Федеральное государственное автономное образовательное учреждение высшего образования

Санкт-Петербургский Политехнический университет Петра Великого Физико-Механический институт

Лабораторная 2 – Графы

**«Вариант 6 – Алгоритм Флойда-Уоршалла»**

Выполнил студент гр. 5030102/20101: Комаров Е. Р. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Преподаватель: Новиков Ф. А. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Работа принята: Дата \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Санкт-Петербург

2024

Содержание

[1. Введение 3](#_Toc186818360)

[1.1. Цели и задачи лабораторной работы: 3](#_Toc186818361)

[1.2. Краткое описание алгоритма Флойда-Уоршалла: 3](#_Toc186818362)

[2. Теоретическая часть 4](#_Toc186818363)

[2.1. Основные понятия графов: 4](#_Toc186818364)

[2.2. Описание алгоритма Флойда-Уоршалла: 4](#_Toc186818365)

[2.3. Временная и пространственная сложность алгоритма: 4](#_Toc186818366)

[3. Практическая часть 5](#_Toc186818367)

[3.1. Реализация графа 5](#_Toc186818368)

[3.2. Алгоритмы работы с графом 5](#_Toc186818369)

[3.3. Примеры использования 6](#_Toc186818370)

[3.3.1. Матрица изначальных путей 6](#_Toc186818371)

[3.3.2. Итерация 1 7](#_Toc186818372)

[3.3.3. Итерация 2 7](#_Toc186818373)

[3.3.4. Итерация 3 7](#_Toc186818374)

[3.3.5. Итерация 4 8](#_Toc186818375)

[3.3.6. Матрица кратчайших путей 8](#_Toc186818376)

[3.3.7. Реконструкция путей 8](#_Toc186818377)

[3.4. Примеры использования на полном орграфе 8](#_Toc186818378)

[3.4.1. Матрица изначальных путей 8](#_Toc186818379)

[3.4.2. Итерация 1 9](#_Toc186818380)

[3.4.3. Итерация 2 9](#_Toc186818381)

[3.4.4. Итерация 3 10](#_Toc186818382)

[3.4.5. Итерация 4 10](#_Toc186818383)

[3.4.6. Матрица кратчайших путей 11](#_Toc186818384)

[4. Интерфейс приложения 12](#_Toc186818385)

[4.1. Основная функция 12](#_Toc186818386)

[4.2. Пример запуска 12](#_Toc186818387)

[5. Заключение 13](#_Toc186818388)

[6. Приложение 14](#_Toc186818389)

1. Введение
   1. Цели и задачи лабораторной работы:

Целью лабораторной работы является изучение алгоритма Флойда-Уоршалла и его реализация для решения задачи поиска кратчайших путей между всеми парами вершин во взвешенном ориентированном графе.

Задачи:

1. Ознакомление с понятием взвешенных ориентированных графов.
2. Изучение принципов работы алгоритма Флойда-Уоршалла.
3. Реализация алгоритма на языке программирования.
4. Применение программы для обработки входных графов, заданных в виде файлов, и получение корректных результатов.
   1. Краткое описание алгоритма Флойда-Уоршалла:

Алгоритм Флойда-Уоршалла — это динамический алгоритм, который позволяет найти кратчайшие пути между всеми парами вершин в графе. Основная идея состоит в последовательном улучшении расстояний между вершинами через промежуточные вершины. Если граф содержит отрицательные веса рёбер, алгоритм корректно обработает такие случаи, но не подходит для графов с отрицательными циклами.

1. Теоретическая часть
   1. Основные понятия графов:

 Граф — структура данных, состоящая из набора вершин V и набора рёбер E, которые соединяют вершины.

 Ориентированный **граф (орграф)** — граф, в котором каждое ребро имеет направление.

 Взвешенный **граф** — граф, в котором каждому ребру приписан числовой вес.

 Кратчайший **путь** — путь между двумя вершинами графа, минимизирующий сумму весов рёбер.

