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 Никитеева В.Д., ИСИБ-24-1

(ФИО, группа, подпись)

Руководитель практики от института ИТиАД

Кононенко Р.В., доцент института ИТиАД

(ФИО, должность, подпись)

Руководитель образовательной программы

https://irkutsk.hh.ru/resume/b7b8e3bcff0efccbde0039ed1f77567145514f

Кононенко Р.В., доцент института ИТиАД

(ФИО, должность, подпись)

**** Оценка по практике

(ФИО, подпись, дата)

Содержание отчета на \_\_\_ стр.

Приложение к отчету на \_\_\_ стр.

https://irkutsk.superjob.ru/resume/razrabotchik-c-55731839.html

Иркутск 2025

**Индивидуальное задание на прохождение**

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| для | Никитеевой Виктории Дмитриевны | | | | |
|  | (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(а), задание принято к исполнению

\_\_\_\_\_\_\_\_\_\_«\_\_\_»\_\_\_\_\_\_\_\_2025 г.

(подпись

**ДНЕВНИК**

прохождения практики

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| обучающегося | | | | | Никитеевой Виктории Дмитриевны,  ИСИБ-24-1 |
|  | | | | | (фамилия, имя, отчество, группа) |
| курс | | 1 | | | |
| направление | | | | Информатика и вычислительная техника | |
| профиль | | | Интеллектуальные системы обработки | | |
| информации и управления | | | | | |
| в | ИРНИТУ | | | | |
|  | (наименование профильной организации) | | | | |

Иркутск 2025

Руководителем практики от структурного подразделения назначен:

Кононенко Роман Владимирович, доцент института ИТиАД

(ФИО, должность)

**Рабочий график (план) прохождения практической подготовки**

(заполняется обучающимся)

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Период  практики | Содержание выполненных работ | Подпись руководителя практики от структурного подразделения |
| 1 | 16.06.2025 | Решила задачу №1,  Решила задачу №2, |  |
| 2 | 17.06.2025 | Решила задачу №3. |  |
| 3 | 18.06.2025 | Решила задачу №4,  Решила задачу №5. |  |
| 4 |  |  |  |
| 5 |  |  |  |
| 6 |  |  |  |
| 7 |  |  |  |
| 8 |  |  |  |
| 9 |  |  |  |
| 10 |  |  |  |

|  |  |
| --- | --- |
| Дата фактического прибытия |  |
| обучающегося в структурное подразделение | 16.06.2025 |
| Дата фактического убытия |  |
| обучающегося из структурного подразделения | 28.06.2025 |

|  |  |
| --- | --- |
| Руководитель образовательной программы | Кононенко Р.В. |
|  | (ФИО, подпись) |
| Директор института | Говорков А.С. |
|  | (ФИО, подпись) |

**Задание №1**

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).

**Алгоритм программы:**

1. *Генерирование горы (Generator)*: пользователь вводит высоту горы. Функция создает список строк, где каждая строка содержит на одно число больше предыдущей. Числа будем генерировать случайные (от 1 до 1000).
2. *Нахождение минимального пути (FindMinPath)*: используем динамическое программирование и будем идти снизу вверх, где dp[i][j] содержит минимальную сумму от вершины pyramid[i][j] до основания. На каждом шаге будем рассматривать минимальный из двух возможных путей вниз.
3. *Восстановление пути*: после вычислений восстанавливаем путь по значению dp, начиная от вершины и выбирая минимальный соседний элемент на каждом уровне.
4. *Вывод*: выводим последовательность вершин, соответствующую восстановленному минимальному пути, вместе с минимальной суммой времени.

**Код программы:**

#include <iostream>

#include <vector>

#include <cstdlib>

#include <ctime>

#include <algorithm>

// Функция генерации горы со случайными числами в диапазоне [1, 1000].

std::vector<std::vector<int>> Generator(int size) {

  std::vector<std::vector<int>> mountain; // создаем гору.

  for (int i = 0; i < size; i++) {

    std::vector<int> row; // временный вектор, содержащий значения текущего уровня горы.

    for (int j = 0; j <= i; j++) {

      row.push\_back(rand() % 1000 + 1); // генерация случаного числа.

    }

    mountain.push\_back(row); // добавляем число в текущую строку row.

