2. ВТОРОЙ ЭТАП

Описание этапа

Второй отборочный этап проводится в командном формате в сети Интернет, работы оцениваются автоматически средствами системы онлайн-тестирования.

Продолжительность второго этапа составляет 52 дня. Задачи требуют от участников наличие навыков программирования, носят междисциплинарный характер и помогают отработать те навыки, которые потребуются для решения командной задачи заключительного этапа.

Участники не были ограничены в выборе языка программирования для решения задач.

Объем и сложность задач этого этапа подобраны таким образом, чтобы решение всех задач одним человеком было маловероятно. Это призвано обеспечить включение командной работы и распределения обязанностей. Решение каждой задачи дает определенное количество баллов. В части задач баллы зачисляются в полном объеме за правильное решение задачи, в другой части задач баллы начислялись за частичное решени. В данном этапе можно получить суммарно от 0 до 105 баллов.

Задачи выкладывались партиями: в начале второго этапа и затем через каждую неделю.

Команды могут выполнять задачи в любом порядке. Задачи допускают неограниченное число попыток сдать решение.

Задачи второго этапа

4.1. Блок задач 1

Задача 4.1.1. Предварительная подготовка изображения (3 балла)

Важным шагом в алгоритмах распознавания объектов на цифровых изображениях (фотографиях или кадрах видеопотока) является предварительная подготовка. Например, алгоритм определения лица с помощью гистограмм направленных градиентов (`https://bit.ly/2nBnUqS`) работает только с изображением в градациях серого.

Существует несколько способов перевести цветное изображение в градации серого цвета. Если исходить, что исходное изображение кодируется интенсивностями трех цветовых каналов: красным, зеленым и синим (R, G и B), то можно предложить следующие преобразования:

Среднее арифметическое

Яркость итогового *серого* пикселя вычисляется как среднее арифметическое трех цветовых каналов:

$$P = \frac{R + G + B}{3}$$

Но поскольку данная характеристика не соответствует человеческому восприятия цвета, то используются подходы, описанные ниже.

Средневзвешенное

Данные способ перевода используется при переводе из цветовой модели RGB в модель YUV (`https://ru.wikipedia.org/wiki/YUV`). Для получения свечения пикселя (Y) из компонент R, G и B вводятся коэффициенты. Поскольку сама цветовая модель YUV пришла из телевидения, *коэффициенты, используемые для расчета свечения, выбраны из соображений коррекции люминисцентов цветного монитора. Дело в том, что из базовых цветов, взятых в одинаковом количестве, человеческих глаз сперва выделяет зеленый, затем красный, а уже потом синий. Подразумевается, что когда зеленый и синий цвета излучаются монитором в одинаковом количестве, зеленый, тем не менее, выглядит ярче. Поэтому преобразование в градации серого цвета путем вычисления среднего арифметического цветовых компонент не отражает воспринимаемую человеком яркость оригинала. Для этого используется средневзвешенная величина.* (`http://www.linuxlib.ru/gimp/gimp/node150.html`)

$$Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

Поиск ближайшей точки на нейтральной оси

Если представить, цветовое пространство, кодируемое компонентами R, G и B в виде куба, то диагональ куба называют нейтральной осью (`http://www.linuxlib.ru/gimp/gimp/node148.html#razlichnijeproektsiinanejtralnujuos`). Тогда можно ввести понятие светимости, для определения которой выбирается ближайшая точка на нейтральной оси, соответствующая конкретным компонентам RGB.

$$L = \frac{\max(R, G, B) + \min(R, G, B)}{2}$$

Определение величины яркости

В цветовой модели HSV (`https://bit.ly/2qzgZNg`) компонента V соответствует величине яркости. Величина яркости определяется как максимальная интенсивость одной из компонент RGB:

$$V = \max(R, G, B)$$

Еще одним преобразованием, которое может использоваться перед применением алгоритмов распознавания, является уменьшение цифрового шума изображения (`https://bit.ly/2zxrEw7`). Для подавления шумов применяют различные алгоритмы, также именуемые "фильтры". При этом следует помнить, что применение шумоподавляющих фильтров всегда негативно сказывает на детализации изображения. Поэтому фильтры параметры фильтров необходимо тщательно подбирать под ту цель, которую необходимо достичь с их помощью.

Наиболее простыми фильтрами являются:

- фильтр, основанный на вычислении среднего геометрического (`https://en.wikipedia.org/wiki/Geometric_mean_filter`)
- медианный фильтр (`https://en.wikipedia.org/wiki/Median_filter`)

Оба этих фильтра оперируют плавающим окном - область пикселей вокруг вычисляемого на основе алгоритма фильтрации пикселя (включая его самого). Окно итеративно перемещается по кадру, обходя тем самым все пиксели кадра и вычисляя их новые значения.

Фильтр с вычислением среднего геометрического

Алгоритм фильтра перемножает все значения пикселей плавающего окна и затем вычисляет корень от полученного числа. Степень корня определяется размером плавающего окна.

Например, для окна размером 3×3 с содержимым, представленном на таблице ниже (значения яркости соответствующих пикселей), будет использоваться 9ая степень ($3 \cdot 3 = 9$).

	$x-1$	x	$x+1$
$y-1$	228	166	145
y	237	10	122
$y+1$	95	94	156

$$P_{x,y} = \sqrt[9]{228 \cdot 166 \cdot 145 \cdot 237 \cdot 10 \cdot 122 \cdot 95 \cdot 94 \cdot 156} = 109$$

Таким образом, значение яркости пикселя, находящегося по центру окна, заменяется на полученное значение.

При сдвиге окно продолжает оперировать изначальными, а не вычисленными заново значениями яркости пикселей. Вычисление значений крайних пикселей по периметру изображения не происходит.

При работе с цветным изображением закодированном с помощью модели RGB фильтр применяется по-отдельности к каждому каналу.

Медианный фильтр

В данном фильтре значения яркостей пикселей из плавающего окна сортируются, после чего берется значение из середины списка, полученного после сортировки.

Для значений пикселей, представленных в таблице выше, значение пикселя $P_{x,y}$ будет определяться следующим образом:

$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$	$i = 6$	$i = 7$	$i = 8$
10	94	95	122	145	156	166	228	237

$$P_{x,y} = 145$$

Напишите программу, которая бы подавляла шумы на изображении и после чего переводила бы его в градации серого.

Формат входных данных

В первой строке входного файла содержится два числа W и H - размер изображение по горизонтали и вертикали ($32 \leqslant W, H \leqslant 64$).

Последующая строка содержит $W \times H$ наборов шестнадцатиричных символов. Каждый набор - пиксель, описанный в модели RGB, где первые (старшие) два символа кодируют красную компоненту пикселя, следующие два символа кодируют зеленую компоненту, а последние (младшие) два символа кодируют синюю компоненту. Первые W наборов - это первая строка изображения, следующие W наборов - вторая строка и т.д.

Следующая строка определяет алгоритм подавления шума F, который должен быть применен к изображению. F может принимать следующие значения:

- 1 – фильтр, основанный на вычислении среднего геометрического;
- 2 – медианный фильтр.

Последняя строка задает алгоритм "обесцвечивания" изображение D. D может принимать следующие значения:

- 1 – использование среднего арифметического;
- 2 – использование средневзвешенное;
- 3 – использование ближайшей точки на нейтральной оси;

- 4 – определение величины яркости.

Формат выходных данных

Выведите в одной строке два целых числа - максимально темное и максимально светлое значения пикселей после проведенных преобразований - подавления шума и ”обесцвечивания”.

Примеры

Пример №1

Стандартный ввод

```
16 16
11211e 102015 132411 101d09 273126 323635 333439 363437
3f3b38 38342b 2f2822 1e1516 1f161b 302b28 333021 35331c
18211e 1b251d 182217 1a2215 1b2117 21201b 3e3936 4c413f
4e3f3c 63504c 4d3a36 402e2e 433535 312926 322f26 2e2f21
1a1915 1d1c1a 1c1a1b 21201e 35302a 443a30 756357 ae938a
d6b5b0 d9b3b2 c49e9d a78885 8d7972 524741 35312e 2f3032
291e18 312527 32252e 2e1f26 564442 baa497 d3b6a4 f3ccbd
ffd6d1 ffd3d6 fbc9c8 eabfb9 c2a49a 685750 221c1e 1f1d28
2f1e17 5d494a 8c777e 624851 cfb1b1 d1afa3 c79e8c d0a092
c28b86 edb5b4 f1bbb9 e6bab1 d6b4a8 816861 261619 392b38
29160f 99847f bea2a1 9b797a deb6b4 d1a49e b18177 a7756c
9e6d68 ab7b77 e3b8b1 b79188 927066 84665e 482a28 694c4e
2d1e19 746057 bea297 d7b1a6 eabbb5 e8b5b4 e6b2b4 b88886
ad847e c2a097 dabcb2 795a55 7a5954 7d564f 582e22 b18373
261b19 837069 dabfae e8c2af f1bfb4 ffd0d1 ffcfd5 f0c2c5
e2bfbb edd5cb e1cac2 947b77 876564 6b4037 93614a c28c6a
2e2329 7a6b66 cbb1a2 e3bda8 f1c0b1 ffc8c3 f4bdc0 cc9b9e
d9b3b2 cfb3af dcc4c2 a98f90 ae8c8b 956b5f 9b6a4c ba875c
4a3f4d c2b3b8 d2bbb5 e2c1b2 e8baab e5b1a4 d49d96 d29f9c
e8baba bc9397 896569 937172 ba9992 9a7764 94724f 9a7549
3e3344 bbadbc c8b3bc bfa3a0 e4beb1 e5b8a3 ce9b88 cf968d
d09597 bc848f 88565f 946c6c 9d8072 8a765b 8e805b 7a7148
32293e 5a4e64 d9cadd ae99a2 c2a7a0 cea996 d3a58e f1bcae
e7abab bd818b 966368 936e66 88775d 898965 808f68 627750
373445 2d283c 776f86 c1b4c5 998688 ab9085 ba9686 ebbfb4
f4c1be d6a4a7 b48d88 85705b 6e704b 758c60 6d9769 619167
1f2630 2b2f3b 323443 84808f b3a9b4 7a696f 876f6f 987b77
ab8c87 ac9186 8b7f69 616743 6b8558 6a9965 5a9964 4c925e
243336 243235 283138 363a46 9996a9 aca2ba 736678 594c53
4f493d 515437 576740 5e7e4f 689866 54955d 439155 4ba05f
1b2f2d 1c2f2b 263434 242d36 3e3e56 b6b0d2 bab3d4 7c7a88
4c5447 495d38 6d8f5c 6b9c65 52915c 509b62 449958 439f56
2
1
```

Стандартный вывод

```
18 203
```

Комментарии

При реализации подавления шума используйте плавающее окно 3×3. При получении не целых результатов - отбрасывайте дробную часть.

