Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Иркутский национальный исследовательский технический

университет»

Институт информационных технологий и анализа данных

**О Т Ч Ё Т**

|  |  |  |
| --- | --- | --- |
| о прохождении | | учебной практики |
|  | | (вид практики: учебная/производственная) |
| технологической (проектно-технологической) практики | | |
| (тип практики: технологическая/научно-исследовательская работа/преддипломная и др.) | | |
|  | | |
| в | ИРНИТУ | |
|  | (наименование профильной организации) | |

****Обучающегося Махлачева А. А.

(ФИО, группа, подпись)

Руководитель практики от института ИТиАД

Кононенко Роман Владимирович, доцент

(ФИО, должность, подпись)

<https://www.superjob.ru/resume/ajti-55731902.html>

Руководитель образовательной программы

Кононенко Р.В., доцент института ИТиАД

****(ФИО, должность, подпись)

Оценка по практике

(ФИО, подпись, дата)

Содержание отчета на \_\_\_ стр. Приложение к отчету на \_\_\_ стр.

Иркутск 2025

<https://hh.ru/resume/f3f5b6ebff0efcd5670039ed1f4479366d4675>

**Индивидуальное задание на прохождение**

**учебной практики: технологической (проектно-технологической) практики**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| для | Махлачева Андрея Александровича | | | | |
|  | (ФИО обучающегося полностью) | | | | |
| обучающегося | | 1 | курса | группы | ИСИб-24-1 |

по направлению подготовки Информационные системы и технологии

профиль Интеллектуальные системы обработки информации и управления

Место прохождения практики: ИРНИТУ

Сроки прохождения практики с «16» июня 2025 г. по «29» июня 2025 г.

Цели и задачи прохождения практики:

Содержание практики, вопросы, подлежащие изучению:

Планируемые результаты практики:

Руководитель практики от

института ИТиАД

\_\_\_\_\_\_\_\_\_\_\_/ Кононенко Р.М. /

(подпись

**Согласовано:**

Руководитель ООП

\_\_\_\_\_\_\_\_\_\_\_/Кононенко Р.В./

(подпись

«\_\_\_»\_\_\_\_\_\_\_\_\_\_2025 г.

С настоящим индивидуальным заданием и с программой практики ознакомлен, задание принято к исполнению

\_\_\_\_\_\_\_\_\_\_«16» июня 2025 г.

(подпись)

**Задание №1**

Постановка задачи:

Незнайка в своей экспедиции на Луну оказался на вершине лунной горы. Спуск вниз опасен, поэтому он взял с собой карту склона горы, где числами обозначено, сколько минут требуется на этот участок маршрута. Спуск происходит сверху вниз на один из соседних участков. Напишите программу, рассчитывающую минимальное время спуска (сумму чисел в пути с вершины до основания).

Ход решения:

*import* random

def find\_min\_path(*mountain*, *sum*):

    cur\_pos\_j = 0 *#текущая позиция по j (горизонтали)*

    print("Кратчайший путь:", mountain[0][0], *end*=' ')

*for* i *in* range(1, n):

*for* j *in* range(cur\_pos\_j, cur\_pos\_j + 1):

*if* mountain[i][j] > mountain[i][j + 1]:

                cur\_pos\_j = j + 1

                sum += mountain[i][j + 1]

*else*:

                cur\_pos\_j = j

                sum += mountain[i][j]

            print(mountain[i][cur\_pos\_j], *end*=' ')

    print()

*return* sum

n = int(input("Введите количество уровней горы: "))

mountain = []

print("Сгенерированная гора (числа от 10 до 100):")

*for* i *in* range(n): *#генерируем гору*

    mountain.append([])

*for* j *in* range(i + 1):

        mountain[i].append(random.randint(10, 100))

        num = mountain[i][j] *#вектор из векторов*

        print(num, *end*=' ')

    print()

sum = mountain[0][0]

print("Минимальное время спуска (сумма):", find\_min\_path(mountain, sum))

Результат:

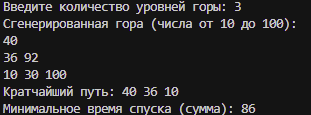


Рисунок 1

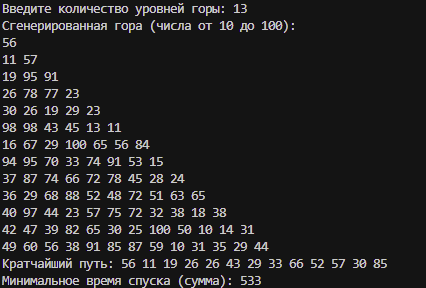
****

Рисунок 2

**Задание №2**

Постановка задачи:

После метеоритной атаки компьютерная сеть для управления лунными заводами разбилась на части, нужно объединить её в единое целое. Каждый фрагмент сети представлен в виде ненаправленного графа.

Вам известно общее число вершин графа (узлы сети, не более 1000) и набор рёбер (сохранившиеся линии связи, не более 1000).

Определите, какое минимальное число линий связи нужно дополнительно построить, чтобы сеть стала единой.

Ход решения:

*import* random

def dfs(*graph*, *start\_node*, *visited*):

    stack = [start\_node]  *# создаем стек и кладем в него стартовый узел*

*while* stack:  *# пока стек не пуст*

        node = stack.pop()  *# берем последний добавленный узел*

*if* node not in visited:  *# если его еще не посещали*

            visited.add(node)  *# отмечаем как посещенный*

*for* neighbor *in* graph[node]:  *# для каждого соседа этого узла*

*if* neighbor not in visited:  *# если сосед еще не посещен*

                    stack.append(neighbor)  *# добавляем его в стек*

def count\_connected\_components(*graph*, *n*):

    visited = set()  *# множество для отслеживания посещенных узлов*

    components = 0   *# счетчик компонент связности*

*for* node *in* range(1, n + 1):

*if* node not in visited:

            dfs(graph, node, visited)  *# обходим всю компоненту связности*

            components += 1

*return* components

def gen\_random\_edges(*n*, *m*):

    edges = [] *# список*

    used\_edges = set() *# множество (удаляет дубликаты)*

*while* len(edges) < m:

*# случайный узел от 1 до n*

        u = random.randint(1, n)

        v = random.randint(1, n)

*# проверяем: не петля? и не дубликат?*

*if* u != v and (u, v) not in used\_edges and (v, u) not in used\_edges:

            edges.append((u, v))

            used\_edges.add((u, v))

*return* edges

n = int(input("Введите количество узлов сети: "))

m = int(input("Введите число линий связи (ребер): "))

edges = gen\_random\_edges(n, m)

*# создаем список из (n+1) пустых списков*

graph = [[] *for* \_ *in* range(n + 1)]

