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 Ненашева Владислава Романовича

(ФИО, группа, подпись)

Руководитель практики от института ИТиАД

Кононенко Р.В., доцент института ИТиАД

(ФИО, должность, подпись)

Ссылка на резюме обучающего на сайте https://www.superjob.ru/

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

Ссылка на резюме обучающего на сайте https://www.hh.ru/

**Индивидуальное задание на прохождение**

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| для | Ненашева Владислава Романовича | | | | |
|  | (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.В. /

(подпись

**Согласовано:**

Руководитель ООП

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

(подпись

«\_\_\_»\_\_\_\_\_\_\_\_\_\_2025 г.

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

\_\_\_\_\_\_\_\_\_\_«\_\_\_»\_\_\_\_\_\_\_\_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. Пример наиболее короткого маршрута выделен красным цветом, сумма чисел = 10.

Напишите программу, рассчитывающую минимальное время спуска (сумму чисел в пути с вершины до основания).

Формат входных данных: В первой строке дано целое число N - высота пирамиды, далее следуют N строк из чисел, разделённых пробелом (в каждой строке на 1 число больше, чем в предыдущей)

Формат выходных данных: Сумма чисел в пути с вершины до основания (одно число)

Последовательность участков маршрута (числа, разделённые пробелом)

**Ход решения:**

#include <iostream>

#include <vector>

#include <algorithm>

#include <ctime>

// генерируем случайную пирамиду с n уровнями

std::vector<std::vector<int>> generatePyramid(int n) {

  int MAX\_VAL = 100;

  std::vector<std::vector<int>> pyramid;

  for (int i = 0; i < n; ++i) {

    std::vector<int> vec;

    for (int j = 0; j <= i; ++j) {

      vec.push\_back(std::rand() % MAX\_VAL); // случайное число от 0 до 99

    }

    pyramid.push\_back(vec);

  }

  return pyramid;

}

// выводим пирамиду на экран

void printPyramid(const std::vector<std::vector<int>>& pyramid) {

  for (const auto& row : pyramid) {

    for (int val : row) {

      std::cout << val << " ";

    }

    std::cout << std::endl;

  }

}

int main() {

  std::srand(std::time(nullptr));

  int N = 0;

  std::cout << "Высота: ";

  std::cin >> N;  // читаем высоту пирамиды

  auto pyramid = generatePyramid(N); // создаём пирамиду

  printPyramid(pyramid); // показываем пирамиду

  std::cout << std::endl;

  std::vector<std::vector<int>> dp = pyramid; // копируем для вычислений

  std::vector<std::vector<int>> path(N, std::vector<int>(N, -1)); // для восстановления пути

  // считаем минимальную сумму снизу вверх

  for (int i = N - 2; i >= 0; --i) {

    for (int j = 0; j <= i; ++j) {

      // выбираем лучший вариант из двух соседних на следующий уровень

      if (dp[i + 1][j] < dp[i + 1][j + 1]) {

        dp[i][j] += dp[i + 1][j];

        path[i][j] = j; // запоминаем путь

      } else {

        dp[i][j] += dp[i + 1][j + 1];

        path[i][j] = j + 1; // запоминаем путь

      }

    }

  }

  std::cout << "Минимальное время спуска: " << dp[0][0] << std::endl; // выводим минимальную сумму

  std::cout << "Путь: ";

  int col = 0;

  for (int i = 0; i < N; ++i) {

    std::cout << pyramid[i][col] << " "; // выводим элементы пути

    col = path[i][col]; // идём дальше по запомненному пути

  }

  std::cout << std::endl;

  return 0;

}

**Результат:**

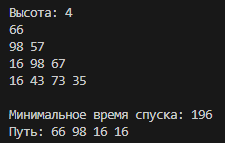


Рисунок 1 – Результаты теста №1 для задачи №1

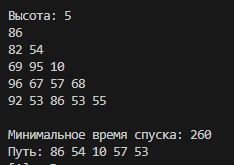


Рисунок 2 – Результаты теста №2 для задачи №1

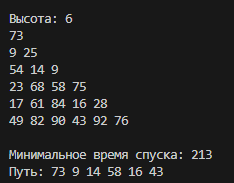


Рисунок 3 – Результаты теста №3 для задачи №1

**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.

Формат входных данных: В первой строке дано целое число N - количество узлов сети и M - число линий связи. Далее следуют M строк из чисел, разделённых пробелом (узлы, которые связывает данная линия)

