**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(ка) гр. 3383 |  | Логинова А.Ю. |
| --- | --- | --- |
| Преподаватель |  | Иванов Д. В |

Санкт-Петербург

2023

## **Цель работы.**

Написать программу, которая решает 3 подзадачи, используя библиотеку Pillow(PIL). Для реализации требуемых функций студент необходимо использовать **numpy** и **PIL.**

Аргумент image в функциях подразумевает объект типа *<class 'PIL.Image.Image'>.*

## **Задание.**

**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.

Задание на лабораторную работу в исходной формулировке. Указывается полностью. Указывается вариант задания, если есть. Обратите внимание, что форматирование текста задания должно быть таким, как и основной текст в отчете (вставляйте текст задания «без сохранения исходного форматирования», или с «применением форматирования документа»).

## **Выполнение работы.**

Для выполнения 1 задачи был использован метод Draw, находящийся в модуле ImageDraw. С помощью этого метода можно нарисовать фигуры, в данном случае было необходимо нарисовать линию. В метод передаются координаты конца и начала, возвращается обновленное изображение.

Для выполнения 2 задачи были использованы методы Crop и Convert, находящиеся в модуле Image. Перед тем как сделать изображение черно-белым, необходимо проверить координаты, это осуществляется посредством сравнения точек. Далее с помощью значения ‘1’ картинка *image* становится черно-белой.

Для выполнения задачи 3 был использован методы из numpy np.zeros(), который создает нулевую матрицу ширины переданной на вход картинки. В этой матрице будет храниться максимальная высота столбцов окрашенных пикселей в картинке по соответствующим координатам (под “окрашенными пикселями” подразумеваются те пиксели, цвет которых совпадает с переданным цветом на вход), на момент прохождения строки, соответствующая этапу обработки. Далее создаются начальные переменные *area\_size, left\_i, left\_j, right\_i, right\_j.* Далее матрица итеративно заполняется значениями.

Далее создается переменная heights\_set, которая хранит в себе набор неповторяющихся элементов матрицы *vector.* Создаются начальные переменные: *max\_area, max\_area\_left\_i, max\_area\_right\_i, max\_area\_left\_j, max\_area\_right\_j.* И запускается цикл, пробегающий по значениям *heights\_set*. Цикл ищет самый большой набор рядом стоящих столбцов, удовлетворяющих требованию быть не меньше соответствующего числа в этом множестве. Цикл проходит по всем числам в множестве. Самый большой ряд стоящих столбцов позволяет найти самую большую площадь для этого числа. Переменная chunk хранит в себе количество рядом стоящих столбцов. Под столбцами понимается столбец подряд идущих окрашенных пикселей. После того как будет найдена самая большая площадь для числа, она сохраняется в переменную, которая позже сравнивается между всеми числами, таким образом находится самая большая площадь до n-й строки. Таким образом, пройдя по всем n строкам, найдется самая большая площадь прямоугольника.

Разработанный программный код см. в приложении А.

## **Тестирование.**

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

| № п/п | Входные данные | Выходные данные | Комментарии |
| --- | --- | --- | --- |
|  |  |  | Наибольший прямоугольник черного цвета был закрашен зеленым. |

## **Выводы.**

Был исследован модуль PIL, который позволяет создавать, открывать, обрабатывать картинки. Также был изучен ImageDraw, позволяющий рисовать на любой картинке.

Была решена задача, которая преобразовывает картинку в черно-белую, которая рисует на картинке линию, и которая находит самый большой прямоугольник заданного цвета и закрашивает его цветом, заданным пользователем. Задачи были решены с помощью ранее изученных модулей.

# **ПРИЛОЖЕНИЕ А ИСХОДНЫЙ КОД ПРОГРАММЫ**

Название файла: main.py

import numpy as np

from PIL import Image, ImageDraw

def user\_func(image, x0, y0, x1, y1, fill, width):

draw = ImageDraw.Draw(image)

draw.line((x0, y0, x1, y1), fill, width)

return image

def check\_coords(image, x0, y0, x1, y1):

width, height = image.size

coord\_not\_negative = x0 >= 0 and x1 >= 0 and y0 >= 0 and y1 >= 0

coord\_fit = x1 <= width and y1 <= height

x1y1\_greater\_x0y0 = x1 > x0 and y1 > y0

if coord\_not\_negative and coord\_fit and x1y1\_greater\_x0y0:

return True

return False

def set\_black\_white(image, x0, y0, x1, y1):

coord\_check\_not\_passed = not (check\_coords(image, x0, y0, x1, y1))

if coord\_check\_not\_passed:

return image

img\_crp = image.crop((x0, y0, x1, y1)).convert('1')

image.paste(img\_crp, (x0, y0, x1, y1))

return image

def find\_rect(image, color):

pixels = image.load()

width, height = image.size

vector = np.zeros(width, dtype=int)

area\_size, left\_i, left\_j, right\_i, right\_j = 0, 0, 0, 0, 0

for j in range(height):

for i in range(width):

if pixels[i, j] == color:

vector[i] += 1

else:

vector[i] = 0

heights\_set = set(vector)

heights\_set.remove(0)

max\_area, max\_area\_left\_i, max\_area\_right\_i, max\_area\_left\_j, max\_area\_right\_j = 0, 0, 0, 0, 0

for set\_elem in heights\_set:

max\_chunk, max\_chunk\_i, max\_chunk\_j, chunk, chunk\_i, chunk\_j = 0, 0, 0, 0, 0, 0

for vec\_i in range(width):

if vector[vec\_i] >= set\_elem:

if chunk == 0:

chunk\_i = vec\_i

chunk\_j = set\_elem

chunk += 1

else:

if chunk > max\_chunk:

max\_chunk = chunk

max\_chunk\_i = chunk\_i

max\_chunk\_j = chunk\_j

chunk = 0

else:

if chunk > max\_chunk:

max\_chunk = chunk

max\_chunk\_i = chunk\_i

max\_chunk\_j = chunk\_j

new\_area\_size = max\_chunk \* set\_elem

if new\_area\_size > max\_area:

max\_area = new\_area\_size

max\_area\_left\_i = max\_chunk\_i

max\_area\_right\_i = max\_area\_left\_i + max\_chunk - 1

max\_area\_left\_j = j - set\_elem + 1

max\_area\_right\_j = j

if max\_area > area\_size:

area\_size = max\_area

left\_i = max\_area\_left\_i

right\_i = max\_area\_right\_i

left\_j = max\_area\_left\_j

right\_j = max\_area\_right\_j

return area\_size, (left\_i, left\_j), (right\_i, right\_j)

def recolor\_rect(image, i, j, new\_color):

drawing = ImageDraw.Draw(image)

drawing.rectangle((i, j), new\_color)

return image

def find\_rect\_and\_recolor(image, old\_color, new\_color):

area\_size, left, right = find\_rect(image, old\_color)

if area\_size > 0:

upd\_image = recolor\_rect(image, left, right, new\_color)

return upd\_image

else:

return image