Contents

1 Reminder

1.1 Bug List

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

1.2 OwO

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

2 Basic

2.1 Vimrc

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

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

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

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

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

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

2.2 Runcpp.sh

```bash
#! /bin/bash
clear
echo "Start compiling $1..."
echo
g++ -O2 -std=c++20 -Wall -Wextra -Wshadow $2/$1 -o $2/
    out
if [ "$?" -ne 0 ]
then
```

```
8       exit 1
9  fi
10 echo
11 echo "Done compiling"
12 echo "========================="
13 echo
14 echo "Input file:"
15 echo
16 cat $2/in.txt
17 echo
18 echo "========================="
19 echo
20 declare startTime=`date +%s%N`
21 $2/out < $2/in.txt > $2/out.txt
22 declare endTime=`date +%s%N`
23 delta=`expr $endTime - $startTime`
24 delta=`expr $delta / 1000000`
25 cat $2/out.txt
26 echo
27 echo "time: $delta ms"
```

2.3   Stress

```
1  g++ gen.cpp -o gen.out
2  g++ ac.cpp -o ac.out
3  g++ wa.cpp -o wa.out
4  for ((i=0;;i++))
5  do
6      echo "$i"
7      ./gen.out > in.txt
8      ./ac.out < in.txt > ac.txt
9      ./wa.out < in.txt > wa.txt
10     diff ac.txt wa.txt || break
11 done
```

2.4   PBDS

```
1  #include <bits/extc++.h>
2  using namespace __gnu_pbds;
3
4  // map
5  tree<int, int, less<>, rb_tree_tag,
       tree_order_statistics_node_update> tr;
6  tr.order_of_key(element);
7  tr.find_by_order(rank);
8
9  // set
10 tree<int, null_type, less<>, rb_tree_tag,
       tree_order_statistics_node_update> tr;
11 tr.order_of_key(element);
12 tr.find_by_order(rank);
13
14 // hash table
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;
            // Big First
22 __gnu_pbds::priority_queue<int, greater<int>> small_q;
        // Small First
23 q1.join(q2);
                                          // join
```

2.5   Random

```
1  mt19937 gen(chrono::steady_clock::now().
       time_since_epoch().count());
2  uniform_int_distribution<int> dis(1, 100);
3  cout << dis(gen) << endl;
4  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 -
               1);
19
20         long long ret = 0;
21         while (l <= r) ret += bit[r], r -= r & -r;
22         return ret;
23     }
24 } 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
           + n + 1, 1); }
5
6      int fh(int x) { return (h[x] == x ? x : h[x] = fh(h
           [x])); }
7
8      bool mer(int x, int y) {
9          x = fh(x), y = fh(y);
10         if (x == y) return 0;
11         if (s[x] < s[y]) swap(x, y);
12         s[x] += s[y], s[y] = 0;
13         h[y] = x;
14         return 1;
15     }
16 } 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
                   /= 2;
15     }
16
17     int query(int l, int r) {
18         l += n, r += n;
19         int ret = inf;
20         while (l < r) {
21             if (l & 1)
22                 ret = min(ret, seg[l++]);
23             if (r & 1)
24                 ret = min(ret, seg[--r]);
25             l /= 2, r /= 2;
26         }
27         return ret;
28     }
29 } bm;
```

3.4   Treap

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

3.5   Persistent Treap

```cpp
struct node {
    node *l, *r;
    char c;
    int v, sz;
    node(char x = '$') : c(x), v(mt()), sz(1) {
        l = r = nullptr;
    }
    node(node* p) { *this = *p; }
    void pull() {
```

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

3.6   Li Chao Tree

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

3.7   Sparse Table

```cpp
const int lgmx = 19;

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

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

```
14  int query(int l, int r) {
15      int ln = len(l, r);
16      int lg = __lg(ln);
17      return min(spt[lg][l], spt[lg][r - (1 << lg) + 1]);
18  }
```

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] +=
            sz[a];
14      return true;
15  };
16  inline void undo() {
17      auto [a, b, s] = his.back();
18      his.pop_back();
19      dsu[a] = a, sz[b] = s;
20  }
21  #define m ((l + r) >> 1)
22  void insert(int ql, int qr, P<int> x, int i = 1, int l
        = 0, int r = q) {
23      // debug(ql, qr, x); return;
24      if (qr <= l || r <= ql) return;
25      if (ql <= l && r <= qr) {
26          arr[i].push_back(x);
27          return;
28      }
29      if (qr <= m)
30          insert(ql, qr, x, i << 1, l, m);
31      else if (m <= ql)
32          insert(ql, qr, x, i << 1 | 1, m, r);
33      else {
34          insert(ql, qr, x, i << 1, l, m);
35          insert(ql, qr, x, i << 1 | 1, m, r);
36      }
37  }
38  void traversal(V<int>& ans, int i = 1, int l = 0, int r
        = q) {
39      int opcnt = 0;
40      // debug(i, l, r);
41      for (auto [a, b] : arr[i])
42          if (merge(a, b))
43              opcnt++, cnt--;
44      if (r - l == 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],
           vis[N + 5];
3      struct edge {
4          int to, cap, rev, cost;
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, 2 * n + 5)
10             par[i] = p_i[i] = vis[i] = 0;
11     }
12     void add(int a, int b, int c, int d) {
13         path[a].pb({b, c, sz(path[b]), d});
14         path[b].pb({a, 0, sz(path[a]) - 1, -d});
15     }
16     void spfa() {
17         FOR(i, 0, n * 2 + 5)
18             dis[i] = INF,
19             vis[i] = 0;
20         dis[s] = 0;
21         queue<int> q;
22         q.push(s);
23         while (!q.empty()) {
24             int now = q.front();
25             q.pop();
26             vis[now] = 0;
27             for (int i = 0; i < sz(path[now]); i++) {
28                 edge e = path[now][i];
29                 if (e.cap > 0 && dis[e.to] > dis[now] +
                       e.cost) {
30                     dis[e.to] = dis[now] + e.cost;
31                     par[e.to] = now;
32                     p_i[e.to] = i;
33                     if (vis[e.to] == 0) {
34                         vis[e.to] = 1;
35                         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 };
```

