Гарин Егор Олегович

Балл: 52

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

```
def get_k(s, letter):
    result = 0
    if letter in s:
        result, s = s.split(letter)
        if result == '-':
            result = -1
        elif result == '+' or '':
            result = 1
        else:
            result = int(result)
    return result, s


def get_distance(exp, x, y, a, b):
    exec('result='+exp)
    result = int(result / (a ** 2 + b ** 2) ** 0.5)
    return result


original_s = input()[-2:]
s = original_s
x, y = map(int, input().split())

a, s = get_k(s,'x')
b, s = get_k(s,'y')
distance = get_distance(a, b, s, x, y)
if distance == 0:
    print("YES")
else:
    print("NO", distance)
```

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

```
xor = lambda a, b: (a or b) - (a and b)
equal = lambda a, b: int(a == b)


def run(vars, f, a, b, c):
    nums = []
    for vari in range(len(vars)):
        if vars[vari] == 'a':
            vars[vari] = a
        elif vars[vari] == 'b':
            vars[vari] = b
        elif vars[vari] == 'c':
            vars[vari] = c

    result = vars[0]
    for vari in range(1, len(vars)):
        result = f(result, vars[vari])

    return result


s = input()

equals = []
if '=' in s:
    equals = s.split('=')
else:
    equals.append(s)

result = ''

for n in range(0, 8):
    abc = list(map(int, list(bin(n)[2:])))
    abc = [0]*(3 - len(abc)) + abc
    local_eq = []
    for eqi in range(len(equals)):
        local_eq.append(run(equals[eqi].split('^'), xor, abc[0], abc[1], abc[2]))

    result += str(run(local_eq, equal, abc[0], abc[1], abc[2]))

print(result)
```

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

```
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

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

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

for line in range(16):
    field.append(list(input()))

for i in range(16):
    line_set = set(field[i]) - set('_')
    new_set = symbols - line_set
    new_set = sorted(list(new_set))
    for j in range(16):
        if field[i][j] == "_":
            field[i][j] = new_set.pop(0)

for line in field:
    print(*line)

# извините за это
```

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.

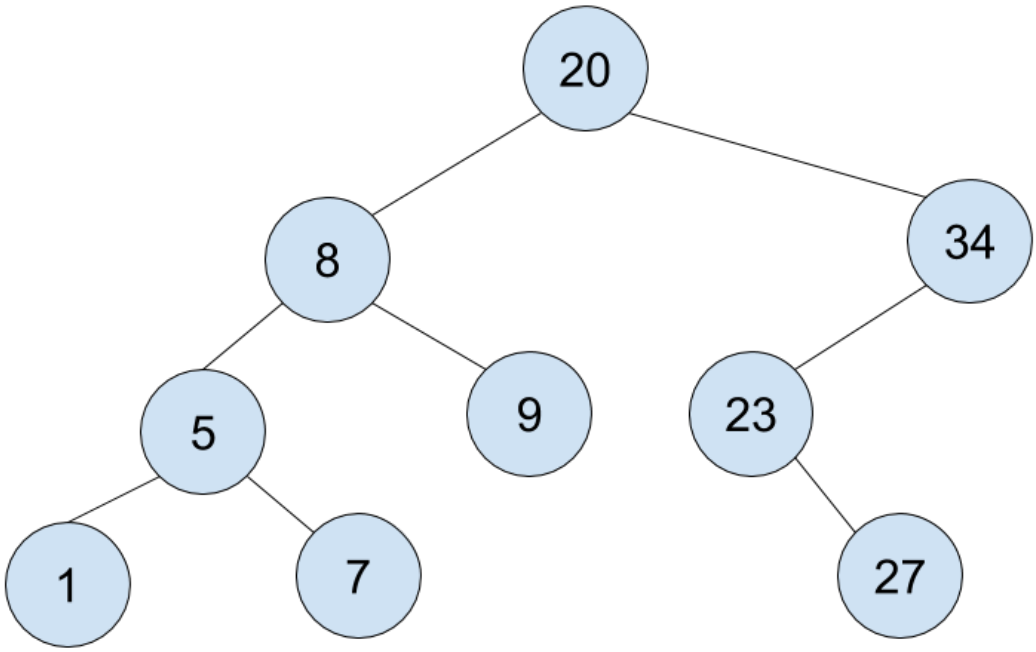
По заданным ключам вершин, полученным при прямом обходе, требуется определить, является ли дерево сбалансированным, а также определит высоту дерева.

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

```
a = []
while True:
    c = input()
    if c == '.':
        break
    else:
        a.append(int(c))


# a = [20, 8, 5, 1, 7, 9, 34, 23, 27]


class Noda:
    def __init__(self, d):
        self.l = None
        self.r = None
        self.h = 0
        self.d = d

    def push_node(self, d):
        if self.d != None:
            if d < self.d:
                if self.l == None:
                    self.l = Noda(d)
                else:
                    self.l.push_node(d)
            elif d > self.d:
                if self.r == None:
                    self.r = Noda(d)
                else:
                    self.r.push_node(d)
        else:
            self.d = d
```

```
    def work_on_heights(self):
        self.h = self.get_height() - 1

    def get_height(self):
        if self.r != None:
            r_height = self.r.get_height()
            self.r.h = r_height - 1
        else:
            r_height = 1
        if self.l != None:
            l_height = self.l.get_height()
            self.l.h = l_height - 1
        else:
            l_height = 1
        return max(r_height, l_height) + 1

    def is_balanced(self):
        if self.h == 0:
            return True
        if self.r == self.l == None:
            return True
        if self.r == None:
            if self.l.h == 1:
                return True
            return False
        if self.l == None:
            if self.r.h == 1:
                return True
            return False
        return abs(self.r.h - self.l.h) <= 1 and self.r.is_balanced() and self.l.is_balanced()


root = Noda(a[0])
for n in a[1:]:
    root.push_node(n)
root.work_on_heights()
print('YES' if root.is_balanced() else 'NO')
print(root.h)  # +-1 ???
```