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.

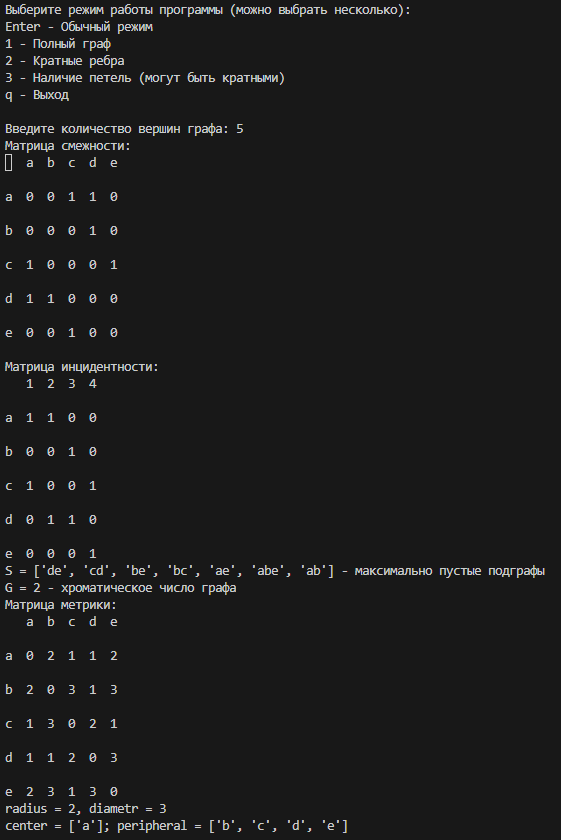


Рисунок 2.1 – Меню и вывод программы

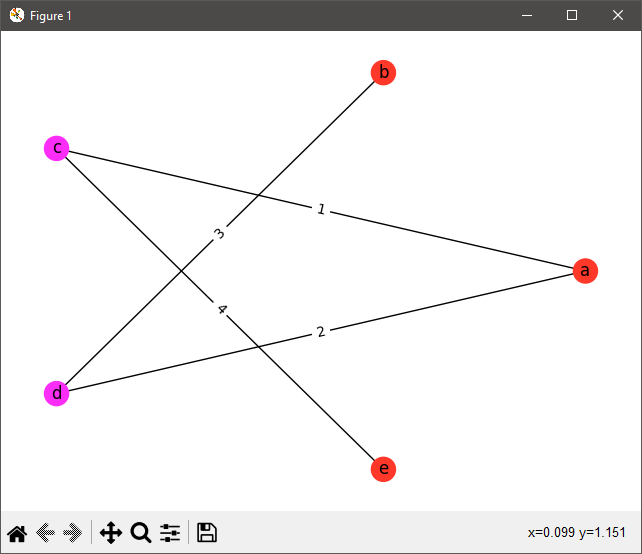


Рисунок 2.2 – Визуализация графа

## 3. Листинг

1. from sympy.parsing.sympy\_parser import (parse\_expr, standard\_transformations, implicit\_multiplication\_application)
2. import sympy as sp
3. import networkx as nx
4. import matplotlib.pyplot as plt
5. import matplotlib as mpl
6. import numpy as np
7. import random
8. import math
9. import os
10. import copy
11. AINDEX = 97
12. COLORS = ['#FF3829', '#FC2DF7', '#3946FF', '#3884FF', '#AD38FF', '#24FF2C', '#D1FC28', '#A0FF24', '#FCB028', '#FFE423']
13. # Создание матрицы размером (n x n)
14. def MakeMatrix(n):
15. matrix = list()
16. for i in range(n):
17. matrix.append(list())
18. for j in range(n):
19. matrix[i].append(0)
20. return matrix
21. # Возведение матрицы в степень
22. def PowerMatrix(matrix, n):
23. newMatrix = MakeMatrix(n)
24. for row in range(n):
25. for col in range(n):
26. for i in range(n):
27. newMatrix[row][col] +=  matrix[row][i] \* matrix[i][col]
29. return newMatrix
30. # Проверка на эквивалентность матриц
31. def EqualMatrix(a, b, n):
32. if a == None or b == None:
33. return False
34. for i in range(n):
35. for j in range(n):
36. if a[i][j] != b[i][j]:
37. return False
38. return True
39. # Глубокое копирование матрицы
40. def CopyMatrix(matrix, n):
41. newMatrix = MakeMatrix(n)
42. for i in range(n):
43. for j in range(n):
44. newMatrix[i][j] = matrix[i][j]
45. return newMatrix
46. def SortCapacities(array):
47. result = copy.deepcopy(array)
48. for i in range(len(result)):
49. for j in range(i, len(result)):
50. if len(result[i]) < len(result[j]):
51. temp = result[i]
52. result[i] = result[j]
53. result[j] = temp
54. return result
55. class Graph:
56. # Конструктор класса
57. def \_\_init\_\_(self, size, bFullGraph = False, bMultiedge = False, bLoop = False):
58. self.\_nodes = size
59. self.\_bFullGraph = bFullGraph
60. self.\_bMultiedge = bMultiedge
61. self.\_bLoop = bLoop
62. self.\_imatrix = None
63. self.\_amatrix = MakeMatrix(self.\_nodes)

