Росатом2024_09-10_Тур_1

Задача 10-1

Условие: "Задача про машинку"

Недавно Гриша получил в подарок интересную игрушку – машинку. Он решил провести для неё испытание в виде гоночной трассы. Трасса строится из квадратных клеток и образует замкнутый маршрут на плоскости.

Гриша хочет прогнать свою машинку по этой трассе N полных кругов. Он задумался о количестве поворотов, которые нужно будет выполнить его игрушке для успешного завершения заезда. Каждый поворот на трассе представляет собой угол в 90 градусов.

Помогите Грише рассчитать, сколько раз его машинка должна повернуть, чтобы успешно проехать заданное число кругов.

Входные данные

В первой строке дано единственное число N $(1 \le N \le 10^9)$ - количество кругов.

В последующих строках даны координаты x_i, y_i $(-10^5 \le x_i, y_i \le 10^5)$ клеток гоночного круга. В каждой строке даны координаты клетки, которую машинка посетит следующей. Координаты необходимо считывать, пока на ввод не подадут стартовую клетку круга.

Гарантируется, что круг содержит не более 10^6 клеток.

Выходные данные

Выведите единственное число - количество поворотов.

Решение

```
def calcDelta(c1, c2):
    return (c1[0] - c2[0], c1[1] - c2[1])

n = int(input())
lap = []
x, y = map(int, input().split())
lap.append((x, y))
while True:
    x, y = map(int, input().split())
    if (x, y) == lap[0]:
        break
    lap.append((x, y))

ansPerLap = 0
lastDelta = (0, 0)
for i in range(1, len(lap)):
    delta = calcDelta(lap[i - 1], lap[i])
    if delta != lastDelta:
        ansPerLap += 1
        lastDelta = delta

# last corner
if (calcDelta(lap[-1], lap[0]) != lastDelta):
    ansPerLap += 1

print(n * ansPerLap)
```

Задача 10-2

Условие: "Задача про солдатиков"

У Вани есть большая коллекция пластиковых солдатиков, и он ей очень гордится. Каждый солдатик в коллекции Вани особенный: у кого-то есть военный вертолёт, у кого-то — пушка, кто-то выше, кто-то ниже. Ваня даже придумал для каждого своего бойца особое имя.

Однажды Ваня решил поиграть со своими пластиковыми солдатиками. Игра началась с того, что Ваня выстроил всех солдатиков в ряд по росту: от самого низкого, который занял первое место в линии, до самого высокого. Но внезапно Ваня решил изменить игру: теперь бойцы должны быть перестроены в алфавитном порядке по их именам!

После этого Ваня задумался, как определить, какой солдатик в результате перестроения переместится на наибольшее количество позиций. Помогите Ване найти ответ на этот вопрос.

Входные данные

В первой строке задано число N - количество солдатиков в коллекции.

В последующих N строках через пробел заданы: число $height_i$ $(1 \le height_i \le 10^9)$ и строка $name_i$ $(1 \le |name_i| \le 10)$ - рост и имя солдатика соответственно.

Гарантируется, что все солдатики различного роста и что их имена различны.

Выходные данные

Выведите единственной число - максимальное значение разницы позиции солдатика между первым и вторым построением.

Решение

```
n = int(input())
pos1 = [0] * n
pos2 = [0] * n

v1 = [0] * n
v2 = [0] * n
for i in range(n):
    hight, name = input().split()
    v1[i] = (int(hight), i)
    v2[i] = (name, i)

v1.sort()
v2.sort()

for i in range(n):
    pos1[v1[i][1]] = i
    pos2[v2[i][1]] = i

ans = 0
for i in range(n):
    ans = max(ans, abs(pos1[i] - pos2[i]))

print(ans)
```

Задача 10-3

Условие: "Задача про верблюда"

Женя очень любит решать шахматные задачки, особенно про коней. Но на этот раз вместо коня ему подсунули верблюда. Верблюд - особенная шахматная фигура, которая может за один ход сместиться на одну клетку вправо или на одну клетку вверх, причём первая координата клетки всегда должна быть больше, чем вторая, или равна ей $(X \geq Y)$.