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[i] = vy[i] = 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);
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 };
```

4.4   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]] ==
                            -1) {
51                         dis[my[y]] = dis[x] + 1;
52                         q.push(my[y]);
53                     }
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;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],
           pa[maxn * 2];
13    int flo_from[maxn * 2][maxn + 1], S[maxn * 2], vis[
           maxn * 2];
14    vector<int> flo[maxn * 2];
15    queue<int> q;
16    int e_delta(const edge &e) { return lab[e.u] + lab[
           e.v] - g[e.u][e.v].w * 2; }
17    void update_slack(int u, int x) {
18        if (!slack[x] || e_delta(g[u][x]) < e_delta(g[
               slack[x]][x])) slack[x] = u;
19    }
20    void set_slack(int x) {
21        slack[x] = 0;
22        for (int u = 1; u <= n; ++u)
23            if (g[u][x].w > 0 && st[u] != x && S[st[u]]
                   == 0)
24                update_slack(u, x);
25    }
26    void q_push(int x) {
27        if (x <= n)
28            q.push(x);
29        else
30            for (size_t i = 0; i < flo[x].size(); i++)
31                q_push(flo[x][i]);
32    }
33    void set_st(int x, int b) {
```

```cpp
 33        st[x] = b;
 34        if (x > n)
 35            for (size_t i = 0; i < flo[x].size(); ++i)
                    set_st(flo[x][i], b);
 36    }
 37    int get_pr(int b, int xr) {
 38        int pr = find(flo[b].begin(), flo[b].end(), xr)
                - flo[b].begin();
 39        if (pr % 2 == 1) {
 40            reverse(flo[b].begin() + 1, flo[b].end());
 41            return (int)flo[b].size() - pr;
 42        }
 43        return pr;
 44    }
 45    void set_match(int u, int v) {
 46        match[u] = g[u][v].v;
 47        if (u <= n) return;
 48        edge e = g[u][v];
 49        int xr = flo_from[u][e.u], pr = get_pr(u, xr);
 50        for (int i = 0; i < pr; ++i) set_match(flo[u][i
              ], flo[u][i ^ 1]);
 51        set_match(xr, v);
 52        rotate(flo[u].begin(), flo[u].begin() + pr, flo
              [u].end());
 53    }
 54    void augment(int u, int v) {
 55        for (;;) {
 56            int xnv = st[match[u]];
 57            set_match(u, v);
 58            if (!xnv) return;
 59            set_match(xnv, st[pa[xnv]]);
 60            u = st[pa[xnv]], v = xnv;
 61        }
 62    }
 63    int get_lca(int u, int v) {
 64        static int t = 0;
 65        for (++t; u || v; swap(u, v)) {
 66            if (u == 0) continue;
 67            if (vis[u] == t) return u;
 68            vis[u] = t;
 69            u = st[match[u]];
 70            if (u) u = st[pa[u]];
 71        }
 72        return 0;
 73    }
 74    void add_blossom(int u, int lca, int v) {
 75        int b = n + 1;
 76        while (b <= n_x && st[b]) ++b;
 77        if (b > n_x) ++n_x;
 78        lab[b] = 0, S[b] = 0;
 79        match[b] = match[lca];
 80        flo[b].clear();
 81        flo[b].push_back(lca);
 82        for (int x = u, y; x != lca; x = st[pa[y]])
 83            flo[b].push_back(x), flo[b].push_back(y =
                  st[match[x]]), q_push(y);
 84        reverse(flo[b].begin() + 1, flo[b].end());
 85        for (int x = v, y; x != lca; x = st[pa[y]])
 86            flo[b].push_back(x), flo[b].push_back(y =
                  st[match[x]]), q_push(y);
 87        set_st(b, b);
 88        for (int x = 1; x <= n_x; ++x) g[b][x].w = g[x
              ][b].w = 0;
 89        for (int x = 1; x <= n; ++x) flo_from[b][x] =
              0;
 90        for (size_t i = 0; i < flo[b].size(); ++i) {
 91            int xs = flo[b][i];
 92            for (int x = 1; x <= n_x; ++x)
 93                if (g[b][x].w == 0 || e_delta(g[xs][x])
                      < e_delta(g[b][x]))
                        g[b][x] = g[xs][x], g[x][b] = g[x][
                          xs];
 95            for (int x = 1; x <= n; ++x)
 96                if (flo_from[xs][x]) flo_from[b][x] =
                      xs;
 97        }
 98        set_slack(b);
 99    }
100    void expand_blossom(int b) {
101        for (size_t i = 0; i < flo[b].size(); ++i)
102            set_st(flo[b][i], flo[b][i]);
103        int xr = flo_from[b][g[b][pa[b]].u], pr =
              get_pr(b, xr);
104        for (int i = 0; i < pr; i += 2) {
105            int xs = flo[b][i], xns = flo[b][i + 1];
106            pa[xs] = g[xns][xs].u;
107            S[xs] = 1, S[xns] = 0;
108            slack[xs] = 0, set_slack(xns);