66. # Вывод матрицы смежности
67. def showAdjacencyMatrix(self):
68. print("Матрица смежности: ")
69. for i in range(self.\_nodes \* 2 + 1):
70. for j in range(self.\_nodes \* 2 + 1):
71. if j % 2 == 1:
72. print(end = "  ") # |
73. elif i % 2 == 1:
74. print(end = " ") # -
75. elif (i == 0 and j != 0) or (j == 0 and i != 0):
76. print(end = f"{chr(AINDEX + (i + j) // 2 - 1)}")
77. elif i // 2 > 0 and j // 2 > 0:
78. print(end = f"{self.\_amatrix[i // 2 - 1][j // 2 - 1]}")
79. else:
80. print(end = " ")
81. print()
83. # Вывод матрицы смежности
84. def showMetricMatrix(self):
85. print("Матрица метрики: ")
86. for i in range(self.\_nodes \* 2 + 1):
87. for j in range(self.\_nodes \* 2 + 1):
88. if j % 2 == 1:
89. print(end = "  ") # |
90. elif i % 2 == 1:
91. print(end = " ") # -
92. elif (i == 0 and j != 0) or (j == 0 and i != 0):
93. print(end = f"{chr(AINDEX + (i + j) // 2 - 1)}")
94. elif i // 2 > 0 and j // 2 > 0:
95. print(end = f"{self.\_mmatrix[i // 2 - 1][j // 2 - 1]}")
96. else:
97. print(end = " ")
98. print()
99. # Вывод графа (при помощи networkx)
100. def showGraph(self):
101. nodeMap = dict()
102. edgeMap = dict()
103. loopMap = dict()
104. colorMap = list()
105. for i in range(self.\_nodes):
106. colorMap.append("blue")
107. for i in range(0, self.\_nodes):
108. nodeMap.update({i: chr(AINDEX + i)})
109. count = 1
110. for i in range(self.\_nodes):
111. for j in range(i, self.\_nodes):
112. if self.\_amatrix[i][j] != 0:
113. edgeName = ""
114. if self.\_amatrix[i][j] > 1:
115. for k in range(self.\_amatrix[i][j] - 1):
116. edgeName += f"{count}, "
117. count += 1
118. edgeName += f"{count}"
119. count += 1
120. else:
121. edgeName = f"{count}"
122. count += 1
123. if i == j:
124. edgeName += "\n\n"
125. loopMap.update({(chr(AINDEX + i), chr(AINDEX + j)): edgeName})
126. else:
127. edgeMap.update({(chr(AINDEX + i), chr(AINDEX + j)): edgeName})
129. for i in range(len(self.\_colorMap)):
130. for j in range(len(self.\_colorMap[i])):
131. colorMap[ord(self.\_colorMap[i][j]) - AINDEX] = COLORS[i]
132. G = nx.Graph(np.array(self.\_amatrix))
133. nx.relabel\_nodes(G, nodeMap, False)
134. pos = nx.circular\_layout(G)
135. nx.draw(G, pos, with\_labels = True, node\_color = colorMap)
136. nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels = edgeMap)
137. nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels = loopMap)
138. plt.show()
139. # Заполнение таблицы смежности при помощи рандома
140. def setRandomMatrix(self):
141. deltaIndex = 0 if self.\_bLoop else 1
142. minEdges = 1 if self.\_bFullGraph else 0
143. maxEdges = 3 if self.\_bMultiedge else 1
144. for i in range(0, self.\_nodes):
145. for j in range(i + deltaIndex, self.\_nodes):
146. value = random.randint(1, maxEdges) if random.randint(minEdges, 1) == 1 else 0
147. self.\_amatrix[i][j] = self.\_amatrix[j][i] = value
149. # Подсчет ребер графа