У Жени есть шахматная доска размера N на N клеток (нижняя левая клетка имеет координаты $(1, 1)$, а верхняя правая - (N, N)). В клетке $(1, 1)$ стоит верблюд, помогите Жене найти число способов, которыми верблюд может добраться до клетки с координатами (x, y). Так как это число может быть очень большим, выведите ответ по модулю 1000000007.

Входные данные

В первой строке дано число N $(1 \leq N \leq 3000)$ - размер шахматной доски.

Во второй строке через пробел даны координаты клетки x и y $(1 \leq x, y \leq N)$.

Выходные данные

Выведите единственное число - количество способов попасть верблюду из точки $(0, 0)$ в точку (x, y).

Решение

```
n = int(input())
X, Y = map(int, input().split())
dp = [[0] * n for i in range(n)]
for i in range(n):
    dp[i][0] = 1

MOD = int(1e9 + 7)
for x in range(1, n):
    for y in range(1, x + 1):
        dp[x][y] = (dp[x-1][y] + dp[x][y-1]) % MOD

print(dp[X - 1][Y - 1])
```

Задача 4

Условие "Основа основ"

В сердце каждой современной вычислительной системы лежит принцип мгновенной и безупречной синхронизации. И нет более важного компонента в этом процессе, чем генератор синхронизирующего сигнала, своевременно координирующий работу всех последовательностных частей ЭВМ.

Генератор синхросигнала обеспечивает слаженную работу всех систем и устройств, управляя моментами считывания, записи, передачи и преобразования данных.

В нашем идеализированном и упрощенном варианте ЭВМ, мы сталкиваемся с трудностью: синхросигнал от генератора должен доходить до каждого компонента одновременно, чтобы гарантировать, что все процессы в системе выполняются в пределах точно отрегулированных временных рамок и без каких-либо задержек.

Несинхронизированные компоненты могут привести к катастрофическим ошибкам в вычислениях, потере данных и сбою операций — чего мы ни в коем случае не допустим при проектировании нашей совершенной ЭВМ! Именно поэтому исключительно важно, чтобы длина каждого провода от генератора до схемы была **одинаковой**.

Схема ЭВМ представлена с некоторыми упрощениями:

- ЭВМ представляет собой прямоугольник в клетку.
- Одна из клеток - выход генератора синхронизирующей частоты.
- Все компоненты ЭВМ - прямоугольники, чьи стороны параллельны сторонам схемы ЭВМ.
- Провод с синхросигналом проходит через некоторую последовательность клеток, что каждая следующая клетка имеет хотя бы общую сторону с предыдущей (или совпадает с ней).
- В каждой клетке провод проходит через центр, возможно делая там поворот на 90 или 180 градусов.
- Провод может проходить по некоторым клеткам более одного раза.
- На пути провода любые 2 соседние клетки отличаются друг от друга. Например, клетка (1, 1) не может быть после клетки (1, 1). А последовательность клеток (1, 1), (1, 2), (1, 1), (1, 2) допустима.
- За границы схемы провода выходить не могут.

Ответом на задачу будет единственное число - минимальная сумма длин проводов, проведенных от генератора до компонентов ЭВМ.

Входные данные

В первой строке вводятся два числа: N, M - размеры схемы в клетках ($N \times M$). ($1 \le N, M \le 2000$).
Далее следует N строк, в каждой из которых по M символов $a_{i,j}$. ($a_{i,j} \in$ «.»,«X»,«I»,«B»).
Точка обозначает пустую клетку.
Символ "X" - позицию источника на схеме. Гарантируется, что ровно один символ "X" встретится на схеме.
Символ "B" обозначает, что клетка занята компонентом ЭВМ. Гарантируется, что каждый элемент схемы - прямоугольник без полостей внутри.
Элементы схемы не касаются друг друга, т.е. среди их клеток нет соседних по стороне.
Символ "I" обозначает выход схемы, куда надо подвести провод с синхросигналом. Гарантируется, что в каждом элементе схемы ровно одна клетка помечена как "I" и расположена на границе компонента, возможно в углу.
Каждый провод начинается в клетке с символом "X", заканчивается в клетке с символом "I".

