1 A. Разминка для рук

1.1 Условие

Одиннадцатиклассник Василий умеет очень хорошо программировать. Во время изучения стереометрии ему не раз приходилось вычислять скалярное произведение двух векторов, зная их координаты. $((\{x1, y1, z1\}, \{x2, y2, z2\}) = x1 * x2 + y1 * y2 + z1 * z2)$
Аналогичная задача возникала в планиметрии, если использовать координатный метод.
У Василия с устным счетом очень плохо, поэтому он решил написать программу, которая будет вычислять скалярное произведение двух векторов по их координатам.

Василий узнал у старшего брата, что скалярное произведение двух векторов $a = \{a1, a2, ..., an\}$ и $b = \{b1, b2, ..., bn\}$ в $N-$ пространстве равняется $a1 * b1 + a2 * b2 + ... + an * bn$. Он смог написать программу для этой задачи. Сможете ли вы?

1.2 Входные данные

В первой строке входных данных задано одно целое число: $2 \leq n \leq 10^5$.

Во второй строке входных заданы через пробел n целых чисел — координаты вектора $\vec{a}(-100 \leq a_i \leq 100)$.

В третьей строке входных заданы через пробел n целых чисел — координаты вектора $\vec{b}(-100 \leq b_i \leq 100)$.

1.3 Вывод

В единственной строке выходных данных выведите одно целое число — результат скалярного произведения $\vec{a} \cdot \vec{b}$.

1.4 Пример входных данных

Sample Input:

5
1 2 3 4 5
5 4 3 2 1

Sample Output:

35

1.5 Решение

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

using namespace std;

int main()
{
    size_t n;
    cin >> n;

    vector<int64_t> a(n), b(n);

    for (size_t i = 0; i < n; ++i)
        cin >> a[i];

    for (size_t i = 0; i < n; ++i)
        cin >> b[i];

    int64_t ans = 0;
    for (size_t i = 0; i < n; ++i)
        ans += a[i] * b[i];

    cout << ans << endl;

    return 0;
}
```

2 B. Тяжкая подготовка

2.1 Условие

Поликарп сейчас готовится к экзамену по информатике, а, конкретно, он изучает инверсии. Инверсией в массиве A называется такая пара индексов i и j, что $i < j$ и при этом $a_i > a_j$.

Поликарп уже научился считать количество инверсий в массиве, но на экзамене есть задача посложнее: посчитать количество инверсий с заданной суммой. Пока что эта задачка слишком сложная для Поликарпа, поэтому он просит вашей помощи.

Помогите Поликарпу посчитать количество таких пар (i, j), что $i < j$, $a_i > a_j$ и $a_i + a_j = M$.

2.2 Входные данные

В первой строке входных данных через пробел заданы два целых числа: $1 < N \leq 10^5, 0 \leq M \leq 1000$, где N - размер массива.

Во второй строке входных данных через пробел заданы N целых чисел: $0 \leq a_i \leq 100$.

2.3 Вывод

В единственной строке выходных данных выведите одно целое число — количество искомых инверсий.

2.4 Пример входных данных

Sample Input:

10 11
1 2 3 4 5 6 7 8 9 10

Sample Output:

0

Sample Input:
5 3
3 2 2 5 1

Sample Output:
2

2.5 Решение

Давайте с помощью словаря для каждого числа от 0 до 100 подсчитывать сколько таких чисел слева от текущей позиции. Тогда количество инверсий с суммой m для данного числа x - количество чисел, равных (m-x).
Возьмем их количество из словаря и добавим к ответу.
И не забудем обновить словарь, добавив в него информацию об x.

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

using namespace std;

int main()
{
    long long n, sum;
    long long answer = 0;
    cin >> n >> sum;
    unordered_map<long long, int> cnt;
    int tmp;
    for (int i = 0; i < n; ++i)
    {
        cin >> tmp;
        cnt[tmp]++;
        const long long rest = sum - tmp;
        const bool isInversion = rest > tmp;
        if (isInversion)
            answer += cnt[rest];
    }

    cout << answer;

    return 0;
}
```

//

3 C. Мечта детства

3.1 Условие

Ваня с детства мечтал создавать компьютерные игры. Когда он вырос, то смог устроиться в компанию "Игродел" на должность разработчика. Первой его задачей было написание генератора уровней к игре "Линия".

