Московская олимпиада школьников. Информатика. 9 класс. Дистанционный этап, 2024/25

20 дек 2024 г., 10:00 — 20 янв 2025 г., 23:59

№ 1

100 баллов

Шутка с таймером

В семье две сестры, Аня и Катя. Однажды, в 7:30 утра, Аня установила таймер на телефоне Кати, который должен сработать через k минут. Катя не подозревала об этом и отправилась в школу с телефоном. Занятия у Кати начинаются в 9:00. Каждый урок длится n минут, а перерыв между ними составляет m минут. Всего у неё запланировано t уроков. Определите, на каком уроке прозвонит таймер и удастся ли Ане осуществить свою шутку. Учтите, что звонок на урок или с урока начинается в первую секунду минуты, а таймер срабатывает в тридцатую секунду минуты.

Вам необходимо выяснить номер урока, когда прозвонит таймер. Если таймер сработает во время перемены, до начала первого урока или после завершения последнего, выведите -1.

Формат входных данных

Четыре целых числа даны каждый в своей строке в следующем порядке: k, n, m, t — число минут на таймере, длина урока, длина перемены и количество уроков,
$1 \le k \le 1000, 10 \le n \le 100, 10 \le m \le 100, 2 \le t \le 10$.

Формат выходных данных

Выведите единственное число — номер урока в течение которого сработал будильник или -1, если будильник сработал во время перемены, до первого или после последнего урока, т. е. вне уроков.

Критерии оценивания

Каждый тест оценивается независимо.

Примеры

```
95
45
10
5
```

```
1
```

```
40
45
10
5
```

```
-1
```

```
400
45
15
5
```

```
-1
```

Ограничения

Время выполнения: 1 секунда

Память: 256 MB

Код C++

```cpp
#include "iostream"

typedef long double ld;

#define int long long
#define double ld

using namespace std;
signed main()
{
		int k, n, m, t;
	cin >> k >> n >> m >> t;

	int start = 7 * 60;
	int end = start + k;

	int l = 8 * 60 + 30;

	for (int i = 0; i < t; i++)
	{
		int r = l + n - 1;

		if (end >= l && end <= r)
			return cout << i + 1, 0;

		l = r + 1 + m;
	}

	cout << -1;
}


```

№ 2

100 баллов

Подходящие числа

Математикам обычно нравятся только особенные числа, например, простые, у которых нет делителей кроме 1 и самого числа, или степени двойки. Назовем такие числа *подходящими*. Поэтому, если к ужину смарт-часы показывают некоторое количество пройденных за день шагов, и оно не является подходящим числом, то математик сделает минимально возможное дополнительное число шагов, чтобы счетчик шагов показывал подходящее число.

Помогите математикам в определении ближайшего к текущим показаниям часов подходящего числа.

Формат входных данных

В первой строке дано одно целое число n ($1 \le n \le 10^5$) — число дней, в которые снимались показания часов.

Во второй строке даны n целых чисел a_i ($0 \le a_i \le 10^7$) — вечерние показания часов в течение n дней.

Формат выходных данных

Выведите n чисел: для каждого a_i ближайшее подходящее число, не меньшее его.

Критерии оценивания

Подзадача	Баллы	Ограничения
0	0	Тесты из условия
1	25	$n \le 1000, a_i \le 1000$
2	17	$n \le 1000$
3	28	$a_i \le 1000$
4	30	Без дополнительных ограничений

Примеры

```
5
2020 1023 0 101 10000
```

```
2027 1024 1 101 10007
```

Ограничения

Время выполнения: 4 секунды

Память: 256 MB

Код C++

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

#define s second
#define f first
#define pb push_back
#define forn(i, n) for (int i = 0; i < (int)(n); ++i)
#define all(a) (a).begin(), (a).end()
#define len(a) (int)(a).size()
#define INF (int)(1e9)
//#define endl '\n'
// #define int long long

using namespace std;

const int N = 100100;
const int M = 10010010;

bool r[M]; // 0 -> prime
vector<int> good;

void preprocess() {
    r[0] = r[1] = 1;
    vector<int> p;
    forn(i, M) {
        if (!r[i]) {
            p.pb(i);
            for (int j = min(M + 0LL, 1LL * i * i); j < M; j += i)
{
                r[j] = 1;
            }
        }
    }
    int puw = 1;
    vector<int> pw;
    while (puw < M) {
        pw.pb(puw);
        puw *= 2;
    }
    good.resize(len(p) + len(pw));
    merge(all(p), all(pw), good.begin());
}

signed main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    cout.precision(37);

    preprocess();

    int n;
    cin >> n;
    forn(i, n) {
        int x;
        cin >> x;
        cout << good[lower_bound(all(good), x) - good.begin()] << '
';
    }