Решение

Для изображения применяем нужный фильтр, затем нужный алгоритм обесцвечивания. В итоговом изображении ищем максимальную и минимальную яркость.

Пример программы-решения

Ниже представлено решение на языке Python3

```
import numpy as np
from functools import reduce

first = lambda x: sum(x) // 3
second = lambda x: int(0.299 * x[0] + 0.587 * x[1] + 0.114 * x[2])
third = lambda x: (min(x) + max(x)) // 2
forth = lambda x: max(x)
first_filter = lambda i, j, k, picture: int((reduce(lambda x, y: x * y,
                            picture[i - 1:i + 2, j - 1:j + 2, k].flatten())) ** (1 / 9))
second_filter = lambda i, j, k, picture:
                             sorted(picture[i - 1:i + 2, j - 1:j + 2, k].flatten())[4]


def apply_filter(picture: np.ndarray, f):
    ans = picture.copy()
    for k in range(3):
        for i in range(1, picture.shape[0] - 1):
            for j in range(1, picture.shape[1] - 1):
                ans[i, j, k] = f(i, j, k, picture)
    return ans


def convert(picture, converter):
    return np.array(list(map(converter, picture.reshape((-1, 3)))))


w, h = map(int, input().split())
str_pixels = input()
filter_num = int(input())
converter_num = int(input())

a_pixels = map(lambda x:
            (int(x[:2], 16), int(x[2:4], 16), int(x[4:6], 16)), str_pixels.split())
a_pixels = list(a_pixels)
picture = np.array(a_pixels, dtype=object)
picture.resize((h, w, 3))

picture = apply_filter(picture, first_filter if filter_num == 1 else second_filter)
picture = convert(picture,
                  first if converter_num == 1 else second if converter_num == 2
                  else third if converter_num == 3 else forth)

print(min(picture), max(picture))
```

Задача 4.1.2. Гистограммы направленных градиентов (3 балла)

Гистограммы направленных градиентов (Histagrams of oriented gradients, HOG, `https://bit.ly/2nBnUqS`) используются для обнаружения объектов на изображении. Например, целиком человека (`https://bit.ly/2RE1eAi`) или его лица (`https://bit.ly/2zASMKI`).

Если кратко, то алгоритм построения гистограмм градиентов можно описать следующим образом

- Для каждого пикселя изображения определяется вектор (градиент), определяющий в какую сторону происходит изменение яркости;
- Векторы группируются в наборы ($X \times X$);
- Внутри каждого набора градиенты, направленные примерно в одну сторону, объединяются - получается несколько групп градиентов - гистограмма, по которой можно сказать какое направление преобладает у пикселей одного набора;
- Гистограммы градиентов нормализуются по яркости, чтобы сделать гистограмму независимой от яркости изначального изображения;
- Нормализованные гистограммы градиентов сверяются с шаблоном (возможно, с применением алгоритмов машинного обучения), чтобы определить наличие на изначальном изображении объекта, определяемого шаблоном.

Более подробно алгоритмы описаны на следующих ресурсах:

- Статья о вычислении векторов (градиентов): `http://mccormickml.com/2013/05/07/gradient-vectors/`
- Статья о построении гистограммы градиентов и нормализации: `http://mccormickml.com/2013/05/09/hog-person-detector-tutorial/`
- Пошаговое описание построения гистограммы градиентов для всей картинки: `https://www.learnopencv.com/histogram-of-oriented-gradients/`

Напишите программу, которая бы используя алгоритм построения гистограмм направленных градиентов, определала бы лежит или стоит человек на изображении.

Формат входных данных

В первой строке входного файла содержится два числа W и H - размер изображение по горизонтали и вертикали ($100 \leqslant W, H \leqslant 200$).

Последующая строка содержит $W \times H$ наборов шестнадцатиричных символов. Каждый набор - пиксель, описанный в модели RGB, где первые (старшие) два символа кодируют красную компоненту пикселя, следующие два символа кодируют зеленую компоненту, а последние (младшие) два символа кодируют синюю компоненту. Первые W наборов - это первая строка изображения, следующие W наборов - вторая строка и т.д.

Формат выходных данных

Выведите одно число:

- 0 – если человека на картинке нет
- 1 – если человек стоит
- 2 – если человек лежит

Примеры

Пример №1

`https://drive.google.com/file/d/11INAwaWig6HQ9UIZmp-jZRorBDdfQuZ6/view?usp=sharing`

Пример №2

`https://drive.google.com/file/d/1GQoS3CXXJHV7kDIs4HorJM-H5x_xQlIh/view?usp=sharing`

Пример №3

`https://drive.google.com/file/d/1p0JRD0sJyOgiIjR0cCMGnli3EZj4zvHL/view?usp=sharing`

Пример №4

`https://drive.google.com/file/d/1cRUC6OWawj0X-t4XEY_sfuAoDMwMCQAX/view?usp=sharing`

Комментарии

Если на изображении присутствует человек, то руки будут вытянуты вдоль тела, а ноги будут прямые.

Пол человека может быть любой. Фон будет в среднем однородный (трава, листья, обои, камень и т.п.). Никаких других крупных объектов, кроме человека на изображении не будет.

Решение

Пользуясь алгоритмами построения гистрограмм по ссылкам из условия, находим углы поворота получившихся векторов и сравниваем отношение горизонтальных и вертикальных градиентов, выводим ответ.

Пример программы-решения

Ниже представлено решение на языке Python3

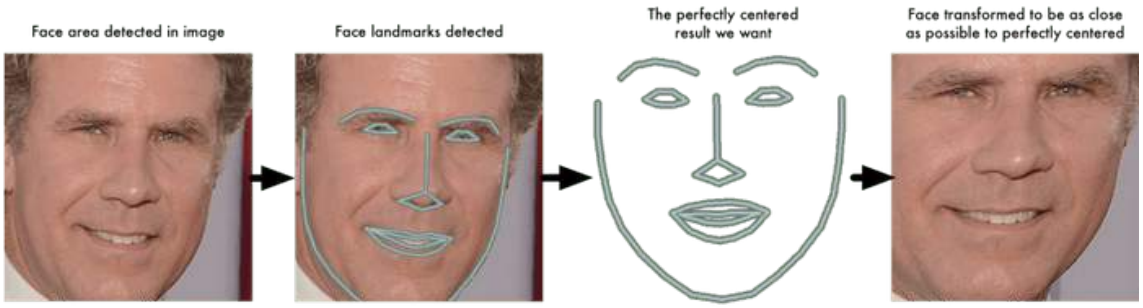
```python
import math

n, m = map(int, input().split())

arr = [None] * n
for i in range(m):
    arr[i] = [None] * m

for i in range(m):
    line = input().split()
    for j in range(n):
        r, g, b = int(line[j][:2], 16), int(line[j][2:4], 16), int(line[j][4:], 16)
        arr[j][i] = round((r + g + b) / 3)

w = 200
w8 = w // 8
h = 200
h8 = h // 8
nbins = 18
angle_per_bin = 2 * math.pi / nbins
angle_shift = angle_per_bin / 2

blocks = [None] * w8
for i in range(w8):
    blocks[i] = [None] * h8
    for j in range(h8):
        blocks[i][j] = [0] * nbins

blocks2 = [None] * w8
for i in range(w8):
    blocks2[i] = [None] * h8
    for j in range(h8):
        blocks2[i][j] = [0] * nbins

for i in range(w):
    for j in range(h):
        i1 = i - 1 if i > 0 else i
        i2 = i + 1 if i < w - 1 else i
        j1 = j - 1 if j > 0 else j
        j2 = j + 1 if j < h - 1 else j

        x = arr[i2][j] - arr[i1][j]
        y = arr[i][j2] - arr[i][j1]

        angle = 2 * math.pi + math.atan2(y, x)
        lower = (angle - angle_shift) // angle_per_bin
        upper = lower + 1

        mag = math.sqrt(x ** 2 + y ** 2)
        mag_upper = ((angle - angle_shift) % angle_per_bin) / angle_per_bin * mag
        mag_lower = mag - mag_upper

        blocks[i // 8][j // 8][int(lower % nbins)] += mag_lower
        blocks[i // 8][j // 8][int(upper % nbins)] += mag_upper

for i in range(w8 - 1):
    for j in range(h8 - 1):
        vector_len = 0
        for q in range(nbins):
            vector_len += blocks[i][j][q] ** 2
```

```
61              vector_len += blocks[i + 1][j][q] ** 2
62              vector_len += blocks[i][j + 1][q] ** 2
63              vector_len += blocks[i + 1][j + 1][q] ** 2
64          vector_len = math.sqrt(vector_len)
65
66          for w in range(nbins):
67              blocks2[i][j][w] = blocks2[i][j][w] + 0 if vector_len == 0 else\
68                  (blocks[i][j][w] / vector_len)
69              blocks2[i + 1][j][w] = blocks2[i + 1][j][w] + 0 if vector_len == 0 else\
70                  (blocks[i + 1][j][w] / vector_len)
71              blocks2[i][j + 1][w] = blocks2[i][j + 1][w] + 0 if vector_len == 0 else\
72                  (blocks[i][j + 1][w] / vector_len)
73              blocks2[i + 1][j + 1][w] = blocks2[i + 1][j + 1][w] + 0 if vector_len == 0\
74                  else (blocks[i + 1][j + 1][w] / vector_len)
75
76  hor, ver = 0, 0
77  for i in range(w8):
78      for j in range(h8):
79          ans = (0, 0)
80          for q in range(nbins):
81              ans = (ans[0] + blocks2[i][j][q] * math.cos(angle_per_bin * q
82              + angle_shift),
83                     ans[1] + blocks2[i][j][q] * math.sin(angle_per_bin * q
84                     + angle_shift))
85
86          if math.fabs(ans[0]) - math.fabs(ans[1]) > 0:
87              ver += 1
88          elif math.fabs(ans[1]) - math.fabs(ans[0]) > 0:
89              hor += 1
90
91  if hor / (ver + 1) > 1.3:
92      print(2)
93  elif ver / (hor + 1) > 1.3:
94      print(1)
95  else:
96      print(0)
```

Задача 4.1.3. Аффинные преобразования (3 балла)

Ранние алгоритмы распознования лиц требовали, чтобы распозноваемое лицо располагалось в положении "анфас" (`https://bit.ly/2yY7iwA`). Очевидно, что для такое условие можно соблюсти в тех случаях, когда есть возможность явно попросить распознаваемого встать определенным образом и смотреть в камеру. В большинстве случаев, применимых к реальной жизни, приходится работать с изображением лица, которое может быть повернуто относительно фокальной плоскости камеры.