Игровой уровень "Линии" представляет собой множество из N клеток, расположенных вдоль одной прямой и пронумерованных от 0 до $N-1$. Каждая клетка может быть в одном из состояний:

- в клетке стоит персонаж;
- в клетке находится препятствие: змеи, шипы или лава;
- клетка свободна.

Цель главного героя - добраться до клетки с номером $(N-1)$. Единственное, что он может делать, - перепрыгивать через непустое множество препятствий (клетка, в которую прыгает персонаж, должна быть свободна). Каждый раз, когда персонаж совершает прыжок, в точке, откуда он прыгнул, равновероятно возникает одно из препятствий.

Сегодня Ваня доделал генератор уровней. И всё было бы хорошо, если бы не одно "но": Ваня не знает, возможно ли пройти уровень, который получится на выходе генератора. Справиться с этой задачей он не смог. Поэтому Ваня обратился к вам за помощью.

3.2 Входные данные

В первой строке входных данных задано одно целое число: $2 \leq N \leq 100000$. Во второй строке входных данных задана строка S, состоящая из N символов. Строка S описывает игровое поле и состоит из символов:

* X - клетка, в которой находится главный герой;
* ? - клетка, в которой находятся змеи;
* - клетка, в которой находятся шипы;
* $*$ - клетка, в которой находится лава;
* . - свободная клетка

Гарантируется, что X всегда есть.

3.3 Вывод

На первой и единственной строке выведите одно слово:

* YES, если существует последовательность прыжков, после которой персонаж окажется в клетке с номером $(N-1)$;
* NO, иначе.

3.4 Пример входных данных

Sample Input:

3
X..

Sample Output:

NO

Sample Input:

4
X?.

Sample Output:

YES

Sample Input:

5
X?.*.

Sample Output:

YES

3.5 Решение

Запишем в set все свободные клетки.
Дальше будем двигаться по следующему алгоритму.
1. Попробуем прыгнуть вперед на ближайшую клетку справа. Если она не соседняя, прыгаем на нее. Если она соседняя, то посмотрим на ближайшую клетку слева.
2. Если эта клетка не соседняя, - прыгаем на нее. Если соседняя, - уровень пройти невозможно.
Покажем почему это корректное решение.

Ситуация, когда уровень пройти невозможно, может возникнуть только при старте, и, действительно, ни один ход сделать нельзя.
Напомним, что мы прыгаем через отрезок, состоящий хотя бы из одного препятствия и не содержащий свободных клеток.
Единственная ситуация, когда мы не можем двигаться вперед, - несколько подряд соседних свободных клеток.
Логично, что для продвижения надо прыгнуть назад, а потом вернуться на клетку правее. Причем прыжок налево либо невозможен, либо делается в ближайшую слева клетку (следует из условия).
Откуда можно сделать вывод, что чем больше у нас есть клеток слева, тем больше соседних клеток справа от нас мы можем пройти.
Именно поэтому стоит стараться прыгать только вперед. Тем самым экономя шаги назад.

```
#include <chrono>
#include <iostream>
#include <set>
#include <vector>

using namespace std;
using namespace std::chrono;

int main() {
    auto start = high_resolution_clock::now();

    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);

    int n;
    cin >> n;
    string s;
    cin >> s;

    int pos = -1;
    set<int> free;
    for (int i = 0; i < n; ++i) {
        if (s[i] == '.') {
            free.emplace(i);
        } else if (s[i] == 'X') {
            pos = i;
        }
    }

    while (pos != n - 1) {
        auto right = free.lower_bound(pos);
        if (right == free.end()) {
            break;
        }
        if (*right > pos + 1) {
            pos = *right;
            free.erase(pos);
            continue;
        }

        auto left = right;
        if (left == free.begin()) {
            break;
        }
        --left;

        if (*left < pos - 1) {
            pos = *left;
            free.erase(pos);
            continue;
        }

        break;
    }

    cout << (pos == n - 1 ? "YES" : "NO") << endl;

    return 0;
}
```

//

4 D. Секретная разработка

4.1 Условие

На этот раз ученым поручили секретную разработку рельсотрона, но на заключительном этапе у них возникла небольшая трудность.

