
17        vis[x] = true;
18        Each(y, g[x]) {
19            int px = my[y];
20            if (px == -1 ||
21                (dis[px] == dis[x] + 1 &&
22                 !vis[px] && dfs(px))) {
23                mx[x] = y;
24                my[y] = x;
25                return true;
26            }
27        }
28        return false;
29    }
30    void get() {
31        mx.clear();
32        mx.resize(n, -1);
33        my.clear();
34        my.resize(n, -1);
35    }
36    while (true) {
37        queue<int> q;
38        dis.clear();
39        dis.resize(n, -1);
40        for (int x = 1; x <= nx; x++) {
41            if (mx[x] == -1) {
42                dis[x] = 0;
43                q.push(x);
44            }
45        }
46        while (!q.empty()) {
47            int x = q.front();
48            q.pop();
49            Each(y, g[x]) {
50                if (my[y] != -1 && dis[my[y]] ==
51                    -1) {
52                    dis[my[y]] = dis[x] + 1;
53                    q.push(my[y]);

```

```

54         }
55     }
56
57     bool brk = true;
58     vis.clear();
59     vis.resize(n, 0);
60     for (int x = 1; x <= nx; x++)
61         if (mx[x] == -1 && dfs(x))
62             brk = false;
63
64     if (brk) break;
65 }
66 MXCNT = 0;
67 for (int x = 1; x <= nx; x++)
68     if (mx[x] != -1) MXCNT++;
69 }
70 } hk;

```

4.5 Blossom

```

1  const int N=5e2+10;
2  struct Graph{
3      int to[N],bro[N],head[N],e;
4      int lnk[N],vis[N],stp,n;
5      void init(int _n){
6          stp=0;e=1;n=_n;
7          FOR(i,0,n+1)head[i]=lnk[i]=vis[i]=0;
8      }
9      void add(int u,int v){
10         to[e]=v,bro[e]=head[u],head[u]=e++;
11         to[e]=u,bro[e]=head[v],head[v]=e++;
12     }
13     bool dfs(int x){
14         vis[x]=stp;
15         for(int i=head[x];i=bro[i])
16         {
17             int v=to[i];
18             if(!lnk[v])
19             {
20                 lnk[x]=v;lnk[v]=x;
21                 return true;
22             }
23             else if(vis[lnk[v]]<stp)
24             {
25                 int w=lnk[v];
26                 lnk[x]=v,lnk[v]=x,lnk[w]=0;
27                 if(dfs(w))return true;
28                 lnk[w]=v,lnk[v]=w,lnk[x]=0;
29             }
30         }
31         return false;
32     }
33     int solve(){
34         int ans=0;
35         FOR(i,1,n+1){
36             if(!lnk[i]){
37                 stp++;
38                 ans+=dfs(i);
39             }
40         }
41         return ans;
42     }
43     void print_matching(){
44         FOR(i,1,n+1)
45             if(i<graph.lnk[i])
46                 cout<<i<<" "<<graph.lnk[i]<<endl;
47     }
48 };

```

4.6 Weighted Blossom

```

1  struct WeightGraph { // 1-based
2      static const int inf = INT_MAX;
3      static const int maxn = 514;
4      struct edge {
5          int u, v, w;
6          edge() {}
7          edge(int u, int v, int w) : u(u), v(v), w(w) {}
8      };
9      int n, n_x;

```

```

10     edge g[maxn * 2][maxn * 2];
11     int lab[maxn * 2];
12     int match[maxn * 2], slack[maxn * 2], st[maxn * 2],
13         pa[maxn * 2];
14     int flo_from[maxn * 2][maxn + 1], S[maxn * 2], vis[
15         maxn * 2];
16     vector<int> flo[maxn * 2];
17     queue<int> q;
18     int e_delta(const edge &e) { return lab[e.u] + lab[
19         e.v] - g[e.u][e.v].w * 2; }
20     void update_slack(int u, int x) {
21         if (!slack[x] || e_delta(g[u][x]) < e_delta(g[
22             slack[x]][x])) slack[x] = u;
23     }
24     void set_slack(int x) {
25         slack[x] = 0;
26         for (int u = 1; u <= n; ++u)
27             if (g[u][x].w > 0 && st[u] != x && S[st[u]]
28                 == 0)
29                 update_slack(u, x);
30     }
31     void q_push(int x) {
32         if (x <= n)
33             q.push(x);
34         else
35             for (size_t i = 0; i < flo[x].size(); i++)
36                 q_push(flo[x][i]);
37     }
38     void set_st(int x, int b) {
39         st[x] = b;
40         if (x > n)
41             for (size_t i = 0; i < flo[x].size(); ++i)
42                 set_st(flo[x][i], b);
43     }
44     int get_pr(int b, int xr) {
45         int pr = find(flo[b].begin(), flo[b].end(), xr)
46             - flo[b].begin();
47         if (pr % 2 == 1) {
48             reverse(flo[b].begin() + 1, flo[b].end());
49             return (int)flo[b].size() - pr;
50         }
51         return pr;
52     }
53     void set_match(int u, int v) {
54         match[u] = g[u][v].v;
55         if (u <= n) return;
56         edge e = g[u][v];
57         int xr = flo_from[u][e.u], pr = get_pr(u, xr);
58         for (int i = 0; i < pr; ++i) set_match(flo[u][i
59             ], flo[u][i ^ 1]);
60         set_match(xr, v);
61         rotate(flo[u].begin(), flo[u].begin() + pr, flo
62             [u].end());
63     }
64     void augment(int u, int v) {
65         for (;;) {
66             int xnv = st[match[u]];
67             set_match(u, v);
68             if (!xnv) return;
69             set_match(xnv, st[pa[xnv]]);
70             u = st[pa[xnv]], v = xnv;
71         }
72     }
73     int get_lca(int u, int v) {
74         static int t = 0;
75         for (++t; u || v; swap(u, v)) {
76             if (u == 0) continue;
77             if (vis[u] == t) return u;
78             vis[u] = t;
79             u = st[match[u]];
80             if (u) u = st[pa[u]];
81         }
82         return 0;
83     }
84     void add_blossom(int u, int lca, int v) {
85         int b = n + 1;
86         while (b <= n_x && st[b]) ++b;
87         if (b > n_x) ++n_x;
88         lab[b] = 0, S[b] = 0;
89         match[b] = match[lca];
90         flo[b].clear();
91         flo[b].push_back(lca);

```



```

82     for (int x = u, y; x != lca; x = st[pa[y]]) 151
83         flo[b].push_back(x), flo[b].push_back(y = 152
            st[match[x]]), q_push(y); 153
84     reverse(flo[b].begin() + 1, flo[b].end()); 154
85     for (int x = v, y; x != lca; x = st[pa[y]]) 155
86         flo[b].push_back(x), flo[b].push_back(y = 156
            st[match[x]]), q_push(y);
87     set_st(b, b); 157
88     for (int x = 1; x <= n_x; ++x) g[b][x].w = g[x] 158
        ][b].w = 0; 159
89     for (int x = 1; x <= n; ++x) flo_from[b][x] = 160
        0;
90     for (size_t i = 0; i < flo[b].size(); ++i) { 161
91         int xs = flo[b][i]; 162
92         for (int x = 1; x <= n_x; ++x)
93             if (g[b][x].w == 0 || e_delta(g[xs][x]) 163
                < e_delta(g[b][x])) 164
94                 g[b][x] = g[xs][x], g[x][b] = g[x] 165
                    xs]; 166
95         for (int x = 1; x <= n; ++x) 167
96             if (flo_from[xs][x]) flo_from[b][x] = 168
                xs; 169
97     } 170
98     set_slack(b); 171
99 } 172
100 void expand_blossom(int b) { 173
101     for (size_t i = 0; i < flo[b].size(); ++i) 174
102         set_st(flo[b][i], flo[b][i]); 175
103     int xr = flo_from[b][g[b][pa[b]].u], pr = 176
        get_pr(b, xr); 177
104     for (int i = 0; i < pr; i += 2) { 178
105         int xs = flo[b][i], xns = flo[b][i + 1]; 179
106         pa[xs] = g[xns][xs].u; 180
107         S[xs] = 1, S[xns] = 0;
108         slack[xs] = 0, set_slack(xns);
109         q_push(xns); 181
110     }
111     S[xr] = 1, pa[xr] = pa[b]; 182
112     for (size_t i = pr + 1; i < flo[b].size(); ++i) 183
113     { 184
114         int xs = flo[b][i]; 185
115         S[xs] = -1, set_slack(xs); 186
116     }
117     st[b] = 0; 187
118 } 188
119 bool on_found_edge(const edge &e) { 189
120     int u = st[e.u], v = st[e.v]; 190
121     if (S[v] == -1) { 191
122         pa[v] = e.u, S[v] = 1; 192
123         int nu = st[match[v]]; 193
124         slack[v] = slack[nu] = 0; 194
125         S[nu] = 0, q_push(nu); 195
126     } else if (S[v] == 0) { 196
127         int lca = get_lca(u, v); 197
128         if (!lca) 198
129             return augment(u, v), augment(v, u), 199
                true;
130         else 200
131             add_blossom(u, lca, v); 201
132     } 202
133     return false; 203
134 } 204
135 bool matching() { 205
136     memset(S + 1, -1, sizeof(int) * n_x); 206
137     memset(slack + 1, 0, sizeof(int) * n_x);
138     q = queue<int>(); 207
139     for (int x = 1; x <= n_x; ++x) 208
140         if (st[x] == x && !match[x]) pa[x] = 0, S[x] 209
            = 0, q_push(x); 210
141     if (q.empty()) return false; 211
142     for (;;) { 212
143         while (q.size()) { 213
144             int u = q.front();
145             q.pop();
146             if (S[st[u]] == 1) continue;
147             for (int v = 1; v <= n; ++v)
148                 if (g[u][v].w > 0 && st[u] != st[v] 1
                    ) { 2
149                     if (e_delta(g[u][v]) == 0) { 3
150                         if (on_found_edge(g[u][v])) 4
151                             return true;
152                     } else
153                         update_slack(u, st[v]);
154                 }
155             }
156         int d = inf;
157         for (int b = n + 1; b <= n_x; ++b)
158             if (st[b] == b && S[b] == 1) d = min(d,
159                 lab[b] / 2);
160         for (int x = 1; x <= n_x; ++x)
161             if (st[x] == x && slack[x]) {
162                 if (S[x] == -1)
163                     d = min(d, e_delta(g[slack[x]]
164                         [x]));
165                 else if (S[x] == 0)
166                     d = min(d, e_delta(g[slack[x]]
167                         [x]) / 2);
168             }
169         for (int u = 1; u <= n; ++u) {
170             if (S[st[u]] == 0) {
171                 if (lab[u] <= d) return 0;
172                 lab[u] -= d;
173             } else if (S[st[u]] == 1)
174                 lab[u] += d;
175         }
176         for (int b = n + 1; b <= n_x; ++b)
177             if (st[b] == b) {
178                 if (S[st[b]] == 0)
179                     lab[b] += d * 2;
180                 else if (S[st[b]] == 1)
181                     lab[b] -= d * 2;
182             }
183         q = queue<int>();
184         for (int x = 1; x <= n_x; ++x)
185             if (st[x] == x && slack[x] && st[slack[
186                 x]] != x && e_delta(g[slack[x]][x])
187                 == 0)
188                 if (on_found_edge(g[slack[x]][x]))
189                     return true;
190         for (int b = n + 1; b <= n_x; ++b)
191             if (st[b] == b && S[b] == 1 && lab[b]
192                 == 0) expand_blossom(b);
193     }
194     return false;
195 }
196 pair<long long, int> solve() {
197     memset(match + 1, 0, sizeof(int) * n);
198     n_x = n;
199     int n_matches = 0;
200     long long tot_weight = 0;
201     for (int u = 0; u <= n; ++u) st[u] = u, flo[u].
202         clear();
203     int w_max = 0;
204     for (int u = 1; u <= n; ++u)
205         for (int v = 1; v <= n; ++v) {
206             flo_from[u][v] = (u == v ? u : 0);
207             w_max = max(w_max, g[u][v].w);
208         }
209     for (int u = 1; u <= n; ++u) lab[u] = w_max;
210     while (matching()) ++n_matches;
211     for (int u = 1; u <= n; ++u)
212         if (match[u] && match[u] < u)
213             tot_weight += g[u][match[u]].w;
214     return make_pair(tot_weight, n_matches);
215 }
216 void add_edge(int ui, int vi, int wi) { g[ui][vi].w
217     = g[vi][ui].w = wi; }
218 void init(int _n) {
219     n = _n;
220     for (int u = 1; u <= n; ++u)
221         for (int v = 1; v <= n; ++v)
222             g[u][v] = edge(u, v, 0);
223 }

```

4.7 Cover / Independent Set

- 1 V(E) Cover: choose some V(E) to cover all E(V)
- 2 V(E) Independ: set of V(E) not adj to each other
- 3 M = Max Matching
- 4 Cv = Min V Cover
- 5 Ce = Min E Cover