Выходные данные

Выведите одно число - суммарную длину всех проводов с синхросигналом, если возможно сделать так, чтобы до каждого компонента ЭВМ сигнал доходил одновременно.
Выведите -1, если так сделать невозможно.

Пример

Группа - ($1 \le N, N \le 50$), все компоненты состоят из одной клетки "I". Также гарантируется, что до каждого компонента схемы существует путь по клеткам с длиной $|X_x - I_x| + |X_y - I_y|$, т.е. элементы ЭВМ не перекрывают все пути минимальной длины от источника до каждого элемента схемы. (20 баллов)
Группа - ($1 \le N, M \le 50$) (45 баллов)
Группа - полные ограничения (35 баллов)

Пояснения к 1 тесту.
Для подключения схемы в левом верхнем углу необходим провод длины хотя бы 5. Также два других компонента можно подключить проводом длины 5.
Для подключения компонента справа в середине (строка, считая сверху; столбец, считая слева) (5, 3)->(5, 4)->(5,5)->(4, 5)->(4, 4)->(3, 4) - один из примеров расположения провода.
А для подключения схемы слева снизу провод можно уложить так (5, 3)->(4, 3)->(4, 2)->(4, 3)->(4, 2)->(4, 1)

Решение

```
#include <iostream>
#include <vector>
#include <cassert>
#include <queue>
#include <set>

using namespace std;

void solve() {
    int n, m;
    cin >> n >> m;
    vector<vector<char>> pole(n, vector<char>(m));
    for (int x = 0; x < n; ++x) {
        for (int y = 0; y < m; ++y) {
            cin >> pole[x][y];
        }
    }

    // Search X coordinates
    int xX = -1, yX = -1;
    for (int x = 0; x < n; ++x) {
        for (int y = 0; y < m; ++y) {
            if (pole[x][y] == 'X') {
                if (xX == -1) {
                    xX = x;
                    yX = y;
                } else {
                    assert(false);
                }
            }
        }
    }
    assert(xX != -1);


    vector<vector<int>> used_check(n, vector<int>(m));
    auto isCorn = [&pole](int x, int y) {
```

```
    return pole[x][y] == 'B' || pole[x][y] == 'I';
};
auto valid = [&n, &m](int x, int y) {
    return (
        0 <= x && x < n &&
        0 <= y && y < m
    );
};
vector<pair<int, int>> targets;
for (int x_corm = 0; x_corm < n; ++x_corm) {
    for (int y_corn = 0; y_corn < m; ++y_corn) {
        if (isCorn(x_corm, y_corn)) {
            if (used_check[x_corm][y_corn]) {
                continue;
            }
            // New chema detected
            int wX = 0;
            while (x_corm + wX < n && isCorn(x_corm + wX, y_corn)) {
                ++wX;
            }
            int wY = 0;
            while (x_corm + wY < m && isCorn(x_corm, y_corn + wY)) {
                ++wY;
            }
            int prevTargetSize = targets.size();
            for (int dx = 0; dx < wX; ++dx) {
                for (int dy = 0; dy < wY; ++dy) {
                    int x_now = x_corm + dx;
                    int y_now = y_corn + dy;
                    assert(isCorn(x_now, y_now));
                    used_check[x_now][y_now] = 1;
                    if (pole[x_now][y_now] == 'I') {
                        targets.push_back({x_now, y_now});
                    }
                }
            }
            //validate around empty
            for (int dx : {-1, 0, wX - 1, wX}) {
                for (int dy : {-1, 0, wX - 1, wY}) {
                    int cnt = 0;
                    if (dx == -1 || dx == wX) {
                        ++cnt;
                    }
                    if (dy == -1 || dy == wY) {
                        ++cnt;
                    }
                    if (cnt != 1) {
                        continue;
                    }
                    int x_now = x_corm + dx;
                    int y_now = y_corn + dy;
                    if (valid(x_now, y_now)) {
                        assert(used_check[x_now][y_now] == 0);
                    }
                }
            }
            assert(prevTargetSize + 1 == targets.size());