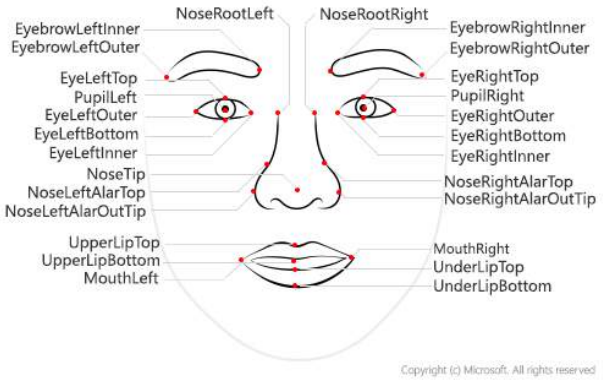
Для решения данной проблемы, предлагалось использовать афинные преобразования (`https://bit.ly/2QxNoz5`, `https://compgraphics.info/2D/affine_transform.php`).



После применения преобразований ключевые точки лица (например, кончик носа, уголки губ и глаз) располагались на тех местах изображения, где их мог бы корректно обработать следующий уровень алгоритма распознования (`https://bit.ly/2zICQbH`).

Microsoft Face API

Сервис распознования лиц *Microsoft Face API* (`https://azure.microsoft.com/en-us/services/cognitive-services/face/`) для каждого изображения лица, загруженного на сервис может вернуть набор данных в формате JSON, описывающий расположение на изображении ключевых точек лица.



Координаты ключевых точек из набора данных, возвращаемых *MS Face API*, указывают в каком месте изображения находится тот или иной элемент лица. Следовательно, если лицо повернуто, то, например, координата $X_{UnderLipBottom}$ нижней части нижней губы будет смещена влево или вправо относительно координаты $X_{NoseTip}$ кончика носа.

Напишите программу, которая бы определяла положение ключевых точек лица после применения афинных преобразований, если известно, что эталонные координаты ключевых точек *eyeLeftOuter* (внешний уголок левого глаза), *noseTip* (кончик носа)и *eyeRightOuter* (внешний уголок правого глаза) должны быть следующие:

eyeLeftOuter	(252, 331)
noseTip	(520, 634)
eyeRightOuter	(782, 321)

Формат входных данных

На вход приходит структура в формате JSON с координатами четырех ключевых точек лица. Среди них есть точно координаты для точек *eyeLeftOuter*, *noseTip* и

eyeRightOuter, а также четверта точка - одна из тех, что представлены на рисунке выше (возвращаемые *MS Face API*).

Формат выходных данных

Выведите два целых числа в одной строке - координату X и Y четвертой ключевой точки, подразумевая, что к ней применено такое преобразование, при котором точки *eyeLeftOuter*, *noseTip* и *eyeRightOuter* расположатся в эталонных координатах.

Примеры

Пример №1

Стандартный ввод
`[{"faceLandmarks": {"eyeLeftOuter": {"x": 224.1, "y": 199.0}, "noseTip": {"x": 286.9, "y": 254.6}, "eyeRightOuter": {"x": 355.2, "y": 195.6}, "noseRightAlarOutTip": {"x": 317.6, "y": 261.1}}}]`
Стандартный вывод
`645 670`

Решение

Преобразование задаётся уравнением вида $v^{'} = A * v$, где v - исходный вектор точек, $v^{'}$ - преобразованный вектор, A - матрица перехода. Зная исходные координаты некоторых исходных и преобразованных точек, можем найти матрицу: $A = v^{'} * v^{-1}$. Пользуясь матрицей перехода, применим её к нужным исходным точкам.

Пример программы-решения

Ниже представлено решение на языке Python3

```
import numpy as np
from json import loads

ideal_landmarks = np.array([
  [252, 520, 782],
  [331, 634, 321],
  [1, 1, 1]
])

actual_landmarks = loads(input())[0]['faceLandmarks']

eye_left_outer = actual_landmarks.pop("eyeLeftOuter")
nose_tip = actual_landmarks.pop("noseTip")
eye_right_outer = actual_landmarks.pop("eyeRightOuter")

other_landmark = actual_landmarks[list(actual_landmarks.keys())[0]]
other_landmark = np.array([
    [other_landmark['x']],
    [other_landmark['y']],
```

```
    [1]
])

actual_landmarks = np.array([
  [eye_left_outer['x'], nose_tip['x'], eye_right_outer['x']],
  [eye_left_outer['y'], nose_tip['y'], eye_right_outer['y']],
  [1, 1, 1]
])

t = np.matmul(ideal_landmarks, np.linalg.inv(actual_landmarks))

result_landmark = np.matmul(t, other_landmark)
x, y = result_landmark[0][0], result_landmark[1][0]

print(int(x), int(y))
```

Задача 4.1.4. Идентификация по лицу (3 балла)

Одним из подходов, используемым для распознования лиц, является измерение расстояний между ключевыми элементами лица - глазами, носом, ртом и т.п. Для этого между одними и теми же точками в эталонном и сравниваемом изображении считается расстояние (длина вектора). Если сумма таких расстояний большая, то, скорее всего, на изображениях лица двух разных людей.

При этом нужно помнить, что поскольку расстояние будет зависеть от масштаба изображения, а также от поворота лица относительно оптической оси камеры или фокальной плоскости, то исходный набор ключевых точек нужно нормализовать: центрировать, привести к одному масштабу и повернуть на один и тот же угол.

Напишите программу, которая бы по эталонному набору точек, описывающих лицо какого-то человека в формате *MS Face API*, определяла бы какой из трех дополнительных наборов ключевых точек лица принадлежит этому же человеку.

Формат входных данных

На вход приходит структура в формате JSON, содержащяя четрые набора ключевых точек в в формате *MS Face API*.

Формат выходных данных

Выведите одно число 0, 1 или 2, в зависимости от того какой из первых трех наборов ключевых точек описывает лицо того же человека, что и в послденем - четвертом наборе.

Примеры

Пример №1

Стандартный ввод

[{"faceRectangle":{"top":228,"left":119,"width":311,"height":311},
"faceLandmarks":{"pupilLeft":{"x":210,"y":306},"pupilRight":{"x":340,"y":318},
"noseTip":{"x":269,"y":377},"mouthLeft":{"x":202,"y":448},"mouthRight":
{"x":320,"y":457},"eyebrowLeftOuter":{"x":160,"y":269},"eyebrowLeftInner":{"x":
244,"y":274},"eyeLeftOuter":{"x":188,"y":309},"eyeLeftTop":{"x":211,"y":300},
"eyeLeftBottom":{"x":210,"y":317},"eyeLeftInner":{"x":231,"y":310},
"eyebrowRightInner":{"x":305,"y":282},"eyebrowRightOuter":{"x":389,"y":285},
"eyeRightInner":{"x":316,"y":318},"eyeRightTop":{"x":337,"y":310},
"eyeRightBottom":{"x":336,"y":327},"eyeRightOuter":{"x":359,"y":321},
"noseRootLeft":{"x":258,"y":313},"noseRootRight":{"x":295,"y":316},
"noseLeftAlarTop":{"x":242,"y":354},"noseRightAlarTop":{"x":301,"y":359},
"noseLeftAlarOutTip":{"x":224,"y":380},"noseRightAlarOutTip":{"x":313,"y":389},
"upperLipTop":{"x":263,"y":430},"upperLipBottom":{"x":262,"y":441},
"underLipTop":{"x":262,"y":452},"underLipBottom":{"x":260,"y":468}}},
{"faceRectangle":{"top":536,"left":393,"width":540,"height":540},
"faceLandmarks":{"pupilLeft":{"x":520,"y":661},"pupilRight":{"x":723,"y":665},
"noseTip":{"x":628,"y":781},"mouthLeft":{"x":511,"y":861},"mouthRight":{"x":
717,"y":864},"eyebrowLeftOuter":{"x":417,"y":630},"eyebrowLeftInner":{"x":578,
"y":612},"eyeLeftOuter":{"x":475,"y":663},"eyeLeftTop":{"x":519,"y":644},
"eyeLeftBottom":{"x":514,"y":684},"eyeLeftInner":{"x":557,"y":670},
"eyebrowRightInner":{"x":660,"y":612},"eyebrowRightOuter":{"x":806,"y":627},
"eyeRightInner":{"x":685,"y":674},"eyeRightTop":{"x":720,"y":649},
"eyeRightBottom":{"x":725,"y":686},"eyeRightOuter":{"x":756,"y":666},
"noseRootLeft":{"x":593,"y":679},"noseRootRight":{"x":649,"y":680},
"noseLeftAlarTop":{"x":584,"y":751},"noseRightAlarTop":{"x":659,"y":754},
"noseLeftAlarOutTip":{"x":552,"y":789},"noseRightAlarOutTip":{"x":689,"y":792},
"upperLipTop":{"x":626,"y":854},"upperLipBottom":{"x":625,"y":873},
"underLipTop":{"x":620,"y":877},"underLipBottom":{"x":619,"y":907}}},
{"faceRectangle":{"top":462,"left":313,"width":605,"height":605},
"faceLandmarks":{"pupilLeft":{"x":491,"y":587},"pupilRight":{"x":685,"y":616},
"noseTip":{"x":576,"y":732},"mouthLeft":{"x":444,"y":834},"mouthRight":{"x":
668,"y":861},"eyebrowLeftOuter":{"x":413,"y":549},"eyebrowLeftInner":{"x":552,
"y":557},"eyeLeftOuter":{"x":454,"y":588},"eyeLeftTop":{"x":492,"y":575},
"eyeLeftBottom":{"x":487,"y":601},"eyeLeftInner":{"x":520,"y":600},
"eyebrowRightInner":{"x":626,"y":573},"eyebrowRightOuter":{"x":776,"y":587},
"eyeRightInner":{"x":649,"y":615},"eyeRightTop":{"x":685,"y":604},
"eyeRightBottom":{"x":679,"y":630},"eyeRightOuter":{"x":712,"y":629},
"noseRootLeft":{"x":554,"y":612},"noseRootRight":{"x":619,"y": 618},
"noseLeftAlarTop":{"x":533,"y":684},"noseRightAlarTop":{"x":631,"y":695},
"noseLeftAlarOutTip":{"x":498,"y":724},"noseRightAlarOutTip":{"x":656,"y":744},
"upperLipTop":{"x":561,"y":817},"upperLipBottom":{"x":556,"y":841},
"underLipTop":{"x":552,"y":870},"underLipBottom":{"x":546,"y":902}}},
{"faceRectangle":{"top":217,"left":129,"width":258,"height":258},
"faceLandmarks":{"pupilLeft":{"x":185,"y":269},"pupilRight":{"x":267,"y":271},
"noseTip":{"x":224,"y":313},"mouthLeft":{"x":179,"y":355},"mouthRight":{"x":
258,"y":358},"eyebrowLeftOuter":{"x":149,"y":248},"eyebrowLeftInner":{"x":204,
"y":248},"eyeLeftOuter":{"x":172,"y":270},"eyeLeftTop":{"x":185,"y":266},
"eyeLeftBottom":{"x":186,"y":273},"eyeLeftInner":{"x":198,"y":269},
"eyebrowRightInner":{"x":247,"y":247},"eyebrowRightOuter":{"x":300,"y":254},
"eyeRightInner":{"x":252,"y":270},"eyeRightTop":{"x":265,"y":268},
"eyeRightBottom":{"x":266,"y":275},"eyeRightOuter":{"x":278,"y":273},
"noseRootLeft":{"x":214,"y":270},"noseRootRight":{"x":236,"y":270},
"noseLeftAlarTop":{"x":206,"y":295},"noseRightAlarTop":{"x":244,"y":295},
"noseLeftAlarOutTip":{"x":195,"y":312},"noseRightAlarOutTip":{"x":254,"y":314},
"upperLipTop":{"x":222,"y":341},"upperLipBottom":{"x":221,"y":349},
"underLipTop":{"x":219,"y":368},"underLipBottom":{"x":220,"y":379}}}]