Формат выходных данных:Число необходимых линий связи (одно число)

**Ход решения:**

#include <iostream>

#include <vector>

// обход в глубину для пометки связной компоненты

void DFS(int v, const std::vector<std::vector<int>>& graph, std::vector<bool>& visited) {

  visited[v] = true;

  for (int u : graph[v]) {

    if (!visited[u]) {

      DFS(u, graph, visited);

    }

  }

}

int main() {

  int n = 0;

  int m = 0;

  std::cout << "Введите число узлов сети (N) и рёбер (M):";

  std::cin >> n >> m;  // читаем число узлов и рёбер

  std::cout << "Введите рёбра:";

  std::vector<std::vector<int>> graph(n); // список смежности

  for (int i = 0; i < m; ++i) {

    int u = 0;

    int v = 0;

    std::cin >> u >> v;

    --u; --v; // переходим к индексам с нуля

    graph[u].push\_back(v);

    graph[v].push\_back(u);

  }

  std::vector<bool> visited(n, false); // отмечаем посещённые вершины

  int components = 0;

  // считаем количество компонент связности

  for (int i = 0; i < n; ++i) {

    if (!visited[i]) {

      DFS(i, graph, visited); // запускаем dfs из новой вершины

      ++components;

    }

  }

  std::cout << "Число необходимых связей: " << (components - 1) << std::endl; // нужно добавить (k - 1) рёбер

  return 0;

}

**Результат:**

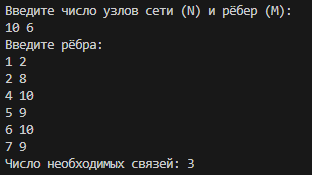


Рисунок 4 – Результаты теста №1 для задачи №2

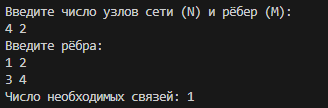


Рисунок 5 – Результаты теста №2 для задачи №2

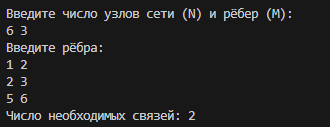


Рисунок 6 – Результаты теста №3 для задачи №2

**Задание на практику № 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.

Формат входных данных: Первая строка входных данных содержит два целых числа – n (1 или больше) и k (0 или больше) это количество чисел в массиве и количество запросов соответственно.

Следующие k строк содержат:

* либо 1 i x – Учетная информация о количестве, выпавшего на улице i (больше 0) x миллиметров снега.
* либо 2 u r – Запрос на подсчет количества снега на улицах от u до r (u и r больше 0 и могут быть равны друг другу)

Формат выходных данных: На каждый запрос второго типа надо вывести единственное число – суммарное выпавшего на них снега с момента начала наблюдения.

**Ход решения:**

#include <iostream>

#include <vector>

int main() {

  int n = 0;

  int k = 0;

  std::cout << "Введите количество улиц и число запросов:";

  std::cin >> n >> k; // читаем количество улиц и запросов

  std::vector<int> snow(n + 1, 0); // массив для хранения количества снега на каждой улице

  std::cout << "Введите запросы: ";

  for (int i = 0; i < k; ++i) {

    int type = 0;

    std::cin >> type; // читаем тип события: 1 — добавить снег, 2 — запрос суммы

    if (type == 1) {

      int index = 0;

      int amount = 0;

      std::cin >> index >> amount;

      snow[index] += amount; // добавляем снег на улице index

    } else if (type == 2) {

      int l = 0;

      int r = 0;

      std::cin >> l >> r;

      int sum = 0;

      for (int j = l; j <= r; ++j) {

        sum += snow[j]; // считаем сумму снега от l до r включительно

      }

      std::cout << "Результат: " << sum << std::endl; // выводим результат запроса

    }

  }

  return 0;

