Бурлаков Илья Андреевич

Балл: 53

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

```
import math
linestr = input()
apart, linestr = linestr.split('x')
bpart, linestr = linestr.split('y')
cpart, linestr = linestr.split('=')
a = int(apart)
b = int(bpart)
c = int(cpart)
px, py = input().split()
px = int(px)
py = int(py)
r = a*px+b*py+c
if (r == 0):
    print('YES')
else:
    d = abs(r)/math.sqrt(a*a+b*b)
    print(r,d)
    d = math.floor(d)
    print('NO',d)
```

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

```
def evalute(line,a,b,c):
    line = line.replace('a', a)
    line = line.replace('b', b)
    line = line.replace('c', c)

    while(len(line)>1):
        left = line[0]
        right = line[2]
        op = line[1]

        if (op == '='):
            if (left==right):
                line = '1'+line[3:]
            else:
                line = '0'+line[3:]
        else:
            if (left==right):
                line = '0'+line[3:]
            elif (left=='0' or right=='0'):
                line = '1'+line[3:]
            else:
                line = '0'+line[3:]
    return line

string = input()
res = ""
for a in ['0','1']:
    for b in ['0','1']:
        for c in ['0','1']:
            res = res + evalute(string,a,b,c)
print(res)
```

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 try_multi(x, res):
    return res/x


n1 = int(input(),16)
n2 = int(input(),16)
n3 = int(input(),16)

if (n1*n2==n3):
    n1 = n1
else:
    nt = try_multi(n2,n3)
    if (nt.is_integer()):
        n1 = nt
    else:
        nt = try_multi(n1,n3)
        if (nt.is_integer()):
            n2 = nt
        else:
            n3 = n1*n2

n1 = hex(int(n1))[2:].upper()
n2 = hex(int(n2))[2:].upper()
n3 = hex(int(n3))[2:].upper()

print(n1)
print(n2)
print(n3)
```

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

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.

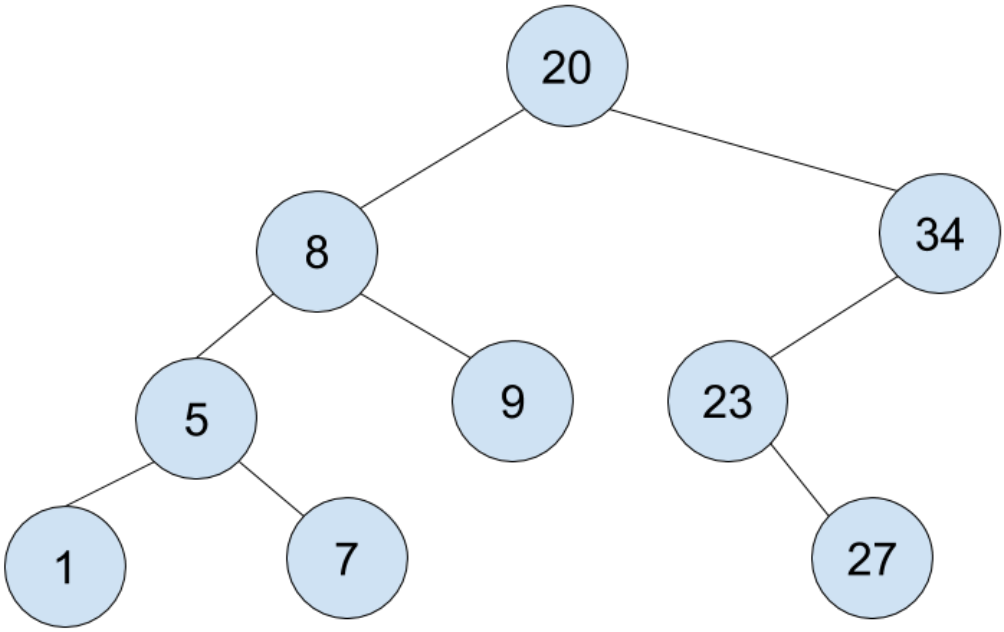
По заданным ключам вершин, полученным при прямом обходе, требуется определить, является ли дерево сбалансированным, а также определит высоту дерева.

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

Исходный код

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

```
class Tree:
    def __init__(self, left, right, value):
        self.left = left
        self.right = right
        self.value = value

    def add (self, val):
        if (val < self.value):
            if (self.left == None):
                self.left = Tree(None, None, val)
            else:
                self.left.add(val)
        else:
            if (self.right == None):
                self.right = Tree(None, None, val)
            else:
                self.right.add(val)

    def fwdprint(self):
        print(self.value)
        if (self.left):
            self.left.fwdprint()
        if (self.right):
            self.right.fwdprint()

    def checkbalance(self):
        if (self.left == None and self.right==None):
            return 1, True

        ld = 0
        lb = True
        rd = 0
        rb = True

        if (self.left):
            ld, lb = self.left.checkbalance()
        if (self.right):
            rd, rb = self.right.checkbalance()

        if (abs(ld-rd) <= 1):
            sb = True
        else:
            sb = False

        cd = max(ld,rd)+1

        return cd, (sb and lb and rb)


line = input()
if (line == '.'):
    quit()

t = Tree(None,None,int(line))

line = input()
while (line != '.'):
    t.add(int(line))
    line = input()


depth, balance = t.checkbalance()
if (balance):
    print('YES')
else:
    print('NO')

print(depth)
```