

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.

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 А в вершину В, но используя алгоритм Дейкстры мы можем найти все кратчайшие пути из вершины А 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.

```

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: 4
Матрица смежности:
  a  b  c  d
a  0  0  0  0
b  0  0  1  1
c  0  1  0  0
d  1  0  1  0

Матрица инцидентности:
  1  2  3  4  5
a  0  0  0  1  0
b -1 -1  1  0  0
c  1  0 -1  0  1
d  0  1  0 -1 -1

Матрица весов:
  a  b  c  d
a  0  inf  inf  inf
b  inf  0  4  2
c  inf  3  0  inf
d  5  inf  3  0

```

Рисунок 2.1 – Меню и матриц

```

Матрица Дейкстры из вершины a:
  a  b  c  d
a  0  inf inf inf
b  0  inf inf inf
c  0  inf inf inf
d  0  inf inf inf
Матрица Дейкстры из вершины b:
  a  b  c  d
a  inf 0  4  2
b  7  0  4  2
c  7  0  4  2
d  7  0  4  2
Матрица Дейкстры из вершины c:
  a  b  c  d
a  inf 3  0  inf
b  inf 3  0  5
c  10  3  0  5
d  10  3  0  5
Матрица Дейкстры из вершины d:
  a  b  c  d
a  5  inf 3  0
b  5  6  3  0
c  5  6  3  0
d  5  6  3  0