*# циклом добавляем рёбра в граф*

*for* u, v *in* edges:  *# для каждого ребра (u, v) в списке ребер*

    graph[u].append(v)  *# добавляем v в список соседей u*

    graph[v].append(u)  *# добавляем u в список соседей v*

print("Сгенерированные узлы:")

*for* u, v *in* edges: *# edges = [(1, 2), (2, 3), (1, 3)]  # пример списка ребер*

    print(f"{u} {v}")

components = count\_connected\_components(graph, n) *# находим количество компонент связности*

print(f"Количество необходимых связей: {components - 1}")

Результат:

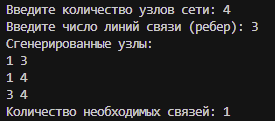


Рисунок 3

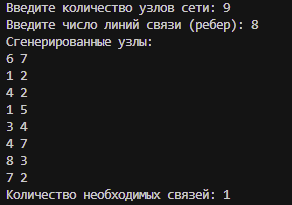


Рисунок 4

**Задача №3**

Постановка задачи:

В Иркутске раз в году наступает зима. Не смотря на то что событие это довольно регулярное, оно всегда внезапно. Снег буквально заваливает все улицы, не давая проехать на чём-то меньше трактора. В этом году терпение лопнуло и специальным указом был создан кризисный центр по борьбе с сугробами. Центру были переданы спутники, лазеры, метеорологические зонды и несколько десятков лопат.

Вам поручено возглавить отдел разведки снежной ситуации и быть способным чрезвычайно быстро отвечать на запросы центра. Сам город состоит из нескольких, расположенных подряд, улиц, каждая из которых абсолютна похожа на любую другую.

* Информация о снеге передается вам в виде тройки чисел – 1 в качестве идентификатора события, уникального индекса улицы и количество миллиметров выпавшего снега.
* Запросы в свою очередь так же имеют вид тройки чисел – 2 в качестве идентификатора события, индекс улицы с которой нужно суммировать количество выпавшего снега и индекс улицы по которую нужно суммировать, крайние улицы должны быть включены.

Ход решения:

print("Введите количество улиц(n) и запросов(k): ")

n, k = map(int, input().split()) *#применяет int к каждому введенному элементу списка, разделенного пробелами*

streets = [0] \* (n + 1)

queries = [] *#все запросы*

print("Введите запросы: ")

*for* \_ *in* range(k):

    q = list(map(int, input().split()))

    queries.append(q)

answers = []

*for* q *in* queries:

*if* q[0] == 1:

*# добавить снег на улицу i*

        \_, i, x = q *#распаковка списка \_ значит что первая переменная не используется*

        streets[i] += x

*else*:

*# запрос суммы от u до r*

        \_, u, r = q

        total = sum(streets[u:r+1])

        answers.append(total)

print("Ответы на запросы: ")

*for* i *in* answers:

    print(i)

Результат:

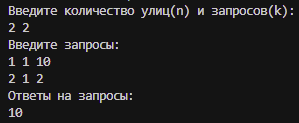


Рисунок 5

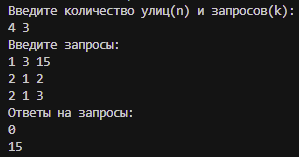


Рисунок 6

**Задача №4**

Постановка задачи:

Перестановка P длины n − это упорядоченный набор, содержащий числа от 1 до n, каждое из которых входит в него ровно один раз. Например, перестановкой длины 13 является набор (5 11 13 12 6 1 8 4 10 9 7 2 3). Само название говорит о том, для чего предназначен этот объект. Например, можно при помощи перестановки букв зашифровать слово. Для примера возьмем приведенную выше перестановку и слово transposition, которое состоит тоже из 13 букв. Далее, следуя перестановке, на первую позицию поставим пятую букву слова, на вторую − одиннадцатую букву и так далее. В итоге получим sinoptsntiora. К этому слову снова применим эту же перестановку и получим poartsnoitsin. Повторив эти стадии шифрования k раз, получим зашифрованное сообщение.

Вам дано зашифрованное таким образом слово, шифрующая перестановка P и число k. Необходимо восстановить слово.

**Формат входных данных**

Первая строка входных данных содержит 2 числа – n и k (1 или больше, могут быть равны). Следующая строка содержит перестановку длиной n, числа разделяются пробелом. Третья строка содержит зашифрованное слово длиной n.

**Формат выходных данных**

Вывести одну строку − исходное слово.

Ход решения:

print("Введите количество элементов(n), запросы(k): ")

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

print("Введите перестановку: ")

P = list(map(int, input().split()))

print("Введите строку: ")

s = input().strip() *#удаляет пробелы, табуляции, переносы строк с начала и конца строки, но не трогает пробелы внутри строки*

orig = [''] \* n *#создает список из n пустых строк*

visited = [False] \* n *#создает список из n False*

*for* i *in* range(n):

*if* visited[i]:

*continue*

    cycle = []

    cur = i

*while* not visited[cur]:

        visited[cur] = True

        cycle.append(cur)

        cur = P[cur] - 1  *# перестановка 1-based*

    L = len(cycle)

    shift = k % L

*for* idx\_in\_cycle, pos *in* enumerate(cycle):

        j = cycle[(idx\_in\_cycle + shift) % L]

        orig[j] = s[pos]

print("Результат: ")

print("".join(orig)) *#соединяет элементы списка orig в одну строку*

Результат:

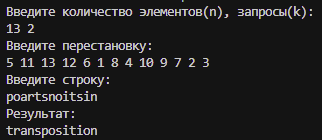


Рисунок 7

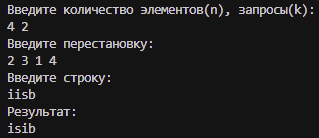


Рисунок 8

**Задача №5**

Постановка задачи:

Дана матрица, состоящая из 1 и 0. Значениями 1 в матрице нарисована некоторая фигура. Необходимо определить координаты верхнего левого и нижнего правого углов параллельного осям ограничивающего прямоугольника, т.е. такого прямоугольника, минимального размера, в который фигура помещается полностью и при этом ни одна точка исходной фигуры не попадает на стороны прямоугольника.

**Формат входных данных**

В первой строке через пробел заданы высота h и ширина w матрицы (длина и ширина 10 или больше, но не больше 50, могут быть равны). В следующих строках заданы значения матрицы по строкам и столбцам. В матрице всегда есть только одна фигура. Фигура отстоит от краев матрицы минимум на один ноль. Начало координат в левом верхнем углу. Координаты растут вниз и вправо.

**Формат выходных данных**

Координаты верхнего левого и правого нижнего угла прямоугольника отделенные пробелами. Координаты задаются номером строки и номером столбца. Нумерация начинается с 0.

