1 A. Магическая битва

1.1 Условие

Давным-давно на земле жили легендарные Лицеситы - народ, который обладал невероятными способностями в области математики. Они изучали числа и магию, связанную с ними, и придумали уникальную игру.

В игре на каждом ходу вместо числа X записывается число, равное сумме квадратов его цифр. Игрок выигрывает, если получится превратить исходное число в 1 за несколько ходов. Эта игра стала известна как "Игра счастливого числа".

Вы - один из Лицеситов и получаете на вход число N, которое нужно проверить на возможность превращения в счастливое число. Вам нужно написать программу, которая определяет, является ли число N счастливым и может ли оно быть превращено в 1 за несколько ходов.

Если это возможно, то программа должна вывести $LUCKY$.
Если нет, то программа должна вывести $COMMON$.

1.2 Входные данные

В первое строке вводится одно число N, $1 \leq N \leq 10^9$.

1.3 Вывод

В единственной строке выведите $LUCKY$ или $COMMON$.

1.4 Группы

1) $1 \leq N \leq 10^2$ – 40 баллов.
2) $1 \leq N \leq 10^9$ – 60 баллов.

1.5 Пример входных данных

Sample Input:

44

Sample Output:

LUCKY

Sample Input:

59

Sample Output:

COMMON

1.6 Решение

Промоделируем ходы лицеситов. Будем запоминать, какие числа уже были записаны. Либо мы дойдем до единицы, либо пройдем один круг по циклу и на этом остановимся.

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

#define iom std::cin.tie(0); std::cout.tie(0); std::ios_base::sync_with_stdio(false);
#define out std::cout.flush();
#define endl "\n"

using ll = long long;

ll get_num_sum(ll x) {
  ll ans = 0;
  while (x) {
    ans += (x % 10) * (x % 10);
    x /= 10;
  }
  return ans;
}

bool solver(ll n) {
  std::set<ll> s;

  while (n != 1 && s.find(n) == s.end()) {
    s.insert(n);
    n = get_num_sum(n);
  }
  s.insert(n);

  if (s.find(1) != s.end()) {
    return true;
  } else {
    return false;
  }
}

void solve(std::istream &cin, std::ostream &cout) {
  ll n; cin >> n;

  if (solver(n)) {
    cout << "LUCKY\n";
  } else {
    cout << "COMMON\n";
  }
}

int32_t main() {
  iom;

  int T = 1; // cin >> T;
  while (T--) {
    solve(std::cin, std::cout);
  }

  out;
  return 0;
}
```

//

2 B. Новая электростанция

2.1 Условие

Поликарп работает в компании "РосАтом и ему нужно решить задачу по размещению электростанции на карте размером N на M, где каждая клетка имеет определенное значение прибыли или убытка для компании. Цель Поликарпа - выбрать прямоугольную область на карте, которая принесет максимальную прибыль компании.

2.2 Входные данные

Первая строка содержит два числа: N и M - размеры карты. Следующие N строк содержат M чисел x_{ij} ($1 \le i \le N$, $1 \le j \le M$, $-10^5 \le x_{ij} \le 10^5$) - значения прибыли или убытка для компании в каждой клетке.

2.3 Вывод

Вывести одно число - максимальную возможную прибыль компании.

2.4 Группы

1) $1 \le N, M \le 5 - 20$ баллов
2) $1 \le N, M \le 50 - 30$ баллов
3) $1 \le N, M \le 200 - 50$ баллов

2.5 Пример входных данных

Sample Input:

2 2
-2 1
2 4

Sample Output:

6

Sample Input:

3 3
1 0 4
2 -1 1
-1 5 2

Sample Output:

13

2.6 Решение

Чтобы выполнить задания группы 1, надо написать полный перебор всех вариантов.

Чтобы выполнить задания группы 2, воспользуемся идеей префиксных сумм. Простой вариант: делаем префиксные суммы в каждой строчке. Теперь для подсчета суммы в прямоугольнике перебираем строку и за $O(1)$, вычитая соответствующие префиксные суммы, находим сумму отрезка строки без перебора.

Сложный вариант (задания группы 3): префиксные суммы на прямоугольнике. Давайте скажем, что в двумерном массиве $pref[n+1][m+1]$ значение ячейки (i, j) равняется сумме всех чисел в прямоугольнике с вершинами $(0,0)(0,i)(j,0),(i,j)$. Тогда, зная вершины прямоугольника $(x1, y1)$ и $(x2, y2)$, найдем сумму в прямоугольнике, которая равняется $pref[x1][y1] + prev[x2][y2] - pref[x1][y2] - pref[x2][y1]$. Перебирая чертырьмя циклами края прямоугольника, можно получить либо 50 баллов, либо 100. Сложность такого решения $O(n^2 * m^2)$.

Полное решение подразумевает сложный вариант получения 50 баллов. Заметим что зная числа $x2, y2, x1$, нам интересен такой $y1$, чтобы значение $prev[x1][y1] - prev[x1][y1]$ было максимально. Давайте перебирать числа следующим образом