    return 0;
```

```
58  }
59
```

№ 3

100 баллов

Симметричная впадина

Массив натуральных чисел называется *впадиной*, если он сначала строго убывает, затем строго возрастает. Длина впадины как минимум 3. Например, 4 3 2 6 9 — это впадина, потому что первые три числа в нем строго убывают, а потом с третьего по пятое число — строго возрастают.

Массив натуральных чисел называется *палиндромом*, если он один и тот же при записи слева направо и справа налево. Например, 1 3 2 3 1 — это палиндром.

Подмассив массива — массив, полученный при удалении любого префикса (начала) и любого суффикса (конца) исходного массива. Он состоит из подряд идущих элементов исходного массива.

Для заданного массива определите, какова длина самого длинного подмассива данного массива, являющегося одновременно и впадиной, и палиндромом. Например, 4 3 2 3 4 — это палиндром и впадина одновременно.

Формат входных данных

В первой строке дано одно целое число n — размер массива, $1 \le n \le 10^6$. Во второй строке n целых чисел через пробел a_i — элементы массива, $1 \le a_i \le 10^9$.

Формат выходных данных

Выведите единственное целое число — максимальную длину палиндромной ямы или -1, если такой нет. Напомним, что длина ямы должна быть как минимум 3.

Критерии оценивания

Все тесты разбиты на группы со следующими ограничениями:

$1 \le n \le 50$, 15 баллов

$1 \le n \le 300$, 15 баллов

$1 \le n \le 7000$, 25 баллов

$1 \le n \le 1000000$, 45 баллов

Очередная группа проверяется только после прохождения всех тестов всех предыдущих групп.

Примеры

```
8
11 9 5 2 1 2 5 7
```

```
5
```

Ограничения

Время выполнения: 2 секунды
Память: 256 MB

Код C++

```cpp
#include "iostream"
#include "algorithm"
#include "string"
#include "vector"
#include "cstring"


using namespace std;

#define maxi(a,b) a = max(a, b);
#define mini(a,b) a = min(a, b);

#define all(a) a.begin(), a.end()
#define rall(a) a.rbegin(), a.rend()
#define sqr(x) ((x) * (x))
#define SZ(a) ((int)(a.size()))
#define watch(x) cout << (#x) << " = " << x << endl;
typedef long double ld;

#define int long long
typedef vector<int> vi;

signed main()
{
	ios::sync_with_stdio(0);
	cout.tie(0); cin.tie(0);

	int n;
	cin >> n;

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

	int  ans = 0;
	for (int i = 1; i < n - 1; i++)
	{
		if (a[i] < a[i - 1] && a[i] < a[i + 1])
		{
			int len = 0;
			while (1)
			{
				if (i - len - 1 < 0 || i + len + 1 >= n)
					break;

				if (a[i - len - 1] <= a[i - len])
					break;

				if (a[i + len + 1] <= a[i + len])
					break;

				if (a[i - len - 1] != a[i + len + 1])
					break;

				len++;
			}

			maxi(ans, 1 + 2 * len);
```

```
                }
            }

            ans >= 3 ? cout << ans : cout << -1;
    }


```

№ 4

100 баллов

Дорога в школу

Будем считать, что дорога в школу начинается в момент времени 0 и состоит из n участков. Ученик передвигается по i-му участку пути со скоростью V_i в течение T_i единиц времени. Требуется q запросов определить, чему равна **средняя скорость** на части пути ученика в различные промежутки времени.

Средняя скорость на части пути вычисляется как отношение всего преодолённого расстояния к затраченному на это времени. $V = \frac{S}{T}$.

Формат входных данных

В первой строке записаны два натуральных числа $n, q (1 \le n, q \le 10^5)$ — количество участков пути и количество запросов.

В следующих n строках записаны пары целых чисел $V_i, T_i (1 \le V_i, T_i \le 10^5)$ — информация об участках пути ученика в школу.

В следующих q строках даны запросы. Каждая строка содержит два целых числа $L_q, R_q (0 \le L_q < R_q \le \sum_{i=1}^{N} T_i)$ — начальный и конечный моменты времени интересующего промежутка.

Формат выходных данных

Для каждого запроса выведите среднюю скорость на части пути в соответствующем промежутке времени. Ответ выведите с точностью не менее чем четыре знака после точки (но, например, если ответ целый, лишние нули после точки можно не выводить, и саму точку можно не ставить).

Критерии оценивания

Тесты разделены на несколько подзадач. Баллы выставляются только за подзадачу в целом. Следующая подзадача проверяется только при прохождении всех тестов предыдущих подзадач.

Подзадача 1 (16 баллов): $N \le 100, Q \le 100, \sum_{i=1}^{N} T_i \le 1000$.

Подзадача 2 (14 баллов): на N и Q нет дополнительных ограничений, $\sum_{i=1}^{N} T_i \le 10^5$.

Подзадача 3 (15 баллов): $N \le 100, Q \le 100$, на $\sum_{i=1}^{N} T_i$ нет дополнительных ограничений.

Подзадача 4 (10 баллов): Гарантируется, что для каждого запроса q в момент времени L_q и в момент времени R_q Ученик находился на одном и том же участке пути (участок может быть разным для разных запросов).

Подзадача 5 (15 баллов): Гарантируется, что для каждого запроса q выполняется следующее: либо ученик преодолел участок пути целиком, либо вообще не передвигался по нему.

Подзадача 6 (30 баллов): Ограничения из условия.

Примеры

