Московский Авиационный Институт
(национальный исследовательский университет)

Институт информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Журнал по исследовательской практике (индивидуальный план)

Студенты	Группа
Артемьев Дмитрий Иванович	М8О-406Б-18
Белоусов Егор Владимирович	М8О-207Б-20
Инютин Максим Андреевич	М8О-307Б-19
Команда:	MAI #2 Artemiev, Belousov, Inyutin

Москва, 2021

Сводная таблица за осень 2021

Дата	Название	Время	Место проведения	Решенные задачи	Дорешанные задачи
12.09.2021	Grand Prix of Dolgoprudny	11:00-16:00	Дистанционно	G, H	D, I
19.09.2021	Grand Prix of IMO	11:00-16:00	Дистанционно	A, E, H, N	F
10.10.2021	XXII Открытая Всесибирская олимпиада	10:00-15:00	Дистанционно	1, 3	-
17.10.2021	ICPC training MAI 21-22	11:00-16:00	Дистанционно	A, D, F	G, H
24.10.2021	Grand Prix of Korea	11:00-16:00	Дистанционно	H, G	C
07.11.2021	Grand Prix of Siberia	11:00-16:00	Дистанционно	1, 3, 5, 8	-
14.11.2021	Grand Prix of EDG	11:00-16:00	Дистанционно	A, D, E, G, I	B, F, K
21.11.2021	RuCode 4.0 Div A-B Champoinship	11:00-16:00	Дистанционно	B, K	D
22.11.2021	Div A + B Contest 1	09:00-14:00	Дистанционно	C, D, H, L	F, J, K
26.11.2021	Div A Contest 4: The Korean Contest	09:00-14:00	Дистанционно	B, C, L	D, G
12.12.2021	Grand Prix of Nanjing	11:00-16:00	Дистанционно	A, C, H, M	-
19.12.2021	Moscow Regional Contest	11:00-16:00	Дистанционно	A, D, E, F, G, H, N	-

Явка на контесты

Дата	Название	Присутствующие
12.09.2021	Grand Prix of Dolgoprudny	Артемьев, Белоусов, Инютин
19.09.2021	Grand Prix of IMO	Артемьев, Белоусов, Инютин
10.10.2021	XXII Открытая Всесибирская олимпиада	Артемьев, Белоусов, Инютин
17.10.2021	ICPC training MAI 21-22	Артемьев, Белоусов, Инютин
24.10.2021	Grand Prix of Korea	Артемьев, Белоусов, Инютин
07.11.2021	Grand Prix of Siberia	Артемьев, Белоусов, Инютин
14.11.2021	Grand Prix of EDG	Артемьев, Белоусов, Инютин
21.11.2021	RuCode 4.0 Div A-B Champoinship	Артемьев, Белоусов, Инютин
22.11.2021	Div A + B Contest 1	Артемьев, Белоусов, Инютин
26.11.2021	Div A Contest 4: The Korean Contest	Артемьев, Белоусов, Инютин
12.12.2021	Grand Prix of Nanjing	Артемьев, Белоусов, Инютин
19.12.2021	Moscow Regional Contest	Артемьев, Белоусов, Инютин

Problem G. Nikanor Loves Games

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	512 mebibytes

Nikanor spends all his free time on games. Because of this, he gets bad marks at the university, but that's another story. He also likes gambling. In this problem, we consider the modification of the game called "Orlyanka". There are two players, and each of them has his own coin. Each of the two sides of a coin contains an integer. Players toss their coins, and the winner is the one with the highest number. We can assume that for each coin the probabilities of coming up both sides are equal.

Tonight Nikanor is playing this game with his friends. Nikanor has n friends, and he will play with each of them for a bet of x_i rubles. Fortunately, Nikanor knows that his i-th friend has a coin with the numbers a_i and b_i. If Nikanor wins against his friend, he will receive x_i rubles. Otherwise, he will pay x_i rubles to his friend. If Nikanor and his friend dropped the same value, Nikanor is declared the winner.

Now Nikanor is going to go to the store and buy one coin for all games to maximize his expected profit, taking the coin cost into account. In this shop, a coin with the numbers a and b costs $a \cdot b$ rubles. Nikanor can buy any coin with positive integers.

It's so hard for Nikanor to make the right decision... Nikanor asks you to help him choose a coin so that the expected profit is as high as possible.

Input

The first line contains one integer n ($1 \le n \le 2 \cdot 10^5$) denoting the number of friends.

Each of the following n lines contains three integers a_i, b_i, and x_i ($1 \le a_i, b_i, x_i \le 10^9$) representing the numbers on i-th friend's coin and i-th bet in rubles.

Output

Print a single integer — the maximum expected profit.

Your answer will be considered correct if its absolute or relative error does not exceed 10^{-6}.

Formally, let your answer be a, and the jury's answer be b. Your answer will be accepted if and only if $\frac{|a-b|}{\max(1,|b|)} \le 10^{-6}$.

Examples

standard input	standard output
2 1 4 15 3 5 10	2.5000000000
1 2 2 8	4.0000000000

Идея

Очевидно, что нужно всегда брать грани монет, которые присутствуют у оппонентов, то есть при наличии только одной монеты со значениями 1 и 2 не имеет смысла брать номинал 3 на одну из граней нашей монеты, так как можно взять 2 и потратить меньше денег. Пока мы не взяли ни одной грани полагаем, что наше математическое ожидание равняется $bad = -\sum_i x_i$, пусть тогда мы взяли на одну из сторон монеты a', а на другую b', тогда наше математическое ожидание равняется $E(a', b') = \sum_{i:a_i \leqslant a'}(\frac{x_i}{2}) + \sum_{i:b_i \leqslant a'}(\frac{x_i}{2}) + \sum_{i:a_i \leqslant b'}(\frac{x_i}{2}) + \sum_{i:b_i \leqslant b'}(\frac{x_i}{2}) + bad - a' \cdot b'$. Закинем все значения a_i и b_i в массив z с коэффициентами $\frac{1}{2}$, отсортируем, посчитаем $pref_i = \sum_{j=0}^{i-1} z_i$. Упростим нашу сумму для подсчета математического ожидания: $E(a', b') = pref_{i'} + pref_{j'} + bad - a \cdot b$, где $i' : z_{i'} = a'$ и $j' : z_{j'} = b'$. В таком случае мы можем посчитать ответ за $O(n^2 + n \cdot \log n) \approx O(n^2)$ перебрав индексы в z. Улучшим решение до $O(n \cdot \log n)$. Пусть мы уже выбрали a', тогда мы хотим выбрать b' такое, что $E(a', b') \to max$. Т.е. $E(a') = max_{b' \leqslant a'}(a' \cdot b' + pref_{j'}) + bad + pref_{i'}$. Заметим, что $a' \cdot b' + pref_{j'}$ при фиксированной a' является прямой с $k = b'$ и $b = pref_{j'}$. В таком случае можно воспользоваться *Convex hull trick*. Мы будем хранить множество прямых слева направо, так как угол наклона растет, то новую прямую мы будем добавлять справа, а также a' тоже растет, поэтому мы сможем быстро узнавать максимум. Этот прием работает амортизованно за $O(n)$. Итоговая сложность: $O(n + n \cdot \log n) \approx O(n \cdot \log n)$.

Исходный код

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

using namespace std;

#define flush cout.flush

using ll = long long;
using ull = unsigned long long;
using ld = long double;
using pl = pair<ll, ll>;
const ll INF = 1e9 + 7;
const ll mod = 1e9 + 7;
const ll mod2 = 998244353;
const ld eps = 1e-9;
const ld PI = acos(-1);

struct line {
    ld k;
    ld b;
};
```

```
ld inter(line f, line s) {
    return (s.b - f.b) / (f.k - s.k);
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    ll n;
    cin >> n;
    vector<ll> a(n), b(n), x(n);
    for (ll i = 0; i < n; ++i)cin >> a[i] >> b[i] >> x[i];
    vector<pl> z;
    for (ll i = 0; i < n; ++i) {
        z.push_back({a[i], i});
        z.push_back({b[i], i});
    }
    z.push_back({1, -1});
    sort(z.begin(), z.end());
    ll m = z.size();
    vector<ld> pref(m, 0.0);
    ld bad = 0.0;
    for (ll i = 1; i < m; ++i) {
        pref[i] = pref[i - 1] + (ld) x[z[i].second] * 0.5;
        bad += (ld) x[z[i].second] * 0.5;
    }
    ld res = -1e18;
    deque<line> lines;
    for (ll i = 0; i < m; ++i) {
        ld k = -z[i].first;
        ld b = pref[i] - bad;
        line now = {k, b};
        bool need = true;
        while (!lines.empty()) {
            if (lines.front().k == k) {
                if (b > lines.front().b) {
                    lines.pop_front();
                } else {
                    need = false;
                    break;
                }
            } else {
                if (lines.size() == 1)break;
                line f = lines.front();
                lines.pop_front();
                line s = lines.front();
                lines.push_front(f);
                ld xx = inter(f, s);
                ld it = inter(f, now);
```

