Random Useful Resources (from oiwiki etc.)

ethening

The Chinese University of Hong Kong

January 11, 2024

Contents

Start

Data Structure

OD Tree

```

struct Node_t {
  int l, r;
  mutable int v;

  Node_t(const int &il, const int &ir, const int &iv) : l(il), r(ir), v(iv) {}

  bool operator<(const Node_t &o) const { return l < o.l; }
};

auto split(int x) {
  if (x > n) return odt.end();
  auto it = --odt.upper_bound(Node_t{x, 0, 0});
  if (it->l == x) return it;
  int l = it->l, r = it->r, v = it->v;
  odt.erase(it);
  odt.insert(Node_t(l, x - 1, v));
  return odt.insert(Node_t(x, r, v)).first;
}

void assign(int l, int r, int v) {
  auto itr = split(r + 1), itl = split(l);
  odt.erase(itl, itr);
  odt.insert(Node_t(l, r, v));
}

void performance(int l, int r) {
  auto itr = split(r + 1), itl = split(l);
  for (; itl != itr; ++itl) {
    // Perform Operations here
  }
}
```

Math

Matrix Inverse

Given matrix A, find inverse B. You are given matrix C which is matrix B edited at most 12 position.

jiangly’s team solution

```
#include <bits/stdc++.h>

using i64 = long long;

constexpr int P = 1000000007;

std::mt19937 rng(std::chrono::steady_clock::now().time_since_epoch().count());

int power(int a, int b) {
    int res = 1;
    for (; b; b /= 2, a = 1LL * a * a % P) {
        if (b % 2) {
            res = 1LL * res * a % P;
        }
    }
    return res;
}

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

```

```cpp
    int n;
    std::cin >> n;

    std::vector A(n, std::vector<int>(n));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            std::cin >> A[i][j];
            // A[i][j] = (i == j);
        }
    }
    std::vector C(n, std::vector<int>(n));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            std::cin >> C[i][j];
            // C[i][j] = (i == j);
        }
    }
    // for (int i = 0; i < 12; i++) {
    //     C[rng() % n][rng() % n] = rng() % P;
    // }

    std::vector<int> idr(n), idc(n);
    for (int t = 0; t < 10; t++) {
        std::vector<int> v(n);
        for (int i = 0; i < n; i++) {
            v[i] = rng() % P;
        }
        std::vector<int> vA(n), vAC(n), Av(n), CAv(n);
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                vA[j] = (vA[j] + 1LL * v[i] * A[i][j]) % P;
            }
        }
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                vAC[j] = (vAC[j] + 1LL * vA[i] * C[i][j]) % P;
            }
        }
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                Av[i] = (Av[i] + 1LL * A[i][j] * v[j]) % P;
            }
        }
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                CAv[i] = (CAv[i] + 1LL * C[i][j] * Av[j]) % P;
            }
        }
        for (int i = 0; i < n; i++) {
            if (vAC[i] != v[i]) {
                idc[i] = 1;
            }
            if (CAv[i] != v[i]) {
                idr[i] = 1;
            }
        }
    }

    // for (int i = 0; i < n; i++) {
    //     std::cout << idr[i];
    // }
    // std::cout << "\n";
    // for (int i = 0; i < n; i++) {
    //     std::cout << idc[i];
    // }
    // std::cout << "\n";

    std::vector<int> row, col;
    int nr = 0, nc = 0;
    for (int i = 0; i < n; i++) {
        if (idr[i]) {
```

```
            idr[i] = nr++;
            row.push_back(i);
        } else {
            idr[i] = -1;
        }
        if (idc[i]) {
            idc[i] = nc++;
            col.push_back(i);
        } else {
            idc[i] = -1;
        }
    }

    int tot = nr * nc;
    int rank = 0;
    std::vector f(tot, std::vector<int>(tot + 1));
    while (tot > rank) {
        // std::cerr << "rank : " << rank << "\n";
        std::vector<int> v(n);
        for (int i = 0; i < n; i++) {
            v[i] = rng() % P;
        }
        std::vector<int> vA(n);
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                vA[j] = (vA[j] + 1LL * v[i] * A[i][j]) % P;
            }
        }
        for (int j = 0; j < n; j++) {
            if (idc[j] == -1) {
                continue;
            }
            int res = 0;
            std::vector<int> g(tot + 1);
            for (int i = 0; i < n; i++) {
                if (idr[i] == -1) {
                    res = (res + 1LL * vA[i] * C[i][j]) % P;
                } else {
                    g[idr[i] * nc + idc[j]] = vA[i];
                }
            }
            g[tot] = (v[j] - res + P) % P;
            // for (int i = 0; i <= tot; i++) {
            //     std::cerr << g[i] << " \n"[i == tot];
            // }
            for (int i = 0; i < tot; i++) {
                if (g[i] != 0) {
                    if (f[i][i] == 0) {
                        int v = power(g[i], P - 2);
                        for (int j = 0; j <= tot; j++) {
                            g[j] = 1LL * g[j] * v % P;
                        }
                        f[i] = g;
                        for (int j = 0; j < i; j++) {
                            int x = f[j][i];
                            for (int k = j; k <= tot; k++) {
                                f[j][k] = (f[j][k] + 1LL * (P - x) * g[k]) % P;
                            }
                        }
                        rank++;
                        break;
                    }
                    int x = g[i];
                    for (int j = i; j <= tot; j++) {
                        g[j] = (g[j] + 1LL * (P - x) * f[i][j]) % P;
                    }
                }
            }
        }
    }