* 1. Описание алгоритма Флойда-Уоршалла:

Алгоритм выполняется в три этапа:

1. Инициализация матрицы расстояний DDD:
   * Если вершины i и j соединены ребром с весом w, то D[i][j]=w.
   * Если i=j, то D[i][j]=0.
   * Если вершины i и j не соединены, то D[i][j]=∞.
2. Основной цикл:
   * Для каждой промежуточной вершины k:
     + Для каждой пары вершин i и j:

.

1. Проверка на наличие отрицательных циклов:
   * Если на диагонали матрицы D[i][i]<0, то граф содержит отрицательный цикл.
   1. Временная и пространственная сложность алгоритма:

 Временная **сложность**: O(V3), где V — количество вершин графа. Это обусловлено тройным вложенным циклом, который проходит через каждую пару вершин для всех промежуточных вершин.

 Пространственная **сложность**: O(V2), так как для хранения расстояний используется матрица размером V×V.

1. Практическая часть
   1. Реализация графа

В данной лабораторной работе граф представлен в виде **матрицы смежности**, которая создаётся и обрабатывается с использованием языка Python и библиотеки openpyxl для работы с Excel-файлами.

* **Структура хранения графа:**
  + Граф хранится в виде двухмерного массива (списка списков), где:
    - Строки и столбцы соответствуют вершинам графа.
    - Значение в ячейке [i][j] определяет вес ребра из вершины i в вершину j.
    - Если между вершинами ребра нет, значение в ячейке [i][j] равно ∞ (бесконечность).
    - На главной диагонали стоят нули, так как путь от вершины до самой себя равен 0.
* **Генерация графа:**
  + Функция generate\_random\_graphs создаёт случайный граф с заданными параметрами:
    - Количество вершин num\_nodes.
    - Количество рёбер num\_edges.
  + Генерируемые рёбра имеют случайные веса в диапазоне от 1 до 10.
  + Граф сохраняется в файл Excel, где:
    - В первой строке указаны количество вершин и рёбер.
    - Далее перечислены рёбра в формате: начальная вершина, конечная вершина, вес.
    - В отдельной колонке приводится список всех вершин.
* **Чтение графа:**
  + Функция read\_graph\_from\_excel считывает граф из файла Excel.
  + На основе данных файла формируется матрица смежности.
  1. Алгоритмы работы с графом

**Алгоритм Флойда-Уоршалла**

* **Назначение:**
  + Алгоритм используется для поиска кратчайших путей между всеми парами вершин графа.
  + Работает как для графов с положительными, так и с отрицательными весами рёбер (но не поддерживает отрицательные циклы).
* **Реализация:**
  + Функция floyd\_warshall\_with\_path вычисляет матрицы:
    - dist — матрицу минимальных расстояний между всеми парами вершин.
    - next\_node — матрицу для восстановления пути между вершинами.
  + Алгоритм выполняется за O(V3), где V — количество вершин.
* **Ключевые шаги:**
  + Инициализация:
    - Матрица расстояний dist копируется из матрицы смежности.
    - Матрица next\_node заполняется индексами вершин.
  + Обновление расстояний:
    - Для каждой промежуточной вершины k:
      * Для каждой пары вершин i,j:
        + Если путь i→k→j короче текущего пути i→j, обновить: dist[i][j]=dist[i][k]+dist[k][j]  
          next\_node[i][j] = next\_node[i][k]

**Реконструкция пути**

* Функция reconstruct\_path восстанавливает последовательность вершин кратчайшего пути между двумя вершинами, используя матрицу next\_node.
* Алгоритм:
  + Начальная вершина добавляется в путь.
  + По матрице next\_node определяется следующая вершина на пути до конечной.
  + Процесс повторяется, пока путь не будет завершён.

**Запись результатов**

* Функция write\_result\_to\_excel сохраняет:
  + Матрицу кратчайших расстояний.
  + Описание всех кратчайших путей.
* Формат:
  + Лист "Distances" содержит таблицу расстояний.
  + Лист "Paths" содержит список всех путей в текстовом формате.