```
for (x1)
    for (x2)
        best_y = 0
        for (y2 = 1...m)
```

где $best_y$ - максимальное значение выражения $prev[x1][y] - prev[x1][y]$ среди y в интервале $[0, y2 - 1]$, которое будет обновляться на каждой итерации цикла. Сложность такого решения $O(n^2 * m)$.

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

using namespace std;

using ll = long long;

int main() {
    cin.tie(nullptr);
    cout.tie(nullptr);
    ios_base::sync_with_stdio(false);
    int n, m;
    cin >> n >> m;
    vector<vector<ll>> v(n, vector<ll>(m));
    for (auto& line : v) {
        for (auto& pos : line) {
```

```
            cin >> pos;
        }
    }
    vector<vector<ll>> pref(n + 1, vector<ll>(m + 1));
    for (int x = 1; x <= n; ++x) {
        for (int y = 1; y <= m; ++y) {
            pref[x][y] = v[x - 1][y - 1] + pref[x][y - 1] + pref[x - 1][y] - pref[x - 1][y - 1];
        }
    }
    ll ans = v[0][0];
    for (int x1 = 0; x1 < n; ++x1) {
        for (int x2 = x1 + 1; x2 <= n; ++x2) {
            int best_y = 0;
            for (int y2 = 1; y2 <= m; ++y2) {
                ll cur = pref[x2][y2] - pref[x1][y2] - pref[x2][best_y] + pref[x1][best_y];
                ans = max(ans, cur);
                if (pref[x1][best_y] - pref[x2][best_y] < pref[x1][y2] - pref[x2][y2]) {
                    best_y = y2;
                }
            }
        }
    }
    cout << ans << '\n';
    return 0;
}
```

//

3 C. Кража на производстве

3.1 Условие

Вы − разработчик системы безопасности для крупного предприятия, на котором случилась кража. Чтобы помочь следователям в раскрытии преступления, вам необходимо разработать программу, которая позволит отследить перемещение людей по территории предприятия и количество людей в определенном кластере в заданный момент времени. Предприятие состоит из нескольких кластеров, каждый из которых окружен забором и имеет только один вход/выход.

По списку событий из логов пропускной системы кластеров ваша программа должна уметь отвечать на 2 типа запросов.

1. Где находился человек с идентификатором ID в момент времени TIME?

2. Сколько человек находилось в помещении CLUSTER в момент времени TIME?

3.2 Входные данные

В первой строке даны 2 числа: N и M − количество записей о событиях на производстве и количество запросов к вашей программе соответственно.

В N следующих строках следуют записи в формате: "[TIME]:ID_TYPE_CLUSTER"

В M следующих строках даны запросы к вашей программе в одном из следующих форматов.
1) Первый запрос имеет формат: "1 ID TIME".

2) Второй запрос имеет формат: "2 CLUSTER TIME".

