Отборочный (заочный) онлайн-этап академического соревнования Олимпиады школьников «Шаг в будущее» по программированию «Профессор Лебедев» (общеобразовательный предмет информатика), осень 2020 год

11 класс

Вариант 1

Задача 1 (8 баллов)

Условие

Для N (от 1 до 100) заданных натуральных чисел (каждое не превышает 1 000 000 000) определить, какое из них требуется изменить на наименьшее значение, чтобы оно делилось без остатка на заданный делитель (не превышает 1 000 000).

Входные данные: в первой строке - число N, затем N строк с заданными числами, записанными по одному в строке. Затем в последней строке - делитель.

Выходные данные: через пробел вывести номер искомого числа и значение, на которое требуется его изменить. Числа нумеровать с единицы. Если существует несколько чисел, которые требуется изменить на одно и то же значение - вывести первое из них.

Пример

Ввод	Вывод
2 12 13 5	1 2
3 17 139 5 7	2 1

Проверочные тесты

Входные данные	Ожидаемый результат
2 12 13 5	1 2
3 17 139 5 7	2 1

1 1001 10	1 1
1 999 10	1 1
2 10 20 5	1 0
10 111 112 113 114 115 116 126 136 137 138 20	10 2
10 111 112 113 114 115 116 126 136 18 138 20	9 2

Пример решения

```
nums = []
n = int(input())
for i in range(n):
    nums.append(int(input()))
divider = int(input())
smen = 0
flag = True

while flag:
    for i in nums:
        if (i + smen) % divider == 0 or (i - smen) % divider == 0:
```

```
            flag = False
            ind = nums.index(i) + 1
            print(ind, smen)
            break
    smen += 1
```

Задача 2 (12 баллов)

Условие
Для двух заданных больших натуральных чисел (количество цифр в каждом числе не превышает 200) определить, делится ли их сумма на 9, и остаток от деления.

Входные данные: 2 числа, записанных по одному в строке.

Выходные данные: число 0, если сумма делится на 9 без остатка, и остаток в ином случае.

Пример

Ввод	Вывод
888888888888888888 11111111111111111111	0
987 11	8

Проверочные тесты

Входные данные	Ожидаемый результат
888888888888888888 11111111111111111111	0
987 11	8
1 1	2
121212121212121212121212121212121212121212121212121212121212121212121212121 212121212121212121212121212121212121212121212121212121212121212121212121212 878787878787878787878787878787878787878787878787878787878787878787878787878 787878787878787878787878787878787878787878787878787878787878787878787878787	0
121212121212121212121212121212121212121212121212121212121212121212121212121 212121212121212121212121212121212121212121212121212121212121212121212121212 878787878787878787878787878787878787878787877787878787878787878787878787878 787878787878787878787878787878787878787878787878787878787878787878787878787	8

10000000000000000000000000000000000000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000000000000000 800	0
99999999999999999999999999999999999999999999999999999999999999999999999999999 99999999999999999999929999999999999999999 99989	1

Пример решения

```
s1 = input()
s2 = input()

result = 0

for i in s1:
    result += int(i)

for i in s2:
    result += int(i)

print(result % 9)
```

Задача 3 (16 баллов)

Условие
Прямоугольный лист бумаги сложили вчетверо: сначала пополам по горизонтали слева направо, затем - по вертикали сверху вниз. После этого в нём проделали несколько отверстий.

Требуется восстановить вид листа после того, как его развернули обратно.

Входные данные: в первой строке - 2 натуральных числа N и M (каждое в диапазоне от 2 до 20), записанных через пробел. N - количество строк в описании сложенного листа, M - количество столбцов в описании сложенного листа. Далее - N строк по M символов, описывающих вид сложенного листа. Символом + обозначается участок листа без отверстия, пробелом - с отверстием.

Выходные данные: 2*N строк по 2*M столбцов, соответствующих развёрнутому листу.