        }
    }
}

queue<pair<int, int>> q;
vector<pair<int, int>> d_coord = {{-1, 0}, {1, 0}, {0, 1}, {0, -1}};
const int INF = 1e9 + 7;
vector<vector<int>> dist(n, vector<int>(m, INF));
q.push({xX, yX});
dist[xX][yX] = 0;
while (q.size()) {
    auto [x_tmp, y_tmp] = q.front();
    q.pop();
    for (auto [dx, dy] : d_coord) {
        int x_now = x_tmp + dx;
        int y_now = y_tmp + dy;
        if (valid(x_now, y_now) && dist[x_now][y_now] == INF) {
            if (pole[x_now][y_now] == '.' || pole[x_now][y_now] == 'I') {
                dist[x_now][y_now] = dist[x_tmp][y_tmp] + 1;
                if (pole[x_now][y_now] == '.') {
                    q.push({x_now, y_now});
                }
            }
        }
    }
}
int max_dist = 0;
set<int> osts;
int ans = 0;
for (auto [x_fin, y_fin] : targets) {
    if (dist[x_fin][y_fin] == INF) {
        ans = -1;
    } else {
        max_dist = max(max_dist, dist[x_fin][y_fin]);
        osts.insert(dist[x_fin][y_fin] % 2);
    }
}
if (ans == -1 || osts.size() == 2) {
    cout << -1 << endl;
    return;
} else {
    cout << targets.size() * max_dist << endl;
```

```
        return;
    }
}

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

Задача 5

Условие

В далекой галактике живет юный исследователь Вовочка. В один прекрасный день, при изучении новой планеты, он наткнулся на загадочную древнюю цивилизацию. Было обнаружено, что эта цивилизация обладала способностью оставлять послания, состоящие из трех камней: `1, 2, 3`.

Послания можно разделить на два типа: "Правый взлет" (1-2-3) и "Левый взлет" (3-2-1). Вовочка также располагает специальной аппаратурой, которая позволяет ему изменять значение определенного камня, что помогает ему уточнить информацию или изучить новые варианты посланий.

Вовочке, как истинному исследователю, интересно, сколько таких посланий содержится на конкретном участке местности, который они обозначали в виде последовательного числового массива.

Существует два типа запросов.

- Запрос первого типа имеет вид `(? l r t)`. Необходимо искать количество "Правых взлетов" в отрезке [l, r], если t = 1. Если же t = 2, то запрос будет искать общее число взлетов: Левых и Правых. Одно число может быть частью двух взлетов одновременно. Например 1 2 3 2 1 содержит 2 взлета, где 3 - это последний элемент правого взлета и первый элемент левого.
- Запрос второго типа имеет вид `(! pos val)`. Необходимо изменить значение камня на позиции *pos* на заданное значение *val*.

  Помогите Вовочке расшифровать послания древних! Напишите программу, которая выполнит все запросы первого типа и выведет результат. Вывод каждого нового запроса должен начинаться с новой строки.

Входные данные

Первая строка содержит натуральное число N $(3 \le N \le 10^5)$ - длина массива.
Вторая строка содержит N натуральных чисел a_i $(1 \le a_i \le 3)$ - элементы исходного массива.
Третья строка содержит натуральное число Q $(1 \le Q \le 10^5)$ - число запросов.
Следующие Q строк описывают запросы в одном из 2 форматов.
Запрос первого типа имеет вид `? l r t` $(1 \le l < r \le N), (t \in \{1, 2\})$.
Запрос второго типа имеет вид `! pos val` $(1 \le pos \le N)$, $(1 \le val \le 3)$.

Гарантируется, что будет хотя бы один запрос первого типа.

Выходные данные

На каждый запрос одно вида выведите одно число - ответ на запрос. Каждый новый ответ должен начинаться с новой строки.

Пример

Группа 1 - гарантируется, что $(N, Q \le 1000)$. За эту группу вы получите 25 баллов.
Группа 2 - нет запросов второго типа. За эту группу вы получите 35 баллов. Будет тестироваться только при прохождении группы 1.
Группа 3 - полные ограничения. За эту группу вы получите 40 баллов. Будет тестироваться только при прохождении группы 2.

Решение

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

using namespace std;

class TSegtree {
public:
    TSegtree(int n) {