```

```
    std::vector<std::array<int, 3>> ans;
    for (int i = 0; i < tot; i++) {
        if (f[i][tot] != C[row[i / nc]][col[i % nc]]) {
            ans.push_back({row[i / nc] + 1, col[i % nc] + 1, f[i][tot]});
        }
    }
    std::cout << ans.size() << "\n";
    for (auto [x, y, z] : ans) {
        std::cout << x << " " << y << " " << z << "\n";
    }

    return 0;
}
```

Graph

LGV Lemma

简介

Lindström–Gessel–Viennot lemma，即 LGV 引理，可以用来处理有向无环图上不相交路径计数等问题。

LGV 引理仅适用于 **有向无环图**。

题意：有一个 $n \times m$ 的格点棋盘，其中某些格子可走，某些格子不可走。有一只海龟从 (x, y) 只能走到 $(x+1, y)$ 和 $(x, y+1)$ 的位置，求海龟从 $(1, 1)$ 到 (n, m) 的不相交路径数对 $10^9 + 7$ 取模之后的结果。$2 \le n, m \le 3000$。

比较直接的 LGV 引理的应用。考虑所有合法路径，发现从 $(1, 1)$ 出发一定要经过 $A = \{(1, 2), (2, 1)\}$，而到达终点一定要经过 $B = \{(n-1, m), (n, m-1)\}$，则 A, B 可立即选定。应用 LGV 引理可得答案为：

$$\begin{vmatrix} f(a_1, b_1) & f(a_1, b_2) \\ f(a_2, b_1) & f(a_2, b_2) \end{vmatrix} = f(a_1, b_1) \times f(a_2, b_2) - f(a_1, b_2) \times f(a_2, b_1)$$

其中 $f(a, b)$ 为图上 $a \to b$ 的路径数，带有障碍格点的路径计数问题可以直接做一个 $O(nm)$ 的 dp，则 f 易求。最终复杂度 $O(nm)$。

```
#include <cstring>
#include <iostream>
#include <vector>

using namespace std;

using ll = long long;
const int MOD = 1e9 + 7;
const int SIZE = 3010;

char board[SIZE][SIZE];
int dp[SIZE][SIZE];

int f(int x1, int y1, int x2, int y2) {
  memset(dp, 0, sizeof dp);

  dp[x1][y1] = board[x1][y1] == '.';
  for (int i = 1; i <= x2; i++) {
    for (int j = 1; j <= y2; j++) {
      if (board[i][j] == '#') {
        continue;
      }
      dp[i][j] = (dp[i][j] + dp[i - 1][j]) % MOD;
      dp[i][j] = (dp[i][j] + dp[i][j - 1]) % MOD;
    }
  }
  return dp[x2][y2] % MOD;
}

int main() {
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
  cout.tie(nullptr);
```

```
  int n, m;
  cin >> n >> m;

  for (int i = 1; i <= n; i++) {
    cin >> (board[i] + 1);
  }

  ll f11 = f(1, 2, n - 1, m);
  ll f12 = f(1, 2, n, m - 1);
  ll f21 = f(2, 1, n - 1, m);
  ll f22 = f(2, 1, n, m - 1);

  ll ans = ((f11 * f22) % MOD - (f12 * f21) % MOD + MOD) % MOD;
  cout << ans << '\n';

  return 0;
}
```

