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: «Метрики графа»

Студент гр. 582-1

Полушвайко Константин Николаевич

__ декабря 2023 г.

2023

1. Задание

1. Для сгенерированного графа (из 1 лабораторной работы) реализовать функцию вывода матрицы метрик;
2. Реализовать по матрице метрик функцию нахождения радиуса, диаметра, центральных и периферийных вершин;
3. Для возведения матрицы в степень лучше реализовать функцию умножения матриц (это надо для поиска матрицы метрик)

2. Ход работы

Для начала работы определим алгоритм нахождения матрицы метрики:

1. Задаем матрицу метрики $\mu = (m_{ij})$ размерности равной размерности матрицы смежности с неопределенными элементами;
2. Создаем матрицу S = R + E, где R – матрица смежности, E – единичная матрица. Элементам главной диагонали матрицы метрики приравниваем 0, так как нет расстояния между одной и той же точкой;
3. Начальное значение степени k = 1. Всем элементам m_{ij}, которые неопределенны и выполняется условие $s_{ij} \neq 0$ в S^k приравнивается k, т.е. $m_{ij} = k$;
4. k += 1;
5. Проверяем матрицу метрики на устойчивость: $\mu^n == \mu^{n+1}$, где n – это номер итерации в алгоритме. Если матрица неустойчива, то возвращаемся к шагу 3;
6. Всем элементам m_{ij}, которые неопределенны: $m_{ij} = \infty$.

Данный алгоритм выполняется в методе MakeMetricMatrix, который можно найти в листинге (пункт 3).

Для нахождения радиуса и диаметра графа пользуемся матрицей метрики: ищем максимумы каждой строки, затем из этих значений ищем минимум – радиус и максимум – диаметр. Если максимум строки матрицы метрики соответствующей вершины равен радиусу – то это центральная вершина, если равна диаметру – то это периферийная вершина. Реализация нахождения этих характеристик представлена в методе FindMetrics.

Конечный код программы приведен в листинге (пункт 3).

На рисунках 2.1 и 2.2 представлен пример работы конечной программы.