```

7 Iv = Max V Ind
8 Ie = Max E Ind (equiv to M)
9
10 M = Cv (Konig Theorem)
11 Iv = V \ Cv
12 Ce = V - M
13
14 Construct Cv:
15 1. Run Dinic
16 2. Find s-t min cut
17 3. Cv = {X in T} + {Y in S}

```

5 Graph

5.1 Heavy-Light Decomposition

```

1 const int N = 2e5 + 5;
2 int n, dfn[N], son[N], top[N], num[N], dep[N], p[N];
3 vector<int> path[N];
4 struct node {
5     int mx, sum;
6 } seg[N << 2];
7 void update(int x, int l, int r, int qx, int val) {
8     if (l == r) {
9         seg[x].mx = seg[x].sum = val;
10        return;
11    }
12    int mid = (l + r) >> 1;
13    if (qx <= mid) update(x << 1, l, mid, qx, val);
14    else update(x << 1 | 1, mid + 1, r, qx, val);
15    seg[x].mx = max(seg[x << 1].mx, seg[x << 1 | 1].mx);
16    seg[x].sum = seg[x << 1].sum + seg[x << 1 | 1].sum;
17 }
18 int big(int x, int l, int r, int ql, int qr) {
19     if (ql <= l && r <= qr) return seg[x].mx;
20     int mid = (l + r) >> 1;
21     int res = -INF;
22     if (ql <= mid) res = max(res, big(x << 1, l, mid, ql, qr));
23     if (mid < qr) res = max(res, big(x << 1 | 1, mid + 1, r, ql, qr));
24     return res;
25 }
26 int ask(int x, int l, int r, int ql, int qr) {
27     if (ql <= l && r <= qr) return seg[x].sum;
28     int mid = (l + r) >> 1;
29     int res = 0;
30     if (ql <= mid) res += ask(x << 1, l, mid, ql, qr);
31     if (mid < qr) res += ask(x << 1 | 1, mid + 1, r, ql, qr);
32     return res;
33 }
34 void dfs1(int now) {
35     son[now] = -1;
36     num[now] = 1;
37     for (auto i : path[now]) {
38         if (!dep[i]) {
39             dep[i] = dep[now] + 1;
40             p[i] = now;
41             dfs1(i);
42             num[now] += num[i];
43             if (son[now] == -1 || num[i] > num[son[now]]) son[now] = i;
44         }
45     }
46 }
47 int cnt;
48 void dfs2(int now, int t) {
49     top[now] = t;
50     cnt++;
51     dfn[now] = cnt;
52     if (son[now] == -1) return;
53     dfs2(son[now], t);
54     for (auto i : path[now])
55         if (i != p[now] && i != son[now]) dfs2(i, i);
56 }
57 int path_big(int x, int y) {
58     int res = -INF;
59     while (top[x] != top[y]) {

```

```

60         if (dep[top[x]] < dep[top[y]]) swap(x, y);
61         res = max(res, big(1, 1, n, dfn[top[x]], dfn[x]));
62         x = p[top[x]];
63     }
64     if (dfn[x] > dfn[y]) swap(x, y);
65     res = max(res, big(1, 1, n, dfn[x], dfn[y]));
66     return res;
67 }
68 int path_sum(int x, int y) {
69     int res = 0;
70     while (top[x] != top[y]) {
71         if (dep[top[x]] < dep[top[y]]) swap(x, y);
72         res += ask(1, 1, n, dfn[top[x]], dfn[x]);
73         x = p[top[x]];
74     }
75     if (dfn[x] > dfn[y]) swap(x, y);
76     res += ask(1, 1, n, dfn[x], dfn[y]);
77     return res;
78 }
79 void buildTree() {
80     FOR(i, 0, n - 1) {
81         int a, b;
82         cin >> a >> b;
83         path[a].pb(b);
84         path[b].pb(a);
85     }
86 }
87 void buildHLD(int root) {
88     dep[root] = 1;
89     dfs1(root);
90     dfs2(root, root);
91     FOR(i, 1, n + 1) {
92         int now;
93         cin >> now;
94         update(1, 1, n, dfn[i], now);
95     }
96 }

```

5.2 Centroid Decomposition

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int N = 1e5 + 5;
4 vector<int> a[N];
5 int sz[N], lv[N];
6 bool used[N];
7 int f_sz(int x, int p) {
8     sz[x] = 1;
9     for (int i : a[x])
10         if (i != p && !used[i])
11             sz[x] += f_sz(i, x);
12     return sz[x];
13 }
14 int f_cen(int x, int p, int total) {
15     for (int i : a[x]) {
16         if (i != p && !used[i] && 2 * sz[i] > total)
17             return f_cen(i, x, total);
18     }
19     return x;
20 }
21 void cd(int x, int p) {
22     int total = f_sz(x, p);
23     int cen = f_cen(x, p, total);
24     lv[cen] = lv[p] + 1;
25     used[cen] = 1;
26     // cout << "cd: " << x << " " << p << " " << cen << "\n";
27     for (int i : a[cen]) {
28         if (!used[i])
29             cd(i, cen);
30     }
31 }
32 int main() {
33     ios_base::sync_with_stdio(0);
34     cin.tie(0);
35     int n;
36     cin >> n;
37     for (int i = 0, x, y; i < n - 1; i++) {
38         cin >> x >> y;
39         a[x].push_back(y);

```



```

40     a[y].push_back(x);
41 }
42 cd(1, 0);
43 for (int i = 1; i <= n; i++)
44     cout << (char)('A' + lv[i] - 1) << " ";
45 cout << "\n";
46 }

```

5.3 Bellman-Ford + SPFA

```

1  int n, m;
2
3  // Graph
4  vector<vector<pair<int, ll> > > g;
5  vector<ll> dis;
6  vector<bool> negCycle;
7
8  // SPFA
9  vector<int> rlx;
10 queue<int> q;
11 vector<bool> inq;
12 vector<int> pa;
13 void SPFA(vector<int>& src) {
14     dis.assign(n + 1, LINF);
15     negCycle.assign(n + 1, false);
16     rlx.assign(n + 1, 0);
17     while (!q.empty()) q.pop();
18     inq.assign(n + 1, false);
19     pa.assign(n + 1, -1);
20
21     for (auto& s : src) {
22         dis[s] = 0;
23         q.push(s);
24         inq[s] = true;
25     }
26
27     while (!q.empty()) {
28         int u = q.front();
29         q.pop();
30         inq[u] = false;
31         if (rlx[u] >= n) {
32             negCycle[u] = true;
33         } else
34             for (auto& e : g[u]) {
35                 int v = e.first;
36                 ll w = e.second;
37                 if (dis[v] > dis[u] + w) {
38                     dis[v] = dis[u] + w;
39                     rlx[v] = rlx[u] + 1;
40                     pa[v] = u;
41                     if (!inq[v]) {
42                         q.push(v);
43                         inq[v] = true;
44                     }
45                 }
46             }
47     }
48 }
49
50 // Bellman-Ford
51 queue<int> q;
52 vector<int> pa;
53 void BellmanFord(vector<int>& src) {
54     dis.assign(n + 1, LINF);
55     negCycle.assign(n + 1, false);
56     pa.assign(n + 1, -1);
57
58     for (auto& s : src) dis[s] = 0;
59
60     for (int rlx = 1; rlx <= n; rlx++) {
61         for (int u = 1; u <= n; u++) {
62             if (dis[u] == LINF) continue; // Important
63             !!
64             for (auto& e : g[u]) {
65                 int v = e.first;
66                 ll w = e.second;
67                 if (dis[v] > dis[u] + w) {
68                     dis[v] = dis[u] + w;
69                     pa[v] = u;
70                     if (rlx == n) negCycle[v] = true;
71                 }
72             }
73         }
74     }
75 }

```

```

71     }
72 }
73 }
74 }
75
76 // Negative Cycle Detection
77 void NegCycleDetect() {
78     /* No Neg Cycle: NO
79     Exist Any Neg Cycle:
80     YES
81     v0 v1 v2 ... vk v0 */
82
83     vector<int> src;
84     for (int i = 1; i <= n; i++)
85         src.emplace_back(i);
86
87     SPFA(src);
88     // BellmanFord(src);
89
90     int ptr = -1;
91     for (int i = 1; i <= n; i++)
92         if (negCycle[i]) {
93             ptr = i;
94             break;
95         }
96
97     if (ptr == -1) {
98         return cout << "NO" << endl, void();
99     }
100
101     cout << "YES\n";
102     vector<int> ans;
103     vector<bool> vis(n + 1, false);
104
105     while (true) {
106         ans.emplace_back(ptr);
107         if (vis[ptr]) break;
108         vis[ptr] = true;
109         ptr = pa[ptr];
110     }
111     reverse(ans.begin(), ans.end());
112
113     vis.assign(n + 1, false);
114     for (auto& x : ans) {
115         cout << x << ' ';
116         if (vis[x]) break;
117         vis[x] = true;
118     }
119     cout << endl;
120 }
121
122 // Distance Calculation
123 void calcDis(int s) {
124     vector<int> src;
125     src.emplace_back(s);
126     SPFA(src);
127     // BellmanFord(src);
128
129     while (!q.empty()) q.pop();
130     for (int i = 1; i <= n; i++)
131         if (negCycle[i]) q.push(i);
132
133     while (!q.empty()) {
134         int u = q.front();
135         q.pop();
136         for (auto& e : g[u]) {
137             int v = e.first;
138             if (!negCycle[v]) {
139                 q.push(v);
140                 negCycle[v] = true;
141             }
142         }
143     }
144 }