В данный момент рельсотрон представляет из себя квадратный разгонный блок с областями высокого магнитного поля. При попадании заряженного снаряда в область магнитного поля, он поворачинается на 90 градусов по часовой стрелке.

Ученые находятся на последнем этапе сборки рельстрона и не могут определиться, с какой стороны лучше расположить дуло.

Помогите ученым написать программу, которая по заданной конфигурации разгонного блока определит направление снаряда после всех поворотов.

В программе блок задается в виде квадратной сетки размером $N \times N$ ячеек. В одной из ячеек поля находится снаряд, в некоторых находятся области очень сильного магнитного поля, остальные ячейки пустые.

На сетке введена система координат. При визуализации сетки на плоскости ось x направлена слева направо, ось y сверху вниз. Начальное направление движения снаряда − вверх (противоположно направление оси y).

4.2 Входные данные

В первой строке входных данных задано натуральное число $2 \leq N \leq 1000$ − размер поля.

Далее следуют N строк $L_y, 0 \leq y \leq N$.

В каждой строке заданы N символов $C_{xy}, 0 \leq x \leq N$.

Таким образом, поле имеет вид
$C_{00}C_{10}C_{20}...C_{N-10}$
$C_{00}C_{10}C_{20}...C_{N-10}$
...

Каждый символ C_{xy} может быть либо '.', либо 'X', либо 'I', где
. — свободная ячейка;
X — ячейка с высоким магнитным полем;
I — начальное положение снаряда.
Гарантируется, что снаряд всегда есть и он - один.

4.3 Вывод

В единственной строке выходных данных вывести один из следующих вариантов:

$OUT\ MINUS\_Y$ — снаряд выйдет в направлении, противоположном оси $0y$;

$OUT\ PLUS\_X$ — снаряд выйдет в направлении, сонаправленном оси $0x$;

$OUT PLUS\_Y$ — снаряд выйдет в направлении, сонаправленном оси $0y$;

$OUT\ MINUS\_X$ — снаряд выйдет в направлении, противоположном оси $0x$;

$LOOP$ — снаряд не выйдет за пределы блока.

4.4 Пример входных данных

Sample Input:

```
2
..
.I
```

Sample Output:

```
OUT MINUS_Y
```

Sample Input:

```
10
.X...X....
..........
..X....X..
..........
..........
.X.....X..
..I.......
..........
.....X.X..
..........
```

Sample Output:

```
OUT MINUS_X
```

Sample Input:

10
.X...X....
.....
..X....X..
.....
.....
.X.....X..
..I.....
.....
..X..X.X..
.....

Sample Output:

LOOP

4.5 Примечания к тестам из условия

Комментарий к первому примеру — снаряд вылетит из начальной точки в направлении $MINUS\_Y$, не пройдет через области магнитного поля и покинет пределы сетки в том же направлении.

Комментарий ко второму примеру — снаряд вылетит из начальной точки в направлении $MINUS\_Y$, попадет в магнитное поле, повернётся всего 7 раз и покинет пределы сетки в направлении $MINUS\_X$.

Комментарий к третьему примеру — снаряд зациклится и не покинет пределов сетки.