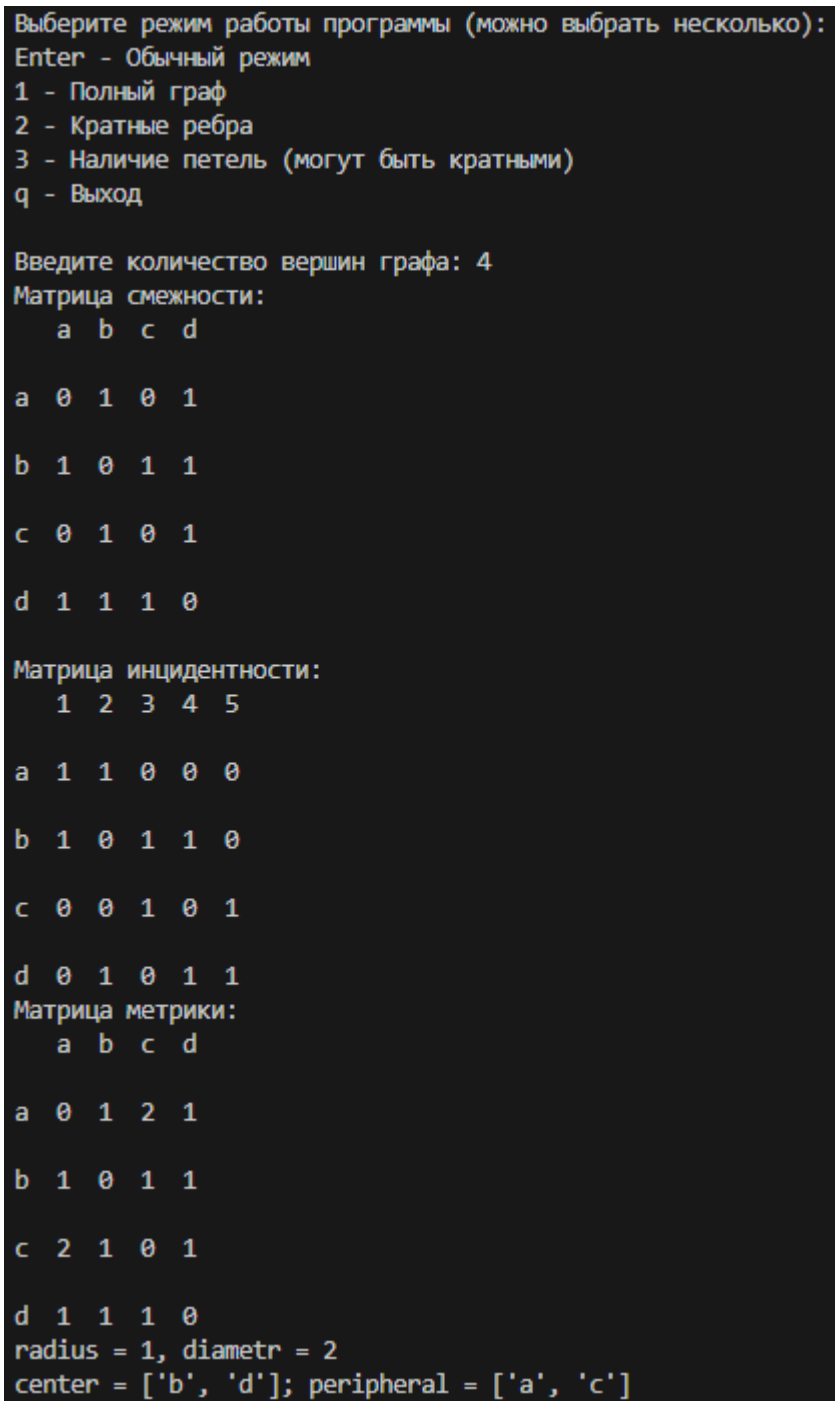

Рисунок 2.1 – Меню и вывод программы

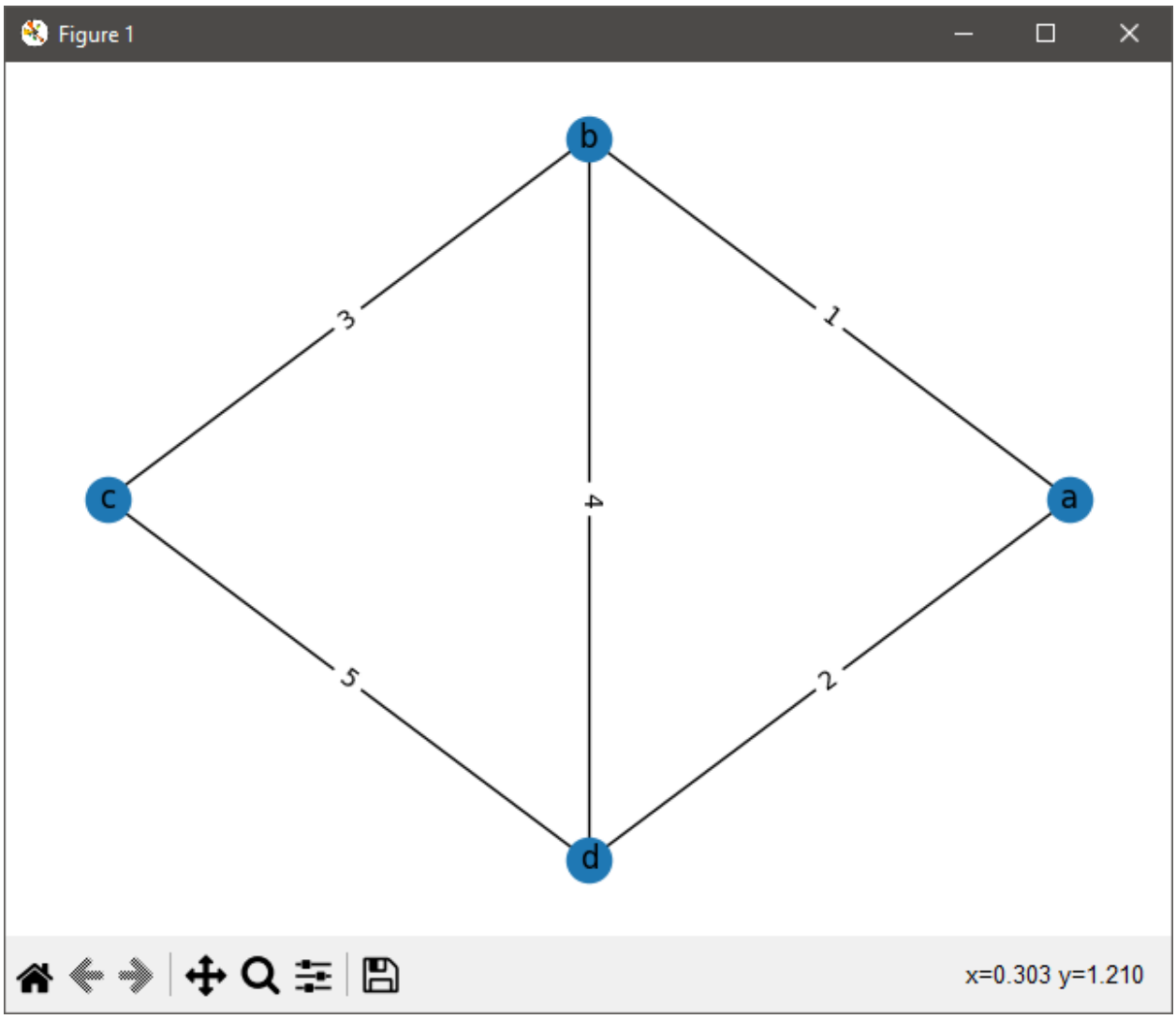

Рисунок 2.2 – Визуализация графа

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

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

# Возведение матрицы в степень
def PowerMatrix(matrix, n):
    newMatrix = MakeMatrix(n)

    for row in range(n):
        for col in range(n):
            for i in range(n):
                newMatrix[row][col] +=  matrix[row][i] * matrix[i][col]

    return newMatrix

def EqualMatrix(a, b, n):
    if a == None or b == None:
        return False
    for i in range(n):
        for j in range(n):
            if a[i][j] != b[i][j]:
                return False
    return True

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 матрицы смежности
    def showMetricMatrix(self):
        print("Матрица метрики: ")
        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._mmatrix[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(i, self._nodes):
                if self._amatrix[i][j] != 0:
                    edgeName = ""
                    if self._amatrix[i][j] > 1:
                        for k in range(self._amatrix[i][j] - 1):
                            edgeName += f"{count}, "
                            count += 1
                        edgeName += f"{count}"
                        count += 1
                    else:
                        edgeName = f"{count}"
                        count += 1
                    if i == j:
                        edgeName += "\n\n"
                        loopMap.update({(chr(AINDEX + i), chr(AINDEX + j)): edgeName})
                    else:
                        edgeMap.update({(chr(AINDEX + i), chr(AINDEX + j)): edgeName})

        G = nx.Graph(np.array(self._amatrix))
        nx.relabel_nodes(G, nodeMap, False)
        pos = nx.circular_layout(G)
        nx.draw(G, pos, with_labels = True)
        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(0, self._nodes):
```

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

    # Подсчет ребер графа
    def updateEdges(self):
        edges = 0
        for i in range(self._nodes):
            for j in range(i, 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(i, self._nodes):
                if self._amatrix[i][j] != 0:
                    value = 1
                    if i == j:
                        value = 2