题意：有一个 $n \times n$ 的棋盘，一个棋子从 (x, y) 只能走到 $(x, y+1)$ 或 $(x+1, y)$，有 k 个棋子，一开始第 i 个棋子放在 $(1, a_i)$，最终要到 (n, b_i)，路径要两两不相交，求方案数对 10^9+7 取模。$1 \le n \le 10^5$，$1 \le k \le 100$，保证 $1 \le a_1 < a_2 < \cdots < a_n \le n$，$1 \le b_1 < b_2 < \cdots < b_n \le n$。

观察到如果路径不相交就一定是 a_i 到 b_i，因此 LGV 引理中一定有 $\sigma(S)_i = i$，不需要考虑符号问题。边权设为 1，直接套用引理即可。

从 $(1, a_i)$ 到 (n, b_j) 的路径条数相当于从 $n-1+b_j-a_i$ 步中选 $n-1$ 步向下走，所以 $e(A_i, B_j) = \binom{n-1+b_j-a_i}{n-1}$。

行列式可以使用高斯消元求。

复杂度为 $O(n + k(k^2 + \log p))$，其中 $\log p$ 是求逆元复杂度。

```
#include <algorithm>
#include <cstdio>

typedef long long ll;

const int K = 105;
const int N = 100005;
const int mod = 1e9 + 7;

int T, n, k, a[K], b[K], fact[N << 1], m[K][K];

int qpow(int x, int y) {
  int out = 1;
  while (y) {
    if (y & 1) out = (ll)out * x % mod;
    x = (ll)x * x % mod;
    y >>= 1;
  }
  return out;
}

int c(int x, int y) {
  return (ll)fact[x] * qpow(fact[y], mod - 2) % mod *
         qpow(fact[x - y], mod - 2) % mod;
}

int main() {
  fact[0] = 1;
  for (int i = 1; i < N * 2; ++i) fact[i] = (ll)fact[i - 1] * i % mod;

  scanf("%d", &T);

  while (T--) {
    scanf("%d%d", &n, &k);

    for (int i = 1; i <= k; ++i) scanf("%d", a + i);
    for (int i = 1; i <= k; ++i) scanf("%d", b + i);

    for (int i = 1; i <= k; ++i) {
      for (int j = 1; j <= k; ++j) {
```

```
        if (a[i] <= b[j])
          m[i][j] = c(b[j] - a[i] + n - 1, n - 1);
        else
          m[i][j] = 0;
      }
    }

    for (int i = 1; i < k; ++i) {
      if (!m[i][i]) {
        for (int j = i + 1; j <= k; ++j) {
          if (m[j][i]) {
            std::swap(m[i], m[j]);
            break;
          }
        }
      }
      if (!m[i][i]) continue;
      int inv = qpow(m[i][i], mod - 2);
      for (int j = i + 1; j <= k; ++j) {
        if (!m[j][i]) continue;
        int mul = (ll)m[j][i] * inv % mod;
        for (int p = i; p <= k; ++p) {
          m[j][p] = (m[j][p] - (ll)m[i][p] * mul % mod + mod) % mod;
        }
      }
    }

    int ans = 1;

    for (int i = 1; i <= k; ++i) ans = (ll)ans * m[i][i] % mod;

    printf("%d\n", ans);
  }

  return 0;
}
```

Flow with bound

在阅读这篇文章之前请先阅读 最大流 并确保自己熟练掌握最大流算法。

概述

上下界网络流本质是给流量网络的每一条边设置了流量上界 $c(u,v)$ 和流量下界 $b(u,v)$。也就是说，一种可行的流必须满足 $b(u,v) \leq f(u,v) \leq c(u,v)$。同时必须满足除了源点和汇点之外的其余点流量平衡。

