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: «Генерация графа»

Студент гр. 582-1

Полушвайко Константин Николаевич

\_\_ ноября 2023 г.

2023

## 1. Задание

1. Сгенерировать матрицу смежности произвольной задаваемой размерности n.
2. Генерация должна предусматривать возможность (по выбору) генерации полных графов, графом с кратными ребрами и петлями.
3. Сделать отрисовку графа с помощью NetworkX (или подобные библиотеки)
4. Реализовать функцию преобразования сгенерированной в 1 матрицы смежности в матрицу инцидентности.

## 2. Ход работы

В начале, определим язык и структуру кода: будем использовать язык python и придерживаться парадигмы ООП. Создадим класс Graph и определим поля для массива смежности, инцидентности, разных режимов работы с графом и количества вершин и ребер.

Определим методы для класса: конструктор, заполнение матрицы смежности случайным образом, вывод матрицы смежности и инцидентности, вывод графа, создание матрицы инцидентности, подсчет ребер графа, а также определим главную функцию (исполняет роль менеджера-меню).

Конечный код программы приведен в листинге (пункт 3).

На рисунках 2.1-4 представлены примеры работы конечной программы.

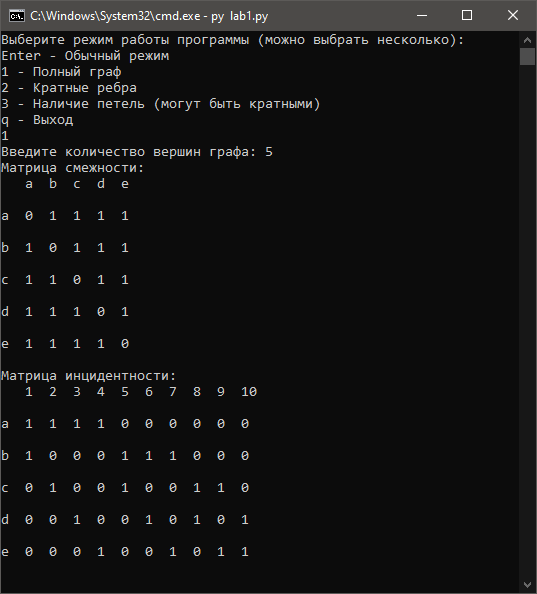


Рисунок 2.1 – Первый запуск программы

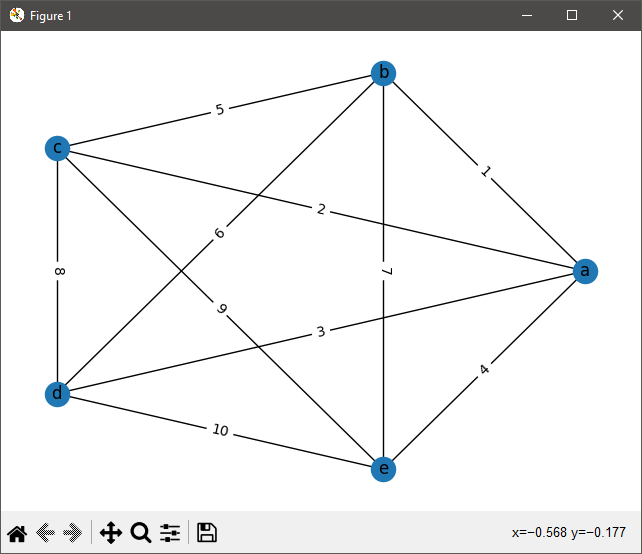


Рисунок 2.2 – Первый граф

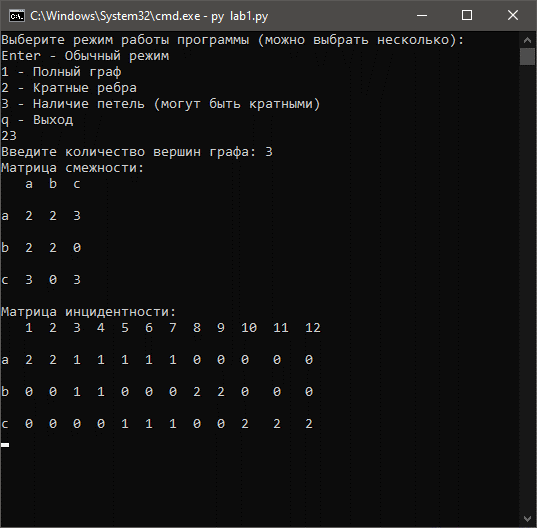


Рисунок 2.3 – Второй запуск программы

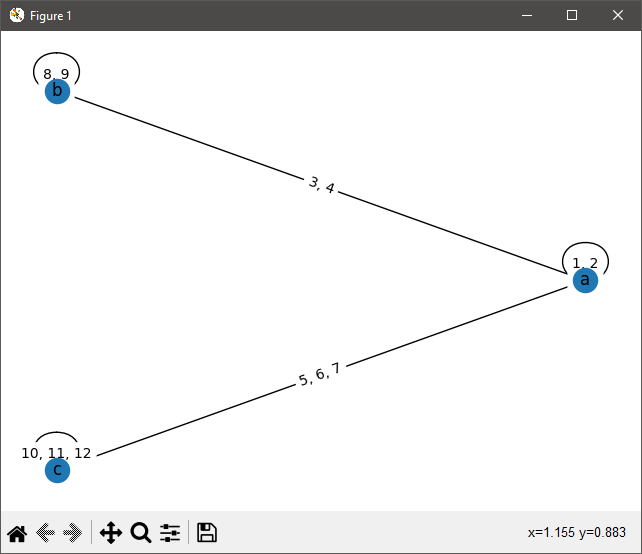


Рисунок 2.4 – Второй граф

## 3. Листинг

1. import networkx as nx
2. import matplotlib.pyplot as plt
3. import numpy as np
4. import random
5. import os
6. AINDEX = 97
7. class Graph:
8. # Конструктор класса
9. def \_\_init\_\_(self, size, bFullGraph = False, bMultiedge = False, bLoop = False):
10. self.\_nodes = size
11. self.\_bFullGraph = bFullGraph
12. self.\_bMultiedge = bMultiedge
13. self.\_bLoop = bLoop
14. self.\_imatrix = None
15. self.\_amatrix = list()
16. for i in range(self.\_nodes):
17. self.\_amatrix.append(list())
18. for j in range(self.\_nodes):
19. self.\_amatrix[i].append(0)
21. # Вывод матрицы смежности
22. def showAdjacencyMatrix(self):
23. print("Матрица смежности: ")
24. for i in range(self.\_nodes \* 2 + 1):
25. for j in range(self.\_nodes \* 2 + 1):
