▷ Задача 1. Спартанцы, в атаку!(2D)

Царь Леонид решил повести своё войско в наступательный поход.

Он построил войско в N рядов по M воинов в каждом, оглядел их и решил проверить их боеспособность.

Для проверки боеспособности он придумал следующую схему:

Он подошёл к некоторым бойцам, и шепнул им на ухо команду "в бой!".

За одну секунду боец, который знает команду, может передать её правому, левому, переднему и заднему соседям в ряду.

Леонид хочет узнать, за сколько секунд все бойцы войска узнают команду?

Воины представлены бинарной строкой. (1 — если воин изначально знал команду, 0 — если не знал)

Входные данные

В первой строке написаны два целых положительных числа N и $M(1 \leqslant N, M \leqslant 500)$ — размеры войска.

Следующие N строк по M символов описывают войско.

Выходные данные:

В ответе выведите единственное число - минимальное количество секунд, через которое все спартанцы уже будут знать команду.

Ограничение времени выполнения программы:

1 секунда

Примеры тестовых данных:

Входные данные	Результат работы программы
3 4 1000 1000 1010	3
5 4 1001 0000 0010 0000 0000	2
10 10 0000000000 0000000000 0001000000 0000000000 0000000100 0000000000 0000010000 0000000000 0000000000 0000101001	6

Решение задачи на языке Python

```
from collections import deque
def F(N, M, armya):
    napr = [(0, 1),
```

```
                (0, -1),
                (1, 0),
                (-1, 0)]
    queue = deque()
    bylo = [[False for _ in range(M)] for _ in range(N)]
    for i in range(N):
        for j in range(M):
            if armya[i][j] == '1':
                queue.append((i, j, 0))
                bylo[i][j] = True
    while queue:
        x, y, time = queue.popleft()
        for dx, dy in napr:
            nx, ny = x + dx, y + dy
            if ((0 <= nx < N )and (0 <= ny < M) and (not (bylo[nx][ny]))):
                bylo[nx][ny] = True
                queue.append((nx, ny, time + 1))
    return time
N, M = map(int,input().split())
armya = []
for n in range(N):
    stroka = input()
    armya.append(stroka)
print(F(N, M, armya))
```

▷ Задача 2. Не короткое замыкание

Вам дана парта размера $N \times M$ — клетчатое поле. В каждой клетке либо находится электрический элемент (он обозначается за #), либо ничего не находится (*).

Два элемента соединены, если они находятся в клетках, граничащих по стороне.

Гарантируется, что цепь связная и незамкнутая.

Вову заинтересовал вопрос: если замкнуть какие-то два элемента (например, соединить их проволокой), то сколько в замыкании максимум будет участвовать элементов?

(Примеч: элемент находится в замыкании, если он находится в цикле, который образовался, когда Вова замкнул два элемента).

Входные данные

В первой строке записано два целых числа: N и $M (2 \leqslant N, M \leqslant 100)$ — высота и ширина парты соответственно. После этого идут N строк длины M — описание парты.

Выходные данные:

Выведите максимальную длину цикла, который получится, если замкнуть какие-то два элемента.

Ограничение времени выполнения программы:

1 секунда

Примеры тестовых данных:

Входные данные	Результат работы программы
3 3 ### ..# .##	6

Входные данные	Результат работы программы
4 5 ..#.. .##.. ..##.	4
5 4 .#.. ##.. .### ##.# .#..	6

Решение на C++

С заданием полностью справились только два участника. Приводим решение призера отборочного тура, Адиля Канатбекова, 9 класс, г. Бишкек.