**Вывод графа в консоль**

* Функция print\_graph выводит:
  + Матрицу смежности изначального графа.
  + Матрицу кратчайших расстояний.
  1. Примеры использования
     1. Матрица изначальных путей

Вводные данные: Граф с вершинами A,B,C,D и рёбрами с весами:

* A→B:5
* A→C:10
* B→D:3
* C→D:1

Изначальная матрица смежности:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Из\до | A | B | C | D |
| A | 0 | 5 | 10 | ∞ |
| B | ∞ | 0 | ∞ | 3 |
| C | ∞ | ∞ | 0 | 1 |
| D | ∞ | ∞ | ∞ | 0 |

* + 1. Итерация 1

Используем вершину A как промежуточную. Для каждой пары вершин i,j проверяем:

dist[i][j] = min(dist[i][j], dist[i][A] + dist[A][j])

Обновлённая матрица

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Из/до | A | B | C | D |
| A | 0 | 5 | 10 | ∞ |
| B | ∞ | 0 | ∞ | 3 |
| C | ∞ | ∞ | 0 | 1 |
| D | ∞ | ∞ | ∞ | 0 |

Изменений нет, так как A не улучшает пути через себя.

* + 1. Итерация 2

Используем вершину B как промежуточную:

dist[i][j]=min(dist[i][j],dist[i][B]+dist[B][j])

Обновлённая матрица:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Из/до | A | B | C | D |
| A | 0 | 5 | 10 | 8 |
| B | ∞ | 0 | ∞ | 3 |
| C | ∞ | ∞ | 0 | 1 |
| D | ∞ | ∞ | ∞ | 0 |

Путь A→D обновился: A→B→D=5+3=8.

* + 1. Итерация 3

Используем вершину C как промежуточную:

dist[i][j]=min(dist[i][j],dist[i][C]+dist[C][j])

Обновлённая матрица:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Из/до | A | B | C | D |
| A | 0 | 5 | 10 | 8 |
| B | ∞ | 0 | ∞ | 3 |
| C | ∞ | ∞ | 0 | 1 |
| D | ∞ | ∞ | ∞ | 0 |

Изменений нет, так как C не улучшает пути через себя.

* + 1. Итерация 4

Используем вершину D как промежуточную:

dist[i][j]=min(dist[i][j],dist[i][D]+dist[D][j])

Обновлённая матрица:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Из/до | A | B | C | D |
| A | 0 | 5 | 10 | 8 |
| B | ∞ | 0 | ∞ | 3 |
| C | ∞ | ∞ | 0 | 1 |
| D | ∞ | ∞ | ∞ | 0 |

Изменений нет, так как D не улучшает пути через себя.

* + 1. Матрица кратчайших путей

Итоговая матрица кратчайших путей:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Из/до | A | B | C | D |
| A | 0 | 5 | 10 | 8 |
| B | ∞ | 0 | ∞ | 3 |
| C | ∞ | ∞ | 0 | 1 |
| D | ∞ | ∞ | ∞ | 0 |

Все кратчайшие пути между вершинами определены.

* + 1. Реконструкция путей

Пример кратчайшего пути A→D:

Матрица next\_node указывает на путь:

* A→B
* B→D

Итоговый путь: A→B→D.

* 1. Примеры использования на полном орграфе
     1. Матрица изначальных путей

**Входные данные:** Полный орграф с вершинами A,B,C,D и рёбрами со следующими весами:

A→B:2

A→C:5

A→D:9

B→A:1

B→C:3

B→D:4

C→A:7

C→B:6

C→D:8

D→A:3

D→B:2

D→C:4

**Изначальная матрица смежности:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Из/до | A | B | C | D |
| A | 0 | 2 | 5 | 9 |
| B | 1 | 0 | 3 | 4 |
| C | 7 | 6 | 0 | 8 |
| D | 3 | 2 | 4 | 0 |

* + 1. Итерация 1

**Промежуточная вершина:** A

Применяем вершину AAA как промежуточную для обновления матрицы расстояний:

dist[i][j] = min(dist[i][j], dist[i][A] + dist[A][j])

**Обновлённая матрица:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Из/до | A | B | C | D |
| A | 0 | 2 | 5 | 9 |
| B | 1 | 0 | 3 | 4 |
| C | 7 | 6 | 0 | 8 |
| D | 3 | 2 | 4 | 0 |

