МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

"МИРЭА - Российский технологический университет"

РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра математического обеспечения и стандартизации информационных технологий (МОСИТ)

ОТЧЁТ ПО ПРАКТИЧЕСКОМУ ЗАДАНИЮ №5

по дисциплине «Структуры и алгоритмы обработки данных»

Тема: «Основные алгоритмы работы с графами.»

Отчет представлен к рассмотрению:
Студент группы ИНБО-01-20 «1» ноября 2021 г. _____ Салов В.Д.
(подпись)

Преподаватель «1» ноября 2021 г. _____ Сорокин А.В.
(подпись)

Москва, 2021 г.

СОДЕРЖАНИЕ

Цель работы

Получение практических навыков по выполнению операций над структурой данных «граф».

Постановка задачи

Выполнить разработку программы управления графом, в соответствии с вариантом, на основе класса «Граф». Предусмотреть в качестве данных: количество вершин в графе, структура для хранения графа.

Вариант 9.

Представление графа в памяти	Задачи варианта
Матрица смежности	Ввод с клавиатуры графа (применение операции вставки ребра в граф). Определить медиану неориентированного графа. Составить программу нахождения кратчайшего пути в графе от заданной вершины к другой заданной вершине методом «Дейкстры» и вывести этот путь.

Дано:

Произвольный граф (ориентированный или неориентированный, связный или несвязный) с известным количеством вершин.

Результат.

Отображение графа в виде матрицы смежности.

Реализованные операции варианта.

Подход к решению.

1) В ходе работы был разработан класс графа, реализующий, согласно варианту, следующие методы: создание графа посредством применения операций вставки ребра в граф, вывод матрицы смежности графа, определение медианы графа, нахождение величины кратчайшего пути и вывод кратчайшего пути от заданной вершины к другой заданной вершине методом «Дейкстры».
2) Разработан класс узла графа, содержащий информационную часть и поле индекса.
3) Разработан консольный пользовательский интерфейс для тестирования работоспособности программы.

4) Разработаны методы обработки графа:

1. Метод вставки ребра в граф – добавление ребра с заданным весом между двумя заданными вершинами;
2. Метод получения индекса узла – возврат индекса для заданного узла графа;
3. Метод определения медианы графа – возврат индекса вершины, для которой сумма кратчайших путей до других вершин минимальна.
4. Метод «Дейкстры» – поиск величин кратчайших путей до каждой вершины для заданной вершины;
5. Метод восстановления кратчайшего пути – возврат кратчайшего маршрута из одной заданной вершины в другую;
6. Метод нахождения величины кратчайшего пути – возврат величины кратчайшего пути между двумя заданными вершинами графа, найденного с помощью метода алгоритма Дейкстры.

5) Разработаны методы приложения для тестирования:

1. Метод для тестирования вывода графа – организация ввода графа и вывода его матрицы смежности;
2. Метод для тестирования нахождения кратчайшего пути – организация нахождения кратчайшего пути от одной заданной вершины к другой с помощью алгоритма Дейкстры и вывода результатов работы алгоритмов в консоль;
3. Метод для тестирования определения медианы графа – организация определения медианы графа и вывода результата работы алгоритма в консоль.

Алгоритмы операций на псевдокоде.

Метод вставки ребра в граф:

```
процедура connect(первая_вершина, вторая_вершина, вес := 1):
    первая_вершина := self.get_index(первая_вершина)
    вторая_вершина := self.get_index(вторая_вершина)
    матрица_смежности[первая_вершина][вторая_вершина] := вес
```

Метод получения индекса узла:

```
функция get_index(узел):
    если принадлежность(узел, int):
        возврат узел
    иначе:
        возврат узел.индекс
```