109            q_push(xns);
110        }
111        S[xr] = 1, pa[xr] = pa[b];
112        for (size_t i = pr + 1; i < flo[b].size(); ++i)
              {
113            int xs = flo[b][i];
114            S[xs] = -1, set_slack(xs);
115        }
116        st[b] = 0;
117    }
118    bool on_found_edge(const edge &e) {
119        int u = st[e.u], v = st[e.v];
120        if (S[v] == -1) {
121            pa[v] = e.u, S[v] = 1;
122            int nu = st[match[v]];
123            slack[v] = slack[nu] = 0;
124            S[nu] = 0, q_push(nu);
125        } else if (S[v] == 0) {
126            int lca = get_lca(u, v);
127            if (!lca)
128                return augment(u, v), augment(v, u),
                      true;
129            else
130                add_blossom(u, lca, v);
131        }
132        return false;
133    }
134    bool matching() {
135        memset(S + 1, -1, sizeof(int) * n_x);
136        memset(slack + 1, 0, sizeof(int) * n_x);
137        q = queue<int>();
138        for (int x = 1; x <= n_x; ++x)
139            if (st[x] == x && !match[x]) pa[x] = 0, S[x
                  ] = 0, q_push(x);
140        if (q.empty()) return false;
141        for (;;) {
142            while (q.size()) {
143                int u = q.front();
144                q.pop();
145                if (S[st[u]] == 1) continue;
146                for (int v = 1; v <= n; ++v)
147                    if (g[u][v].w > 0 && st[u] != st[v
                          ]) {
148                        if (e_delta(g[u][v]) == 0) {
149                            if (on_found_edge(g[u][v]))
                                return true;
150                        } else
                              update_slack(u, st[v]);
151                    }
152            }
153            int d = inf;
154            for (int b = n + 1; b <= n_x; ++b)
155                if (st[b] == b && S[b] == 1) d = min(d,
                      lab[b] / 2);
156            for (int x = 1; x <= n_x; ++x)
157                if (st[x] == x && slack[x]) {
158                    if (S[x] == -1)
159                        d = min(d, e_delta(g[slack[x]][
                              x]));
160                    else if (S[x] == 0)
161                        d = min(d, e_delta(g[slack[x]][
                              x]) / 2);
162                }
163            for (int u = 1; u <= n; ++u) {
164                if (S[st[u]] == 0) {
165                    if (lab[u] <= d) return 0;
166                    lab[u] -= d;
167                } else if (S[st[u]] == 1)
168                    lab[u] += d;
169            }
170            for (int b = n + 1; b <= n_x; ++b)
171                if (st[b] == b) {
172                    if (S[st[b]] == 0)
173                        lab[b] += d * 2;
174                    else if (S[st[b]] == 1)
```

```
176                    lab[b] -= d * 2;
177                }
178            q = queue<int>();
179            for (int x = 1; x <= n_x; ++x)
180                if (st[x] == x && slack[x] && st[slack[
                       x]] != x && e_delta(g[slack[x]][x])
                       == 0)
181                    if (on_found_edge(g[slack[x]][x]))
182                        return true;
183                for (int b = n + 1; b <= n_x; ++b)
184                    if (st[b] == b && S[b] == 1 && lab[b]
                           == 0) expand_blossom(b);
185                }
186            return false;
187        }
188        pair<long long, int> solve() {
189            memset(match + 1, 0, sizeof(int) * n);
190            n_x = n;
191            int n_matches = 0;
192            long long tot_weight = 0;
193            for (int u = 0; u <= n; ++u) st[u] = u, flo[u].
                   clear();
194            int w_max = 0;
195            for (int u = 1; u <= n; ++u)
196                for (int v = 1; v <= n; ++v) {
197                    flo_from[u][v] = (u == v ? u : 0);
198                    w_max = max(w_max, g[u][v].w);
199                }
200            for (int u = 1; u <= n; ++u) lab[u] = w_max;
201            while (matching()) ++n_matches;
202            for (int u = 1; u <= n; ++u)
203                if (match[u] && match[u] < u)
204                    tot_weight += g[u][match[u]].w;
205            return make_pair(tot_weight, n_matches);
206        }
207        void add_edge(int ui, int vi, int wi) { g[ui][vi].w
               = g[vi][ui].w = wi; }
208        void init(int _n) {
209            n = _n;
210            for (int u = 1; u <= n; ++u)
211                for (int v = 1; v <= n; ++v)
212                    g[u][v] = edge(u, v, 0);
213        }
    };
```