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define pb push_back
#define db double
#define pll pair<ll, ll>
#define fs first
#define sc second
ll n,m;
vector<ll> dx{1, 0, -1, 0};
vector<ll> dy{0, 1, 0, -1};
vector<vector<char>> g;
vector<vector<ll>> dist1, dist2;
pll dfs(pll a, vector<vector<ll>> &dist){
    pll mx={0, 0};
    ll x=a.fs;
    ll y=a.sc;
    for(ll i=0; i<4; i++){
        if(x+dx[i]>0 && x+dx[i]<=m && y+dy[i]>0 && y+dy[i]<=n && g[x+dx[i]][y+dy[i]]=='#'
         && dist[x+dx[i]][y+dy[i]]==-1){
            dist[x+dx[i]][y+dy[i]]=1;
            dist[x+dx[i]][y+dy[i]]=dist[a.fs][a.sc]+1;
            if(dist[mx.fs][mx.sc]<dist[x+dx[i]][y+dy[i]]) mx={x+dx[i], y+dy[i]};
            pll mx1=dfs({x+dx[i], y+dy[i]}, dist);
            if(dist[mx1.fs][mx1.sc]>dist[mx.fs][mx.sc]) mx=mx1;
        }
    }
    return mx;
}
int main(){
    ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    cin>>n>>m;
```

```
    pll id;
    g.resize(m+1, vector<char>(n+1));
    dist1.resize(m+1, vector<ll>(n+1, -1));
    dist2.resize(m+1, vector<ll>(n+1, -1));
    for(ll i=1; i<=n; i++){
        for(ll j=1; j<=m; j++){
            cin>>g[j][i];
            if(g[j][i]=='#') id={j, i};
        }
    }
    dist1[id.fs][id.sc]=1;
    pll mx=dfs(id, dist1);
    dist2[mx.fs][mx.sc]=1;
    pll ans=dfs(mx, dist2);
    cout<<dist2[ans.fs][ans.sc]<<"\n";
}
```

▷ Задача 3. Апельсин и спаниель

Малыш давно просил у родителей собаку. И не простую собаку, а спаниеля.

К сожалению, папа Малыша не захотел покупать ему собаку.

Чтобы Малыш не сильно расстраивался, Карлсон подарил ему апельсин, ведь если переставить буквы в слове "апельсин то можно получить слово "спаниель".

Малыш заинтересовался анаграммами (словами, которые получаются друг из друга перестановкой букв), поэтому в тетрадку он записал имена всех предметов в своей комнате. Помогите Малышу понять, какое наибольшее количество анаграмм одного слова встречается в его тетрадке.

Входные данные

Первая строка содержит одно натуральное число N — количество слов в тетрадке малыша.

В следующих N строках написано по одному слову — названия предметов. Гарантируется, что нет двух совпадающих строк.

Выходные данные:

В ответе выведите одно число - максимальное количество слов из списка, которые являются анаграммой одного слова.

Пояснение:

Например, при входных данных:

5

apelsin

spaniel

abc

bca

cab

Ответ должен быть 3. (abc, bca и cab образуют 3 анаграммы одного слова).

Ограничение времени выполнения программы:

1 секунда

Примеры тестовых данных:

Входные данные	Результат работы программы
3 apelsin spaniel stul	3
7 acbaaba aaabcab abacaab acbbaaa baaaabc baacaab abbacaa	7

Решение задачи на языке C++

```
#include<iostream>
#include <vector>
#include <cmath>
#include <algorithm>
#include <string>
#include <iomanip>
#include <queue>
#include <map>
using namespace std;
int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    int n;
    cin>>n;
    map<vector <int>, int> m;
    int ma=0;
    for (int i=0; i<n; i++){
        vector <int> a(26);
        string s;
        cin>>s;
        int l=s.size();
        for (int j=0; j<l; j++){
            a[s[j]-'a']++;
        }
        m[a]++;
        if (m[a]>ma) ma=m[a];
    }
    cout<<ma;
    return 0;
}
```

▷ **Задача 4. Чебурашка и апельсины**

Чебурашка очень любит апельсины. В магазине, где он живёт, продавец иногда даёт Чебурашке его любимое лакомство.

Однажды когда Чебурашке стало скучно, он решил взять все свои апельсины и положить их на чашечные весы так, чтобы весы встали в положение равновесия.

Помогите Чебурашке — сообщите ему, может ли он положить свои апельсины на весы нужным образом.

Входные данные

Первая строка содержит целое положительное число $N(2 \leqslant N \leqslant 20)$ — количество апельсинов.

Вторая строка содержит N натуральных чисел — веса апельсинов, которые не превосходят 10^6.

Выходные данные:

Выведите единственную строку:

NO — если нельзя положить апельсины на весы

YES — если можно расположить апельсины на весах требуемым образом

Ограничение времени выполнения программы:

1 секунда

Примеры тестовых данных:

Входные данные	Результат работы программы
4 1 2 3 4	YES
10 560969 173071 784074 994974 581388 391519 76845 939307 420796 59099	YES
10 35762 258056 461849 499400 977649 613476 430517 825321 23036 56093	NO

Решение задачи на языке Python