TIME − целое число, время события ($1 \leq TIME \leq 10^9$).
ID − целое число, уникальный идентификатор сотрудника ($1 \leq ID \leq 10^9$).
TYPE − строка 'IN', если человек вошёл в кластер, и 'OUT', если вышел.
CLUSTER − целое число, номер кластера ($1 \leq ID \leq 10^9$).

Если существуют событие и запрос с одинаковым временем TIME, − считать, что событие происходит раньше запроса.

Гарантируется, что человек не может совершить более одного действия в момент времени TIME.

Гарантируется, что все записи о входе и выходе являются корректными и непротиворечивыми, а также, что в момент первой записи никого не бы-

ло на предприятии.

3.3 Вывод

Ответом на первый тип запроса будет целое число $x \geq 1$ - номер кластера, где находится человек с идентификатором ID в момент времени TIME и 0, если он не находится ни на одном из кластеров.

Ответом на второй тип запроса будет целое число $y \geq 0$ - количество людей на кластере CLUSTER в момент времени TIME.

В единственной строке через пробел выведите M чисел — ответы на запросы в том порядке, в котором их задавали вашей программе.

3.4 Группы

1) $(1 \leq N, M \leq 100)$— 15 баллов.
2) $(1 \leq N, M \leq 2 * 10^5)$ и поступают запросы только первого типа — 20 баллов
3) $(1 \leq N, M \leq 2 * 10^5)$— 65 баллов

3.5 Пример входных данных

Sample Input:

2 2
[1]:1_IN_100
[2]:2_IN_100
2 100 1
1 1 4

Sample Output:

1 100

Sample Input:

6 4
[1]:7_IN_1
[2]:5_IN_2
[101]:3_OUT_1
[3]:3_IN_1
[100]:5_OUT_2

[103]:7_OUT_1
1 3 1
2 1 75
1 5 50
2 2 200

Sample Output:

0 2 2 0

3.6 Решение

Давайте скажем, что запрос к базе или лог - событие, привязанное ко времени.

Считаем все N записей и M запросов, расположим их в хронологическом порядке при помощи сортировки. Далее промоделируем происходящие события. Обновляем состояние базы или же отвечаем на запрос по ее текущему состоянию. Если подумать, для ответа на запрос надо знать для каждого кластера актуальное число человек внутри него, а также для каждого человека кластер, где он находится. Запишем эту информацию в словари. И по мере обработки событий будем обновлять их содержимое.

Сложность решения O((n + m) * log(n + m)).
Сортировка событий за O((n + m) * log(n + m)), и обработка событий за O(m * log(n)).

```
#include <iostream>
#include <vector>
#include <iomanip>
#include <cmath>
#include <chrono>
#include <set>
#include <algorithm>
#include <map>

using namespace std;

struct event{
    int time = 0;
    bool is_check = false;
    int check_type = -1;
    int id = -1, type = -1, cluster = -1;
    int num = -1;
    event() {}
    event(int time, int id, int type, int cluster):
        time(time), id(id), type(type), cluster(cluster) {}
    event(int time, bool is_check, int id, int num):
        time(time), is_check(true), id(id), check_type(1), num(num) {}
    event(int time, int cluster, int num):
        time(time), is_check(true), cluster(cluster), check_type(2), num(num) {}
};

bool cmp(event a, event b) {
    if (a.time == b.time) {
        return !a.is_check;
    }
    return a.time < b.time;
}
```

```
map<int, int> cluster_by_id;
map<int, int> cnt_by_cluster;

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    int n, m;
    cin >> n >> m;
    vector<event> E;
    for (int i = 0; i < n; ++i) {
        char c;
        int time, id, cluster;
        int type;
        string tmp;
        cin >> c >> time >> c >> c >> tmp;
        for (int j = 0; j < tmp.size(); ++j) {
            if (tmp[j] == '_') {
                id = stoi(tmp.substr(0, j));
                tmp = tmp.substr(j + 1);
                break;
            }
        }
        for (int j = 0; j < tmp.size(); ++j) {
            if (tmp[j] == '_') {
                string tp = tmp.substr(0, j);
                if (tp == "IN") {
                    type = 0;
                } else {
                    type = 1;
                }
                tmp = tmp.substr(j + 1);
                break;
            }
        }
        cluster = stoi(tmp);
        E.emplace_back(time, id, type, cluster);
    }
    for (int i = 0; i < m; ++i) {
        int op;
        cin >> op;
        if (op == 1) {
            int id, time;
            cin >> id >> time;
            E.emplace_back(time, true, id, i);
        } else {
            int cluster, time;
            cin >> cluster >> time;
            E.emplace_back(time, cluster, i);
        }
    }
    sort(E.begin(), E.end(), cmp);
    vector<int> ans(m, 0);
    for (const auto& e : E) {
        if (!e.is_check) {
            if (e.type == 0) {
                ++cnt_by_cluster[e.cluster];
                cluster_by_id[e.id] = e.cluster;
            } else {
                --cnt_by_cluster[e.cluster];
                cluster_by_id[e.id] = 0;
            }
        } else {
            if (e.check_type == 1) {
                ans[e.num] = cluster_by_id[e.id];
            } else {
                ans[e.num] = cnt_by_cluster[e.cluster];
            }
        }
    }
    for (auto i : ans)
        cout << i << " ";
    cout << endl;
    return 0;
}
```