4.6 Решение

Заметим, что через каждую клетку в фиксированном направлении мы пролетим не более одного раза. Если пролетаем второй, - зациклились.
Если начать моделировать процесс, произойдет одно из двух - или мы вылетим за предела поля (количество смещений на одну клетку $\leq (4 * N * M)$) или же мы зациклимся.
Столько действий мы с запасом успеваем сделать за секунду. А значит остается просто промоделировать полет снаряда по полю, записывая в каких клетках и в каком направлении мы пролетали.

```
#include <iostream>
#include <string>
#include <locale>
#include <vector>
#include <array>
```

```cpp
using namespace std;

enum Cell : char
{
    EMPTY = '.',
    OBSTACLE = 'X',
};

using DirectionT = pair<int, int>;
using PositionT = DirectionT;

array<DirectionT, 5> directions = {
    make_pair( 0,  0),
    make_pair( 0, -1),
    make_pair( 1,  0),
    make_pair( 0,  1),
    make_pair(-1,  0),
};

array<string, 5> directionNames = {
    "None",
    "MINUS_Y",
    "PLUS_X",
    "PLUS_Y",
    "MINUS_X",
};

enum Direction : int
{
    NONE = 0,
    MINUS_Y,
    PLUS_X,
    PLUS_Y,
    MINUS_X,
    DIRECTIONS_COUNT,
};

int main()
{
    size_t n;
    cin >> n;

    vector<vector<Cell>> field(n, vector<Cell>(n));
    vector<vector<int>> mem(n, vector<int>(n, 0));

    PositionT pos;
    Direction dir = MINUS_Y;

    for (size_t y = 0; y < n; ++y)
    {
        for (size_t x = 0; x < n; ++x)
        {
            char c;
            cin >> c;

            if (c == 'I')
                pos.first = x, pos.second = x, c = EMPTY;
            field[x][y] = (Cell)c;
        }
    }

    while (true)
    {
        if (field[pos.first][pos.second] == OBSTACLE)
        {
            // cerr << "OBST " <<  pos.first << ',' << pos.second << endl;
            // cerr << "DIR " <<  directionNames[dir] << endl;

            auto& field_mem = mem[pos.first][pos.second];
            if (field_mem & (1 << (int)dir))
            {
                cout << "LOOP\n";
                return 0;
            }

            field_mem |= (1 << (int)dir);
            dir = (Direction)((int)dir + 1);
            if (dir == DIRECTIONS_COUNT)
                dir = MINUS_Y;
        }

        pos.first += directions[dir].first;
        pos.second += directions[dir].second;

        if (pos.first < 0 || pos.first >= (int)n || pos.second < 0 || pos.second >= (int)n)
        {
            cout << "OUT " << directionNames[dir] << "\n";
            return 0;
        }
    }
```

```
    return 0;
}
```

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

5 E. Ученые уехали в отпуск...

5.1 Условие

Ученые уехали в отпуск, но на их место пришли экономисты.
В одном городе развернулась нешуточная борьба за место главной энергетической станции города.
Изначально присутствуют N станций и N площадок с определенным количеством генераторов, которые обеспечивают соответствующую им станцию.
Некоторые станции, понимая свое невыгодное положение, могут стать подстанцией другой станции. Тогда все генераторы, обеспечивающие их, начинают снабжать другую станцию.
Со временем некоторые генераторы могут выйти из строя, их придется отправить на утилизацию и закупать новые. Все генераторы, поставляющиеся на какую-то площадку, начинают обеспечивать ту же станцию, которую питали прежние. Гарантируется, что линии питания генератора не могут создавать циклов и подключаться к другим станциям помимо основной.
Экономистам необходимо определять, какая станция является самой мощной после каждого изменения.
Будем считать все генераторы одинаковыми, тогда мощность прямо пропорциональна количеству генераторов, обеспечивающих станцию.

5.2 Входные данные

На первой строке входных данных задано одно целое число $1 \leq N \leq 10^5$ - количество площадок.
На второй строке входных данных заданы через пробел N целых чисел. $a_i \leq 10^{10}$ - количество генераторов на площадке i в начальный момент времени.
На третьей строке входных данных задано одно целое число $1 \leq Q \leq 10^5$ - количество событий, которое произошло.
На следующих Q строках заданы через пробел по три целых числа. Первое из них означает тип события.

