Заключительный (очный) этап научно-образовательного соревнования

Олимпиады школьников «Шаг в будущее» по профилю «Инженерное дело» специализации «Техника и технологии» (общеобразовательный предмет информатика), весна 2020 г.

11 класс

Вариант 1

Задача 1

Учительница математики Марина Григорьевна попросила старшеклассников составить программу для тестирования младших учеников по геометрии, которая будет выдавать задачи на проверку принадлежности точки - прямой.

Требуется написать программу, которая будет контролировать правильность ответов учеников.

Входные данные: в первой строке уравнение вида ax+by+c=0, где a, b и c - целые числа от 0 до 1000, а знаками операции могут быть как "+", так и "-". Во второй строке - целые числа - координаты проверяемой точки.

Выходные данные: если точка лежит на прямой - слово "YES", иначе - слово "NO" и через пробел - целая часть расстояния от точки до прямой.

Пример

Исходные данные	Результат
1x+1y+0=0 1 -1	YES
2x-3y+1=0 2 0	NO 1

Проверочные тесты

1x+1y+0=0 1 -1	YES
2x-3y+1=0 2 0	NO 1
1000x+1000y+1000=0 1 -2	YES
0x+2y+0=0 1 2	NO 2
3x+0y-1=0 2 1	NO 1

Пример решения

```
Y = input()
xt, yt = map(int, input().split())
```

```
f = 0
a = 0
b = 0
c = 0
A = Y.split('x')
B = A[1].split('y')
C = B[1].split('=')[0]
a = int(A[0])
b = int(B[0])
c = int(C)

if a*xt+b*yt+c == 0:
        print("YES")
else:
        print("NO", int(abs(a*xt+b*yt+c)//(a*a+b*b)**(0.5)))
```

Задача 2

В строке записано истинное логическое выражение с тремя переменными a, b и c. Над переменными применяются 2 операции: эквивалентность и исключающее «или». Требуется восстановить значения таблицы истинности функции, соответствующей этому выражению.

Исключающее «или» - булева функция двух переменных, результат которой истинен тогда и только тогда, когда один аргумент истинен, а второй - ложен.

Эквивалентность - логическое выражение, которое является истинным тогда, когда оба простых логических выражения (левая и правая части) имеют одинаковую истинность.

В рамках данной задачи будем обозначать исключающее «или» знаком "^", а эквивалентность - знаком "=" без кавычек.

Входные данные: логическое выражение, состоящее из имён переменных a, b, c, знаков исключающего «или» и эквивалентности. Длина выражения не превышает 20 символов.

Выходные данные: 8 цифр 0 и 1, записанных неразрывно и означающих значения функции для каждой из комбинаций значений переменных:

0, 0, 0;
0, 0, 1;
0, 1, 0;
0, 1, 1;
1, 0, 0;
1, 0, 1;
1, 1, 0;
1, 1, 1.

Пример

Исходные данные	Результат
a^b^c	01101001
a^b=b^c	10100101

Примечание: операция исключающего «или» имеет более высокий приоритет по сравнению с эквивалентностью.

Проверочные тесты

a^b^c	01101001
a^b=b^c	10100101
b	00110011
a=a	11111111
b^c^b=c^b^a	11000011

Пример решения

```
def xor(a, b):
	return a ^ b

def eq(a, b):
	return a == b

def op_index(lst, op):

	for i, e in enumerate(lst):
	if e == op:
	return i
	return None

def eval(lst, op):
	op_idx = op_index(lst, op)
	if op_idx is None:
	return False
	else:
	f = xor if lst[op_idx] == '^' else eq
	arg1 = lst[op_idx - 1]
	arg2 = lst[op_idx + 1]
	result = f(arg1, arg2)

	lst[op_idx] = result
	del lst[op_idx + 1]
	del lst[op_idx - 1]

	return True

def zamena(x_lst, d):
	ks = d.keys()
	for k in ks:
	for i, e in enumerate(x_lst):
	if e == k:
		x_lst[i] = d[k]

x = input()
x_lst = list(x)

for a in [False, True]:
```

```
for b in [False, True]:
for c in [False, True]:
d = {'a': a, 'b': b, 'c': c}
x_lst_copy = list(x_lst)
zamena(x_lst_copy, d)

while eval(x_lst_copy, '^'):
        pass

while eval(x_lst_copy, '='):
        pass

result = int(x_lst_copy[0])
print(result, end="")
```

Задача 3

Дана запись двух больших целых чисел в шестнадцатеричной системе счисления и их произведения, при этом в записи допущена одна ошибка.