```
def can_balance_oranges(N, weights):
    total_weight = sum(weights)

    for i in range(1 << N):
        current_weight = 0

        for j in range(N):
            if i & (1 << j):
                current_weight += weights[j]

        if current_weight == total_weight / 2:
            return "YES"

    return "NO"


N = int(input())
weights = list(map(int, input().split()))

print(can_balance_oranges(N, weights))
```

▷ **Задача 5. Послание коротышкам с Земли**

После того, как Незнайка улетел, коротышки с Луны искали способ доставить без потерь сообщение. Было решено применить код Хэмминга(7,4)

Код Хэмминга — самоконтролирующийся и самокорректирующийся код. Построен применительно к двоичной системе счисления.

Позволяет исправлять одиночную ошибку (ошибка в одном бите слова) и находить двойную. Назван в честь американского математика Хэмминга Ричарда Уэсли, предложившего код.

Для кодирования сообщения его разбивают на блоки по 4 символа($d_1d_2d_3d_4$) и в каждый из этих блоков добавляют проверочные биты ($p_1p_2p_3$), на позиции 1,2 и 4 и блок преобразуется к виду $p_1p_2d_1p_3d_2d_3d_4$.

На рисунке показано, как был построен код. Бит p_1 соответствует четности зеленого кружка, если $d_1 + d_2 + d_4$ четно, p_1 равен нулю, в противном случае p_1 равен единице. Следовательно, если повреждения не происходит, сумма битов в зеленом кружке будет четной. Эта логика продолжается на красном и синем кругах.

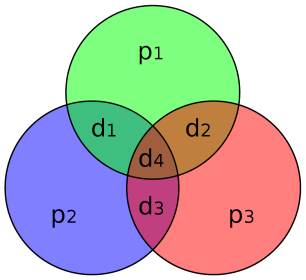

Рис. 5.1 — Построение кода Хэмминга

Помогите лунатикам зашифровать сообщение следующим образом:

1. Каждый символ сообщения зашифрован с помощью ASCII кода
2. ASCII код преобразован в двоичный код
3. Полученная строка разбита на блоки по 4 бита
4. Каждый блок закодирован кодом Хэмминга

Входные данные:

Исходное сообщение

Выходные данные:

Строка из нулей и единиц, длиной не более 1000 символов — зашифрованное послание

Ограничение времени выполнения программы:

1 секунда

Примеры тестовых данных:

Входные данные	Результат работы программы
LIKE	1001100011110010011000011 0011001100011001110011000 100101

here we go	110011011100001100110010 101000111101010110011100 100101010101000000000011 110001111110011001010101 010100000000110011000111 111001101111111
why	000111100011111100110111 00000011110011001

Решение задачи на языке Python

