Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ

УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра КСУП

Отчет по лабораторной работе

по дисциплине «Дискретная математика»

Тема: «Алгоритм Дейкстры»

Студент гр. 582-1

Полушвайко Константин Николаевич

__ декабря 2023 г.

2023

1. Задание

1. Реализовать алгоритм Дейкстры: поиск кратчайшего пути от 1 вершины до другой.

2. Ход работы

Алгоритм Дейкстры находит кратчайший путь от одной вершины до другой. Его используют на ориентированные графы, на каждой дуге которого есть свой вес. Этот алгоритм пользуется популярностью, так как он прост в реализации и универсален для пласта задач.

Для реализации алгоритма переделаем немного наш код из 3 лабораторной работы:

1. Матрица смежности может быть несимметричная у ориентированного графа;
2. Нужно создать матрицу весов, размерность которой совпадает с матрицей смежности;
3. Нужно визуализировать вес на дугах через networkx;
4. Реализовать алгоритм Дейкстры.

При создании матрицы весов стоит учитывать, что у нагруженного графа на каждой дуге есть свой вес, поэтому если вершины не соединены дугой, то путь до нее будет равен бесконечности. Реализация составления матрицы весов представлена в методе MakeWeightMatrix().

Обычно при нахождения ставиться задача нахождения кратчайшего пути из вершины A в вершину B, но использую алгоритм Дейкстры мы можем найти все кратчайшие пути из вершины A в любую другую вершину. Я использовал это, поэтому в моей реализации выводятся сразу n матриц расчета по алгоритму Дейкстры, где n – количество вершин. Реализация представлена в методе DijkstraAlgorithm().

Конечный код программы приведен в листинге (пункт 3).

На рисунках 2.1 – 2.3 представлен пример работы конечной программы.