Написать программу, которая выявит допущенную ошибку и исправит её.

Входные данные: три строки, в каждой из которых записано по шестнадцатеричному числу. Количество знаков в каждом из чисел не превышает 100. Цифры от A до F записаны заглавными латинскими буквами.

Выходные данные: исправленная запись в том же формате.

Пример:

Исходные данные	Результат
10 A A2	10 A A0
1F 27 49A	1F 26 49A

Проверочные тесты

10 A A2	10 A A0
1F 27 49A	1F 26 49A
2 7 10	2 8 10
1000000000000000000000000000000000000000 0	1000000000000000000000000000000000000000 0

1000000000000000000000000000000000000007 1000000000000000000000000000000000000007 0000000000000000000000000000000000000003	1000000000000000000000000000000000000007 1000000000000000000000000000000000000007 0000000000000000000000000000000000000000
123456789ABCDEF0 8 91A2B3C4D5E77780	123456789ABCEEF0 8 91A2B3C4D5E77780

Пример решения

```
symbols = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F']

def check(a, b, c):
	return int(a, 16) * int(b, 16) == int(c, 16)

hexs = []
hexs.append(input())
hexs.append(input())
hexs.append(input())

if check(hexs[0], hexs[1], hexs[2]):
	print(hexs[0], hexs[1], hexs[2], sep="\n")
	exit()

for hexi in range(len(hexs)):
	for sym in range(len(hexs[hexi])-1, -1, -1):
	for v in symbols:
	new = hexs[hexi][:sym] + v + hexs[hexi][sym+1:]
	if hexi == 0:
		if check(new, hexs[1], hexs[2]):
		print(new, hexs[1], hexs[2], sep='\n')
		exit()
	elif hexi == 1:
		if check(hexs[0], new, hexs[2]):
		print(hexs[0], new, hexs[2], sep='\n')
		exit()
	elif hexi == 2:
		if check(hexs[0], hexs[1], new):
		print(hexs[0], hexs[1], new, sep='\n')
		exit()
```

Задача 4

Пете нравится решать судоку, и он задумался, как можно изменить эту головоломку, чтобы она стала интереснее. После размышлений ему пришла идея сделать судоку в 16-ричной системе счисления, но он засомневался, получится ли её решать так же, как обычную.

Правила 16-ричных судоку, которые придумал Петя: дано поле 16x16, разделённое на квадраты 4x4. Допустимы все возможные 16-ричные цифры от 0 до F, при этом не должно быть повторов одной цифры в строках и столбцах поля, а также в отдельных квадратах.

Требуется написать программу, которая заполнит недостающие клетки в заданном 16-ричном судоку.

Входные данные: 16 строк по 16 символов с цифрами от 0 до F. Пустые поля обозначены символами "_" (знак подчёркивания). Количество пустых полей не превышает 30.

Выходные данные: решённое судоку - 16 строк по 16 символов с цифрами от 0 до F, такие, что все пустые поля заполнены пропущенными цифрами.

Пример:

Исходные данные	Результат
12_456_890ABCDEF 567890ABCDEF1234 90ABCDEF12345678 CDEF123_567890AB 3_567890A_CDE_12 7890ABCDEF123456 ABCDEF1234567890 EF12_4567890ABCD 678_0ABCDEF12345 234__7890A__DEF1 DEF1234567890_BC 0ABC_EF123456789 890ABCDEF123456_ _567890ABCDEF1_3 _1234567890ABCDE BCDEF123456789_A	1234567890ABCDEF 567890ABCDEF1234 90ABCDEF12345678 CDEF1234567890AB 34567890ABCDEF12 7890ABCDEF123456 ABCDEF1234567890 EF1234567890ABCD 67890ABCDEF12345 234567890ABCDEF1 DEF1234567890ABC 0ABCDEF123456789 890ABCDEF1234567 4567890ABCDEF123 F1234567890ABCDE BCDEF1234567890A

Примечание: гарантируется, что исходные данные корректны, судоку может быть решено и решение единственное.