Ход решения:

print("Введите размеры матрицы: ")

h, w = map(int, input().split())

matrix = []

print("Введите матрицу: ")

*for* \_ *in* range(h):

    row = list(map(int, input().split()))

    matrix.append(row)

*# поиск границ фигуры*

min\_row = h

max\_row = 0

min\_col = w

max\_col = 0

*# проходим по всем элементам матрицы*

*for* i *in* range(h):

*for* j *in* range(w):

*if* matrix[i][j] == 1:

*# обновляем границы*

*if* i < min\_row:

                min\_row = i

*if* i > max\_row:

                max\_row = i

*if* j < min\_col:

                min\_col = j

*if* j > max\_col:

                max\_col = j

print("Результат: ")

print(f"{min\_row} {min\_col} {max\_row} {max\_col}")

Результат:

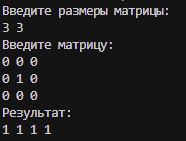


Рисунок 9

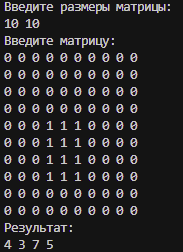


Рисунок 10

**Задача №6**

Постановка задачи:

В школьном кружке робототехники есть два вида микроконтроллеров (условно тип A и тип B) и два вида модулей управления мотором (условно тип 1 и тип 2). Выяснилось, что контроллер типа B и модуль управления типа 2 несовместимы. Использование микроконтроллеров и модулей управления в других комбинациях возможно. Имеется a микроконтроллеров типа A, b микроконтроллеров типа B, x модулей управления типа 1 и y модулей типа 2. Определите, какое максимальное число работающих пар из микроконтроллера и модуля управления мотором можно составить. Ваша программа должна ответить на n запросов.

**Формат входных данных**

В первой строке пишем число n (не больше 50). Далее в n строках пишем по 4 натуральных числа (a, b, x, y).

**Формат выходных данных**

Выводим n чисел через пробел, каждое число – максимальное число работающих пар из микроконтроллера и модуля управления мотором можно составить для строки.

Ход решения:

print("Введите количество пар: ")

n = int(input().strip()) *#удаляет пробелы, табуляции, переносы строк с начала и конца строки, но не трогает пробелы внутри строки*

results = []

print("Введите данные: ")

*for* \_ *in* range(n):

    data = list(map(int, input().split()))

    a, b, x, y = data

    pairs\_B1 = min(b, x)

    rem\_x = x - pairs\_B1

    pairs\_A = min(a, rem\_x + y)

    total = pairs\_B1 + pairs\_A

    results.append(str(total))

print("Результат: ")

print(" ".join(results))

Результат:

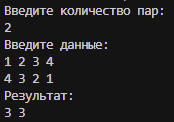


Рисунок 11

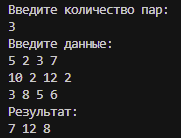


Рисунок 12

**Задача №7**

Постановка задачи:

На компьютере работника автосервиса нашли файл с последовательностью автомобильных номеров, обслуживавшихся в этом автосервисе. Так как файл был поврежден, некоторые данные отображаются неверно. Нужно определить, какие из них остались невредимыми.

Автомобильным номером является строка из шести символов. Первый символ – заглавная латинская буква, далее следует 3 цифры, и после – две заглавные латинские буквы. Например, строка "P142EQ" является номером. Вам будет дана строка, состоящая из шести символов, необходимо ответить, является ли строка автомобильным номером.

**Формат входных данных**

В единственной строке находится строка из шести символов, состоящая из цифр и заглавных латинских букв.

**Формат выходных данных**

Если строка является автомобильным номером, то необходимо вывести "Yes", в ином случае – "No" без кавычек.

Ход решения:

print("Введите номер автомобиля: ")

s = input().strip()

print("Результат: ")

*if* len(s) != 6:

    print("No")

*else*:

*# проверяем первый символ (должна быть заглавная буква)*

*if* not s[0].isupper() or not s[0].isalpha():

        print("No")

*# проверяем следующие три символа (должны быть цифрами)*

*elif* not s[1:4].isdigit():

        print("No")

*# проверяем последние два символа (должны быть заглавными буквами)*

*elif* not s[4:6].isalpha() or not s[4:6].isupper():

        print("No")

*else*:

        print("Yes")

Результат:

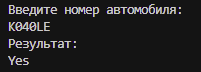


Рисунок 13

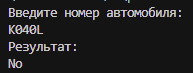


Рисунок 14

**Задача №8**

Постановка задачи:

Составить светодиодную матрицу размером не менее 8 на 8 светодиодов (пример на рисунке ниже размером 4 на 4)

Обратите внимание на ориентацию светодиодов на поле!

На матрицу вывести инфографику с различными динамично меняющимися изображениями.



Рисунок *а*

Ход решения:

#include <Adafruit\_NeoPixel.h>

#define DATA\_PIN 6

#define NUM\_LEDS 64

*// создание объекта для управления светодиодной матрицей*

Adafruit\_NeoPixel matrix(NUM\_LEDS, DATA\_PIN, NEO\_GRB + *NEO\_KHZ800*);

*// NUM\_LEDS - количество светодиодов (64 для матрицы 8x8)*

*// DATA\_PIN - пин Arduino, к которому подключена матрица (пин 6)*

*// NEO\_GRB - порядок цветов: Green, Red, Blue*

*// NEO\_KHZ800 - частота передачи данных (800 kHz для NeoPixel)*

*// вычисляем индекс светодиода по координатам*

int **indexFromXY**(int *col*, int *row*)

{

    if (*row* % 2 == 0)

    { *// если строка четная*

        return *row* \* 8 + *col*; *// светодиоды идут слева направо*

    }

    else

    { *// если строка нечетная*

        return *row* \* 8 + (7 - *col*); *// светодиоды идут зигзагом*

    }

}

void **setup**()

{

    matrix.begin(); *// инициализация библиотеки*

    matrix.clear(); *// очистка матрицы (все светодиоды выключены)*

    matrix.show(); *// применение изменений (отправка данных на матрицу)*

}

void **loop**() *// выполняется бесконечно после setup()*

{

**drawArrow**();

**drawCross**();

**rainbowFill**();

}

*// отображение стрелки*

void **drawArrow**()

{

    byte arrow[8][8] = {

        {0, 0, 0, 1, 1, 0, 0, 0},

        {0, 0, 1, 1, 1, 1, 0, 0},

        {0, 1, 0, 1, 1, 0, 1, 0},

        {1, 0, 0, 1, 1, 0, 0, 1},

        {0, 0, 0, 1, 1, 0, 0, 0},

        {0, 0, 0, 1, 1, 0, 0, 0},

        {0, 0, 0, 1, 1, 0, 0, 0},

        {0, 0, 0, 1, 1, 0, 0, 0}};

    for (int rep = 0; rep < 2; rep++)

    { *// повторить 2 раза*

        matrix.clear(); *// очистить матрицу*

*// проход по всем пикселям матрицы 8x8*

        for (int y = 0; y < 8; y++)