}

**Результат:**

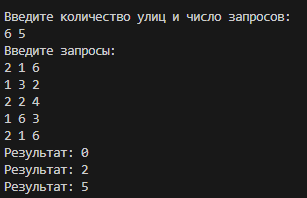


Рисунок 7 – Результаты теста №1 для задачи №3

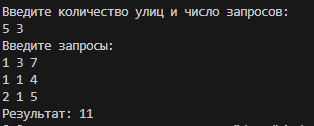


Рисунок 8 – Результаты теста №2 для задачи №3

**Задание на практику № 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:**

#include <iostream>

#include <vector>

int main() {

  int n = 0;

  int k = 0;

  std::cout << "Введите длину слова и количество перестановок: ";

  std::cin >> n >> k; // читаем длину слова и количество применений перестановки

  std::vector<int> permutation(n); // вектор для хранения перестановки

  std::string word;

  std::cout << "Введите перестановку: ";

  for (int i = 0; i < n; i++) {

    std::cin >> permutation[i]; // читаем перестановку

  }

  std::cout << "Введите зашифрованное слово: ";

  std::cin >> word; // читаем зашифрованное слово

  std::string word2 = word; // временное слово для следующей итерации

  for (int j = 0; j < k; j++) {

    for (int i = 0; i < n; i++) {

      word2[permutation[i] - 1] = word[i]; // по перестановке ставим буквы на нужные позиции

    }

    word = word2; // обновляем текущее слово

  }

  std::cout << "Восстановленное слово: ";

  std::cout << word; // выводим восстановленное слово

  return 0;

}

**Результат:**

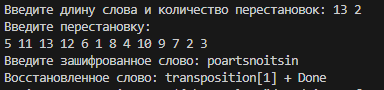


Рисунок 9 – Результаты теста №1 для задачи №4

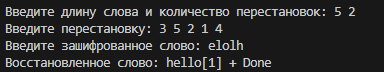


Рисунок 10 – Результаты теста №2 для задачи №4

**Задание на практику №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:**

#include <iostream>

#include <vector>

int main() {

  int h = 0;

  int w = 0;

  std::cout << "Введите высоту и ширину матрицы: ";

  std::cin >> h >> w; // читаем размеры матрицы

  std::vector<std::vector<int>> matrix(h, std::vector<int>(w)); // создаём матрицу h×w

  std::cout << "Введите значения матрицы: ";

  for (int i = 0; i < h; i++) {

    for (int j = 0; j < w; j++) {

      std::cin >> matrix[i][j]; // считываем значения в матрицу

    }

  }

  int min\_i = h;  // минимальная строка, где встретилась 1

  int min\_j = w;  // минимальный столбец, где встретилась 1

  int max\_i = -1; // максимальная строка, где встретилась 1

  int max\_j = -1; // максимальный столбец, где встретилась 1

  for (int i = 0; i < h; i++) {

    for (int j = 0; j < w; j++) {

      if (matrix[i][j] == 1) {

        if (i < min\_i) {

          min\_i = i; // обновляем верхнюю границу

        }

        if (j < min\_j) {

          min\_j = j; // обновляем левую границу

        }

        if (i > max\_i) {

          max\_i = i; // обновляем нижнюю границу

        }

        if (j > max\_j) {

          max\_j = j; // обновляем правую границу

        }

      }

    }

  }

  std::cout << "Координаты прямоугольника: " << min\_i - 1 << ' ' << min\_j - 1 << ' ' << max\_i + 1 << ' ' << max\_j + 1; // выводим координаты прямоугольника