Пример

<table>
<tr><th>Ввод</th><th>Вывод</th></tr>
<tr><td><pre>4 5
+++++
+   +
+ +++
+++++</pre></td><td><pre>++++++++++
+++ ++ +++
+   ++   +
++++++++++
++++++++++
+   ++   +
+++ ++ +++
++++++++++</pre></td></tr>
</table>

Проверочные тесты

Входные данные	Ожидаемый результат
4 5 +++++ + + + +++ +++++	++++++++++ +++ ++ +++ + ++ + ++++++++++ ++++++++++ + ++ + +++ ++ +++ ++++++++++
2 2 ++ ++	++++ ++++ ++++ ++++
10 10 ++++++++++ + + + ++ + +++ ++++++++++ ++++++++++ ++++ + +++ + + + ++++++++++	++++++++++++++++++++ + ++ + + ++++++ + + ++++++++ + ++++++++++++++++++++ ++++++++++++++++++++ +++ ++ +++ ++ ++ ++ + ++ + ++++++++++++++++++++ ++++++++++++++++++++ + ++ + ++ ++ ++ +++ ++ +++ ++++++++++++++++++++ ++++++++++++++++++++ + ++++++++ + + ++++++ + + ++ + ++++++++++++++++++++
3 4 ++++ ++ + ++++	++++++++ + ++++ + ++++++++ ++++++++ + ++++ + ++++++++

5 3 +++ + + + + +++ +++	++++++ ++++++ + ++ + + ++ + ++++++ ++++++ + ++ + + ++ + ++++++ ++++++

Пример решения

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

using namespace std;

int main()
{
    int a, s, d, f, k, temp;
    cin >> a >> s;
    string ch;
    vector<vector<char>> c(a, vector<char>(s));
    getline(cin, ch);
    for (int i = 0; i < a; i++)
    {
        getline(cin, ch);

        for (int j = 0; j < s; j++)
        {
            c[i][j] = ch[j];
        }
    }
    for (int i = a - 1; i >= 0; i--)
    {
        for (int j = s - 1; j >= 0; j--)
        {
            cout << c[i][j];
        }
        for (int j = 0; j < s; j++)
        {
            cout << c[i][j];
        }
        cout << endl;
    }
    for (int i = 0; i < a; i++)
    {
        for (int j = s - 1; j  >= 0; j--)
        {
            cout << c[i][j];
        }
```

```
        for (int j = 0; j < s; j++)
        {
            cout << c[i][j];
        }
        cout << endl;
    }
    return 0;
}
```

Задача 4 (20 баллов)

Условие
Начинающий программист Петя решил написать свою первую компьютерную 3D-игру. По его задумке, ландшафт игрового мира хранится следующим образом: каждый высотный элемент представляет собой вертикальный отрезок с известными координатами в двумерном пространстве и высотой. Начала всех отрезков имеют нулевую Z-координату. Кроме того, все отрезки "растут" вверх, то есть абсцисса и ордината концов каждого отрезка совпадают, а z-координата верхнего конца равна высоте.

Вам требуется определить по заданному ландшафту и положению наблюдателя самый высокий объект, который виден наблюдателю (не заслоняется полностью другими объектами).

Входные данные: в первой строке - вещественные координаты "точечного" наблюдателя x, y, z, записанные через пробел. Во второй строке - натуральное число K от 1 до 100, соответствующее количеству объектов ландшафта. Далее - K строк, содержащих по 3 вещественных числа, записанных через пробел. В каждой строке числа идут в следующем порядке: x-координата объекта ландшафта, затем соответствующая y-координата, затем высота.

Выходные данные: Номер самого высокого объекта, видимого наблюдателю (считать, что объекты нумеруются по порядку с единицы).

Пример

Ввод	Вывод
1 0 0 2 0 1 2 2 1 1	1
0.5 0.5 1 3 2 2 4 1 1 3 0 1 2	2

Примечание: наблюдатель имеет обзор во все стороны из указанной точки на неограниченное расстояние, его видимость ограничена только отрезками, заслоняющими другие.

Проверочные тесты

Входные данные	Ожидаемый результат
1 0 0 2 0 1 2 2 1 1	1
0.5 0.5 1 3 2 2 4 1 1 3 0 1 2	2
1 1 1 4 2 2 2 4 4 10 10 10 100 100 100 200	3
0 0 0 1 1 1 0.01	1
-0.1 -0.1 0.1 6 -0.1 100 1 -0.1 200 100 -0.1 300 150 100 -0.1 1 200 -0.1 101 300 -0.1 150	5

Пример решения