```
                if (it > xx - eps) {
                    lines.pop_front();
                } else break;
            }
        }
        if (need) {
            lines.push_front(now);
        }
        while (lines.size() > 1) {
            line f = lines.front();
            lines.pop_front();
            line s = lines.front();
            lines.push_front(f);
            ld xx = inter(f, s);
            if (xx < z[i].first + eps) {
                lines.pop_front();
            } else break;
        }
        ld it = lines.front().k * z[i].first + lines.front().b + pref[i];
        res = max(res, it);
// cout << "found: " << it << " | ";
// for (ll j = 0; j <= i; ++j) {
// ld E = pref[i] + pref[j] - bad;
// E -= (ld) (z[i].first * z[j].first);
// cout << E << " ";
// }
// cout << "\n";
    }
    cout << fixed << setprecision(20);
    cout << res << "\n";
    return 0;
}
```

Положение команды

№	Участник Я	A 0/24	B 2/11	C 3/11	D 59/605	E 13/74	F 1/14	G 104/442	H 128/330	I 73/140	J 36/111	K 2/98	Очки	Штраф
104	MAI #2 :	—	—	—	—	—	—	+10 04:22	+12 02:30	-3 02:41	—	—	2	852

Problem H. Hamiltonian

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

You are given a positive integer $K \le 60$. Construct a graph with at most 20 vertices with the following property: there are exactly K unordered pairs of vertices (u, v) such that there is a Hamiltonian path between u and v in this graph.

It can be shown that, under these constraints, the solution always exists.

Recall that a Hamiltonian path is a path between two vertices of a graph that visits each vertex exactly once.

Input

The only line of the input contains a single integer K ($1 \le K \le 60$).

Output

On the first line, output two integers n and m ($2 \le n \le 20$, $0 \le m \le \frac{n(n-1)}{2}$), the number of vertices and the number of edges in your graph respectively.

In each of the next m lines, output two integers u and v ($1 \le u, v \le n$, $u \ne v$), representing the edge (u, v) of your graph. All edges have to be distinct.

Examples

standard input	standard output
1	2 1 1 2
2	4 4 1 2 1 3 2 3 3 4
3	3 3 1 2 2 3 3 1

Идея

Как известно, задача поиска Гамильтонова пути в графе является NP-полной. С помощью динамического программирования по подмножествам вычислительную сложность можно уменьшить с $O(n!)$ до $O(n^2 \cdot 2^n)$. Реализуем алгоритм, проверяющий какой-то граф на наличие Гамильтоновых путей между K парами вершин u и v.

Для маленьких K нетрудно найти ответ. Для большего K будем случайно генерировать графы и искать в них количество пар вершин u и v таких, что между ними есть Гамильтонов путь. Сгенерировав достаточно графов, можно увидеть закономерность, тогда решение становится полностью конструктивным, его асимптотика $O(K)$.

Исходный код

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

using graph_t = std::vector< std::vector<int> >;
using edge_t = std::pair<int, int>;
using graph_edges_t = std::vector<edge_t>;

bool find(int u, int v, const graph_t & g, std::vector<bool> & dp, const int mask) {
    if (dp[mask]) {
        return true;
    }
    std::cout << "u = " << u + 1 << ", v = " << v + 1 << std::endl;
    if (!(mask & (1 << v))) {
        return false;
    }
    for (size_t i = 0; i < g[v].size(); ++i) {
        int go_to = g[v][i];
        dp[mask] = dp[mask] or find(u, go_to, g, dp, mask ^ (1 << v));
    }
    return dp[mask];
}

bool find_path(int u, int v, const graph_t & g) {
    const int n = g.size();
    std::vector<bool> dp(1 << n);
    dp[1 << u] = true;
    return find(u, v, g, dp, (1 << n) - 1);
}

edge_t rand_edge(const int n) {
    int u = std::rand() % n;
    int v = std::rand() % n;
    while (v == u) {
        v = std::rand() % n;
```

```
    }
    return std::make_pair(u, v);
}

edge_t rev_edge(const edge_t & uv) {
    return std::make_pair(uv.second, uv.first);
}

graph_edges_t gen_graph(const int n, const int m) {
    std::set<edge_t> exist_edges;
    graph_edges_t res;
    for (int i = 0; i < m; ++i) {
        edge_t rnd_edge = rand_edge(n);
        while (exist_edges.count(rnd_edge) > 0 or exist_edges.count(rev_edge(rnd_edge)) > 0) {
            rnd_edge = rand_edge(n);
        }
        exist_edges.insert(rnd_edge);
        exist_edges.insert(rev_edge(rnd_edge));
        res.push_back(rnd_edge);
    }
    return res;
}

graph_t read_graph(const graph_edges_t & ge, const int n) {
    graph_t g(n);
    for (size_t i = 0; i < ge.size(); ++i) {
        int u = ge[i].first;
        int v = ge[i].second;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    return g;
}

int calc(int n, int m, graph_t & g) {
    const int nax = (1 << n);
    int res = 0;
    for (int st = 0; st < n; ++st) {
        std::vector< std::vector<char> > dp(nax, std::vector<char>(n, 0));
        dp[1 << st][st] = 1;
        for (int mask = (1 << st); mask < nax; ++mask) {
            for (int v = 0; v < n; ++v) {
                if (((1 << v) & mask)) {
                    for (auto to : g[v]) {
                        if ((1 << to) & mask)continue;
                        dp[mask | (1 << to)][to] |= dp[mask][v];
                    }
                }
```

```
            }
        }
        for (int i = 0; i < n; ++i) {
            if (i == st)continue;
            if (dp[nax - 1][i])res++;
        }
    }
    return res / 2;
}

int main() {
    std::srand(std::time(NULL));
    // int n, m;
    // std::cin >> n >> m;
    // graph_t g(n);
    // for (int i = 0; i < m; ++i) {
    // int u, v;
    // std::cin >> u >> v;
    // --u;
    // --v;
    // g[u].push_back(v);
    // g[v].push_back(u);
    // }
    // for (int i = 0; i < n; ++i) {
    // std::cout << "i = " << i + 1 << " : ";
    // for (size_t j = 0; j < g[i].size(); ++j) {
    // std::cout << g[i][j] + 1 << " ";
    // }
    // std::cout << std::endl;
    // }
    // int ans = 0;
    // for (int i = 0; i < n; ++i) {
    // for (int j = i + 1; j < n; ++j) {
    // std::cout << "find path " << i + 1 << " " << j + 1 << std::endl;
    // if (find_path(i, j, g)) {
    // std::cout << "found path " << i + 1 << " " << j + 1 << std::endl;
    // ++ans;
    // }
    // }
    // }
    // std::cout << ans << std::endl;

    int k;
    std::cin >> k;
    if (k == 1) {
        std::cout << "2 1" << std::endl;
        std::cout << "2 1" << std::endl;
        return 0;
    }
```

```cpp
    if (k == 2) {
        std::cout << "4 4" << std::endl;
        std::cout << "1 2" << std::endl;
        std::cout << "1 3" << std::endl;
        std::cout << "2 3" << std::endl;
        std::cout << "3 4" << std::endl;
        return 0;
    }
    // int n, m;

    if (k <= 20) {
        std::cout << k << " " << k << std::endl;
        for (int i = 0; i < k - 1; ++i) {
            std::cout << i + 1 << " " << i + 2 << std::endl;
        }
        std::cout << k << " 1" << std::endl;
        return 0;
    }
    if (k - 11 <= 20) {
        int d = k - 11;
        std::cout << d << " " << d + 3 << std::endl;
        for (int i = 0; i < d - 1; ++i) {
            std::cout << i + 1 << " " << i + 2 << std::endl;
        }
        std::cout << d << " 1" << std::endl;

        std::cout << "1 3" << std::endl;
        std::cout << "3 5" << std::endl;
        std::cout << "2 4" << std::endl;
        return 0;
    }
    if (k - 17 <= 20) {
        int d = k - 17;
        std::cout << d << " " << d + 4 << std::endl;
        for (int i = 0; i < d - 1; ++i) {
            std::cout << i + 1 << " " << i + 2 << std::endl;
        }
        std::cout << d << " 1" << std::endl;

        std::cout << "1 3" << std::endl;
        std::cout << "3 5" << std::endl;
        std::cout << "2 4" << std::endl;
        std::cout << "4 6" << std::endl;
        return 0;
    }

    if (k - 24 <= 20) {
        int d = k - 24;
        std::cout << d << " " << d + 5 << std::endl;
```

```
        for (int i = 0; i < d - 1; ++i) {
            std::cout << i + 1 << " " << i + 2 << std::endl;
        }
        std::cout << d << " 1" << std::endl;

        std::cout << "1 3" << std::endl;
        std::cout << "3 5" << std::endl;
        std::cout << "5 7" << std::endl;
        std::cout << "2 4" << std::endl;
        std::cout << "4 6" << std::endl;
        return 0;
    }

    if (k - 32 <= 20) {
        int d = k - 32;
        std::cout << d << " " << d + 6 << std::endl;
        for (int i = 0; i < d - 1; ++i) {
            std::cout << i + 1 << " " << i + 2 << std::endl;
        }
        std::cout << d << " 1" << std::endl;

        std::cout << "1 3" << std::endl;
        std::cout << "3 5" << std::endl;
        std::cout << "5 7" << std::endl;
        std::cout << "2 4" << std::endl;
        std::cout << "4 6" << std::endl;
        std::cout << "6 8" << std::endl;
        return 0;
    }

    if (k - 41 <= 20) {
        int d = k - 41;
        std::cout << d << " " << d + 7 << std::endl;
        for (int i = 0; i < d - 1; ++i) {
            std::cout << i + 1 << " " << i + 2 << std::endl;
        }
        std::cout << d << " 1" << std::endl;

        std::cout << "1 3" << std::endl;
        std::cout << "3 5" << std::endl;
        std::cout << "5 7" << std::endl;
        std::cout << "7 9" << std::endl;
        std::cout << "2 4" << std::endl;
        std::cout << "4 6" << std::endl;
        std::cout << "6 8" << std::endl;
        return 0;
    }
    // for (int aaa = 0; aaa < 1000; ++aaa) {
    // graph_edges_t ge = gen_graph(n, m);
```

```
    // graph_t g = read_graph(ge, n);
    // int ans = calc(n, m, g);
    // // for (int i = 0; i < n; ++i) {
    // // for (int j = i + 1; j < n; ++j) {
    // // // std::cout << "find path " << i << " " << j << std::endl;
    // // if (find_path(i, j, g)) {
    // // // std::cout << "found path " << i + 1 << " " << j + 1 << std::endl;
    // // ++ans;
    // // }
    // // }
    // // }
    // std::cout << ans << std::endl;
    // if (ans == k) {
    // std::cout << n << " " << m << std::endl;
    // for (int i = 0; i < m; ++i) {
    // std::cout << ge[i].first + 1 << " " << ge[i].second + 1 << std::endl;
    // }
    // return 0;
    // }
    // }
    return 123;
}
```

Положение команды

№	Участник	A 157/225	B 38/131	C 0/43	D 35/199	E 141/237	F 130/334	G 55/73	H 110/278	I 5/17	J 37/54	K 8/23	L 19/35	M 159/321	Очки	Штраф
124	MAI #2	+ 00:31	—	—	-1 04:55	+1 01:39	-1 02:46	—	+3 03:30	—	—	—	—	+2 01:10	4	532

Задача 1. Драма

Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

После прочтения античной драмы и произведений Шекспира Густав решил, что многие драматические произведения можно разделить на пять частей: экспозицию, развитие действия, кульминацию, развязку и постпозицию.

При таком делении *уровень конфликта* в произведении во время экспозиции постоянен, во время развития действия строго возрастает и достигает пика в кульминации, во время развязки строго убывает и, наконец, во время постпозиции остаётся постоянным.

Экспозиция и постпозиция могут происходить мгновенно, то есть занимать один момент времени. Развитие действия и развязка занимают минимум два момента времени, кульминация всегда одномоментна. Смотрите рисунки на следующей странице.

Для нескольких произведений Густав выписал уровни конфликта в хронологическом порядке и просит вас определить, соответствуют ли они уровням конфликта при вышеописанном делении. Если соответствуют, он просит вас сообщить моменты времени, когда

1. экспозиция переходит в развитие действия — t_1;
2. уровень конфликта достигает кульминации — t_2;
3. развязка переходит в постпозицию — t_3.

Формат входных данных

В первой строке входного файла записано одно целое положительное число D — количество произведений, проанализированных Густавом. Далее в отдельных строках следует описание этих произведений.

Первым в строке расположено целое число T_d — количество моментов времени, для которых выписаны уровни конфликта в произведении, за которым следует T_d целых положительных чисел c_t, соответствующих уровню конфликта в момент времени t ($3 \leqslant T_d \leqslant 10^5$, $1 \leqslant d \leqslant D, 1 \leqslant c_t \leqslant 2 \cdot 10^9$, $1 \leqslant t \leqslant T_d$).

Гарантируется, что сумма всех T_d не превосходит 10^5.

Формат выходных данных

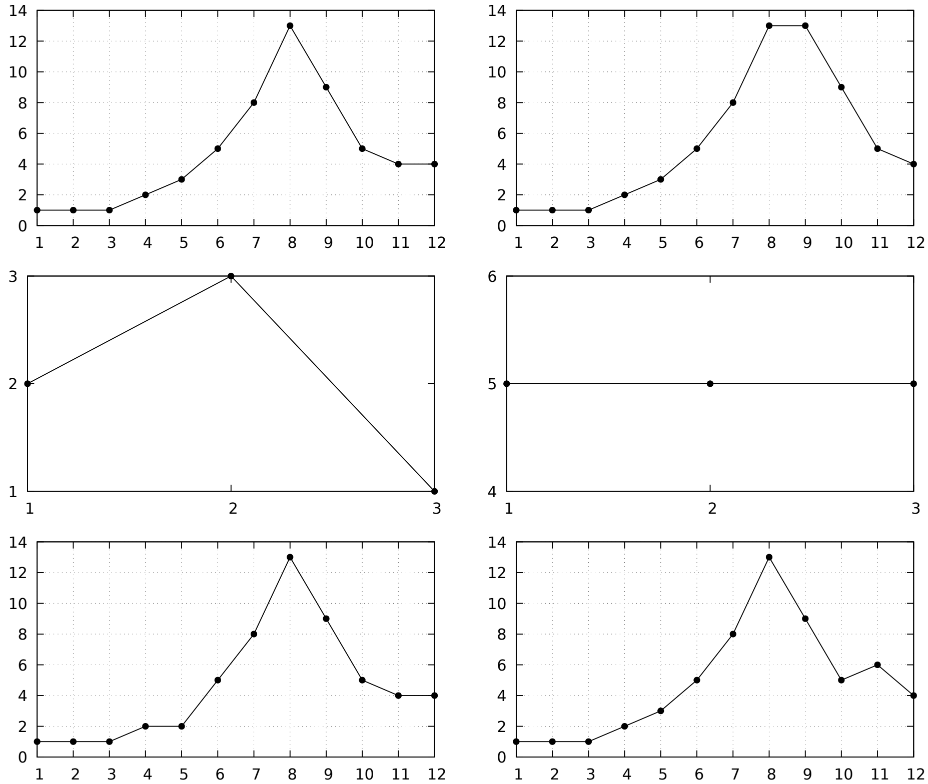
Для каждого произведения в порядке следования во входном файле выведите в отдельной строке выходного файла: `Freytag`, если уровни конфликта в произведении изменяются в соответствии с делением Густава, и `Nein` иначе. В случае ответа `Freytag` дополнительно выведите три целых положительных числа t_1, t_2, t_3 — вышеописанные моменты времени ($1 \leqslant t_i \leqslant T_d$, $1 \leqslant i \leqslant 3$, $t_1 < t_2 < t_3$).

