1 A. Оптимизация

1.1 Условие

Полиевкт собирается заключить крупные контракты с N атомными станциями на поставку высокотехнологичных компонентов для них.
В ближайшие N дней ему необходимо встретиться с представителем каждой из компаний и обговорить объем поставки. И так сложилось, что из-за особенностей логистики в j-ый день представитель i-ой компании будет согласен на поставку a_{ij} компонентов.
Помогите Полиевкту составить расписание для встреч, чтобы принести своей компании как можно большую прибыль.

1.2 Входные данные

В первой строке вводится одно число N, $1 \leq N \leq 10$.
В следующих N строках заданы N чисел: на i-ой строке находится N чисел $0 \leq a_{ij} \leq 100$, где a_{ij} − количество компонентов, которые i-я компания согласится купить в j-й день.

1.3 Вывод

В единственной строке выведите одно число − максимальное количество компонентов, которые удастся поставить компаниям.

1.4 Пример входных данных

Sample Input:
1
10

Sample Output:
10

Sample Input:
2
5 10
10 5
Sample Output:
20

Sample Input:

2
10 4
12 5

Sample Output:

16

1.5 Решение

Переберем все перестановки чисел 1, 2, .., N. Для каждой перестановки p посчитаем сумму a[i][p[i] - 1], ($i = 0, 1, ..., N - 1$). Максимальное значение такой суммы и будет ответом на задачу.

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

using namespace std;

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

  int n;
  cin >> n;

  vector<vector<int>> a(n, vector<int>(n));

  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
      cin >> a[i][j];
    }
  }

  int _max = -1;

  vector<int> cur(n);
  for (int i = 0; i < n; ++i) {
    cur[i] = i;
  }

  do {
    int sum = 0;
    for (int i = 0; i < n; ++i) {
      sum += a[cur[i]][i];
    }
    _max = max(_max, sum);

  } while (next_permutation(cur.begin(), cur.end()));

  cout << _max << endl;

  return 0;
}
```

//

2 B. Горшок живой

2.1 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.

2.2 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 вводятся K чисел: $(h_1, h_2, ..., h_K)$, где h_i − толщина i-й струны из набора.
В третьей строке через пробел вводятся N чисел: $(v_1, v_2, ..., v_N)$, где v_i − толщина i-й струны после транспортировки.

2.3 Вывод

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

2.4 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

2.5 Пример входных данных

Sample Input:
10 2
1 2
1 2 1 2 1 2 1 2 1 2

Sample Output:
9

2.6 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;
}
```

//

3 C. Программисты строят забор

3.1 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 $1e9 + 7$.

3.2 Входные данные

В первой строке вводится число 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.

3.3 Вывод

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

3.4 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.

3.5 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

3.6 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;
}

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

4 D. Подозрительное число

4.1 Условие

Ученый Василий наблюдает явление, суть которого держится в строжайшем секрете. Однако записи измерений с приборов необходимо обработать. Известно, что это явление наблюдается в особый промежуток времени. Найдем число X, которое встречается чаще всего на этом промежутке. Если количество чисел X будет составлять не менее $2/3$ от общего количества чисел в промежутке, то тогда загадочное явление можно наблюдать.
У Василия есть список временных отрезков. Про каждый он хотел бы узнать, наблюдалось ли явление. Помогите ему! Выведите либо то самое число X, либо (-1), если такого числа нет.

4.2 Входные данные

В первой строке задано единственное число N, $(1 \leq N \leq 10^5)$ - число измерений.
Во второй строке записаны N чисел a_i, $(1 \leq a_i \leq 10^9)$ - результаты измерений.
В третьей строке записано число Q, $(1 \leq Q \leq 10^5)$ - число временных отрезков Василия.
В следующих Q строках записаны по 2 числа l_i, r_i, $(1 \leq l_i \leq r_i \leq N)$ - границы запросов Василия.

4.3 Вывод

Вывести Q строк. Каждая строка - ответ на соответствующий ей запрос.
Если на отрезке $[l_i, r_i]$ есть число X, вывести его. Иначе вывести -1.