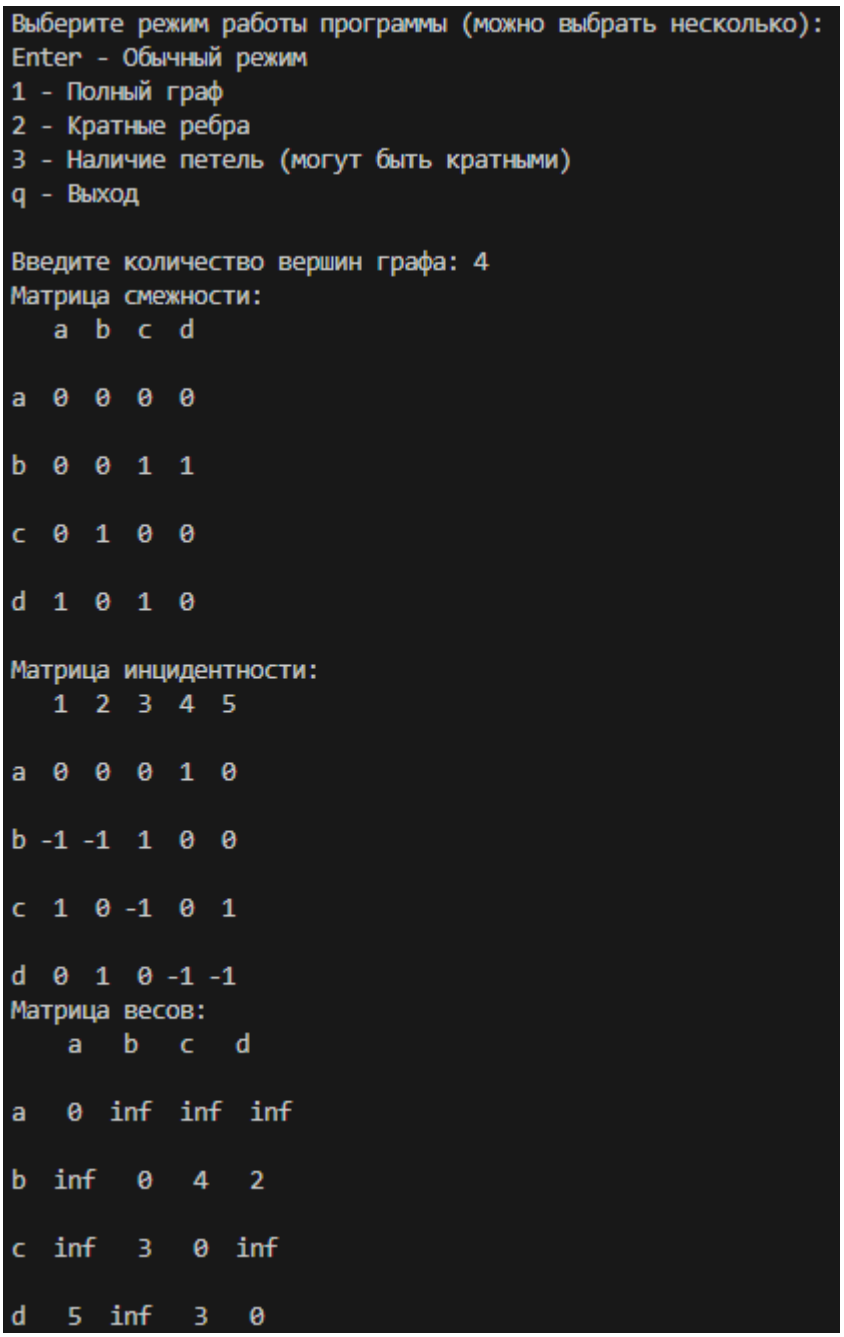

Рисунок 2.1 – Меню и матриц

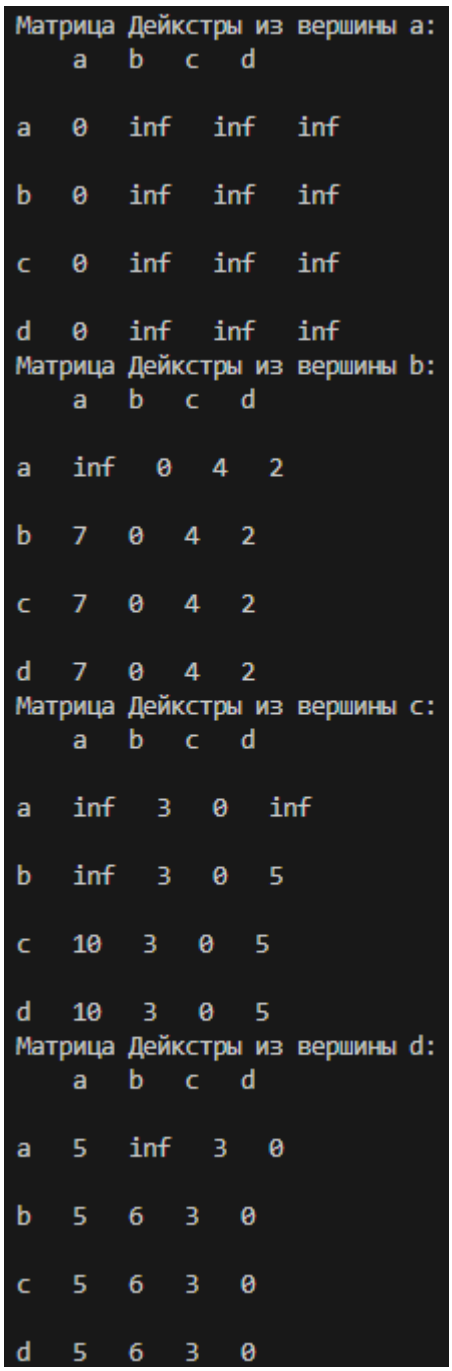

Рисунок 2.2 – Вывод всех расчетов и минимальных путей

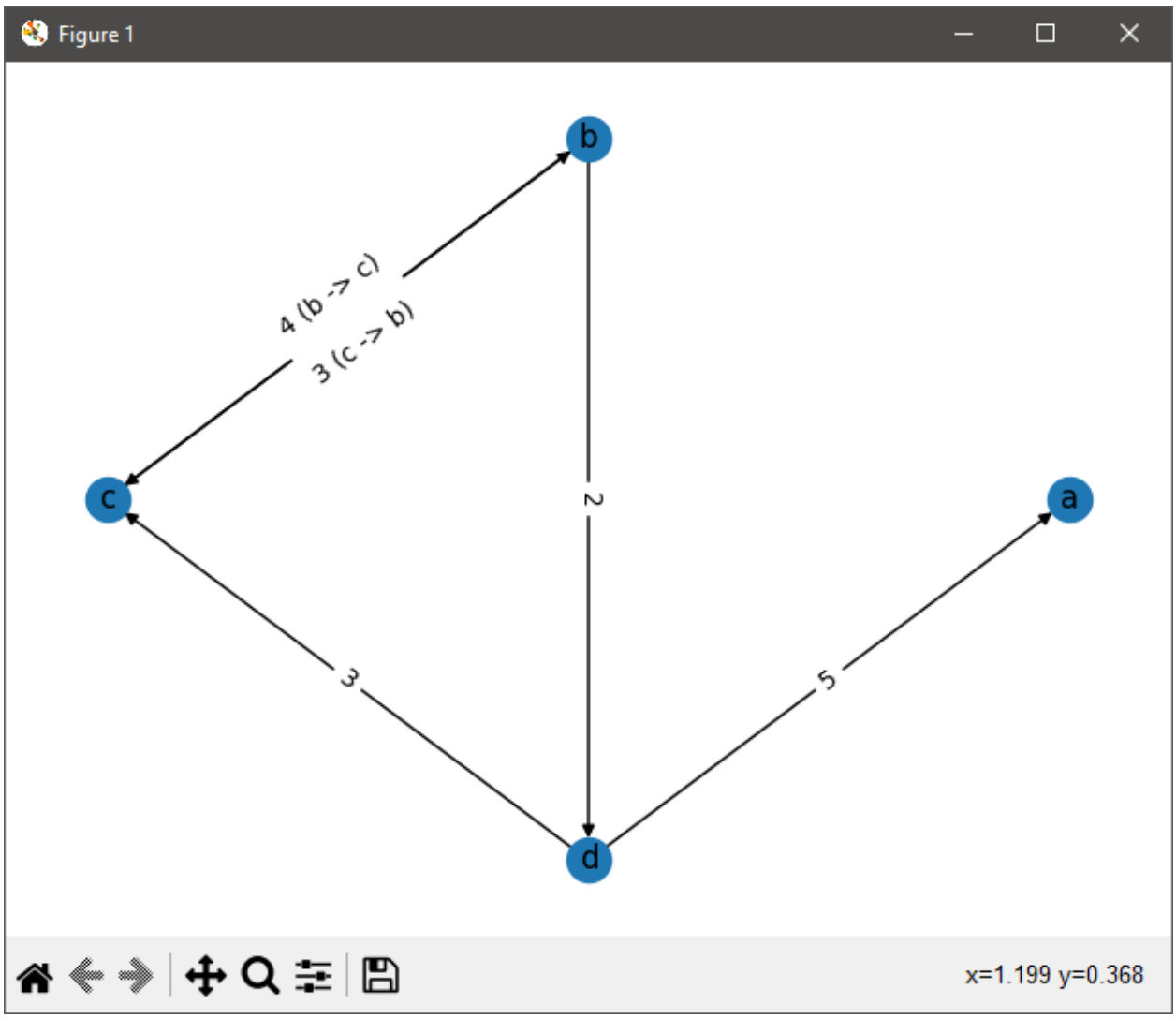

Рисунок 2.1 – Визуализация графа

3. Листинг

```
import networkx as nx
import matplotlib.pyplot as plt
import matplotlib as mpl
import numpy as np
import random
import math
import os
import copy

AINDEX = 97

# Создание матрицы размером (n x n)
def MakeMatrix(n):
    matrix = list()
    for i in range(n):
            matrix.append(list())
            for j in range(n):
                matrix[i].append(0)
    return matrix

# Глубокое копирование матрицы
def CopyMatrix(matrix, n):
    newMatrix = MakeMatrix(n)
    for i in range(n):
        for j in range(n):
            newMatrix[i][j] = matrix[i][j]
    return newMatrix

class Graph:
    # Конструктор класса
    def __init__(self, size, bFullGraph = False, bMultiedge = False, bLoop = False):
        self._nodes = size
        self._bFullGraph = bFullGraph
        self._bMultiedge = bMultiedge
        self._bLoop = bLoop
        self._imatrix = None
        self._amatrix = MakeMatrix(self._nodes)


    # Вывод матрицы смежности
    def showAdjacencyMatrix(self):
        print("Матрица смежности: ")
        for i in range(self._nodes * 2 + 1):
            for j in range(self._nodes * 2 + 1):
                if j % 2 == 1:
                    print(end = "  ") # |
                elif i % 2 == 1:
                    print(end = " ") # -
                elif (i == 0 and j != 0) or (j == 0 and i != 0):
                    print(end = f"{chr(AINDEX + (i + j) // 2 - 1)}")
                elif i // 2 > 0 and j // 2 > 0:
                    print(end = f"{self._amatrix[i // 2 - 1][j // 2 - 1]}")
                else:
                    print(end = " ")
            print()

    # Вывод графа (при помощи networkx)
    def showGraph(self):
        nodeMap = dict()
        edgeMap = dict()
        loopMap = dict()

        for i in range(0, self._nodes):
            nodeMap.update({i: chr(AINDEX + i)})

        count = 1
```

```
        for i in range(self._nodes):
            for j in range(self._nodes):
                if self._weightMatrix[i][j] != 0 and self._weightMatrix[i][j] != math.inf:
                    edgeName = ''
                    if self._weightMatrix[j][i] != 0 and self._weightMatrix[j][i] != math.inf:
                        edgeName = f"{self._weightMatrix[j][i]} ({chr(j + AINDEX)} -> {chr(i + AINDEX)})\n\n{self._weightMatrix[i][j]} ({chr(i + AINDEX)} -> {chr(j + AINDEX)})"