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
4  M = Max Matching
5  Cv = Min V Cover
6  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              for (auto& e : g[u]) {
64                  int v = e.first;
65                  ll w = e.second;
66                  if (dis[v] > dis[u] + w) {
67                      dis[v] = dis[u] + w;
68                      pa[v] = u;
69                      if (rlx == n) negCycle[v] = true;
70                  }
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 sccnt, 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      sccnt = 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].
       size()) gg;
33     while (!stk.empty()) {
34         int u = stk.top();
35         stk.pop();
36         cout << u << ' ';
37     }
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
24 int main() {
25     WiwiHorz
26     init();
27
28     REP(i, m) {
29         int u, v;
30         cin >> u >> v;
31         if (u == v) continue;
32         adj[--u][--v]++;
33     }
34
35     dp[0][1] = 1;
36     FOR(i, 1, n, 1) {
37         dp[i][1] = 0;
38         dp[i][1|(1<<i)] = adj[0][i];
39     }
40     FOR(msk, 1, (1<<n), 1) {
41         if (msk == 1) continue;
42         dp[0][msk] = 0;
43     }
44
45
46     DP(n-1, (1<<n)-1);
47     cout << dp[n-1][(1<<n)-1] << endl;
48
49     return 0;
50 }
```

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; ll d;
6      nd(int ui=0,int vi=0,ll di=INF){ u=ui; v=vi; d=di;
         }
7    };
8    struct heap{ nd* edge; int dep; heap* chd[4]; };
9    static int cmp(heap* a,heap* b)
10   { return a->edge->d > b->edge->d; }
11   struct node{
12     int v; ll d; heap* H; nd* E;
13     node(){}
14     node(ll _d,int _v,nd* _E){ d =_d; v=_v; E=_E; }
15     node(heap* _H,ll _d){ H=_H; d=_d; }
16     friend bool operator<(node a,node b)
17     { return a.d>b.d; }
18   };
19   int n,k,s,t,dst[N]; nd *nxt[N];
20   vector<nd*> g[N],rg[N]; heap *nullNd,*head[N];
21   void init(int _n,int _k,int _s,int _t){
22     n=_n; k=_k; s=_s; t=_t;
23     for(int i=1;i<=n;i++){
24       g[i].clear(); rg[i].clear();
25       nxt[i]=NULL; head[i]=NULL; dst[i]=-1;
26     }
27   }
28   void addEdge(int ui,int vi,ll di){
29     nd* e=new nd(ui,vi,di);
30     g[ui].push_back(e); rg[vi].push_back(e);
31   }
32   queue<int> dfsQ;
33   void dijkstra(){
```

```
34      while(dfsQ.size()) dfsQ.pop();
35      priority_queue<node> Q; Q.push(node(0,t,NULL));
36      while (!Q.empty()){
37        node p=Q.top(); Q.pop(); if(dst[p.v]!=-1)continue
            ;
38        dst[p.v]=p.d; nxt[p.v]=p.E; dfsQ.push(p.v);
39        for(auto e:rg[p.v]) Q.push(node(p.d+e->d,e->u,e))
            ;
40      }
41    }
42   heap* merge(heap* curNd,heap* newNd){
43      if(curNd==nullNd) return newNd;
44      heap* root=new heap;memcpy(root,curNd,sizeof(heap))
          ;
45      if(newNd->edge->d<curNd->edge->d){
46        root->edge=newNd->edge;
47        root->chd[2]=newNd->chd[2];
48        root->chd[3]=newNd->chd[3];
49        newNd->edge=curNd->edge;
50        newNd->chd[2]=curNd->chd[2];
51        newNd->chd[3]=curNd->chd[3];
52      }
53      if(root->chd[0]->dep<root->chd[1]->dep)
54        root->chd[0]=merge(root->chd[0],newNd);
55      else root->chd[1]=merge(root->chd[1],newNd);
56      root->dep=max(root->chd[0]->dep,
57              root->chd[1]->dep)+1;
58      return root;
59    }
60   vector<heap*> V;
61   void build(){
62     nullNd=new heap; nullNd->dep=0; nullNd->edge=new nd
          ;
63     fill(nullNd->chd,nullNd->chd+4,nullNd);
64     while(not dfsQ.empty()){
65       int u=dfsQ.front(); dfsQ.pop();
66       if(!nxt[u]) head[u]=nullNd;
67       else head[u]=head[nxt[u]->v];
68       V.clear();
69       for(auto&& e:g[u]){
70         int v=e->v;
71         if(dst[v]==-1) continue;
72         e->d+=dst[v]-dst[u];
73         if(nxt[u]!=e){
74           heap* p=new heap;fill(p->chd,p->chd+4,nullNd)
              ;
75           p->dep=1; p->edge=e; V.push_back(p);
76         }
77       }
78       if(V.empty()) continue;
79       make_heap(V.begin(),V.end(),cmp);
80 #define L(X) ((X<<1)+1)
81 #define R(X) ((X<<1)+2)
82       for(size_t i=0;i<V.size();i++){
83         if(L(i)<V.size()) V[i]->chd[2]=V[L(i)];
84         else V[i]->chd[2]=nullNd;
85         if(R(i)<V.size()) V[i]->chd[3]=V[R(i)];
86         else V[i]->chd[3]=nullNd;
87       }
88       head[u]=merge(head[u],V.front());
89     }
90   }
91   vector<ll> ans;
92   void first_K(){
93     ans.clear(); priority_queue<node> Q;
94     if(dst[s]==-1) return;
95     ans.push_back(dst[s]);
96     if(head[s]!=nullNd)
97       Q.push(node(head[s],dst[s]+head[s]->edge->d));
98     for(int _=1;_<k and not Q.empty();_++){
99       node p=Q.top(),q; Q.pop(); ans.push_back(p.d);
100      if(head[p.H->edge->v]!=nullNd){
101        q.H=head[p.H->edge->v]; q.d=p.d+q.H->edge->d;
102        Q.push(q);
103      }
104      for(int i=0;i<4;i++)
105        if(p.H->chd[i]!=nullNd){
106          q.H=p.H->chd[i];
107          q.d=p.d-p.H->edge->d+p.H->chd[i]->edge->d;
108          Q.push(q);
109  } }   }
110  void solve(){ // ans[i] stores the i-th shortest path
111      dijkstra(); build();
112      first_K(); // ans.size() might less than k
113    }
114 } 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  }
```