Проверочные тесты

12_456_890ABCDEF 567890ABCDEF1234 90ABCDEF12345678 CDEF123_567890AB 3_567890A_CDE_12 7890ABCDEF123456 ABCDEF1234567890 EF12_4567890ABCD 678_0ABCDEF12345 234__7890A__DEF1 DEF1234567890_BC 0ABC_EF123456789 890ABCDEF123456_ _567890ABCDEF1_3 _1234567890ABCDE BCDEF123456789_A	1234567890ABCDEF 567890ABCDEF1234 90ABCDEF12345678 CDEF1234567890AB 34567890ABCDEF12 7890ABCDEF123456 ABCDEF1234567890 EF1234567890ABCD 67890ABCDEF12345 234567890ABCDEF1 DEF1234567890ABC 0ABCDEF123456789 890ABCDEF1234567 4567890ABCDEF123 F1234567890ABCDE BCDEF1234567890A
34567890ABCDEF12 7890ABCDEF123456 ABCDEF1234567890	34567890ABCDEF12 7890ABCDEF123456 ABCDEF1234567890

EF1234567890ABCD 67890ABCDEF12345 234567890ABCDEF1 DEF1234567890ABC _____ 890ABCDEF1234567 4567890ABCDEF123 F1234567890ABCDE BCDEF1234567890A 1234567890ABCDEF 567890ABCDEF1234 90ABCDEF12345678 CDEF1234567890AB	EF1234567890ABCD 67890ABCDEF12345 234567890ABCDEF1 DEF1234567890ABC 0ABCDEF123456789 890ABCDEF1234567 4567890ABCDEF123 F1234567890ABCDE BCDEF1234567890A 1234567890ABCDEF 567890ABCDEF1234 90ABCDEF12345678 CDEF1234567890AB
34567890ABCDEF1_ 7890ABCDEF12345_ ABCDEF123456789_ EF1234567890ABC_ 67890ABCDEF1234_ 234567890ABCDEF_ DEF1234567890AB_ 0ABCDEF12345678_ 890ABCDEF123456_ 4567890ABCDEF12_ F1234567890ABCD_ BCDEF1234567890_ 1234567890ABCDE_ 567890ABCDEF123_ 90ABCDEF1234567_ CDEF1234567890A_	34567890ABCDEF12 7890ABCDEF123456 ABCDEF1234567890 EF1234567890ABCD 67890ABCDEF12345 234567890ABCDEF1 DEF1234567890ABC 0ABCDEF123456789 890ABCDEF1234567 4567890ABCDEF123 F1234567890ABCDE BCDEF1234567890A 1234567890ABCDEF 567890ABCDEF1234 90ABCDEF12345678 CDEF1234567890AB
3456789_ABCDEF12 7890ABC_EF123456 ABCDEF1_34567890 EF12345_7890ABCD 67890AB_DEF12345 2345678_0ABCDEF1 DEF1234_67890ABC 0ABCDEF_23456789 _____ 4567890_BCDEF123 F123456_890ABCDE BCDEF12_4567890A 1234567_90ABCDEF 567890A_CDEF1234 90ABCDE_12345678 CDEF123_567890AB	34567890ABCDEF12 7890ABCDEF123456 ABCDEF1234567890 EF1234567890ABCD 67890ABCDEF12345 234567890ABCDEF1 DEF1234567890ABC 0ABCDEF123456789 890ABCDEF1234567 4567890ABCDEF123 F1234567890ABCDE BCDEF1234567890A 1234567890ABCDEF 567890ABCDEF1234 90ABCDEF12345678 CDEF1234567890AB
34567890ABCDEF12 7890ABCDEF123456 ABCDEF1234567890 EF1234567890ABCD	34567890ABCDEF12 7890ABCDEF123456 ABCDEF1234567890 EF1234567890ABCD

6789____DEF12345	67890ABCDEF12345
23456_8_0ABCDEF1	234567890ABCDEF1
DEF1234_67890ABC	DEF1234567890ABC
0ABCDEF_23456789	0ABCDEF123456789
890ABCDE____4567	890ABCDEF1234567
4567890AB_D_F123	4567890ABCDEF123
F1234567890_BCDE	F1234567890ABCDE
BCDEF123456_890A	BCDEF1234567890A
1234567890ABCDEF	1234567890ABCDEF
567890ABCDEF1234	567890ABCDEF1234
90ABCDEF12345678	90ABCDEF12345678
CDEF1234567890AB	CDEF1234567890AB