4.4 Группы

1) $(N, Q \leq 100)$, $(a_i \leq 100)$ (15 баллов)
2) $(a_i \leq 100)$ (20 баллов)
3) Без дополнительных ограничений (65 баллов)

4.5 Пример входных данных

Sample Input:

10
1 2 3 3 3 2 3 3 1 3
5
2 4
2 5
1 2
2 10
1 10

Sample Output:

3
3
-1
3
-1

Sample Input:

5
1 1 5 2 2
3
1 3
3 5
1 5

Sample Output:

1
2
-1

4.6 Решение

Начнем с первой группы, где числа не более 100. Можно выписать все вхождения для каждого числа. Для каждого числа бинарным поиском* найти количество вхождений в отрезок [l, r].
Найти количество таких чисел в отрезке. Сравнить каждое число с 2/3 длины отрезка, и если такое число есть, вывести его в качестве ответа. Таким способом можно набрать 35 баллов.

Покажем, что в отрезке может быть только одно число, являющееся ответом. Если бы их было хотя бы два, то в отрезке было бы (4/3)*(r-l+1) чисел, но их всего (r-l+1).

Давайте выберем 100 случайных чисел в отрезке. Найдем то, которое встречается чаще всего. Как и в решении на 35 баллов, проверим бинпоиском*, является ли оно ответом.
Давайте предположим, что некоторое число X является ответом. Оценим, с какой вероятностью мы сделаем ошибку.
Договоримся, что не будем соблюдать математическую строгость доказательства. Доказательство ожидаемых фактов займет несколько страниц и будет использовать неизвестные школьникам формулы, и, что главное, - никак не поможет понять основную идею решения.
После такого допущения рассмотрим "худший случай": отрезок, 2/3 чисел в котором это - X, оставшаяся часть - Y. Число X мы выбираем с вероятностью 2/3, число Y - с вероятностью 1/3. Пусть чисел X оказалось больше, чем Y. X+Y=100. Значит X $>$ 50.
Пусть B_N - вероятность выбрать ровно N чисел X из этих 100. Она равняется $C_{100}^N * \frac{2}{3}^{100-N} * \frac{1}{3}^N = C_{100}^N * \frac{2^{100}}{3^{100} * 2^N}$.
Вероятность выпадения более 50-ти чисел X равняется $B_{51} + B_{52} + ... + B_{100} < 50 * B_{50} = C_{100}^{50} * \frac{2^{50}}{3^{100}}$. Если вычислить это значение, то получим 0.0002204084401347895. То есть, с вероятностью 0.02% (или же в одном из 4537 случаев), мы ошибаемся. Учитывая, что это очень грубая оценка сверху, реальная вероятность ошибки еще меньше.
Но если вам очень не повезло, у вас есть еще 99 попыток)
Решение имеет сложность O(N*(100 + log(N))).
Выписываем позиции числа в порядке возрастания. В массиве позиций находим A = количество чисел <= r. Находим B = количество чисел <= (l-1). (A-B) - искомое значение.

```
#include <set>
#include <iostream>
#include <algorithm>
#include <vector>
#include <map>
#include <cassert>
#include <unordered_map>

using namespace std;

using ll = long long;

int ask(int l, int r, vector<int>&v, map<int, vector<int>>&inps) {
    unordered_map<int, int> cnts;
    if (r - l < 200) {
        for (int i = l; i < r; ++i)
            ++cnts[v[i]];
    } else {
        set<int> ban;
        for (int i = 0; i < 100; ++i) {
            int len = (r - l);
            int pos = l + rand() % len;
            if (ban.find(pos) != ban.end())
                continue;
            ban.insert(pos);
            ++cnts[v[pos]];
        }
    }

    int mx_val = v[l];
    int mx_count = cnts[mx_val];
    for (const auto& [val, count] : cnts) {
        if (mx_count < count) {
            mx_val = val;
            mx_count = count;
        }
    }

    int len = (r - l);
```

```
    int count = lower_bound(inps[mx_val].begin(), inps[mx_val].end(), r) -
    lower_bound(inps[mx_val].begin(), inps[mx_val].end(), l);
    if (count * 3 >= len * 2) {
        return mx_val;
    }
    return -1;
}

void solve() {
    int n;
    cin >> n;
    vector<int> v(n);
    for (auto& x : v) {
        cin >> x;
        assert(1 <= x && x <= 1e9);
    }

    map<int, vector<int>> inps;
    for (int i = 0; i < n; ++i) {
        inps[v[i]].push_back(i);
    }

    int q;
    cin >> q;
    while (q--) {
        int l, r;
        cin >> l >> r;
        assert(1 <= l && r <= n);
        --l;
        int res = ask(l, r, v, inps);
        cout << res << '\n';
    }
}

int main() {
    srand(time(nullptr));
    cin.tie(nullptr);
    cout.tie(nullptr);
    ios_base::sync_with_stdio(false);
    int t = 1; // cin >> t;
    while (t--)
        solve();
    return 0;
}
```