* + 1. Итерация 2

**Промежуточная вершина:** B

Применяем вершину B как промежуточную для обновления матрицы расстояний:

dist[i][j] = min(dist[i][j], dist[i][B] + dist[B][j])

**Обновления:**

A → C : min(5,2+3)=5→5 (нет изменения)

A → D : min(9,2+4)=6 (обновляется)

C → A : min(7,6+1)=6 (обновляется)

C → D : min(8,6+4)=8 (нет изменения)

D → C : min(4,2+3)=4→4 (нет изменения)

**Обновлённая матрица:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Из/до | A | B | C | D |
| A | 0 | 2 | 5 | 6 |
| B | 1 | 0 | 3 | 4 |
| C | 6 | 5 | 0 | 8 |
| D | 3 | 2 | 4 | 0 |

* + 1. Итерация 3

**Промежуточная вершина:** C

Применяем вершину C как промежуточную для обновления матрицы расстояний:

dist[i][j]=min(dist[i][j],dist[i][C]+dist[C][j])

**Обновления:**

A → D : min(6,5+8)=6 (нет изменения)

B → A : min(1,3+6)=1 (нет изменения)

B → D : min(4,3+8)=4 (нет изменения)

D → A : min(3,4+6)=3 (нет изменения)

D → B : min(2,4+5)=2 (нет изменения)

**Обновлённая матрица:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Из/до | A | B | C | D |
| A | 0 | 2 | 5 | 6 |
| B | 1 | 0 | 3 | 4 |
| C | 6 | 5 | 0 | 8 |
| D | 3 | 2 | 4 | 0 |

* + 1. Итерация 4

**Промежуточная вершина:** D

Применяем вершину D как промежуточную для обновления матрицы расстояний:

dist[i][j]=min(dist[i][j],dist[i][D]+dist[D][j])

**Обновления:**

A→C:min(5,6+4)=5 (нет изменения)

B→A:min(1,4+3)=1 (нет изменения)

B→C:min(3,4+4)=3 (нет изменения)

C→D:min(8,8+0)=8 (нет изменения)

Обновлённая матрица:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Из/до | A | B | C | D |
| A | 0 | 2 | 5 | 6 |
| B | 1 | 0 | 3 | 4 |
| C | 6 | 5 | 0 | 8 |
| D | 3 | 2 | 4 | 0 |

* + 1. Матрица кратчайших путей

Итоговая матрица кратчайших путей:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Из/до | A | B | C | D |
| A | 0 | 2 | 5 | 6 |
| B | 1 | 0 | 3 | 4 |
| C | 6 | 5 | 0 | 8 |
| D | 3 | 2 | 4 | 0 |

Описание кратчайших путей:

A→B: Прямой путь A→B, вес = 2.

A→C: Прямой путь A→C, вес = 5.

A→D: Кратчайший путь A→B→D, вес = 2 + 4 = 6.

B→A: Прямой путь B→A, вес = 1.

B→C: Прямой путь B→C, вес = 3.

B→D: Прямой путь B→D, вес = 4.

C→A: Кратчайший путь C→B→A, вес = 5 + 1 = 6.

C→B: Кратчайший путь C→B, вес = 5.

C→D: Прямой путь C→D, вес = 8.

D→A: Прямой путь D→A, вес = 3.

D→B: Прямой путь D→B, вес = 2.

D→C: Прямой путь D→C, вес = 4.

1. Интерфейс приложения
   1. Основная функция

Основная функция реализована в программе через функцию main():

Генерация графа: Пользователь может создать случайный граф с заданным количеством вершин и рёбер, который сохраняется в Excel-файл.

Анализ графа: Пользователь может загрузить граф из Excel-файла и выполнить алгоритм Флойда-Уоршалла для поиска кратчайших путей.

Результаты:

Выводятся в консоль матрицы расстояний и пути между всеми парами узлов.

Результаты записываются в Excel-файл.

* 1. Пример запуска

**Пример 1: Генерация случайного графа**

python main.py

**Описание:** Программа предложит выбрать один из двух режимов:

Генерация случайного графа.

Выполнение вычислений с использованием заранее созданного графа.

**Действия:**

Выберите режим 1.

Укажите количество узлов и рёбер.

