1 A. Неожиданное открытие

1.1 Условие

Учеными из вуза M была открыта новая реакция между радиоактивными атомами.

Два атома, a, b, с количеством электронов N и M, соотвественно, после конечного количества соударений переходят в ионизированное состояние с количеством электронов G.
Пусть p_a и p_b − количество электронов в атомах a и b соответственно.
Реакция проходит следующим образом.

1. Если $p_a = p_b$, реакция останавливается.
2. Если $p_a < p_b$, $p_b = p_b - p_a$.
3. Если $p_a > p_b$, $p_a = p_a - p_b$.

Вам необходимо определить конечное число G электронов в атомах.

1.2 Входные данные

В единственной строке входных данных заданы через пробел два целых числа: $1 \leq N, M \leq 1000$.

1.3 Вывод

В единственной строке выходных данных выведите одно число G − конечное число электронов в атомах a и b.

1.4 Пример входных данных

Sample Input:

6 10

Sample Output:

2

Sample Input:

2 3

Sample Output:

1

1.5 Решение

Если присмотреться внимательно к ходу реакции, можно заметить сходство с алгоритмом Евклида для нахождения НОД.
Считаем 2 числа и выведем их НОД.

```
def get_gcd(a, b):
  while b > 0:
    a, b = b, a % b
  return a
n, m = map(int, input().split())
print(get_gcd(n, m))
```

2 B. Вопрос тысячелетия

2.1 Условие

Поликарп недавно перешёл в новую школу. Ему стало интересно, сколько людей на физкультуре будет стоять слева и справа от него.
Слева от Поликарпа будут стоять все одноклассники с ростом, меньшим или равным росту Поликарпа, а справа - все с большим ростом.
Помогите Поликарпу найти ответ на данный вопрос.

2.2 Входные данные

В первой строке входных данных через пробел заданы два числа: N ($2 \leq N \leq 10^5$) и H ($1 \leq H \leq 10^9$) — количество одноклассников Поликарпа и его рост в бипметрах (единица измерения роста в стране Поликарпа).
Во второй строке входных данных через пробел даны N значений h_i. ($1 \leq h_i \leq 10^9$) - рост i-го одноклассника в бипметрах.
Гарантируется, что $min(h_i) \leq H \leq max(h_i)$.

2.3 Вывод

В единственной строке выходных данных выведите через пробел два числа: L и R — количество одноклассников, стоящих слева и справа от него соответственно.

2.4 Пример входных данных

Sample Input:

2 1
1 2

Sample Output:

1 1

Sample Input:

2 2
1 3

Sample Output:

1 1

2.5 Решение

Используя цикл, посчитаем, сколько чисел меньше или равны значению роста Поликарпа, а сколько - строго больше.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    int n, pivot;
    cin >> n >> pivot;
    vector<int> a(n);
    int left = 0;
    int right = 0;
    for (int i = 0; i < n; ++i) {
        cin >> a[i];
        if (a[i] <= pivot) {
            ++left;
        }
        if (a[i] > pivot) {
            ++right;
        }
    }

    cout << left << " " << right << endl;

    return 0;
}


\begin{lstlisting}[language=cpp]

#include <iostream>
#include <vector>

using namespace std;

using ll = long long;

int main() {
    cin.tie(nullptr);
    cout.tie(nullptr);
    ios_base::sync_with_stdio(false);
    int n, m;
    cin >> n >> m;
    vector<vector<ll>> v(n, vector<ll>(m));
    for (auto& line : v) {
        for (auto& pos : line) {
            cin >> pos;
        }
    }
    vector<vector<ll>> pref(n + 1, vector<ll>(m + 1));
    for (int x = 1; x <= n; ++x) {
        for (int y = 1; y <= m; ++y) {
            pref[x][y] = v[x - 1][y - 1] + pref[x][y - 1] + pref[x - 1][y] - pref[x - 1][y - 1];
        }
    }
    ll ans = v[0][0];
    for (int x1 = 0; x1 < n; ++x1) {
        for (int x2 = x1 + 1; x2 <= n; ++x2) {
            int best_y = 0;
            for (int y2 = 1; y2 <= m; ++y2) {
                ll cur = pref[x2][y2] - pref[x1][y2] - pref[x2][best_y] + pref[x1][best_y];
                ans = max(ans, cur);
                if (pref[x1][best_y] - pref[x2][best_y] < pref[x1][y2] - pref[x2][y2]) {
                    best_y = y2;
                }
            }
        }
    }
    cout << ans << '\n';
    return 0;
}

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

3 C. Программирование любой ценой

3.1 Условие

Вася очень любит программировать, однако дома на компьютере стоял пароль. Путем долгих наблюдений он выяснил какие буквы могут использоваться при наборе пароля. Наконец родители поехали на дачу, оставив Васю дома. Помогите выписать Васе всевозможные комбинации символов, чтобы подобрать заветный пароль.

Для удобства все буквы записаны в строку длины N. Паролем может быть любая строка длины L от 1 до N включительно, являющаяся некоей перестановкой букв изначальной строки.

Во входной строке могут быть пробелы, однако Вася точно знает, что в пароле от компьютера их нет. Буквами считаются любые символы, отличные от пробела (ASCII код 32).

3.2 Входные данные

На вход подается одна строка - изначальная фраза. Всего в ней не более 8 символов (вместе с пробелами).

3.3 Вывод

Выведите все строки получившегося множества, по одной на строке.

Порядок слов в ответе не важен.

3.4 Пример входных данных

Sample Input:

aaa

Sample Output:

a
aa
aaa

Sample Input:
a bc

Sample Output:

a
b

c
ab
ac
ba
bc
ca
cb
abc
acb
bac
bca
cab
cba

3.5 Решение

Переберем все перестановки букв. Для каждой перестановки попробуем вывести вначале 1 ее элемент, потом 2, потом 3, ..., потом все элементы. Чтобы не повторяться, будем записывать в set те строки, которые мы уже выводили. А перед выводом - проверять, не пытаемся ли вывести то, что уже когда-то выводили.

```
#include <iostream>
#include <unordered_set>
#include <string>
#include <algorithm>

using namespace std;

int main()
{
    string in;
    string letters;
    while (cin >> in)
    {
        letters += in;
    }
    unordered_set<string> used;
    sort(letters.begin(), letters.end());
    do
    {
        for (size_t i = 0; i < letters.size(); ++i)
        {
            const string out = letters.substr(0, i + 1);

            if (used.find(out) != used.end())
                continue;

            used.insert(out);
            cout << out << '\n';
        }
    } while (next_permutation(letters.begin(), letters.end()));

    return 0;
}
```

//

4 C. Мечта детства

4.1 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.

4.2 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.

4.3 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.

4.4 Пример входных данных

Sample Input:

3
X..

Sample Output:

NO

Sample Input:

4
X?.

Sample Output:

YES

Sample Input:

5
X?.*.

Sample Output:

YES

4.5 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;
}
```

//

5 D. Секретная разработка

5.1 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).

5.2 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.

5.3 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.

5.4 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

5.5 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.

5.6 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>
```

```
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;
}
```

//