4 D. Горшок живой

4.1 Условие

В небольшом городке проходит ежегодный фестиваль музыкальных инструментов, на котором собираются музыканты со всей страны. Но в этом году произошла необычная ситуация - во время транспортировки груза с N струнами, из-за неправильного крепления груза, все струны смешались вместе.

Организаторы фестиваля в отчаянии ищут способ быстро разделить струны на определенные наборы, состоящие из K струн, каждая толщиной h_i.

На помощь им приходит команда разработчиков, которые предлагают использовать алгоритм, который определит, сколько есть отрезков с подходящими наборами струн.

При этом важно помнить, что на отрезке важно множество струн, а не их порядок.

4.2 Входные данные

В первой строке через пробел вводятся два числа: N, K ($1 \leq K \leq N \leq 10^5$).
Во второй строке через пробел вводится K чисел: $(h_1, h_2, ..., h_K)$, где h_i – толщина i-ой струны из набора.
В третьей строке через пробел вводятся N чисел: $(v_1, v_2, ..., v_N)$, где v_i – толщина i-ой струны после транспортировки.

4.3 Вывод

В единственной строке выведите одно число – количество подходящих отрезков, содержащих необходимый набор.

4.4 Группы

1) $1 \leq N \leq 100, 1 \leq K \leq 6 - 12$ баллов
2) $1 \leq N \leq 100000, 1 \leq K \leq 10 - 17$ баллов
3) $1 \leq N \leq 100000, 1 \leq K \leq 1000 - 22$ баллов
3) $1 \leq N, K \leq 100000 - 49$ баллов

4.5 Пример входных данных

Sample Input:

10 2
1 2
1 2 1 2 1 2 1 2 1 2

Sample Output:

9

4.6 Решение

Воспользуемся хешированием. Сопоставим множеству струн $a, b, c, ...$ пару чисел, равных $((p^a + p^b + p^c + ...)$ % M1, $(q^a + q^b + q^c + ...)$ % M2).
Если присмотреться внимательно, пара чисел не зависит от порядка струн в множестве, однако зависит от самого набора струн. Если равны наборы струн, то равны и числа. Взяв $p = 31$, $q = 41$, $M1 = 10^9 + 7$, $M2 = 10^9 + 9$, с очень высокой вероятностью верно и обратное: раз равны числа, то равны и наборы. Расчет этой вероятности оставим читателю в качестве упражнения при изучении хеширования.
При расчете понадобится использовать китайскую теорему об остатках.
Подсчитав степени p, q заранее, получим решение за O(N).