```

5.4 BCC - AP

```

1  int n, m;
2  int low[maxn], dfn[maxn], instp;
3  vector<int> E, g[maxn];
4  bitset<maxn> isap;

```

```

5 bitset<maxm> vis;
6 stack<int> stk;
7 int bccnt;
8 vector<int> bcc[maxn];
9 inline void popout(int u) {
10     bccnt++;
11     bcc[bccnt].emplace_back(u);
12     while (!stk.empty()) {
13         int v = stk.top();
14         if (u == v) break;
15         stk.pop();
16         bcc[bccnt].emplace_back(v);
17     }
18 }
19 void dfs(int u, bool rt = 0) {
20     stk.push(u);
21     low[u] = dfn[u] = ++instp;
22     int kid = 0;
23     Each(e, g[u]) {
24         if (vis[e]) continue;
25         vis[e] = true;
26         int v = E[e] ^ u;
27         if (!dfn[v]) {
28             // tree edge
29             kid++;
30             dfs(v);
31             low[u] = min(low[u], low[v]);
32             if (!rt && low[v] >= dfn[u]) {
33                 // bcc found: u is ap
34                 isap[u] = true;
35                 popout(u);
36             }
37         } else {
38             // back edge
39             low[u] = min(low[u], dfn[v]);
40         }
41     }
42     // special case: root
43     if (rt) {
44         if (kid > 1) isap[u] = true;
45         popout(u);
46     }
47 }
48 void init() {
49     cin >> n >> m;
50     fill(low, low + maxn, INF);
51     REP(i, m) {
52         int u, v;
53         cin >> u >> v;
54         g[u].emplace_back(i);
55         g[v].emplace_back(i);
56         E.emplace_back(u ^ v);
57     }
58 }
59 void solve() {
60     FOR(i, 1, n + 1, 1) {
61         if (!dfn[i]) dfs(i, true);
62     }
63     vector<int> ans;
64     int cnt = 0;
65     FOR(i, 1, n + 1, 1) {
66         if (isap[i]) cnt++, ans.emplace_back(i);
67     }
68     cout << cnt << endl;
69     Each(i, ans) cout << i << ' ';
70     cout << endl;
71 }

```

5.5 BCC - Bridge

```

1 int n, m;
2 vector<int> g[maxn], E;
3 int low[maxn], dfn[maxn], instp;
4 int bccnt, bccid[maxn];
5 stack<int> stk;
6 bitset<maxm> vis, isbrg;
7 void init() {
8     cin >> n >> m;
9     REP(i, m) {
10         int u, v;
11         cin >> u >> v;

```

```

12         E.emplace_back(u ^ v);
13         g[u].emplace_back(i);
14         g[v].emplace_back(i);
15     }
16     fill(low, low + maxn, INF);
17 }
18 void popout(int u) {
19     bccnt++;
20     while (!stk.empty()) {
21         int v = stk.top();
22         if (v == u) break;
23         stk.pop();
24         bccid[v] = bccnt;
25     }
26 }
27 void dfs(int u) {
28     stk.push(u);
29     low[u] = dfn[u] = ++instp;
30
31     Each(e, g[u]) {
32         if (vis[e]) continue;
33         vis[e] = true;
34
35         int v = E[e] ^ u;
36         if (dfn[v]) {
37             // back edge
38             low[u] = min(low[u], dfn[v]);
39         } else {
40             // tree edge
41             dfs(v);
42             low[u] = min(low[u], low[v]);
43             if (low[v] == dfn[v]) {
44                 isbrg[e] = true;
45                 popout(u);
46             }
47         }
48     }
49 }
50 void solve() {
51     FOR(i, 1, n + 1, 1) {
52         if (!dfn[i]) dfs(i);
53     }
54     vector<pii> ans;
55     vis.reset();
56     FOR(u, 1, n + 1, 1) {
57         Each(e, g[u]) {
58             if (!isbrg[e] || vis[e]) continue;
59             vis[e] = true;
60             int v = E[e] ^ u;
61             ans.emplace_back(mp(u, v));
62         }
63     }
64     cout << (int)ans.size() << endl;
65     Each(e, ans) cout << e.F << ' ' << e.S << endl;
66 }

```

5.6 SCC - Tarjan

```

1 // 2-SAT
2 vector<int> E, g[maxn]; // 1~n, n+1~2n
3 int low[maxn], in[maxn], instp;
4 int scnt, sccid[maxn];
5 stack<int> stk;
6 bitset<maxn> ins, vis;
7 int n, m;
8 void init() {
9     cin >> m >> n;
10     E.clear();
11     fill(g, g + maxn, vector<int>());
12     fill(low, low + maxn, INF);
13     memset(in, 0, sizeof(in));
14     instp = 1;
15     scnt = 0;
16     memset(sccid, 0, sizeof(sccid));
17     ins.reset();
18     vis.reset();
19 }
20 inline int no(int u) {
21     return (u > n ? u - n : u + n);
22 }
23 int ecnt = 0;

```

```

24 inline void clause(int u, int v) {
25     E.eb(no(u) ^ v);
26     g[no(u)].eb(ecnt++);
27     E.eb(no(v) ^ u);
28     g[no(v)].eb(ecnt++);
29 }
30 void dfs(int u) {
31     in[u] = instp++;
32     low[u] = in[u];
33     stk.push(u);
34     ins[u] = true;
35
36     Each(e, g[u]) {
37         if (vis[e]) continue;
38         vis[e] = true;
39
40         int v = E[e] ^ u;
41         if (ins[v])
42             low[u] = min(low[u], in[v]);
43         else if (!in[v]) {
44             dfs(v);
45             low[u] = min(low[u], low[v]);
46         }
47     }
48     if (low[u] == in[u]) {
49         sccnt++;
50         while (!stk.empty()) {
51             int v = stk.top();
52             stk.pop();
53             ins[v] = false;
54             sccid[v] = sccnt;
55             if (u == v) break;
56         }
57     }
58 }
59 int main() {
60     init();
61     REP(i, m) {
62         char su, sv;
63         int u, v;
64         cin >> su >> u >> sv >> v;
65         if (su == '-') u = no(u);
66         if (sv == '-') v = no(v);
67         clause(u, v);
68     }
69     FOR(i, 1, 2 * n + 1, 1) {
70         if (!in[i]) dfs(i);
71     }
72     FOR(u, 1, n + 1, 1) {
73         int du = no(u);
74         if (sccid[u] == sccid[du]) {
75             return cout << "IMPOSSIBLE\n", 0;
76         }
77     }
78     FOR(u, 1, n + 1, 1) {
79         int du = no(u);
80         cout << (sccid[u] < sccid[du] ? '+' : '-') << '
81         ';
82     }
83     cout << endl;
84 }

```

5.7 SCC - Kosaraju

```

1 const int N = 1e5 + 10;
2 vector<int> ed[N], ed_b[N]; // 反邊
3 vector<int> SCC(N); // 最後SCC的分組
4 bitset<N> vis;
5 int SCC_cnt;
6 int n, m;
7 vector<int> pre; // 後序遍歷
8
9 void dfs(int x) {
10     vis[x] = 1;
11     for (int i : ed[x]) {
12         if (vis[i]) continue;
13         dfs(i);
14     }
15     pre.push_back(x);
16 }
17

```

```

18 void dfs2(int x) {
19     vis[x] = 1;
20     SCC[x] = SCC_cnt;
21     for (int i : ed_b[x]) {
22         if (vis[i]) continue;
23         dfs2(i);
24     }
25 }
26
27 void kosaraju() {
28     for (int i = 1; i <= n; i++) {
29         if (!vis[i]) {
30             dfs(i);
31         }
32     }
33     SCC_cnt = 0;
34     vis = 0;
35     for (int i = n - 1; i >= 0; i--) {
36         if (!vis[pre[i]]) {
37             SCC_cnt++;
38             dfs2(pre[i]);
39         }
40     }
41 }

```

5.8 Eulerian Path - Undir

```

1 // from 1 to n
2 #define gg return cout << "IMPOSSIBLE\n", void();
3
4 int n, m;
5 vector<int> g[maxn];
6 bitset<maxn> inodd;
7
8 void init() {
9     cin >> n >> m;
10    inodd.reset();
11    for (int i = 0; i < m; i++) {
12        int u, v;
13        cin >> u >> v;
14        inodd[u] = inodd[u] ^ true;
15        inodd[v] = inodd[v] ^ true;
16        g[u].emplace_back(v);
17        g[v].emplace_back(u);
18    }
19 }
20 stack<int> stk;
21 void dfs(int u) {
22     while (!g[u].empty()) {
23         int v = g[u].back();
24         g[u].pop_back();
25         dfs(v);
26     }
27     stk.push(u);
28 }

```

5.9 Eulerian Path - Dir

```

1 // from node 1 to node n
2 #define gg return cout << "IMPOSSIBLE\n", 0;
3
4 int n, m;
5 vector<int> g[maxn];
6 stack<int> stk;
7 int in[maxn], out[maxn];
8
9 void init() {
10    cin >> n >> m;
11    for (int i = 0; i < m; i++) {
12        int u, v;
13        cin >> u >> v;
14        g[u].emplace_back(v);
15        out[u]++, in[v]++;
16    }
17    for (int i = 1; i <= n; i++) {
18        if (i == 1 && out[i] - in[i] != 1) gg;
19        if (i == n && in[i] - out[i] != 1) gg;
20        if (i != 1 && i != n && in[i] != out[i]) gg;
21    }
22 }

```

```

23 void dfs(int u) {
24     while (!g[u].empty()) {
25         int v = g[u].back();
26         g[u].pop_back();
27         dfs(v);
28     }
29     stk.push(u);
30 }
31 void solve() {
32     dfs(1) for (int i = 1; i <= n; i++) if ((int)g[i].
33         size()) gg;
34     while (!stk.empty()) {
35         int u = stk.top();
36         stk.pop();
37         cout << u << ' ';
38     }