}

**Результаты:**

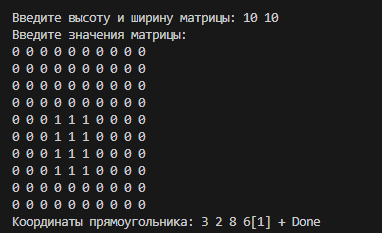


Рисунок 11 – Результаты теста №1 для задачи №5

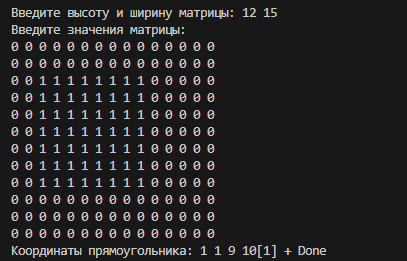


Рисунок 12 – Результаты теста №2 для задачи №5

**Задание на практику №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:**

#include <iostream>

#include <vector>

int main() {

  int n = 0;

  std::cout << "Введите количество запросов: ";

  std::cin >> n; // читаем количество запросов

  std::vector<std::vector<int>> requests(n, std::vector<int>(4)); // создаём массив для хранения всех запросов

  std::cout << "Введите запросы (a, b, x ,y): ";

  for (int i = 0; i < n; i++) {

    for (int j = 0; j < 4; j++) {

      std::cin >> requests[i][j]; // считываем a, b, x, y для каждого запроса

    }

  }

  std::cout << "Результат: ";

  for (int i = 0; i < n; i++) {

    std::vector<int> request = requests[i];

    int a = request[0]; // количество контроллеров A

    int b = request[1]; // количество контроллеров B

    int x = request[2]; // количество модулей типа 1

    int y = request[3]; // количество модулей типа 2

    int max\_b\_pair = 0;

    // пытаемся подключить B только к модулям типа 1 (они совместимы)

    if (b >= x) {

      max\_b\_pair = x; // b хватит на все x

      x = 0; // все модули типа 1 ушли на B

    } else {

      max\_b\_pair = b; // b меньше x, значит используем всех B

      x -= b; // уменьшаем x на использованные

    }

    // теперь A можно подключить к оставшимся x и ко всем y

    int max\_a\_pair = std::min(a, x + y);

    std::cout << max\_b\_pair + max\_a\_pair << ' '; // выводим общее количество пар

  }

  return 0;

}

**Результаты:**

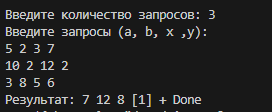


Рисунок 13 – Результаты теста №1 для задачи №6

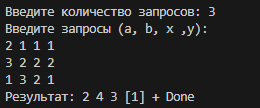


Рисунок 14 – Результаты теста №2 для задачи №6

**Задание на практику №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:**

#include <iostream>

int main() {

  std::string word;

  std::cout << "Введите строку из шести символов: ";

  std::cin >> word; // считываем строку из 6 символов

  // проверяем, соответствует ли строка формату номера:

  // первая буква, потом три цифры, потом две буквы

  std::cout << "Результат: ";

  if (std::isalpha(word[0]) && std::isdigit(word[1]) &&

      std::isdigit(word[2]) && std::isdigit(word[3]) &&

      std::isalpha(word[4]) && std::isalpha(word[5])) {

    std::cout << "Yes"; // все ок

  } else {

    std::cout << "No"; // не ок

  }

  return 0;

}

**Результаты:**



Рисунок 15 – Результаты теста №1 для задачи №7



Рисунок 16 – Результаты теста №2 для задачи №7

**Задание на практику №8**

**Постановка задачи:**

Составить светодиодную матрицу размером не менее 8 на 8 светодиодов.

На матрицу вывести инфографику с различными динамично меняющимися изображениями.

**Ход решения:**

#include <FastLED.h>  // подключаем библиотеку для управления светодиодами

#define NUM\_LEDS 64   // количество светодиодов в ленте

#define DATA\_PIN 3    // пин arduino, к которому подключена лента

CRGB leds[NUM\_LEDS]; // массив для хранения цвета каждого светодиода

void setup() {

  FastLED.addLeds<WS2812, DATA\_PIN, GRB>(leds, NUM\_LEDS); // настраиваем ленту

  FastLED.setBrightness(100); // устанавливаем яркость (0-255)