Примеры

input.txt	output.txt
6 12 1 1 1 2 3 5 8 13 9 5 4 4 12 1 1 1 2 3 5 8 13 13 9 5 4 3 2 3 1 3 5 5 5 12 1 1 1 2 2 5 8 13 9 5 4 4 12 1 1 1 2 3 5 8 13 9 5 6 4	Freytag 3 8 11 Nein Freytag 1 2 3 Nein Nein Nein

XXII Открытая Всесибирская олимпиада 10.10.2021

Пояснение к примеру



Графики в порядке сверху вниз, слева направо соответствуют произведениям из примера входных данных. По горизонтальной оси отложены моменты времени, по вертикальной — уровни конфликта в произведениях.

1. Отрезок времени $[1;3]$ — экспозиция, $[3;8]$ — развитие действия, в момент времени 8 происходит кульминация, $[8;11]$ — развязка, $[11;12]$ — постпозиция.
2. После развития действия и кульминации не наступает сразу развязка.
3. Отрезок времени $[1;1]$ — экспозиция, $[1;2]$ — развитие действия, в момент времени 2 происходит кульминация, $[2;3]$ — развязка, $[3;3]$ — постпозиция.
4. Отсутствуют развитие действия, кульминация и развязка.
5. В зависимости от интерпретации либо при развитии действия нет строгого возрастания уровня конфликта на отрезке $[4;5]$, либо после кульминации в момент времени 4 не начинается развязка.
6. В зависимости от интерпретации либо после момента времени 10 в постпозиции не сохраняется постоянный уровень конфликта, либо в развязке уровень конфликта не убывает на отрезке $[10;11]$.

Идея

Для решения задачи требуется аккуратно реализовать проверку условий уровней конфликта. Если найденные моменты времени не противоречат условиям, то ответ «Freytag», иначе «Nein». Сложность решения $O(n)$.

Исходный код

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

int main() {
    std::ios::sync_with_stdio(false);
    std::cout.tie(0);
    std::cin.tie(0);

    int t;
    std::cin >> t;
    while (t--) {
        int n;
        std::cin >> n;
        std::vector<int> a(n);
        for (int i = 0; i < n; ++i) {
            std::cin >> a[i];
        }
        int t1 = -1, t2 = -1, t3 = -1;
        bool ans = true;
        for (int i = 0; i < n - 1; ++i) {
            if (a[i] != a[i + 1]) {
                t1 = i;
                break;
            }
        }
        for (int i = n - 1; i > 0; --i) {
            if (a[i] != a[i - 1]) {
                t3 = i;
                break;
            }
        }
        if (t3 <= t1 or t3 == -1 or t1 == -1) {
            ans = false;
        } else {
            for (int i = t1; i < std::min(t3, n - 1); ++i) {
                if (a[i + 1] <= a[i]) {
                    t2 = i;
                    break;
                }
            }
            if (t2 != -1) {
```

```cpp
                for (int i = t1; i < t2; ++i) {
                    ans = ans & (a[i] < a[i + 1]);
                }
                for (int i = t2; i < std::min(n - 1, t3); ++i) {
                    ans = ans & (a[i] > a[i + 1]);
                }
            } else {
                ans = false;
            }
            ans = ans and (t1 < t2 and t2 < t3);
        }
        if (ans) {
            std::cout << "Freytag " << t1 + 1 << ' ' << t2 + 1 << ' ' << t3 + 1 << '\n'
                ;
        } else {
            std::cout << "Nein\n";
        }
    }
}
```

Положение команды

Место	Участник	1	2	3	4	5	6	7	8	Задачи	Штраф
133	MAI #2	+4 66:56		+1 117:21		-1 288:52	-6 295:10	-9 273:28		2	283

Problem F. Fake or Leak?

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	512 megabytes

In Someland a great programming competition was held. The scoreboard is now frozen, and everyone is waiting for the final results announcement. At the same time, a group of hackers published a screenshot with the part of the final unfrozen scoreboard. However, looking on the published part of the scoreboard you have started to doubt it is not a fake.

You are given the frozen scoreboard and what is claimed to be a part of the final scoreboard. You have to find out whether it is possible that you are given a real part of the final scoreboard.

The rules of this competition are as follows:

1. There are m teams participating and n problems available to solve.
2. The contest lasts for five hours. The scoreboard is frozen after four hours.
3. Each time a team submits a solution for some problem, this submission is either accepted or rejected. Each team is allowed to submit each problem multiple times. A team never submits a solution for a problem for which it already made a successful submission.
4. Teams are ranked according to the most problems solved.
5. Teams that have solved the same number of problems are ranked by total time (the less the better) and, if need be, by the earliest time of the last accepted submission.
6. The total time is the sum of the time consumed for each problem solved. The time consumed for a solved problem is the time elapsed from the beginning of the contest to the submittal of the first accepted run plus 20 penalty minutes for every previously rejected run for that problem. There is no time consumed for a problem that is not solved.
7. If two or more teams are still tied by the total number of problems solved, the total time consumed and the time of the last accepted submission, they are ranked by their names in lexicographical order.

You are given a frozen scoreboard and what is claimed to be k **consecutive** lines of the final scoreboard. Your goal is to check whether it is possible that the claim is true.

Input

The first line of the input contains three integers n, m and k ($1 \le n \le 26$, $1 \le k \le m \le 1000$) — the size of the problem set, the number of teams participating, and the size of the leaked part of the final scoreboard.

In the next m lines, the results of the teams in the frozen scoreboard are listed from top to bottom, i.e. from the best to the worst result. Then follow k lines describing the allegedly leaked continuous part of the final scoreboard.

Each line that describes the results of a team obeys the following format. The line starts with a team name. The name is a non-empty string of at most 10 lower-case English letters. Then follow n triples c, a, t ($0 \le a \le 100$, $0 \le t \le 299$), describing the results of this team for each problem. The character c is one of '+', '-', or '.', meaning the team has already solved this problem, the team has made some unsuccessful attempts and haven't solved this problem yet, or the team has never attempted this problem respectively. Integer a stands for the number of submissions, and integer t stands for the minute when the last submission was made.

It is guaranteed that both scoreboards are self-consistent and non-contradictory. In particular, that means:

1. All names in the frozen scoreboard are distinct. All names in the allegedly leaked part of the final scoreboard are distinct. Any name that is present in the allegedly leaked part of the scoreboard is present in the frozen scoreboard.
2. If some c equals '`+`' or '`-`', the corresponding value of a is positive.
3. If some c equals '`.`', the corresponding values of a and t are 0.
4. If some a equals 0, corresponding value of t is 0 and c is '`.`'.
5. Both the frozen scoreboard and the allegedly leaked part have teams ranked according to the rules described above.
6. All values of t in the frozen scoreboard do not exceed 239.
7. Any team's results present in the allegedly leaked scoreboard is consistent with its results in the frozen scoreboard. Let c_1, a_1 and t_1 be some team's result for some problem in the frozen scoreboard, and c_2, a_2 and t_2 be results of the same team for the same problem in the allegedly leaked part of the scoreboard. Then,
   - $a_1 \leq a_2$;
   - if $a_1 = a_2$, then $c_1 = c_2$ and $t_1 = t_2$;
   - if $c_1 =$ '`+`', then $c_2 =$'`+`' and $a_1 = a_2$;
   - if $a_1 < a_2$, then c_1 is either '`-`' or '`.`', c_2 is either '`+`' or '`-`', and $t_2 \geq 240$.

Output

If it is possible that the last k lines of the input are k consecutive lines of the final scoreboard, print "`Leaked`" (without quotes). Otherwise, print "`Fake`" (without quotes).

Examples

standard input	standard output
3 3 2 crabs + 1 1 + 1 2 + 1 3 lions . 0 0 - 5 239 . 0 0 wombats . 0 0 . 0 0 . 0 0 wombats + 1 241 + 3 299 - 22 299 lions + 1 241 + 6 240 - 3 299	Leaked
3 4 2 crabs + 1 1 + 1 2 + 1 3 lions . 0 0 + 5 239 . 0 0 wolves . 0 0 . 0 0 . 0 0 wombats . 0 0 . 0 0 . 0 0 crabs + 1 1 + 1 2 + 1 3 wombats . 0 0 + 2 299 . 0 0	Fake

Идея

Для проверки подлинности таблицы составим наихудший и наилучший возможный рейтинг команды. Зная это, мы можем легко проверить, является ли часть таблицы верной. Асимптотика решения $O(n \cdot (k + m))$.

Исходный код

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

struct problem {
    std::string solved;
    int a;
    int t;
};

struct team {
    std::string name;
    std::vector<problem> total;

    team() = default;

    team(const std::string & s, const std::vector<problem> & stotal) {
        name = s;
        total = stotal;
    }

    int getSolved() {
        int res = 0;
        for (size_t i = 0; i < total.size(); ++i) {
            if (total[i].solved == "+") {
                ++res;
            }
        }
        return res;
    }

    int getPenalty() {
        int res = 0;
        for (size_t i = 0; i < total.size(); ++i) {
            if (total[i].solved == "+") {
                res = res + total[i].t + 20 * (total[i].a - 1);
            }
        }
        return res;
    }

    int getLastAc() {
```

```
        int res = 0;
        for (size_t i = 0; i < total.size(); ++i) {
            if (total[i].solved == "+" and total[i].t >= res) {
                res = total[i].t;
            }
        }
        return res;
    }
};

struct team_score {
    int solved;
    int penalty;
    int lastAc;
    std::string name;

    team_score(team t) {
        name = t.name;
        solved = t.getSolved();
        penalty = t.getPenalty();
        lastAc = t.getLastAc();
    }

    friend bool operator < (const team_score & lhs, const team_score & rhs) {
        if (lhs.solved != rhs.solved) {
            return lhs.solved > rhs.solved;
        } else if (lhs.penalty != rhs.penalty) {
            return lhs.penalty < rhs.penalty;
        } else if (lhs.lastAc != rhs.lastAc) {
            return lhs.lastAc < rhs.lastAc;
        } else {
            return lhs.name < rhs.name;
        }
    }
};

team_score getBetter(team t) {
    for (size_t i = 0; i < t.total.size(); ++i) {
        if (t.total[i].solved != "+") {
            t.total[i].solved = "+";
            ++t.total[i].a;
            t.total[i].t = 240;
        }
    }
    return team_score(t);
}

team_score getWorse(team t) {
    return team_score(t);
```

```
}

team readTeam(int n) {
    std::string s;
    std::cin >> s;
    std::vector<problem> stotal;
    problem cur;
    for (int i = 0; i < n; ++i) {
        std::cin >> cur.solved >> cur.a >> cur.t;
        stotal.push_back(cur);
    }
    team res(s, stotal);
    // std::cout << "team name = " << s << res.name << std::endl;
    return res;
}

int main() {
    std::ios::sync_with_stdio(false);
    std::cout.tie(0);
    std::cin.tie(0);

    int n, m, k;
    std::cin >> n >> m >> k;
    std::vector<team> frozen;
    for (int i = 0; i < m; ++i) {
        team cur = readTeam(n);
        frozen.push_back(cur);
    }
    std::set<std::string> used;
    std::vector<team> leak;
    for (int i = 0; i < k; ++i) {
        team cur = readTeam(n);
        leak.push_back(cur);
        used.insert(cur.name);
        // std::cout << "insert " << leak.back().name << std::endl;
    }
    team_score best(leak[0]);
    team_score worst(leak.back());
    bool ans = false;

    int could_be_better = 0;
    int could_be_worse = 0;
    int could_be = 0;
    std::set<std::string> used_better;
    for (int i = 0; i < m; ++i) {
        if (used.count(frozen[i].name)) {
            continue;
        }
        team_score better = getBetter(frozen[i]);
```

```cpp
        team_score worse = getWorse(frozen[i]);
        // std::cout << frozen[i].name << " bet = " << better.solved << " wor = " << worse.solved << std::endl;
        if (worst < worse) {
            if (better < best) {
                ++could_be;
            } else {
                ++could_be_worse;
            }
        } else if (better < best) {
            ++could_be_better;
        }
    }

    // std::cout << "worse " << could_be_worse << " better" << could_be_better << " be" << could_be << std::endl;
    if (could_be_worse + could_be_better + could_be == m - k) {
        ans = true;
    }

    if (ans) {
        std::cout << "Leaked\n";
    } else {
        std::cout << "Fake\n";
    }
}
```

Положение команды

1 ICPC training MAI 21-22

Результаты

Вы можете использовать двойной щелчок (или Ctrl + щелчок) по ячейкам таблицы для просмотра истории

Положение

Совпадений: 2 MAI #2

№	Кто	=	Штраф	A	B	C	D	E	F	G	H	I	J
1	MAI #2 Artemiev, Belousov, Inyutin	3	468	+4 01:12		-6	+3 02:02	-1	+2 01:34		-7	-1	
	* MAI #2 Artemiev, Belousov, Inyutin	2				-1				+	+4		

Grand Prix of Korea 24.10.2021

Problem H. Or Machine

Input file:	`standard input`
Output file:	`standard output`
Time limit:	4 seconds
Memory limit:	1024 megabytes

We are developing the Or Machine, a computer heavily optimized solely for one kind of operation: the `|=` operator in C++'s term.