- $x_u - x_v \le c \Rightarrow$ add(v, u, c)

- $x_u - x_v \ge c \Rightarrow$ add(u, v, -c)

- $x_u - x_v = c \Rightarrow$ add(v, u, c), add(u, v -c)

- $x_u \ge c \Rightarrow$ add super vertex $x_0 = 0$, then $x_u - x_0 \ge c \Rightarrow$ add(u, 0, -c)

- Don't for get non-negative constraints for every variable if specified implicitly.

- Interval sum $\Rightarrow$ Use prefix sum to transform into differential constraints. Don't for get $S_{i+1} - S_i \ge 0$ if x_i needs to be non-negative.

- $\frac{x_u}{x_v} \le c \Rightarrow \log x_u - \log x_v \le \log c$

6 String

6.1 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
          .size()), mod(_mod) {
9          Cexp.assign(n, 0);
10         hs.assign(n, 0);
11         Cexp[0] = 1;
12         for (int i = 1; i < n; i++) {
13             Cexp[i] = Cexp[i - 1] * C;
14             if (Cexp[i] >= mod) Cexp[i] %= mod;
15         }
16         hs[0] = id(s[0]);
17         for (int i = 1; i < n; i++) {
18             hs[i] = hs[i - 1] * C + id(s[i]);
19             if (hs[i] >= mod) hs[i] %= mod;
20         }
21     }
22     inline ll query(int l, int r) {
23         ll res = hs[r] - (l ? hs[l - 1] * Cexp[r - l +
             1] : 0);
24         res = (res % mod + mod) % mod;
25         return res;
26     }
27 };
```

6.2 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 }
```

6.3   KMP

```
1  int n, m;
2  string s, p;
3  vector<int> f;
4  void build() {
5      f.clear(); f.resize(m, 0);
6      int ptr = 0; for (int i = 1; i < m; i++) {
7          while (ptr && p[i] != p[ptr]) ptr = f[ptr-1];
8          if (p[i] == p[ptr]) ptr++;
9          f[i] = ptr;
10 }}
11 void init() {
12     cin >> s >> p;
13     n = (int)s.size();
14     m = (int)p.size();
15     build(); }
16 void solve() {
17     int ans = 0, pi = 0;
18     for (int si = 0; si < n; si++) {
19         while (pi && s[si] != p[pi]) pi = f[pi-1];
20         if (s[si] == p[pi]) pi++;
21         if (pi == m) ans++, pi = f[pi-1];
22     }
23 cout << ans << endl; }
```

6.4   Z Value

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

6.5   Manacher

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

6.6   Suffix Array

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

6.7   SA-IS

```
1  const int N=300010;
2  struct SA{
3  #define REP(i,n) for(int i=0;i<int(n);i++)
4  #define REP1(i,a,b) for(int i=(a);i<=int(b);i++)
5    bool _t[N*2]; int _s[N*2],_sa[N*2];
6    int _c[N*2],x[N],_p[N],_q[N*2],hei[N],r[N];
7    int operator [](int i){ return _sa[i]; }
8    void build(int *s,int n,int m){
9      memcpy(_s,s,sizeof(int)*n);
10     sais(_s,_sa,_p,_q,_t,_c,n,m); mkhei(n);
11   }
12   void mkhei(int n){
13     REP(i,n) r[_sa[i]]=i;
14     hei[0]=0;
15     REP(i,n) if(r[i]) {
16       int ans=i>0?max(hei[r[i-1]]-1,0):0;
17       while(_s[i+ans]==_s[_sa[r[i]-1]+ans]) ans++;
```

```
18      hei[r[i]]=ans;
19    }
20  }
21  void sais(int *s,int *sa,int *p,int *q,bool *t,int *c
        ,int n,int z){
22    bool uniq=t[n-1]=true,neq;
23    int nn=0,nmxz=-1,*nsa=sa+n,*ns=s+n,lst=-1;
24 #define MS0(x,n) memset((x),0,n*sizeof(*(x)))
25 #define MAGIC(XD) MS0(sa,n);\
26 memcpy(x,c,sizeof(int)*z); XD;\
27 memcpy(x+1,c,sizeof(int)*(z-1));\
28 REP(i,n) if(sa[i]&&!t[sa[i]-1]) sa[x[s[sa[i]-1]]++]=sa[
        i]-1;\
29 memcpy(x,c,sizeof(int)*z);\
30 for(int i=n-1;i>=0;i--) if(sa[i]&&t[sa[i]-1]) sa[--x[s[
        sa[i]-1]]]=sa[i]-1;
31    MS0(c,z); REP(i,n) uniq&=++c[s[i]]<2;
32    REP(i,z-1) c[i+1]+=c[i];
33    if(uniq) { REP(i,n) sa[--c[s[i]]]=i; return; }
34    for(int i=n-2;i>=0;i--)
35      t[i]=(s[i]==s[i+1]?t[i+1]:s[i]<s[i+1]);
36    MAGIC(REP1(i,1,n-1) if(t[i]&&!t[i-1]) sa[--x[s[i
        ]]]=p[q[i]=nn++]=i);
37    REP(i,n) if(sa[i]&&t[sa[i]]&&!t[sa[i]-1]){
38      neq=lst<0||memcmp(s+sa[i],s+lst,(p[q[sa[i]]+1]-sa
          [i])*sizeof(int));
39      ns[q[lst=sa[i]]]=nmxz+=neq;
40    }
41    sais(ns,nsa,p+nn,q+n,t+n,c+z,nn,nmxz+1);
42    MAGIC(for(int i=nn-1;i>=0;i--) sa[--x[s[p[nsa[i
        ]]]]]=p[nsa[i]]);
43  }
44 }sa;
45 int H[N],SA[N],RA[N];
46 void suffix_array(int* ip,int len){
47  // should padding a zero in the back
48  // ip is int array, len is array length
49  // ip[0..n-1] != 0, and ip[len]=0
50  ip[len++]=0; sa.build(ip,len,128);
51  memcpy(H,sa.hei+1,len<<2); memcpy(SA,sa._sa+1,len<<2)
      ;
52  for(int i=0;i<len;i++) RA[i]=sa.r[i]-1;
53  // resulting height, sa array \in [0,len)
54 }
```