```
5 7
3 4
1 1
4 2
9 5
2 4
0 16
9 10
5 7
13 15
2 14
0 6
5 10
```

```
4.6250
9.0000
```

```
4.0000
2.0000
5.3333
2.8333
7.0000
```

Ограничения

Время выполнения: 2 секунды

Память: 256 MB

Код C++

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

using namespace std;
using ll = long long int;


#define trace(arg) #arg << ": " << (arg) << "\n"
#define trace_log cerr << trace(__LINE__)

const int max_n = 1e5 + 5;


ll segments_amount;
ll time_pref[max_n];
ll dist_pref[max_n];
ll speed[max_n];


void solve();

int main() {
    cin.tie(NULL);
    cout.tie(NULL);
    cin.sync_with_stdio(false);

    int n;
    int q = 1;
    cin >> n >> q;
    segments_amount = n;

    for (int i = 1; i <= n; i++) {
        ll v, t;
        cin >> v >> t;
        time_pref[i] = time_pref[i - 1] + t;
        dist_pref[i] = dist_pref[i - 1] + v * t;
        speed[i-1] = v;
    }

    while (q--) solve();
    return 0;
}

ll get_coord(ll t) {
    ll l = 0, r = segments_amount;
    ll mid = (l + r) / 2;
    while (r - l > 1) {
        if (t <= time_pref[mid]) r = mid;
        else l = mid;
        mid = (l + r) / 2;
    }
    return dist_pref[l] + speed[l]*(t-time_pref[l]);
}

void solve() {
    ll st, ft;
    cin >> st >> ft;

    double dur = ft - st;
    double dist =  get_coord(ft)- get_coord(st);

    cout << setprecision(4) << fixed << dist / dur << endl;
```

```
63  }
64
```

№ 5

100 баллов

Шахматный путь

Друзья играют в следующую игру: один называет шахматную фигуру, второй — стартовую клетку на стандартной шахматной доске, третий — конечную клетку. Далее они соревнуются — кто раньше определит минимальное количество ходов, за которое эта фигура может добраться из **стартовой** клетки в **конечную**. Друзья договорились, что все фигуры будут белыми, а шахматная доска будет считаться свободной от других фигур.

Ряды на доске обозначаются буквами и цифрами, начиная от левого нижнего углового поля $a1$ — чёрного цвета. Вертикальные ряды — латинскими буквами a–h, горизонтальные — цифрами 1–8.

Правила передвижения фигур (адаптированы под задачу):

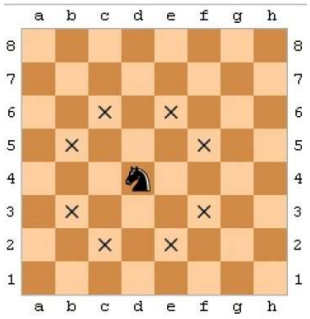
Король (KING) передвигается со своего поля на одно из свободных смежных полей.

Ферзь (QUEEN) ходит по вертикалям, диагоналям и горизонталям, на которых он находится.

Ладья (ROOK) ходит по вертикалям и горизонталям, на которых она находится.

Слон (BISHOP) ходит по диагоналям, на которых он находится.

Конь (KNIGHT) может пойти на одно из полей, ближайших к тому, на котором он стоит, но не на той же самой горизонтали, вертикали или диагонали.



Пешка (PAWN) передвигается на одно поле только вперёд. Если стартовая клетка соответствует стартовой позиции белой пешки в настоящей игре (это горизонталь a2–h2), пешка может пойти как на одну, так и на две клетки вперёд в первом ходу. Любая пешка, достигающая крайней горизонтали (для белых это горизонталь a8–h8), должна быть тем же ходом заменена на ферзя того же цвета, что и пешка. Если в тестовом случае пешка изначально находится на горизонтали a8–h8, то она сразу заменяется на ферзя (эта замена ходом не считается).

Формат входных данных

В первой строке записано число $T (1 \le T \le 24576)$ — количество случаев, на которые нужно получить ответ.

В следующих T строках описаны сами случаи. Каждое описание начинается со строки, задающей фигуру:

KING — король, QUEEN — ферзь, ROOK — ладья, BISHOP — слон, KNIGHT — конь, PAWN — пешка.

Далее, через пробел, записаны координаты стартовой клетки в формате cx, где c — строчная английская буква от 'a' до 'h', обозначающая ряд, а x — натуральное число от 1 до 8, обозначающее столбец. Далее, через пробел, задана конечная клетка в аналогичном формате.

Формат выходных данных

Для каждого тестового случая выведите ответ в отдельной строке — минимальное количество ходов, которое необходимо сделать указанной фигуре, чтобы прийти из стартовой клетки в конечную клетку.

В случае, если конечная клетка недосягаема из стартовой, выведите -1.

Замечание

В примере у первой пешки стартовая позиция совпадает со стартовой позицией в настоящей игре, поэтому она может достичь конечной за один ход. Для второй пешки это не так, поэтому она может ходить вперед только в соседнюю клетку.

Критерии оценивания

Все тесты оцениваются независимо.

Во многих тестах во всех случаях упоминается только одна из фигур.

