МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №2
по дисциплине «Информатика»
Тема: Введение в архитектуру компьютера

Студент гр. 2381 _____ Ильясов М.Р.

Преподаватель _____ Шевская Н.В.

Санкт-Петербург
2018

Цель работы

Изучить основные функции библиотеки Pillow (PIL) для работы с изображениями, применить полученные знания на практическом задании.

Задание

Вариант 4

Предстоит решить 3 подзадачи, используя библиотеку Pillow (PIL). Для реализации требуемых функций студент должен использовать numpy и PIL. Аргумент image в функциях подразумевает объект типа <class 'PIL.Image.Image'>

1) Рисование отрезка. Отрезок определяется:

- координатами начала
- координатами конца
- цветом
- толщиной.

Необходимо реализовать функцию user_func(), рисующую на картинке отрезок

Функция user_func() принимает на вход:

- изображение;
- координаты начала (x0, y0);
- координаты конца (x1, y1);
- цвет;
- толщину.

Функция должна вернуть обработанное изображение.

2) Преобразовать в Ч/Б изображение (любым простым способом).

Функционал определяется:

- Координатами левого верхнего угла области;
- Координатами правого нижнего угла области;
- Алгоритмом, если реализовано несколько алгоритмов преобразования изображения (по желанию студента).

Нужно реализовать 2 функции:

check_coords(image, x0, y0, x1, y1) - проверяет координаты области *(x0, y0, x1, y1)* на корректность (они должны быть неотрицательными, не превышать размеров изображения, поскольку *x0, y0* - координаты левого верхнего угла, *x1, y1* - координаты правого нижнего угла, то *x1* должен быть больше *x0*, а *y1* должен быть больше *y0*);

set_black_white(image, x0, y0, x1, y1) - преобразовывает заданную область изображения в черно-белый (используйте для конвертации параметр *'1'*). В этой функции должна вызываться функция проверки, и, если область некорректна, то должно быть возвращено исходное изображение без изменений. Примечание: поскольку черно-белый формат изображения (*greyscale*) является самостоятельным форматом, а не вариацией RGB-формата, для его получения необходимо использовать метод *Image.convert*.

3) Найти самый большой прямоугольник заданного цвета и перекрасить его в другой цвет. Функционал определяется:

Цветом, прямоугольник которого надо найти

Цветом, в который надо его перекрасить.

Написать функцию *find_rect_and_recolor(image, old_color, new_color)*, принимающую на вход изображение и кортежи rgb-компонент старого и нового цветов. Она выполняет задачу и возвращает изображение. При необходимости можно писать дополнительные функции.

Выполнение работы

Импорт необходимых модулей классов из модулей.

Задача 1. Объявление функции *user_func* с параметрами *image, x0, y0, x1, y1, fill, width* согласно тексту задания. Создаётся объект для рисования *draw* типа *ImageDraw*. При помощи метода *line* строится линия с указанными выше параметрами. Возвращается отредактированное изображение.

Задача 2. Объявление вспомогательной функции *check_coords(image, x0, y0, x1, y1)*. Проверка типа координат (должен быть *int*) с помощью

встроенных функций *all(), map()* и *isinstance()*. Сравнение координат на соответствие заданному диапазону. Возврат *True/False* в зависимости от результата предыдущих проверок.

Объявление основной функции *set_black_white(image, x0, y0, x1, y1)*. Проверка координат с помощью написанной *check_coords*. Кадрирование изображение по заданных координатам (с помощью *crop*), с сохранением в переменную place. Конвертирование области в *grayscale* через *convert*. Вставка отредактированного фрагмента на изначальное изображение. Возврат изображения.

Задача 3. Объявление вспомогательной функции *max_rectangle_under_histogram(data)*, для поиска прямоугольника максимальной площади под гистограммой, высоты столбцов которой заданы массивом *data*. Создание пустого стека индексов stack, инициализация нулями массива max_rectangle, содержащего площадь, ширину, высоту и координату *x* прямоугольника, добавление счётчика *i = 0*. Цикл while, работа которого завершается после превышения счётчиком границ массива и полного опустошения стека. В теле цикла реализовано ветвление. В случае выполнения ряда условий (проверка счётчика, проверка стека на пустоту и сравнение с предыдущим элементом стека) выполняется добавление номера очередного столбца в стек. В противном случае инициализируются переменные *width*, *height* и *area* шириной, высотой и площадью, которая сравнивается с площадью *max_rectangle[0]*. В случае, когда вычисленная площадь оказывается больше, чем сохранённая, *max_rectangle* перезаписывается.

Объявление вспомогательной функции *max_rectangle_in_matrix(matrix)*, которая ищет в двумерном массиве нулей и единиц прямоугольник максимальной площади.

Исходный код см. в приложении А.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
from PIL import Image, ImageDraw, ImageOps

# Задача 1
def user_func(image, x0, y0, x1, y1, fill, width):
      draw = ImageDraw.Draw(image)
      draw.line(((x0,y0),(x1,y1)), fill, width)
      return image

# Задача 2
def check_coords(image, x0, y0, x1, y1):
      if all(map(lambda p: isinstance(p, int), (x0, y0, x1, y1))):
            if 0 <= x0 < x1 <= image.size[0]:
                  if 0 <= y0 < y1 <= image.size[1]:
                        return True
      return False

def set_black_white(image, x0, y0, x1, y1):
      if check_coords(image, x0, y0, x1, y1):
            place = image.crop((x0, y0, x1, y1))
            place = place.convert('1')
            image.paste(place, (x0, y0))
      return image

# Задача 3
def max_rectangle_under_histogram(data):
      stack = []
      max_rectangle = [0, 0, 0, 0] # площадь, ширина, высота, x
      i = 0
      while (i < len(data) or stack):
            if (i < len(data)) and (not stack or data[stack[-1]]
<= data[i]):
                  stack.append(i)
                  i += 1
            else:
                  width = i
                  height = data[stack.pop()]
                  if stack:
                        width = (i-stack[-1]-1)
                  area = width * height
                  if area > max_rectangle[0]:
                        max_rectangle = [area, width, height, i-1]
      return max_rectangle

def max_rectangle_in_matrix(matrix):
      max_rectangle  =  max_rectangle_under_histogram(matrix[0])  +
[0] # добавляем y
      for y in range(1, len(matrix)):
            for x in range(len(matrix[0])):
                  if matrix[y][x]:
                        matrix[y][x] += matrix[y-1][x]
```

```
                new_rectangle                                                =
max_rectangle_under_histogram(matrix[y]) + [y]
                if max_rectangle[0] < new_rectangle[0]:
                     max_rectangle = new_rectangle
          return max_rectangle

     def find_rect_and_recolor(image, old_color, new_color):
          pix = image.load()
          matrix = []
          for y in range(image.height):
               matrix.append([])
               for x in range(image.width):
                    flag = (pix[x, y] == old_color)
                    matrix[y].append(int(flag))
          area, w, h, x2, y2 = max_rectangle_in_matrix(matrix)
          x1, y1 = x2 - w + 1, y2 - h + 1
          draw = ImageDraw.Draw(image)
          draw.rectangle((x1, y1, x2, y2), fill=new_color)
          return image
```