**Результат:** Случайный граф будет сохранён в файл.

**Пример 2: Выполнение вычислений на графе**

python main.py

**Описание:** Если граф уже создан в файле, программа вычислит:

Матрицу кратчайших расстояний.

Пути между всеми парами узлов.

**Действия:**

Выберите режим 2.

**Результат:** Результаты сохраняются в файл и отображаются в консоли.

1. Заключение

Данный проект демонстрирует практическую реализацию алгоритма Флойда-Уоршелла для поиска кратчайших путей в графе. Решение включает генерацию случайных графов, их сохранение и обработку, а также наглядное представление результатов как в консоли, так и в формате Excel.

В процессе работы были достигнуты следующие цели:

**Автоматизация генерации графов.** Пользователь может создать случайный граф с заданным количеством узлов и рёбер. Это упрощает тестирование алгоритма на различных входных данных.

**Чтение и обработка графов из файла.** Программа позволяет загружать граф из файла, что удобно для работы с заранее подготовленными данными.

**Реализация алгоритма Флойда-Уоршелла.** Алгоритм эффективно вычисляет кратчайшие пути между всеми парами узлов, что находит широкое применение в теории графов и реальных задачах.

**Удобный вывод результатов.** Результаты сохраняются в Excel, предоставляя пользователю возможность детально изучить матрицу расстояний и маршруты между узлами.

Проект предоставляет базовый, но гибкий инструмент для работы с графами, который можно расширять и модифицировать в зависимости от требований. Например, возможно добавить поддержку других форматов файлов, оптимизировать алгоритмы для работы с большими графами или добавить визуализацию графов.

Благодаря своей универсальности, решение может быть использовано как в учебных целях для изучения алгоритмов на графах, так и для решения прикладных задач в инженерии, логистике, анализе данных и других областях.

1. Приложение

import openpyxl  
import os  
import random  
  
def generate\_random\_graphs(num\_nodes, num\_edges, filename = "input\_graph.xlsx"):  
 if os.path.exists(filename):  
 os.remove(filename)  
  
 workbook = openpyxl.Workbook()  
 sheet = workbook.active  
 # nodes = [str(i) for i in range(num\_nodes)]  
 nodes = [chr(ord("A")+i) for i in range(num\_nodes)]  
 edges = {}  
  
 while len(edges) < num\_edges:  
 from\_node = random.choice(nodes)  
 to\_node = random.choice(nodes)  
 if from\_node != to\_node:   
 weight = random.randint(1, 10)  
 edges[from\_node, to\_node] = weight  
   
 sheet.cell(row=1, column=1, value=num\_nodes)   
 sheet.cell(row=1, column=2, value=num\_edges)   
  
 for row, edge in enumerate(edges.items()):   
 (from\_node, to\_node), weight = edge  
 sheet.cell(row=row+2, column=1, value=from\_node)   
 sheet.cell(row=row+2, column=2, value=to\_node)   
 sheet.cell(row=row+2, column=3, value=weight)   
  
 for row, node in enumerate(nodes):  
 sheet.cell(row=row + 2, column=5, value=node)  
  
 workbook.save(filename)  
  
def read\_graph\_from\_excel(file\_path):  
 workbook = openpyxl.load\_workbook(file\_path)  
 sheet = workbook.active  
  
 edges = {}  
 num\_nodes = sheet.cell(row=1, column=1).value  
 num\_edges = sheet.cell(row=1, column=2).value  
  
 for row in range(2, num\_edges + 1):  
 from\_node = sheet.cell(row=row, column=1).value  
 to\_node = sheet.cell(row=row, column=2).value  
 weight = sheet.cell(row=row, column=3).value  
   
 edges[(from\_node, to\_node)] = weight  
   
 nodes = {}  
 for row in range(num\_nodes):   
 node = sheet.cell(row=row+2, column=5).value   
 nodes[node] = row  
  
 workbook.close()  
 graph = [[float('inf')] \* num\_nodes for \_ in range(num\_nodes)]  
  
 for i in range(num\_nodes):  
 graph[i][i] = 0  
 for (from\_node, to\_node), weight in edges.items():  
 graph[nodes[from\_node]][nodes[to\_node]] = weight  
  