                    for k in range(self._amatrix[i][j]):
                            self._imatrix[i][edgeIndex] = self._imatrix[j][edgeIndex] = value
                            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:
                    print(end = f"{self._imatrix[i // 2 - 1][j // 2 - 1]}")
                    if j >= 20:
                        print(end = " ")
                else:
                    print(end = " ")
            print()


    def MakeMetricMatrix(self):
        self._mmatrix = None
        tempMatrix = MakeMatrix(self._nodes)
        smatrix = MakeMatrix(self._nodes)
        k = 1
        for i in range(self._nodes):
            for j in range(self._nodes):
                smatrix[i][j] = 1 if self._amatrix[i][j] != 0 else 0
```

```
                if i == j:
                    smatrix[i][j] += 1
        while not EqualMatrix(self._mmatrix, tempMatrix, self._nodes):
            self._mmatrix = CopyMatrix(tempMatrix, self._nodes)
            for i in range(self._nodes):
                for j in range(self._nodes):
                    if (not i==j and smatrix[i][j] != 0 and tempMatrix[i][j] == 0):
                        tempMatrix[i][j] += k
            smatrix = PowerMatrix(smatrix, self._nodes)
            k+=1

    def FindMetrics(self):
        maxRow = list()
        for i in range(self._nodes):
            maxRow.append(max(self._mmatrix[i]))
        self.radius = min(maxRow) if min(maxRow) != 0 else math.inf
        self.diametr = max(maxRow) if self.radius != math.inf else math.inf
        print (f"radius = {self.radius}, diametr = {self.diametr}")
        self.peripheral = list()
        self.central = list()
        for i in range(self._nodes):
            if max(self._mmatrix[i]) == self.radius:
                self.central.append(chr(AINDEX + i))
            if max(self._mmatrix[i]) == self.diametr:
                self.peripheral.append(chr(AINDEX + i))
        print (f"center = {self.central}; peripheral = {self.peripheral}")

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.MakeMetricMatrix()
        graph.showMetricMatrix()
        graph.FindMetrics()
        graph.showGraph()
        os.system("cls")
        mode = input(menu)
    os.system("cls")

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

4. Заключение

В ходе выполнения лабораторной работы изучили алгоритм создания матрицы метрики, получили периферийные и центральные точки графа, а также нашли радиус и диаметр.