* Если первое число равно 1, то далее идут два целых числа: номер станции A и номер станции B. При этом стация A становится подстанцией B и все генераторы, которые снабжали ее, начинают снабжать станцию B.

* Если первое число равно 2, то далее идут два целых числа: номер площадки A и $0 \leq c_i \leq 10^{10}$ - новое количество генераторов на площадке A.

5.3 Вывод

На Q строках выходных данных выведите по два целых числа, разделённых пробелом. На строке с номером i выведите номер станции с наибольшим количеством генераторов и количество генераторов, которые ее снабжают. Если две станции имеют одинаковое количество, то выведите наименьший номер станции.

5.4 Пример входных данных

Sample Input:

2
2 3
3
2 1 4
1 1 2
2 1 9

Sample Output:

1 4
2 7
2 12

Sample Input:

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

Sample Output:

1 3
1 3
3 4
1 3
3 6

5.5 Решение

Воспользуемся системой непересекающихся множеств(DSU). Нужно уметь объединить 2 множества в одно, а также узнать главного в этом множестве. Добавим туда поддержку суммы весов в множестве и задача решена.

```cpp
#include <cassert>
#include <chrono>
#include <iostream>
#include <set>
#include <vector>

using namespace std;
using namespace std::chrono;

const int UNITE = 1;
const int CHANGE = 2;

using TGroup = pair<long long, int>;

class TDsu {
public:
    TDsu(int n) {
        Parent.resize(n);
        for (int i = 0; i < n; ++i) {
            Parent[i] = i;
        }
        Weight.resize(n);
        Weight.assign(n, 0);
        WeightOfGroup.resize(n);
        WeightOfGroup.assign(n, 0);
        for (int i = 0; i < n; ++i) {
            MaxWeight.emplace(0, i);
        }
    }

    void SetWeight(int i, long long w) {
        int group = GetParent(i);
        MaxWeight.erase(make_pair(WeightOfGroup[group], group));

        WeightOfGroup[group] = WeightOfGroup[group] - Weight[i] + w;
        MaxWeight.emplace(WeightOfGroup[group], group);
        Weight[i] = w;
    }

    TGroup GetMax() const {
        return *MaxWeight.begin();
    }

    void Unite(int a, int b) {
        assert(a == GetParent(a));
        assert(b == GetParent(b));
        assert(a != b);

        MaxWeight.erase(make_pair(Weight[a], a));
        MaxWeight.erase(make_pair(Weight[b], b));

        Parent[a] = b;
        WeightOfGroup[b] += WeightOfGroup[a];
        MaxWeight.emplace(WeightOfGroup[b], b);
    }

private:
    struct Comparer {
        bool operator ()(const TGroup& lhs, const TGroup& rhs) {
            return lhs.first > rhs.first
                || (lhs.first == rhs.first && lhs.second < rhs.second);
        }
    };

    vector<int> Parent;
    vector<long long> Weight;
    vector<long long> WeightOfGroup;
    set<TGroup, Comparer> MaxWeight;

    int GetParent(int u) {
        return Parent[u] == u ? u : Parent[u] = GetParent(Parent[u]);
    }
};

int main() {
    auto start = high_resolution_clock::now();

    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);

    int n;
    cin >> n;
    TDsu dsu(n);

    for (int i = 0; i < n; ++i) {
        int w;
        cin >> w;
        dsu.SetWeight(i, w);
    }
```

```
    int q;
    cin >> q;

    for (int i = 0; i < q; ++i) {
        int type;
        cin >> type;
        if (type == UNITE) {
            int a, b;
            cin >> a >> b;
            --a;
            --b;

            dsu.Unite(a, b);
        } else if (type == CHANGE) {
            int i;
            long long w;
            cin >> i >> w;
            --i;

            dsu.SetWeight(i, w);
        } else {
            assert(false);
        }

        pair<long long, int> _max = dsu.GetMax();
        cout << _max.second + 1 << " " << _max.first << endl;
    }

    return 0;
}
```

//