6.8 Minimum Rotation

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

6.9 Aho Corasick

```
1 struct ACautomata{
2   struct Node{
3     int cnt;
4     Node *go[26], *fail, *dic;
5     Node (){
6       cnt = 0; fail = 0; dic=0;
7       memset(go,0,sizeof(go));
8     }
9   }pool[1048576],*root;
10  int nMem;
11  Node* new_Node(){
12    pool[nMem] = Node();
13    return &pool[nMem++];
14  }
15  void init() { nMem = 0; root = new_Node(); }
16  void add(const string &str) { insert(root,str,0); }
17  void insert(Node *cur, const string &str, int pos){
18    for(int i=pos;i<str.size();i++){
```

```
19      if(!cur->go[str[i]-'a'])
20        cur->go[str[i]-'a'] = new_Node();
21      cur=cur->go[str[i]-'a'];
22    }
23    cur->cnt++;
24  }
25  void make_fail(){
26    queue<Node*> que;
27    que.push(root);
28    while (!que.empty()){
29      Node* fr=que.front(); que.pop();
30      for (int i=0; i<26; i++){
31        if (fr->go[i]){
32          Node *ptr = fr->fail;
33          while (ptr && !ptr->go[i]) ptr = ptr->fail;
34          fr->go[i]->fail=ptr=(ptr?ptr->go[i]:root);
35          fr->go[i]->dic=(ptr->cnt?ptr:ptr->dic);
36          que.push(fr->go[i]);
37  } } } }
38 }AC;
```

7 Geometry

7.1 Basic Operations

```
1 typedef long long T;
2 // typedef long double T;
3 const long double eps = 1e-8;
4
5 short sgn(T x) {
6    if (abs(x) < eps) return 0;
7    return x < 0 ? -1 : 1;
8 }
9
10 struct Pt {
11 T x, y;
12 Pt(T _x=0, T _y=0):x(_x), y(_y) {}
13 Pt operator+(Pt a) { return Pt(x+a.x, y+a.y); }
14 Pt operator-(Pt a) { return Pt(x-a.x, y-a.y); }
15 Pt operator*(T a)  { return Pt(x*a, y*a); }
16 Pt operator/(T a)  { return Pt(x/a, y/a); }
17 T operator*(Pt a)  { return x*a.x + y*a.y; }
18 T operator^(Pt a)  { return x*a.y - y*a.x; }
19 bool operator<(Pt a)
20    { return x < a.x || (x == a.x && y < a.y); }
21 //return sgn(x-a.x) < 0 || (sgn(x-a.x) == 0 && sgn(y-a.
      y) < 0); }
22 bool operator==(Pt a)
23    { return sgn(x-a.x) == 0 && sgn(y-a.y) == 0; }
24 };
25
26 Pt mv(Pt a, Pt b) { return b-a; }
27 T len2(Pt a) { return a*a; }
28 T dis2(Pt a, Pt b) { return len2(b-a); }
29
30 short ori(Pt a, Pt b) { return ((a^b)>0) - ((a^b)<0); }
31 bool onseg(Pt p, Pt l1, Pt l2) {
32    Pt a = mv(p, l1), b = mv(p, l2);
33    return ((a^b) == 0) && ((a*b) <= 0);
34 }
```

7.2 InPoly

```
1 short inPoly(Pt p) {
2 // 0=Bound 1=In -1=Out
3 REP(i, n) if (onseg(p, E[i], E[(i+1)%n])) return 0;
4 int cnt = 0;
5 REP(i, n) if (banana(p, Pt(p.x+1, p.y+2e9),
6                E[i], E[(i+1)%n])) cnt ^= 1;
7 return (cnt ? 1 : -1);
8 }
```

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(ALL(E), [&](const Pt& a, const Pt& b){
7      if (ud(a) != ud(b)) return ud(a) < ud(b);
8      return (a^b) > 0;
9  });
```

7.4 Line Intersect Check

```
1  inline bool banana(Pt p1, Pt p2, Pt q1, Pt q2) {
2  if (onseg(p1, q1, q2) || onseg(p2, q1, q2) ||
3      onseg(q1, p1, p2) || onseg(q2, p1, p2)) {
4      return true;
5  }
6  Pt p = mv(p1, p2), q = mv(q1, q2);
7  return (ori(p, mv(p1, q1)) * ori(p, mv(p1, q2)) < 0 &&
8          ori(q, mv(q1, p1)) * ori(q, mv(q1, p2)) < 0);
9  }
```

7.5 Line Intersection

```
1  // T: long double
2  Pt bananaPoint(Pt p1, Pt p2, Pt q1, Pt q2) {
3  if (onseg(q1, p1, p2)) return q1;
4  if (onseg(q2, p1, p2)) return q2;
5  if (onseg(p1, q1, q2)) return p1;
6  if (onseg(p2, q1, q2)) return p2;
7  double s = abs(mv(p1, p2) ^ mv(p1, q1));
8  double t = abs(mv(p1, p2) ^ mv(p1, q2));
9  return q2 * (s/(s+t)) + q1 * (t/(s+t));
10 }
```

7.6 Convex Hull

```
1  vector<Pt> hull;
2  void convexHull() {
3  hull.clear(); sort(ALL(E));
4  REP(t, 2) {
5      int b = SZ(hull);
6      Each(ei, E) {
7          while (SZ(hull) - b >= 2 &&
8                  ori(mv(hull[SZ(hull)-2], hull.back()),
9                      mv(hull[SZ(hull)-2], ei)) == -1) {
10             hull.pop_back();
11         }
12         hull.eb(ei);
13     }
14     hull.pop_back();
15     reverse(ALL(E));
16 } }
```

7.7 Lower Concave Hull

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

```
25   ll query(ll x) {
26     assert(!empty());
27     auto l = *lower_bound(x);
28     return l.m * x + l.b;
29   }
30 };
```

7.8 Polygon Area

