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).

**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

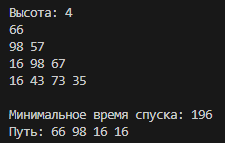
  }

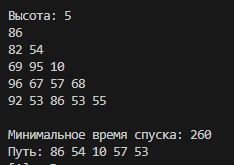
  std::cout << std::endl;

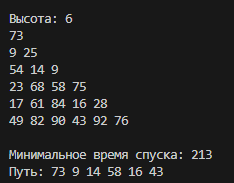
  return 0;

}

**Результат:**







**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:**

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

  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 << (components - 1) << std::endl; // нужно добавить (k - 1) рёбер

  return 0;

}

**Результат:**







**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.

**Ход решения:**

#include <iostream>

#include <vector>

int main() {

  int n = 0;

  int k = 0;

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

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

  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 << sum << std::endl; // выводим результат запроса

    }

  }

  return 0;

}

**Результат:**