Примеры

```
9
ROOK c5 h2
KING a1 a2
PAWN b2 b4
PAWN b1 b4
BISHOP e5 e6
PAWN c5 e5
QUEEN e5 g7
KNIGHT a1 c1
KNIGHT a1 a1
```

```
2
1
1
3
-1
5
1
2
0
```

Ограничения

Время выполнения: 1 секунда

Память: 256 MB

Код C++

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

#define vi vector<int>
#define pii pair<int, int>
#define vii vector<pii>
using namespace std;

int solveKing(int x1, int y1, int x2, int y2) {
    return max(abs(x2 - x1), abs(y2 - y1));
}

int solveQueen(int x1, int y1, int x2, int y2) {
    if (x1 == x2 && y1 == y2) return 0;
    if (x1 == x2) return 1;
    if (y1 == y2) return 1;
    if (x1 + y1 == x2 + y2) return 1;
    if (x1 - y1 == x2 - y2) return 1;
    return 2;
}

int solveRook(int x1, int y1, int x2, int y2) {
    if (x1 == x2 && y1 == y2) return 0;
    if (x1 == x2) return 1;
    if (y1 == y2) return 1;
    return 2;
}

int solveKnight(int x1, int y1, int x2, int y2) {
    queue <pii> q;
    x1--;
    y1--;
    x2--;
    y2--;

    const int dx[8] = {1, 1, -1, -1, 2, 2, -2, -2};
    const int dy[8] = {2, -2, 2, -2, 1, -1, 1, -1};

    int dist[8][8];
    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) dist[i][j] = -1;
    }
    dist[x1][y1] = 0;
    q.push({x1, y1});
    while (!q.empty()) {
        int x = q.front().first;
        int y = q.front().second;
        q.pop();
        for (int i = 0; i < 8; i++) {
            int xto = x + dx[i];
            int yto = y + dy[i];
            bool inside = (xto >= 0 && xto < 8 && yto >= 0 && yto < 8);
            if (!inside) continue;
            if (dist[xto][yto] == -1) {
                dist[xto][yto] = dist[x][y] + 1;
                q.push({xto, yto});
            }
        }
    }
```

```
    return dist[x2][y2];
}

int solveBishop(int x1, int y1, int x2, int y2) {
    if ((x1 + y1) % 2 != (x2 + y2) % 2) return -1;
    if (x1 == x2 && y1 == y2) return 0;
    if (y1 - x1 == y2 - x2) return 1;
    if (x1 + y1 == x2 + y2) return 1;
    return 2;
}

int solvePawn(int x1, int y1, int x2, int y2) {
    bool first = true;
    int res = 0;
    if (x1 == x2 && y1 == y2) return 0;
    if (x2 == x1 + 1 && y2 == y1) return 1;

    while (x1 != 8) {
        if (x1 == 2 && first) x1 += 2;
        else x1++;
        res++;
        if (x1 == x2 && y1 == y2) return res;
        first = false;
    }

    return res + solveQueen(x1, y1, x2, y2);
}

void solve() {
    string w1, w2;
    int x1, y1, x2, y2;
    string s;
    cin >> s >> w1 >> w2;
    y1 = w1[0] - 'a' + 1;
    x1 = w1[1] - '0';
    y2 = w2[0] - 'a' + 1;
    x2 = w2[1] - '0';
    if (s == "KING") {
        cout << solveKing(x1, y1, x2, y2);
    }
    else if (s == "BISHOP") {
        cout << solveBishop(x1, y1, x2, y2);
    }
    else if (s == "KNIGHT") {
        cout << solveKnight(x1, y1, x2, y2);
    }
    else if (s == "ROOK") {
        cout << solveRook(x1, y1, x2, y2);
    }
    else if (s == "PAWN") {
        cout << solvePawn(x1, y1, x2, y2);
    }
    else {
        cout << solveQueen(x1, y1, x2, y2);
    }
    cout << "\n";
}

int main()
{
    int T = 1;
    cin >> T;
    while (T--) {
```

```
125             solve();
126         }
127         return 0;
128     }
```

№ 6

100 баллов

Правильные подматрицы

Дана матрица (таблица) из n строк и m столбцов, заполненная строчными буквами латинского алфавита.

Назовем матрицу *правильной*, если в ней встречаются **ровно две различные** буквы, и они расположены в шахматном порядке (одна буква на местах белых клеток, вторая — чёрных).

Следующие матрицы **являются** правильными:

o	x	o
x	o	x
o	x	o

x	o	x
o	x	o
x	o	x

a
b
a

Следующие матрицы **не являются** правильными:

o	x	o
x	x	x
o	x	o

a	a	a
a	a	a
a	a	a

b

Требуется найти количество *правильных* подматриц данной матрицы.

Подматрицей называется любая прямоугольная часть исходной матрицы. Она получается из исходной матрицы сначала отбрасыванием нескольких (возможно, 0) подряд идущих строк в начале и нескольких строк (возможно, 0) в конце, а затем в полученной матрице можно отбросить несколько (возможно, 0) столбцов в начале и несколько (возможно, 0) столбцов в конце.

Формат входных данных

В первой строке даны два целых числа n и m $(2 \le n,\ m \le 300)$ — количество строк и столбцов в матрице соответственно.

В каждой из следующих n строк задана последовательность, состоящая из m строчных букв латинского алфавита.

Формат выходных данных

Выведите единственное число — количество правильных подматриц данной матрицы.

Критерии оценивания

Все тесты оцениваются независимо.

Примеры

```
2 2
aa
aa
```

```
0
```

```
2 2
ab
cd
```

```
4
```

```
2 2
ab
ba
```

5

Ограничения

Время выполнения: 3 секунды

Память: 256 MB

Код C++