}

void loop() {

  rainbowCycle(10); // запускаем эффект радуги с задержкой 10 мс

}

void rainbowCycle(uint8\_t wait) {

  static uint8\_t hue = 0; // переменная цвета, которая будет меняться

  // проходим по всем светодиодам

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

    leds[i] = CHSV(hue + (i \* 256 / NUM\_LEDS), 255, 255); // устанавливаем цвет светодиода

  }

  FastLED.show(); // обновляем цвета на ленте

  delay(wait);    // ждём указанное время

  hue++;          // сдвигаем цвета, чтобы создать движение радуги

}

**Результаты:**[**https://wokwi.com/projects/434249637306918913**](https://wokwi.com/projects/434249637306918913)

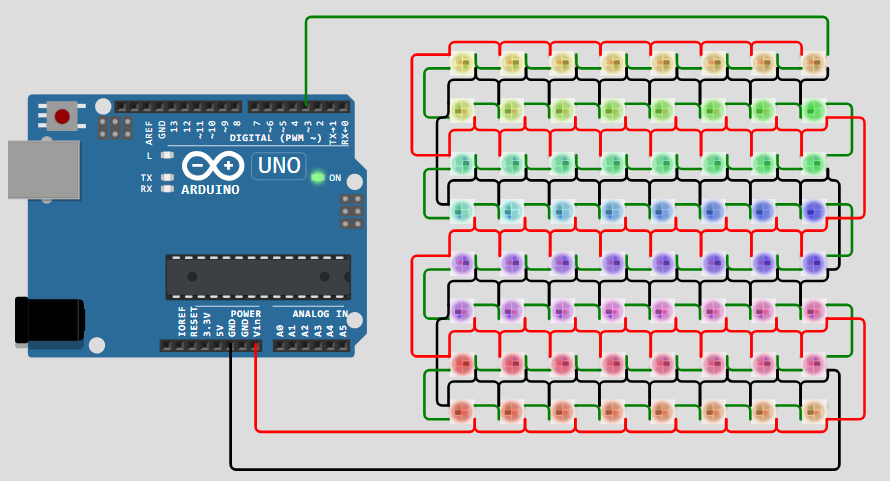


Рисунок 17 – Светодиодная матрица с переливающейся радугой, момент 1

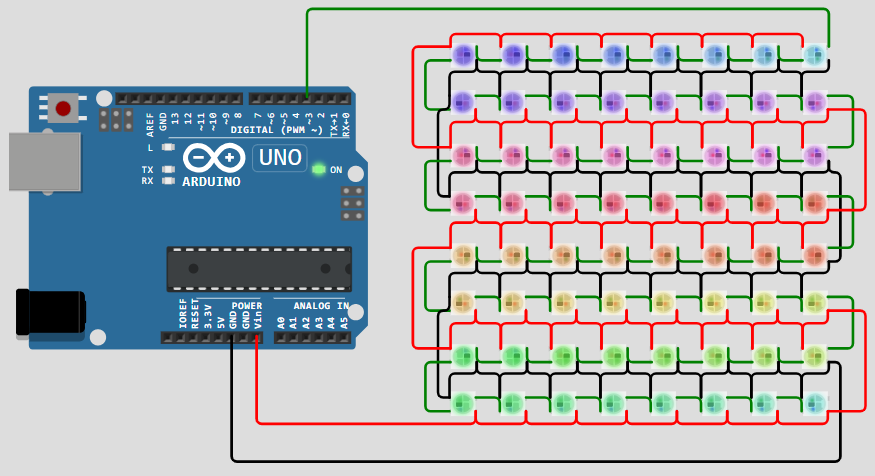


Рисунок 18 – Светодиодная матрица с переливающейся радугой, момент 2

**Задание на практику №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.

Зеленый светодиод – двери отрываются.

Красный светодиод ­– двери закрываются.

Фоторезистор имитируют процесс приближения-удаления человека от дверей.

