Чичунов Михаил Эдуардович

Балл: 46

Задача №1

Условие

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

Исходный код

```
from math import sqrt

eq = list(input())


a = eq[:eq.index('x')]
a = int(''.join(a))


b = eq[(len(str(a))+1):eq.index('y')]
b = int(''.join(b))


c = eq[len(str(a))+len(str(b)) + 2:eq.index('=')]
c = int(''.join(c))



data = list(map(int, input().split(' ')))
x = data[0]
y = data[1]
if a*x + b*y + c == 0:
    print('YES')
else:
    k = -a/b
    if not y == 0:
        c-=y
        y=0

    sin = sqrt(1/((1/(k**2))+1))
    z = (-c*b/a) + x
    r = int(abs(z*sin))
    print('NO', r)
```

Задача №2

Условие

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.

Исходный код

```
v = input().split('=')
answers = []


for i in range(8):
    c = i%2
    i //=2
    b = i%2
    i //=2
    a = i%2

    pa = []
    for d in v:
        aa = d.count('a')
        ba = d.count('b')
        ca = d.count('c')
        r = (aa*a + ba*b+ca*c)%2
        pa.append(r)

    for i in range(len(pa)-1):
        if pa[i] == pa[i+1]:
            x = 1
        else:
            x = 0
        pa[i+1] = x
    answers.append(pa[-1])

for a in answers:
    print(a, end='')
```

Задача №3

Условие

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

Исходный код

```
def hextodec(x):
    i = 0
    res = 0
    chars='0123456789ABCDEF'
    while not i == len(x):
        a = x[-1*(i+1)]
        res += 16**i * chars.index(a)
        i += 1
    return res


def dectohex(x):
    x = str(hex(int(x)))[2:].upper()
    return x


ahex = input()
bhex = input()
chex = input()

a = (hextodec(ahex))
b = (hextodec(bhex))
c = (hextodec(chex))


a1 = ahex
b1 = bhex
c1 = chex
if c%a == 0:
    b1 = c//a
    b1 = dectohex(b1)
elif c%b == 0:
    a1 = c//b
    a1 = dectohex(a1)
else:
    c1 = a*b
    c1 = dectohex(c1)

print(a1)
print(b1)
print(c1)
```

Задача №4

Условие

Пете нравится решать судоку, и он задумался, как можно изменить эту головоловку, 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	1234567890ABCDEF
567890ABCDEF1234	567890ABCDEF1234
90ABCDEF12345678	90ABCDEF12345678
CDEF123_567890AB	CDEF1234567890AB
3_567890A_CDE_12	34567890ABCDEF12
7890ABCDEF123456	7890ABCDEF123456
ABCDEF1234567890	ABCDEF1234567890
EF12_4567890ABCD	EF1234567890ABCD
678_0ABCDEF12345	67890ABCDEF12345
234__7890A__DEF1	234567890ABCDEF1
DEF1234567890_BC	DEF1234567890ABC
0ABC_EF123456789	0ABCDEF123456789
890ABCDEF123456_	890ABCDEF1234567
_567890ABCDEF1_3	4567890ABCDEF123
_1234567890ABCDE	F1234567890ABCDE
BCDEF123456789_A	BCDEF1234567890A

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

Исходный код

```
sudoku = []

def check_full(matrix):
    for s in matrix:
        if '_' in s:
            return False
    return True

for i in range(16):
    s = list(input())
    sudoku.append(s)
    s = []

chars = '0123456789ABCDEF'

while not check_full(sudoku):
    hcount = 0
    vcount = 0
    for i in range(16):
        hcount = 0
        for j in range(16):
            if sudoku[i][j] == '_':
                hcount += 1
        if hcount == 1:
            x = sudoku[i].index('_')
            for char in chars:
                if not char in sudoku[i]:
                    sudoku[i][x] = char

    for i in range(16):
        vcount = 0
        for j in range(16):
            if sudoku[j][i] == '_':
                vcount += 1
        if vcount == 1:
            help = [sudoku[k][i] for k in range(16)]
            x = help.index('_')
            for char in chars:
                if not char in help:
                    sudoku[x][i] = char
for s in sudoku:
    print(''.join(s))


'''
12_456_890ABCDEF
3_567890A_CDE_12
567890ABCDEF1234
7890ABCDEF123456
```

```
90ABCDEF12345678
ABCDEF1234567890
CDEF123_567890AB
EF12_4567890ABCD
_1234567890ABCDE
DEF1234567890_BC
BCDEF123456789_A
0ABC_EF123456789
890ABCDEF123456_
678_0ABCDEF12345
_567890ABCDEF1_3
234__7890A__DEF1
   '''
```

Задача №5

Условие

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.

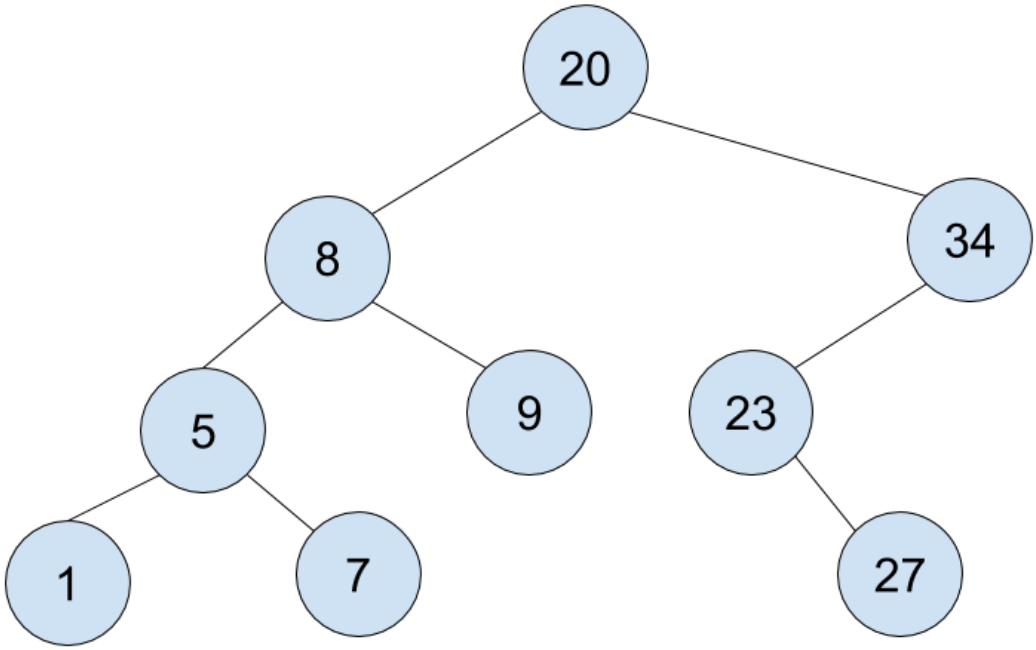
По заданным ключам вершин, полученным при прямом обходе, требуется определить, является ли дерево сбалансированным, а также определит высоту дерева.

Входные данные: ключи всех вершин двоичного дерева поиска в порядке прямого обхода. Каждый ключ - натуральное число, не превышающее 10_5. 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

Исходный код