Метод «Дейкстры»:

```
функция dijkstra(узел):
    узел := get_index(узел)
    из collections импортировать default_словарь
    граф := default_словарь(список)
    для каждого row от 0 до длина(матрица_смежности) - 1:
        для каждого col от 0 до длина(матрица_смежности) - 1:
            если матрица_смежности[row][col] есть не None и
матрица_смежности[row][col] != 0:
                граф[row] := граф[row] + [(матрица_смежности[row][col], col)]
    nodes_to_visit := массив[]
    nodes_to_visit.добавить_элемент_в_конец((0, узел))
    visited := множество()
    min_dist := {i: ∞ для каждого i от 0 до длина(матрица_смежности) - 1}
    min_dist[узел] := 0
    пока длина(nodes_to_visit) > 0:
        вес, текущий_узел := минимальный_элемент(nodes_to_visit)
        nodes_to_visit.удалить_элемент((вес, текущий_узел))
        если текущий узел в visited:
            принудительный_запуск_следующего_прохода_цикла
        visited.добавить_элемент(текущий_узел)
        для след_вес, след_узел в graph[текущий_узел]:
            если вес + след_вес < min_dist[след_узел] и след_узел не в visited:
                min_dist[след_узел] := вес + след_вес
                nodes_to_visit.добавить_элемент_в_конец((вес + след_вес,
след_узел))
    возврат min_dist
```

Метод восстановления кратчайшего пути:

```
функция path_restoring(узел1, узел2):
    visited := [None] * длина(матрица_смежности)
    узел1 := self.get_index(узел1)
    узел2 := self.get_index(узел2)
    visited[0] := узел2 + 1
    пред_индекс := 1
    вес := shortest_path(узел1, узел2)
    пока узел2 != узел1:
        для каждого i от 0 до длина(матрица_смежности) - 1:
            если матрица_смежности[i][узел2] есть не None и
матрица_смежности[i][узел2] != 0:
                temp := вес - матрица_смежности[i][узел2]
                если temp == shortest_path(узел1, i):
                    вес := temp
                    узел2 := i
                    visited[пред_индекс] := i + 1
                    пред_индекс := пред_индекс + 1
    пока None в visited:
        visited.удалить_элемент(None)
    возврат развернуть_массив(visited)
```

Метод нахождения величины кратчайшего пути:

```
функция shortest_path(узел1, узел2):
    узел2 := get_index(узел2)
    возврат dijkstra(узел1)[узел2]
```

Метод определения медианы графа:

```
функция median():
    array := массив[]
    для каждого узел1 от 0 до длина(матрица_смежности) - 1:
        сумма_путей := 0
        для каждого узел2 от 0 до длина(матрица_смежности) - 1:
            сумма_путей := сумма_путей + shortest_path(узел1, узел2)
        array.добавить_элемент_в_конец(сумма_путей)
    возврат array.индекс(минимальный_элемент(array))
```

Код программы.

Класс узла графа:

```
class Node:
    def __init__(self, data, indexloc=None):
        self.data = data
        self.index = indexloc
```

Рисунок 1 – Класс узла графа.

Класс графа:

```
class Graph:

    @classmethod
    def create_from_nodes(self, nodes):
        return Graph(len(nodes), len(nodes), nodes)

    def __init__(self, row, col, nodes=None):
        # Матрица смежности.
        self.adj_matrix = [[None] * col for _ in range(row)]
        for i in range(len(self.adj_matrix)):
            for j in range(len(self.adj_matrix)):
                if i == j:
                    self.adj_matrix[i][j] = 0
        self.nodes = nodes
        for i in range(len(self.nodes)):
            self.nodes[i].index = i

    # Операция вставки ребра в граф.
    def connect(self, node1, node2, weight):
        node1, node2 = self.get_index(node1), self.get_index(node2)
        self.adj_matrix[node1][node2] = weight

    # Получение индекса узла.
    @staticmethod
    def get_index(node):
        if isinstance(node, int):
            return node
        else:
            return node.index

    # Алгоритм Дейкстры.
    def dijkstra(self, node):
        node = self.get_index(node)
        from collections import defaultdict
        graph = defaultdict(list)  # Инициализация графа словарём.
        # Заполнение графа по матрице смежности.
        for row in range(len(self.adj_matrix)):
            for col in range(len(self.adj_matrix)):
                if self.adj_matrix[row][col] is not None and self.adj_matrix[row][col] != 0:
                    graph[row] += [(self.adj_matrix[row][col], col)]
        nodes_to_visit = []  # Инициализация списка вершин для посещения.
        nodes_to_visit.append((0, node))  # Добавление стартовой вершины в список как первой вершины для посещения.
        visited = set()  # Множество для хранения посещённых вершин.
        min_dist = {i: float('inf') for i in range(len(self.adj_matrix))}  # Заполнение расстояний до вершин.
        min_dist[node] = 0  # Заполнение расстояния до стартовой вершины.
        while len(nodes_to_visit):  # Пока nodes_to_visit не пустой:
            weight, current_node = min(nodes_to_visit)  # Выбор ближней вершины.
            nodes_to_visit.remove((weight, current_node))  # Удаление этой вершины из списка вершин для посещения.
            if current_node in visited:  # Если выбранная вершина уже посещена:
                continue  # Запустить следующий проход цикла, не выполняя оставшееся тело цикла.
            visited.add(current_node)  # Добавление выбранной вершины в список посещённых.
            # next_weight - вес связи из текущей вершины, next_node - прикреплённая вершина, в которую необходимо попасть.
            for next_weight, next_node in graph[current_node]:  # Проход по всем соединённым вершинам.
                # Проверка на оптимальность пути.
                if weight + next_weight < min_dist[next_node] and next_node not in visited:
                    min_dist[next_node] = weight + next_weight  # Обновление расстояния.
                    nodes_to_visit.append((weight + next_weight, next_node))  # Добавление вершины в список вершин для посещения
        return min_dist  # Возврат множества из словарей {номер_узла: кратчайший путь до него от заданного узла}.
```