Стандартный вывод

0

Решение

В решении требуется учитывать только те точки, которые незначительно меняются при улыбке и другой мимике. Такие как:

['eyeLeftInner','eyeRightInner','noseRootLeft', 'noseRootRight', 'eyeLeftTop', 'eyeLeftBottom', 'eyeRightTop', 'eyeRightBottom', 'mouthRight', 'mouthLeft', 'eyeLeftOuter', 'eyeRightOuter', 'eyebrowLeftOuter', 'eyebrowRightOuter', 'eyebrowLeftInner', 'eyebrowRightInner']

Далее выполняем нормализацию лиц, сравниваем Евклидово расстояние между соответствующими точками и выводим ближайшее.

Пример программы-решения

Ниже представлено решение на языке Python3

```
import json
import math

[a, b, c, x] = list(map(lambda x : x['faceLandmarks'], json.loads(input())))

keys = ['eyeLeftInner','eyeRightInner','noseRootLeft', 'noseRootRight', 'eyeLeftTop',
        'eyeLeftBottom', 'eyeRightTop', 'eyeRightBottom', 'mouthRight', 'mouthLeft',
        'eyeLeftOuter', 'eyeRightOuter', 'eyebrowLeftOuter', 'eyebrowRightOuter',
        'eyebrowLeftInner', 'eyebrowRightInner']


def check(a, x):
    length = 0
    unit1 = (a['eyeLeftInner']['x'] - a['eyeRightInner']['x']) ** 2 + (
                a['eyeLeftInner']['y'] - a['eyeRightInner']['y']) ** 2
    unit2 = (x['eyeLeftInner']['x'] - x['eyeRightInner']['x']) ** 2 + (
                x['eyeLeftInner']['y'] - x['eyeRightInner']['y']) ** 2
    for key in keys:
        for keyx in keys:
            length += (math.sqrt(((a[key]['x'] - a[keyx]['x']) ** 2 +
                                  (a[key]['y'] - a[keyx]['y']) ** 2) / unit1)
                       - math.sqrt(((x[key]['x'] - x[keyx]['x']) ** 2 +
                                    (x[key]['y'] - x[keyx]['y']) ** 2) / unit2)) ** 2

    return length


ax, bx, cx = check(a, x), check(b, x), check(c, x)

if ax < bx and ax < cx:
    print(0)
elif bx < cx:
    print(1)
else:
    print(2)
```

4.2. Блок задач 2

Задача 4.2.1. Язык структурированных запросов (3 балла)

С самых ранних времен финансовые отношения всегда ассоциировались с информацией. Объемы информации росли, уже невозможно было использовать только бухгалтерскую книгу (гроссбух, ledger). А затем, товаро-денежные отношения между участниками усложнились, что выборка информации, определение цен, подведение итогов и т.п. стала невозможна вручную. Между тем, типовые операции с данными, выполняемые в рамках таких отношений, демонстрировали очевидность их автоматизации.

Сначала появились реляционные базы данных (`https://bit.ly/2IZvBjL`). А затем для работы с данными в базах данных с реляционной моделью был придуман и стандартизирован язык структурированный запросов (structured query language, SQL, `https://bit.ly/2kip6IQ`).

С точки зрения языка SQL данные хранятся в таблицах. Команды языка SQL позволяют управлять таблицами, изменять их содержимое, читать данные из таблиц.

Например, команда

```
SELECT * FROM Payments;
```

выведет все строки (записи) из таблицы `Payments`, а команда

```
SELECT Sender, Receiver FROM Payments WHERE (Value > 50);
```

выведет только отправителя и получателя для тех записей, где сумма платежа больше 50.

Ознакомьтесь с основами языка SQL в курсе "Базы данных" (`https://stepik.org/2614`, модули 4 - 8) и найдите решение к следующей задаче:

В реляционной базе данных есть две таблицы. В одной таблице `transactions` указаны транзакции пользователей (три столбца: `sender`, `recipient`, `val`). Другая таблица (`result`) используется для промежуточного хранения результата (один столбец: `res`).

Вставьте в таблицу `result` только одну строку, в которой будет подсчитан конечный баланс пользователя *"Frank"*. Начальный баланс пользователя считать 0.

Комментарии

Нужно нужно написать такую одну сложносоставную команду на языке SQL, которая бы как запрашивала нужные данные из таблицы `transactions`, так и вставляла бы данные в таблицу `result`. Из-за особенностей проверки задач на SQL-запросы платформы Stepik, даже правильные ответы будут проверяться на полное решение. Т.е. ответы подобные `INSERT INTO result (res) VALUES (число);` приниматься не будут.

Примеры

Пример №1

Чтобы попрактиковаться в составлении запроса, вставьте следующий код в онлайн-консоль для тестирования SQL команд `https://www.tutorialspoint.com/execute_sql_online.php`:

```
BEGIN TRANSACTION;

CREATE TABLE transactions (sender VARCHAR(20), recipient VARCHAR(20), val INT);
INSERT INTO transactions (sender, recipient, val) VALUES ('Alice', 'Frank', 10);
INSERT INTO transactions (sender, recipient, val) VALUES ('Alice', 'Frank', 20);
INSERT INTO transactions (sender, recipient, val) VALUES ('Clare', 'Frank', 5);
INSERT INTO transactions (sender, recipient, val) VALUES ('Clare', 'Alice', 7);
INSERT INTO transactions (sender, recipient, val) VALUES ('Frank', 'Alice', 3);

CREATE TABLE result (res INT);
COMMIT;

/* Данный запрос должен быть изменен на тот, что будет использоваться в
   Stepik в качестве ответа */
INSERT INTO result (res) VALUES(32);
/* Конец запроса для Stepik */

SELECT * FROM result;
```

Исполните данный запрос, нажав кнопку ”Execute”.

Затем измените команду `INSERT INTO result (res) VALUES (32);` на ту, что будете использовать в качестве ответа.

Задача 4.2.2. Обработка данных (3 балла)

Несмотря на относительную простоту SQL-запросов, вряд ли обычный бухгалтер, финансист или аналитик должны знать его, чтобы получать и обрабатывать данные, хранящиеся в реляционных базах данных. Огромное количество разработчиков программного обеспечения ежедневно работат на инструментарием, который бы позволял получать доступ к данным. Следовательно, нужно не только изучить непосредственно язык структурированных запросов, но и овладеть способами работы с базами данных из пользовательских приложений.

Одним из самых простых инструментов, позволяющих реализовать реляционную базу данных и обращаться к ней посредством SQL запросов, - это система управления базами данных (СУБД) SQLite (`https://ru.wikipedia.org/wiki/SQLite`). Все, что необходимо для использования SQLite в пользовательском приложении, - файл с данными и библиотека для соответствующего языка. Не нужно специально настраивать какой-то сервер базы данных, создавать на сервере разные базы для разных нужд - в SQLite базу данных создать также просто, как создать обычный файл.

Подержка SQLite есть для многих популярных языков программирования. Например, API для Python подробно описано здесь `https://docs.python.org/3/library/sqlite3.html` (есть перевод на русский язык, но, возможно, устарел - `http://john16blog.blogspot.com/2011/03/python-sqlite3.html`).

Еще одним полезным навыком для специалиста работающего с данными является умение понять связь данных между собой в базах данных (особенно, подвергшихся нормализации высокого уровня `https://bit.ly/1pH8FGt`), где нет четкого описания или документации, какие таблицы и каким образом объединяют между собой данные. Поняв связь, из набора чисел можно извлечь полезные данные.

Представьте, у вас есть доступ к базе данных, и все что вы про нее знаете, что в ней есть информация о пользователях, магазинах, начальных балансах пользователей и транзакциях, описывающих траты пользователей в магазинах и переводы пользователей друг другу. Никакой другой документации, описывающей связи в базе данных не сохранилось.

Вам, как разработчику приложения для магазинов, необходимо написать модуль определения наличия скидки для покупателей. А именно, вашему модулю нужно ответить на вопрос: в скольких магазинах покупатель A будет получать скидки, если магазины, предоставляющие скидки требуют, чтобы на балансе покупателя в текущий момент было не меньше X условных денежных единиц, и покупатель до этого совершил не меньше Y покупок в конкретном магазине.

Формат входных данных

В первой строке входных данных содержится имя покупателя A. Во второй строке - требования к балансу покупателя - количество необходимых денежных единиц X, заданное целым числом. Третья строка содержит требования по количеству покупок (Y) в каждом из искомых магазинов.

Последняя строка - ссылка на Google Drive, где необходимо скачать файл, описывающий базу данных в формате SQLite.

