Ткаченко Владислав Львович

Балл: 51

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

exp = input()
x, y = map(int, input().split())

x_idx = exp.index('x')
A = int(exp[:x_idx])

y_idx = exp.index('y')
B = int(exp[x_idx+1:y_idx])

equal_idx = exp.index('=')
C = int(exp[y_idx+1:equal_idx])

if A == 0 and B == 0 and C != 0: # крайний случай
 print('NO')
 exit()

if A * x + B * y + C == 0:
 print('YES')
else:
 distance = int(abs(A * x + B * y + C) / sqrt(A * A + B * B))

 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

```
exp = input()

exp_for_eval_template = exp.replace('=', '==')

ans = []

for a in range(2):
 for b in range(2):
  for c in range(2):
   exp_for_eval = exp_for_eval_template

   exp_for_eval = exp_for_eval.replace('a', str(a))
   exp_for_eval = exp_for_eval.replace('b', str(b))
   exp_for_eval = exp_for_eval.replace('c', str(c))

   res = int(eval(exp_for_eval))
   ans.append(res)

print(*ans, sep='')
```

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 to_dec_num(hex_str):
 return int(hex_str, 16)

def to_hex_str(dec_num):
 return hex(dec_num).upper()[2:]

def diff_mul(mul1, mul2):
 # Возвращает индекс ошибки записи в произведении, если такая ошибка одна, иначе -1
 len_mul1 = len(mul1)
 len_mul2 = len(mul2)
 cnt_wrongs = 0
 idx_wrong = -1

 if len_mul1 != len_mul2:
  return -1

 for i in range(len_mul1):
  if mul1[i] != mul2[i]:
   cnt_wrongs += 1
   idx_wrong = i

  if cnt_wrongs >= 2: return -1

 return idx_wrong


def define_rights_numbers(dec_num1, dec_num2, dec_mul):
 div1 = dec_mul / dec_num2 # потенциально правильное число под номером 1
 div2 = dec_mul / dec_num1 # потенциально правильное число под номером 2

 if int(div1) == div1:
  return to_hex_str(int(div1)), num2
 elif int(div2) == div2:
  return num1, to_hex_str(int(div2))
 else:
  assert 0
```

```
num1 = input()
num2 = input()
mul = input()

dec_num1 = to_dec_num(num1)
dec_num2 = to_dec_num(num2)
dec_mul = to_dec_num(mul)

if dec_mul == 0 and dec_num1 * dec_num2 != 0: # крайний случай
 if len(num1) != 1:
  num1 = 0
 else:
  num2 = 0

 print(num1)
 print(num2)
 print(mul)
 exit()


hex_mul_right = to_hex_str(dec_num1 * dec_num2) # правильное произведение

idx_wrong = diff_mul(mul, hex_mul_right)

if ~idx_wrong:
 print(num1)
 print(num2)
 print(hex_mul_right)
else: # mul и dec_mul правильные произведения

 # только одно число изменится
 new_num1, new_num2 = define_rights_numbers(dec_num1, dec_num2, dec_mul)

 print(new_num1)
 print(new_num2)
 print(mul)
```

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 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.

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.

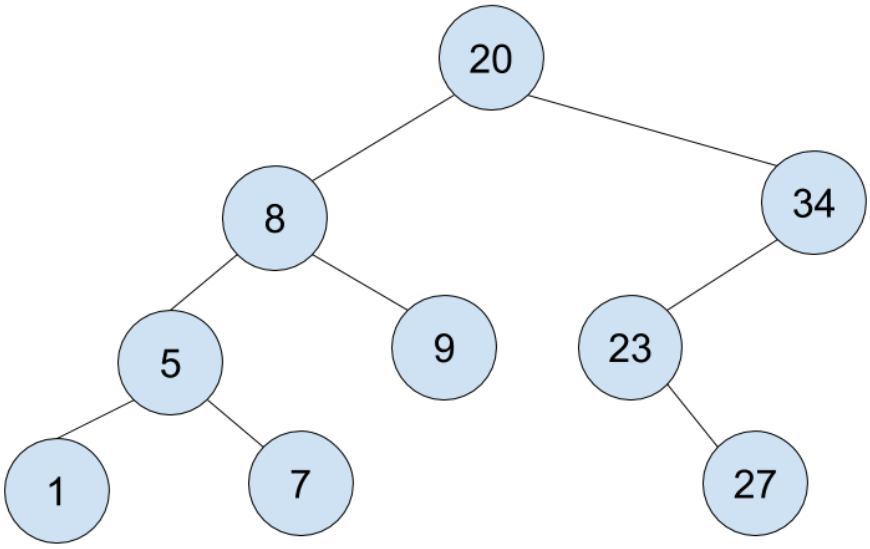
По заданным ключам вершин, полученным при прямом обходе, требуется определить, является ли дерево сбалансированным, а также определит высоту дерева.

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

Исходный код

```
def bin_search(arr, el):
 left = -1
 right = len(arr)

 while right - left > 1:
  middle = (left + right) // 2

  if arr[middle] < el:
   left = middle
  else:
   right = middle

 return left+1


def update_tree(node, node_height, child, graph_parents, vertex_data):
 if node > child:
  height_side = 0
 else:
  height_side = 1
 vertex_data[node][height_side] = node_height

 height1 = vertex_data[node][0]
 height2 = vertex_data[node][1]
 diff_height = abs(height1 - height2)

 real_height = max(height1, height2)
 vertex_data[node][2] = real_height
```

```
 if diff_height > 1:
  vertex_data[node][3] = 'unbalanced'
 else:
  vertex_data[node][3] = 'balanced'

 if graph_parents.get(node):
  parent = graph_parents[node]
  update_tree(parent, real_height + 1, node, graph_parents, vertex_data)
 else:
  return


def init_node(node, vertex_data):
 vertex_data[node] = [1, 1, 1, 'balanced']


sorted_vertex = []

graph_parents = {}
vertex_data = {}

root = node = int(input())
while True:

 init_node(node, vertex_data)

 node_idx = bin_search(sorted_vertex, node)
 sorted_vertex.insert(node_idx, node)

 parent = 0
 parent1 = parent2 = 0
 parent1_height = parent2_height = -1
 if node_idx - 1 >= 0:
  parent1 = sorted_vertex[node_idx - 1]
 if node_idx + 1 < len(sorted_vertex):
  parent2 = sorted_vertex[node_idx + 1]

 if parent1:
  parent1_height = vertex_data[parent1][1]
 if parent2:
  parent2_height = vertex_data[parent2][0]

 if parent1_height < parent2_height and parent1_height > -1:
  parent = parent1
 elif parent2_height > -1:
  parent = parent2


 if parent:
  if node != parent:
   graph_parents[node] = parent

  update_tree(parent, 2, node, graph_parents, vertex_data)


 node = input()
 if node == '.': break
 node = int(node)

is_balanced = True
for data in vertex_data.values():
 if data[3] == 'unbalanced':
  is_balanced = False
  break

if is_balanced:
 print('YES')
else:
 print('NO')

print(vertex_data[root][2])
```