Рисунок 2 – Класс графа.

```
    # Восстановление кратчайшего пути между node1 и node2.
    def path_restoring(self, node1, node2):
        visited = [None] * len(self.adj_matrix)  # Массив посещённых вершин.
        node1 = self.get_index(node1)
        node2 = self.get_index(node2)
        visited[0] = node2 + 1  # Начальный элемент - конечная вершина.
        pre = 1  # Индекс предыдущей вершины.
        weight = self.shortest_path(node1, node2)  # Вес конечной вершины.
        while node2 != node1:  # Пока не дошли до начальной вершины:
            for i in range(len(self.adj_matrix)):  # Проход по всем вершинам.
                if self.adj_matrix[i][node2] is not None and self.adj_matrix[i][node2] != 0:  # При наличии связи:
                    temp = weight - self.adj_matrix[i][node2]  # Определение веса пути из предыдущей вершины.
                    if temp == self.shortest_path(node1, i):  # Если вес совпал с рассчитанным, то из этой вершины был переход.
                        weight = temp
                        node2 = i
                        visited[pre] = i + 1
                        pre += 1
        while None in visited:
            visited.remove(None)
        return visited[::-1]

    # Нахождение величины кратчайшего пути между двумя заданными вершинами.
    def shortest_path(self, node1, node2):
        node2 = self.get index(node2)
        return self.dijkstra(node1)[node2]

    # Нахождение медианы.
    def median(self):
        array = []  # Массив из сумм кратчайших путей для каждой вершины.
        for node1 in range(len(self.adj_matrix)):
            paths_sum = 0
            for node2 in range(len(self.adj_matrix)):
                paths_sum += self.shortest_path(node1, node2)
            array.append(paths_sum)
        return array.index(min(array))

    # Приложение для ввода графа через консоль и вывода его матрицы смежности.
    def app_adj_matrix(self, directed=True):
        print("Операции вставки в граф взвешенного ребра:")
        while True:
            connection = list(map(int, input().split()))
            if not connection:
                break
            if directed is True:
                self.connect(connection[0] - 1, connection[1] - 1, connection[2])
            else:
                self.connect(connection[0] - 1, connection[1] - 1, connection[2])
                self.connect(connection[1] - 1, connection[0] - 1, connection[2])
        print("Матрица смежности графа:")
        for row in self.adj_matrix:
            print(row)

    # Приложение для нахождения кратчайшего пути от заданной вершины к другой заданной вершине и его величины.
    @staticmethod
    def app_shortest_path():
        print("Алгоритм Дейкстры поиска кратчайшего пути.")
        start_node = int(input("Номер начальной вершины: ")) - 1
        end_node = int(input("Номер конечной вершины: ")) - 1
        print("Величина кратчайшего пути:", w_graph.shortest_path(start_node, end_node))
        if w_graph.shortest_path(start_node, end_node) != float('inf'):
            print("Кратчайший путь:", w_graph.path_restoring(start_node, end_node))
        else:
            print("Путь не существует.")

    # Приложение для нахождения медианы графа.
    @staticmethod
    def app_median():
        print("Осуществляется поиск медианы графа.")
        print("Медиана найдена: это вершина под номером", w_graph.median() + 1, '.')
```

Рисунок 3 – Класс графа (продолжение).