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

#define iom                                                                    \
  std::cin.tie(0);                                                             \
  std::cout.tie(0);                                                            \
  std::ios_base::sync_with_stdio(false);
#define out std::cout.flush();
#define endl "\n"
using ll = int64_t;

const ll pp1 = 31;
const ll pp2 = 41;

const ll MOD1 = 1e9 + 7;
const ll MOD2 = 1e9 + 9;

ll template_hash1 = 0, template_hash2 = 0;

ll n, k;
std::vector<ll> p1, p2;
std::vector<ll> h, v;

ll normalize(ll x, const ll &MOD) { return (x + MOD) % MOD; }

ll add(ll x, ll y, const ll &MOD) {
  return (normalize(x, MOD) + normalize(y, MOD)) % MOD;
}

ll prod(ll x, ll y, const ll &MOD) {
  return normalize(x, MOD) * normalize(y, MOD) % MOD;
}

uint64_t NumberHash(uint64_t x) {
  x = (x ^ (x >> 30)) * static_cast<uint64_t>(0xbf58476d1ce4e5b9);
  x = (x ^ (x >> 27)) * UINT64_C(0x94d049bb133111eb);
  x = x ^ (x >> 31);
  return (x + n) % n;
}

void precalc() {
  p1.resize(n);
  p2.resize(n);
  p1[0] = 1;
  p2[0] = 1;
  for (size_t i = 1; i < n; ++i) {
    p1[i] = prod(p1[i - 1], pp1, MOD1);
    p2[i] = prod(p2[i - 1], pp2, MOD2);
  }
}

ll GetPower(size_t idx, ll MOD, std::vector<ll> p, std::vector<ll> &seq) {
  return prod(seq[idx], p[NumberHash(seq[idx])], MOD);
}

void FindTemplateHash(std::istream &cin) {
  h.resize(k);
```

```
  template_hash1 = template_hash2 = 0;
  for (size_t i = 0; i < k; ++i) {
    cin >> h[i];
    template_hash1 = add(template_hash1, GetPower(i, MOD1, p1, h), MOD1);
    template_hash2 = add(template_hash2, GetPower(i, MOD2, p2, h), MOD2);
  }
}

bool IsEq(ll &val1, ll &val2) { return val1 == val2; }

bool CheckHashes(ll &val1, ll &val2) {
  return IsEq(template_hash1, val1) && IsEq(template_hash2, val2);
}

ll solver() {
  ll ans = 0;

  std::vector<ll> hash1(n), hash2(n);

  hash1[0] = GetPower(0, MOD1, p1, v);
  hash2[0] = GetPower(0, MOD2, p2, v);
  for (size_t i = 1; i < k; ++i) {
    hash1[i] = add(hash1[i - 1], GetPower(i, MOD1, p1, v), MOD1);
    hash2[i] = add(hash2[i - 1], GetPower(i, MOD2, p2, v), MOD2);
  }
  if (CheckHashes(hash1[k - 1], hash2[k - 1])) {
    ++ans;
  }

  for (size_t i = k; i < n; ++i) {
    hash1[i] = add(hash1[i - 1], -GetPower(i - k, MOD1, p1, v), MOD1);
    hash2[i] = add(hash2[i - 1], -GetPower(i - k, MOD2, p2, v), MOD2);

    hash1[i] = add(hash1[i], GetPower(i, MOD1, p1, v), MOD1);
    hash2[i] = add(hash2[i], GetPower(i, MOD2, p2, v), MOD2);

    if (CheckHashes(hash1[i], hash2[i])) {
      ++ans;
    }
  }

  return ans;
}

void solve(std::istream &cin, std::ostream &cout) {
  cin >> n >> k;

  precalc();

  FindTemplateHash(cin);

  v.resize(n);
  for (size_t i = 0; i < n; ++i) {
    cin >> v[i];
  }

  cout << solver() << endl;
}

int32_t main() {
  iom;