26. if j % 2 == 1:
27. print(end = "  ") # |
28. elif i % 2 == 1:
29. print(end = " ") # -
30. elif (i == 0 and j != 0) or (j == 0 and i != 0):
31. print(end = f"{chr(AINDEX + (i + j) // 2 - 1)}")
32. elif i // 2 > 0 and j // 2 > 0:
33. print(end = f"{self.\_amatrix[i // 2 - 1][j // 2 - 1]}")
34. else:
35. print(end = " ")
36. print()
37. # Вывод графа (при помощи networkx)
38. def showGraph(self):
39. nodeMap = dict()
40. edgeMap = dict()
41. loopMap = dict()
42. for i in range(0, self.\_nodes):
43. nodeMap.update({i: chr(AINDEX + i)})
44. count = 1
45. for i in range(self.\_nodes):
46. for j in range(i, self.\_nodes):
47. if self.\_amatrix[i][j] != 0:
48. edgeName = ""
49. if self.\_amatrix[i][j] > 1:
50. for k in range(self.\_amatrix[i][j] - 1):
51. edgeName += f"{count}, "
52. count += 1
53. edgeName += f"{count}"
54. count += 1
55. else:
56. edgeName = f"{count}"
57. count += 1
58. if i == j:
59. edgeName += "\n\n"
60. loopMap.update({(chr(AINDEX + i), chr(AINDEX + j)): edgeName})
61. else:
62. edgeMap.update({(chr(AINDEX + i), chr(AINDEX + j)): edgeName})
63. G = nx.Graph(np.array(self.\_amatrix))
64. nx.relabel\_nodes(G, nodeMap, False)
65. pos = nx.circular\_layout(G)
66. nx.draw(G, pos, with\_labels = True)
67. nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels = edgeMap)
68. nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels = loopMap)
69. plt.show()
70. # Заполнение матрицы смежности при помощи рандома
71. def setRandomMatrix(self):
72. deltaIndex = 0 if self.\_bLoop else 1
73. minEdges = 1 if self.\_bFullGraph else 0
74. maxEdges = 3 if self.\_bMultiedge else 1
75. for i in range(0, self.\_nodes):
76. for j in range(i + deltaIndex, self.\_nodes):
77. self.\_amatrix[i][j] = self.\_amatrix[j][i] = random.randint(minEdges, maxEdges)
79. # Подсчет ребер графа
80. def updateEdges(self):
81. edges = 0
82. for i in range(self.\_nodes):
83. for j in range(i, self.\_nodes):
84. edges += self.\_amatrix[i][j]
85. self.\_edges = edges
87. # Создание матрицы инцидентности
88. def makeIncidenceMatrix(self):
89. self.updateEdges()
90. self.\_imatrix = list()
91. for i in range(self.\_nodes):
92. self.\_imatrix.append(list())
93. for j in range(self.\_edges):
94. self.\_imatrix[i].append(0)