                    else:
                        edgeName = f"{self._weightMatrix[i][j]}"
                    edgeMap.update({(chr(AINDEX + i), chr(AINDEX + j)): edgeName})

        G = nx.DiGraph(np.array(self._amatrix))
        nx.relabel_nodes(G, nodeMap, False)
        pos = nx.circular_layout(G)
        nx.draw(G, pos, with_labels = True, arrows = True, arrowstyle = '-|>')
        nx.draw_networkx_edge_labels(G, pos, edge_labels = edgeMap)
        #nx.draw_networkx_edge_labels(G, pos, edge_labels = loopMap)
        plt.show()

    # Заполнение таблицы смежности при помощи рандома
    def setRandomMatrix(self):
        deltaIndex = 0 if self._bLoop else 1
        minEdges = 1 if self._bFullGraph else 0
        maxEdges = 3 if self._bMultiedge else 1

        for i in range(self._nodes):
            for j in range(self._nodes):
                value = random.randint(1, maxEdges) if random.randint(minEdges, 1) == 1 else 0
                self._amatrix[i][j] = value if (i != j or deltaIndex == 0) else 0

    # Подсчет ребер графа
    def updateEdges(self):
        edges = 0
        for i in range(self._nodes):
            for j in range(self._nodes):
                edges += self._amatrix[i][j]
        self._edges = edges

    # Создание матрицы инцидентности
    def makeIncidenceMatrix(self):
        self.updateEdges()

        self._imatrix = list()
        for i in range(self._nodes):
            self._imatrix.append(list())
            for j in range(self._edges):
                self._imatrix[i].append(0)

        edgeIndex = 0
        for i in range(self._nodes):
            for j in range(self._nodes):
                if self._amatrix[i][j] != 0:

                    for k in range(self._amatrix[i][j]):
                            self._imatrix[i][edgeIndex] = -1
                            self._imatrix[j][edgeIndex] = 1
                            edgeIndex += 1

    # Вывод матрицы инцидентности
    def showIncidenceMatrix(self):
        if self._imatrix is None:
            self.makeIncidenceMatrix()
        print("Матрица инцидентности: ")
        for i in range(self._nodes * 2 + 1):
```

```
            for j in range(self._edges * 2 + 1):
                if j % 2 == 1:
                    print(end = "  ") # |
                elif i % 2 == 1:
                    print(end = " ") # -
                elif j == 0 and i != 0:
                    print(end = f"{chr(AINDEX + (i + j) // 2 - 1)}")
                elif i == 0 and j != 0:
                    print(end = f"{j // 2}")