Формат выходных данных

В ответе необходимо предоставить только одно целое число - количество магазинов, где данный покупатель может получить скидки.

Примеры

Пример №1

Стандартный ввод

`customer71`
`4945`
`6`
`https://drive.google.com/open?id=1s3kyPQrKxd9fIOIQpmGpoYIBQYcHtNgR`

Стандартный вывод
`17`

Решение

Для того, чтобы решить задание, нужно ответить на вопросы:

1. Конечный баланс пользователя A меньше требуемого значения X? Если баланс меньше X, ответ равен 0; Если не меньше, то решением будет ответ на вопрос №2.
2. В скольки магазинах пользователь A совершил не менее Y покупок?

Для взаимодействия с базой данных воспользуемся модулем sqlite3.

Пример программы-решения

Ниже представлено решение на языке Python3

```
import sqlite3

customer = input()
min_balance = int(input())
min_buys = int(input())

conn = sqlite3.connect('database.db')
cur = conn.cursor()

# получаем начальный баланс пользователя
cur.execute('''
SELECT balance
FROM balances
JOIN customers ON balances.id = customers.id
WHERE customers.name = '{}';
'''.format(customer))
balance = cur.fetchall()[0][0]

# находим количество денежных едениц, полученных пользователем
cur.execute('''
SELECT SUM(amount) as recieved
FROM transactions
JOIN customers ON transactions.[to] = customers.id
WHERE customers.name = '{}';
'''.format(customer))
received = cur.fetchall()[0][0]

# находим количество денежных едениц, потраченных пользователем
cur.execute('''
SELECT SUM(amount) as spent
FROM transactions
JOIN customers ON transactions.[from] = customers.id
WHERE customers.name = '{}';
'''.format(customer))
spent = cur.fetchall()[0][0]

# balance + received - spent = конечный баланс пользователя
if (balance + received - spent) < min_balance:
    print(0)
else:
    # получаем выборку магазинов,
    # в которых пользователь customer совершал покупки не менее min_buys раз
    cur.execute('''
    SELECT shops.name, COUNT(*) AS n_buys
    FROM transactions
    JOIN customers ON transactions.[from] = customers.id
    JOIN shops ON transactions.[to] = shops.id
```

```
48      WHERE customers.name = '{}'
49      GROUP BY [from], [to]
50      HAVING n_buys >= {};
51      '''.format(customer, min_buys))
52      # находим количество строк в выборке,
53      # что и является искомым числом магазинов
54      n_buys = len(cur.fetchall())
55      print(n_buys)
```

4.3. Блок задач 3

Задача 4.3.1. Простая обработка видео-потока (3 балла)

Современные системы распознавания лиц работают как со статическими изображениями, когда делается снимок, с которым работает система, так и с видео потоком, когда извлекаются кадры для распознавания.

Статические системы могут применяться там, где есть дополнительная проверка на подлинность человека - оператор банка или кассир могут удостовериться, что перед ними живой человек, а не фотография или манекен.

Системы, работающие с видео потоками, используются там, где нет возможности применять дополнительные проверки оператором-человеком, вместо этого алгоритм автоматически должен распознавать факты мошенничества.

Для выполнения следующего задания - рекомендация - познакомиться с сервисом *MS Face API* (`https://docs.microsoft.com/ru-ru/azure/cognitive-services/face/overview`). Данный сервис работает через REST API. Это значит, что вся обработка и распознавание изображения происходит в облаке Azure, а программисту необходимо загружать туда данные или получать результаты, используя HTTP POST запросы.

Сервис MS Face API может следующее:

- **Обнаружение лиц** — выявляет лица на изображениях и возвращает координаты прямоугольника, в котором они расположены, извлекает ряд атрибутов, связанных с лицом, например поза, пол, возраст, положение головы.
- **Проверка лиц** — оценивает, принадлежат ли два лица одному человеку.
- **Поиск похожих лиц** — сравнивает целевое лицо и набор потенциальных лиц для поиска и находит небольшое количество лиц, очень похожих на целевое.
- **Группировка лиц** — делит неизвестные лица на несколько групп, основываясь на сходстве.
- **Идентификация личности** — позволяет идентифицировать новое обнаруженное лицо путем сравнения с базой данных заранее загруженных групп лиц.

Напишите программу, которая бы позволяла выполняла обработку отдельных кадров видео-файла с фотографиям лиц.

Формат входных данных

Ссылка на Google Drive, где необходимо скачать видео-файл.

Формат выходных данных

Целое число — сколько раз сменялись фотографии людей.

Примеры

Пример №1

Стандартный ввод
`https://drive.google.com/open?id=16nKi8GgNgLLRd_Z0JjYMjBsaJi3Hp8oz`

Стандартный вывод
`190`

Комментарии

Платформа Stepik позволяет отслеживать попытки решать задачу перебором. Такие попытки будут приводить к нулевым баллам за данную задачу.

Решение

Несмотря на то, что рекоммендовано воспользоваться MS Face API, есть способ проще, который не требует обращений к сторонним сервисам. Необходимо для каждого кадра сформировать хэш, который является его численным представлением и далее сравнивать эти хэши, чтобы найти разные кадры в видеоряде. Даже если два кадра на видео выглядят одинаковыми, они могут содержать различающиеся пиксели, в силу специфики работы алгоритмов сжатия видео, например mp4. Значит, обычные криптографические алгоритмы не смогут нам помочь в решении данной задачи, так как они очень чувствительны к изменениям входных данных, и будут возвращать абсолютно разные хэши, даже если два изображения имеют всего один различающийся бит. Поэтому, мы воспользуемся перцептивными хэшами, а именно алгоритмом хэша по среднему (average hash). Узнать больше про такие алгоритмы можно здесь: `https://habr.com/post/120562/`, `https://www.pyimagesearch.com/2017/11/27/image-hashing-opencv-python/`, `www.hackerfactor.com/blog/index.php?/archives/529-Kind-of-Like-That.html`.

Для работы с видеорядом, воспользуемся модулем cv2.

Пример программы-решения

Ниже представлено решение на языке Python3

```
import cv2
import numpy as np


# алгоритм хэширования по среднему,
# на вход подается массив пикселей в виде (r, g, b),
# на выходе получается число, являющееся цифровым представлением изображения
```

```
def average_hash(image):
    resized_grey = cv2.cvtColor(cv2.resize(image, (12, 12)), cv2.COLOR_BGR2GRAY)
    pixels = np.asarray(resized_grey)
    average = pixels.mean()
    difference = pixels > average
    return sum([2 ** (i % 8) for i, v in enumerate(difference.flatten()) if v])


vidcap = cv2.VideoCapture('1.mp4')
success, image = vidcap.read()
count = 0
prev_hash = average_hash(image)
while success:
    new_hash = average_hash(image)
    # если хэши отличаются, значит на кадре уже другое изображение
    if prev_hash != new_hash:
        count += 1
    prev_hash = new_hash
    success, image = vidcap.read()
print(count)
```

Задача 4.3.2. Анализ видео-потока (3 балла)

Данная задача отличается от предыдущей только тем, что в ней необходимо придумать способ определения одинаковых фотографий с лицами в видео ряде.

Формат входных данных

Ссылка на Google Drive, где необходимо скачать видео-файл.

Формат выходных данных

Целое число — сколько различных фотографий людей во входном файле.

Примеры

Пример №1

Стандартный ввод
https://drive.google.com/open?id=1SVXlpDhwMKN2oDAes0u47vkQs4sKt7yf

Стандартный вывод
110

Комментарии

Платформа Stepik позволяет отслеживать попытки решать задачу перебором. Такие попытки будут приводить к нулевым баллам за данную задачу.

Решение

Несмотря на то, что рекоммендовано воспользоваться MS Face API, есть способ проще, который не требует обращений к сторонним сервисам. Для сравнения двух кадров в этой задаче воспользуемся простым алгоритмом, который сравнивает 100 случайных пикселей в двух изображениях.

Пример программы-решения

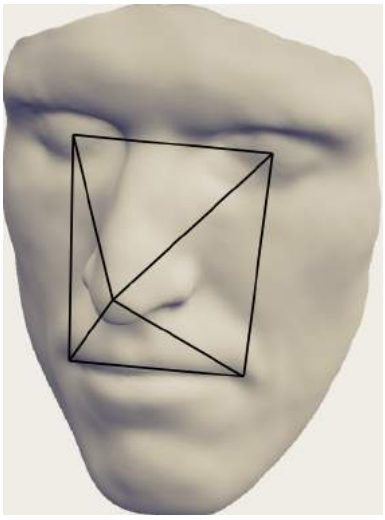
Ниже представлено решение на языке Python3

```
import cv2
from random import randint
import numpy as np


def image_diff(image1, image2, threshold=0):
    difference = 0
    for i in range(100):
        x = randint(0, image1.shape[0] - 1)
        y = randint(0, image1.shape[1] - 1)
        z = randint(0, image1.shape[2] - 1)
        difference += abs(image1[x, y, z] - image2[x, y, z]) > threshold
    return difference


vidcap = cv2.VideoCapture('154.mp4')
success, image = vidcap.read()
prev_image = image

frames = []
old_frame = 0
unique_images = []
last_image = 0
while success:
    success, image = vidcap.read()
    # добавляем корректную обработку последнего изображения
    if not success:
        image = np.zeros(prev_image.shape, np.uint8)
    frames.append(image)

    diff = image_diff(prev_image, image)
    # если кадры различаются более чем на 30%, значит на кадре уже другое изображение
    if diff > 30:
        unique_found = True
        # для того, чтобы сравнивать изображения, возьмем самый стабильный кадр
        stable_image = frames[(len(frames) + old_frame) // 2]
        old_frame = len(frames)
        # проверяем, есть ли в массиве уникальных кадров данное изображение
        for unique_image in unique_images:
            diff = image_diff(stable_image, unique_image, 3)
            if diff < 50:
                unique_found = False
                break
        if unique_found:
            unique_images.append(stable_image)
    prev_image = image

print(len(unique_images))
```

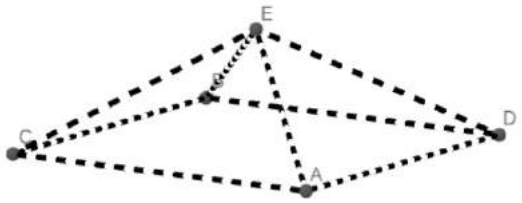
Задача 4.3.3. Угол поворота лица (3 балла)

Не всегда задача распознования лица сводится только к определению того, как расположены его части: нос, глаза, губы, брови. Иногда бывает необходимо ответить на вопрос - куда повернуто лицо. Например, таким образом можно сказать в какую сторону смотрит человек или общаются ли два человека между собой.