  }

  return mountain;

}

// Поиск минимального пути.

std::pair<int, std::vector<int>> FindMinPath(const std::vector<std::vector<int>>& mountain) {

  int size = mountain.size(); // храним высоту горы.

  std::vector<std::vector<int>> dp = mountain; // создаем копию горы, в которую будем записывать минимальные суммы на каждом шаге.

  // Строим dp снизу вверх.

  for(int i = size - 2; i >= 0; --i) {

    for (int j = 0; j <= i; j++) {

      dp[i][j] += std::min(dp[i+1][j], dp[i + 1][j + 1]);

      // обновляем значения по самому минимальному соседу.

      //В dp[0][0] будет храниться минимальная сумма пути от вершины до основания.

    }

  }

  // Восстановление пути.

  std::vector<int> path;

  path.push\_back(mountain[0][0]); // добавляем верхний элемент.

  int col = 0; // индекс, по которому отслеживаем, где сейчас находимся.

  for (int i = 1; i < size; i++) {

    if (dp[i][col] < dp[i][col + 1]) { // смотрим соседей.

      path.push\_back(mountain[i][col]); // идем влево вниз.

    } else {

      path.push\_back(mountain[i][col + 1]); // идем вправо вниз.

      col++; // сдвигаем индекс вправо.

    }

  }

  return {dp[0][0], path};

  // возвращаем пару значений, dp[0][0] — это минимальная сумма пути от вершины до основания,

  //path — последовательность чисел, по которым проходил минимальный путь.

}

int main() {

  srand(time(0)); // инициализация генератора случайных чисел.

  int size = 0;

  std::cout << "Высота: ";

  std::cin >> size;

  auto mount = Generator(size); // генерация горы.

  // вывод горы.

  for (const auto& row : mount) {

    for (int num : row) {

      std::cout << num << " ";

    }

    std::cout << std::endl;

  }

  // поиск кратчайшего пути.

  auto result = FindMinPath(mount);

  int min = result.first;

  std::vector<int> path = result.second;

  // вывод результата.

  std::cout << "Минимальная сумма: " << min << std::endl;

  std::cout << "Путь: ";

  for (int num : path) {

    std::cout << num << " ";

  }

  std::cout << std::endl;

  return 0;

}

**Таблица тестов:**

|  |  |  |
| --- | --- | --- |
| Ввод | 4 | 5 |
| Вывод | 968  212 427  912 914 221  630 806 272 292  Минимальная сумма: 1888  Путь: 968 427 221 272 | 928  303 665  64 211 868  49 166 741 533  25 969 242 782 80  Минимальная сумма: 1369  Путь: 928 303 64 49 25 |

****

**Задача №2**

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.

**Алгоритм программы:**

1. ***Представление сети в виде графа:*** вершины графа соответствуют узлам сети, а рёбра — существующим линиям связи между ними.
2. ***Поиск компонент связности:*** с помощью обхода графа в глубину (DFS) программа определяет количество компонент связности в графе. Компонента связности — это множество узлов, которые можно достичь друг из друга, следуя по существующим линиям связи.
3. *Подсчёт дополнительных линий связи:* если в графе существует более одной компоненты связности, то необходимо построить дополнительные линии связи для объединения этих компонент. Минимальное количество таких линий равно количеству компонент минус один.
4. *Вывод результата:* программа выводит минимальное количество линий связи, которые нужно добавить, чтобы сделать граф связным, то есть чтобы все узлы были соединены в единую сеть.

**Код программы:**

#include <iostream>

#include <vector>

#include <list>

// используем функцию DFS для обхода графа в глубину и поиска всех вершин.

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

  visited[v] = true; // стартовая вершина изначально считается посещенной.

  for (int u : adj[v]) { // проходим по всем соседям вершины v.

    if (!visited[u]) { // если сосед еще не посещен, запускаем рекурсию.

      DFS(u, adj, visited);

    }

  }

}

// Функция для подсчета количества компонент связности.

int CountConnectedComponents(int N, const std::vector<std::pair<int, int>>& edges) {

  // Создаем список смежности.

  std::vector<std::list<int>> adj(N + 1); // Нумерация узлов с 1 до N.

  // Заполняем список смежности

  for (const auto& edge : edges) {

    int u = edge.first;

    int v = edge.second;

    adj[u].push\_back(v);

    adj[v].push\_back(u); // граф неориентированный.

  }

  std::vector<bool> visited(N + 1, false); // массив для отслеживания посещенных вершин.

  int components = 0; // счетчик компонент.