The Or Machine has n registers, each containing a nonnegative integer less than 2^8. We label them $x_1, x_2, \cdots, x_n$. A program is represented by a list of l operations. Each operation is represented by a pair of integers (a, b), meaning that the machine should update x_a with the bitwise OR of x_a's and x_b's values.

The Or Machine takes a program, the initial values of the registers, and a positive integer t. When run, the program performs each operation in the program one by one. When the last operation is performed, it goes back to the first operation and repeats the process. The machine stops after performing exactly t operations.

We want our machine to be much faster than general-purpose computers, and hardware optimization is probably not enough. Can you help us with some software optimization?

Input

The first line contains three integers, n, l, and t ($1 \le n, l \le 2^{18}$, $1 \le t \le 10^{18}$). l is the length of the program.

The program is given on the next l lines. Each line contains two integers a and b ($1 \le a, b \le n$) representing the pair of registers that participate in the given operation.

The final line contains n integers, the initial values of the registers $x_1, \cdots, x_n$ ($0 \le x_i < 2^8$).

Output

Output n integers on a single line, the values of the registers $x_1, \cdots, x_n$ after t operations.

Example

standard input	standard output
5 4 5 1 2 2 3 2 4 4 4 8 0 5 3 10	15 7 5 3 10

Идея

Давайте решим задачу для одного бита. Понятно, что решив такую задачу мы сможем с легкостью решить изначальную задачу ввиду того, что OR одного бита не влияет на остальные.

Построим граф, вершины будут представлять собой регистры. Добавим ориентированные ребра, ребро из a в b с весом w будет обозначать, что w команда будет присваивать $x_b = x_b|x_a$. Тогда нам потребуется найти минимальное время для каждой вершины, через которое произойдет замена бита 0 на 1. Это проблема легко решается с помощью алгоритма Дейкстры: просто запустим Дейкстру из всех таких регистров, в которых изначально стоит 1, и немного поменяем метрику при пересчете расстояний.

Проделаем данный алгоритм для всех 8 битов. В конце концов, для каждой вершины поставим 1 в i-ом бите, если мы успеем его достичь быстрее чем за t операций.

Итого решение работает за такую асимптотику $O(k \cdot l \cdot \log n)$, где k - это количество битов, в нашем случае $k = 8$.

Исходный код

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

using namespace std;

#define flush cout.flush

using ll = long long;
using ull = unsigned long long;
using ld = long double;
using pl = pair<ll, ll>;
const ll INF = 2e18;
const ll mod = 1e9 + 7;
const ll mod2 = 998244353;
const ld eps = 1e-9;
const ld PI = acos(-1);

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    ll n, l, t;
    cin >> n >> l >> t;
    vector<vector<pl>> g(n);
    for (ll i = 0; i < l; ++i) {
        ll a, b;
        cin >> a >> b;
        a--;
```

```
        b--;
        g[b].push_back({a, i});
    }
    vector<ll> a(n);
    for (ll &i : a)cin >> i;
    vector<ll> res(n, 0);
    for (ll i = 0; i < 8; ++i) {
        set<pl> s;
        vector<ll> d(n, INF);
        for (ll j = 0; j < n; ++j) {
            if ((1ll << i) & a[j]) {
                d[j] = -1;
                s.insert({d[j], j});
            }
        }
        while (!s.empty()) {
            auto it = *(s.begin());
            s.erase(it);
            ll v = it.second;
            ll w = it.first;
            for (auto to : g[v]) {
                ll rem = (w % l);
                ll edge_rem = (to.second % l);
                ll new_w = (edge_rem > rem ? w + edge_rem - rem : (w / l + 1) * l + edge_rem);
                if (w == -1)new_w = edge_rem;
                if (new_w < d[to.first]) {
                    s.erase({d[to.first], to.first});
                    d[to.first] = new_w;
                    s.insert({d[to.first], to.first});
                }
            }
        }
        for (ll k = 0; k < n; ++k) {
            if (d[k] + 1 <= t) {
                res[k] |= (1ll << i);
            }
        }
    }
    for (auto x : res) {
        cout << x << " ";
    }
    return 0;
}
```

Положение команды

№	Участник Я	A 47/108	B 7/52	C 64/151	D 4/14	E 44/229	F 1/4	G 18/28	H 70/185	I 2/3	J 68/166	K 34/163	L 2/31	M 4/11	Очки	Штраф
69	MAI #2 Artemiev, Belousov, Inyutin :	—	—	—	—	-6 04:59	—	—	+7 02:29	—	+3 03:43	—	—	—	2	572

Problem 5. Hockey

Input file:	`input.txt` or *standard input*
Output file:	`output.txt` or *standard output*
Time limit:	1 second
Memory limit:	256 mebibytes

Rules described in this problem differ from the conventional hockey rules.

A hockey match lasts 60 minutes, with two teams trying to score as many goals as possible. A hockey team consists of five field players and a goalkeeper.

Penalties are an important part of hockey. A field player can be given a penalty: in this case, the offending player leaves the ice for a period of time which depends on the violation. As the result, the number of players on the ice temporarily decreases for the team the offending player belongs to. There are two types of penalties in hockey: *major* and *minor*. A major penalty means the player leaves the ice for five minutes; with minor penalty, it is two minutes. When penalty time runs out, the player returns to the ice.

A minor penalty can be ended prematurely. A team is said to be playing short-handed when it has less players on the ice than the other team. If a team is playing short-handed and opponent scores a goal, then one of its players with minor penalty returns to the ice with his penalty expired ahead of time. If the team has several players with minor penalty, only the player who got the penalty first returns to the ice. If there are no players with minor penalties in the team, no one returns ahead of time.

Penalties during the game mean that the teams can play in various formats regarding the number of field players on the ice. We will denote the game format by `AxB`, meaning the first team currently has `A` field players on the ice, and the second team has `B`. For instance, in the beginning of the game each team has five players on the ice, and this format is denoted as `5x5`. If the first team currently has two players with penalty, and the second team has one, the format is denoted as `3x4`.

You are given a game protocol, registering the time of all penalties and goals. Calculate which formats happened during the game and for how long each format was played.

Input

The first line of the input file contains an integer N — the number of events in the match ($0 \le N \le 1\,000$).

The following N lines describe the events of the match, one per line. Events are described in the following format:

```
mm:ss.d team type
```

Where `mm:ss.d` — time of event with the precision of tenths of a second ($0 \le$ `mm` ≤ 59, $0 \le$ `ss` ≤ 59, $0 \le$ `d` ≤ 9), `team` — team number (either 1 or 2), `type` — event type:

- `goal` — team scores a goal;
- `minor` — team player receives minor penalty;
- `major` — team player receives major penalty.

It is guaranteed that events of the type `goal` have non-zero decimal of a second, i.e. `d` $\ne 0$, and events of the type `minor` and `major` always have zero decimals of a second, i.e. `d` $= 0$.

Events are listed chronologically, i.e. they are arranged in the order of non-reduction of event times. It is guaranteed that at any moment of time each team has no more than 5 players.

Output

For each format of the game in which the teams have played non-zero time, print the format denotation and the time spent by the teams in this format in a separate line, separated by a space character. The format of time must be exactly the same as the format used in the input data. Lines can be printed in arbitrary order.

Example

input.txt or standard input	output.txt or standard output
10 06:41.0 1 minor 07:20.4 2 goal 22:22.0 2 minor 22:32.0 1 minor 23:00.1 1 goal 23:12.0 2 minor 23:59.9 1 goal 41:02.0 1 major 41:04.5 2 goal 59:00.0 1 minor	4x3 00:47.9 4x4 01:12.1 4x5 06:39.4 5x4 00:50.0 5x5 50:30.6

Example explanation

The game from the example had the following intervals:

- `[00:00.0; 06:41.0)` — until the first penalty, the game went in the initial format `5x5`;
- `[06:41.0; 07:20.4)` — after a penalty, the teams were playing in the format `4x5` until the first team lost a goal while playing short-handed, and the player removed for a minor penalty returned to the ice;
- `[07:20.4; 22:22.0)` — the teams were playing in full numbers `5x5` until a penalty;
- `[22:22.0; 22:32.0)` — until the next penalty, the teams were playing in the format `5x4`;
- `[22:32.0; 23:00.1)` — until a goal was scored, the teams were playing in the format `4x4`, however, no players returned to the ice after the goal, because it was scored with equally-sized teams;
- `[23:00.1; 23:12.0)` — the teams continued playing in the format `4x4` until another penalty;
- `[23:12.0; 23:59.9)` — after that, the teams were playing in the format `4x3` until a goal was scored, and the second team player who had been penalized at `22:22.0` returned to the ice;
- `[23:59.9; 24:32.0)` — the teams were playing `4x4` until the first team player's penalty ran out;
- `[24:32.0; 25:12.0)` — the teams were playing in the `5x4` format until the second team player's penalty ran out;
- `[25:12.0; 41:02.0)` — the teams were playing in the full format `5x5` until a major penalty;
- `[41:02.0; 41:04.5)` — before the goal, the teams played in the format `4x5`, but because a player of the team that lost a goal had a major penalty, that player does **not** leave the penalty box;
- `[41:04.5; 46:02.0)` — the teams continued playing as `4x5`, until the first player's penalty ran out;
- `[46:02.0; 59:00.0)` — before the penalty, the teams were playing with all players `5x5`;
- `[59:00.0; 60:00.0)` — teams ending the game in the format `4x5`.

Идея

Основная сложность — правильно реализовать систему штрафов в хоккее. Для этого каждую секунду игры. Если в какой-то момент времени на поле произошло событие, то его следует обработать согласно правилам игры. Для этого для каждого игрока обеих команд будем хранить тип штрафа и оставшееся время вне игры, постепенно уменьшая его. Остаётся проверять, сколько игроков на поле и увеличивать время для соответствующего состояния игры. Моделирование работает за константу, поэтому сложность решения $O(60 \cdot 60 \cdot 10 + n) \approx O(n)$.

Исходный код

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

int parse_time(const std::string & s) {
    int minutes = std::stoi(s.substr(0, 2));
    int seconds = std::stoi(s.substr(3, 2));
    int mseconds = s[6] - '0';
    return 60 * 10 * minutes + 10 * seconds + mseconds;
}

std::string to_time(int time) {
    int mseconds = time % 10;
    int seconds = (time / 10) % 60;
    int minutes = time / 600;
    std::string s1;
    s1.push_back(minutes / 10 + '0');
    s1.push_back(minutes % 10 + '0');
    std::string s2;
    s2.push_back(seconds / 10 + '0');
    s2.push_back(seconds % 10 + '0');
    std::string s3;
    s3.push_back(mseconds + '0');
    return s1 + ":" + s2 + "." + s3;
}

struct item_t {
    int t;
    int team;
    std::string what;

    friend bool operator < (const item_t & lhs, const item_t & rhs) {
        return lhs.t < rhs.t;
    }
};

int main() {
```

```
    std::ios::sync_with_stdio(false);
    std::cout.tie(0);
    std::cin.tie(0);

    const int MAX_TIME = 60 * 60 * 10;

    int n;
    std::cin >> n;
    std::vector<item_t> data;
    for (int i = 0; i < n; ++i) {
        std::string tt;
        int team;
        std::string s;
        std::cin >> tt >> team >> s;
        data.push_back({parse_time(tt), team, s});
    }
    std::stable_sort(data.begin(), data.end());

    const int TN = 5;
    const int PLAY = 0;
    const int MINOR = 1;
    const int MAJOR = 2;
    const int MINOR_TIME = 2 * 60 * 10;
    const int MAJOR_TIME = 5 * 60 * 10;
    std::vector<int> penalty_a(TN);
    std::vector<int> penalty_ta(TN);
    std::vector<int> penalty_b(TN);
    std::vector<int> penalty_tb(TN);
    std::vector< std::vector<int> > res(TN + 1, std::vector<int>(TN + 1));
    int j = 0;
    for (int i = 0; i < MAX_TIME; ++i) {

        for (int ii = 0; ii < TN; ++ii) {
            if (penalty_ta[ii] != PLAY) {
                --penalty_a[ii];
                if (penalty_a[ii] == 0) {
                    penalty_ta[ii] = PLAY;
                }
            }
            if (penalty_tb[ii] != PLAY) {
                --penalty_b[ii];
                if (penalty_b[ii] == 0) {
                    penalty_tb[ii] = PLAY;
                }
            }
        }

        while (j < n and data[j].t <= i) {
            item_t event = data[j];
```

```
            if (event.team == 1) {
                if (event.what == "major") {
                    for (int ii = 0; ii < TN; ++ii) {
                        if (penalty_ta[ii] == PLAY) {
                            penalty_ta[ii] = MAJOR;
                            penalty_a[ii] = MAJOR_TIME;
                            break;
                        }
                    }
                }

                if (event.what == "minor") {
                    for (int ii = 0; ii < TN; ++ii) {
                        if (penalty_ta[ii] == PLAY) {
                            penalty_ta[ii] = MINOR;
                            penalty_a[ii] = MINOR_TIME;
                            break;
                        }
                    }
                }

                if (event.what == "goal") {
                    int count_a = 0;
                    int count_b = 0;
                    for (int ii = 0; ii < TN; ++ii) {
                        if (penalty_ta[ii] == PLAY) {
                            ++count_a;
                        }
                        if (penalty_tb[ii] == PLAY) {
                            ++count_b;
                        }
                    }
                    if (count_b < count_a) {
                        int min_time_left = -1;
                        for (int ii = 0; ii < TN; ++ii) {
                            if (penalty_tb[ii] == MINOR) {
                                if (min_time_left == -1 or penalty_b[ii] < penalty_b[
                                    min_time_left]) {
                                    min_time_left = ii;
                                }
                            }
                        }
                        if (min_time_left != -1) {
                            penalty_tb[min_time_left] = PLAY;
                            penalty_b[min_time_left] = 0;
                        }
                    }
                }
            }
```

```

        if (event.team == 2) {
            if (event.what == "major") {
                for (int ii = 0; ii < TN; ++ii) {
                    if (penalty_tb[ii] == PLAY) {
                        penalty_tb[ii] = MAJOR;
                        penalty_b[ii] = MAJOR_TIME;
                        break;
                    }
                }
            }

            if (event.what == "minor") {
                for (int ii = 0; ii < TN; ++ii) {
                    if (penalty_tb[ii] == PLAY) {
                        penalty_tb[ii] = MINOR;
                        penalty_b[ii] = MINOR_TIME;
                        break;
                    }
                }
            }

            if (event.what == "goal") {
                int count_a = 0;
                int count_b = 0;
                for (int ii = 0; ii < TN; ++ii) {
                    if (penalty_ta[ii] == PLAY) {
                        ++count_a;
                    }
                    if (penalty_tb[ii] == PLAY) {
                        ++count_b;
                    }
                }
                if (count_a < count_b) {
                    int min_time_left = -1;
                    for (int ii = 0; ii < TN; ++ii) {
                        if (penalty_ta[ii] == MINOR) {
                            if (min_time_left == -1 or penalty_a[ii] < penalty_a[min_time_left]) {
                                min_time_left = ii;
                            }
                        }
                    }
                    if (min_time_left != -1) {
                        penalty_ta[min_time_left] = PLAY;
                        penalty_a[min_time_left] = 0;
                    }
                }
            }
```

```
            }
            ++j;
        }
        {
            int count_a = 0;
            int count_b = 0;
            for (int ii = 0; ii < TN; ++ii) {
                if (penalty_ta[ii] == PLAY) {
                    ++count_a;
                }
                if (penalty_tb[ii] == PLAY) {
                    ++count_b;
                }
            }
            ++res[count_a][count_b];
        }
    }

    for (int i = 0; i < TN + 1; ++i) {
        for (int j = 0; j < TN + 1; ++j) {
            if (res[i][j] > 0) {
                std::cout << i << 'x' << j << " " + to_time(res[i][j]) << '\n';
            }
        }
    }
}
```