Если взять некоторые точки на лице человека (например, глаза, уголки губ, нос), то приняв их за вершины, можно построить многогранник.

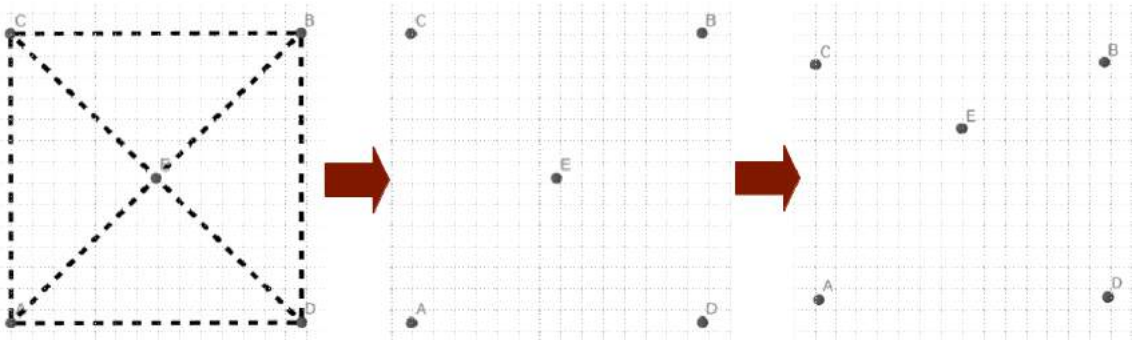


Для простоты, допустим, что многогранник - пирамида, в основании которой - квадрат.



Будем счиать, что если лицо направлено в камеру, то основание пирамиды лежит в плоскости, параллельной фокальной плоскости камеры (параллельной плоскости светочувствительной матрицы). Очевидно, что проекция основания пирамиды на кадр - квадрат, а вершина пирамиды будет лежать на пересечении диагоналей квадрата.

Тогда, если человек повернет свое лицо, например, вверх, то проеция пирамиды на кадр изменится: квадрат может превратиться в просто многоугольник, а вершина пирамиды больше не будет лежать на диагоналях многугольника.



Напишите программу, которая бы по проекции правильной пирамиды на кадр могла бы определить на какой угол α она была повернута.

Формат входных данных

В первой строке - целое число - высота h правильной пирамиды в миллиметрах. В следующих пяти строках по два целых числа x_i, y_i - координаты проекции пирамиды на плоскость кадра. Для простоты считать, что расстояние между центрами соседних пикселей - 1 миллиметр. Порядок следования вершин - неопределен.

Формат выходных данных

Введите одно число - угол поворота пирамиды относительно оси, проходившей до поворота через вершину пирамиды и центр квадрата ее основания. Угол укажите в радианах с точностью до сотых.

Примеры

Пример №1

Стандартный ввод
510 489 571 870 14 609 387 965 489 393 97
Стандартный вывод
0.23

Решение

Находим основания пирамиды, на их основе вычисляем центр пирамиды и находим отклоение от высоты. Зная противолежащий угол и гипотенузу, можем вычислить арксинус нужного угла.

Пример программы-решения

Ниже представлено решение на языке Python3

```python
import math

# Reading parameters
epsilon = 3
height = float(input())
coords = {}
letters = ['A', 'B', 'C', 'D', 'E']
for i in letters:
    coords[i] = [float(k) for k in input().split()]

# Creating all possible vectors

vectors = {}
for i in letters:
    for j in letters:
        if i == j:
            continue
        vectorCoords = [coords[j][0] - coords[i][0], coords[j][1] - coords[i][1]]
        vectors[i + j] = vectorCoords

# Extracting base points

found = False
for i in vectors:
    for j in vectors:
        if i == j:
            continue
        if (abs(vectors[i][0] - vectors[j][0]) <= epsilon) and (abs(vectors[i][1]
                                                    - vectors[j][1]) <= epsilon):
            basePoints = [i[0], i[1], j[0], j[1]]
            found = True
            break
    if found:
        break

# Guessing the H point

for k in letters:
    if k not in basePoints:
        heightCoords = [coords[k][0], coords[k][1]]
        break

# Calculating centroid of the base

#             ********** <- anotherPoint
#             *        **
#             *      * *
#             *     *  *
#             *    *   *
#             *   *    *
#             *  *     *
#             * *      *
# onePoint -> **********

onePoint = coords[basePoints[0]]
anotherPoint = coords[basePoints[3]]
centroidPoint = [(onePoint[0] + anotherPoint[0]) / 2,
                     (onePoint[1] + anotherPoint[1]) / 2]

```

```
60  # Calculating distance between centroid and height
61
62  x = heightCoords[0] - centroidPoint[0]
63  y = heightCoords[1] - centroidPoint[1]
64  heightProjection = math.hypot(x, y)
65
66  # Calculating the angle
67
68  alpha = math.asin(float(heightProjection / height))
69
70  # Returning answer
71
72  print(alpha)
```

4.4. Блок задач 4

Задача 4.4.1. Двойная идентификация (3 балла)

Специалист, разрабатывающий программное обеспечение, описывает процесс получения наличных денег в банкомате следующим образом:

1. Банкомат с самого начала проверяет PIN-код владельца карты. Если PIN-код совпадает, то переходим к следующему пункту.
2. Пользователь запрашивает определенную сумму.
3. Банкомат делает запрос к банку, который ответственный за установку данного банкомата.
4. Банк связывается с платежной системой (VISA, MasterCard, МИР) и передает ей номер карты, по которому определяется банк, выпустивший карту.
5. Платежная система устанавливает сессию с авторизационным узлом, банка, который выпустил карту.
6. Авторизационный узел, поскольку имеет доступ к базе банка, определяет для данной карты счет, остаток на счету и шлет ответ, можно ли выдать запрошенную сумму или нет.

В случае, если карта того же банка, что и банкомат, то запроса к платежной системе не происходит.

Если карта магнитная или не работает считыватель чипа карты, то PIN-код в зашифрованном виде (pin block, `https://en.wikipedia.org/wiki/PIN_pad`) передается в авторизационный узел. При этом важно заметить, что время пребывания PIN-кода в памяти устройства, считывающего код должно быть минимально, чтобы избегать возможности взлома и поиска PIN-кода в памяти.

Представим, что есть необходимость сделать банкомат для блокчейн сети на базе Ethereum, который бы производил идентификацию пользователей в сети блокчейн с помощью лица. Очевидно, что в этом случае должна быть база данных, которая бы ставила в соответствие некоторый идентификатор человека, опознанного с помощью камеры, и приватного ключа, позволившего бы данному человеку авторизовывать себя в блокчейн сети.

Хранение приватного ключа на каком-то сервере - не очень хорошая идея. Поэтому необходим способ, который бы из идентификатора человека мог бы получить

приватный ключ. При этом злоумышленник, даже зная алгоритм и идентификатор, не мог бы произвести подобное преобразование. Тогда в алгоритме должно использоваться что-то, что не знает злоумышленник, но знает пользователь. Таким образом, снова появляется необходимость использовать PIN-код: пользователь идентифицирует себя с помощью камеры и распознавания лица, но авторизируется на доступ к своим средствам только с помощью PIN-кода.

В качестве примера, рассмотрим следующий алгоритм генерации приватного ключа K:

$$K = keccak256(keccak256(keccak256(keccak256(keccak256(''), I, P_1), I, P_2), I, P_3), I, P_4)$$

где '' - "пустая" последовательность байт, I — это идентификатор, который возвращает система распознавания по лицу, приведенный к длине в 16 байт, а (P_1, P_2, P_3, P_4) — последовательно введенные четыре цифры PIN-кода, где каждая цифра представлена целым числом длиной 1 байт, P_1 — цифра самого старшего разряда в PIN-коде (первая введенная цифра), а P_4 — цифра самого младшего разряда в PIN-коде (последняя введенная цифра).

Напишите программу, которая бы по идентификатору человека и PIN-коду получала бы баланс счета в сети Sokol (тестовая сеть, совместимая с Ethereum Virtual Machine), принадлежащего данному человека.

Формат входных данных

Первая строка содержит последовательность из 36 шестнадцатиричных символов – идентификатор пользователя в виде UUID (Universally Unique Identifier, `https://ru.wikipedia.org/wiki/UUID`).

Вторая строка содержит четырехзначное число - PIN-код, необходимый для доступа к счету пользователя.

Формат выходных данных

Введите одно число - баланс счета пользователя в Wei.

Примеры

Пример №1

Стандартный ввод
`447b017b-317b-4568-b456-a37c4b905870` `9570`
Стандартный вывод
`173259686473389`

Комментарии

Если вам неизвестны концепции приватного ключа и адреса сети Ethereum и баланса счета, то обратитесь к следующим задачам второго этапа профиля "Программная инженерия финансовых технологий" сезона 2016-2017 годов:

- Криптография с открытым ключом, `https://stepik.org/lesson/59926/step/3`
- Адреса Ethereum, `https://stepik.org/lesson/59926/step/4`
- Получение баланса, `https://stepik.org/lesson/62023/step/2`

Для конкатенации данных перед хэшированием используйте тот же подход, что используется в языке Solidity при вызове `abi.encodePacked` (`https://bit.ly/2EqKxES`).

Получение информации из тестовой сети Sokol может происходить без необходимости синхронизировать свой собственный узел сети. Вместо этого можно отправлять JSON-RPC запросы на URL: `https://sokol.poa.network`.

Решение

Для решения задачи воспользуемся библиотеками *web3* и *pyethereum*.

Вычислим итеративно приватный ключ используя функцию *web3.sha3*. Для получения конечного адреса используем функцию *privtoaddr* из библиотеки *ethereum.utils*.

Подключимся к сети Sokol, используя *HTTPProvider*, запросим баланс с помощью функции *web3.eth.getBalance*.