```

5.10 Hamilton Path

```

1 // top down DP
2 // Be Aware Of Multiple Edges
3 int n, m;
4 ll dp[maxn][1<<maxn];
5 int adj[maxn][maxn];
6
7 void init() {
8     cin >> n >> m;
9     fill(dp[0], dp[maxn-1]+(1<<maxn), -1);
10 }
11
12 void DP(int i, int msk) {
13     if (dp[i][msk] != -1) return;
14     dp[i][msk] = 0;
15     REP(j, n) if (j != i && (msk & (1<<j)) && adj[j][i]) {
16         int sub = msk ^ (1<<i);
17         if (dp[j][sub] == -1) DP(j, sub);
18         dp[i][msk] += dp[j][sub] * adj[j][i];
19         if (dp[i][msk] >= MOD) dp[i][msk] %= MOD;
20     }
21 }
22
23 int main() {
24     WiWiHorz
25     init();
26
27     REP(i, m) {
28         int u, v;
29         cin >> u >> v;
30         if (u == v) continue;
31         adj[--u][--v]++;
32     }
33
34     dp[0][1] = 1;
35     FOR(i, 1, n, 1) {
36         dp[i][1] = 0;
37         dp[i][1|(1<<i)] = adj[0][i];
38     }
39     FOR(msk, 1, (1<<n), 1) {
40         if (msk == 1) continue;
41         dp[0][msk] = 0;
42     }
43
44     DP(n-1, (1<<n)-1);
45     cout << dp[n-1][(1<<n)-1] << endl;
46
47     return 0;
48 }

```

5.11 Kth Shortest Path

```

1 // time: O(|E| \lg |E|+|V| \lg |V|+K)
2 // memory: O(|E| \lg |E|+|V|)
3 struct KSP { // 1-base
4     struct nd {
5         int u, v;
6         ll d;

```

```

7         nd(int ui = 0, int vi = 0, ll di = INF) {
8             u = ui;
9             v = vi;
10            d = di;
11        }
12    };
13    struct heap {
14        nd* edge;
15        int dep;
16        heap* chd[4];
17    };
18    static int cmp(heap* a, heap* b) { return a->edge->
19        d > b->edge->d; }
20    struct node {
21        int v;
22        ll d;
23        heap* H;
24        nd* E;
25        node() {}
26        node(ll _d, int _v, nd* _E) {
27            d = _d;
28            v = _v;
29            E = _E;
30        }
31        node(heap* _H, ll _d) {
32            H = _H;
33            d = _d;
34        }
35        friend bool operator<(node a, node b) { return
36            a.d > b.d; }
37    };
38    int n, k, s, t, dst[N];
39    nd* nxt[N];
40    vector<nd*> g[N], rg[N];
41    heap *nullNd, *head[N];
42    void init(int _n, int _k, int _s, int _t) {
43        n = _n;
44        k = _k;
45        s = _s;
46        t = _t;
47        for (int i = 1; i <= n; i++) {
48            g[i].clear();
49            rg[i].clear();
50            nxt[i] = NULL;
51            head[i] = NULL;
52            dst[i] = -1;
53        }
54    }
55    void addEdge(int ui, int vi, ll di) {
56        nd* e = new nd(ui, vi, di);
57        g[ui].push_back(e);
58        rg[vi].push_back(e);
59    }
60    queue<int> dfsQ;
61    void dijkstra() {
62        while (dfsQ.size()) dfsQ.pop();
63        priority_queue<node> Q;
64        Q.push(node(0, t, NULL));
65        while (!Q.empty()) {
66            node p = Q.top();
67            Q.pop();
68            if (dst[p.v] != -1) continue;
69            dst[p.v] = p.d;
70            nxt[p.v] = p.E;
71            dfsQ.push(p.v);
72            for (auto e : rg[p.v]) Q.push(node(p.d + e
73                ->d, e->u, e));
74        }
75    }
76    heap* merge(heap* curNd, heap* newNd) {
77        if (curNd == nullNd) return newNd;
78        heap* root = new heap;
79        memcpy(root, curNd, sizeof(heap));
80        if (newNd->edge->d < curNd->edge->d) {
81            root->edge = newNd->edge;
82            root->chd[2] = newNd->chd[2];
83            root->chd[3] = newNd->chd[3];
84            newNd->edge = curNd->edge;
85            newNd->chd[2] = curNd->chd[2];
86            newNd->chd[3] = curNd->chd[3];
87        }
88        if (root->chd[0]->dep < root->chd[1]->dep)

```

```

86     root->chd[0] = merge(root->chd[0], newNd); 165
87     else 166
88         root->chd[1] = merge(root->chd[1], newNd); 167
89     root->dep = max(root->chd[0]->dep,
90                     root->chd[1]->dep) +
91         1;
92     return root;
93 }
94 vector<heap*> V;
95 void build() {
96     nullNd = new heap;
97     nullNd->dep = 0;
98     nullNd->edge = new nd;
99     fill(nullNd->chd, nullNd->chd + 4, nullNd);
100    while (not dfsQ.empty()) {
101        int u = dfsQ.front();
102        dfsQ.pop();
103        if (!nxt[u])
104            head[u] = nullNd;
105        else
106            head[u] = head[nxt[u]->v];
107        V.clear();
108        for (auto&& e : g[u]) {
109            int v = e->v;
110            if (dst[v] == -1) continue;
111            e->d += dst[v] - dst[u];
112            if (nxt[u] != e) {
113                heap* p = new heap;
114                fill(p->chd, p->chd + 4, nullNd);
115                p->dep = 1;
116                p->edge = e;
117                V.push_back(p);
118            }
119        }
120        if (V.empty()) continue;
121        make_heap(V.begin(), V.end(), cmp);
122    #define L(X) ((X << 1) + 1)
123    #define R(X) ((X << 1) + 2)
124    for (size_t i = 0; i < V.size(); i++) {
125        if (L(i) < V.size())
126            V[i]->chd[2] = V[L(i)];
127        else
128            V[i]->chd[2] = nullNd;
129        if (R(i) < V.size())
130            V[i]->chd[3] = V[R(i)];
131        else
132            V[i]->chd[3] = nullNd;
133    }
134    head[u] = merge(head[u], V.front());
135 }
136 }
137 vector<ll> ans;
138 void first_K() {
139     ans.clear();
140     priority_queue<node> Q;
141     if (dst[s] == -1) return;
142     ans.push_back(dst[s]);
143     if (head[s] != nullNd)
144         Q.push(node(head[s], dst[s] + head[s]->edge
145                     ->d));
146     for (int _ = 1; _ < k and not Q.empty(); _++) {
147         node p = Q.top(); Q.pop();
148         ans.push_back(p.d);
149         if (head[p.H->edge->v] != nullNd) {
150             q.H = head[p.H->edge->v];
151             q.d = p.d + q.H->edge->d;
152             Q.push(q);
153         }
154         for (int i = 0; i < 4; i++)
155             if (p.H->chd[i] != nullNd) {
156                 q.H = p.H->chd[i];
157                 q.d = p.d - p.H->edge->d + p.H->chd
158                     [i]->edge->d;
159                 Q.push(q);
160             }
161     }
162     void solve() { // ans[i] stores the i-th shortest
163         path
164         dijkstra();
165         build();

```

```

first_K(); // ans.size() might less than k
}
} solver;

```

5.12 System of Difference Constraints

```

1 vector<vector<pair<int, ll>>> G;
2 void add(int u, int v, ll w) {
3     G[u].emplace_back(make_pair(v, w));
4 }

```

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

```

1 struct ACautomata {
2     struct Node {
3         int cnt;
4         Node *go[26], *fail, *dic;
5         Node() {
6             cnt = 0;
7             fail = 0;
8             dic = 0;
9             memset(go, 0, sizeof(go));
10        }
11    } pool[1048576], *root;
12    int nMem;
13    Node *new_Node() {
14        pool[nMem] = Node();
15        return &pool[nMem++];
16    }
17    void init() {
18        nMem = 0;
19        root = new_Node();
20    }
21    void add(const string &str) { insert(root, str, 0);
22    }
23    void insert(Node *cur, const string &str, int pos)
24    {
25        for (int i = pos; i < str.size(); i++) {
26            if (!cur->go[str[i] - 'a'])
27                cur->go[str[i] - 'a'] = new_Node();
28            cur = cur->go[str[i] - 'a'];
29        }
30        cur->cnt++;
31    }
32    void make_fail() {
33        queue<Node *> que;
34        que.push(root);
35        while (!que.empty()) {
36            Node *fr = que.front();
37            que.pop();
38            for (int i = 0; i < 26; i++) {
39                if (fr->go[i]) {
40                    Node *ptr = fr->fail;
41                    while (ptr && !ptr->go[i]) ptr = ptr->fail;
42                    fr->go[i]->fail = ptr = (ptr ? ptr->go[i] : root);

```

```

41     fr->go[i]->dic = (ptr->cnt ? ptr : 15
        ptr->dic);
42     que.push(fr->go[i]);
43     }
44     }
45     }
46     }
47 } AC;

```

6.2 KMP

```

1 vector<int> f;
2 void buildFailFunction(string &s) {
3     f.resize(s.size(), -1);
4     for (int i = 1; i < s.size(); i++) {
5         int now = f[i - 1];
6         while (now != -1 and s[now + 1] != s[i]) now =
            f[now];
7         if (s[now + 1] == s[i]) f[i] = now + 1;
8     }
9 }
10
11 void KMPmatching(string &a, string &b) {
12     for (int i = 0, now = -1; i < a.size(); i++) {
13         while (a[i] != b[now + 1] and now != -1) now =
            f[now];
14         if (a[i] == b[now + 1]) now++;
15         if (now + 1 == b.size()) {
16             cout << "found a match start at position "
                << i - now << endl;
17             now = f[now];
18         }
19     }
20 }

```

6.3 Z Value

```

1 string is, it, s;
2 int n;
3 vector<int> z;
4 void init() {
5     cin >> is >> it;
6     s = it + '0' + is;
7     n = (int)s.size();
8     z.resize(n, 0);
9 }
10 void solve() {
11     int ans = 0;
12     z[0] = n;
13     for (int i = 1, l = 0, r = 0; i < n; i++) {
14         if (i <= r) z[i] = min(z[i - 1], r - i + 1);
15         while (i + z[i] < n && s[z[i]] == s[i + z[i]])
            z[i]++;
16         if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
17         if (z[i] == (int)it.size()) ans++;
18     }
19     cout << ans << endl;
20 }

```

6.4 Manacher

```

1 int n;
2 string S, s;
3 vector<int> m;
4 void manacher() {
5     s.clear();
6     s.resize(2 * n + 1, '.');
7     for (int i = 0, j = 1; i < n; i++, j += 2) s[j] = S
        [i];
8     m.clear();
9     m.resize(2 * n + 1, 0);
10    // m[i] := max k such that s[i-k, i+k] is
        palindrome
11    int mx = 0, mxk = 0;
12    for (int i = 1; i < 2 * n + 1; i++) {
13        if (mx - (i - mx) >= 0) m[i] = min(m[mx - (i -
            mx)], mx + mxk - i);
14        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]
            ]++;
16        if (i + m[i] > mx + mxk) mx = i, mxk = m[i];
17    }
18 }
19 void init() {
20     cin >> S;
21     n = (int)S.size();
22 }
23 void solve() {
24     manacher();
25     int mx = 0, ptr = 0;
26     for (int i = 0; i < 2 * n + 1; i++)
27         if (mx < m[i]) {
28             mx = m[i];
29             ptr = i;
30         }
31     for (int i = ptr - mx; i <= ptr + mx; i++)
32         if (s[i] != '.') cout << s[i];
33     cout << endl;
34 }