Положение команды

№	Участник Я	1	2	3	4	5	6	7	8	9	10	11	Очки	Штраф
		52/631	26/214	70/70	8/35	65/151	58/165	43/114	69/174	47/194	33/198	5/12		
57	MAI #2 Artemiev, Belousov, Inyutin	+10 02:11	—	+ 00:14	—	+3 02:20	-16 04:56	-5 04:59	+1 00:33	—	-2 03:54	—	4	600

Problem B. A Plus B Problem

Input file:	*standard input*
Output file:	*standard output*
Time limit:	3 seconds
Memory limit:	512 mebibytes

JB gets a machine that can solve "A Plus B Problem" and feels curious about the mechanism. He hears that you are proficient in competitive programming and have learned many advanced data structures and algorithms such as Link-Cut tree, Lagrange Inversion formula, Sweepline Mo, and so on. Hence, he asks you to help implement a program that can solve "A Plus B Problem" as same as the machine.

The machine consists of $3 \times n$ digits. The digits of the first two rows can be changed arbitrarily, and the third row always equals the decimal sum of the first two rows. The third row only consists of the lowest n digits even if the sum exceeds n digits.

For example, when $n = 5$, the three rows can be "01234", "56789", "58023" or "56789", "58023", "14812".

To test your function, you are given q queries. In the i-th query, the c_i-th digit of the r_i-th row is updated to d_i (the digit may not change). Because the digits are too many and JB has no time to check your answer, he only asks you to find the c_i-th digit of the third row after the query and how many digits of the machine change in the query.

Input

The first line contains two integers n and q ($1 \le n, q \le 10^6$).

The second line contains a string consisting of n digits, representing the first row of the machine.

The third line contains a string consisting of n digits, representing the second row of the machine.

There are q lines in the following. The i-th of the following line consists of three integers r_i, c_i and d_i ($1 \le r_i \le 2$, $1 \le c_i \le n$, $0 \le d_i \le 9$).

Output

Output q lines. In the i-th line, output two integers - the c_i-th digit of the third row after the query and how many digits of the machine change in the query.

Example

standard input	standard output
5 5 01234 56789 2 1 0 2 2 1 2 3 2 2 4 3 2 5 4	0 2 3 2 5 3 7 3 8 3

Note

In the example, the initial rows are "01234", "56789", "58023".

After the 1-st query, the rows are "01234", "06789", "08023".

After the 2-nd query, the rows are "01234", "01789", "03023".

After the 3-th query, the rows are "01234", "01289", "02523".

After the 4-th query, the rows are "01234", "01239", "02473".

Идея

Тривиальный случай, когда изменяется только одна цифра суммы, нам не очень интересен, так как изменяется всего две цифры во всех числах. Гораздо более сложный случай — замена девяток на нули и нулей на девятки при изменении суммы. Заметим, что для какого-то префикса числа достаточно знать количество цифр «0» и «9», чтобы корректно отвечать на запрос.

Используем струкуру данных, поддерживающую модификацию на отрезке для эффективного ответа на запрос и изменение. Во время контеста было реализовано решение с использованием Декартова дерева, которое не прошло по времени из-за большой константы. При дорешивании было реализовано решение, использующее дерево отрезков.

Сложность операций с деревом $O(\log n)$. Асимптотика решения $O(n \cdot \log n + q \cdot \log n)$.

Исходный код

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

#ifndef TREAP_HPP
#define TREAP_HPP

#include <ctime>
#include <iostream>

class treap_t {
    static const int TAB_SIZE = 4;

    struct treap_node_t {
        treap_node_t * _left;
        treap_node_t * _right;
        int _key;
        int _priority;

        int _value;
        int _delay;
        int _suf0;
        int _suf9;

        treap_node_t(int key, int num) {
            _left = nullptr;
            _right = nullptr;
            _key = key;
            _priority = std::rand();
            _value = num;
            _suf0 = 0;
            _suf9 = 0;
```

```
            if (num == 0) {
                ++_suf0;
            }
            if (num == 9) {
                ++_suf9;
            }
            _delay = -1;
        }
    };

    using treap_ptr = treap_node_t *;

    treap_ptr root;

    int get_key(treap_ptr t) {
        if (t != nullptr) {
            return t->_key;
        } else {
            return 0;
        }
    }

    int get_value(treap_ptr t) {
        if (t != nullptr) {
            return t->_value;
        } else {
            return 0;
        }
    }

    int get_suf0(treap_ptr t) {
        if (t != nullptr) {
            if (t->_delay != -1) {
                return t->_delay == 0 ? t->_key : 0;
            } else {
                return t->_suf0;
            }
        } else {
            return 0;
        }
    }

    int get_suf9(treap_ptr t) {
        if (t != nullptr) {
            if (t->_delay != -1) {
                return t->_delay == 9 ? t->_key : 0;
            } else {
                return t->_suf9;
            }
```

```
        } else {
            return 0;
        }
    }

    void enqueue_delay(treap_ptr t, int delay_value) {
        if (t != nullptr) {
            t->_delay = delay_value;
        }
    }

    void update(treap_ptr t) {
        if (t != nullptr) {
            t->_key = 1 + get_key(t->_left) + get_key(t->_right);
            int delay_val = t->_delay;
            if (delay_val != -1) {
                t->_value = delay_val;
                t->_delay = -1;
                t->_suf0 = 0;
                t->_suf9 = 0;
                if (delay_val == 0) {
                    t->_suf0 = t->_key;
                }
                if (delay_val == 9) {
                    t->_suf9 = t->_key;
                }
                enqueue_delay(t->_left, delay_val);
                enqueue_delay(t->_right, delay_val);
            } else {
                if (get_key(t->_right) == get_suf0(t->_right)) {
                    if (t->_value == 0) {
                        t->_suf0 = get_suf0(t->_left) + get_suf0(t->_right) + 1;
                    } else {
                        t->_suf0 = get_suf0(t->_right);
                    }
                } else {
                    t->_suf0 = get_suf0(t->_right);
                }

                if (get_key(t->_right) == get_suf9(t->_right)) {
                    if (t->_value == 9) {
                        t->_suf9 = get_suf9(t->_left) + get_suf9(t->_right) + 1;
                    } else {
                        t->_suf9 = get_suf9(t->_right);
                    }
                } else {
                    t->_suf9 = get_suf9(t->_right);
                }
            }
```

```
        }
    }

    void split(treap_ptr t, int key, treap_ptr & l, treap_ptr & r) {
        if (t == nullptr) {
            l = nullptr;
            r = nullptr;
            return;
        }
        update(t);
        int key_left_subtree = get_key(t->_left);
        if (key > key_left_subtree + 1) {
            split(t->_right, key - key_left_subtree - 1, t->_right, r);
            l = t;
        } else {
            split(t->_left, key, l, t->_left);
            r = t;
        }
        update(l);
        update(r);
    }

    treap_ptr merge(treap_ptr l, treap_ptr r) {
        if (l == nullptr) {
            return r;
        }
        if (r == nullptr) {
            return l;
        }
        update(l);
        update(r);
        if (l->_priority > r->_priority) {
            l->_right = merge(l->_right, r);
            update(l);
            return l;
        } else {
            r->_left = merge(l, r->_left);
            update(r);
            return r;
        }
    }

    void delete_treap(treap_ptr t) {
        if (t != nullptr) {
            delete_treap(t->_left);
            delete_treap(t->_right);
            delete t;
        }
    }
```

```
    friend std::ostream & operator << (std::ostream & out, treap_t t) {
        print_treap(out, t.root, 0);
        return out;
    }

    friend std::ostream & operator << (std::ostream & out, treap_ptr t) {
        print_treap(out, t, 0);
        return out;
    }

    friend void print_treap(std::ostream & out, treap_ptr t, int h) {
        if (t != nullptr) {
            print_treap(out, t->_left, h + 1);
            for (int i = 0; i < TAB_SIZE * h; ++i) {
                out << " ";
            }
            out << "{ x = " << t->_key << ", val = " << t->_value << ", suf0 = " << t->
                _suf0 << ", suf9 = " << t->_suf9 << ", delay = " << t->_delay << "}\n";
            print_treap(out, t->_right, h + 1);
        }
    }

public:
    treap_t() {
        std::srand(std::time(NULL));
        root = nullptr;
    }

    void enqueue(int key_l, int key_r, int value) {
        treap_ptr l = nullptr;
        treap_ptr m = nullptr;
        treap_ptr r = nullptr;
        split(root, key_r + 1, m, r);
        split(m, key_l, l, m);
        enqueue_delay(m, value);
        root = merge(merge(l, m), r);
    }

    int get(int key_l, int key_r) {
        treap_ptr l = nullptr;
        treap_ptr m = nullptr;
        treap_ptr r = nullptr;
        split(root, key_r + 1, m, r);
        split(m, key_l, l, m);
        int ans = get_value(m);
        root = merge(merge(l, m), r);
        return ans;
```