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:**

// пины подключения светодиодов и фоторезистора

#define RED\_LED 9       // красный светодиод - двери закрыты

#define GREEN\_LED 5     // зеленый светодиод - двери открыты

#define LDR A0          // фоторезистор на аналоговый пин

#define THRESHOLD 512   // порог, при котором дверь открывается

#define OPEN\_TIME 3000  // минимальное время, на которое дверь остаётся открытой (мс)

bool doorIsOpen = false;       // дверь сейчас открыта или закрыта

unsigned long openTimer = 0;   // таймер для минимального времени открытия

void setup() {

  pinMode(RED\_LED, OUTPUT);

  pinMode(GREEN\_LED, OUTPUT);

  pinMode(LDR, INPUT);

  // по умолчанию дверь закрыта

  digitalWrite(RED\_LED, HIGH);

  digitalWrite(GREEN\_LED, LOW);

  Serial.begin(9600);

  Serial.println("автоматические двери запущены");

}

void loop() {

  int lightValue = analogRead(LDR);  // считываем значение освещенности

  // если Lux больше порога и дверь закрыта, открываем дверь

  if (lightValue > THRESHOLD && !doorIsOpen) {

    doorIsOpen = true;

    openTimer = millis();           // запускаем таймер

    digitalWrite(GREEN\_LED, HIGH);

    digitalWrite(RED\_LED, LOW);

    Serial.println("дверь открывается!");

  }

  // если дверь открыта

  if (doorIsOpen) {

    // проверяем, прошло ли минимальное время OPEN\_TIME

    if (millis() - openTimer >= OPEN\_TIME) {

      // если человек ушёл, закрываем дверь сразу

      if (lightValue <= THRESHOLD) {

        doorIsOpen = false;

        digitalWrite(GREEN\_LED, LOW);

        digitalWrite(RED\_LED, HIGH);

        Serial.println("дверь закрыта");

      }

      // иначе, оставляем дверь открытой и таймер не сбрасываем, чтобы минимальное время уже прошло

    }

  }

  delay(50); // пауза для стабильной работы

}

**Результаты:**

[**https://wokwi.com/projects/442094862730727425**](https://wokwi.com/projects/442094862730727425)

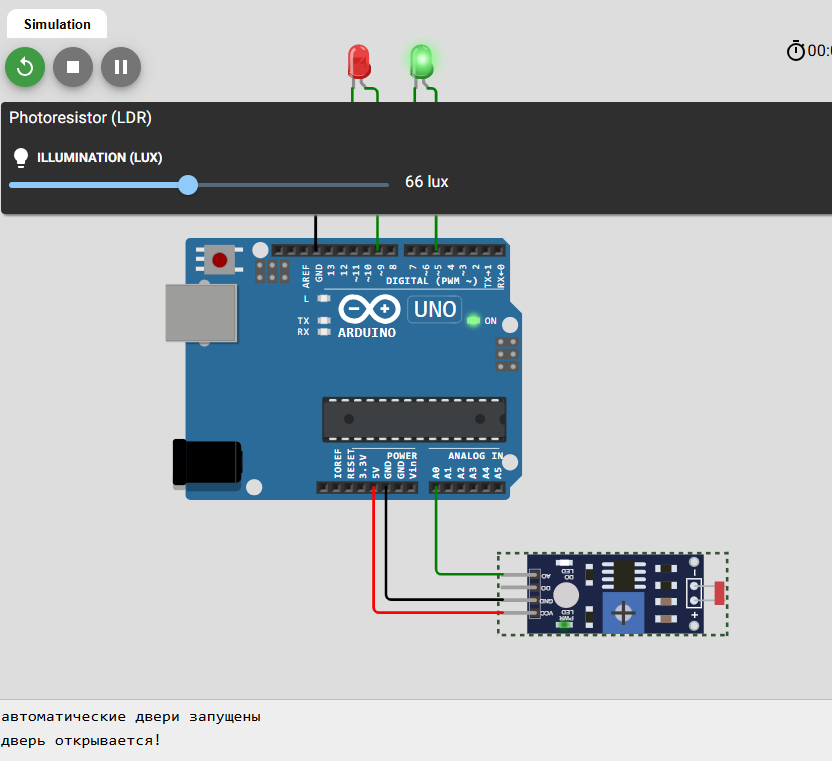


Рисунок 19 – Состояние двери: дверь открыта (зелёный светодиод горит)