根据题目要求，我们可以使用上下界网络流解决不同问题。

无源汇上下界可行流

给定无源汇流量网络 G。询问是否存在一种标定每条边流量的方式，使得每条边流量满足上下界同时每一个点流量平衡。

不妨假设每条边已经流了 $b(u,v)$ 的流量，设其为初始流。同时我们在新图中加入 u 连向 v 的流量为 $c(u,v)-b(u,v)$ 的边。考虑在新图上进行调整。

由于最大流需要满足初始流量平衡条件（最大流可以看成是下界为 0 的上下界最大流），但是构造出来的初始流很有可能不满足初始流量平衡。假设一个点初始流入流量减初始流出流量为 M。

若 $M=0$，此时流量平衡，不需要附加边。

若 $M>0$，此时入流量过大，需要新建附加源点 S'，S' 向其连流量为 M 的附加边。

若 $M<0$，此时出流量过大，需要新建附加汇点 T'，其向 T' 连流量为 $-M$ 的附加边。

如果附加边满流，说明这一个点的流量平衡条件可以满足，否则这个点的流量平衡条件不满足。（因为原图加上附加流之后才会满足原图中的流量平衡。）

在建图完毕之后跑 S' 到 T' 的最大流，若 S' 连出去的边全部满流，则存在可行流，否则不存在。

有源汇上下界可行流

给定有源汇流量网络 G。询问是否存在一种标定每条边流量的方式，使得每条边流量满足上下界同时除了源点和汇点每一个点流量平衡。

假设源点为 S，汇点为 T。

则我们可以加入一条 T 到 S 的上界为 ∞，下界为 0 的边转化为无源汇上下界可行流问题。

若有解，则 S 到 T 的可行流流量等于 T 到 S 的附加边的流量。

有源汇上下界最大流

给定有源汇流量网络 G。询问是否存在一种标定每条边流量的方式，使得每条边流量满足上下界同时除了源点和汇点每一个点流量平衡。如果存在，询问满足标定的最大流量。

我们找到网络上的任意一个可行流。如果找不到解就可以直接结束。

否则我们考虑删去所有附加边之后的残量网络并且在网络上进行调整。

我们在残量网络上再跑一次 S 到 T 的最大流，将可行流流量和最大流流量相加即为答案。

“一个非常易错的问题” S 到 T 的最大流直接在跑完有源汇上下界可行的残量网络上跑。千万不可以在原来的流量网络上跑。

有源汇上下界最小流

给定有源汇流量网络 G。询问是否存在一种标定每条边流量的方式，使得每条边流量满足上下界同时除了源点和汇点每一个点流量平衡。如果存在，询问满足标定的最小流量。

类似的，我们考虑将残量网络中不需要的流退掉。

我们找到网络上的任意一个可行流。如果找不到解就可以直接结束。

否则我们考虑删去所有附加边之后的残量网络。

我们在残量网络上再跑一次 T 到 S 的最大流，将可行流流量减去最大流流量即为答案。

AHOI 2014 支线剧情 对于每条 x 到 y 花费 v 的剧情边设上界为 ∞, 下界为 1。对于每个点，向 T 连边权 c, 上界 ∞, 下界为 1。S 点为 1 号节点。跑一次上下界带源汇最小费用可行流即可。因为最小费用可行流解法与最小可行流类似，这里不再展开。