Пример решения

```
SZ = 4
SZ2 = SZ*SZ
matrix = []
for i in range(0, SZ2):
        row = list(input())
        row_dec = [int(i, 16) if i != '_' else i for i in row]
        matrix.append(row_dec)

def check(x, y, board):
        temp = board[x][y]
        board[x][y] = "_"

        for row in range(SZ2):
        if board[row][y] == temp:
        return False

        for col in range(SZ2):
        if board[x][col] == temp:
        return False

        for row in range(SZ):
        for col in range(SZ):
        if board[(x//SZ) * SZ+row][(y//SZ) * SZ + col] == temp:
                return False
        board[x][y] = temp
        return True

def find(board):
        for row in range(SZ2):
        for col in range(SZ2):
        if board[row][col] == '_':
                for number in range(SZ2+1):
                board[row][col] = number
                if check(row, col, board) and find(board):
                return board
                else:
                board[row][col] = '_'
```

```
                return None
        return board
# proverka
res = find(matrix)
for row in res:
        for char in row:
        print(str(hex(char))[2:].upper(), end = "")
        print()
```

Задача 5

Двоичным деревом называется иерархическая структура данных, в которой каждый узел имеет не более двух потомков.

Двоичное дерево поиска - разновидность двоичного дерева, у которого оба поддерева (левое и правое) являются двоичными деревьями поиска; все узлы левого поддерева произвольного узла A меньше, чем значение самого узла A; все узлы правого поддерева произвольного узла A больше либо равны значению узла A.

Высотой узла называется максимальная длина нисходящего пути от этого узла к самому нижнему узлу, называемому листом. Высота корневого узла равна высоте всего дерева.

Дерево называется сбалансированным, если для любой его вершины высота левого и правого поддерева для этой вершины различаются не более чем на 1.

При прямом обходе дерева сначала обрабатывается ключ (значение) корня, затем - ключи левого и правого поддеревьев.

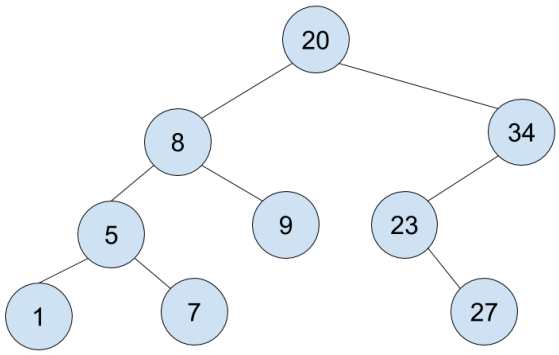
По заданным ключам вершин, полученным при прямом обходе, требуется определить, является ли дерево сбалансированным, а также определит высоту дерева.

Входные данные: ключи всех вершин двоичного дерева поиска в порядке прямого обхода. Каждый ключ - натуральное число, не превышающее 105. Каждое значение задано в отдельной строке. Последняя строка состоит из одного символа - точки.

Выходные данные: первая строка - слово YES, если дерево сбалансированное, и NO, если нет. Вторая строка - число, соответствующее высоте дерева.

Пример

Исходные данные и результат соответствуют дереву, изображённому на рисунке:



Исходные данные	Результат
20 8 5 1 7 9 34 23 27 .	NO 4

Проверочные тесты

20 8 5 1 7 9 34 23 27 .	NO 4
7 3 2 1 5 4 6 9 8 .	YES 4
1 .	YES 1
4 2 1 3 6 5 7 .	YES 3
3 2 1 .	NO 3

Пример решения

```
class Tree:
	@staticmethod
	def parse(xs):
	l = len(xs)
	if l == 0:
	return List()
	else:
	head = xs[0]
	tail = xs[1:]
	split = next((i for i, e in enumerate(tail) if e >= head), len(tail))
	xs_l = tail[:split]
	xs_r = tail[split:]

	return Uzel(head, Tree.parse(xs_l), Tree.parse(xs_r))
class Uzel(Tree):
	def __init__(self, e, l, r):
	self.e = e
	self.r = r
	self.l = l

	def depth(self):
	return 1 + max(self.l.depth(), self.r.depth())

	def bal(self):
	d_l = self.l.depth()
	d_r = self.r.depth()

	return (abs(d_l - d_r) < 2) and self.l.bal() and self.r.bal()

class List(Tree):
	def depth(self):
	return 0
	def bal(self):
	return True

xs = []

while(True):
	x_st = input()
	if x_st == ".":
	break
	else:
	x = int(x_st)
	xs.append(x)

tree = Tree.parse(xs)
print("YES" if tree.bal() else "NO")
print(tree.depth())
```