```cpp
    }

    int get0(int key_l, int key_r) {
        treap_ptr l = nullptr;
        treap_ptr m = nullptr;
        treap_ptr r = nullptr;
        split(root, key_r + 1, m, r);
        split(m, key_l, l, m);
        int ans = get_suf0(m);
        root = merge(merge(l, m), r);
        return ans;
    }

    int get9(int key_l, int key_r) {
        treap_ptr l = nullptr;
        treap_ptr m = nullptr;
        treap_ptr r = nullptr;
        split(root, key_r + 1, m, r);
        split(m, key_l, l, m);
        int ans = get_suf9(m);
        root = merge(merge(l, m), r);
        return ans;
    }

    void push_back(int value) {
        treap_ptr m = new treap_node_t(1, value);
        root = merge(root, m);
    }

    void push_front(int value) {
        treap_ptr m = new treap_node_t(1, value);
        root = merge(m, root);
    }

    void destruct() {
        delete_treap(root);
    }
};

#endif /* TREAP_HPP */

int main() {
    std::ios::sync_with_stdio(false);
    std::cout.tie(0);
    std::cin.tie(0);

    int n, q;
    std::cin >> n >> q;
    std::string str1, str2;
```

```cpp
    std::cin >> str1 >> str2;
    std::vector<int> s1(n);
    std::vector<int> s2(n);
    for (int i = 0; i < n; ++i) {
        s1[i] = str1[i] - '0';
        s2[i] = str2[i] - '0';
    }
    std::vector<int> arr(n);
    int remainder = 0;
    for (int i = n - 1; i >= 0; --i) {
        arr[i] = s1[i] + s2[i] + remainder;
        remainder = 0;
        if (arr[i] > 9) {
            arr[i] = arr[i] - 10;
            remainder = 1;
        }
    }
    treap_t t;
    for (int i = 0; i < n; ++i) {
        t.push_back(arr[i]);
    }
    // std::cout << t << "------" << std::endl;
    while (q--) {
        int r, c, d;
        std::cin >> r >> c >> d;
        // std::cout << "query " << r << ", " << c << ", " << d << '\n';
        bool dec = true;
        int num_was = t.get(c, c);
        int delta = 0;
        if (r == 1) {
            if (s1[c - 1] == d) {
                std::cout << num_was << " 0\n";
                continue;
            }
            delta = std::abs(s1[c - 1] - d);
            if (s1[c - 1] < d) {
                dec = false;
            }
            s1[c - 1] = d;
        }
        if (r == 2) {
            if (s2[c - 1] == d) {
                std::cout << num_was << " 0\n";
                continue;
            }
            delta = std::abs(s2[c - 1] - d);
            if (s2[c - 1] < d) {
                dec = false;
            }
```

```
            s2[c - 1] = d;
        }
        int ans = 1;
        if (dec) {
            int num_next = num_was - delta;
            if (num_next < 0) {
                int suf = t.get0(1, c - 1);
                int pos = c - 1 - suf;
                t.enqueue(c, c, num_next + 10);
                // std::cout << "dec suf = " << suf << ", pos = " << pos << ", num_next
                    = " << num_next << std::endl;
                if (suf == c - 1) {
                    t.enqueue(1, c - 1, 9);
                    ans = ans + suf + 1;
                } else {
                    t.enqueue(pos + 1, c - 1, 9);
                    // std::cout << t << std::endl;
                    t.enqueue(pos, pos, t.get(pos, pos) - 1);
                    ans = ans + suf + 2;
                }
                // std::cout << t << std::endl;
            } else {
                t.enqueue(c, c, num_next);
                ++ans;
            }
        } else {
            int num_next = num_was + delta;
            if (num_next > 9) {
                int suf = t.get9(1, c - 1);
                int pos = c - 1 - suf;
                t.enqueue(c, c, num_next - 10);
                // std::cout << "inc suf = " << suf << ", pos = " << pos << ", num_next
                    = " << num_next << std::endl;
                // std::cout << t << std::endl;
                if (suf == c - 1) {
                    t.enqueue(1, c - 1, 0);
                    ans = ans + suf + 1;
                } else {
                    t.enqueue(pos + 1, c - 1, 0);
                    t.enqueue(pos, pos, 1 + t.get(pos, pos));
                    ans = ans + suf + 2;
                }
                // std::cout << t << std::endl;
            } else {
                t.enqueue(c, c, num_next);
                ++ans;
            }
        }
        // std::cout << t << std::endl;
```

```
        std::cout << t.get(c, c) << ' ' << ans << '\n';

        // std::cout << t << "------" << std::endl;
    }
}
```

Положение команды

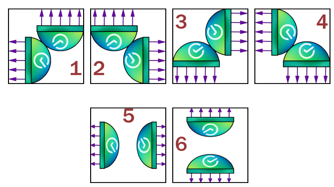
№	Участник Я	A 86/104	B 58/286	C 1/42	D 67/171	E 84/173	F 51/290	G 85/152	H 9/35	I 86/92	J 43/174	K 50/266	L 5/54	Очки	Штраф
74	MAI #2 Artemiev, Belousov, Inyutin	+2 00:28	-4 04:59	—	+1 02:40	+1 01:50	—	+ 01:13	—	+ 00:11	—	—	—	5	463

Задача B. Склады СберМаркет

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	2 секунды
Ограничение по памяти:	512 мегабайт

Поддержка работы крупного онлайн-магазина, доставляющего товары по всей стране, такого, как СберМаркет, требует организации огромных складов. Склады расположены под землёй и обслуживаются автоматическими тележками. На некоторых участках тележки должны подниматься на поверхность. Для этого устанавливаются специальные впуско-выпускные ворота.

На участке 1×1 километр допустимы следующие конфигурации установки ворот:



Для СберМаркета оборудуется новый центральный склад. Территория склада представляет собой огороженный прямоугольник w на h километров. В каждом секторе размера 1×1 километр **нужно** разместить одну из 6 конфигураций ворот.

Из стороны, где располагаются ворота, проведены фиолетовые стрелочки.

Автоматические тележки двигаются строго по прямой от одних ворот до других; в случае встречи двух тележек, едущих навстречу, они разъезжаются с помощью системы предотвращения столкновений и продолжают двигаться по той же прямой каждая в прежнем направлении.

Автоматические тележки могут выезжать из любых ворот и въезжать в любые ворота; также в конфигурациях 5 и 6 тележка может проехать между воротами по вертикали для конфигурации 5 и по горизонтали для конфигурации 6.

Если тележка попадает в ворота с обратной стороны или врезается в ограждение склада, происходит авария.

Требуется указать для каждого сектора склада, какая из 6 конфигураций ворот должна в нём располагаться, чтобы аварии были невозможны.

Формат входных данных

Первая строка входных данных содержит два целых числа w и h ($1 \leqslant w, h \leqslant 10^3$) — ширина и высота склада СберМаркета.

Формат выходных данных

Выведите h строк, в каждой строке w чисел в диапазоне от 1 до 6, разделённые пробелами — номера конфигураций ворот в соответствующих квадратах склада. Если ответов несколько, выведите любой.

Если расположить ворота без риска аварий невозможно, выведите -1.

Примеры

стандартный ввод	стандартный вывод
2 3	3 4 6 6 2 1
4 4	3 5 5 4 6 3 4 6 6 2 1 6 2 5 5 1

Идея

Решение задачи полностью конструктивное, полностью описано в программе. Самый сложный случай — при нечётных ширине или высоте складов СберМаркета. Асимптотика $O(w \cdot h)$.

Исходный код

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

using grid_t = std::vector< std::vector<int> >;

void draw_sq(grid_t & a, int x1, int y1, int x2, int y2) {
    a[x1][y1] = 3;
    a[x2][y1] = 2;
    a[x1][y2] = 4;
    a[x2][y2] = 1;
    for (int i = x1 + 1; i < x2; ++i) {
        a[i][y1] = 6;
        a[i][y2] = 6;
    }
    for (int i = y1 + 1; i < y2; ++i) {
        a[x1][i] = 5;
        a[x2][i] = 5;
    }
}

int rot[7] = {0, 1, 4, 3, 2, 6, 5};

grid_t transpose(const grid_t & gr) {
    int h = gr.size();
    int w = gr.back().size();
    grid_t res(w, std::vector<int>(h));
    for (int i = 0; i < w; ++i) {
        for (int j = 0; j < h; ++j) {
            res[i][j] = rot[gr[j][i]];
        }
    }
    return res;
}

int main() {
    std::ios::sync_with_stdio(false);
    std::cout.tie(0);
    std::cin.tie(0);

    int w, h;
    std::cin >> w >> h;
```

```
    grid_t ans(h, std::vector<int>(w));
    if (w == 1 or h == 1) {
        std::cout << "-1\n";
        return 0;
    }
    if (w == 2 or h == 2) {
        draw_sq(ans, 0, 0, h - 1, w - 1);
    } else if (w & 1) {
        if (h & 1) {
            if (h == 3 and w == 3) {
                std::cout << "-1\n";
                return 0;
            }
            bool flag = false;
            if (h == 3) {
                std::swap(h, w);
                flag = true;
                ans = transpose(ans);
            }
            ans[0][0] = 3;
            ans[0][1] = 5;
            for (int i = 2; i < w - 1; ++i) {
                ans[0][i] = (i & 1 ? 3 : 4);
            }
            ans[0].back() = 4;
            ans[1][0] = 2;
            ans[1][1] = 4;
            for (int i = 2; i < w; ++i) {
                ans[1][i] = 6;
            }
            for (int j = 2; j < h - 2; ++j) {
                ans[j][0] = (j & 1 ? 2 : 3);
                for (int i = 1; i < w - 1; ++i) {
                    ans[j][i] = 6;
                }
                ans[j].back() = (j & 1 ? 4 : 1);
            }
            ans[h - 2][w - 1] = 4;
            ans[h - 2][w - 2] = 2;
            for (int i = 0; i < w - 2; ++i) {
                ans[h - 2][i] = 6;
            }
            ans.back()[0] = 2;
            for (int i = 1; i < w - 2; ++i) {
                ans.back()[i] = (i & 1 ? 1 : 2);
            }
            ans.back()[w - 2] = 5;
            ans.back()[w - 1] = 1;
            if (flag) {
```

```
            std::swap(h, w);
            ans = transpose(ans);
        }
    } else {
        for (int i = 0; i < std::min(h / 2, w / 2); ++i) {
            draw_sq(ans, i, i + 1, h - 1 - i, w - 1 - i);
        }
        ans[0][0] = 3;
        ans[0][1] = 5;
        for (int i = 1; i < h - 1; i = i + 2) {
            ans[i][0] = 2;
            ans[i][1] = 4;
            ans[i + 1][0] = 3;
            ans[i + 1][1] = 1;
        }
        ans.back()[0] = 2;
        ans.back()[1] = 5;
    }
} else {
    if (h & 1) {
        for (int i = 0; i < std::min(h / 2, w / 2); ++i) {
            draw_sq(ans, i + 1, i, h - 1 - i, w - 1 - i);
        }
        ans[0][0] = 3;
        ans[1][0] = 6;
        for (int i = 1; i < w - 1; i = i + 2) {
            ans[0][i] = 4;
            ans[1][i] = 2;
            ans[0][i + 1] = 3;
            ans[1][i + 1] = 1;
        }
        ans[0].back() = 4;
        ans[1].back() = 6;
    } else {
        for (int i = 0; i < std::min(h / 2, w / 2); ++i) {
            draw_sq(ans, i, i, h - 1 - i, w - 1 - i);
        }
    }
}
for (int i = 0; i < h; ++i) {
    for (int j = 0; j < w; ++j) {
        std::cout << ans[i][j] << ' ';
    }
    std::cout << '\n';
}
}
```

Положение команды

№	Участник	A 4/53	B 24/221	C 10/52	D 29/44	E 1/2	F 37/158	G 5/8	H 4/133	I 30/122	J 41/196	K 59/240	L 41/129	Очки	Штраф
50	Moscow AI #2 (Maxim Inyutin, Egor Belousov, Dmitry Artemyev)	—	+4 02:54	—	—	—	—	—	—	-12 04:38	—	+4 01:53	—	2	448

Problem F. Fence

Input file: `standard input`
Output file: `standard output`
Time limit: 3 seconds
Memory limit: 1024 mebibytes

Bajtazar is widely believed to be the greatest scrooge in the whole borough. One can find many examples to support this claim, and the least important one of them is that his estate has not even got a fence. However, Bajtazar has recently found n old planks in his basement, so he decided to build at least a fragment of a hoarding.