String

GSAM

约定

字符串个数为 k 个，即 $S_1, S_2, S_3 \dots S_k$

约定字典树和广义后缀自动机的根节点为 0 号节点

应用

所有字符中不同子串个数

可以根据后缀自动机的性质得到，以点 i 为结束节点的子串个数等于 $len[i] - len[link[i]]$

所以可以遍历所有的节点求和得到

例题：【模板】广义后缀自动机（广义 SAM）

```cpp
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 2000000;  // 双倍字符串长度
const int CHAR_NUM = 30;   // 字符集个数，注意修改下方的 (-'a')

struct exSAM {
  int len[MAXN];             // 节点长度
  int link[MAXN];            // 后缀链接，link
  int next[MAXN][CHAR_NUM];  // 转移
```

```
  int tot;                    // 节点总数: [0, tot)

  void init() {  // 初始化函数
    tot = 1;
    link[0] = -1;
  }

  int insertSAM(int last, int c) {  // last 为父 c 为子
    int cur = next[last][c];
    if (len[cur]) return cur;
    len[cur] = len[last] + 1;
    int p = link[last];
    while (p != -1) {
      if (!next[p][c])
        next[p][c] = cur;
      else
        break;
      p = link[p];
    }
    if (p == -1) {
      link[cur] = 0;
      return cur;
    }
    int q = next[p][c];
    if (len[p] + 1 == len[q]) {
      link[cur] = q;
      return cur;
    }
    int clone = tot++;
    for (int i = 0; i < CHAR_NUM; ++i)
      next[clone][i] = len[next[q][i]] != 0 ? next[q][i] : 0;
    len[clone] = len[p] + 1;
    while (p != -1 && next[p][c] == q) {
      next[p][c] = clone;
      p = link[p];
    }
    link[clone] = link[q];
    link[cur] = clone;
    link[q] = clone;
    return cur;
  }

  int insertTrie(int cur, int c) {
    if (next[cur][c]) return next[cur][c];  // 已有该节点 直接返回
    return next[cur][c] = tot++;            // 无该节点 建立节点
  }

  void insert(const string &s) {
    int root = 0;
    for (auto ch : s) root = insertTrie(root, ch - 'a');
  }

  void insert(const char *s, int n) {
    int root = 0;
    for (int i = 0; i < n; ++i)
      root =
          insertTrie(root, s[i] - 'a');  // 一边插入一边更改所插入新节点的父节点
  }

  void build() {
    queue<pair<int, int>> q;
    for (int i = 0; i < 26; ++i)
      if (next[0][i]) q.push({i, 0});
    while (!q.empty()) {  // 广搜遍历
      auto item = q.front();
      q.pop();
      auto last = insertSAM(item.second, item.first);
      for (int i = 0; i < 26; ++i)
        if (next[last][i]) q.push({i, last});
    }
  }
```

```
} exSam;

char s[1000100];

int main() {
  int n;
  cin >> n;
  exSam.init();
  for (int i = 0; i < n; ++i) {
    cin >> s;
    int len = strlen(s);
    exSam.insert(s, len);
  }
  exSam.build();
  long long ans = 0;
  for (int i = 1; i < exSam.tot; ++i) {
    ans += exSam.len[i] - exSam.len[exSam.link[i]];
  }
  cout << ans << endl;
}
```

