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: «Раскраска графа»

Студент гр. 582-1

Полушвайко Константин Николаевич

__ декабря 2023 г.

2023

1. Задание

1. Реализовать алгоритм поиска максимального пустого подграфа с выводом множеств вершин;
2. Правильно раскрасить граф и отрисовать его;
3. Вывести хроматическое число графа.

2. Ход работы

Чтобы правильно раскрасить граф можно применить алгоритм Магу-Вейсмана, в ходе которого сначала надо найти максимально пустые подграфы. Для нахождения максимально пустых подграфов реализуем функцию FindEmptyGraphs, приведенную в листинге пунктом ниже. Для того, чтобы задать цвет каждой вершины нужно задать массив цветов. Для этой цели определим метод GraphColoring. Хроматическим числом графа является количество используемых цветов в правильной раскраске графа.

Конечный код программы приведен в листинге (пункт 3).

На рисунках 2.1 и 2.2 представлен пример работы конечной программы.

```
Выберите режим работы программы (можно выбрать несколько):
Enter - Обычный режим
1 - Полный граф
2 - Кратные ребра
3 - Наличие петель (могут быть кратными)
q - Выход

Введите количество вершин графа: 5
Матрица смежности:
  a  b  c  d  e

a  0  0  1  1  0

b  0  0  0  1  0

c  1  0  0  0  1

d  1  1  0  0  0

e  0  0  1  0  0

Матрица инцидентности:
   1  2  3  4

a  1  1  0  0

b  0  0  1  0

c  1  0  0  1

d  0  1  1  0

e  0  0  0  1
S = ['de', 'cd', 'be', 'bc', 'ae', 'abe', 'ab'] - максимально пустые подграфы
G = 2 - хроматическое число графа
Матрица метрики:
   a  b  c  d  e

a  0  2  1  1  2

b  2  0  3  1  3

c  1  3  0  2  1

d  1  1  2  0  3

e  2  3  1  3  0
radius = 2, diametr = 3
center = ['a']; peripheral = ['b', 'c', 'd', 'e']
```