        {

            for (int x = 0; x < 8; x++)

            {

                if (arrow[y][x])

                { *// если в матрице стрелки 1*

*// включить светодиод с цветом (0,200,255) - голубой*

                    matrix.setPixelColor(**indexFromXY**(x, y), matrix.Color(0, 200, 255));

                }

            }

        }

        matrix.show(); *// показать стрелку*

        delay(600); *// ждать 600 мс*

        matrix.clear(); *// очистить матрицу*

        matrix.show(); *// применить очистку*

        delay(300); *// ждать 300 мс (мигание)*

    }

}

*// отображение крестика*

void **drawCross**()

{

    byte cross[8][8] = {

        {0, 0, 0, 1, 1, 0, 0, 0},

        {0, 0, 0, 1, 1, 0, 0, 0},

        {0, 0, 0, 1, 1, 0, 0, 0},

        {1, 1, 1, 1, 1, 1, 1, 1},

        {1, 1, 1, 1, 1, 1, 1, 1},

        {0, 0, 0, 1, 1, 0, 0, 0},

        {0, 0, 0, 1, 1, 0, 0, 0},

        {0, 0, 0, 1, 1, 0, 0, 0}};

    for (int rep = 0; rep < 2; rep++)

    {

        matrix.clear();

        for (int y = 0; y < 8; y++)

        {

            for (int x = 0; x < 8; x++)

            {

                if (cross[y][x])

                {

                    matrix.setPixelColor(**indexFromXY**(x, y), matrix.Color(255, 50, 50));

                }

            }

        }

        matrix.show();

        delay(600);

        matrix.clear();

        matrix.show();

        delay(300);

    }

}

*// эффект последовательного заполнения цветами*

void **rainbowFill**()

{

    int pal[3][3] = {

        {255, 0, 0},

        {0, 255, 0},

        {0, 0, 255}};

    for (int c = 0; c < 3; c++)

    {

        for (int i = 0; i < NUM\_LEDS; i++)

        {

            matrix.setPixelColor(i, matrix.Color(pal[c][0], pal[c][1], pal[c][2]));

            matrix.show();

            delay(25);

        }

        delay(350);

        matrix.clear();

        matrix.show();

        delay(250);

    }

}

*// как работает отрисовка:*

*//     создается битовая карта (массив 8x8), где 1 - светить, 0 - не светить*

*//     двойной цикл проходит по всем координатам матрицы*

*//     проверка условия if (arrow[y][x]) - если элемент равен 1*

*//     установка цвета для конкретного светодиода через indexFromXY()*

*//     show() отправляет все изменения на матрицу*

*//     мигание создается чередованием показа и очистки*

Результат:

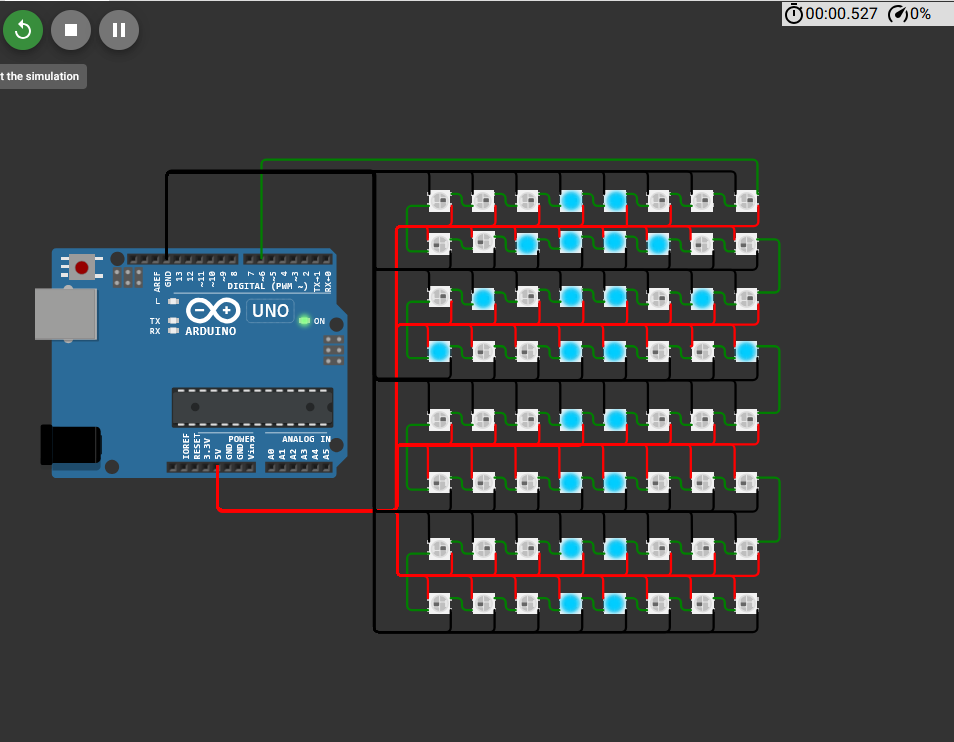


Рисунок 15

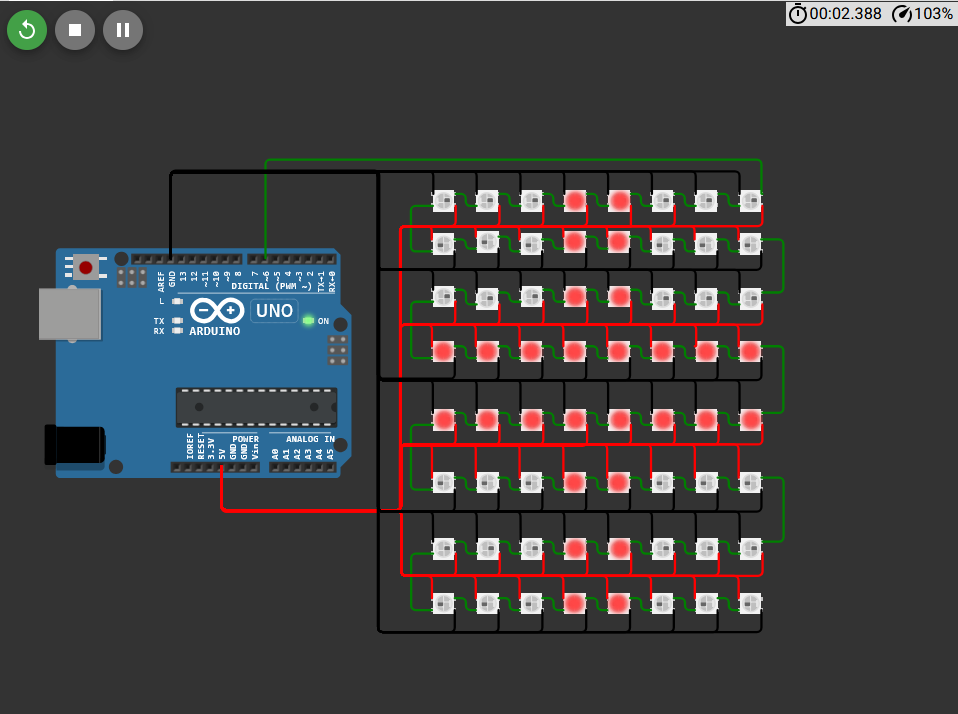


Рисунок 16

**Задача №9**

Постановка задачи:

Задачи:

1. Собрать схему имитирующую работу автоматических дверей
2. Подобрать номинал резисторов для светодиодов
3. Написать программу для управления процессом работы автоматических дверей.