He stacked the planks one over the other such that their consecutive lengths were $a_1, \ldots, a_n$. He took the first one, cut out a fragment of length b, and nailed it as the leftmost piece of the fence. Then, he cut out the next fragment of length b, and nailed it next to the previous one. He continued doing so until what was left in his hands was a piece of length c ($1 \le c \le b$). *Well, this one might seem a little bit too short, but it would be a pity if such a good plank went to waste*, he thought... and added it to the fence as well. He then took the second plank from the stack, then the next one, and repeated the whole procedure for each one of them.

When the job was done, Bajtazar looked at the result... and concluded that using those shorter pieces might really not have been the best idea. *This doesn't look like a cohesive design at all* – he thought – and decided to paint the whole fence white to give it at least a pretence of consistency. *Still*, it occurred to him suddenly, *if I only paint every other plank white, and leave the remaining ones brown, I could use (roughly) twice as little paint and yet the fence will still look like a well thought-out, coherent construction!* And so he only painted every second plank, starting from the leftmost one (apparently, the rumours of Bajtazar's meanness must have been somewhat exaggerated. He could have started from the second leftmost plank after all.).

However, in the evening, a frightening thought struck him: maybe if he had chosen a different value of b, the overall amount of paint used could have been smaller? Well, there's not much that can be done anymore, but just the thought of such an unnecessary wastage keeps Bajtazar awake – he needs to know how much paint he would have used if he had chosen to build a fence of any other possible height. Help him to find the answer so he can finally fall asleep peacefully (or not, depending on the result of your computation).

Input

The first line of input contains the number of test cases z ($1 \le z \le 5$).

The descriptions of the test cases follow.

The first line of every test case contains a single integer n ($1 \le n \le 1\,000\,000$) – the number of planks. Then follow n integers a_i ($1 \le a_i \le 1\,000\,000$, $\sum_{i=1}^{n} a_i \le 1\,000\,000$) – the lengths of consecutive planks.

Output

For each test case, output M lines, where M is the maximum of all values of a_i within this test case. The i-th line should contain a single integer f_i: the total length of planks which Bajtazar would have needed to paint white if he had chosen the fence height to be $b = i$.

Div A + B Contest 1 22.11.2021

Example

standard input	standard output
1 4 10 7 2 8	14 13 15 13 15 16 21 23 24 12

Note

For height $b = 4$, consecutive fence pickets would have heights:

4 4 2 4 3 2 4 4.

Bajtazar would have needed to paint the total length of $4+2+3+4$, so the answer in the 4-th line is 13.

For height $b = 5$, consecutive fence pickets would have heights:

5 5 5 2 2 5 3.

Bajtazar would have needed to paint the total length of $5+5+2+3$, so the answer in the 5-th line is 15.

Идея

Для решения задачи построим дерево отрезков на сумму чисел, стоящих на чётных и нечётных позициях для всего забора. Будет обновлять каждый фрагмент забора и выводить требуемую сумму. Так как суммарно обновлений в дереве будет не более $\sum a_i \leqslant 10^6$ и $n \leqslant 10^6$, то итоговая временная сложность решения $O(n \cdot \log n)$.

Исходный код

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

using pii = std::pair<int, int>;

class segment_tree_t {
private:
    struct item_t {
        int64_t even;
        int64_t odd;
        int n;

        item_t() = default;

        item_t(int64_t x, int64_t b) {
            if (x % b == 0) {
                n = x / b;
                odd = b * (n / 2);
                even = b * (n / 2);
                if (n & 1) {
                    even += b;
                }
            } else {
                n = (x + b - 1) / b;
                odd = b * (n / 2);
                even = b * (n / 2);
                if (n & 1) {
                    even = even + x % b;
                } else {
                    odd = odd - b + x % b;
                }
            }
            // std::cout << "x = " << x << ", b = " << b << ", odd = " << odd << ",
                even = " << even << ", n = " << n << std::endl;
        }

        friend item_t operator + (const item_t & lhs, const item_t & rhs) {
            item_t res;
            res.n = lhs.n + rhs.n;
            if (lhs.n & 1) {
```

```
            res.even = lhs.even + rhs.odd;
            res.odd = lhs.odd + rhs.even;
        } else {
            res.even = lhs.even + rhs.even;
            res.odd = lhs.odd + rhs.odd;
        }
        return res;
    }
};

std::vector<item_t> data;
int n;

public:
segment_tree_t(int _n) {
    n = _n;
    data.resize(4 * n);
}

void set(int p, int64_t x, int64_t b) {
    set(1, 1, n, p, x, b);
}

void set(int id, int l, int r, int p, int64_t x, int64_t b) {
    if (l >= r) {
        data[id] = item_t(x, b);
        return;
    }
    int m = (l + r) / 2;
    if (p <= m) {
        set(id * 2, l, m, p, x, b);
    } else {
        set(id * 2 + 1, m + 1, r, p, x, b);
    }
    data[id] = data[id * 2] + data[id * 2 + 1];
}

int64_t get(int l, int r) {
    item_t ans = get(1, 1, n, l, r);
    return ans.even;
}

item_t get(int id, int l, int r, int ql, int qr) {
    if (ql <= l and r <= qr) {
        return data[id];
    }
    int m = (l + r) / 2;
    if (qr <= m) {
        return get(id * 2, l, m, ql, qr);
```

```cpp
        }
        if (ql > m) {
            return get(id * 2 + 1, m + 1, r, ql, qr);
        }
        return get(id * 2, l, m, ql, qr) + get(id * 2 + 1, m + 1, r, ql, qr);
    }
};

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(0);

    int t;
    std::cin >> t;
    while (t--) {
        int n;
        std::cin >> n;
        using item_t = std::pair<int64_t, int>;
        std::list<item_t> lst;
        int64_t max_a = 0;
        for (int i = 0; i < n; ++i) {
            int64_t a;
            std::cin >> a;
            max_a = std::max(a, max_a);
            lst.push_back({a, i + 1});
        }
        segment_tree_t st(n);
        using iter = std::list<item_t>::iterator;
        for (int b = 1; b <= max_a; ++b) {
            for (iter it = lst.begin(); it != lst.end();) {
                iter next = it;
                ++next;
                item_t elem = *it;
                if (elem.first < b) {
                    lst.erase(it);
                } else {
                    st.set(elem.second, elem.first, b);
                }
                it = next;
            }
            std::cout << st.get(1, n) << '\n';
        }
    }
}
```

Положение команды

№	Участник	A1 18/67	B1 1/8	C1 44/131	D1 47/102	E1 8/32	F1 33/59	G1 12/30	H1 58/66	I1 15/97	J1 47/134	K1 42/278	L1 43/268	M1 0/0	Очки	Штраф
18	Moscow AI #2 (Maxim Inyutin, Egor Belousov, Dmitry Artemyev)	—	—	+2 6д. 20ч.	+1 6д. 17ч.	—	+ 6д. 21ч.	—	+ 4д. 18ч.	—	+1 4д. 16ч.	+2 4д. 18ч.	+2 6д. 19ч.	—	36	227718

Div A Contest 4: The Korean Contest 26.11.2021

Problem L. Trio

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	1024 mebibytes

Let A be any set of n natural numbers whose decimal representations consist of exactly four digits without 0 in any decimal place.

A *trio* is a set of three numbers $\{a, b, c\}$ chosen from A such that the following conditions are fulfilled simultaneously:

- The ones decimals of three numbers a, b, c are either all equal or all distinct.
- The tens decimals of three numbers a, b, c are either all equal or all distinct.
- The hundreds decimals of three numbers a, b, c are either all equal or all distinct.
- The thousands decimals of three numbers a, b, c are either all equal or all distinct.

For examples, $\{1425, 1113, 1354\}$ is a trio if the three numbers are members of A because the ones decimals of the three numbers are all distinct, their tens decimals are all distinct, their hundreds decimals are all distinct, and their thousands decimals are all equal. The set $\{1425, 1113, 5436\}$, however, is not a trio, even if A contains those three numbers.

Given a set A as input, write a program that computes and prints out the number of different trios.

Input

Your program is to read from standard input. The input starts with a line consisting of a single integer n ($1 \le n \le 2,000$) that represents the number of members in A. Each of the following n lines consists of a positive integer in decimal form that consists of exactly four digits without 0 in any decimal place. These n numbers are supposed to be all distinct and the members of the input set A.

Output

Your program is to write to standard output. Print exactly one line. The line should consists of a single integer that represents the number of different trios for the input set A.

standard input	standard output
6 1234 1235 1244 1233 7133 8133	1
9 1234 5678 9123 4567 8912 3456 7891 2345 6789	84

Идея

Используем *std::bitset* для эффективного хранения чисел. Для каждого разряда и для каждой цифры хранится битовое множество позиций чисел из множества A. Добавление и удаление чисел из такой структуры очень простое и выполняется фактически за $O(1)$. Пространственная сложность такого хранения $O(\frac{4 \cdot 9 \cdot n}{64}) \approx O(n)$.

Зафиксируем два числа тройки. Выберем только те цифры, которые удовлетворяют условию тройки, добавим в ответ количество индексов чисел. Операция *count* для *std::bitset* выполняется за $O(\frac{n}{64})$, поэтому временная сложность решения $O(\frac{n^3}{64})$.

Исходный код

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

using pii = std::pair<int, int>;
using bs = std::bitset<MAX_N>;

const int MAX_N = 2021;

int get_num(int x, int pos) {
    pos = 3 - pos;
    int res = x % 10;
    while (pos--) {
        x = x / 10;
        res = x % 10;
    }
    return res - 1;
}

std::vector< std::vector<bs> > dat(4, std::vector<bs>(9));

void add_num(const std::vector<int> & a, int i) {
    for (int j = 0; j < 4; ++j) {
        dat[j][get_num(a[i], j)][i] = 1;
    }
}

void rm_num(const std::vector<int> & a, int i) {
    for (int j = 0; j < 4; ++j) {
        dat[j][get_num(a[i], j)][i] = 0;
    }
}

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(0);
    int n;
```

```cpp
    std::cin >> n;
    std::vector<int> a(n);
    for (int i = 0; i < n; ++i) {
        std::cin >> a[i];
    }
    for (int i = 0; i < n; ++i) {
        add_num(a, i);
    }
    // for (int k = 0; k < 4; ++k) {
    // for (int j = 0; j < 9; ++j) {
    // std::cout << k << ' ' << j + 1 << ' ' << dat[k][j] << std::endl;
    // }
    // }
    int64_t ans = 0;
    for (int i = 0; i < n; ++i) {
        rm_num(a, i);
        for (int j = i + 1; j < n; ++j) {
            std::vector<bool> check(4);
            rm_num(a, j);
            // std::cout << a[i] << ' ' << a[j] << std::endl;
            for (int k = 0; k < 4; ++k) {
                check[k] = (get_num(a[i], k) == get_num(a[j], k));
            }
            bs cur;
            cur.flip();
            for (int k = 0; k < 4; ++k) {
                if (check[k]) {
                    cur &= dat[k][get_num(a[i], k)];
                } else {
                    bs pos;
                    for (int m = 1; m <= 9; ++m) {
                        int digit = m - 1;
                        if (digit != get_num(a[i], k) and digit != get_num(a[j], k)) {
                            pos |= dat[k][digit];
                        }
                    }
                    cur &= pos;
                }
                // std::cout << cur << std::endl;
            }
            // std::cout << "fin " << cur << std::endl;
            ans += cur.count();
        }
        for (int j = i + 1; j < n; ++j) {
            add_num(a, j);
        }
    }
    std::cout << ans << '\n';
}
```

Положение команды

№	Участник Я	A 8/57	B 53/97	C 53/54	D 46/100	E 15/29	F 39/144	G 27/98	H 0/2	I 0/27	J 9/80	K 11/39	L 38/89	Очки	Штраф
43	Moscow AI #2 (Maxim Inyutin, Egor Belousov, Dmitry Artemyev)	—	+ 00:05	+ 00:20	-1 04:35	—	-2 01:31	—	—	—	—	—	+3 02:35	3	241

Problem H. Crystalfly

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	256 megabytes

Paimon is catching crystalflies on a tree, which are a special kind of butterflies in Teyvat. A tree is a connected graph consisting of n vertices and $(n-1)$ undirected edges.