Пример программы-решения

Ниже представлено решение на языке Python3

```
from ethereum.utils import privtoaddr

from web3 import Web3

web3 = Web3(Web3.HTTPProvider('https://sokol.poa.network'))
p_size = 1

uuid = 'f4a0a066-527e-4b44-92c4-b1c71fb3d0ce'
pin = '7530'

id = web3.toBytes(hexstr='0x' + uuid.replace('-', ''))

a = web3.sha3(bytes(0))
b = web3.sha3(a + id + bytes([int(pin[0])]))
c = web3.sha3(b + id + bytes([int(pin[1])]))
d = web3.sha3(c + id + bytes([int(pin[2])]))
e = web3.sha3(d + id + bytes([int(pin[3])]))

addr = web3.toChecksumAddress(privtoaddr(e).hex())

print(web3.eth.getBalance(addr))
```

Задача 4.4.2. UTXO в смарт-контракте (3 балла)

Модель UTXO (Unspent Transaction Output) известна тем, что используется в блокчейн Bitcoin. Все Bitcoin транзакции образуют связные списки: каждая тран-

закция, которая списывает средства со счета пользователя явно связана с другой транзакцией, в рамках которой эти средства на счете появились. При этом, очевидно, что транзакция начисляющая на счет средства, будет одновременно и транзакцией списывающей с какого-то счета средства. Исключением являются, так называемые, coinbase транзакции - транзакции начисляющие на счет пользователя средства, полученные в качестве вознаграждения за выпуск (майнинг) нового блока.

Поскольку для формирования данных одной транзакции на списание средств используется идентификатор другой транзакции, начислявшей средства, а данные полученной транзакции используются для вычисления хэша транзакции - ее идентификатора, то получается такой связный список, изменение элементов которого задним числом приведет к потере связности. Т.е. помимо связи на уровне блоков в блокчейн Bitcoin имеется связь на уровне транзакций.

Если транзакция, начисляющая средства на счет, еще не была использована для организации связного списка, а иными словами, даныые средства еще не списывались, то такая транзакция называется непотраченная (unspent). А имея в виду то, что внутри одной списывающей транзакции (input) можно начислять средства сразу на несколько счетов (каждое такое начисление - output), то список всех непотраченных начислений называет - непотраченные выходы траназкций (unspent transaction output).

Подробнее об входах и выходах в транзакциях Bitcoin можно почитать в книге "Mastering Bitcoin":

- Основы транзакций в Bitcoin, `https://bit.ly/2SKOOHs`
- Детальное описание транзакций, `https://bit.ly/2QRlIIQ`
- Про coinbase транзакции, `https://bit.ly/2SKoBJ8`

Несмотря на сложности в виде необходимости обхода всей цепочки блоков для подсчета баланса счета, у UTXO модели есть ряд преимуществ:

- Благодаря выстроенным цепочкам можно отслеживать какие средства откуда пришли и куда ушли. Даже не расскрывая принадлежность Bitcoin адресов частным лицам или организациям можно отслеживать и анализировать перемещение средств.
- Отдельные непотраченные выходы могут быть помечены, как использующиеся в платежных каналах (Lightening Network), а следовательно можно с одного аккаунта совершать параллельные оффчейн платежи без опасений двойных трат.

Технология, на которой построен блокчейн Ethereum, не подразумевает использование UTXO модели. Вместо этого используется модель состояний, когда перевод средств со счета на счет или изменение данных контракта формируют новое состояние конкретного аккаунта в частности и всего блокчейна в целом.

Тем не менее, UTXO модель можно реализовать в ценностях, которые строятся на основе смарт-контрактов Ethereum, иногда такие ценности называют токенами. Тогда преимущества, перечисленные выше, будет применимы и к таким токенам.

Ниже, представлен код на языке Solidity для токена, использующего модель UTXO.

```
pragma solidity ^0.5.1;
```

```
contract UTXOBasedToken {
    event Transfer(bytes32 indexed tx_source, bytes32 indexed tx_address,
                   address indexed recipient, uint256 value, uint256 vout);

    address owner;

    struct Transaction {
        address recipient;
        uint256 value;
    }

    uint256 coinbaseSeq = 0;
    mapping (bytes32 => Transaction) utxoPool;

    constructor() public {
        require(msg.sender != address(0));
        owner = msg.sender;
    }

    function transfer(bytes32 _txHash, uint256 _vout, address[] memory _recipients,
                      uint256[] memory _values) public {
        require(_recipients.length == _values.length);
        require(_recipients.length<=20);
        uint256 total;
        bytes32 db_key = keccak256(abi.encodePacked(_txHash, _vout));
        require(utxoPool[db_key].recipient == msg.sender);
        uint256 utxo_value = utxoPool[db_key].value;

        bytes32 newTxHash = keccak256(abi.encodePacked(_txHash, _vout,
            _recipients, _values));

        for(uint256 vout=0; vout<_recipients.length; vout++) {
            require(_recipients[vout] != address(0));
            bytes32 new_db_key = keccak256(abi.encodePacked(newTxHash, vout));
            utxoPool[new_db_key] = Transaction(_recipients[vout], _values[vout]);
            require(total < total+_values[vout]);
            total += _values[vout];
            emit Transfer(_txHash, newTxHash, _recipients[vout], _values[vout], vout);
        }

        require(total == utxo_value);
        delete utxoPool[db_key];
    }

    function() external payable {
        require(msg.value > 0);
        bytes32 txHash = keccak256(abi.encodePacked(msg.sender, msg.value,
            coinbaseSeq));
        bytes32 db_key = keccak256(abi.encodePacked(txHash, uint256(0)));
        utxoPool[db_key] = Transaction(msg.sender, msg.value);
        coinbaseSeq++;
        emit Transfer(bytes32(0), txHash, msg.sender, msg.value, uint256(0));
    }

    function withdraw(uint256 _value) public {
        require(msg.sender == owner);
        require(address(this).balance >= _value);
        msg.sender.transfer(_value);
    }
}
```

Если известно, что данный контракт зарегистрирован по адресу 0xe87a3686b0a42d66eee76d48c9a8307c27d14d1c (может быть найден в браузере блоков `https://blockscout.com/poa/sokol/`) в сети Sokol (тестовая сеть, совместимая с Ethereum Virtual Machine), напишите программу, которая бы позволяла для конкретного пользователя контракта отследить в каких блоках происходила покупка токенов, составляющих текущий баланс данного пользователя.

Формат входных данных

Одна строка — адрес пользователя контракта — последовательность начинающаяся с *0x*, за которыми следует 40 шестнадцатиричных символов.

Формат выходных данных

Строка — номера блоков, разделенные пробелом, перечисленные в порядке возрастания.

Примеры

Пример №1

Стандартный ввод
`0x2C7Cc50973b57b2b03f569643e4e604977D4F7fC`
Стандартный вывод
`6070845 6071386 6071669 6071796`

Комментарии

Если вам неизвестна концепция токенов, то можете обратиться к задачае "Получение баланса ERC-20 token" (`https://stepik.org/lesson/62024/step/4`) второго этапа профиля "Программная инженерия финансовых технологий" сезона 2016-2017 годов.

Для решения данной задачи необходимо познакомиться с языком написания Ethereum смарт-контрактов Solidity (`https://solidity.readthedocs.io`).

Получение информации из тестовой сети Sokol может происходить без необходимости синхронизировать свой собственный узел сети. Вместо этого можно отправлять JSON-RPC запросы на URL: `https://sokol.poa.network`.

Решение

Для решения задачи воспользуемся механизмом фильтров в библиотеке web3.

Первым делом, найдем все входящие транзакции пользователя, отфильтровав события *Transfer* по *{recepient: givenAddress}*. Затем, среди всех событий выберем только относящиеся к еще непотраченным транзакциям, то есть те, для которых существует запись в *utxoPool*(Для этого можно использовать функцию *web3.eth.getStorageAt*). По всем оставшимся событиям *Transfer* пройдем рекурсивно по цепочке до транзакции покупки токенов, используя фильтр событий по *{tx_address: event.tx_source}*, пока *tx_source != 0x000...00*. Для получения конечного ответа получим все номера блоков, оставим только уникальные, запишем в возрастающем порядке.

Пример программы-решения

Ниже представлено решение на языке Python3

```
from web3 import Web3

web3 = Web3(Web3.HTTPProvider('https://sokol.poa.network'))

contract_address = '0x2C7Cc50973b57b2b03f569643e4e604977D4F7fC'
address = '0x2C7Cc50973b57b2b03f569643e4e604977D4F7fC'

# This huge string is just our contract ABI :)
contract = web3.eth.contract(address=contract_address, abi='''[
        {
                "constant": false,
                "inputs": [
                        {
                                "name": "_value",
                                "type": "uint256"
                        }
                ],
                "name": "withdraw",
                "outputs": [],
                "payable": false,
                "stateMutability": "nonpayable",
                "type": "function"
        },
        {
                "constant": false,
                "inputs": [
                        {
                                "name": "_txHash",
                                "type": "bytes32"
                        },
                        {
                                "name": "_vout",
                                "type": "uint256"
                        },
                        {
                                "name": "_recipients",
                                "type": "address[]"
                        },
                        {
                                "name": "_values",
                                "type": "uint256[]"
                        }
                ],
                "name": "transfer",
                "outputs": [],
                "payable": false,
                "stateMutability": "nonpayable",
                "type": "function"
        },
        {
                "inputs": [],
                "payable": false,
                "stateMutability": "nonpayable",
                "type": "constructor"
        },
        {
```

```
                "payable": true,
                "stateMutability": "payable",
                "type": "fallback"
        },
        {
                "anonymous": false,
                "inputs": [
                        {
                                "indexed": true,
                                "name": "tx_source",
                                "type": "bytes32"
                        },
                        {
                                "indexed": true,
                                "name": "tx_address",
                                "type": "bytes32"
                        },
                        {
                                "indexed": true,
                                "name": "recipient",
                                "type": "address"
                        },
                        {
                                "indexed": false,
                                "name": "value",
                                "type": "uint256"
                        },
                        {
                                "indexed": false,
                                "name": "vout",
                                "type": "uint256"
                        }
                ],
                "name": "Transfer",
                "type": "event"
        }
]''')

# Filtering all Transfer events related to given address
filter = contract.events.Transfer().createFilter(
    fromBlock=0,
    toBlock='latest',
    argument_filters={'recipient': address})
events = filter.get_all_entries()

# utxoPool mapping slot number
slot = (2).to_bytes(32, byteorder='big')

ans = set()

for e in events:
    # Evaluating db_key
    key = web3.sha3(e.args['tx_address'] + e.args['vout'].to_bytes(32, byteorder='big'))
    # Evaluating ethereum storage key
    storage_key = web3.sha3(key + slot)
    # Retrieve utxoPool[db_key]
    storage_value = web3.eth.getStorageAt(contract_address, storage_key)

    # if utxoPool[db_key] exists
    if storage_value != bytes(32):
```

```
117         # recursively go to the root Transfer event (where tx_source is 0x000...00)
118         e1 = e
119         while e1.args['tx_source'] != bytes(32):
120             filter = contract.events.Transfer().createFilter(
121                 fromBlock=0,
122                 toBlock='latest',
123                 argument_filters={'tx_address': e1.args['tx_source']})
124             e1 = filter.get_all_entries()[0]
125 
126         ans.add(e1.blockNumber)
127 
128 for x in sorted(ans):
129     print(x, end=' ')
```

4.5. Блок задач 5

Задача 4.5.1. Проверка на настоящее лицо человека (3 балла)

Насколько сложно обмануть систему распознающую лицо? Что если вместо лица живого человека перед камерой расположить фотографию лица этого человека?