Схема приведена на рисунке *б*.

Зеленый светодиод – двери отрываются.

Красный светодиод ­– двери закрываются.

Фоторезистор имитируют процесс приближения-удаления человека от дверей.



Рисунок *б* – Схема управления работой автоматических дверей

Изменение значений фоторезистора осуществляется при помощи ползунка (рисунок *в*), изменение значения фоторезистора доступно только, когда запущен процесс моделирования.



Рисунок *в* – Изменение значения фоторезистора

Логика работы программы:

1. По умолчанию горит светодиод, имитирующий закрытую дверь
2. Микроконтроллер считывает значение фоторезистора с аналогово пина
3. Если значение на пине превышает 512, на определённое время загорается светодиод, имитирующий открытую дверь, в последовательный порт выводится сообщение о событии.
4. После истечения заданного временного промежутка проверяется значение фоторезистора, если оно всё ещё превышает 512, дверь должна остаться открытой, в противном случае нужно включить индикацию закрытой двери, в последовательный порт выводится сообщение о событии.

Ход решения:

const byte PIN\_RED = 9;

const byte PIN\_GREEN = 5;

const byte SENSOR\_PIN = A0;

*// параметры для расчёта освещённости*

const float FACTOR = 0.7; *// коэффициент для формулы пересчета*

const float RESIST\_REF = 50.0; *// опорное сопротивление для калибровки*

*// функция пересчёта аналогового входа в люксы*

int **calcLux**(int *raw*)

{

*// преобразуем сырое значение (0-1023) в напряжение (0-5В)*

    float u = (*raw* / 1023.0) \* 5.0;

*// вычисляем сопротивление фоторезистора по формуле делителя напряжения*

    float r = 2000.0 \* u / (5.0 - u);

*// пересчитываем сопротивление в уровень освещенности в люксах*

    float lux = pow((RESIST\_REF \* 1000.0 \* pow(10, FACTOR)) / r, 1.0 / FACTOR);

    return (int)lux;

}

void **setup**()

{

*// настраиваем пины светодиодов как выходы*

    pinMode(PIN\_RED, OUTPUT);

    pinMode(PIN\_GREEN, OUTPUT);

*// инициализируем последовательный порт для отладки*

    Serial.begin(115200);

*// устанавливаем начальное состояние - дверь закрыта*

    digitalWrite(PIN\_RED, HIGH); *// красный горит - закрыто*

    digitalWrite(PIN\_GREEN, LOW); *// зеленый выключен*

}

void **loop**()

{

*// читаем значение с фоторезистора*

    int sensor = analogRead(SENSOR\_PIN);

    delay(3000); *// ждем 3 секунды между измерениями*

*// проверяем — если ярко, открываем дверь*

    if (**calcLux**(sensor) >= 512)

    {

*// открываем дверь - включаем зеленый, выключаем красный*

        digitalWrite(PIN\_RED, LOW);

        digitalWrite(PIN\_GREEN, HIGH);

        Serial.println("Двери открываются");

*// ждем 3 секунды и проверяем снова*

        delay(3000);

        sensor = analogRead(SENSOR\_PIN);

*// повторная проверка освещенности*

        if (**calcLux**(sensor) < 512)

        {

*// если стало темно - закрываем дверь*

            digitalWrite(PIN\_RED, HIGH);

            digitalWrite(PIN\_GREEN, LOW);

            Serial.println("Двери закрываются");

        }

        delay(3000); *// ждем перед следующим циклом*

    }

}

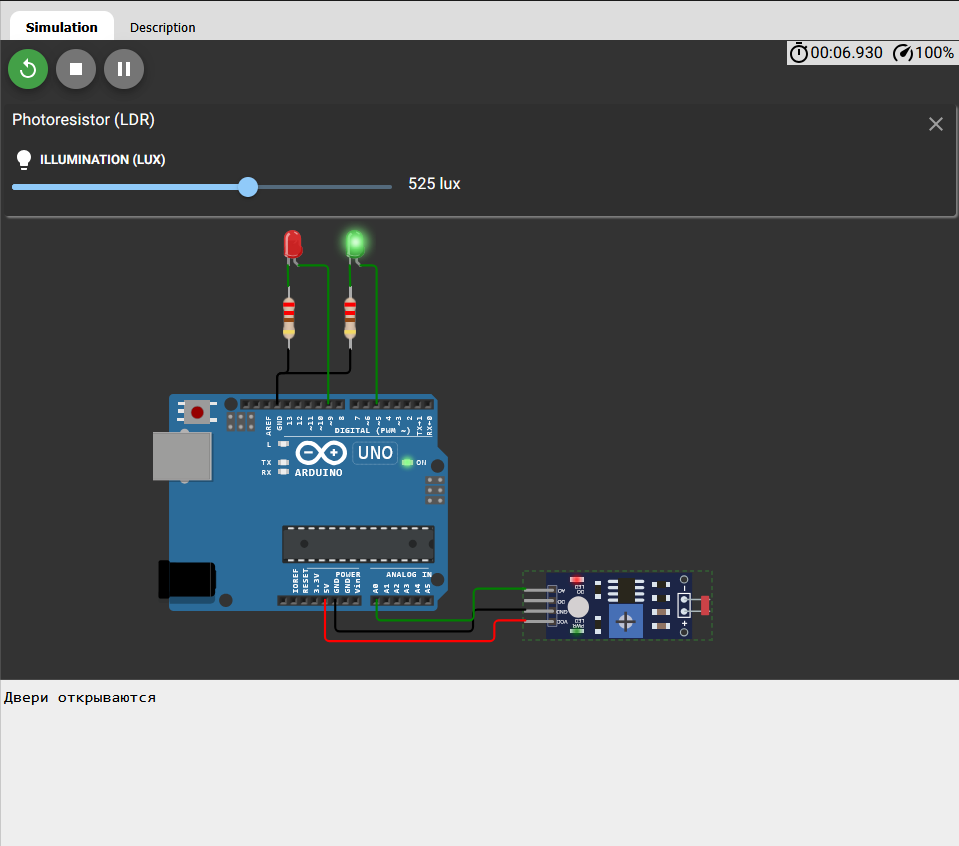
Результат: 