多个字符串间的最长公共子串

我们需要对每个节点建立一个长度为 k 的数组 `flag`（对于本题而言，可以仅为标记数组，若需要求出此子串的个数，则需要改成计数数组）

在字典树插入字符串时，对所有节点进行计数，保存在当前字符串所在的数组

然后按照 `len` 递减的顺序遍历，通过后缀链接将当前节点的 `flag` 与其他节点的合并

遍历所有的节点，找到一个 `len` 最大且满足对于所有的 k，其 `flag` 的值均为非 0 的节点，此节点的 len 即为解

例题：SPOJ Longest Common Substring II

```
#include <bits/stdc++.h>
using namespace std;

const int MAXN = 2000000;  // 双倍字符串长度
const int CHAR_NUM = 30;   // 字符集个数，注意修改下方的 (-'a')
const int NUM = 15;        // 字符串个数

struct exSAM {
  int len[MAXN];              // 节点长度
  int link[MAXN];             // 后缀链接，link
  int next[MAXN][CHAR_NUM];  // 转移
  int tot;                    // 节点总数：[0, tot)
  int lenSorted[MAXN];   // 按照 len 排序后的数组，仅排序 [1, tot)
                         // 部分，最终下标范围 [0, tot - 1)
  int sizeC[MAXN][NUM];  // 表示某个字符串的子串个数
  int curString;         // 字符串实际个数
  /**
   * 计数排序使用的辅助空间数组
   */
  int lc[MAXN];  // 统计个数

  void init() {
    tot = 1;
    link[0] = -1;
  }

  int insertSAM(int last, int c) {
    int cur = next[last][c];
    len[cur] = len[last] + 1;
    int p = link[last];
    while (p != -1) {
      if (!next[p][c])
        next[p][c] = cur;
      else
        break;
      p = link[p];
    }
```

```
    if (p == -1) {
      link[cur] = 0;
      return cur;
    }
    int q = next[p][c];
    if (len[p] + 1 == len[q]) {
      link[cur] = q;
      return cur;
    }
    int clone = tot++;
    for (int i = 0; i < CHAR_NUM; ++i)
      next[clone][i] = len[next[q][i]] != 0 ? next[q][i] : 0;
    len[clone] = len[p] + 1;
    while (p != -1 && next[p][c] == q) {
      next[p][c] = clone;
      p = link[p];
    }
    link[clone] = link[q];
    link[cur] = clone;
    link[q] = clone;
    return cur;
  }

  int insertTrie(int cur, int c) {
    if (!next[cur][c]) next[cur][c] = tot++;
    sizeC[next[cur][c]][curString]++;
    return next[cur][c];
  }

  void insert(const string &s) {
    int root = 0;
    for (auto ch : s) root = insertTrie(root, ch - 'a');
    curString++;
  }

  void insert(const char *s, int n) {
    int root = 0;
    for (int i = 0; i < n; ++i) root = insertTrie(root, s[i] - 'a');
    curString++;
  }

  void build() {
    queue<pair<int, int>> q;
    for (int i = 0; i < 26; ++i)
      if (next[0][i]) q.push({i, 0});
    while (!q.empty()) {  // 广搜遍历
      auto item = q.front();
      q.pop();
      auto last = insertSAM(item.second, item.first);
      for (int i = 0; i < 26; ++i)
        if (next[last][i]) q.push({i, last});
    }
  }

  void sortLen() {
    for (int i = 1; i < tot; ++i) lc[i] = 0;
    for (int i = 1; i < tot; ++i) lc[len[i]]++;
    for (int i = 2; i < tot; ++i) lc[i] += lc[i - 1];
    for (int i = 1; i < tot; ++i) lenSorted[--lc[len[i]]] = i;
  }

  void getSizeLen() {
    for (int i = tot - 2; i >= 0; --i)
      for (int j = 0; j < curString; ++j)
        sizeC[link[lenSorted[i]]][j] += sizeC[lenSorted[i]][j];
  }
} exSam;

int main() {
  exSam.init();  // 初始化
  string s;
```

```
  while (cin >> s) exSam.insert(s);
  exSam.build();
  exSam.sortLen();
  exSam.getSizeLen();
  int ans = 0;
  for (int i = 0; i < exSam.tot; ++i) {
    bool flag = true;
    for (int j = 0; j < exSam.curString; ++j) {
      if (!exSam.sizeC[i][j]) {
        flag = false;
        break;
      }
    }
    if (flag) ans = max(ans, exSam.len[i]);
  }
  cout << ans << endl;
}
```