Рисунок 2.1 – Меню и вывод программы

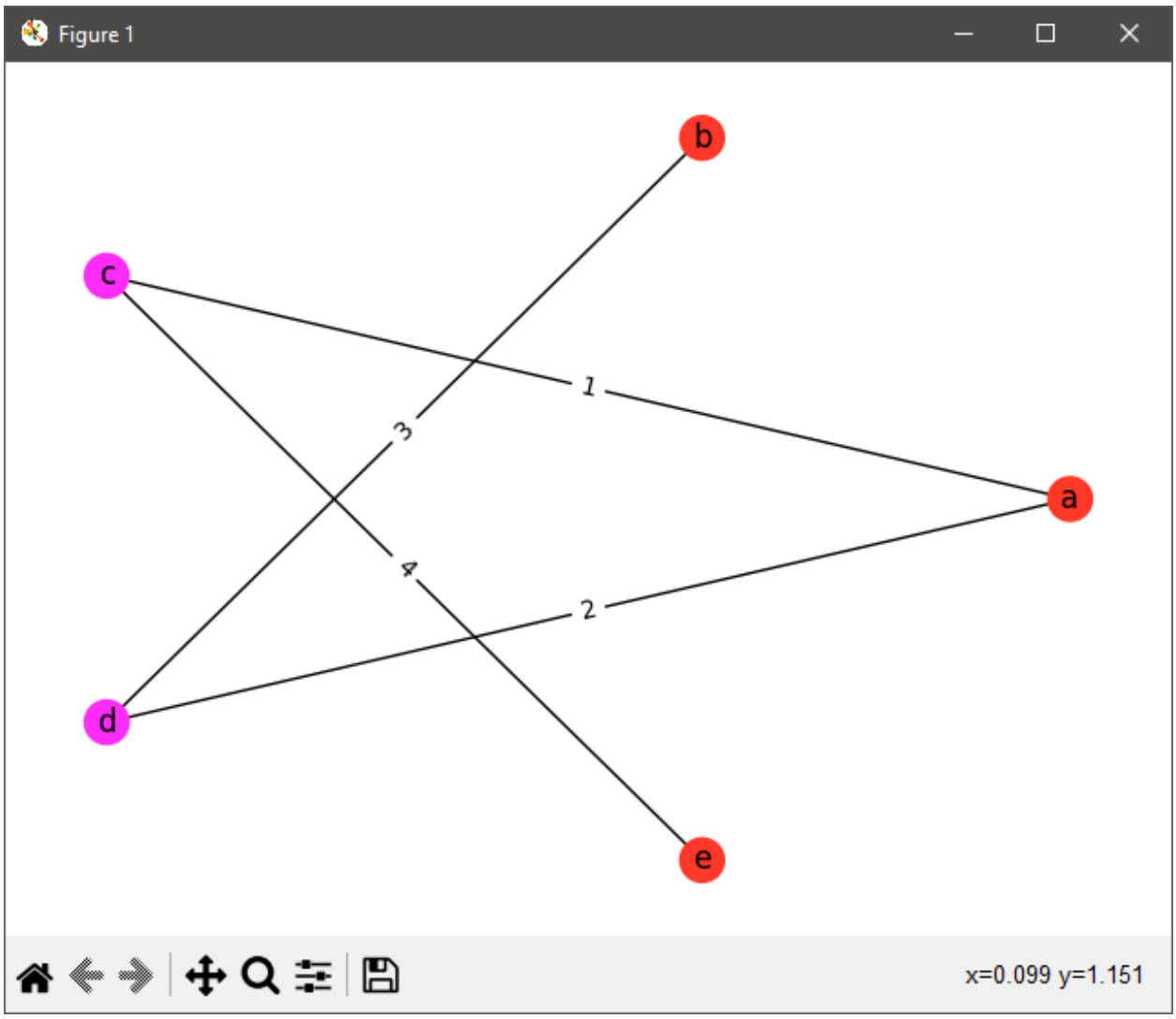

Рисунок 2.2 – Визуализация графа

3. Листинг

```
from sympy.parsing.sympy_parser import (parse_expr, standard_transformations,
implicit_multiplication_application)
import sympy as sp
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
COLORS = ['#FF3829', '#FC2DF7', '#3946FF', '#3884FF', '#AD38FF', '#24FF2C', '#D1FC28',
'#A0FF24', '#FCB028', '#FFE423']

# 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

# Проверка на эквивалентность матриц
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

# 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

def SortCapacities(array):
    result = copy.deepcopy(array)
    for i in range(len(result)):
        for j in range(i, len(result)):
            if len(result[i]) < len(result[j]):
                temp = result[i]
                result[i] = result[j]
                result[j] = temp
    return result

class Graph:
    # Конструктор класса
```

```
65:     def __init__(self, size, bFullGraph = False, bMultiedge = False, bLoop = False):
66:         self._nodes = size
67:         self._bFullGraph = bFullGraph
68:         self._bMultiedge = bMultiedge
69:         self._bLoop = bLoop
70:         self._imatrix = None
71:         self._amatrix = MakeMatrix(self._nodes)
72:
73:
74:     # Вывод матрицы смежности
75:     def showAdjacencyMatrix(self):
76:         print("Матрица смежности: ")
77:         for i in range(self._nodes * 2 + 1):
78:             for j in range(self._nodes * 2 + 1):
79:                 if j % 2 == 1:
80:                     print(end = "  ") # |
81:                 elif i % 2 == 1:
82:                     print(end = " ") # -
83:                 elif (i == 0 and j != 0) or (j == 0 and i != 0):
84:                     print(end = f"{chr(AINDEX + (i + j) // 2 - 1)}")
85:                 elif i // 2 > 0 and j // 2 > 0:
86:                     print(end = f"{self._amatrix[i // 2 - 1][j // 2 - 1]}")
87:                 else:
88:                     print(end = " ")
89:             print()
90:
91:     # Вывод матрицы смежности
92:     def showMetricMatrix(self):
93:         print("Матрица метрики: ")
94:         for i in range(self._nodes * 2 + 1):
95:             for j in range(self._nodes * 2 + 1):
96:                 if j % 2 == 1:
97:                     print(end = "  ") # |
98:                 elif i % 2 == 1:
99:                     print(end = " ") # -
100:                elif (i == 0 and j != 0) or (j == 0 and i != 0):
101:                    print(end = f"{chr(AINDEX + (i + j) // 2 - 1)}")
102:                elif i // 2 > 0 and j // 2 > 0:
103:                    print(end = f"{self._mmatrix[i // 2 - 1][j // 2 - 1]}")
104:                else:
105:                    print(end = " ")
106:            print()
107:
108:    # Вывод графа (при помощи networkx)
109:    def showGraph(self):
110:        nodeMap = dict()
111:        edgeMap = dict()
112:        loopMap = dict()
113:        colorMap = list()
114:        for i in range(self._nodes):
115:            colorMap.append("blue")
116:
117:        for i in range(0, self._nodes):
118:            nodeMap.update({i: chr(AINDEX + i)})
119:
120:        count = 1
121:        for i in range(self._nodes):
122:            for j in range(i, self._nodes):
123:                if self._amatrix[i][j] != 0:
124:                    edgeName = ""
125:                    if self._amatrix[i][j] > 1:
126:                        for k in range(self._amatrix[i][j] - 1):
127:                            edgeName += f"{count}, "
128:                            count += 1
129:                        edgeName += f"{count}"
130:                        count += 1
131:                    else:
132:                        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})

        for i in range(len(self._colorMap)):