```cpp
#include<bits/stdc++.h>
using namespace std;
#define pb  push_back
#define ll  long long
#define vi  vector<int>
#define vvi vector<vi >
#define all(x) x.begin(), x.end()

ll res = 0;
ll n, m;
string s[300], t[300];
ll d[300];

ll co(ll x) {
    return (x * (x - 1)) / 2;
}

int main() {
    ios_base::sync_with_stdio(false); cin.tie(0);
    cin >> n >> m;
    for (ll i = 0; i < n; ++i)
        cin >> s[i];
    for (ll i = 0; i < n; ++i) {
        ll b = 0;
        for (ll j = 1; j < m; ++j) {
            if (s[i][j] == s[i][j - 1]) {
                res += co(j - b);
                b = j;
            }
            else if (j > 1 && s[i][j] == s[i][j - 2])
                ;
            else {
                res += co(j - b);
                b = j - 1;
            }
        }
        res += co(m - b);
    }
    for (ll i = 0; i + 1 < n; ++i) {
        for (ll j = 0; j < m; ++j) {
            d[j] = 2; t[j] = "++";
        }
        for (ll j = 0; j < m; ++j) {
            if (s[i][j] == s[i + 1][j]) continue;
            t[j][0] = s[i][j];
            t[j][1] = s[i + 1][j];
            if (j % 2) swap(t[j][0], t[j][1]);
            for (ll I = i + 2; I < n; ++I) {
                if (s[I][j] != s[I - 2][j]) break;
                ++d[j];
            }
        }
        for (ll h = 2; i + h - 1 < n; ++h) {
            ll b = 0;
            for (ll j = 0; j < m; ++j) {
                if (t[j] == "++" || d[j] < h) {
                    res += co(j - b + 1);
                    b = j + 1;
                }
                else if (j > 0 && t[j] == t[j - 1])
                    ;
                else {
```

```
                    res += co(j - b + 1);
                    b = j;
                }
            }
            res += co(m - b + 1);
        }
    }
    cout << res << endl;

    return 0;
}

```

№ 7

100 баллов

Доминирующий элемент

Дан массив из n натуральных чисел, проиндексированный с 1 до n включительно. Рассмотрим некоторый отрезок массива $[l, r]$ включительно, и назовем элемент x *доминирующим* на этом отрезке, если он встречается на нём строго больше раз, чем все остальные элементы вместе взятые.

Дано q запросов, в каждом запросе даны $[l_q, r_q]$ — границы отрезка. Найдите доминирующий элемент этого отрезка либо сообщите, что на этом отрезке его нет.

Формат входных данных

В первой строке вам даны два числа $n, q (1 \leq n, q \leq 10^5)$ — количество элементов в массиве и количество запросов.

Во второй строке задан массив из n натуральных чисел $a_i (1 \leq a_i \leq 10^9)$.

В следующих q строках заданы запросы, в каждой новой строке по два числа $l_q, r_q (1 \leq l_q \leq r_q \leq n)$ — границы отрезка в очередном запросе.

Формат входных данных

Для каждого запроса в новой строке выведите значение доминирующего элемента на этом отрезке, либо -1, если доминирующего элемента на отрезке нет.

Критерии оценивания

Подзадача 1 (18 баллов): $n \leq 500, q \leq 500$.

Подзадача 2 (14 баллов): $n \leq 3000, q \leq 3000$.

Подзадача 3 (20 баллов): $n \leq 20000, q \leq 20000$.

Подзадача 4 (16 баллов): В массиве присутствует не более 10 различных значений.

Подзадача 5 (32 балла): Ограничения из условия.

Баллы за подзадачу выставляются только при прохождении всех тестов подзадачи. Подзадача проверяется, если все тесты всех предыдущих подзадач пройдены.

Примеры

```
5 4
3 2 3 1 3
1 3
3 5
1 5
1 4
```

```
3
3
3
-1
```

Ограничения

Время выполнения: 4 секунды
Память: 512 MB

Код C++

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

#define vi vector<int>
#define pii pair<int, int>
#define vii vector<pii>
using namespace std;

void compress(vi &a, vi &val) {
    int n = a.size();
    vii p(n);
    for (int i = 0; i < n; i++) p[i] = {a[i], i};
    sort(p.begin(), p.end());
    int curval = 0;
    for (int i = 0; i < n; i++) {
        if (i && p[i].first != p[i - 1].first) curval++;
        val[curval] = p[i].first;
        a[p[i].second] = curval;
    }
}


vi tree;
vector <vi> occ;
vi val, a;

bool check(int x, int l, int r) {
    int L = (int)(lower_bound(occ[x].begin(), occ[x].end(), l) - occ[x].begin());
    int R = (int)(lower_bound(occ[x].begin(), occ[x].end(), r + 1) - occ[x].begin() - 1);
    return max(0, R - L + 1) > (r - l + 1) / 2;
}

void build(int v, int tl, int tr) {
    if (tl == tr) {
        tree[v] = a[tl];
        return;
    }
    int tm = (tl + tr) / 2;
    build(v * 2, tl, tm);
    build(v * 2 + 1, tm + 1, tr);
    if (tree[v * 2] != -1 && check(tree[v * 2], tl, tr)) tree[v] = tree[v * 2];
    else if (tree[v * 2 + 1] != -1 && check(tree[v * 2 + 1], tl, tr)) tree[v] = tree[v * 2 + 1];
    else tree[v] = -1;
}

int get(int v, int tl, int tr, int l, int r) {
    if (l > r) return -1;
    if (l == tl && r == tr) return tree[v];
    int tm = (tl + tr) / 2;
    int x1 = get(v * 2, tl, tm, l, min(r, tm));
    int x2 = get(v * 2 + 1, tm + 1, tr, max(l, tm + 1), r);
    if (x1 != -1 && check(x1, l, r)) return x1;
    if (x2 != -1 && check(x2, l, r)) return x2;
    return -1;
}

void solve() {
```

```
    int n, q;
    cin >> n >> q;
    a.resize(n);
    val.resize(n);
    for (int i = 0; i < n; i++) {
        cin >> a[i];
    }
    compress(a, val);
    tree.resize(4 * n);
    occ.resize(n);
    for (int i = 0; i < n; i++) occ[a[i]].push_back(i);
    build(1, 0, n - 1);
    for (int i = 0; i < q; i++) {
        int l, r;
        cin >> l >> r;
        l--;
        r--;
        int x = get(1, 0, n - 1, l, r);
        if (x != -1) x = val[x];
        cout << x << "\n";
    }
}

int main()
{
    int T = 1;
    //cin >> T;
    while (T--) {
        solve();
    }
    return 0;
}
```