  // запускаем DFS из каждой непосещенной вершины.

  for (int i = 1; i <= N; ++i) {

    if (!visited[i]) {

      DFS(i, adj, visited);

      components++; // увеличиваем количество компонент.

    }

  }

  return components;

}

int main() {

  int n = 0;

  int m = 0;

  std::cin >> n >> m;

  //заполнения списка ребер

  std::vector<std::pair<int, int>> edges;

  for (int i = 0; i < m; ++i) {

    int u = 0;

    int v = 0;

    std::cin >> u >> v;

    edges.emplace\_back(u, v);

  }

  int components = CountConnectedComponents(n, edges);

  // Чтобы объединить все компоненты, нужно (components - 1) новых рёбер

  std::cout << components - 1 << "\n";

  return 0;

}

**Таблица тестов:**

|  |  |  |
| --- | --- | --- |
| Ввод | 10 6  1 2  2 8  4 10  5 9  6 10  7 9 | 7 4  1 2  2 3  4 5  6 7 |
| Вывод | 3 | 2 |



**Задача №3**

В Иркутске раз в году наступает зима. Несмотря 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.

1. Информация о снеге передается вам в виде тройки чисел – 1 в качестве идентификатора события, уникального индекса улицы и количество миллиметров выпавшего снега.
2. 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.

**Алгоритм программы:**

1. *Инициализация:* создадим структуру запросов.
2. *Обработка запросов:* Программа принимает m запросов двух типов:

* Тип 1: обновляет количество снега на конкретной улице a. Значение y записывается в соответствующий индекс массива.
* Тип 2: вычисляет и выводит сумму снега на отрезке улиц от a до b (включительно). Программа проходит по элементам массива в этом диапазоне и суммирует значения.

1. *Вывод результатов:* выводим результат суммирования по запросам типа 2.

**Код программы:**

#include <iostream>

#include <vector>

// Структура для хранения запроса.

struct Query {

  int type;

  int a;

  int b;

};

// Обработка всех запросов.

std::vector<int> process\_queries(int n, const std::vector<Query>& queries) {

  std::vector<int> snow(n + 1, 0); // Вектор для хранения количества снега.

  std::vector<int> results; // Результаты для запросов типа 2.

  for (const Query& q : queries) {

    if (q.type == 1) {

      // Запрос: добавить снег на улице.

      snow[q.a] += q.b;

    } else if (q.type == 2) {

        // Запрос: посчитать сумму на отрезке.

        int sum = 0;

        for (int i = q.a; i <= q.b; ++i) {

          sum += snow[i];

        }

        results.push\_back(sum); // Сохраняем результат.

    }

  }

  return results;

}

int main() {

  int n = 0;

  int k = 0;

  std::cin >> n >> k;

  std::vector<Query> queries;

  // Считываем все запросы

  for (int i = 0; i < k; ++i) {

    int type, a, b;

    std::cin >> type >> a >> b;

    queries.push\_back({type, a, b});

  }

  // Обрабатываем запросы и получаем результаты

  std::vector<int> results = process\_queries(n, queries);

  // Выводим результаты

  for (int res : results) {

    std::cout << res << '\n';

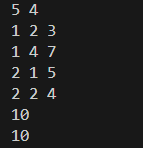
  }

  return 0;

}

**Таблица тестов:**

|  |  |  |
| --- | --- | --- |
| Ввод | 6 5  2 1 6  1 3 2  2 2 4  1 6 3  2 1 6 | 0  2  5 |
| Вывод | 5 3  1 3 7  1 1 4  2 1 5 | 11 |



**Задача №4**

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.

**Алгоритм программы:**

1. *Входные данные:*

* Размер строки n и количество итераций k.
* Перестановка p.
* Зашифрованная строка.

1. *Описание перестановки:* перестановка P представляет собой набор чисел от 1 до n, где каждое число соответствует новому порядку символов в слове. Например, если первая позиция перестановки содержит число 5, это означает, что первый символ новой строки будет взят с пятой позиции исходной строки.
2. *Процесс декодирования:*

* Строим обратную перестановку: для каждой i позиции находим invp[p[i] – 1] = i + 1.
* Применяем invp к строке k раз (перемешиваем символы обратно).

1. *Результат:* после применения перестановки m раз, программа выводит восстановленное слово, которое является результатом декодирования зашифрованного слова.