```

6.5 Suffix Array

```

1 #define F first
2 #define S second
3 struct SuffixArray { // don't forget s += "$";
4     int n;
5     string s;
6     vector<int> suf, lcp, rk;
7     vector<int> cnt, pos;
8     vector<pair<pii, int>> buc[2];
9     void init(string _s) {
10         s = _s;
11         n = (int)s.size();
12         // resize(n): suf, rk, cnt, pos, lcp, buc[0~1]
13     }
14     void radix_sort() {
15         for (int t : {0, 1}) {
16             fill(cnt.begin(), cnt.end(), 0);
17             for (auto& i : buc[t]) cnt[(t ? i.F.F : i.F
                .S)]++;
18             for (int i = 0; i < n; i++)
19                 pos[i] = (i ? 0 : pos[i - 1] + cnt[i -
                    1]);
20             for (auto& i : buc[t])
21                 buc[t ^ 1][pos[(t ? i.F.F : i.F.S)]++]
                    = i;
22         }
23     }
24     bool fill_suf() {
25         bool end = true;
26         for (int i = 0; i < n; i++) suf[i] = buc[0][i].
            S;
27         rk[suf[0]] = 0;
28         for (int i = 1; i < n; i++) {
29             int dif = (buc[0][i].F != buc[0][i - 1].F);
30             end &= dif;
31             rk[suf[i]] = rk[suf[i - 1]] + dif;
32         }
33         return end;
34     }
35     void sa() {
36         for (int i = 0; i < n; i++)
37             buc[0][i] = make_pair(make_pair(s[i], s[i]),
                i);
38         sort(buc[0].begin(), buc[0].end());
39         if (fill_suf()) return;
40         for (int k = 0; (1 << k) < n; k++) {
41             for (int i = 0; i < n; i++)
42                 buc[0][i] = make_pair(make_pair(rk[i],
                    rk[(i + (1 << k)) % n]), i);
43             radix_sort();
44             if (fill_suf()) return;
45         }
46     }
47     void LCP() {
48         int k = 0;
49         for (int i = 0; i < n - 1; i++) {
50             if (rk[i] == 0) continue;
51             int pi = rk[i];

```



```

52     int j = suf[pi - 1];
53     while (i + k < n && j + k < n && s[i + k]
54           == s[j + k]) k++;
55     lcp[pi] = k;
56     k = max(k - 1, 0);
57 }
58 };
59 SuffixArray suffixarray;

```

6.6 Minimum Rotation

```

1 // rotate(begin(s), begin(s)+minRotation(s), end(s))
2 int minRotation(string s) {
3     int a = 0, n = s.size();
4     s += s;
5     for (int b = 0; b < n; b++)
6         for (int k = 0; k < n; k++) {
7             if (a + k == b || s[a + k] < s[b + k]) {
8                 b += max(0, k - 1);
9                 break;
10            }
11            if (s[a + k] > s[b + k]) {
12                a = b;
13                break;
14            }
15        }
16     return a;
17 }

```

6.7 Lyndon Factorization

```

1 vector<string> duval(string const& s) {
2     int n = s.size();
3     int i = 0;
4     vector<string> factorization;
5     while (i < n) {
6         int j = i + 1, k = i;
7         while (j < n && s[k] <= s[j]) {
8             if (s[k] < s[j])
9                 k = i;
10            else
11                k++;
12            j++;
13        }
14        while (i <= k) {
15            factorization.push_back(s.substr(i, j - k));
16            i += j - k;
17        }
18    }
19    return factorization; // O(n)
20 }

```

6.8 Rolling Hash

```

1 const ll C = 27;
2 inline int id(char c) { return c - 'a' + 1; }
3 struct RollingHash {
4     string s;
5     int n;
6     ll mod;
7     vector<ll> Cexp, hs;
8     RollingHash(string& _s, ll _mod) : s(_s), n((int)_s
9           .size()), mod(_mod) {
10         Cexp.assign(n, 0);
11         hs.assign(n, 0);
12         Cexp[0] = 1;
13         for (int i = 1; i < n; i++) {
14             Cexp[i] = Cexp[i - 1] * C;
15             if (Cexp[i] >= mod) Cexp[i] %= mod;
16         }
17         hs[0] = id(s[0]);
18         for (int i = 1; i < n; i++) {
19             hs[i] = hs[i - 1] * C + id(s[i]);
20             if (hs[i] >= mod) hs[i] %= mod;
21         }
22     }
23     inline ll query(int l, int r) {

```

```

24         ll res = hs[r] - (l ? hs[l - 1] * Cexp[r - l +
25           1] : 0);
26         res = (res % mod + mod) % mod;
27         return res;
28     }
29 };

```

6.9 Trie

```

1 pii a[N][26];
2
3 void build(string &s) {
4     static int idx = 0;
5     int n = s.size();
6     for (int i = 0, v = 0; i < n; i++) {
7         pii &now = a[v][s[i] - 'a'];
8         if (now.first != -1)
9             v = now.first;
10        else
11            v = now.first = ++idx;
12        if (i == n - 1)
13            now.second++;
14    }
15 }

```

7 Geometry

7.1 Basic Operations

```

1 typedef long long T;
2 // typedef long double T;
3 const long double eps = 1e-8;
4 short sgn(T x) {
5     if (abs(x) < eps) return 0;
6     return x < 0 ? -1 : 1;
7 }
8 struct Pt {
9     T x, y;
10    Pt(T _x = 0, T _y = 0) : x(_x), y(_y) {}
11    Pt operator+(Pt a) { return Pt(x + a.x, y + a.y); }
12    Pt operator-(Pt a) { return Pt(x - a.x, y - a.y); }
13    Pt operator*(T a) { return Pt(x * a, y * a); }
14    Pt operator/(T a) { return Pt(x / a, y / a); }
15    T operator*(Pt a) { return x * a.x + y * a.y; }
16    T operator^(Pt a) { return x * a.y - y * a.x; }
17    bool operator<(Pt a) { return x < a.x || (x == a.x
18          && y < a.y); }
19    // return sgn(x-a.x) < 0 || (sgn(x-a.x) == 0 && sgn
20          (y-a.y) < 0); }
21    bool operator==(Pt a) { return sgn(x - a.x) == 0 &&
22          sgn(y - a.y) == 0; }
23 };
24 Pt mv(Pt a, Pt b) { return b - a; }
25 T len2(Pt a) { return a * a; }
26 T dis2(Pt a, Pt b) { return len2(b - a); }
27 short ori(Pt a, Pt b) { return ((a ^ b) > 0) - ((a ^ b)
28       < 0); }
29 bool onseg(Pt p, Pt l1, Pt l2) {
30     Pt a = mv(p, l1), b = mv(p, l2);
31     return ((a ^ b) == 0) && ((a * b) <= 0);
32 }

```

7.2 SVG Writer

7.3 Sort by Angle

```

1 int ud(Pt a) { // up or down half plane
2     if (a.y > 0) return 0;
3     if (a.y < 0) return 1;
4     return (a.x >= 0 ? 0 : 1);
5 }
6 sort(pts.begin(), pts.end(), [&](const Pt& a, const Pt&
7     b) {
8     if (ud(a) != ud(b)) return ud(a) < ud(b);
9     return (a ^ b) > 0;
10 });

```

7.4 Line Intersection

```

1 bool line_intersect_check(Pt p1, Pt p2, Pt q1, Pt q2) {
2     if (onseg(p1, q1, q2) || onseg(p2, q1, q2) || onseg
3         (q1, p1, p2) || onseg(q2, p1, p2)) return true;
4     Pt p = mv(p1, p2), q = mv(q1, q2);
5     return (ori(p, mv(p1, q1)) * ori(p, mv(p1, q2)) <
6             0) && (ori(q, mv(q1, p1)) * ori(q, mv(q1, p2))
7                 < 0);
8 }
9 // long double
10 Pt line_intersect(Pt a1, Pt a2, Pt b1, Pt b2) {
11     Pt da = mv(a1, a2), db = mv(b1, b2);
12     T det = da ^ db;
13     if (sgn(det) == 0) { // parallel
14         // return Pt(NAN, NAN);
15     }
16     T t = ((b1 - a1) ^ db) / det;
17     return a1 + da * t;
18 }

```

```

1 double point_segment_dist(Pt q0, Pt q1, Pt p) {
2     if (q0 == q1) {
3         double dx = double(p.x - q0.x);
4         double dy = double(p.y - q0.y);
5         return sqrt(dx * dx + dy * dy);
6     }
7     T d1 = (q1 - q0) * (p - q0);
8     T d2 = (q0 - q1) * (p - q1);
9     if (d1 >= 0 && d2 >= 0) {
10         double area = fabs(double((q1 - q0) ^ (p - q0)));
11         double base = sqrt(double(dis2(q0, q1)));
12         return area / base;
13     }
14     double dx0 = double(p.x - q0.x), dy0 = double(p.y -
15         q0.y);
16     double dx1 = double(p.x - q1.x), dy1 = double(p.y -
17         q1.y);
18     return min(sqrt(dx0 * dx0 + dy0 * dy0), sqrt(dx1 *
19         dx1 + dy1 * dy1));
20 }

```

7.5 Polygon Area

```

1 // 2 * area
2 T dbPoly_area(vector<Pt>& e) {
3     T res = 0;
4     int sz = e.size();
5     for (int i = 0; i < sz; i++) {
6         res += e[i] ^ e[(i + 1) % sz];
7     }
8     return abs(res);
9 }

```

7.6 Convex Hull

```

1 vector<Pt> convexHull(vector<Pt> pts) {
2     vector<Pt> hull;
3     sort(pts.begin(), pts.end());
4     for (int i = 0; i < 2; i++) {
5         int b = hull.size();
6         for (auto ei : pts) {
7             while (hull.size() - b >= 2 && ori(mv(hull[
8                 hull.size() - 2], hull.back()), mv(hull[
9                 hull.size() - 2], ei)) == -1) {
10                 hull.pop_back();
11             }
12             hull.emplace_back(ei);
13         }
14         hull.pop_back();
15         reverse(pts.begin(), pts.end());
16     }
17     return hull;
18 }

```

7.7 Point In Convex

```

1 bool point_in_convex(const vector<Pt> &C, Pt p, bool
2     strict = true) {
3     // only works when no three point are collinear
4     int n = C.size();
5     int a = 1, b = n - 1, r = !strict;
6     if (n == 0) return false;
7     if (n < 3) return r && onseg(p, C[0], C.back());
8     if (ori(mv(C[0], C[a]), mv(C[0], C[b])) > 0) swap(a
9         , b);
10     if (ori(mv(C[0], C[a]), mv(C[0], p)) >= r || ori(mv
11         (C[0], C[b]), mv(C[0], p)) <= -r) return false;
12     while (abs(a - b) > 1) {
13         int c = (a + b) / 2;
14         if (ori(mv(C[0], C[c]), mv(C[0], p)) > 0) b = c
15         ;
16         else a = c;
17     }
18     return ori(mv(C[a], C[b]), mv(C[a], p)) < r;
19 }