                elif i // 2 > 0 and j // 2 > 0:
                    if self._imatrix[i // 2 - 1][j // 2 - 1] < 0:
                        print(end = '\b')
                    print(end = f"{self._imatrix[i // 2 - 1][j // 2 - 1]}")
                    if j >= 20:
                        print(end = " ")
                else:
                    print(end = " ")
            print()

    def MakeWeightMatrix(self):
        self._weightMatrix = MakeMatrix(self._nodes)
        for i in range(self._nodes):
            for j in range(self._nodes):
                if i != j:
                    if self._amatrix[i][j] == 0:
                        self._weightMatrix[i][j] = math.inf
                    else:
                        self._weightMatrix[i][j] = random.randint(1, 5)

    # Вывод матрицы весов
    def showWeightMatrix(self):
        if self._weightMatrix == None:
            self.MakeWeightMatrix()
        print("Матрица весов: ")
        for i in range(self._nodes * 2 + 1):
            for j in range(self._nodes * 2 + 1):
                if j % 2 == 1:
                    print(end = "   ") # |
                elif i % 2 == 1:
                    print(end = " ") # -
                elif (i == 0 and j != 0) or (j == 0 and i != 0):
                    print(end = f"{chr(AINDEX + (i + j) // 2 - 1)}")
                elif i // 2 > 0 and j // 2 > 0:
                    if self._weightMatrix[i // 2 - 1][j // 2 - 1] == math.inf:
                        print(end = "\b")
                    print(end = f"{self._weightMatrix[i // 2 - 1][j // 2 - 1]}")
                else:
                    print(end = " ")
            print()

    def DijkstraAlgorithm(self):
        self._dijkstraMatrix = list()
        for i in range(self._nodes):
            self._dijkstraMatrix.append(MakeMatrix(self._nodes))

        for l in range(self._nodes):
            mask = ''
            for k in range(self._nodes):
                mask += chr(AINDEX + k)

            currentNode = l
            for i in range(self._nodes):
                #print(currentNode)
                if i == 0:
                    for j in range(self._nodes):
                        self._dijkstraMatrix[l][i][j] =
self._weightMatrix[currentNode][j]
                else:
```

```
                    for j in range(self._nodes):
                        if self._dijkstraMatrix[l][i-1][j] > self._dijkstraMatrix[l][i - 1][currentNode] + self._weightMatrix[currentNode][j]:
                            self._dijkstraMatrix[l][i][j] = self._dijkstraMatrix[l][i - 1][currentNode] + self._weightMatrix[currentNode][j]