Основная функция для тестирования:

```
# Главная функция.
if __name__ == '__main__':

    # Создание графа.
    node_list = []  # Список узлов.
    quantity = int(input("Количество вершин графа: "))
    for node in range(quantity):
        node_list.append(Node(str(node)))
    w_graph = Graph.create_from_nodes(node_list)

    # Вставка рёбер и вывод матрицы смежности.
    directed = bool(int(input("Для ориентированного графа введите '1', для неориентированного – '0': ")))
    w_graph.app_adj_matrix(directed)

    print()
    # Нахождение кратчайшего пути и его величины методом "Дейкстры".
    w_graph.app_shortest_path()

    # Нахождение медианы неориентированного графа.
    if directed is False:
        print()
        w_graph.app_median()

    print()
    print("Работа программы завершена.")
```

Рисунок 4 – Основная функция.

Тестирование программы.

Рассмотрим в качестве примера следующий ориентированный граф:

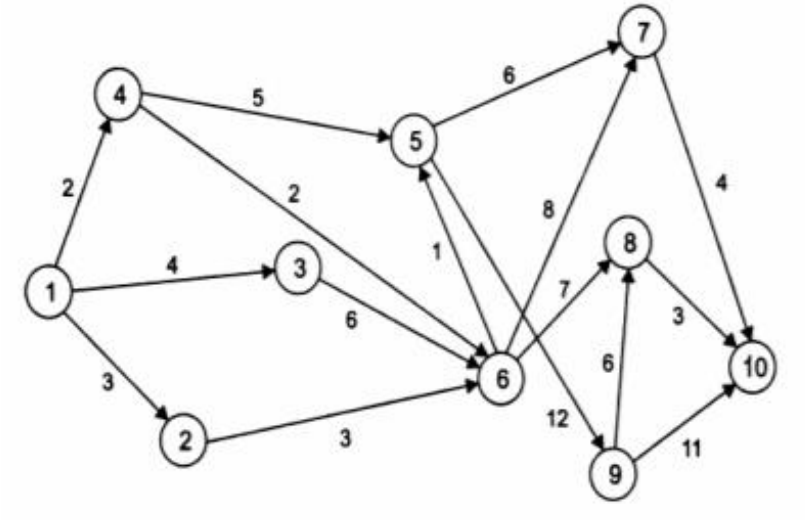

Рисунок 5 – Ориентированный граф для тестирования программы.

Было запущено тестирование: через консоль введено количество вершин графа, тип графа; были произведены операции вставки ребра в граф.

```
Количество вершин графа: 10
Для ориентированного графа введите '1', для неориентированного – '0': 1
Операции вставки в граф взвешенного ребра:
1 4 2
1 2 3
1 3 4
2 6 3
3 6 6
4 6 2
4 5 5
5 7 6
5 9 12
6 5 1
6 8 7
6 7 8
7 10 4
8 10 3
9 8 6
9 10 11
```

Рисунок 6 – Ввод графа посредством операций вставки ребра в граф.

В результате был выведен граф в виде матрицы смежности, в которой каждая строка является набором длин исходящих путей в каждую вершину графа для соответствующей вершины графа. Путь от любой вершины до самой себя равен 0. Если с какой-либо вершиной отсутствует прямая связь, то в ячейке хранится None.

Результат работы программы:

```
Матрица смежности графа:
[0, 3, 4, 2, None, None, None, None, None, None]
[None, 0, None, None, None, 3, None, None, None, None]
[None, None, 0, None, None, 6, None, None, None, None]
[None, None, None, 0, 5, 2, None, None, None, None]
[None, None, None, None, 0, None, 6, None, 12, None]
[None, None, None, None, 1, 0, 8, 7, None, None]
[None, None, None, None, None, None, 0, None, None, 4]
[None, None, None, None, None, None, None, 0, None, 3]
[None, None, None, None, None, None, None, 6, 0, 11]
[None, None, None, None, None, None, None, None, None, 0]
```

Рисунок 7 – Вывод матрицы смежности графа.

