ГУАП

КАФЕДРА № 41

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Старший преподаватель |  |  |  | Н.А. Соловьева |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ |
| АЛГОРИТМЫ НА ГРАФАХ |
| по курсу: АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ гр. № | 4314 |  |  |  | Д. М. Развеев |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2024

1. **Цель работы:**

познакомиться с основными алгоритмами, используемыми при работе с графами.

1. **Вариант 4**:

*Задание 4*: объявите структуру данных «Граф» на основе матрицы смежности с возможностью вывода ее текущего представления в терминал, после чего используйте его для реализации алгоритма Дейкстры и Форда-Беллмана. Добавьте возможность сохранения текущего представления графа в файл и загрузки из него.

1. **Ход работы**

*Задание 4*

Программа реализации:

import math

import heapq

from typing import List, Tuple

class Graph:

    def \_\_init\_\_(self, num\_vertices: int):

        """

        Инициализация графа с заданным числом вершин.

        Матрица смежности заполняется бесконечностями, кроме диагонали (0).

        """

        self.num\_vertices = num\_vertices

        self.graph = [[math.inf] \* num\_vertices for \_ in range(num\_vertices)]

        for i in range(num\_vertices):

            self.graph[i][i] = 0

    def add\_edge(self, u: int, v: int, weight: float) -> None:

        """Добавляет ребро от вершины u к v с весом weight."""

        self.graph[u][v] = weight

        #добавить, если ориентированный

        #self.graph[v][u] = weight

    def print\_graph(self) -> None:

        """Выводит матрицу смежности графа в терминал."""

        print("Матрица смежности графа:")

        for row in self.graph:

            print(" ".join(str(i) if i != math.inf else "∞" for i in row))

    def save\_to\_file(self, filename: str) -> None:

        """Сохраняет граф в файл."""

        with open(filename, "w") as f:

            f.write(f"{self.num\_vertices}\n")

            for row in self.graph:

                f.write(" ".join(str(i) if i != math.inf else "∞" for i in row) + "\n")

    def load\_from\_file(self, filename: str) -> 'Graph':

        """Загружает граф из файла."""

        with open(filename, "r") as f:

            self.num\_vertices = int(f.readline().strip())

            self.graph = []

            for line in f:

                self.graph.append([math.inf if x == "∞" else float(x) for x in line.split()])

        return self

    def dijkstra(self, start: int) -> List[float]:

        """Алгоритм Дейкстры для поиска кратчайших путей от вершины start."""

        dist: List[float] = [math.inf] \* self.num\_vertices

        dist[start] = 0

        pq: List[tuple]  = [(0, start)]  # (расстояние, вершина)

        while pq:

            current\_dist, u = heapq.heappop(pq)

            if current\_dist > dist[u]:

                continue

            for v in range(self.num\_vertices):

                if self.graph[u][v] != math.inf:

                    weight = self.graph[u][v]

                    if dist[u] + weight < dist[v]:

                        dist[v] = dist[u] + weight

                        heapq.heappush(pq, (dist[v], v))

        return dist

    def bellman\_ford(self, start: int) -> Tuple[bool, List[float]]:

        """

        Алгоритм Форда-Беллмана для поиска кратчайших путей от вершины start.

        Возвращает (bool, List[float]), где bool — наличие отрицательного цикла.

        """

        dist = [math.inf] \* self.num\_vertices

        dist[start] = 0

        for \_ in range(self.num\_vertices - 1):

            for u in range(self.num\_vertices):

                for v in range(self.num\_vertices):

                    if self.graph[u][v] != math.inf and dist[u] + self.graph[u][v] < dist[v]:

                        dist[v] = dist[u] + self.graph[u][v]

        # Проверка на отрицательные циклы

        for u in range(self.num\_vertices):

            for v in range(self.num\_vertices):

                if self.graph[u][v] != math.inf and dist[u] + self.graph[u][v] < dist[v]:

                    return False, []  # Обнаружен отрицательный цикл

        return True, dist

**Описание кода**

Этот код реализует граф, представленный матрицей смежности, и включает два алгоритма поиска кратчайших путей: Дейкстры и Форда-Беллмана. Также предусмотрены функции для сохранения и загрузки графа из файла.