```
#include <iostream>
#include <cmath>
#include <vector>
#include <algorithm>
#include <string>
using namespace std;

int main()
{
    double x, y, z, x_cur, y_cur, z_cur;
    cin >> x >> y >> z;
    int n, pos;
    cin >> n;
    double tgz = -10000000;
    double tgx, tgy;
    for (int i = 0; i < n; i++)
    {
        cin >> x_cur >> y_cur >> z_cur;
```

```
        if (sqrt((x - x_cur)*(x - x_cur) + (y - y_cur)*(y - y_cur))/(z-
z_cur) > tgz)
        {
            tgz = sqrt((x - x_cur)*(x - x_cur) + (y - y_cur)*(y -
y_cur))/(z-z_cur);
            pos = i+1;
            tgy = y_cur;
            tgx = x_cur;

        }
        else if ((sqrt((x - x_cur)*(x - x_cur) + (y - y_cur)*(y -
y_cur))/(z-z_cur) == tgz) and ((((tgx -x > 0) and (x_cur -x> 0)) or
((tgx -x< 0) and (x_cur -x< 0))) and (((tgy -y> 0) and (y_cur -y> 0))
or ((tgy -y< 0) and (y_cur -y< 0))) and (((x-tgx) / (y-tgy == 0 ? 0 :
y-tgy ) == (x-x_cur) / (y-y_cur == 0 ? 0 : y-y_cur ))) and((sqrt((x -
x_cur)*(x - x_cur) + (y - y_cur)*(y - y_cur)) < sqrt((x - tgx)*(x - tgx)
+ (y - tgy)*(y - tgy))))))
        {
            tgz = sqrt((x - x_cur)*(x - x_cur) + (y - y_cur)*(y -
y_cur))/(z-z_cur);
            tgx = x_cur;
            tgy = y_cur;
            pos = i+1;
        }
    }
    cout << pos;
    return 0;
}
```

Задача 5 (20 баллов)

Условие

Префиксными кодами называются такие коды, в которых ни одна более короткая комбинация не является началом более длинной комбинации. В частности, двоичные префиксные коды используются в алгоритмах сжатия информации.

Требуется составить программу, которая по заданной кодовой таблице определит, является ли он префиксным, и если является - выведет декодированное сообщение.

Входные данные: в первой строке - натуральное число N (не превышающее 60), описывающее число существующих кодов. Далее идёт N строк в формате "символ" (без кавычек), затем пробел, затем запись кода, состоящая из цифр 0 и 1. Далее идёт строка с закодированным сообщением. Символами могут являться строчные и заглавные латинские буквы, а также цифры.

Выходные данные: если код является префиксным, то в первой строке - слово YES, а во второй - декодированное сообщение. Если код не является префиксным, то в одной строке следует вывести NO и через пробел номер первого кода, нарушающего условие префиксности (являющегося префиксом другого кода). Коды нумеруются с единицы.

Пример

Ввод	Вывод
5 a 1 B 001 m 000 n 010 u 011 00110110001010	YES Bauman
3 E 0 S 1 Y 01 0101	NO 1

Проверочные тесты

Входные данные	Ожидаемый результат
5 a 1 B 001 m 000 n 010 u 011 00110110001010	YES Bauman
3 E 0 S 1 Y 01 0101	NO 1
1 K 000 000000	YES KK
10 v 00 n 01 i 100 u 101 e 1100 r 1101 s 11100 t 11101 y 11110 ! 11111 10101100001100110111100100111011111011111	YES university!

10 a 0000 b 0001 c 0010 d 0011 e 0100 f 0101 g 010 h 0111 j 1000 k 100	NO 7

Пример решения

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

#define pb push_back
#define X first
#define Y second
#define endl '\n'
#define int long long

using namespace std;

const int P = 7, MOD = 1e9 + 7;

int p_pow[1000010];

signed main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0), cout.tie(0);
    p_pow[0] = 1;
    for (int i = 0; i < 1e6 + 5; ++i) {
        p_pow[i + 1] = (p_pow[i] * P) % MOD;
    }
    int n;
    cin >> n;
    vector<vector<int>> hs(n);
    map<int, char> mp;
    for(int j = 0; j < n; ++j) {
        char ch;
        string s;
        cin >> ch >> s;
        int h = 0;
        for(int i = 0; i < s.size(); ++i) {
            h = (h + (s[i] - '0' + 1) * p_pow[i] % MOD) % MOD;
            hs[j].pb(h);
        }
        mp[h] = ch;
    }
    string s;
    cin >> s;
    for(int i = 0; i < n; ++i) {
```

```
        for(int j = 0; j < n; ++j) {
            if (i != j && hs[j].size() >= hs[i].size() &&
hs[j][hs[i].size() - 1] == hs[i].back()) {
                cout << "NO " << i + 1;
                return 0;
            }
        }
    }
    cout << "YES\n";
    int po = 0;
    int h = 0;
    for(int i = 0; i < s.size(); ++i) {
        h = (h + (s[i] - '0' + 1) * p_pow[po] % MOD) % MOD;
        po++;
        if (mp.find(h) != mp.end()) {
            cout << mp[h];
            po = 0;
            h = 0;
        }
    }
}
```