Далее вводятся две вершины графа для нахождения величины кратчайшего пути методом «Дейкстры» и вывода кратчайшего пути от первой заданной вершины ко второй.

Для тестирования были взяты вершины «1» и «10». По рисунку видно, что кратчайшим путём от вершины «1» к вершине «10» является путь: 1 – 4 – 6 – 8 – 10. Величина этого пути: 14.

Результат работы программы подтверждает полученный результат:

```
Алгоритм Дейкстры поиска кратчайшего пути.
Номер начальной вершины: 1
Номер конечной вершины: 10
Величина кратчайшего пути: 14
Кратчайший путь: [1, 4, 6, 8, 10]

Работа программы завершена.
```

Рисунок 8 – Вывод кратчайшего пути и его величины методом «Дейкстры».

Теперь пусть дан неориентированный граф следующего вида:

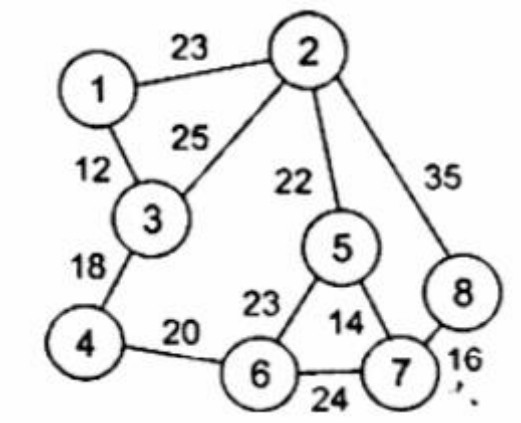

Рисунок 9 – Неориентированный граф для тестирования программы.

Было запущено тестирование: через консоль введено количество вершин графа, указан тип графа, произведены операции вставки ребра в граф.

```
Количество вершин графа: 8
Для ориентированного графа введите '1', для неориентированного – '0': 0
Операции вставки в граф взвешенного ребра:
1 2 23
2 3 25
1 3 12
3 4 18
4 6 20
6 5 23
2 5 22
5 7 14
6 7 24
7 8 16
2 8 35
```

Рисунок 10 – Ввод графа посредством операций вставки ребра в граф.

В результате была выведена матрица смежности для данного графа.

Результат работы программы:

```
Матрица смежности графа:
[0, 23, 12, None, None, None, None, None]
[23, 0, 25, None, 22, None, None, 35]
[12, 25, 0, 18, None, None, None, None]
[None, None, 18, 0, None, 20, None, None]
[None, 22, None, None, 0, 23, 14, None]
[None, None, None, 20, 23, 0, 24, None]
[None, None, None, None, 14, 24, 0, 16]
[None, 35, None, None, None, None, 16, 0]
```

Рисунок 11 – Вывод матрицы смежности графа.

Найдём кратчайший путь из вершины «5» к вершине «1» и его величину методом «Дейкстры».

По графу видно, что кратчайшим путём от вершины «5» к вершине «1» является путь: 5 – 2 – 1. Его величина: 45.

Результат работы программы это подтверждает:

```
Алгоритм Дейкстры поиска кратчайшего пути.
Номер начальной вершины: 5
Номер конечной вершины: 1
Величина кратчайшего пути: 45
Кратчайший путь: [5, 2, 1]
```

Рисунок 12 – Вывод кратчайшего пути и его величины методом «Дейкстры».

Так как граф неориентированный, найдём для него медиану:

```
Осуществляется поиск медианы графа.
Медиана найдена: это вершина под номером 5 .

Работа программы завершена.
```

Рисунок 13 – Определение медианы графа.

Тестирование пройдено успешно: все задачи варианта выполнены; программа работает правильно.

Вывод

В ходе работы были приобретены умения и навыки разработки и реализации операций над структурой данных «граф».

Список информационных источников

1. Лекции по дисциплине «Структуры и алгоритмы обработки данных» / Л. А. Скворцова, МИРЭА – Российский технологический университет, 2021.