```

7.8 Point Segment Distance

7.9 Point in Polygon

```

1 short inPoly(vector<Pt>& pts, Pt p) {
2     // 0=Bound 1=In -1=Out
3     int n = pts.size();
4     for (int i = 0; i < pts.size(); i++) if (onseg(p,
5         pts[i], pts[(i + 1) % n])) return 0;
6     int cnt = 0;
7     for (int i = 0; i < pts.size(); i++) if (
8         line_intersect_check(p, Pt(p.x + 1, p.y + 2e9),
9         pts[i], pts[(i + 1) % n])) cnt ^= 1;
10     return (cnt ? 1 : -1);
11 }

```

7.10 Lower Concave Hull

```

1 struct Line {
2     mutable ll m, b, p;
3     bool operator<(const Line& o) const { return m < o.m;
4     }
5     bool operator<(ll x) const { return p < x; }
6 };
7
8 struct LineContainer : multiset<Line, less<>> {
9     // (for doubles, use inf = 1/.0, div(a,b) = a/b)
10     const ll inf = LLONG_MAX;
11     ll div(ll a, ll b) { // floored division
12         return a / b - ((a ^ b) < 0 && a % b); }
13     bool isect(iterator x, iterator y) {
14         if (y == end()) { x->p = inf; return false; }
15         if (x->m == y->m) x->p = x->b > y->b ? inf : -inf;
16         else x->p = div(y->b - x->b, x->m - y->m);
17         return x->p >= y->p;
18     }
19     void add(ll m, ll b) {
20         auto z = insert({m, b, 0}), y = z++, x = y;
21         while (isect(y, z)) z = erase(z);
22         if (x != begin() && isect(--x, y)) isect(x, y =
23             erase(y));
24         while ((y = x) != begin() && (--x)->p >= y->p)
25             isect(x, erase(y));
26     }
27     ll query(ll x) {
28         assert(!empty());
29         auto l = *lower_bound(x);
30         return l.m * x + l.b;
31     }
32 };

```

7.11 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:

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

7.12 Vector In Polygon

7.13 Minkowski Sum

```

1  /* convex hull Minkowski Sum*/
2  #define INF 10000000000000LL
3  int pos(const Pt& tp) {
4      if (tp.Y == 0) return tp.X > 0 ? 0 : 1;
5      return tp.Y > 0 ? 0 : 1;
6  }
7  #define N 300030
8  Pt pt[N], qt[N], rt[N];
9  LL Lx, Rx;
10 int dn, un;
11 inline bool cmp(Pt a, Pt b) {
12     int pa = pos(a), pb = pos(b);
13     if (pa == pb) return (a ^ b) > 0;
14     return pa < pb;
15 }
16 int minkowskiSum(int n, int m) {
17     int i, j, r, p, q, fi, fj;
18     for (i = 1, p = 0; i < n; i++) {
19         if (pt[i].Y < pt[p].Y ||
20             (pt[i].Y == pt[p].Y && pt[i].X < pt[p].X))
21             p = i;
22     }
23     for (i = 1, q = 0; i < m; i++) {
24         if (qt[i].Y < qt[q].Y ||
25             (qt[i].Y == qt[q].Y && qt[i].X < qt[q].X))
26             q = i;
27     }
28     rt[0] = pt[p] + qt[q];
29     r = 1;
30     i = p;
31     j = q;
32     fi = fj = 0;
33     while (1) {
34         if ((fj && j == q) ||
35             ((!fi || i != p) &&
36              cmp(pt[(p + 1) % n] - pt[p], qt[(q + 1) %
37                  m] - qt[q]))) {
38             rt[r] = rt[r - 1] + pt[(p + 1) % n] - pt[p];
39             p = (p + 1) % n;
40             fi = 1;
41         } else {
42             rt[r] = rt[r - 1] + qt[(q + 1) % m] - qt[q];
43             q = (q + 1) % m;
44             fj = 1;
45         }
46         if (r <= 1 || ((rt[r] - rt[r - 1]) ^ (rt[r - 1] -
47             rt[r - 2])) != 0) r++;
48         else rt[r - 1] = rt[r];
49         if (i == p && j == q) break;
50     }
51     return r - 1;
52 }
53 void initInConvex(int n) {
54     int i, p, q;
55     LL Ly, Ry;
56     Lx = INF;
57     Rx = -INF;
58     for (i = 0; i < n; i++) {
59         if (pt[i].X < Lx) Lx = pt[i].X;
60         if (pt[i].X > Rx) Rx = pt[i].X;
61     }
62     Ly = Ry = INF;
63     for (i = 0; i < n; i++) {
64         if (pt[i].X == Lx && pt[i].Y < Ly) {
65             Ly = pt[i].Y;
66             p = i;
67         }
68         if (pt[i].X == Rx && pt[i].Y < Ry) {
69             Ry = pt[i].Y;
70             q = i;
71         }
72     }
73 }

```

```

69     for (dn = 0, i = p; i != q; i = (i + 1) % n)
70         qt[dn++] = pt[i];
71     qt[dn] = pt[q];
72     Ly = Ry = -INF;
73     for (i = 0; i < n; i++) {
74         if (pt[i].X == Lx && pt[i].Y > Ly) {
75             Ly = pt[i].Y;
76             p = i;
77         }
78         if (pt[i].X == Rx && pt[i].Y > Ry) {
79             Ry = pt[i].Y;
80             q = i;
81         }
82     }
83     for (un = 0, i = p; i != q; i = (i + n - 1) % n)
84         rt[un++] = pt[i];
85     rt[un] = pt[q];
86 }
87 inline int inConvex(Pt p) {
88     int L, R, M;
89     if (p.X < Lx || p.X > Rx) return 0;
90     L = 0;
91     R = dn;
92     while (L < R - 1) {
93         M = (L + R) / 2;
94         if (p.X < qt[M].X) R = M;
95         else L = M;
96     }
97     if (tri(qt[L], qt[R], p) < 0) return 0;
98     L = 0;
99     R = un;
100    while (L < R - 1) {
101        M = (L + R) / 2;
102        if (p.X < rt[M].X) R = M;
103        else L = M;
104    }
105    if (tri(rt[L], rt[R], p) > 0) return 0;
106    return 1;
107 }
108 int main() {
109     int n, m, i;
110     Pt p;
111     scanf("%d", &n);
112     for (i = 0; i < n; i++) scanf("%lld%lld", &pt[i].X,
113         &pt[i].Y);
114     for (i = 0; i < m; i++) scanf("%lld%lld", &qt[i].X,
115         &qt[i].Y);
116     n = minkowskiSum(n, m);
117     for (i = 0; i < n; i++) pt[i] = rt[i];
118     scanf("%d", &m);
119     for (i = 0; i < m; i++) scanf("%lld%lld", &qt[i].X,
120         &qt[i].Y);
121     n = minkowskiSum(n, m);
122     for (i = 0; i < n; i++) pt[i] = rt[i];
123     initInConvex(n);
124     scanf("%d", &m);
125     for (i = 0; i < m; i++) {
126         scanf("%lld %lld", &p.X, &p.Y);
127         p.X *= 3;
128         p.Y *= 3;
129         puts(inConvex(p) ? "YES" : "NO");
130     }
131 }

```

7.14 Rotating SweepLine

7.15 Half Plane Intersection

```

1  const long double eps = 1e-9, inf = 1e9;
2  struct Point {
3      long double x, y;
4      explicit Point(long double x = 0, long double y =
5          0) : x(x), y(y) {}
6      friend Point operator+(const Point& p, const Point&
7          q) {
8          return Point(p.x + q.x, p.y + q.y);
9      }
10     friend Point operator-(const Point& p, const Point&
11         q) {
12         return Point(p.x - q.x, p.y - q.y);
13     }

```

```

10     }
11     friend Point operator*(const Point& p, const long
12         double& k) {
13         return Point(p.x * k, p.y * k);
14     }
15     friend long double dot(const Point& p, const Point&
16         q) {
17         return p.x * q.x + p.y * q.y;
18     }
19     friend long double cross(const Point& p, const
20         Point& q) {
21         return p.x * q.y - p.y * q.x;
22     }
23 };
24 struct Halfplane {
25     Point p, pq;
26     long double angle;
27     Halfplane() {}
28     Halfplane(const Point& a, const Point& b) : p(a),
29         pq(b - a) {
30         angle = atan2(pq.y, pq.x);
31     }
32     bool out(const Point& r) {
33         return cross(pq, r - p) < -eps;
34     }
35     bool operator<(const Halfplane& e) const {
36         return angle < e.angle;
37     }
38     friend Point inter(const Halfplane& s, const
39         Halfplane& t) {
40         long double alpha = cross((t.p - s.p), t.pq) /
41             cross(s.pq, t.pq);
42         return s.p + (s.pq * alpha);
43     }
44 };
45 vector<Point> hp_intersect(vector<Halfplane>& H) {
46     Point box[4] = { // Bounding box in CCW order
47         Point(-inf, inf),
48         Point(inf, inf),
49         Point(inf, -inf),
50         Point(-inf, -inf) };
51     for (int i = 0; i < 4; i++) { // Add bounding box
52         half-planes.
53         Halfplane aux(box[i], box[(i + 1) % 4]);
54         H.push_back(aux);
55     }
56     sort(H.begin(), H.end());
57     deque<Halfplane> dq;
58     int len = 0;
59     for (int i = 0; i < int(H.size()); i++) {
60         while (len > 1 && H[i].out(inter(dq[len - 1],
61             dq[len - 2]))) {
62             dq.pop_back();
63             --len;
64         }
65         while (len > 1 && H[i].out(inter(dq[0], dq[1])))
66             {
67                 dq.pop_front();
68                 --len;
69             }
70         if (len > 0 && fabs1(cross(H[i].pq, dq[len -
71             1].pq)) < eps) {
72             if (dot(H[i].pq, dq[len - 1].pq) < 0.0)
73                 return vector<Point>();
74             if (H[i].out(dq[len - 1].p)) {
75                 dq.pop_back();
76                 --len;
77             } else
78                 continue;
79         }
80         dq.push_back(H[i]);
81         ++len;
82     }
83     while (len > 2 && dq[0].out(inter(dq[len - 1], dq[
84         len - 2]))) {
85         dq.pop_back();
86         --len;
87     }
88     while (len > 2 && dq[len - 1].out(inter(dq[0], dq
89         [1]))) {
90         dq.pop_front();
91         --len;
92     }

```

```

80     }
81     if (len < 3) return vector<Point>();
82     vector<Point> ret(len);
83     for (int i = 0; i + 1 < len; i++) {
84         ret[i] = inter(dq[i], dq[i + 1]);
85     }
86     ret.back() = inter(dq[len - 1], dq[0]);
87     return ret;
88 }