 return graph, list(nodes.keys())  
  
def floyd\_warshall\_with\_path(graph):  
 num\_vertices = len(graph)  
 dist = [row[:] for row in graph]  
 next\_node = [[None if i != j and graph[i][j] == float('inf') else j for j in range(num\_vertices)] for i in range(num\_vertices)]  
  
 for k in range(num\_vertices):  
 for i in range(num\_vertices):  
 for j in range(num\_vertices):  
 if dist[i][j] > dist[i][k] + dist[k][j]:  
 dist[i][j] = dist[i][k] + dist[k][j]  
 next\_node[i][j] = next\_node[i][k]  
  
 return dist, next\_node  
  
def reconstruct\_path(next\_node, start, end):  
 if next\_node[start][end] is None:  
 return []   
 path = []  
 while start != end:  
 path.append(start)  
 start = next\_node[start][end]  
 path.append(end)  
 return path  
  
def write\_result\_to\_excel(dist, nodes, next\_node, filename='graph.xlsx'):  
 wb = openpyxl.Workbook()  
   
 ws\_dist = wb.active  
 ws\_dist.title = "Distances"  
  
 ws\_dist.cell(row=1, column=1, value='из\\до')  
 for col, node in enumerate(nodes, start=2):  
 ws\_dist.cell(row=1, column=col, value=node)  
  
 for i in range(len(dist)):  
 ws\_dist.cell(row=i + 2, column=1, value=nodes[i])  
 for j in range(len(dist[i])):  
 ws\_dist.cell(row=i + 2, column=j + 2, value=dist[i][j])   
  
 ws\_paths = wb.create\_sheet(title="Paths")  
  
 ws\_paths.cell(row=1, column=1, value='Из')  
 ws\_paths.cell(row=1, column=2, value='До')  
 ws\_paths.cell(row=1, column=3, value='Путь')  
  
 row = 2  
 for i in range(len(nodes)):  
 for j in range(len(nodes)):  
 if i != j:  
 path = reconstruct\_path(next\_node, i, j)  
 ws\_paths.cell(row=row, column=1, value=nodes[i])   
 ws\_paths.cell(row=row, column=2, value=nodes[j])   
 ws\_paths.cell(row=row, column=3, value=' -> '.join(nodes[k] for k in path))   
 row += 1  
  
 wb.save(filename)  
  
def print\_graph(dist, nodes):  
 n = len(dist)  
 for i in range(n):  
 if i == 0:  
 print("из\\до", end="\t")   
 for node in nodes:  
 print(node, end="\t")   
 print()  
 for j in range(n):  
 if j == 0:  
 print(nodes[i], end="\t")   
 print(dist[i][j], end="\t")   
 print()  
 print()  
  
def main():  
 print("Выберите режим работы \n1. Сгенерировать рандомный граф с указанным количеством графов и путей \n2. Провести вычисления созданного графа !!!Внимание ответ будет записан в excel файл \n ")  
 choose = int(input())  
  
 if choose == 1:  
 print("Введите количество узлов")  
 num\_nodes = max(int(input()), 1)  
 print("Введите количество ветвей")  
 num\_edges = max(min(int(input()), num\_nodes\*\*2-num\_nodes), 0)  
 generate\_random\_graphs(num\_nodes, num\_edges)  
 else:  
 input\_file = "input\_graph.xlsx"  
 output\_file = "output\_distances\_and\_paths.xlsx"  
 graph, nodes = read\_graph\_from\_excel(input\_file)  
 print("изначальный граф:")  
 print\_graph(graph, nodes)  
 dist, next\_node = floyd\_warshall\_with\_path(graph)  
 print("Граф минимальных расстояний:")  
 print\_graph(dist, nodes)  
 for i in range(len(nodes)):  
 for j in range(len(nodes)):  
 if i != j:  
 path = reconstruct\_path(next\_node, i, j)  
 print(f"Путь из {nodes[i]} в {nodes[j]}:", path)  
 write\_result\_to\_excel(dist, nodes, next\_node, output\_file)  
 print("Результаты сохранены в", output\_file)  
  
# Вызов функции main  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()