№ 8

100 баллов

Граф. Просто граф

Дан неориентированный связный граф, состоящий из n вершин, пронумерованных от 1 до n, и m ребер, а также q запросов. В каждом из запросов, вам требуется по заданным различным вершинам u и v, найти количество различных простых путей, соединяющих u и v. Напомним, что путь называется простым, если этот путь проходит по каждой из вершин графа не более чем по одному разу.

Формат входных данных

В первой строке записаны целые неотрицательные числа n и m ($1 \le n \le 200\,000$, $0 \le m \le n + 4$) — количество вершин и рёбер графа.

Далее в следующих m строках записано по паре целых чисел u, v, $1 \le u, v \le n$, $u \ne v$.

В следующей строке записано целое число q, $1 \le q \le 2\,000\,000$ количество запросов. В следующих й строках записано по паре целых чисел u, v, $1 \le u, v \le n$, $u \ne v$, описывающих запрос.

Гарантируется, что в графе не существует петель и кратных ребер.

Формат выходных данных

Для каждого запроса выведите в отдельной строке количество различных путей между вершинами u и v.

Критерии оценивания

Каждый тест оценивается независимо. Как минимум в 60% тестов n и m не превосходят 50.

Примеры

```
3 3
1 2
2 3
3 1
3
1 2
3 1
2 3
```

```
2
2
2
```

```
4 4
1 2
2 3
3 1
1 4
6
1 2
1 3
1 4
2 3
2 4
3 4
```

```
2
2
1
2
2
2
```

Ограничения

Время выполнения: 4 секунды

Память: 512 MB

Код C++

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

using namespace std;

const int MOD = 1000000007;
const int MAXLOG = 20;
const int MAX_ADD_EDGES = 6;
const int MAX_BASE_VERTICES = 3 * MAX_ADD_EDGES - 1;

int n, nn, m, nmcd, nm, compNumber;
vector<vector<int> > gr, ngr, achievableBaseVertices;
vector<int> used, h, pr, base, isBased, baseVerticesNumbers, baseEdgeNumbers, abv;
vector<pair<int, int> > edges, baseEdges;

int st[MAXLOG][1<<MAXLOG];
int firstPos[1<<MAXLOG], vs[1<<MAXLOG], hs[1<<MAXLOG], floorlog[1<<MAXLOG];
int stind;

int dp[1 << MAX_BASE_VERTICES][MAX_BASE_VERTICES];
int ans0[MAX_BASE_VERTICES][MAX_BASE_VERTICES];
int ans1[MAX_BASE_VERTICES][MAX_BASE_VERTICES];
int ans2[MAX_BASE_VERTICES][MAX_BASE_VERTICES];

void addEdge(int v1, int v2, vector<vector<int> > &gr) {
    gr[v1].push_back(v2);
    gr[v2].push_back(v1);
}

void readData() {
    scanf("%d%d", &n, &m);
    gr.assign(n, vector<int>());
    for (int i = 0; i < m; ++i) {
        int v1, v2;
        scanf("%d%d", &v1, &v2);
        --v1;
        --v2;
        addEdge(v1, v2, gr);
    }
}

void addToST(int v) {
    vs[stind] = v;
    hs[stind++] = h[v];
}

void dfsLCA(int v, int prv) {
    used[v] = 1;
    firstPos[v] = stind;
    addToST(v);
    for (int nv : gr[v]) {
        if (nv == prv)
            continue;
        if (used[nv] == 0) {
            pr[nv] = v;
            h[nv] = h[v] + 1;
            dfsLCA(nv, v);
            addToST(v);
        }
        else if (used[nv] == 1)
            edges.push_back(make_pair(v, nv));
```