**Класс**Graph**:**

* \_\_init\_\_(self, num\_vertices) — Инициализирует граф с заданным числом вершин. Матрица смежности заполняется бесконечностями, за исключением диагонали.
* add\_edge(self, u, v, weight) — Добавляет ребро между вершинами u и v с весом weight.
* print\_graph(self) — Выводит матрицу смежности в консоль.
* save\_to\_file(self, filename) — Сохраняет граф в файл.
* load\_from\_file(self, filename) — Загружает граф из файла.
* dijkstra(self, start) — Алгоритм Дейкстры для нахождения кратчайших путей от вершины start.
* bellman\_ford(self, start) — Алгоритм Форда-Беллмана для нахождения кратчайших путей от вершины start с проверкой на отрицательные циклы.

А теперь проведем тесты и замер производительности:

def run\_tests() -> None:

    print("Тесты")

    # Тест 1: Создание графа и добавление рёбер

    g = Graph(5)

    g.add\_edge(0, 1, 6)

    g.add\_edge(0, 2, 7)

    g.add\_edge(1, 2, 8)

    g.add\_edge(1, 3, 5)

    g.add\_edge(1, 4, -4)

    g.add\_edge(2, 3, -3)

    g.add\_edge(2, 4, 9)

    g.add\_edge(3, 1, 5) #убрать, если ориентрованный

    g.add\_edge(4, 0, 2) #убрать, если ориентрованный

    g.add\_edge(4, 3, 7) #убрать, если ориентрованный

    # Тест 2: Вывод графа

    print("\nГраф:")

    g.print\_graph()

    # Тест 3: Алгоритм Дейкстры

    distances = g.dijkstra(0)

    print("\nМинимальные расстояния (Дейкстра) от вершины 0:", distances)

    # Тест 4: Алгоритм Форда-Беллмана

    has\_no\_negative\_cycle, distances = g.bellman\_ford(0)

    if has\_no\_negative\_cycle:

        print("\nМинимальные расстояния (Форд-Беллман) от вершины 0:", distances)

    else:

        print("\nОбнаружен отрицательный цикл!")

    # Тест 5: Сохранение графа

    g.save\_to\_file("graph.txt")

    # Тест 6: Загрузка графа

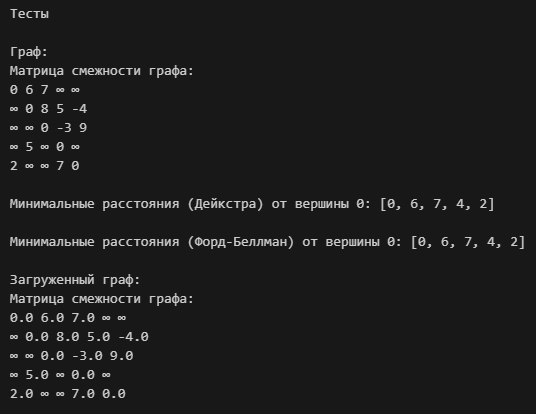
    new\_graph = Graph(0).load\_from\_file("graph.txt")

    print("\nЗагруженный граф:")

    new\_graph.print\_graph()

if \_\_name\_\_ == "\_\_main\_\_":

    run\_tests()



Видно, что класс верно выполнил все свои функции, значит код работает корректно.

# Вывод

В ходе лабораторной работы была реализована структура данных Граф на основе матрицы смежности, также реализованы алгоритмы Дейкстры и Форда-Беллмана. Добавлена возможность сохранения текущего представления графа в файл и загрузки из него, возможность выведения текущего представления матрицы в терминал.