```
1  T dbarea(vector<Pt>& e) {
2  ll res = 0;
3  REP(i, SZ(e)) res += e[i]^e[(i+1)%SZ(e)];
4  return abs(res);
5  }
```

7.9 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:

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

7.10 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)/2.0;
6  T a2 = C.x-A.x, b2 = C.y-A.y, c2 = dis2(A, C)/2.0;
7  T D  = Pt(a1, b1) ^ Pt(a2, b2);
8  T Dx = Pt(c1, b1) ^ Pt(c2, b2);
9  T Dy = Pt(a1, c1) ^ Pt(a2, c2);
10 if (D == 0) return Pt(-INF, -INF);
11 return A + Pt(Dx/D, Dy/D);
12 }
13 Pt center; T r2;
14 void minEncloseCircle() {
15 mt19937 gen(chrono::steady_clock::now().
       time_since_epoch().count());
16 shuffle(ALL(E), gen);
17 center = E[0], r2 = 0;
18
19 for (int i = 0; i < n; i++) {
20     if (dis2(center, E[i]) <= r2) continue;
21     center = E[i], r2 = 0;
22     for (int j = 0; j < i; j++) {
23         if (dis2(center, E[j]) <= r2) continue;
24         center = (E[i] + E[j]) / 2.0;
25         r2 = dis2(center, E[i]);
26         for (int k = 0; k < j; k++) {
27             if (dis2(center, E[k]) <= r2) continue;
28             center = circumcenter(E[i], E[j], E[k]);
29             r2 = dis2(center, E[i]);
30         }
31     }
32 } }
```

7.11 PolyUnion

```
1  struct PY{
2    int n; Pt pt[5]; double area;
3    Pt& operator[](const int x){ return pt[x]; }
4    void init(){ //n,pt[0~n-1] must be filled
5      area=pt[n-1]^pt[0];
6      for(int i=0;i<n-1;i++) area+=pt[i]^pt[i+1];
7      if((area/=2)<0)reverse(pt,pt+n),area=-area;
8    }
9  };
10 PY py[500]; pair<double,int> c[5000];
11 inline double segP(Pt &p,Pt &p1,Pt &p2){
12   if(dcmp(p1.x-p2.x)==0) return (p.y-p1.y)/(p2.y-p1.y);
13   return (p.x-p1.x)/(p2.x-p1.x);
14 }
```

```
15  double polyUnion(int n){ //py[0~n-1] must be filled
16    int i,j,ii,jj,ta,tb,r,d; double z,w,s,sum=0,tc,td;
17    for(i=0;i<n;i++) py[i][py[i].n]=py[i][0];
18    for(i=0;i<n;i++){
19      for(ii=0;ii<py[i].n;ii++){
20        r=0;
21        c[r++]=make_pair(0.0,0); c[r++]=make_pair(1.0,0);
22        for(j=0;j<n;j++){
23          if(i==j) continue;
24          for(jj=0;jj<py[j].n;jj++){
25            ta=dcmp(tri(py[i][ii],py[i][ii+1],py[j][jj]))
               ;
26            tb=dcmp(tri(py[i][ii],py[i][ii+1],py[j][jj
               +1]));
27            if(ta==0 && tb==0){
28              if((py[j][jj+1]-py[j][jj])*(py[i][ii+1]-py[
                 i][ii])>0&&j<i){
29                c[r++]=make_pair(segP(py[j][jj],py[i][ii
                   ],py[i][ii+1]),1);
30                c[r++]=make_pair(segP(py[j][jj+1],py[i][
                   ii],py[i][ii+1]),-1);
31              }
32            }else if(ta>=0 && tb<0){
33              tc=tri(py[j][jj],py[j][jj+1],py[i][ii]);
34              td=tri(py[j][jj],py[j][jj+1],py[i][ii+1]);
35              c[r++]=make_pair(tc/(tc-td),1);
36            }else if(ta<0 && tb>=0){
37              tc=tri(py[j][jj],py[j][jj+1],py[i][ii]);
38              td=tri(py[j][jj],py[j][jj+1],py[i][ii+1]);
39              c[r++]=make_pair(tc/(tc-td),-1);
40        } } }
41        sort(c,c+r);
42        z=min(max(c[0].first,0.0),1.0); d=c[0].second; s
           =0;
43        for(j=1;j<r;j++){
44          w=min(max(c[j].first,0.0),1.0);
45          if(!d) s+=w-z;
46          d+=c[j].second; z=w;
47        }
48        sum+=(py[i][ii]^py[i][ii+1])*s;
49      }
50    }
51    return sum/2;
52  }
```