Задача 6 *Ситуационная задача* (24 балла)

Условие
Компьютерные программы на основе "искусственного интеллекта" для игр в шахматы и шашки показывают улучшение результатов каждый год. Для этого им требуется быстро и качественно распознавать различные ситуации, возникающие на игровой доске.

Вам требуется написать программу, которая для заданных позиций шашечной партии и цвета, чей ход - очередной, определит, какая из шашек может побить (взять) наибольшее число шашек противника.

Правила взятия (ударного хода) следующие. Ударный ход - перемещение шашки на два поля вперед или назад по диагонали, через шашку противника. Шашка при своем ходе обязана побить (взять) шашку противника, если та находится на соседнем (по диагонали) поле и если следующее за ней поле свободно. Берущая шашка становится на это свободное поле, перескакивая через вражескую шашку, которая снимается с доски. Взятие может совершаться не только вперед, но и назад.

Если после взятия одной шашки оказывается возможным побить еще одну шашку противника, взятие продолжается, то есть в один прием (за один ход) шашка должна побить столько шашек соперника, сколько их стоит на ее пути.

При возможности взятия в разных направлениях выбор, вне зависимости от количества или качества снимаемых шашек, предоставляется берущему игроку.

Запрещается при взятии перескакивать более одного раза бьющей шашкой через одну и ту же шашку соперника (а вот на свободное поле наступать несколько раз разрешается).

В рамках данной задачи считать, что дамок (шашек, получивших в ходе игры возможность перемещаться на произвольное количество полей) на доске нет.

Входные данные: в первой строке записано число N (от 1 до 10) - число позиций, которые нужно проверить. Далее идёт N описаний позиций в следующем формате: каждая позиция - одна строка с буквой B или W, обозначающей цвет, который ходит следующим, и далее 8 строк по 8 символов, которыми могут быть буквы B, W и точка. W соответствует белой шашке, B - чёрной, точка - свободной клетке.

Выходные данные: N строк с указанием клетки, в которой находится шашка, способная взять наибольшее число шашек противника в соответствующей позиции. Клетку необходимо указать двумя символами: буквой, соответствующей вертикали поля, и цифрой, соответствующей горизонтали поля. Во всех позициях вертикали обозначаются слева направо заглавными латинскими буквами от A до H, а горизонтали нумеруются снизу вверх цифрами от 1 до 8.