```
    }
    used[v] = 2;
}

void buildLCAAndFindNonTreeEdges() {
    stind = 0;
    used.assign(n, 0);
    h.assign(n, 0);
    pr.assign(n, -1);
    edges.clear();

    dfsLCA(0, -1);

    floorlog[0] = -1;
    for (int i = 0; i < stind; ++i) {
        st[0][i] = i;
        floorlog[i + 1] = 1 + floorlog[(i + 1) / 2];
    }
    for (int l = 0; l + 1 < MAXLOG; ++l)
        for (int i = 0; i < stind; ++i) {
            st[l+1][i] = (((i + (1 << l) >= stind) || (hs[st[l][i]]) <= hs[st[l][i + (1 << l)]]]) ? st[l][i] : st[l][i + (1 << l)]);
        }
}

vector<int> compressVector(vector<int> a) {
    sort(a.begin(), a.end());
    a.resize(unique(a.begin(), a.end()) - a.begin());
    return a;
}

int findLCA(int v1, int v2) {
    v1 = firstPos[v1];
    v2 = firstPos[v2];
    if (v1 > v2)
        swap(v1, v2);
    int l = floorlog[v2 - v1 + 1];
    int ans1 = st[l][v1];
    int ans2 = st[l][v2 - (1 << l) + 1];
    return ((hs[ans1] < hs[ans2]) ? vs[ans1] : vs[ans2]);
}

void compressAndMarkBase() {
    base = compressVector(base);
    for (int v : base)
        isBased[v] = 1;
}

void findBaseVertices() {
    base.clear();
    isBased.assign(n, 0);
    for (pair<int, int> p : edges) {
        base.push_back(p.first);
        base.push_back(p.second);
    }
    compressAndMarkBase();

    int kol = base.size();
    for (int i = 0; i < kol; ++i)
        for (int j = i + 1; j < kol; ++j)
            base.push_back(findLCA(base[i], base[j]));

```

```
    compressAndMarkBase();

    baseVerticesNumbers.assign(n, -1);
    for (int i = 0; i < (int)base.size(); ++i)
        baseVerticesNumbers[base[i]] = i;
}

int getBaseNumber(int v) {
    return lower_bound(base.begin(), base.end(), v) - base.begin();
}

void findBaseEdges() {
    nn = base.size();
    ngr.assign(nn, vector<int>());
    baseEdges.clear();
    for (int v : base) {
        int nv = v;
        do {
            nv = pr[nv];
        }
        while (nv >= 0 && !isBased[nv]);

        if (nv >= 0) {
            int bv = getBaseNumber(v);
            int bnv = getBaseNumber(nv);
            if (bv > bnv)
                swap(bv, bnv);
            baseEdges.push_back(make_pair(bv, bnv));
        }
    }

    assert(base.size() == 0 || baseEdges.size() == base.size() - 1);
    sort(baseEdges.begin(), baseEdges.end());
    for (pair<int, int> p : edges)
        baseEdges.push_back(make_pair(getBaseNumber(p.first), getBaseNumber(p.second)));
}

void dfsABV(int v) {
 //   cerr << "dfsABV " << v << ' ' << compNumber << "\n";
    if (isBased[v]) {
        abv.push_back(v);
        return;
    }

    used[v] = compNumber;
    for (int nv : gr[v])
        if (used[nv] < 0)
            dfsABV(nv);


}

void findAchievableBaseVertices() {
    used.assign(n, -1);
    compNumber = 0;
    abv.clear();
    for (int i = 0; i < n; ++i)
        if (!isBased[i] && used[i] < 0) {
            dfsABV(i);
            ++compNumber;
            achievableBaseVertices.push_back(abv);
```

```
183
184             assert(1 <= abv.size() && abv.size() <= 2);
185             if (abv.size() == 1)
186                 baseEdgeNumbers.push_back(-1);
187             else {
188                 int v1 = getBaseNumber(abv[0]);
189                 int v2 = getBaseNumber(abv[1]);
190                 if (v1 > v2)
191                     swap(v1, v2);
192                 int numEdge = lower_bound(baseEdges.begin(),
    baseEdges.begin() + base.size() - 1, make_pair(v1, v2)) -
    baseEdges.begin();
193                 baseEdgeNumbers.push_back(numEdge);
194             }
195
196             abv.clear();
197         }
198
199     for (int v : base) {
200         achievableBaseVertices.push_back(vector<int>(1, v));
201         used[v] = compNumber++;
202         baseEdgeNumbers.push_back(-1);
203     }
204 }
205
206 void calcDP(int st) {
207     memset(dp, 0, sizeof(dp));
208     dp[1 << st][st] = 1;
209     int nn = base.size();
210     for (int mask = 0; mask < (1 << nn); ++mask)
211         for (int v = 0; v < nn; ++v)
212             if (dp[mask][v])
213                 for (int nv : ngr[v])
214                     if (!(mask & (1 << nv)))
215                         dp[mask | (1 << nv)][nv] += dp[mask][v];
216 }
217
218 void buildNgr(int f1, int f2) {
219     ngr.assign(nn, vector<int>());
220     for (int i = 0; i < (int)baseEdges.size(); ++i)
221         if (i != f1 && i != f2)
222             addEdge(baseEdges[i].first, baseEdges[i].second, ngr);
223 }
224
225 void findAns0() {
226     buildNgr(-1, -1);
227     memset(ans0, 0, sizeof(ans0));
228     for (int v = 0; v < nn; ++v) {
229         calcDP(v);
230         for (int mask = 0; mask < (1 << nn); ++mask)
231             for (int w = 0; w < nn; ++w)
232                 ans0[v][w] += dp[mask][w];
233     }
234 }
235
236 void findAns1() {
237     memset(ans1, 0, sizeof(ans1));
238     for (int e = 0; e < nn - 1; ++e) {
239         buildNgr(e, -1);
240         calcDP(baseEdges[e].first);
241         for (int mask = 0; mask < (1 << nn); ++mask)
242             for (int w = 0; w < nn; ++w)
243                 ans1[e][w] += dp[mask][w];
```