            for j in range(len(self._colorMap[i])):
                colorMap[ord(self._colorMap[i][j]) - AINDEX] = COLORS[i]

        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, node_color = colorMap)
        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()

    # Создаение матрицы метрики
    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

    # Нахождение и вывод метрик (центральные\переферийные точки, радиус, диаметр)
    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 or self.radius == math.inf:
                self.central.append(chr(AINDEX + i))
            if max(self._mmatrix[i]) == self.diametr or self.diametr == math.inf:
                self.peripheral.append(chr(AINDEX + i))
        print (f"center = {self.central}; peripheral = {self.peripheral}")

    # Поиск максимально пустых графов
    def FindEmptyGraphs(self):
        self._emptyGraphs = list()
        polynomial = ""
        mask = ""
        for i in range(self._nodes):
            mask += chr(AINDEX + i)
        for i in range(self._edges):
            split = True
            polynomial += "("
            for j in range(self._nodes):
                if self._imatrix[j][i] == 1:
                    polynomial += chr(AINDEX + j)
                    if split:
```

```
                            polynomial+= " + "
                            split = False
                polynomial += ")"
            #print(polynomial)
            transformations = standard_transformations +
(implicit_multiplication_application,)
            polynomial = str(sp.expand(parse_expr(polynomial,
transformations=transformations)))
            temp = ""
            for i in range(len(polynomial)):
                if polynomial[i] != "*" and not polynomial[i].isdigit():
                    temp += polynomial[i]
            polynomial = temp
            multipliers = polynomial.split(" + ")
            #print(polynomial)
            for i in range(len(multipliers)):
                if len(multipliers[i]) + 1 >= self._nodes:
                    continue
                temp = ""
                for j in range(self._nodes):
                    if mask[j] not in multipliers[i]:
                        temp += mask[j]
                if temp not in self._emptyGraphs:
                    self._emptyGraphs.append(temp)

            print(f"S = {self._emptyGraphs} - максимально пустые подграфы")

        def GraphColoring(self):
            if self._emptyGraphs == None:
                self.FindEmptyGraphs()
            colorMap = list()
            counter = 0
            colorNodes = SortCapacities(self._emptyGraphs)
            while colorNodes and len(colorNodes[0]) != 0:
                oneColor = colorNodes.pop(0)
                colorMap.append(oneColor)
                counter += 1
                for i in range(len(colorNodes)):
                    temp = ""
                    for j in range(len(colorNodes[i])):
                        if colorNodes[i][j] not in oneColor:
                            temp += colorNodes[i][j]
                    colorNodes[i] = temp
                colorNodes = SortCapacities(colorNodes)
            #print(colorMap)
            print(f"G = {len(colorMap)} - хроматическое число графа")
            self._colorMap = colorMap



    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.FindEmptyGraphs()
            graph.GraphColoring()
            graph.MakeMetricMatrix()
            graph.showMetricMatrix()
```

```
332:             graph.FindMetrics()
333:             graph.showGraph()
334:             os.system("cls")
335:             mode = input(menu)
336:         os.system("cls")
337: 
338: if __name__ == "__main__":
339:     main()
```

4. Заключение

В ходе выполнения лабораторной работы изучили алгоритм раскраски графа, нахождения максимально пустых подграфов, хроматического числа графа.