**Код программы:**

#include <iostream>

#include <vector>

#include <string>

// Функция для вычисления обратной перестановки.

std::vector<int> Iniverse(std::vector<int>& p) {

  int n = p.size();

  std::vector<int> invP(n); // вектор для хранения обратной перестановки.

  for (int i = 0; i < n; i++) {

    invP[p[i] - 1] = i + 1; // выполняем перестановку.

  }

  return invP;

}

// Функция для применения перестановки p к строке s.

std::string Permutation(std::string& s, std::vector<int>& p) {

  int n = p.size();

  std::string result(n, ' '); // Создаём новую строку длины n, заполненную пробелами.

  for (int i = 0; i < n; i++) {

    // символ с позиции p[i] - 1 переносится на позицию i.

    result[i] = s[p[i] - 1];

  }

  return result;

}

int main() {

  int n = 0;

  int k = 0;

  std::cin >> n >> k;

  std::vector<int> p(n);

  for (int i = 0; i < n; i++) {

    std::cin >> p[i];

  }

  std::string world;

  std::cin >> world;

  std::vector<int> invp = Iniverse(p); // вычисляем обратную перестановку.

  std::string result = world;

  for (int i = 0; i < k; i++) { // k раз применяем обратную перестановку.

    result = Permutation(result, invp);

  }

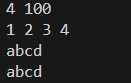
  std::cout << result << "\n";

  return 0;

}

**Таблица тестов:**

|  |  |  |
| --- | --- | --- |
| Ввод | 13 2  5 11 13 12 6 1 8 4 10 9 7 2 3  poartsnoitsin | 5 1  2 3 4 5 1  bcdea |
| Вывод | transposition | abcde |



**Задача №5**

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.

**Алгоритм программы:**

1. *Ввод данных:*

* Считываем высоту h и ширину w матрицы.
* Вводим саму матрицу размером h x w, содержащую 0 и 1.

1. *Поиск границ фигуры:*

* Инициализируем переменные:
* min\_row = INT\_MAX — минимальный индекс строки, где есть 1;
* max\_row = INT\_MIN — максимальный индекс строки, где есть 1;
* min\_col = INT\_MAX — минимальный индекс столбца, где есть 1;
* max\_col = INT\_MIN — максимальный индекс столбца, где есть 1.
* Проходим по каждому элементу матрицы. Если элемент равен 1, обновляем границы:
* min\_row = min(min\_row, i);
* max\_row = max(max\_row, i);
* min\_col = min(min\_col, j);
* max\_col = max(max\_col, ).

1. Расчёт и вывод координат:

* Для того чтобы фигура не касалась сторон прямоугольника, уменьшаем на 1 значения min\_row и min\_col, а значения max\_row и max\_col увеличиваем на 1.
* Выводим координаты верхнего левого угла (min\_row - 1, min\_col - 1) и нижнего правого угла (max\_row + 1, max\_col + 1).

**Код программы:**

#include <iostream>

#include <vector>

#include <climits>

int main() {

  int h = 0;

  int w = 0;

  std::cin >> h >> w;

  std::vector<std::vector<int>> matrix(h, std::vector<int>(w));

  // Считываем матрицу

  for (int i = 0; i < h; ++i) {

    for (int j = 0; j < w; ++j) {

      std::cin >> matrix[i][j];

    }

  }

  // Начальные значения для поиска минимума и максимума

  int minRow = INT\_MAX, maxRow = INT\_MIN;

  int minCol = INT\_MAX, maxCol = INT\_MIN;

  // Проходим по всей матрице

  for (int i = 0; i < h; ++i) {

    for (int j = 0; j < w; ++j) {

      if (matrix[i][j] == 1) {

        // Обновляем границы прямоугольника

        if (i < minRow) minRow = i;

        if (i > maxRow) maxRow = i;

        if (j < minCol) minCol = j;

        if (j > maxCol) maxCol = j;

      }

    }

  }

  // Выводим координаты: верхний левый угол и нижний правый угол

  std::cout << minRow - 1 << " " << minCol - 1 << " " << maxRow + 1 << " " << maxCol + 1 << std::endl;

  return 0;

}

**Таблица тестов:**

|  |  |  |
| --- | --- | --- |
| Ввод | 10 10  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 1 1 1 0 0 0 0  0 0 0 1 1 1 0 0 0 0  0 0 0 1 1 1 0 0 0 0  0 0 0 1 1 1 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 | 10 10  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 1 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 |
| Вывод | 3 2 8 6 | 3 3 5 5 |