```
K = 4
def encode(s):
    res = ''
    while len(s) >= K:
        nibble = s[0:K]
        res += (hamming(nibble))
        s = s[K:]
    return res

def hamming(bits):
     t1 = parity(bits, [0,1,3])
    t2 = parity(bits, [0,2,3])
    t3 = parity(bits, [1,2,3])
    return t1 + t2 + bits[0] + t3 + bits[1:]

def parity(s, indicies):
    sub = ""
    for i in indicies:
        sub += s[i]
    return str(str.count(sub, "1") % 2)

s = "".join(digit for char in input() for digit in f"{ord(char):08b}")
encode(s)
if(len(s)):
    print(encode(s))
```

▷ **Задача 6. Корабль-призрак**

Капитан Джек Воробей и его команда путешествуют на Черной Жемчужине по Карибскому морю в поисках корабля-призрака. Чтобы найти корабль-призрак, им нужно попасть в проклятый многоугольник.

Известна текущая координата корабля и координаты вершин многоугольника. Помогите Джеку определить, насколько далеко им осталось плыть. Если они уже внутри многоугольника, вывести 0.

Входные данные: Координата Черной Жемчужины x, y.

Координаты точек проклятого многоугольника

Выходные данные: Одно число, расстояние до многоугольника или ноль, с округлением до двух знаков после запятой

Ограничение времени выполнения программы:

1 секунда

Примеры тестовых данных:

Входные данные	Результат работы программы
4 1 -4 -4 4 -4 5 5 -5 5	0
6 1 -4 -4 4 -4 5 5 -5 5	1.44
100000 -900000 -4 -4 4 -4 0 5	905534.1

Решение задачи на языке Python

С этой задачей полностью не справился ни один из участников.

```
from typing import List, Tuple
import math

def point_in_polygon(polygon, point) :
    """
    Подробнее: https://en.wikipedia.org/wiki/Point_in_polygon
    """
    xp, yp = point
    lines_intersection = 0
    common_vertex = 0
    for (x1, y1), (x2, y2) in zip(polygon, polygon[1:] + polygon[:1]):
        if (y2 <= yp <= y1 or y1 <= yp <= y2) and not (x1 < xp and x2 < xp):
            if x1 == x2:
                if xp < x1:
                    lines_intersection += 1
            else:
                a = (y2 - y1) / (x2 - x1)
                b = y1 - a * x1
                if (yp - b) / a > xp:
                    lines_intersection += 1
                if yp == y1 or yp == y2:
                    common_vertex += 1
    return bool((lines_intersection - common_vertex // 2) & 1)
```

```
def dot(v,w):
    x,y = v
    X,Y = w
    return x*X + y*Y

def length(v):
    x,y = v
    return math.sqrt(x*x + y*y)

def vector(b,e):
    x,y = b
    X,Y = e
    return (X-x, Y-y)

def unit(v):
    x,y = v
    mag = length(v)
    return (x/mag, y/mag)

def distance(p0,p1):
    return length(vector(p0,p1))

def scale(v,sc):
    x,y = v
    return (x * sc, y * sc)

def add(v,w):
    x,y = v
    X,Y = w
    return (x+X, y+Y)

def pnt2line(pnt, start, end):
    line_vec = vector(start, end)
    pnt_vec = vector(start, pnt)
    line_len = length(line_vec)
    line_unitvec = unit(line_vec)
    pnt_vec_scaled = scale(pnt_vec, 1.0/line_len)
    t = dot(line_unitvec, pnt_vec_scaled)
    if t < 0.0:
        t = 0.0
    elif t > 1.0:
        t = 1.0
    nearest = scale(line_vec, t)
    dist = distance(nearest, pnt_vec)
    nearest = add(nearest, start)
    return dist

def minimal_distance(polygon,point):
    min_dist = 1000000
```

```
    for (x1, y1), (x2, y2) in zip(polygon, polygon[1:] + polygon[:1]):
        dist = pnt2line(point,(x1,y1),(x2,y2))
        if dist<min_dist:
            min_dist = dist
    return min_dist

point = tuple(map(float,input().split(' ')))
polygon  = []

while True :
    try:
        p = tuple(map(float,input().split(' ')))
        polygon.append(p)
    except:
        break


if(point_in_polygon(polygon,point)):
    print(0)
else:
    print(round(minimal_distance(polygon,point),2))
```