7.12 Minkowski Sum

```
1  /* convex hull Minkowski Sum*/
2  #define INF 100000000000000LL
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) ) p=i; }
21   for(i=1,q=0;i<m;i++){
22     if( qt[i].Y<qt[q].Y ||
23     (qt[i].Y==qt[q].Y && qt[i].X<qt[q].X) ) q=i; }
24   rt[0]=pt[p]+qt[q];
25   r=1; i=p; j=q; fi=fj=0;
26   while(1){
27     if((fj&&j==q) ||
28     ( (!fi||i!=p) &&
29     cmp(pt[(p+1)%n]-pt[p],qt[(q+1)%m]-qt[q]) ) ){
30       rt[r]=rt[r-1]+pt[(p+1)%n]-pt[p];
31       p=(p+1)%n;
32       fi=1;
33     }else{
34       rt[r]=rt[r-1]+qt[(q+1)%m]-qt[q];
35       q=(q+1)%m;
36       fj=1;
37     }
38     if(r<=1 || ((rt[r]-rt[r-1])^(rt[r-1]-rt[r-2]))
        !=0) r++;
39     else rt[r-1]=rt[r];
40     if(i==p && j==q) break;
41   }
42   return r-1;
43 }
44 void initInConvex(int n){
45   int i,p,q;
46   LL Ly,Ry;
47   Lx=INF; Rx=-INF;
48   for(i=0;i<n;i++){
49     if(pt[i].X<Lx) Lx=pt[i].X;
50     if(pt[i].X>Rx) Rx=pt[i].X;
51   }
52   Ly=Ry=INF;
53   for(i=0;i<n;i++){
54     if(pt[i].X==Lx && pt[i].Y<Ly){ Ly=pt[i].Y; p=i;
        }
55     if(pt[i].X==Rx && pt[i].Y<Ry){ Ry=pt[i].Y; q=i;
        }
56   }
57   for(dn=0,i=p;i!=q;i=(i+1)%n){ qt[dn++]=pt[i]; }
58   qt[dn]=pt[q]; Ly=Ry=-INF;
59   for(i=0;i<n;i++){
60     if(pt[i].X==Lx && pt[i].Y>Ly){ Ly=pt[i].Y; p=i;
        }
61     if(pt[i].X==Rx && pt[i].Y>Ry){ Ry=pt[i].Y; q=i;
        }
62   }
63   for(un=0,i=p;i!=q;i=(i+n-1)%n){ rt[un++]=pt[i]; }
64   rt[un]=pt[q];
65 }
66 inline int inConvex(Pt p){
67   int L,R,M;
68   if(p.X<Lx || p.X>Rx) return 0;
69   L=0;R=dn;
70   while(L<R-1){ M=(L+R)/2;
71     if(p.X<qt[M].X) R=M; else L=M; }
72   if(tri(qt[L],qt[R],p)<0) return 0;
73   L=0;R=un;
74   while(L<R-1){ M=(L+R)/2;
75     if(p.X<rt[M].X) R=M; else L=M; }
76   if(tri(rt[L],rt[R],p)>0) return 0;
77   return 1;
78 }
79 int main(){
80   int n,m,i;
81   Pt p;
82   scanf("%d",&n);
83   for(i=0;i<n;i++) scanf("%lld%lld",&pt[i].X,&pt[i].Y
      );
84   scanf("%d",&m);
85   for(i=0;i<m;i++) scanf("%lld%lld",&qt[i].X,&qt[i].Y
      );
86   n=minkowskiSum(n,m);
87   for(i=0;i<n;i++) pt[i]=rt[i];
88   scanf("%d",&m);
89   for(i=0;i<m;i++) scanf("%lld%lld",&qt[i].X,&qt[i].Y
      );
90   n=minkowskiSum(n,m);
91   for(i=0;i<n;i++) pt[i]=rt[i];
92   initInConvex(n);
93   scanf("%d",&m);
94   for(i=0;i<m;i++){
95     scanf("%lld %lld",&p.X,&p.Y);
96     p.X*=3; p.Y*=3;
97     puts(inConvex(p)?"YES":"NO");
98   }
99 }
```

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

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

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             */
18         else f[i*j] = ...;  /* -mu[i], phi[i]*phi[j] */
19         if (j >= lpf[i]) break;
20 } } }
```

8.7 Other Formulas

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

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

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

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

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

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

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

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

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

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

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

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

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

8.8 Polynomial

```
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
```

```cpp
12289                   3   12  11
40961                   5   13  3
65537                   1   16  3
786433                  3   18  10
5767169                 11  19  3
7340033                 7   20  3
23068673                11  21  3
104857601               25  22  3
167772161               5   25  3
469762049               7   26  3
1004535809              479 21  3
2013265921              15  27  31
2281701377              17  27  3
3221225473              3   30  5
75161927681             35  31  3
77309411329             9   33  7
206158430209            3   36  22
2061584302081           15  37  7
2748779069441           5   39  3
6597069766657           3   41  5
39582418599937          9   42  5
79164837199873          9   43  5
263882790666241         15  44  7
1231453023109121        35  45  3
1337006139375617        19  46  3
3799912185593857        27  47  5
4222124650659841        15  48  19
7881299347898369        7   50  6
31525197391593473       7   52  3
180143985094819841      5   55  6
1945555039024054273 27  56  5
4179340454199820289 29  57  3
9097271247288401921 505 54  6 */

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

    resize(a);
    return a;
}

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

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

        for (int i = n; i < min(siz(_a), (n<<1)); i++)
```

```
176        a.emplace_back(-_a[i] + (-_a[i] < 0 ? MOD :
               0));
177
178        vector<T> tmp = ia;
179        ia *= a;
180        ia.resize(n<<1);
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; vector<vector<ll> > v;
2  void gauss(vector<vector<ll>>& v) {
3  int r = 0;
4  for (int i = 0; i < n; i++) {
5      bool ok = false;
6      for (int j = r; j < n; j++) {
7          if (v[j][i] == 0) continue;
8          swap(v[j], v[r]);
9          ok = true; break;
10     }
11     if (!ok) continue;
12     ll div = inv(v[r][i]);
13     for (int j = 0; j < n+1; j++) {
14         v[r][j] *= div;
15         if (v[r][j] >= MOD) v[r][j] %= MOD;
16     }
17     for (int j = 0; j < n; j++) {
18         if (j == r) continue;
19         ll t = v[j][i];
20         for (int k = 0; k < n+1; k++) {
21             v[j][k] -= v[r][k] * t % MOD;
22             if (v[j][k] < 0) v[j][k] += MOD;
23     } }
24     r++;
25 } }
```

9.2   Determinant

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

2. Properties of det:

   - Transpose: Unchanged
   - Row Operation 1 - Swap 2 rows: $-det$
   - Row Operation 2 - $k\overrightarrow{r_i}$: $k \times det$
   - Row Operation 3 - $k\overrightarrow{r_i}$ add to $\overrightarrow{r_j}$: Unchaged

10   Combinatorics

10.1   Catalan Number

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

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

10.2   Burnside's Lemma

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

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

11   Special Numbers

11.1   Fibonacci Series

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

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

11.2   Prime Numbers

- First 50 prime numbers:

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

- Very large prime numbers:

  1000001333   1000500889   2500001909
  2000000659   900004151   850001359

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