Рисунок 17

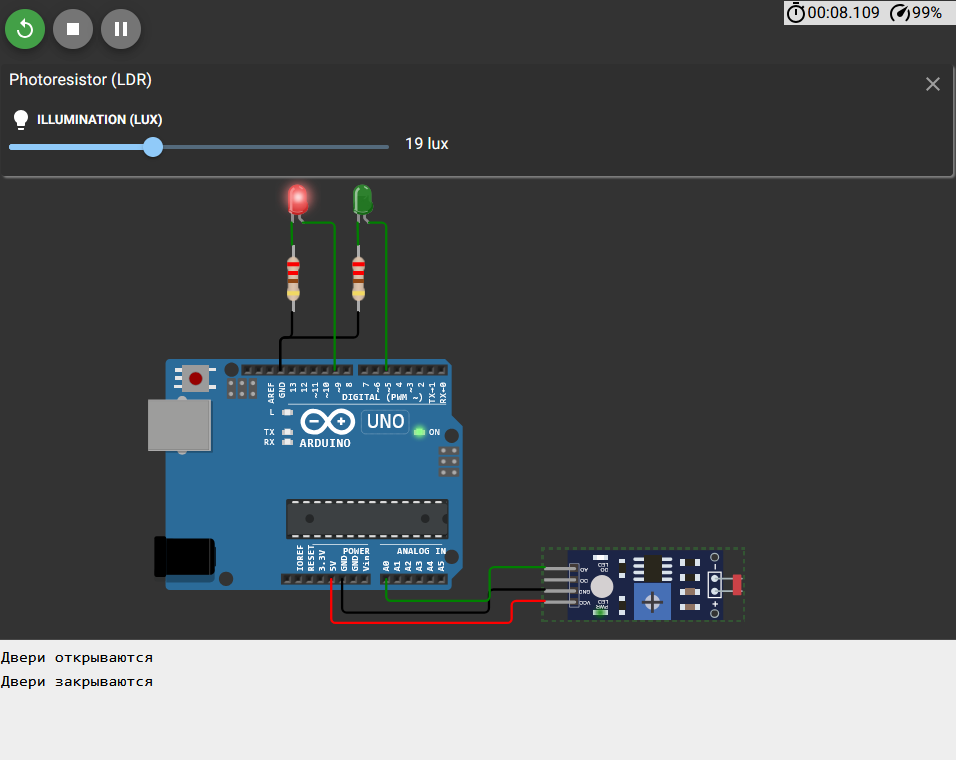


Рисунок 18

**Задача №10**

Постановка задачи:

Задачи:

1. Собрать схему подключения сервопривода
2. Написать программу для управления сервоприводом через последовательный порт



Рисунок *г* – Подключение сервопривода

Логика работы программы:

1. Программа находится в ожидании ввода данных в последовательный порт
2. Когда в последовательный порт вводятся данные, программа проверяет их на корректность (Должны приниматься только числовые значения в диапазоне от 0 до 180)
3. Если введён угол поворота, отличный от текущего угла, сервопривод плавно поворачивается в заданное положение.

Ход решения:

#include <Servo.h>

Servo servoMotor; *// создаем объект для управления сервоприводом*

int lastPos = 0; *// переменная для хранения последней позиции сервопривода*

void **setup**()

{

    Serial.begin(9600); *// инициализируем последовательный порт (монитор порта)*

    servoMotor.attach(4); *// подключаем сервопривод к пину 4 Arduino*

    servoMotor.write(lastPos); *// устанавливаем сервопривод в начальное положение 0 граудсов*

    Serial.println("Введите значение угла от 0 до 180:");

}

void **loop**()

{

*// проверяем, есть ли данные в последовательном порту (пользователь ввел угол*

    if (Serial.available())

    {

*// читаем целое число из последовательного порта*

        int target = Serial.parseInt();

        if (target < 0 || target > 180)

        {

            Serial.println("Некорректный ввод! Диапазон: 0–180");

            return;

        }

*// проверяем, не находится ли сервопривод уже в требуемой позиции*

        if (target == lastPos)

        {

            Serial.println("Сервопривод уже находится в этой позиции.");

            return;

        }

        Serial.print("Движение к углу: ");

        Serial.println(target);

*// определяем направление движения*

        int step = 0;

        if (target > lastPos)

        {

            step = 1; *// двигаемся вперед*

        }

        else

        {

            step = -1; *// двигаемся назад*

        }

*// плавно перемещаем сервопривод от текущей позиции к целевой*

        for (int pos = lastPos; pos != target + step; pos += step)

        {

            servoMotor.write(pos); *// устанавливаем текущую позицию сервопривода*

            delay(15); *// небольшая задержка для плавности движения*

        }

*// сохраняем новую позицию как последнюю*

        lastPos = target;

    }

}

Результат:

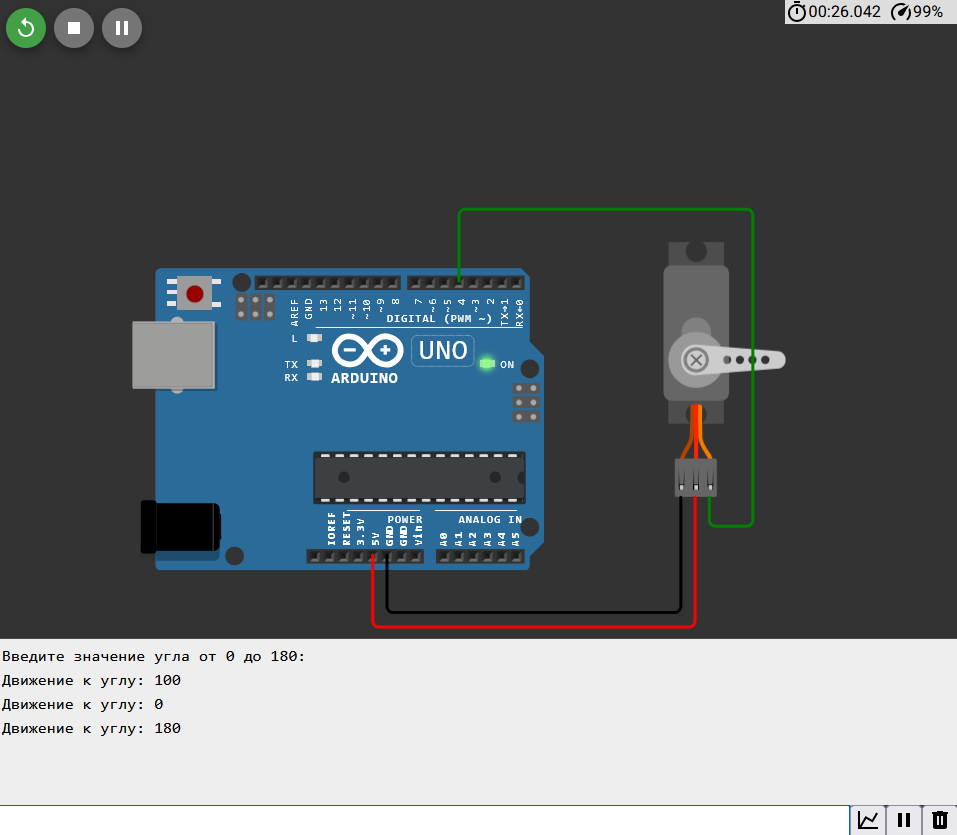


Рисунок 19

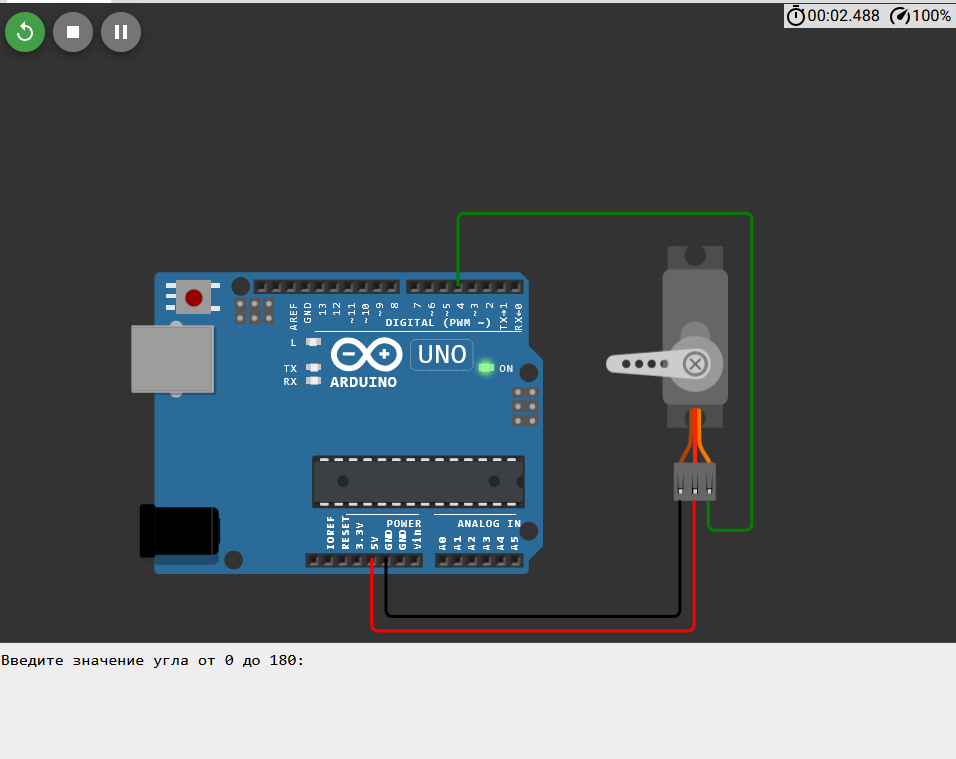


Рисунок 20

**Задача №11**

Вариант: 13 (по старым спискам)

Постановка задачи:

Find all green objects in the image. Find the centers of green objects. Mark the center with a red dot.

Перевод:

Найдите все зелёные объекты на изображении. Найдите центры зелёных объектов. Отметьте центр красной точкой.

Ход решения:

*import* cv2

*import* numpy *as* np

*# загружает изображение из файла*

image = cv2.imread("test\_1.jpg")

result\_image = image.copy()

blurred\_image = cv2.GaussianBlur(image, (11, 11), 0) *#добавляет размытие к изображению*

hsv\_image = cv2.cvtColor(blurred\_image, cv2.COLOR\_BGR2HSV) *#преобразует изображение в цветовую модель HSV*

hsv\_min = np.array((36, 25, 25), np.uint8) *#устанавливает минимальные значения для зеленого цвета*

hsv\_max = np.array((70, 255, 255), np.uint8)

green\_mask = cv2.inRange(hsv\_image, hsv\_min, hsv\_max) *#создает маску для зеленого цвета, создает бинарную маску: 255 для зеленых пикселей, 0 для остальных*

contours, hierarchy = cv2.findContours(green\_mask.copy(), cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE) *#находит контуры на маске*

*#findContours() - находит контуры на бинарном изображении*

*#RETR\_EXTERNAL - возвращает только внешние контуры (игнорирует дыры внутри)*

*#CHAIN\_APPROX\_SIMPLE - упрощает контур (сохраняет только угловые точки)*

*#contours - список найденных контуров*

*#hierarchy - иерархия между контурами (здесь не используется)*

*for* i, contour *in* enumerate(contours):

*if* cv2.contourArea(contour) < 100: *#если площадь контура меньше 100, то пропускаем*

*continue*

    x, y, w, h = cv2.boundingRect(contour) *#находит прямоугольник, в который вписан контур*

    center\_x = int(x + w/2)

    center\_y = int(y + h/2)

    cv2.circle(result\_image, (center\_x, center\_y), 10, (0, 0, 255), -1)

    cv2.drawContours(result\_image, [contour], -1, (255, 0, 0), 2)

    print(f"Объект {i+1}: центр в точке ({center\_x}, {center\_y}), размер: {w}x{h}")

cv2.imshow("Оригинальное изображение", image)

cv2.imshow("Зеленая маска", green\_mask)

cv2.imshow("Результат с отмеченными центрами", result\_image)

cv2.waitKey(0)

cv2.destroyAllWindows()

*#исходное изображение -> размытие -> HSV конвертация -> цветовая маска -> нахождение контуров -> отрисовка результатов*

Результат:

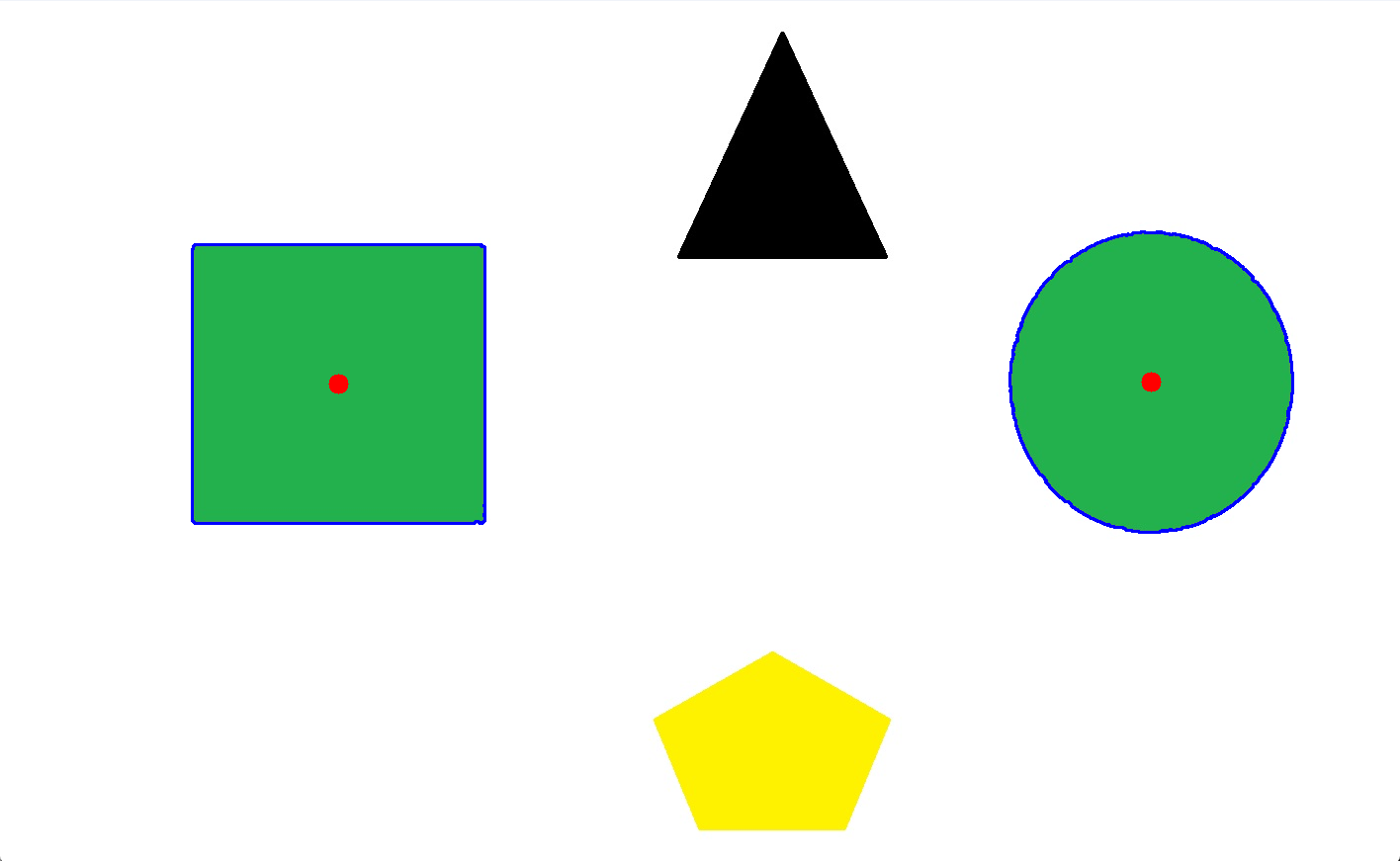


Рисунок 21

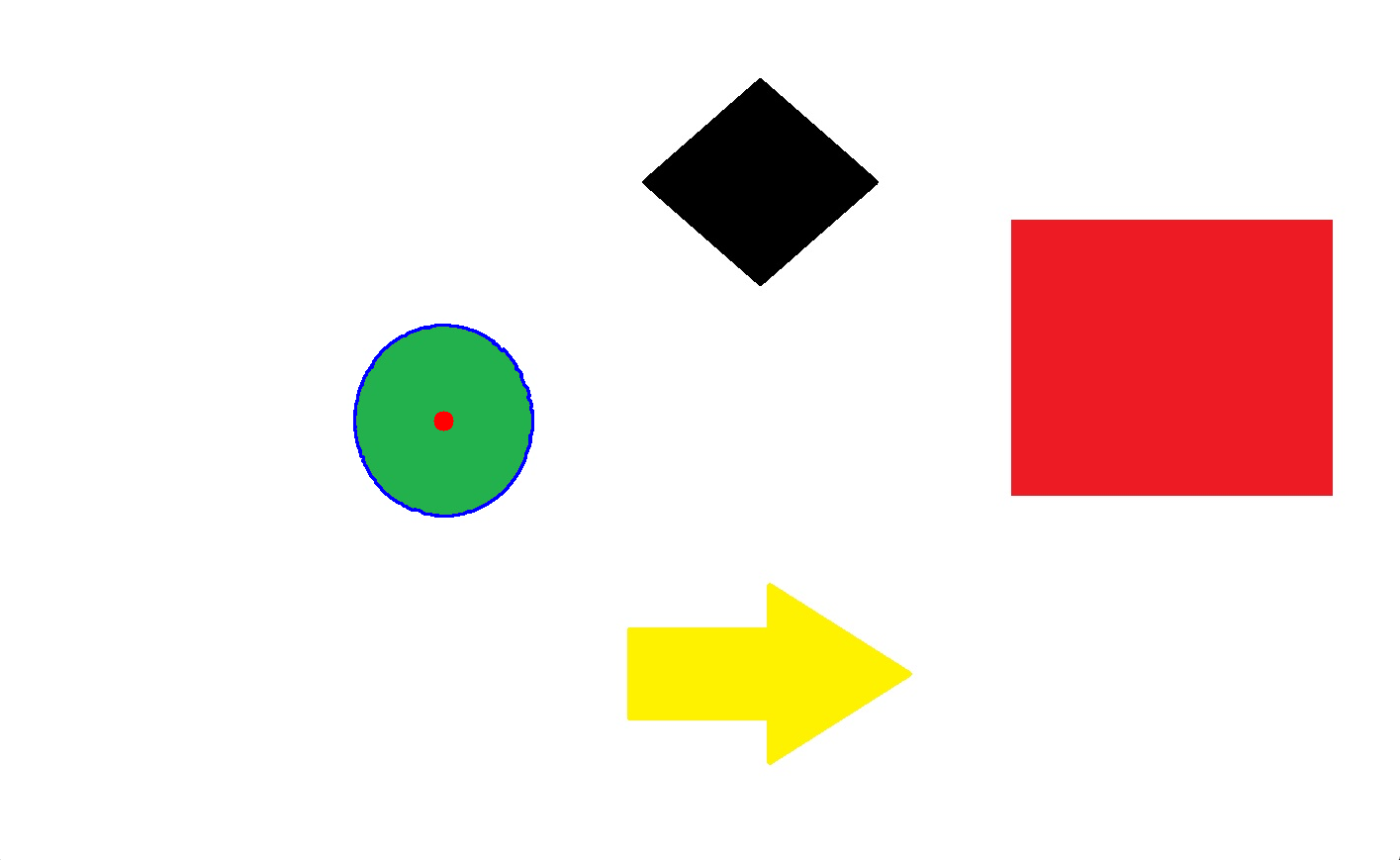


Рисунок 22