Для обработки таких случаев можно, конечно, использовать камеры глубины (`https://habr.com/post/224605/`) или стерео-камеры (`https://habr.com/post/130300/`, `https://habr.com/post/388259/`). Таким образом получится распознать плоская картинка перед камерой или объемная.

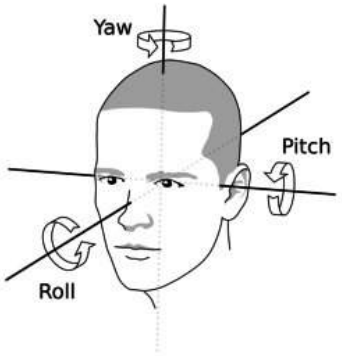
А что если вместо фотографии будет манекен? Тогда приведенный выше способ не подойдет.

В индустрии, одним из способов добавить дополнительную безопасность в систему идентификации по лицу является определения движения - пользователь вращает или кивает головой, во время просмотра картинки - движется зрачок, изменяется мимика лица.

Имея в виду то, что *MS Face API* может возвращать в результате обработки лица угол, на который лицо было повернуто, можно делать выводы настоящее ли лицо человека перед камерой. Для этого в запрос для определения лица нужно добавить параметр `headPose`, тогда в возвращенной JSON структуре будет следующий раздел:

```
"headPose": {
    "roll": 2.1,
    "yaw": 3,
    "pitch": 0
},
```

Объяснение для обозначений позиции головы можно увидеть на следующей картинке:



Теперь, если просить человека повернуть голову в ту или иную сторону в течение какого-то времени, то можно сравнивать реальные действия пользователя с ожидаемыми. Если совпадение действий найденно, то можно говорить о больше вероятности того, что перед камерой настоящее лицо человека.

Напишите программу, которая бы из серии изображений определяла бы те, в которых лицо пользователя повернуто или наклонено в ту или иную сторону. Поворотом лица в нужную сторону нужно считать такое положение лица, когда отклонение от нормали (достигает больше 15 градусов).

Формат входных данных

В первой строке - одно из четырех возможных словосочетаний:

- `turn left` — голова повёрнута (yaw на картинке вверху) влево
- `turn right` — голова повёрнута вправо
- `roll left` — голова наклонена влево
- `roll right` — голова наклонена вправо

Поворот головы в данную сторону нужно будет определить на фотографиях.

Следующая строка - ссылка на Google Drive, откуда нужно загрузить архив с фотографиями, которые нужно обработать.

Формат выходных данных

Выведите в одну строку последовательность из номеров фотографий, в которых поворот головы выполнен в нужную сторону. Номера фотографий перечислены по возрастанию и разделены одним пробелом.

Комментарии

Если изображение содержит несколько лиц, то для решения нужно брать параметры первого лица из списка, возвращаемого *MS Face API*. Сервис возвращает лица в порядке убывания размеров прямоугольников, в рамках которых определяется то или иное лицо на снимке.

Примеры

Пример №1

Стандартный ввод
roll left https://drive.google.com/open?id=1Lo_XafOQxAIyNdmCXCC2qjg6GLQFepPr

Стандартный вывод
0 3 5 6 9 10 11 12 13 14 19 24 25 26 29 30 31 35

Решение

Для каждой фотографии сделаем запрос к Face API, из ответа получим требуемые углы поворота(*pitch* и *yaw*), сравним с граничным значением и сделаем требуемый вывод о фотографии.

Пример программы-решения

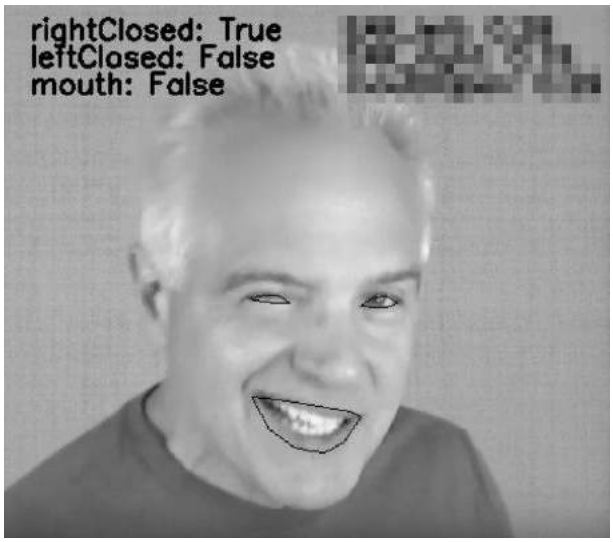
Ниже представлено решение на языке Python3

```
import cognitive_face as CF

KEY = '<your Face API subscription key>'
CF.Key.set(KEY)

# regional Cognitive Service URL
BASE_URL = 'https://westeurope.api.cognitive.microsoft.com/face/v1.0'
CF.BaseUrl.set(BASE_URL)

for i in range(40):
    img_url = './photos/{0}.jpg'.format(i)
    result = CF.face.detect(img_url, False, False, 'headPose')

    roll = result[0]['faceAttributes']['headPose']['roll']
    yaw = result[0]['faceAttributes']['headPose']['yaw']

    #roll left
    if roll > 15: # roll < -15 or yaw > 15 or yaw < -15
        print(i, end=' ')
```

Задача 4.5.2. Обнаружение состояния элементов лица (3 балла)

Чтобы еще более усилить алгоритмы распознования настоящего лица человека, можно просить пользователя не только изменять положение головы, но и подмигнуть, открыть и закрыть рот.

Если набор таких действий задать друг за другом, - например, "закройте правый глаз, откройте рот, откройте правый глаз, закройте левый глаз, закройте рот а человек все корректно повторит, то это позволит не только убедиться, что перед нами живой человек, но и новые данные, получаемые в ходе этой процедуры позволят уточнить - правильно ли вообще произолшла идентификация - тот ли человек пытается пройти идентификацию.



Поскольку ограниченные возможности MS Face API уже не могут реализовать подобные проверки, найдите способ, используя библиотеку OpenCV, воплотить проверки открыт-закрыт у человека глаз, открыт или закрыт рот.

Формат входных данных

В первой строке - одно из четырех возможных словосочетаний, определяющее какое действие будет проверяться от пользователя:

- `close right eye` — закрыть правый глаз
- `close left eye` — закрыть левый глаз
- `close both eyes` — закрыть оба глаза
- `open mouth` — открыть рот

Следующая строка - ссылка на Google Drive, откуда нужно загрузить архив с фотографиями, которые нужно обработать.

Формат выходных данных

Выведите в одну строку последовательность из номеров фотографий, в которых человек выполняет заданное действие. Номера фотографий перечислены по возрастанию и разделены одним пробелом.

Примеры

Пример №1

Стандартный ввод
`close left eye` `https://drive.google.com/open?id=1nNwt3V9nAzTxu-7MGumDUI0uoqmjqWZZ`
Стандартный вывод
`9 17 23 31 51 56 72 77 85`

Пример №2

Стандартный ввод
`close both eyes` `https://drive.google.com/open?id=1vfYapd7kfTqAk3iZ8ragBnnjGFf6QpNR`
Стандартный вывод
`4 7 13 15 22 45 51 60 66 68 76 83`

Пример №3

Стандартный ввод
`open mouth` `https://drive.google.com/open?id=1MrEQTSJakfnOUbylrgT50YBsl04wBDxj`
Стандартный вывод
`0 2 13 19 22 29 33 35 43 48 54 67 74 75 80 82 86 91`

Решение

Для решения задачи воспользуемся библиотекой dlib и shape-predictor'ом на 68 точек.

Для каждого глаза и рта определим нужные точки и вычислим AR (aspect ratio), по аналогии с этой статьёй: `https://www.pyimagesearch.com/2017/04/24/eye-blink-detection-opencv-python-dlib/`. Определим численные границы для глаз и рта, соответсвующие открытому и закрытому положению.

Для каждой фотографии вычислим необходимые параметры, сравним с граничными значениями, сделаем вывод о состоянии элементов лица.

Пример программы-решения

Ниже представлено решение на языке Python3

```
import dlib
import imutils as imutils

EYE_THRESHOLD = 0.2
MOUTH_THRESHOLD = 0.3

def shape_to_list(shape):
    coords = [None] * 68

    for i in range(0, 68):
        coords[i] = (shape.part(i).x, shape.part(i).y)

    return coords

def right_eye_open(points):
    eye = points[36:42]
    return aspect_ratio(eye) > EYE_THRESHOLD

def left_eye_open(points):
    eye = points[42:48]
    return aspect_ratio(eye) > EYE_THRESHOLD

def mouth_open(points):
    mouth = [points[60], points[61], points[63], points[64], points[65], points[67]]
    return aspect_ratio(mouth) > MOUTH_THRESHOLD

def aspect_ratio(eye):
    return (dist(eye[1], eye[5]) + dist(eye[2], eye[4])) / (2 * dist(eye[0], eye[3]))

def dist(a, b):
    return ((a[0] - b[0]) ** 2 + (a[1] - b[1]) ** 2) ** 0.5


predictor_path = '<dlib predictor_68 path>'
faces_folder_path = '<faces folder path>'

detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor(predictor_path)

for i in range(100):
    # load current image
    img_url = faces_folder_path + '/{0}.jpg'.format(i)
    img = dlib.load_grayscale_image(img_url)

    # do some resizing operations
    if img.shape[1] < 300:
        img = imutils.resize(img, width=img.shape[1] * 2)
    elif img.shape[1] < 450:
        img = imutils.resize(img, width=450)
    else:
        img = imutils.resize(img, width=900)

    # detect faces
    dets, scores, idx = detector.run(img, 0, -1)

    # get list of points
    points = shape_to_list(predictor(img, dets[0]))

    if not left_eye_open(points) and right_eye_open(points):
```

```
60          print(i, end=' ')
```