Пример

Вход	Выход
1 W .B.B.B.B B.B.B.B. ...B.B.B ..B..... ...W.... W...W.W. .W.W.W.W W.W.W.W.	D4

Проверочные тесты

Входные данные	Ожидаемый результат
1 W .B.B.B.B B.B.B.B. ...B.B.B ..B..... ...W.... W...W.W. .W.W.W.W W.W.W.W.	D4

1 B ..... .B..... ..W.....	B7
1 B .... .W..... B..... ...W.... ....B...	E1
2 W .....B. .....W.. ..B..... ...W.... .... B .....B.. ....W... .B..... ..W..... ....W.W.	D5 B6
1 W .B..... B.B.B.B. ...B...B ..B..... ...W.B.. W...W.W. .W.W.W.W W.W.W.W.	D4

Пример решения

```
N = int(input())

def find_ways(matrix, position, side, used):
    ways = []
    if side == 'W':
        contr_side = 'B'
    else:
        contr_side = 'W'
    i,j = position[0], position[1]
    if ((i > 0) and (j > 0)) and (matrix[i-1][j-1] == contr_side) and
([i-1, j-1] not in used):
        if (i > 1) and (j > 1) and (matrix[i - 2][j - 2] == '.'):
            ways.append({'free':[i-2, j-2], 'contr':[i-1,j-1]})
    elif ((i > 0) and (j < 7)) and (matrix[i-1][j+1] == contr_side) and
(([i-1, j+1] not in used)):
        if ((i > 1) and (j < 6)) and (matrix[i - 2][j + 2] == '.'):
            ways.append({'free':[i-2, j+2], 'contr':[i-1,j+1]})
    elif ((j > 0) and (i < 7)) and (matrix[i+1][j-1] == contr_side) and
([i+1, j-1] not in used):
        if ((j > 1) and (i < 6)) and (matrix[i + 2][j - 2] == '.'):
            ways.append({'free':[i+2, j-2], 'contr':[i+1,j-1]})
    elif ((i < 7) and (j < 7)) and (matrix[i+1][j+1] == contr_side) and
([i+1, j+1] not in used):
        if ((i < 6) and (j < 6)) and (matrix[i + 2][j + 2] == '.'):
            ways.append({'free':[i+2, j+2], 'contr':[i+1,j+1]})
    return ways

def find_points(side, matrix):
    if side == 'W':
        contr_side = 'B'
    else:
        contr_side = 'W'
    positions = []
    for i in range(len(matrix)):
        for j in range(len(matrix[i])):
            if matrix[i][j] == side:
                if ((i > 0) and (j > 0)) and (matrix[i-1][j-1] ==
contr_side):
                    positions.append([i,j])
                elif ((i > 0) and (j < 7)) and (matrix[i-1][j+1] ==
contr_side):
                    positions.append([i,j])
                elif ((j > 0) and (i < 7)) and (matrix[i+1][j-1] ==
contr_side):
                    positions.append([i,j])
                elif ((i < 7) and (j < 7)) and (matrix[i+1][j+1] ==
contr_side):
                    positions.append([i,j])
    return positions

def DFS(matirx, position, side, k, used):
```

```
    ways = find_ways(matirx, position, side, used)
    if len(ways) == 0:
        return k
    dif_k = []
    for i in range(len(ways)):
        contr = ways[i]['contr']
        used.append(contr)
        k += 1
        dif_k.append(DFS(matirx, ways[i]['free'], side, k, used))
    return max(dif_k)

for q in range(N):
    side = input()
    matrix = []
    for i in range(8):
        matrix.append(input())
    positions = find_points(side, matrix)
    dif_k = []
    for i in range(len(positions)):
        dif_k.append({'pos':positions[i], 'k':DFS(matrix, positions[i],
side, 0, [])})
    result = max(dif_k, key = lambda x: x['k'])['pos']
    result[0] += 1
    result[1] += 1
    print(chr(64 + result[1]), 8 - result[0] + 1, sep = '')
```