Geometry

Planar graph to dual

```
#include "bits/stdc++.h"
#include <algorithm>
#include <ostream>
#include <random>
using namespace std;

using ll = long long;
using LL = long long;
using pii = pair<int, int>;

using ull = unsigned long long;
using int128 = __int128;

ostream& operator<<(ostream& os, int128 p) {
    if (p == 0) return os << 0;
    string s;
    bool flag = 0;
    if (p < 0) { flag = 1; p *= -1; }
    while (p) { s += '0' + p % 10; p /= 10; }
    if (flag) s += '-';
    reverse(s.begin(), s.end());
    return os << s;
}

template<class T> int sgn(T x) { return (x > 0) - (x < 0); }
template<class T>
struct Point {
    typedef Point P;
    T x, y;
    Point(T x = 0, T y = 0) : x(x), y(y) {}
    bool operator<(P p) { return tie(x, y) < tie(p.x, p.y); }
    bool operator==(P p) { return tie(x, y) == tie(p.x, p.y); }

    P operator+(P p) { return P(x+p.x, y+p.y); }
    P operator-(P p) { return P(x-p.x, y-p.y); }
    P operator*(T d) { return P(x*d, y*d); }
    P operator/(T d) { return P(x/d, y/d); }

    T dot(P p) { return x*p.x + y*p.y; }
    T cross(P p) { return x*p.y - y*p.x; }
    T cross(P a, P b) { return (a-*this).cross(b-*this); }
    T dist2() { return x*x + y*y; }
    double dist() { return sqrt(double(dist2())); }
    double angle() { return atan2(ll(y), ll(x)); }

    P unit() { return *this / dist(); }
    P perp() { return P(-y, x); }
    P normal() { return perp().unit(); }
```

```
49
50      int phase() {
51          if (y != 0) return y > 0 ? 0 : 1;
52          return x > 0 ? 0 : 1;
53      }
54
55      friend ostream& operator<<(ostream& os, P p) {
56          return os << "(" << p.x << "," << p.y << ")";
57      }
58  };
59
60  using P = Point<int128>;
61  double eps = 1e-9;
62
63  bool onSegment(P s, P e, P p) {
64      return p.cross(s, e) == 0 && (s - p).dot(e - p) <= 0;
65  }
66
67  struct Edge {
68      int id;
69      int fr, to;
70
71      bool operator==(const Edge &o) {
72          return id == o.id;
73      }
74  };
75
76  void solve(int TC) {
77      int n, m, e;
78      cin >> n >> m >> e;
79      vector<P> baseP(n), sourceP(m);
80
81      for (int i = 0; i < n; i++) {
82          ll x, y; cin >> x >> y;
83          x *= 2, y *= 2;
84          baseP[i] = {x, y};
85      }
86      for (int i = 0; i < m; i++) {
87          ll x, y; cin >> x >> y;
88          x *= 2, y *= 2;
89          sourceP[i] = {x, y};
90      }
91
92      vector<vector<Edge>> g(n);
93      vector<Edge> edge(2 * e);
94      for (int i = 0; i < e; i++) {
95          int u, v; cin >> u >> v;
96          --u, --v;
97          edge[2 * i] = {2 * i, u, v};
98          g[u].push_back(edge[2 * i]);
99          edge[2 * i + 1] = {2 * i + 1, v, u};
100         g[v].push_back(edge[2 * i + 1]);
101     }
102
103     vector<int> pos(2 * e);
104     for (int i = 0; i < n; i++) {
105         sort(g[i].begin(), g[i].end(), [&](Edge& a, Edge& b) {
106             P u = baseP[a.to] - baseP[a.fr], v = baseP[b.to] - baseP[b.fr];
107             if (u.phase() != v.phase()) return u.phase() < v.phase();
108             return u.cross(v) > 0;
109         });
110         for (int j = 0; j < g[i].size(); j++) {
111             pos[g[i][j].id] = j;
112         }
113     }
114
115     int face = 0;
116     vector<vector<int>> facePoly;
117     vector<int128> faceArea;
118     vector<int> belong(2 * e, -1);
119     for (int i = 0; i < 2 * e; i++) {
```

```
        if (belong[i] != -1) continue;
        vector<int> poly;
        int j = i;
        int128 area = 0;
        while (belong[j] == -1) {
            poly.push_back(j);
            belong[j] = face;
            area += baseP[edge[j].fr].cross(baseP[edge[j].to]);

            int nxt = edge[j].to;
            j = g[nxt][(pos[j ^ 1] - 1 + g[nxt].size()) % g[nxt].size()].id;
        }
        facePoly.push_back(poly);
        faceArea.push_back(area);

        face++;
    }
    vector<P> intervalP(e);
    for (int i = 0; i < e; i++) {
        auto [id, u, v] = edge[2 * i];
        intervalP[i] = (baseP[u] + baseP[v]) / 2;
    }

    auto inPolygon = [&](vector<int> &p, P a, bool strict = true) -> bool {
        int cnt = 0, n = int(size(p));
        for (int i = 0; i < n; i++) {
            P fr = baseP[edge[p[i]].fr];
            P to = baseP[edge[p[i]].to];
            if (onSegment(fr, to, a)) return !strict;
            cnt ^= ((a.y < fr.y) - (a.y < to.y)) * a.cross(fr, to) > 0;
        }
        return cnt;
    };

    mt19937_64 rng(42);
    vector<ull> sourcePHash(m), intervalPHash(2 * e);
    for (int i = 0; i < face; i++) {
        if (faceArea[i] <= 0) continue;

        // cout << "face: " << i << endl;
        ull hash = rng();
        for (int j = 0; j < m; j++) {
            if (inPolygon(facePoly[i], sourceP[j])) {
                // cout << "source point: " << j << " poly: " << i << endl;
                sourcePHash[j] ^= hash;
            }
        }
        for (int j = 0; j < e; j++) {
            bool online = false;
            if (belong[2 * j] == i) {
                online = true;
                // cout << "interval mp: " << 2 * j << " poly: " << i << endl;
                intervalPHash[2 * j] ^= hash;
            }
            if (belong[2 * j + 1] == i) {
                online = true;
                // cout << "interval mp: " << 2 * j + 1 << " poly: " << i << endl;
                intervalPHash[2 * j + 1] ^= hash;
            }
            if (!online && inPolygon(facePoly[i], intervalP[j])) {
                // cout << "interval mp: " << 2 * j << " " << 2 * j + 1 << " poly: " << i << endl;
                intervalPHash[2 * j] ^= hash;
                intervalPHash[2 * j + 1] ^= hash;
            }
        }
    }

    // for (int i = 0; i < e; i++) {
    //  cout << intervalPHash[2 * i] << " " << intervalPHash[2 * i + 1] << endl;
    // }