```
244             calcDP(baseEdges[e].second);
245             for (int mask = 0; mask < (1 << nn); ++mask)
246                 for (int w = 0; w < nn; ++w)
247                     ans1[e][w] += dp[mask][w];
248         }
249     }
250
251     void findAns2() {
252         memset(ans2, 0, sizeof(ans2));
253         for (int e1 = 0; e1 < nn - 1; ++e1)
254             for (int e2 = e1 + 1; e2 < nn - 1; ++e2) {
255                 buildNgr(e1, e2);
256                 calcDP(baseEdges[e1].first);
257                 for (int mask = 0; mask < (1 << nn); ++mask)
258                     ans2[e1][e2] += dp[mask][baseEdges[e2].first] +
    dp[mask][baseEdges[e2].second];
259                 calcDP(baseEdges[e1].second);
260                 for (int mask = 0; mask < (1 << nn); ++mask)
261                     ans2[e1][e2] += dp[mask][baseEdges[e2].first] +
    dp[mask][baseEdges[e2].second];
262                 ans2[e2][e1] = ans2[e1][e2];
263             }
264     }
265
266     void findAllPossibleAnswers() {
267         findAns0();
268         findAns1();
269         findAns2();
270     }
271
272     void performQuery() {
273         int v1, v2;
274         scanf("%d%d", &v1, &v2);
275         --v1;
276         --v2;
277
278         if (n == m + 1) {
279             cout << "1\n";
280             return;
281         }
282
283         if (achievableBaseVertices[used[v1]].size() >
    achievableBaseVertices[used[v2]].size())
284             swap(v1, v2);
285
286         vector<int>& abv1 = achievableBaseVertices[used[v1]];
287         vector<int>& abv2 = achievableBaseVertices[used[v2]];
288         vector<int>& bvn = baseVerticesNumbers;
289
290         /*for (int v : abv1)
291             cerr << v << ' ';
292         cerr << "are abv1\n";
293         for (int v : abv2)
294             cerr << v << ' ';
295         cerr << "are abv2\n";
296         */
297
298         if (abv1.size() == 1) {
299             if (abv2.size() == 1)
300                 cout << ans0[bvn[abv1[0]]][bvn[abv2[0]]] << "\n";
301             else
302                 cout << ans1[baseEdgeNumbers[used[v2]]][bvn[abv1[0]]]
    << "\n";
```

```
303         }
304         else {
305             int e1 = baseEdgeNumbers[used[v1]];
306             int e2 = baseEdgeNumbers[used[v2]];
307             if (e1 != e2)
308                 cout << ans2[e1][e2] << "\n";
309             else {
310                 int b1 = abv1[0];
311                 int b2 = abv1[1];
312 
313                 if (h[b1] > h[b2])
314                     swap(b1, b2);
315                 int lca1 = findLCA(b2, v1);
316                 int lca2 = findLCA(b2, v2);
317                 if (lca1 == lca2)
318                     cout << "1\n";
319                 else
320                     cout << ans1[baseEdgeNumbers[used[v1]]][bvn[b2]] -
    1 + 1 << "\n";
321             }
322         }
323     }
324 
325     void performQueries() {
326         int q;
327         scanf("%d", &q);
328         while (q--)
329             performQuery();
330     }
331 
332     int main() {
333         readData();
334         if (m + 1 > n) {
335             buildLCAAndFindNonTreeEdges();
336             findBaseVertices();
337             findBaseEdges();
338 
339             /*for (int v : base)
340                 cerr << "BASE VERTEX: " << v << "\n";
341             for (pair<int, int> e : baseEdges)
342                 cerr << "BASE EDGE: " << e.first << ' ' << e.second <<
    "\n";
343             */
344 
345             findAchievableBaseVertices();
346             findAllPossibleAnswers();
347 
348         }
349         performQueries();
350         return 0;
351     }
352 
353     /*
354     void dfsRebuildGraph(int v, int prv) {
355         used[v] = 1;
356         pr[v] = prv;
357         for (int nv : gr[v]) {
358             if (nv == prv)
359                 continue;
360             if (used[nv] == 0) {
361                 addEdge(v, nv, ngr);
362                 h[nv] = h[v] + 1;
363                 dfsRebuildGraph(nv, v);
```

```
364          }
365          else if (used[nv] == 1) {
366              int newv = ngr.size();
367              ngr.push_back(vector<int>());
368              h.push_back(h[v] + 1);
369              addEdge(nv, newv, ngr);
370              addEdge(newv, nv, ngr);
371              pr.push_back(v);
372          }
373      }
374      used[nv] = 2;
375  }
376
377  void rebuildGraph() {
378      used.assign(n, 0);
379      pr.assign(n, -1);
380      ngr.assign(n, vector<int>());
381      h.assign(n, 0);
382      dfsRebuildGraph(0, -1);
383      basen = n;
384      n = ngr.size();
385      gr.swap(ngr);
386  }
387
388  */
389
```