```

7.16 Minimum Enclosing Circle

```

1 Pt circumcenter(Pt A, Pt B, Pt C) {
2     // a1(x-A.x) + b1(y-A.y) = c1
3     // a2(x-A.x) + b2(y-A.y) = c2
4     // solve using Cramer's rule
5     T a1 = B.x - A.x, b1 = B.y - A.y, c1 = dis2(A, B) /
6         2.0;
7     T a2 = C.x - A.x, b2 = C.y - A.y, c2 = dis2(A, C) /
8         2.0;
9     T D = Pt(a1, b1) ^ Pt(a2, b2);
10    T Dx = Pt(c1, b1) ^ Pt(c2, b2);
11    T Dy = Pt(a1, c1) ^ Pt(a2, c2);
12    if (D == 0) return Pt(-INF, -INF);
13    return A + Pt(Dx / D, Dy / D);
14 }
15 Pt center;
16 T r2;
17 void minEncloseCircle() {
18     mt19937 gen(chrono::steady_clock::now().
19         time_since_epoch().count());
20     shuffle(ALL(E), gen);
21     center = E[0], r2 = 0;
22     for (int i = 0; i < n; i++) {
23         if (dis2(center, E[i]) <= r2) continue;
24         center = E[i], r2 = 0;
25         for (int j = 0; j < i; j++) {
26             if (dis2(center, E[j]) <= r2) continue;
27             center = (E[i] + E[j]) / 2.0;
28             r2 = dis2(center, E[i]);
29             for (int k = 0; k < j; k++) {
30                 if (dis2(center, E[k]) <= r2) continue;
31                 center = circumcenter(E[i], E[j], E[k]);
32                 r2 = dis2(center, E[i]);
33             }
34         }
35     }
36 }

```

7.17 Heart

7.18 Tangents

7.19 Point In Circle

7.20 Union of Circles

7.21 Union of Polygons

7.22 Delaunay Triangulation

7.23 Triangulation Voronoi

7.24 External Bisector

7.25 Intersection Area of Polygon and Circle

7.26 3D Point

7.27 3D Convex Hull

8 Number Theory

8.1 FFT

```

1 typedef complex<double> cp;
2
3 const double pi = acos(-1);
4 const int NN = 131072;
5
6 struct FastFourierTransform{
7     /*

```

```

8      Iterative Fast Fourier Transform
9      How this works? Look at this
10     0th recursion 0(000) 1(001) 2(010) 3(011)
11           4(100) 5(101) 6(110) 7(111)
12     1th recursion 0(000) 2(010) 4(100) 6(110)
13           | 1(011) 3(011) 5(101) 7(111)
14     2th recursion 0(000) 4(100) | 2(010) 6(110)
15           | 1(011) 5(101) | 3(011) 7(111)
16     3th recursion 0(000) | 4(100) | 2(010) | 6(110)
17           | 1(011) | 5(101) | 3(011) | 7(111)
18     All the bits are reversed => We can save the
19     reverse of the numbers in an array!
20
21 */
22 int n, rev[NN];
23 cp omega[NN], iomega[NN];
24 void init(int n_){
25     n = n_;
26     for(int i = 0; i < n; i++){
27         //Calculate the nth roots of unity
28         omega[i] = cp(cos(2*pi*i/n), sin(2*pi*i/n));
29         iomega[i] = conj(omega[i]);
30     }
31     int k = __lg(n);
32     for(int i = 0; i < n; i++){
33         int t = 0;
34         for(int j = 0; j < k; j++){
35             if(i & (1<<j)) t |= (1<<(k-j-1));
36         }
37         rev[i] = t;
38     }
39 }
40
41 void transform(vector<cp> &a, cp* xomega){
42     for(int i = 0; i < n; i++){
43         if(i < rev[i]) swap(a[i], a[rev[i]]);
44     }
45     for(int len = 2; len <= n; len <= 1){
46         int mid = len >> 1;
47         int r = n/len;
48         for(int j = 0; j < n; j += len){
49             for(int i = 0; i < mid; i++){
50                 cp tmp = xomega[r*i] * a[j+mid+i];
51                 a[j+mid+i] = a[j+i] - tmp;
52                 a[j+i] = a[j+i] + tmp;
53             }
54         }
55     }
56 }
57
58 void fft(vector<cp> &a){ transform(a, omega); }
59 void ifft(vector<cp> &a){ transform(a, iomega); for(
60     int i = 0; i < n; i++) a[i] /= n; }
61 } FFT;
62
63 const int MAXN = 262144;
64 // (must be 2^k)
65 // 262144, 524288, 1048576, 2097152, 4194304
66 // before any usage, run pre_fft() first
67 typedef long double ld;
68 typedef complex<ld> cplx; //real() ,imag()
69 const ld PI = acos(-1);
70 const cplx I(0, 1);
71 cplx omega[MAXN+1];
72 void pre_fft(){
73     for(int i=0; i<=MAXN; i++) {
74         omega[i] = exp(i * 2 * PI / MAXN * I);
75     }
76 }
77
78 // n must be 2^k
79 void fft(int n, cplx a[], bool inv=false){
80     int basic = MAXN / n;
81     int theta = basic;
82     for (int m = n; m >= 2; m >= 1) {
83         int mh = m >> 1;
84         for (int i = 0; i < mh; i++) {
85             cplx w = omega[inv ? MAXN - (i * theta %
86                 MAXN) : i * theta % MAXN];
87             for (int j = i; j < n; j += m) {
88                 int k = j + mh;
89                 cplx x = a[j] - a[k];
90                 a[j] += a[k];
91                 a[k] = w * x;
92             }
93         }
94         theta = (theta * 2) % MAXN;
95     }
96     int i = 0;
97     for (int j = 1; j < n - 1; j++) {
98         for (int k = n >> 1; k > (i ^ k); k >>= 1);
99         if (j < i) swap(a[i], a[j]);
100     }
101     if(inv) {
102         for (i = 0; i < n; i++) a[i] /= n;
103     }
104 }
105
106 cplx arr[MAXN + 1];
107 inline void mul(int _n, long long a[], int _m, long long b
108     [], long long ans[]){
109     int n=1, sum = _n + _m - 1;
110     while(n < sum) n <= 1;
111     for(int i = 0; i < n; i++) {
112         double x= (i < _n ? a[i] : 0), y=(i < _m ? b[i]
113             : 0);
114         arr[i] = complex<double>(x + y, x - y);
115     }
116     fft(n, arr);
117     for(int i = 0; i < n; i++) arr[i]=arr[i]*arr[i];
118     fft(n, arr, true);
119     for(int i=0; i<sum; i++) ans[i]=(long long int)(arr[i
120         ].real() / 4 + 0.5);
121 }
122
123 long long a[MAXN];
124 long long b[MAXN];
125 long long ans[MAXN];
126 int a_length;
127 int b_length;

```

8.2 Pollard's rho

```

1 ll add(ll x, ll y, ll p) {
2     return (x + y) % p;
3 }
4 ll qMul(ll x, ll y, ll mod) {
5     ll ret = x * y - (ll)((long double)x / mod * y) *
6     mod;
7     return ret < 0 ? ret + mod : ret;
8 }
9 ll f(ll x, ll mod) { return add(qMul(x, x, mod), 1, mod
10 ); }
11 pollard_rho(ll n) {
12     if (!(n & 1)) return 2;
13     while (true) {
14         ll y = 2, x = rand() % (n - 1) + 1, res = 1;
15         for (int sz = 2; res == 1; sz *= 2) {
16             for (int i = 0; i < sz && res <= 1; i++) {
17                 x = f(x, n);
18                 res = __gcd(llabs(x - y), n);
19             }
20             y = x;
21         }
22         if (res != 0 && res != n) return res;
23     }
24 }
25 vector<ll> ret;
26 void fact(ll x) {
27     if (miller_rabin(x)) {
28         ret.push_back(x);
29         return;
30     }
31     ll f = pollard_rho(x);
32     fact(f);
33     fact(x / f);
34 }

```

8.3 Miller Rabin

```

1 // n < 4,759,123,141      3 : 2, 7, 61
2 // n < 1,122,004,669,633  4 : 2, 13, 23, 1662803
3 // n < 3,474,749,660,383  6 : pirmes <= 13
4 // n < 2^64              7 :
5 // 2, 325, 9375, 28178, 450775, 9780504, 1795265022

```

```

6 bool witness(ll a, ll n, ll u, int t){
7     if(!(a%n)) return 0;
8     ll x = mypow(a, u, n);
9     for(int i=0; i<t; i++){
10         ll nx = mul(x, x, n);
11         if(nx == 1 && x != 1 && x != n-1) return 1;
12         x = nx;
13     }
14     return x != 1;
15 }
16 bool miller_rabin(ll n, int s=100) {
17     // iterate s times of witness on n
18     // return 1 if prime, 0 otherwise
19     if(n<2) return 0;
20     if(!(n&1)) return n == 2;
21     ll u = n-1; int t=0;
22     while(!(u&1)) u>>=1, t++;
23     while(s--){
24         ll a = randll()%(n-1)+1;
25         if(witness(a, n, u, t)) return 0;
26     }
27     return 1;
28 }

```

8.4 Fast Power

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

8.5 Extend GCD

```

1 ll GCD;
2 pll extgcd(ll a, ll b) {
3     if (b == 0) {
4         GCD = a;
5         return pll{1, 0};
6     }
7     pll ans = extgcd(b, a % b);
8     return pll{ans.S, ans.F - a / b * ans.S};
9 }
10 pll bezout(ll a, ll b, ll c) {
11     bool negx = (a < 0), negy = (b < 0);
12     pll ans = extgcd(abs(a), abs(b));
13     if (c % GCD != 0) return pll{-LLINF, -LLINF};
14     return pll{ans.F * c / GCD * (negx ? -1 : 1),
15               ans.S * c / GCD * (negy ? -1 : 1)};
16 }
17 ll inv(ll a, ll p) {
18     if (p == 1) return -1;
19     pll ans = bezout(a % p, -p, 1);
20     if (ans == pll{-LLINF, -LLINF}) return -1;
21     return (ans.F % p + p) % p;
22 }

```

8.6 Mu + Phi

```

1 const int maxn = 1e6 + 5;
2 ll f[maxn];
3 vector<int> lpf, prime;
4 void build() {
5     lpf.clear(); lpf.resize(maxn, 1);
6     prime.clear();
7     f[1] = ...; /* mu[1] = 1, phi[1] = 1 */
8     for (int i = 2; i < maxn; i++) {
9         if (lpf[i] == 1) {
10             lpf[i] = i; prime.emplace_back(i);
11             f[i] = ...; /* mu[i] = 1, phi[i] = i-1 */
12         }
13         for (auto& j : prime) {
14             if (i*j >= maxn) break;
15             lpf[i*j] = j;
16             if (i % j == 0) f[i*j] = ...; /* 0, phi[i]*j */
17             else f[i*j] = ...; /* -mu[i], phi[i]*phi[j] */
18             if (j >= lpf[i]) break;
19         }
20     }
21 }

```

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:

$$a^p \equiv a \pmod{p} \text{ if } p \text{ is prime.}$$

- Euler function:

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

- Euler theorem:

$$a^{\phi(n)} \equiv 1 \pmod{n} \text{ if } \gcd(a, n) = 1.$$

- 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:

$$\sigma_x(n) = \sum_{d|n} d^x. \quad n = \prod_{i=1}^r p_i^{a_i}.$$

$$\sigma_x(n) = \prod_{i=1}^r \frac{p_i^{(a_i+1)x} - 1}{p_i^x - 1} \text{ if } x \neq 0. \quad \sigma_0(n) = \prod_{i=1}^r (a_i + 1).$$