        MAXN = 1;
        while (MAXN < n) {
            MAXN += MAXN;
        }
        values.resize(MAXN + MAXN);
    }
    void set_value(int pos, int val) {
        set_value(pos, val, 1, 0, MAXN);
    }
    int get_sum(int le, int re) {
        return get_sum(le, re, 1, 0, MAXN);
    }

private:
```

```
    void set_value(int pos, int val, int v, int lb, int rb) {
        if (lb + 1 == rb) {
            values[v] = val;
            return;
        }
        int mid = (lb + rb) / 2;
        if (pos < mid) {
            set_value(pos, val, v + v, lb, mid);
        } else {
            set_value(pos, val, v + v + 1, mid, rb);
        }
        values[v] = values[v + v] + values[v + v + 1];
    }

    int get_sum(int le, int re, int v, int lb, int rb) {
        if (le <= lb && rb <= re) {
            return values[v];
        }
        if (re <= lb || rb <= le) {
            return 0;
        }
        int mid = (lb + rb) / 2;
        int left_ans = get_sum(le, re, v + v, lb, mid);
        int right_ans = get_sum(le, re, v + v + 1, mid, rb);
        return left_ans + right_ans;
    }

    int MAXN;
    vector<int> values;
};

int normalise(int val, int val_min, int val_max) {
    if (val <= val_min) {
        return val_min;
    }
    if (val >= val_max) {
        return val_max;
    }
    return val;
}

int get_sum(int l, int r, int n, TSegtree& segtree) {
    l = normalise(l, 0, n);
    r = normalise(r, 0, n);
    if (l >= r) {
        return 0;
    }
    int res = segtree.get_sum(l, r);
    return res;
}

bool my_compare(int pos_to_check, const vector<int>& numbers, int value_equal_to) {
    if (0 <= pos_to_check && pos_to_check < numbers.size()) {
        return numbers[pos_to_check] != value_equal_to;
    }
    return true;
}

void count_left(int pos, const vector<int>& numbers, TSegtree& segtree) {
    if (pos < 0 || pos >= numbers.size()) {
        return;
    }
    segtree.set_value(pos, 0);
    if (my_compare(pos + 2, numbers, 1) || my_compare(pos + 1, numbers, 2) || my_compare(pos, numbers, 3)) {
        return;
    }
    segtree.set_value(pos, 1);
}

void count_right(int pos, const vector<int>& numbers, TSegtree& segtree) {
    if (pos < 0 || pos >= numbers.size()) {
        return;
    }
    segtree.set_value(pos, 0);
    if (my_compare(pos - 2, numbers, 1) || my_compare(pos - 1, numbers, 2) || my_compare(pos, numbers, 3)) {
        return;
    }
    segtree.set_value(pos, 1);
}

void solve() {
    int n;
    cin >> n;
    vector<int> numbers(n);
    for (int pos_read = 0; pos_read < n; ++pos_read) {
        cin >> numbers[pos_read];
    }
    TSegtree left_side(n), right_side(n);
    for (int pos_init = 0; pos_init < n; ++pos_init) {
        count_left(pos_init, numbers, left_side);
        count_right(pos_init, numbers, right_side);
    }
    int q;
    cin >> q;
    for (int ask_id = 0; ask_id < q; ++ask_id) {
        char type;
        cin >> type;
        if (type == '!') {
            int pos, val;
```

```
            cin >> pos >> val;
            --pos;
            numbers[pos] = val;
            for (int pos_left_update = pos - 2; pos_left_update <= pos; ++pos_left_update) {
                count_left(pos_left_update, numbers, left_side);
            }
            for (int pos_right_update = pos; pos_right_update <= pos + 2; ++pos_right_update) {
                count_right(pos_right_update, numbers, right_side);
            }
        } else {
            int le, re, ask_type;
            cin >> le >> re >> ask_type;
            --le;
            int ans_right = get_sum(le + 2, re, n, right_side);
            int ans_left = get_sum(le, re - 2, n, left_side);
            int ans = ans_right;
            if (ask_type == 2) {
                ans += ans_left;
            }
            cout << ans << '\n';
        }
    }
}

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