Княжев Алексей Викторович

Балл: 74

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

```
#include <cstdio>
#include <cmath>
using namespace std;

double square_of_sum_squares (int a, int b) {
    double sum_sq = a * a + b * b;
    return sqrt(sum_sq);
}

int main() {
    int a, b, c;
    char sym1, sym2;
    scanf("%dx%c%dy%c%d=0", &a, &sym1, &b, &sym2, &c);
    int x, y;
    scanf("%d %d", &x, &y);
    double up = a * x;
    if (sym1 == '+') {
        up += b * y;
    } else {
        up -= b * y;
    }

    if (sym2 == '+') {
        up += c;
    } else {
        up -= c;
    }

    if (up == 0) {
        printf("YES\n");
    } else {
        printf("NO %d\n", (int)(abs(up) / square_of_sum_squares(a, b)));
    }
    return 0;
}
```

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 get_abc_from_num(num):
    c = num % 2
    num //= 2
    b = num % 2
    num //= 2
    a = num % 2
    num //= 2
    return a, b, c


equation = input()
equation = equation.replace("=", "==")
for i in range(8):
    a, b, c = get_abc_from_num(i)
    print(int(eval(equation)), sep="", end="")
print()
```

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

```
import string

def count_difference(x, y):
    if x == y:
        return 0
    diff = 0
    while x > 0 or y > 0:
        diff += 1 if x % 16 != y % 16 else 0
        x //= 16
        y //= 16
    return diff
```

```
def out(x):
    return str(hex(x)).replace('0x', '').title()


a = int(input(), 16)
b = int(input(), 16)
c = int(input(), 16)

if count_difference(a * b, c) == 1:
    print(out(a), out(b), out(a*b), sep='\n')
    exit(0)

if c % a == 0 and count_difference(b, c//a) == 1:
    print(out(a), out(c//a), out(c), sep='\n')
    exit(0)

if c % b == 0 and count_difference(a, c//b) == 1:
    print(out(c//b), out(b), out(c), sep='\n')
    exit(0)

print('Incorrect input!')
```

Задача №4

Условие

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	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

```
all_letters = '0123456789ABCDEF'


def search_nones(string):
    result = ''
    string = string.replace('_', '')
    for el in all_letters:
        if string.count(el) == 0:
            result += el
    return result
```

```
def make_arr_col(matrix, index):
    result = ""
    for i in range(len(matrix)):
        result += matrix[i][index]
    return result


def search_letter(list1, list2):
    for elem in list1:
        if list2.count(elem) == 1:
            return elem


inp = []
for i in range(16):
    inp.append(input())

vars = [[], []]
cols = [make_arr_col(inp, i) for i in range(16)]
rows = [row for row in inp]

for el in cols:
    vars[0].append(search_nones(el))
for el in rows:
    vars[1].append(search_nones(el))


for i_outer, some_row in enumerate(inp):
    for i_inner, letter in enumerate(some_row):
        if letter == '_':
            new_letter = search_letter(vars[0][i_inner], vars[1][i_outer])
            vars[0][i_inner] = vars[0][i_inner].replace(new_letter, '')
            vars[1][i_outer] = vars[1][i_outer].replace(new_letter, '')
            inp[i_outer] = inp[i_outer].replace('_', new_letter, 1)

for p in inp:
    print(p)
```

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.

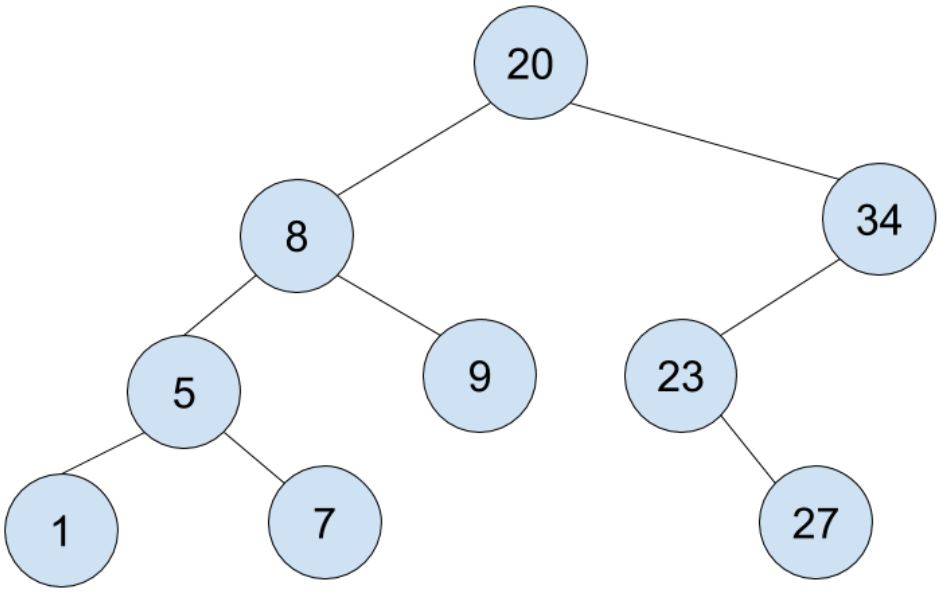
По заданным ключам вершин, полученным при прямом обходе, требуется определить, является ли дерево сбалансированным, а также определит высоту дерева.

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

```
def right_start_index(graph):
    for i, _ in enumerate(graph):
        if graph[i] >= graph[0] and i > 0:
            return i
    return -1


def graph_search(g):
    if len(g) <= 1:
        return len(g)
    start_right = right_start_index(g)
    if start_right == -1:
        left = g[1:]
        right = []
    else:
        left = g[1:start_right]
        right = g[start_right:]

    return 1 + max(graph_search(left), graph_search(right))
```

```
def is_good(g):
    if len(g) <= 1:
        return True
    start_rightf = right_start_index(g)

    if start_rightf == -1:
        leftf = g[1:]
        rightf = []
    else:
        leftf = g[1:start_rightf]
        rightf = g[start_rightf:]

    left_heightf = graph_search(leftf)
    right_heightf = graph_search(rightf)

    if abs(left_heightf - right_heightf) <= 1:
        return True and is_good(leftf) and is_good(rightf)
    else:
        return False


graph = []

while True:
    inp = input()
    if inp == '.':
        break
    graph.append(int(inp))

if len(graph) == 1:
```

```
    print("YES\n1\n")
    exit(0)

if is_good(graph):
    print("YES")
else:
    print("NO")
print(graph_search(graph))
```