  int T = 1; // std::cin >> T;
  while (T--) {
    solve(std::cin, std::cout);
  }

  out;
  return 0;
}
```

//

5 E. Программисты строят забор

5.1 Условие

Программисты разработали собственные доски для строительства специального кругового забора.
Однако, не каждые две доски можно соединить друг с другом. Для каждой пары досок задано, могут ли они быть использованы вместе при строительстве забора. Данная информация представлена в матрице совместимости, где a_{ij} равно 1, если доска i может стоять перед доской j, и 0 в противном случае.
Прежде, чем забор построить, программисты решили узнать, сколькими различными способами они могут это сделать. Каждую доску можно использовать многократно.
Вам нужно помочь программистам решить задачу: есть $N \leq 100$ различных досок, каждая из которых имеет определенную совместимость с другими досками. Необходимо построить круговой забор длины $M \leq 300$ из этих досок.
Из-за того, что ответ может оказаться слишком большим, выведите его по модулю $1e9 + 7$.

5.2 Входные данные

В первой строк вводится N $(2 \leq N \leq 100)$ − количество различных типов досок.
В следующих N строках вводятся по N чисел. $a_{ij} = 1$, если доски i и j совместимы, $a_{ij} = 0$ − в противном случае.
В $(N+2)$-ой строке вводится число M $(2 \leq M \leq 300)$ − необходимая длина для кругового забора.

5.3 Вывод

В единственной строке выведите ответ на задание − количество различных круговых заборов, которые может возвести команда разработчиков.

5.4 Группы

1) Каждую доску можно соединить с каждой, $(M \leq 50) - 20$ баллов.
2) Цикл в совместимости длины K. $(M \ mod \ K = 0) - 30$ баллов.
3) Без ограничений -50.

5.5 Пример входных данных

Sample Input:

5
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
16

Sample Output:

587889561

Sample Input:

3
1 1 1
1 1 1
1 1 1
7

Sample Output:

2187

5.6 Решение

Если бы забор не был круговым, то это была бы классическая задача на динамическое программирование. Первый параметр - длина забора, второй - последняя доска в построенном заборе. Т.е. dp[len][last] - сколько заборов можно построить из len досок, при условии, что на конце забора доска типа $last$. При пересчете проверяем, что концы забора между собой совместимы. А раз забор круговой - выхода два. Или 3-ий параметр динамики, или запустить динамику N раз, фиксируя первую доску в заборе. И в ответ включить лишь те значения динамики, в которых первая и последняя доски совместимы. В предложенном решении первая доска забора - один из параметров динамики.
Сложность решения $O(n^3 * m)$

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

#define iom                                                    \
  std::cin.tie(0);                                             \
  std::cout.tie(0);                                            \
```

```
  std::ios_base::sync_with_stdio(false);
#define out std::cout.flush();
#define endl "\n"
using ll = long long;

size_t n, m;
std::vector<std::set<int>> compatible;
std::vector<std::vector<std::vector<int>>> dp;

void solve(std::istream &cin, std::ostream &cout) {
  cin >> n;

  compatible.resize(n);
  for (size_t i = 0; i < n; ++i) {
    for (size_t j = 0; j < n; ++j) {
      int t;
      cin >> t;
      if (t) {
        compatible[i].insert(j);
      }
    }
  }

  cin >> m;

  dp.resize(n, std::vector<std::vector<int>>(m, std::vector<int>(n, 0)));

  // base
  for (size_t i = 0; i < n; ++i) {
    dp[i][0][i] = 1;
  }

  // calc dp
  for (size_t j = 1; j < m; ++j) {
    for (size_t i = 0; i < n; ++i) {
      for (size_t ii = 0; ii < n; ++ii) {
        if (compatible[i].find(ii) != compatible[i].end()) {
          for (size_t k = 0; k < n; ++k) {
            dp[ii][j][k] += dp[i][j - 1][k];
          }
        }
      }
    }
  }

  ll ans = 0;
  for (size_t i = 0; i < n; ++i) {
    for (size_t j = 0; j < n; ++j) {
      if (compatible[j].find(i) != compatible[j].end()) {
        ans += dp[j][m - 1][i];
      }
    }
  }

  cout << ans << endl;
}

int32_t main() {
  iom;

  int T = 1; // cin >> T;
  while (T--) {
    solve(std::cin, std::cout);
  }

  out;
  return 0;
}

//////////////////////////////////////////////////////////////////////////
```