//

5 E. Туземные шифры

5.1 Условие

Берляндия и Древляндия с давних времен соперничают друг с другом. Недавно берляндская разведка перехватила очень важное зашифрованное сообщение Древляндии.
Зашифрованный текст представляет собой последовательность длины $2 \times n$ из нулей и единиц.
Также разведка узнала, что для расшифровки необходимо выбрать некую подпоследовательность элементов и циклически сдвинуть вправо. После чего надо выделить две одинаковых подпоследовательности длины n, которые не пересекаются.
Возьмем лист бумаги, на котором в строчку записана наша последовательность. Затем каждый элемент первой строки записываем в конец либо второй строки, либо в конец третьей строки. Если строки 2 и 3 имеют длину n и совпадают друг с другом, то эта строка является исходным сообщением.
Выведите расшифрованное сообщение (их может быть несколько, подходит любое) или же -1, если расшифровать не удалось (разведку могли дезинформировать).

5.2 Входные данные

В первой строке дано число n.
Во второй строке дана строка из символов 0 и 1 длины $2 \times n$.

5.3 Вывод

В единственной строке выведите расшифрованное сообщение или -1, если сообщение расшифровать не удалось.

5.4 Группы

1) ($1 \leq n \leq 10^5$), гарантируется, что ответ есть, сдвиг не нужен − 30 баллов
2) ($1 \leq n \leq 10^5$)− 70 баллов

5.5 Пример входных данных

Sample Input: 3
100001

Sample Output:

100

Sample Input:

2
1110

Sample Output:

-1

5.6 Решение

Необходимое условие для существования решения - количество символов 1 в строке должно быть чётным. Докажем, что оно является и достаточным.

Найдём решение в следующем виде:

Составим пары из символов $(s[2i-1], s[2i])$ для $(1 \leq i \leq n)$.

Предположим, что у нас есть x пар, где элементы различаются и $(n-x)$ пар, где элементы одинаковы.

Пусть среди x пар, в которых элементы одинаковы, имеется y пар, в которых элементы равны 1. Тогда общее количество символов 1 в строке равно $x + 2 \cdot y$. Мы знаем, что общее количество единиц чётное, а значит, x – тоже чётное.

Тогда выберем ровно x индексов, по одному из пары, где элементы различны. Будем брать индекс символа 0, если номер пары чётный и индекс символа 1, если нечётный. Таким образом, наша выбранная последовательность символов имеет вид $0, 1, 0, \ldots, 0, 1$. Заметим, что одним циклическим сдвигом данных элементов мы получим новую строку s, где $s[2i] = s[i]$ для $(1 \leq i \leq n)$.

Выберем все элементы на чётных индексах и выведем их в ответ. Итоговая сложность решения – $O(n)$.

```
import sys

n = int(input())
s = input()
ans = []
cur = 0
for i in range(1, 2 * n, 2):
    if s[i] != s[i - 1]:
        if ord(s[i]) - ord('0') == cur:
            ans.append(i)
        else:
```

```
            ans.append(i - 1)
        cur ^= 1

if len(ans) % 2:
    print(-1)
    sys.exit(0)


if len(ans):
    tmp = [0] * len(ans)
    for i in range(len(ans)):
        tmp[i] = s[ans[i]]
    tmp = list(tmp[-1]) + tmp[0:-1]
    for i in range(len(ans)):
        s = s[:ans[i]] + tmp[i] + s[ans[i] + 1:]

for i in range(0, n):
    print(s[2 * i], end='')
print()
```

//