- Chinese remainder theorem (Coprime Moduli):

$$x \equiv a_i \pmod{m_i}.$$

$$M = \prod m_i. \quad M_i = M/m_i. \quad t_i = M_i^{-1}.$$

$$x = kM + \sum a_i t_i M_i, \quad k \in \mathbb{Z}.$$

- 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.

$$x \equiv m_1 p + a_1 \equiv m_2 q + a_2 \pmod{\text{lcm}(m_1, m_2)}$$

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

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

- Important Multiplicative Functions + Properties:

- $\epsilon(n) = [n = 1]$
- $1(n) = 1$
- $id(n) = n$
- $\mu(n) = 0$ if n has squared prime factor
- $\mu(n) = (-1)^k$ if $n = p_1 p_2 \cdots p_k$
- $\epsilon = \mu * 1$
- $\phi = \mu * id$
- $[n = 1] = \sum_{d|n} \mu(d)$
- $[gcd = 1] = \sum_{d|gcd} \mu(d)$

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

8.8 Polynomial

```

1 const int maxk = 20;
2 const int maxn = 1<<maxk;
3 const ll LINF = 1e18;
4
5 /* P = r*2^k + 1
6 P          r      k      g
7 998244353      119  23      3
8 1004535809      479  21      3
9
10 P          r      k      g
11 3          1      1      2
12 5          1      2      2
13 17         1      4      3
14 97         3      5      5
15 193        3      6      5
16 257        1      8      3
17 7681       15     9     17
18 12289      3     12     11
19 40961      5     13      3
20 65537      1     16      3
21 786433     3     18     10
22 5767169    11     19      3

```



```

23 7340033          7  20  3
24 23068673        11 21  3
25 104857601       25 22  3
26 167772161       5  25  3
27 469762049       7  26  3
28 1004535809      479 21  3
29 2013265921      15 27 31
30 2281701377      17 27  3
31 3221225473       3  30  5
32 75161927681     35 31  3
33 77309411329      9  33  7
34 206158430209     3  36 22
35 2061584302081   15 37  7
36 2748779069441    5  39  3
37 6597069766657    3  41  5
38 39582418599937   9  42  5
39 79164837199873   9  43  5
40 263882790666241  15 44  7
41 1231453023109121 35 45  3
42 1337006139375617 19 46  3
43 3799912185593857 27 47  5
44 4222124650659841 15 48 19
45 7881299347898369 7  50  6
46 31525197391593473 7  52  3
47 180143985094819841 5  55  6
48 1945555039024054273 27 56  5
49 4179340454199820289 29 57  3
50 9097271247288401921 505 54 6 */
51
52 const int g = 3;
53 const ll MOD = 998244353;
54
55 ll pw(ll a, ll n) { /* fast pow */ }
56
57 #define siz(x) (int)x.size()
58
59 template<typename T>
60 vector<T>& operator+=(vector<T>& a, const vector<T>& b) {
61     {
62         if (siz(a) < siz(b)) a.resize(siz(b));
63         for (int i = 0; i < min(siz(a), siz(b)); i++) {
64             a[i] += b[i];
65             a[i] -= a[i] >= MOD ? MOD : 0;
66         }
67         return a;
68     }
69 }
70
71 template<typename T>
72 vector<T>& operator-=(vector<T>& a, const vector<T>& b) {
73     {
74         if (siz(a) < siz(b)) a.resize(siz(b));
75         for (int i = 0; i < min(siz(a), siz(b)); i++) {
76             a[i] -= b[i];
77             a[i] += a[i] < 0 ? MOD : 0;
78         }
79         return a;
80     }
81 }
82
83 template<typename T>
84 vector<T> operator-(const vector<T>& a) {
85     vector<T> ret(siz(a));
86     for (int i = 0; i < siz(a); i++) {
87         ret[i] = -a[i] < 0 ? -a[i] + MOD : -a[i];
88     }
89     return ret;
90 }
91
92 vector<ll> X, iX;
93 vector<int> rev;
94
95 void init_ntt() {
96     X.clear(); X.resize(maxn, 1); // x1 = g^((p-1)/n)
97     iX.clear(); iX.resize(maxn, 1);
98
99     ll u = pw(g, (MOD-1)/maxn);
100     ll iu = pw(u, MOD-2);
101
102     for (int i = 1; i < maxn; i++) {
103         X[i] = X[i-1] * u;
104         iX[i] = iX[i-1] * iu;
105         if (X[i] >= MOD) X[i] %= MOD;
106         if (iX[i] >= MOD) iX[i] %= MOD;
107     }
108 }
109
110 }
111
112 rev.clear(); rev.resize(maxn, 0);
113 for (int i = 1, hb = -1; i < maxn; i++) {
114     if (!(i & (i-1))) hb++;
115     rev[i] = rev[i ^ (1<<hb)] | (1<<(maxk-hb-1));
116 }
117
118 template<typename T>
119 void NTT(vector<T>& a, bool inv=false) {
120     int _n = (int)a.size();
121     int k = __lg(_n) + ((1<<__lg(_n)) != _n);
122     int n = 1<<k;
123     a.resize(n, 0);
124
125     short shift = maxk-k;
126     for (int i = 0; i < n; i++)
127         if (i > (rev[i]>>shift))
128             swap(a[i], a[rev[i]>>shift]);
129
130     for (int len = 2, half = 1, div = maxn>>1; len <= n; len<=1, half<=1, div>=1) {
131         for (int i = 0; i < n; i += len) {
132             for (int j = 0; j < half; j++) {
133                 T u = a[i+j];
134                 T v = a[i+j+half] * (inv ? iX[j*div] : X[j*div]) % MOD;
135                 a[i+j] = (u+v >= MOD ? u+v-MOD : u+v);
136                 a[i+j+half] = (u-v < 0 ? u-v+MOD : u-v);
137             }
138         }
139     }
140
141     if (inv) {
142         T dn = pw(n, MOD-2);
143         for (auto& x : a) {
144             x *= dn;
145             if (x >= MOD) x %= MOD;
146         }
147     }
148 }
149
150 template<typename T>
151 inline void resize(vector<T>& a) {
152     int cnt = (int)a.size();
153     for (; cnt > 0; cnt--) if (a[cnt-1]) break;
154     a.resize(max(cnt, 1));
155 }
156
157 template<typename T>
158 vector<T>& operator*(vector<T>& a, vector<T> b) {
159     int na = (int)a.size();
160     int nb = (int)b.size();
161     a.resize(na + nb - 1, 0);
162     b.resize(na + nb - 1, 0);
163
164     NTT(a); NTT(b);
165     for (int i = 0; i < (int)a.size(); i++) {
166         a[i] *= b[i];
167         if (a[i] >= MOD) a[i] %= MOD;
168     }
169     NTT(a, true);
170
171     resize(a);
172     return a;
173 }
174
175 template<typename T>
176 void inv(vector<T>& ia, int N) {
177     vector<T> _a(move(ia));
178     ia.resize(1, pw(_a[0], MOD-2));
179     vector<T> a(1, -_a[0] + (-_a[0] < 0 ? MOD : 0));
180
181     for (int n = 1; n < N; n<=1) {
182         // n -> 2*n
183         // ia' = ia(2-a*ia);
184
185         for (int i = n; i < min(siz(_a), (n<<1)); i++)
186             a.emplace_back(-_a[i] + (-_a[i] < 0 ? MOD : 0));
187
188         vector<T> tmp = ia;
189         ia *= a;
190         ia.resize(n<<1);
191     }
192 }

```

```

181     ia[0] = ia[0] + 2 >= MOD ? ia[0] + 2 - MOD : ia
      [0] + 2;
182     ia *= tmp;
183     ia.resize(n<<1);
184 }
185 ia.resize(N);
186 }
187
188 template<typename T>
189 void mod(vector<T>& a, vector<T>& b) {
190     int n = (int)a.size()-1, m = (int)b.size()-1;
191     if (n < m) return;
192
193     vector<T> ra = a, rb = b;
194     reverse(ra.begin(), ra.end()); ra.resize(min(n+1, n
      -m+1));
195     reverse(rb.begin(), rb.end()); rb.resize(min(m+1, n
      -m+1));
196
197     inv(rb, n-m+1);
198
199     vector<T> q = move(ra);
200     q *= rb;
201     q.resize(n-m+1);
202     reverse(q.begin(), q.end());
203
204     q *= b;
205     a -= q;
206     resize(a);
207 }
208
209 /* Kitamasa Method (Fast Linear Recurrence):
210 Find a[K] (Given a[j] = c[0]a[j-N] + ... + c[N-1]a[j
      -1])
211 Let B(x) = x^N - c[N-1]x^(N-1) - ... - c[1]x^1 - c[0]
212 Let R(x) = x^K mod B(x) (get x^K using fast pow and
      use poly mod to get R(x))
213 Let r[i] = the coefficient of x^i in R(x)
214 => a[K] = a[0]r[0] + a[1]r[1] + ... + a[N-1]r[N-1] */

```

9 Linear Algebra

9.1 Gaussian-Jordan Elimination

```

1 int n;
2 vector<vector<ll>>> v;
3 void gauss(vector<vector<ll>>& v) {
4     int r = 0;
5     for (int i = 0; i < n; i++) {
6         bool ok = false;
7         for (int j = r; j < n; j++) {
8             if (v[j][i] == 0) continue;
9             swap(v[j], v[r]);
10            ok = true;
11            break;
12        }
13        if (!ok) continue;
14        ll div = inv(v[r][i]);
15        for (int j = 0; j < n + 1; j++) {
16            v[r][j] *= div;
17            if (v[r][j] >= MOD) v[r][j] %= MOD;
18        }
19        for (int j = 0; j < n; j++) {
20            if (j == r) continue;
21            ll t = v[j][i];
22            for (int k = 0; k < n + 1; k++) {
23                v[j][k] -= v[r][k] * t % MOD;
24                if (v[j][k] < 0) v[j][k] += MOD;
25            }
26        }
27        r++;
28    }
29 }

```

9.2 Determinant

1. Use GJ Elimination, if there's any row consists of only 0, then $\det = 0$, otherwise $\det = \text{product of diagonal elements}$.

2. Properties of det:

- Transpose: Unchanged
- Row Operation 1 - Swap 2 rows: $-\det$
- Row Operation 2 - $k\vec{r}_i$: $k \times \det$
- Row Operation 3 - $k\vec{r}_i$ add to \vec{r}_j : Unchanged

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 $\leq n \approx n/((\ln n) - 1)$
 $\pi(100) = 25, \pi(200) = 46$
 $\pi(500) = 95, \pi(1000) = 168$
 $\pi(2000) = 303, \pi(4000) = 550$
 $\pi(10^4) = 1229, \pi(10^5) = 9592$
 $\pi(10^6) = 78498, \pi(10^7) = 664579$