Pixiv ID: 93964680

There are initially a_i crystalflies on the i-th vertex. When Paimon reaches a vertex, she can catch all the remaining crystalflies on the vertex immediately. However, the crystalflies are timid. When Paimon reaches a vertex, all the crystalflies on the adjacent vertices will be disturbed. For the i-th vertex, if the crystalflies on the vertex are disturbed for the first time at the beginning of the t'-th second, they will disappear at the end of the $(t' + t_i)$-th second.

At the beginning of the 0-th second, Paimon reaches vertex 1 and stays there before the beginning of the 1-st second. Then at the beginning of each following second, she can choose one of the two operations:

- Move to one of the adjacent vertices of her current vertex and stay there before the beginning of the next second (if the crystalflies in the destination will disappear at the end of that second she can still catch them).
- Stay still in her current vertex before the beginning of the next second.

Calculate the maximum number of crystalflies Paimon can catch in $10^{10^{10^{10^{10}}}}$ seconds.

Input

There are multiple test cases. The first line of the input contains an integer T indicating the number of test cases. For each test case:

The first line contains an integer n ($1 \le n \le 10^5$) indicating the number of vertices.

The second line contains n integers $a_1, a_2, \cdots, a_n$ ($1 \le a_i \le 10^9$) where a_i is the number of crystalflies on the i-th vertex.

The third line contains n integers $t_1, t_2, \cdots, t_n$ ($1 \le t_i \le 3$) where t_i is the time before the crystalflies on the i-th vertex disappear after disturbed.

For the next $(n-1)$ lines, the i-th line contains two integers u_i and v_i ($1 \le u_i, v_i \le n$) indicating an edge connecting vertices u_i and v_i in the tree.

It's guaranteed that the sum of n of all the test cases will not exceed 10^6.

Output

For each test case output one line containing one integer indicating the maximum number of crystalflies Paimon can catch.

Example

standard input	standard output
2 5 1 10 100 1000 10000 1 2 1 1 1 1 2 1 3 2 4 2 5 5 1 10 100 1000 10000 1 3 1 1 1 1 2 1 3 2 4 2 5	10101 10111

Note

For the first sample test case, follow the strategy below.

- During the 0-th second
  - Paimon arrives at vertex 1;
  - Paimon catches 1 crystalfly;
  - Crystalflies in vertices 2 and 3 are disturbed.
- During the 1-st second
  - Paimon arrives at vertex 3;
  - Paimon catches 100 crystalflies.
- During the 2-nd second
  - Paimon arrives at vertex 1;
  - Crystalflies in vertex 2 disappears.
- During the 3-rd second
  - Paimon arrives at vertex 2;
  - Crystalflies in vertices 4 and 5 are disturbed.
- During the 4-th second
  - Paimon arrives at vertex 5;
  - Paimon catches 10000 crystalflies;
  - Crystalflies in vertex 4 disappears.

Идея

Решим сперва чуть более простую задачу, в которой бабочки улетают через 1 или 2 секунды. Возьмем вершину 1 за корень дерева. Пусть мы зашли в какую-то вершину и испугали всех смежных бабочек, тогда в вершины, где бабочки улетают через 1 или 2 секунды после испуга, мы, если хотим собрать бабочек, должны заходить сразу же, так как, если мы зайдем в другую вершину дерева, то нам придется потратить по крайней мере 3 секунды. Тогда пусть dp_i обозначает максимальное кол-во бабочек, которые мы соберем, если начнем собирать в вершине i. Пересчет динамики таков: $dp_i = max_j\{\sum_{c \neq j}(dp_c - a_c) + dp_j\}$ где j и c являются детьми i в дереве, с корнем в 1. Для того, чтобы пересчитать динамику в вершинах с 3 секундами введем еще значение $dpvis_i$ — сколько бабочек мы соберем, начиная с вершины i, если мы не возьмем бабочек в вершине i, а также во всех детях i, но возьмем бабочек во всех детях детей i. Пересчет достаточно прост: $dpvis_i = \sum_c(dp[c] - a[c])$, где c — ребенок i. В конце концов пересчитаем динамику для i дополнительно по вершинам с испугом в 3 секунды: $dp_i = max_{j,k:j \neq k}\{\sum_{c \neq j, c \neq k}(dp_c - a_c) + dpvis_k + a_k + dp_j\}$, j вершина с испугом в 3 секунды. Итого, используя поиск в глубину и предпосчитав суммы через префиксные суммы, можно решить данную задачу за $O(n)$

Исходный код

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

using namespace std;

#define flush cout.flush

using ll = long long;
using ull = unsigned long long;
using ld = long double;
using pl = pair<ll, ll>;
const ll INF = 1e9 + 7;
const ll mod = 1e9 + 7;
const ll mod2 = 998244353;
const ld eps = 1e-9;
const ld PI = acos(-1);

vector<vector<ll>> g;
vector<ll> a;
vector<ll> t;
vector<ll> dp;
vector<ll> dp_vis;

void dfs(ll v, ll p) {
    dp[v] = 0;
```

```
    dp_vis[v] = 0;
    for (auto to : g[v]) {
        if (to == p)continue;
        dfs(to, v);
    }
    ll s = 0;
    for (auto to : g[v]) {
        if (to == p)continue;
        dp_vis[v] += dp[to] - a[to];
        s += dp[to] - a[to];
    }
    for (auto to : g[v]) {
        if (to == p)continue;
        dp[v] = max(dp[v], s - (dp[to] - a[to]) + dp[to]);
    }
    ll m = g[v].size();
    vector<ll> d(m, 0);
    vector<ll> suff(m, 0);
    vector<ll> pref(m, 0);
    for (ll i = 0; i < m; ++i) {
        if (g[v][i] == p)continue;
        ll to = g[v][i];
        d[i] = dp_vis[to] + a[to] - (dp[to] - a[to]);
    }
    for (ll i = 0; i < m; ++i) {
        pref[i] = max((i - 1 >= 0 ? pref[i - 1] : 0), d[i]);
    }
    for (ll i = m - 1; i >= 0; --i) {
        suff[i] = max((i + 1 < m ? suff[i + 1] : 0), d[i]);
    }
    for (ll i = 0; i < m; ++i) {
        if (g[v][i] == p)continue;
        ll to = g[v][i];
        if (t[to] != 3)continue;
        dp[v] = max(dp[v],
                    s - (dp[to] - a[to]) + dp[to] + max((i - 1 >= 0 ? pref[i - 1] : 0),
                        (i + 1 < m ? suff[i + 1] : 0)));
    }
    dp[v] += a[v];
}

void solve() {
    g.clear();
    dp.clear();
    dp_vis.clear();
    ll n;
    cin >> n;
    dp.resize(n);
    dp_vis.resize(n);
```

```cpp
    g.resize(n);
    a.resize(n);
    t.resize(n);
    for (ll &i : a)cin >> i;
    for (ll &i : t)cin >> i;
    for (ll i = 0; i < n - 1; ++i) {
        ll u, v;
        cin >> u >> v;
        u--;
        v--;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    dfs(0, 0);
    cout << dp[0] << "\n";
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    ll t;
    cin >> t;
    while (t--)solve();
    return 0;
}
```

Положение команды

№	Участник Я	A 62/74	B 5/14	C 58/142	D 37/107	E 24/92	F 5/25	G 21/85	H 50/105	I 45/126	J 45/179	K 1/3	L 11/33	M 59/119	Очки	Штраф
51	MAI #2 Artemiev, Belousov, Inyutin	+ 00:15	—	+5 01:10	—	—	—	—	+2 02:43	—	-19 04:58	—	—	+1 00:32	4	442

Problem E. Construct The Integer

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	512 megabytes

For any positive integer y we define $A(y)$ as the set of all positive integers that are anagrams of y. Formally, we consider the sequence of all digits of y in decimal representation without leading zeroes, then we apply all possible permutations to this sequence. For any permutation we throw away leading zeroes and assemble digits back to an integer. All integers that can be obtained that way belong to $A(y)$. For example, for 2021 set $A(2021)$ consists of integers 122, 212, 221, 1220, 2120, 2210, 1202, 2102, 2201, 1022, 2012 and 2021. Note, that set $A(y)$ always contains y, thus it is never empty.

For any positive integer x we define $S(x)$ as the set of all positive integers z such that greatest common divisor of all integers in $A(z)$ equals x.

You are given the integer n, find the **minimum** integer z in $S(x)$, or determine that $S(x)$ is empty.

Input

The first line of the input contains one integer t ($1 \le t \le 50$) — the number of the test cases.

Each test case data consists of one integer n ($1 \le n \le 10^{18}$).

Output

For each test case, print one integer on a separate line. If the corresponding set $S(n)$ is empty, print `-1`. Otherwise, print minimum element of $S(n)$.

Example

standard input	standard output
2 12 2021	48 -1

Идея

Предпосчитаем gcd для первых 10000 чисел. Видно, что для больших чисел почти всегда gcd его анаграм равен 1. Но есть и исключения, которые мы обработаем отдельно. Наиболее интересны числа, состоящие из чётных цифр и сумма цифр которых делится на три. При перестановке цифр в таком числе делимость на 3 и 2 не изменится. Сложность решения $O(t + k)$, где k — константа предподсчёта $k \approx 10^4 \cdot 4! \cdot 4 \cdot \log 10^4$.

Исходный код

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

using pii = std::pair<int, int>;

std::map<int64_t, std::set<int64_t> > mp;

std::set<int64_t> anagrams(int64_t x) {
    std::string s = std::to_string(x);
    std::set<int64_t> res;
    std::sort(s.begin(), s.end());
    do {
        int64_t num = std::stoll(s);
        res.insert(num);
    } while (std::next_permutation(s.begin(), s.end()));
    return res;
}

void process_num(std::string s) {
    if (s.size() > 18) {
        return;
    } else {
        int64_t number = std::stoll(s);
        std::set<int64_t> nums = anagrams(number);
        int64_t gcd_res = *nums.begin();
        for (int64_t elem : nums) {
            gcd_res = std::__gcd(elem, gcd_res);
        }
        mp[gcd_res].insert(number);
    }
}

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(0);
    for (int i = 0; i <= 9; ++i) {
        std::string num;
        for (int j = 1; j <= 18; ++j) {
            num.push_back('0' + i);
```

```cpp
            int64_t nnum = std::stoll(num);
            mp[nnum].insert(nnum);
        }
    }
    for (int i = 0; i <= 9; ++i) {
        std::string num;
        num.push_back('0' + i);
        num.push_back('0');
        for (int j = 1; j <= 18; ++j) {
            num.push_back('0' + i);
            process_num(num);
        }
    }
    for (int i = 1; i <= 9; ++i) {
        std::string num;
        for (int j = 1; j <= 18; ++j) {
            num.push_back('0' + i);
            std::string magic = num;
            for (int k = 1; k <= 18; ++k) {
                magic.push_back('8');
                process_num(magic);
            }
        }
    }
    for (int64_t i = 1; i <= 1e4; ++i) {
        process_num(std::to_string(i));
    }
    // for (auto elem : mp) {
    // std::cout << "gcd = " << elem.first << ", ";
    // std::cout << *elem.second.begin();
    // // for (int64_t num : elem.second) {
    // // std::cout << num << ' ';
    // // }
    // std::cout << std::endl;
    // }
    int t;
    std::cin >> t;
    while (t--) {
        int64_t n;
        std::cin >> n;
        if (!mp.count(n)) {
            std::cout << "-1\n";
        } else {
            std::cout << *mp[n].begin() << '\n';
        }
    }
}
```

Положение команды

№	Участник Я	A 278/352	B 0/8	C 11/27	D 134/681	E 87/849	F 239/511	G 175/322	H 154/972	I 12/623	J 51/109	K 3/8	L 37/104	M 71/345	Очки	Штраф
72	Moscow AI: MAI #2 (Dmitry Artemiev,Egor Belousov,Maksim Iniutin)	+ 00:02	—	—	+1 02:25	+3 02:54	+ 00:15	+ 01:41	+1 00:26	—	—	—	—	—	7	641