150. def updateEdges(self):
151. edges = 0
152. for i in range(self.\_nodes):
153. for j in range(i, self.\_nodes):
154. edges += self.\_amatrix[i][j]
155. self.\_edges = edges
157. # Создание матрицы инцидентности
158. def makeIncidenceMatrix(self):
159. self.updateEdges()
160. self.\_imatrix = list()
161. for i in range(self.\_nodes):
162. self.\_imatrix.append(list())
163. for j in range(self.\_edges):
164. self.\_imatrix[i].append(0)
166. edgeIndex = 0
167. for i in range(self.\_nodes):
168. for j in range(i, self.\_nodes):
169. if self.\_amatrix[i][j] != 0:
170. value = 1
171. if i == j:
172. value = 2
174. for k in range(self.\_amatrix[i][j]):
175. self.\_imatrix[i][edgeIndex] = self.\_imatrix[j][edgeIndex] = value
176. edgeIndex += 1
177. # Вывод матрицы инцидентности
178. def showIncidenceMatrix(self):
179. if self.\_imatrix is None:
180. self.makeIncidenceMatrix()
181. print("Матрица инцидентности: ")
182. for i in range(self.\_nodes \* 2 + 1):
183. for j in range(self.\_edges \* 2 + 1):
184. if j % 2 == 1:
185. print(end = "  ") # |
186. elif i % 2 == 1:
187. print(end = " ") # -
188. elif j == 0 and i != 0:
189. print(end = f"{chr(AINDEX + (i + j) // 2 - 1)}")
190. elif i == 0 and j != 0:
191. print(end = f"{j // 2}")
192. elif i // 2 > 0 and j // 2 > 0:
193. print(end = f"{self.\_imatrix[i // 2 - 1][j // 2 - 1]}")
194. if j >= 20:
195. print(end = " ")
196. else:
197. print(end = " ")
198. print()
200. # Создаение матрицы метрики
201. def MakeMetricMatrix(self):
202. self.\_mmatrix = None
203. tempMatrix = MakeMatrix(self.\_nodes)
204. smatrix = MakeMatrix(self.\_nodes)
205. k = 1
206. for i in range(self.\_nodes):
207. for j in range(self.\_nodes):
208. smatrix[i][j] = 1 if self.\_amatrix[i][j] != 0 else 0
209. if i == j:
210. smatrix[i][j] += 1
211. while not EqualMatrix(self.\_mmatrix, tempMatrix, self.\_nodes):
212. self.\_mmatrix = CopyMatrix(tempMatrix, self.\_nodes)
213. for i in range(self.\_nodes):
214. for j in range(self.\_nodes):
215. if (not i==j and smatrix[i][j] != 0 and tempMatrix[i][j] == 0):
216. tempMatrix[i][j] += k
217. smatrix = PowerMatrix(smatrix, self.\_nodes)
218. k+=1
219. # Нахождение и вывод метрик (центральные\переферийные точки, радиус, диаметр)
220. def FindMetrics(self):
221. maxRow = list()
222. for i in range(self.\_nodes):
223. maxRow.append(max(self.\_mmatrix[i]))
224. self.radius = min(maxRow) if min(maxRow) != 0 else math.inf
225. self.diametr = max(maxRow) if self.radius != math.inf else math.inf
226. print (f"radius = {self.radius}, diametr = {self.diametr}")
227. self.peripheral = list()
228. self.central = list()
229. for i in range(self.\_nodes):
230. if max(self.\_mmatrix[i]) == self.radius or self.radius == math.inf:
231. self.central.append(chr(AINDEX + i))
232. if max(self.\_mmatrix[i]) == self.diametr or self.diametr == math.inf:
233. self.peripheral.append(chr(AINDEX + i))