96. edgeIndex = 0
97. for i in range(self.\_nodes):
98. for j in range(i, self.\_nodes):
99. if self.\_amatrix[i][j] != 0:
100. value = 1
101. if i == j:
102. value = 2
104. for k in range(self.\_amatrix[i][j]):
105. self.\_imatrix[i][edgeIndex] = self.\_imatrix[j][edgeIndex] = value
106. edgeIndex += 1
107. # Вывод матрицы инцидентности
108. def showIncidenceMatrix(self):
109. if self.\_imatrix is None:
110. self.makeIncidenceMatrix()
111. print("Матрица инцидентности: ")
112. for i in range(self.\_nodes \* 2 + 1):
113. for j in range(self.\_edges \* 2 + 1):
114. if j % 2 == 1:
115. print(end = "  ") # |
116. elif i % 2 == 1:
117. print(end = " ") # -
118. elif j == 0 and i != 0:
119. print(end = f"{chr(AINDEX + (i + j) // 2 - 1)}")
120. elif i == 0 and j != 0:
121. print(end = f"{j // 2}")
122. elif i // 2 > 0 and j // 2 > 0:
123. print(end = f"{self.\_imatrix[i // 2 - 1][j // 2 - 1]}")
124. if j >= 20:
125. print(end = " ")
126. else:
127. print(end = " ")
128. print()

131. def main():
132. menu = "Выберите режим работы программы (можно выбрать несколько):\n"
133. menu += "Enter - Обычный режим\n1 - Полный граф\n2 - Кратные ребра\n3 - Наличие петель (могут быть кратными)\nq - Выход\n"
134. mode = input(menu)
135. while ('q' not in mode):
136. NodeNumber = int(input("Введите количество вершин графа: "))
137. graph = Graph(NodeNumber, '1' in mode, '2' in mode, '3' in mode)
138. graph.setRandomMatrix()
139. graph.showAdjacencyMatrix()
140. print()
141. graph.showIncidenceMatrix()
142. graph.showGraph()
143. os.system("cls")
144. mode = input(menu)
145. os.system("cls")
147. if \_\_name\_\_ == "\_\_main\_\_":
148. main()

## 4. Заключение

В ходе выполнения лабораторной работы изучили матрицы смежности, инцидентности, графы на практике. Научились строить графы в python, используя библиотеку networkx, а также переводить матрицу смежности в матрицу инцидентности.

Затраченное время:

* 6 часов на написание конечного кода (с учетом рефакторинга);
* 1 час на отчет.

Итого: затрачено на работу 7 часов.

Больше всего времени заняла индексация ребер и вершин, и их правильный вывод в матрицах и особенно в графе, так как пришлось обращаться к документации библиотеки network, а также немного к numpy и matplotlib.