                        else:
                            self._dijkstraMatrix[l][i][j] = self._dijkstraMatrix[l][i-1][j]

                temp = ''
                for k in range(len(mask)):
                    temp += mask[k] if mask[k] != chr(currentNode + AINDEX) else ''
                mask = temp

                minimum = math.inf
                for j in range(self._nodes):
                    if chr(AINDEX + j) in mask:
                        if minimum > self._dijkstraMatrix[l][i][j]:
                            minimum = self._dijkstraMatrix[l][i][j]
                            currentNode = j

    def ShowDijkstra(self):
        for l in range(self._nodes):
            print(f"Матрица Дейкстры из вершины {chr(l + AINDEX)}: ")
            for i in range(self._nodes * 2 + 1):
                for j in range(self._nodes * 2 + 1):
                    if j % 2 == 1:
                        print(end = "   ") # |
                    elif i % 2 == 1:
                        print(end = " ") # -
                    elif (i == 0 and j != 0) or (j == 0 and i != 0):
                        print(end = f"{chr(AINDEX + (i + j) // 2 - 1)}")
                    elif i // 2 > 0 and j // 2 > 0:
                        if self._dijkstraMatrix[i // 2 - 1][j // 2 - 1] == math.inf:
                            print(end = "\b")
                        print(end = f"{self._dijkstraMatrix[l][i // 2 - 1][j // 2 - 1]}")
                    else:
                        print(end = " ")
                print()



def main():
    menu = "Выберите режим работы программы (можно выбрать несколько):\n"
    menu += "Enter - Обычный режим\n1 - Полный граф\n2 - Кратные ребра\n3 - Наличие петель (могут быть кратными)\nq - Выход\n"

    mode = input(menu)

    while ('q' not in mode):
        NodeNumber = int(input("Введите количество вершин графа: "))
        graph = Graph(NodeNumber, '1' in mode, '2' in mode, '3' in mode)
        graph.setRandomMatrix()
        graph.showAdjacencyMatrix()
        print()
        graph.showIncidenceMatrix()
        graph.MakeWeightMatrix()
        graph.showWeightMatrix()
        graph.DijkstraAlgorithm()
        graph.ShowDijkstra()
        graph.showGraph()
        if os.system("clear"): os.system("cls")
        mode = input(menu)
    if os.system("clear"): os.system("cls")

if __name__ == "__main__":
    main()
```

4. Заключение

В ходе выполнения лабораторной работы изучили алгоритм Дейкстры, ознакомились с ориентированными графами, научились находит кратчайшие пути у нагруженного графа.