234. print (f"center = {self.central}; peripheral = {self.peripheral}")
236. # Поиск максимально пустых графов
237. def FindEmptyGraphs(self):
238. self.\_emptyGraphs = list()
239. polynomial = ""
240. mask = ""
241. for i in range(self.\_nodes):
242. mask += chr(AINDEX + i)
243. for i in range(self.\_edges):
244. split = True
245. polynomial += "("
246. for j in range(self.\_nodes):
247. if self.\_imatrix[j][i] == 1:
248. polynomial += chr(AINDEX + j)
249. if split:
250. polynomial+= " + "
251. split = False
252. polynomial += ")"
253. #print(polynomial)
254. transformations = standard\_transformations + (implicit\_multiplication\_application,)
255. polynomial = str(sp.expand(parse\_expr(polynomial, transformations=transformations)))
256. temp = ""
257. for i in range(len(polynomial)):
258. if polynomial[i] != "\*" and not polynomial[i].isdigit():
259. temp += polynomial[i]
260. polynomial = temp
261. multipliers = polynomial.split(" + ")
262. #print(polynomial)
263. for i in range(len(multipliers)):
264. if len(multipliers[i]) + 1 >= self.\_nodes:
265. continue
266. temp = ""
267. for j in range(self.\_nodes):
268. if mask[j] not in multipliers[i]:
269. temp += mask[j]
270. if temp not in self.\_emptyGraphs:
271. self.\_emptyGraphs.append(temp)
272. print(f"S = {self.\_emptyGraphs} - максимально пустые подграфы")
274. def GraphColoring(self):
275. if self.\_emptyGraphs == None:
276. self.FindEmptyGraphs()
277. colorMap = list()
278. counter = 0
279. colorNodes = SortCapacities(self.\_emptyGraphs)
280. while colorNodes and len(colorNodes[0]) != 0:
281. oneColor = colorNodes.pop(0)
282. colorMap.append(oneColor)
283. counter += 1
284. for i in range(len(colorNodes)):
285. temp = ""
286. for j in range(len(colorNodes[i])):
287. if colorNodes[i][j] not in oneColor:
288. temp += colorNodes[i][j]
289. colorNodes[i] = temp
290. colorNodes = SortCapacities(colorNodes)
291. #print(colorMap)
292. print(f"G = {len(colorMap)} - хроматическое число графа")
293. self.\_colorMap = colorMap

296. def main():
297. menu = "Выберите режим работы программы (можно выбрать несколько):\n"
298. menu += "Enter - Обычный режим\n1 - Полный граф\n2 - Кратные ребра\n3 - Наличие петель (могут быть кратными)\nq - Выход\n"
299. mode = input(menu)
300. while ('q' not in mode):
301. NodeNumber = int(input("Введите количество вершин графа: "))
302. graph = Graph(NodeNumber, '1' in mode, '2' in mode, '3' in mode)
303. graph.setRandomMatrix()
304. graph.showAdjacencyMatrix()
305. print()
306. graph.showIncidenceMatrix()
307. graph.FindEmptyGraphs()
308. graph.GraphColoring()
309. graph.MakeMetricMatrix()
310. graph.showMetricMatrix()
311. graph.FindMetrics()
312. graph.showGraph()
313. os.system("cls")
314. mode = input(menu)
315. os.system("cls")
316. if \_\_name\_\_ == "\_\_main\_\_":
317. main()

## 4. Заключение

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