```

```
    unordered_set<ull> S;
    S.reserve(m * 2);
    for (int i = 0; i < m; i++) S.insert(sourcePHash[i]);

    for (int i = 0; i < e; i++) {
        if (S.count(intervalPHash[2 * i]) || S.count(intervalPHash[2 * i + 1])) {
            cout << "1";
        }
        else {
            cout << "0";
        }
    }
    cout << "\n";
}

int32_t main() {
    cin.tie(0)->sync_with_stdio(0);
    cout << fixed << setprecision(10);

    int t = 1;
    // cin >> t;

    for (int i = 1; i <= t; i++) {
        solve(i);
    }
}
```

Flow

Eval expression

```
bool delim(char c) { return c == ' '; }

bool is_op(char c) { return c == '+' || c == '-' || c == '*' || c == '/'; }

bool is_unary(char c) { return c == '+' || c == '-'; }

int priority(char op) {
  if (op < 0)  // unary operator
    return 3;
  if (op == '+' || op == '-') return 1;
  if (op == '*' || op == '/') return 2;
  return -1;
}

void process_op(stack<int>& st, char op) {
  if (op < 0) {
    int l = st.top();
    st.pop();
    switch (-op) {
      case '+':
        st.push(l);
        break;
      case '-':
        st.push(-l);
        break;
    }
  } else {  // 取出栈顶元素，注意顺序
    int r = st.top();
    st.pop();
    int l = st.top();
    st.pop();
    switch (op) {
      case '+':
        st.push(l + r);
        break;
      case '-':
        st.push(l - r);
        break;
      case '*':
```

```
        st.push(l * r);
        break;
      case '/':
        st.push(l / r);
        break;
    }
  }
}

int evaluate(string& s) {
  stack<int> st;
  stack<char> op;
  bool may_be_unary = true;
  for (int i = 0; i < (int)s.size(); i++) {
    if (delim(s[i])) continue;

    if (s[i] == '(') {
      op.push('(');  // 2. 如果遇到左括号，那么将其放在运算符栈上
      may_be_unary = true;
    } else if (s[i] == ')') {  // 3. 如果遇到右括号，执行一对括号内的所有运算符
      while (op.top() != '(') {
        process_op(st, op.top());
        op.pop();  // 不断输出栈顶元素，直至遇到左括号
      }
      op.pop();  // 左括号出栈
      may_be_unary = false;
    } else if (is_op(s[i])) {  // 4. 如果遇到其他运算符
      char cur_op = s[i];
      if (may_be_unary && is_unary(cur_op)) cur_op = -cur_op;
      while (!op.empty() &&
             ((cur_op >= 0 && priority(op.top()) >= priority(cur_op)) ||
              (cur_op < 0 && priority(op.top()) > priority(cur_op)))) {
        process_op(st, op.top());
        op.pop();  // 不断输出所有运算优先级大于等于当前运算符的运算符
      }
      op.push(cur_op);  // 新的运算符入运算符栈
      may_be_unary = true;
    } else {  // 1. 如果遇到数字，直接输出该数字
      int number = 0;
      while (i < (int)s.size() && isalnum(s[i]))
        number = number * 10 + s[i++] - '0';
      --i;
      st.push(number);
      may_be_unary = false;
    }
  }

  while (!op.empty()) {
    process_op(st, op.top());
    op.pop();
  }
  return st.top();
}
```