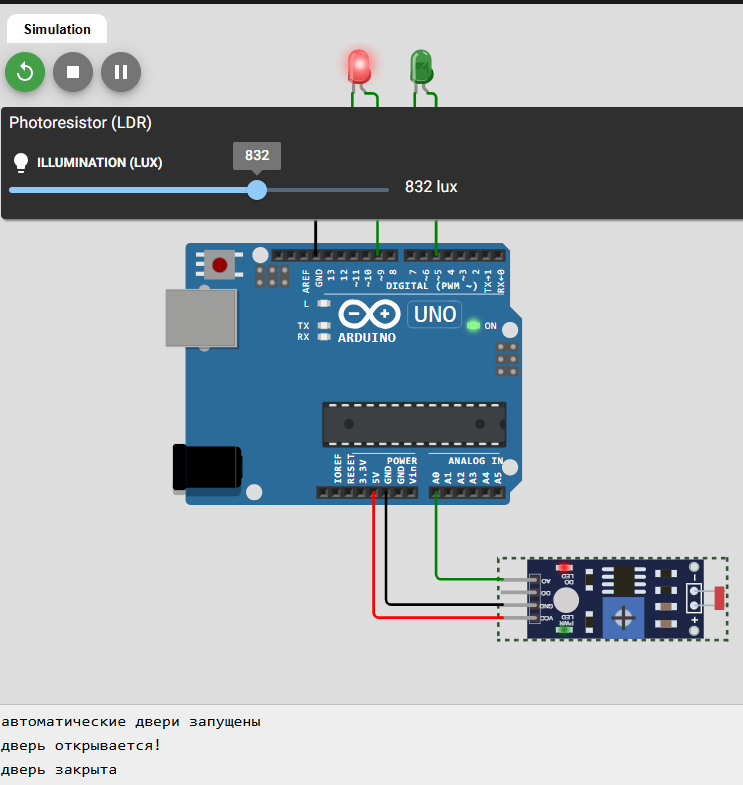


Рисунок 20 – Состояние двери: дверь закрыта (красный светодиод горит)

**Задание на практику №10**

**Постановка задачи:**

Задачи:

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

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 myServo;          // объект для сервопривода

int currentAngle = 0;   // текущий угол сервопривода

void setup() {

  Serial.begin(9600);       // запускаем последовательный порт

  myServo.attach(3);        // подключаем сервопривод к пину 9

  myServo.write(currentAngle); // ставим начальный угол 0

  Serial.println("введите угол от 0 до 180:");

}

void loop() {

  // проверяем, есть ли данные в serial

  if (Serial.available() > 0) {

    String input = Serial.readStringUntil('\n'); // считываем строку до перевода строки

    input.trim(); // убираем пробелы и символы переноса строки

    // проверяем, состоит ли ввод только из цифр

    bool isNumber = true;

    for (unsigned int i = 0; i < input.length(); i++) {

      if (!isDigit(input[i])) {

        isNumber = false;

        break;

      }

    }

    if (isNumber) {

      int angle = input.toInt(); // преобразуем строку в число

      if (angle >= 0 && angle <= 180) {

        // если угол отличается от текущего, плавно поворачиваем сервопривод

        if (angle != currentAngle) {

          Serial.print("поворот сервопривода в угол: ");

          Serial.println(angle);

          if (angle > currentAngle) {

            for (int a = currentAngle; a <= angle; a++) {

              myServo.write(a);

              delay(10); // скорость поворота (можно менять)

            }

          } else {

            for (int a = currentAngle; a >= angle; a--) {

              myServo.write(a);

              delay(10);

            }

          }

          currentAngle = angle; // обновляем текущий угол

        }

      } else {

        Serial.println("ошибка: угол должен быть от 0 до 180");

      }

    } else {

      Serial.println("ошибка: вводите только числа");

    }

    Serial.println("введите угол от 0 до 180:");

  }

}

**Результаты:**

[**https://wokwi.com/projects/442097254312565761**](https://wokwi.com/projects/442097254312565761)

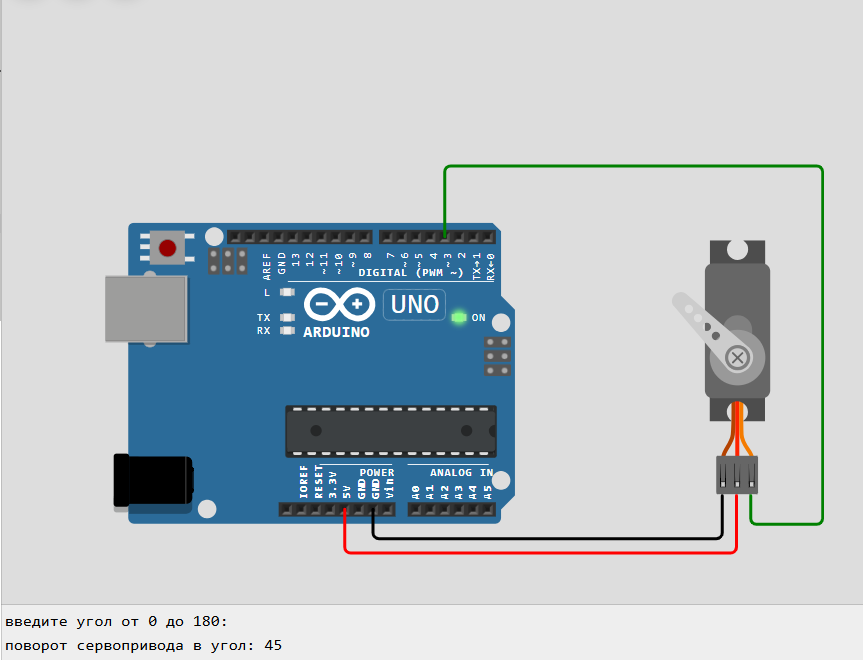


Рисунок 21 – Поворот сервопривода на 45 градусов

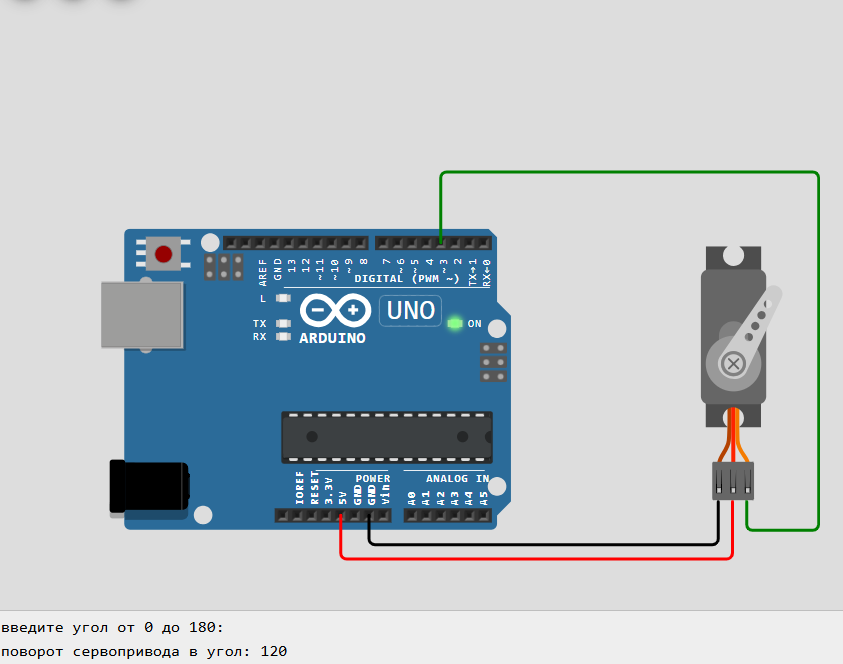


Рисунок 22 – Поворот сервопривода на 120 градусов