```

Рисунок 2.2 – Вывод всех расчетов и минимальных путей

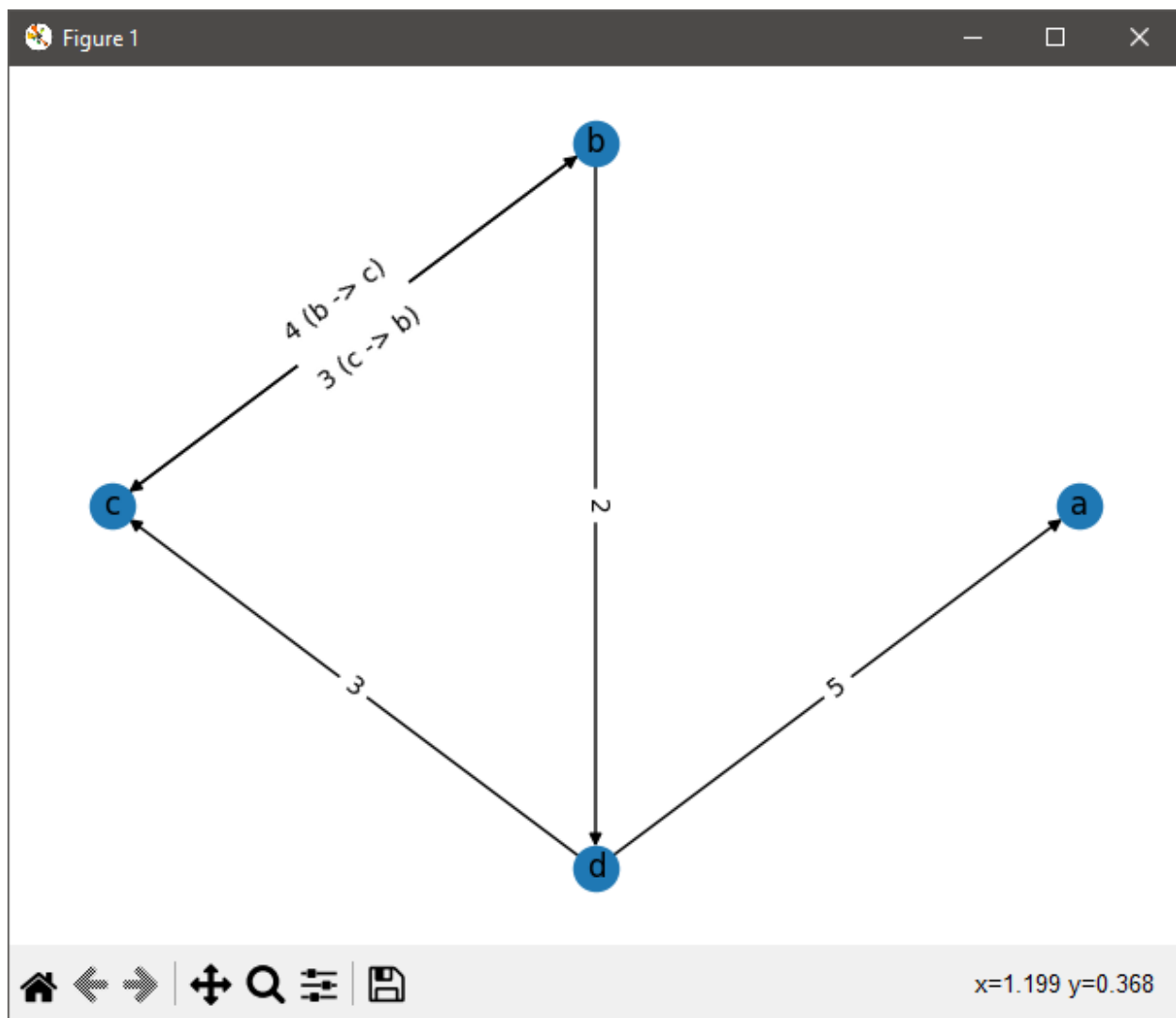


Рисунок 2.1 – Визуализация графа

3. Листинг

```
1: import networkx as nx
2: import matplotlib.pyplot as plt
3: import matplotlib as mpl
4: import numpy as np
5: import random
6: import math
7: import os
8: import copy
9:
10: AINDEX = 97
11:
12: # Создание матрицы размером (n x n)
13: def MakeMatrix(n):
14:     matrix = list()
15:     for i in range(n):
16:         matrix.append(list())
17:         for j in range(n):
18:             matrix[i].append(0)
19:     return matrix
20:
21: # Глубокое копирование матрицы
22: def CopyMatrix(matrix, n):
23:     newMatrix = MakeMatrix(n)
24:     for i in range(n):
25:         for j in range(n):
26:             newMatrix[i][j] = matrix[i][j]
27:     return newMatrix
28:
29: class Graph:
30:     # Конструктор класса
31:     def __init__(self, size, bFullGraph = False, bMultiedge = False, bLoop = False):
32:         self._nodes = size
33:         self._bFullGraph = bFullGraph
34:         self._bMultiedge = bMultiedge
35:         self._bLoop = bLoop
36:         self._imatix = None
37:         self._amatix = MakeMatrix(self._nodes)
38:
39:
40:     # Вывод матрицы смежности
41:     def showAdjacencyMatrix(self):
42:         print("Матрица смежности: ")
43:         for i in range(self._nodes * 2 + 1):
44:             for j in range(self._nodes * 2 + 1):
45:                 if j % 2 == 1:
46:                     print(end = " ") # |
47:                 elif i % 2 == 1:
48:                     print(end = " ") # -
49:                 elif (i == 0 and j != 0) or (j == 0 and i != 0):
50:                     print(end = f"{chr(AINDEX + (i + j) // 2 - 1)}")
51:                 elif i // 2 > 0 and j // 2 > 0:
52:                     print(end = f"{self._amatix[i // 2 - 1][j // 2 - 1]}")
53:                 else:
54:                     print(end = " ")
55:             print()
56:
57:     # Вывод графа (при помощи networkx)
58:     def showGraph(self):
59:         nodeMap = dict()
60:         edgeMap = dict()
61:         loopMap = dict()
62:
63:         for i in range(0, self._nodes):
64:             nodeMap.update({i: chr(AINDEX + i)})
65:
66:         count = 1
```

```

67:         for i in range(self._nodes):
68:             for j in range(self._nodes):
69:                 if self._weightMatrix[i][j] != 0 and self._weightMatrix[i][j] !=
math.inf:
70:                     edgeName = ''
71:                     if self._weightMatrix[j][i] != 0 and self._weightMatrix[j][i] !=
math.inf:
72:                         edgeName = f"{self._weightMatrix[j][i]} ({chr(j + AINDEX)} ->
{chr(i + AINDEX)})\n\n{self._weightMatrix[i][j]} ({chr(i + AINDEX)} -> {chr(j +
AINDEX)})"
73:                     else:
74:                         edgeName = f"{self._weightMatrix[i][j]}"
75:                         edgeMap.update({(chr(AINDEX + i), chr(AINDEX + j)): edgeName})
76:
77:         G = nx.DiGraph(np.array(self._amatrix))
78:         nx.relabel_nodes(G, nodeMap, False)
79:         pos = nx.circular_layout(G)
80:         nx.draw(G, pos, with_labels = True, arrows = True, arrowstyle = '-|>')
81:         nx.draw_networkx_edge_labels(G, pos, edge_labels = edgeMap)
82:         #nx.draw_networkx_edge_labels(G, pos, edge_labels = loopMap)
83:         plt.show()
84:
85:     # Заполнение таблицы смежности при помощи рандома
86:     def setRandomMatrix(self):
87:         deltaIndex = 0 if self._bLoop else 1
88:         minEdges = 1 if self._bFullGraph else 0
89:         maxEdges = 3 if self._bMultiedge else 1
90:
91:         for i in range(self._nodes):
92:             for j in range(self._nodes):
93:                 value = random.randint(1, maxEdges) if random.randint(minEdges, 1) == 1
else 0
94:                 self._amatrix[i][j] = value if (i != j or deltaIndex == 0) else 0
95:
96:     # Подсчет ребер графа
97:     def updateEdges(self):
98:         edges = 0
99:         for i in range(self._nodes):
100:             for j in range(self._nodes):
101:                 edges += self._amatrix[i][j]
102:         self._edges = edges
103:
104:     # Создание матрицы инцидентности
105:     def makeIncidenceMatrix(self):
106:         self.updateEdges()
107:
108:         self._imatrix = list()
109:         for i in range(self._nodes):
110:             self._imatrix.append(list())
111:             for j in range(self._edges):
112:                 self._imatrix[i].append(0)
113:
114:         edgeIndex = 0
115:         for i in range(self._nodes):
116:             for j in range(self._nodes):
117:                 if self._amatrix[i][j] != 0:
118:
119:                     for k in range(self._amatrix[i][j]):
120:                         self._imatrix[i][edgeIndex] = -1
121:                         self._imatrix[j][edgeIndex] = 1
122:                         edgeIndex += 1
123:
124:     # Вывод матрицы инцидентности
125:     def showIncidenceMatrix(self):
126:         if self._imatrix is None:
127:             self.makeIncidenceMatrix()
128:         print("Матрица инцидентности: ")
129:         for i in range(self._nodes * 2 + 1):

```



```

130:         for j in range(self._edges * 2 + 1):
131:             if j % 2 == 1:
132:                 print(end = " ") # |
133:             elif i % 2 == 1:
134:                 print(end = " ") # -
135:             elif j == 0 and i != 0:
136:                 print(end = f"{chr(AINDEX + (i + j) // 2 - 1)}")
137:             elif i == 0 and j != 0:
138:                 print(end = f"{j // 2}")
139:             elif i // 2 > 0 and j // 2 > 0:
140:                 if self._imatrix[i // 2 - 1][j // 2 - 1] < 0:
141:                     print(end = '\b')
142:                 print(end = f"{self._imatrix[i // 2 - 1][j // 2 - 1]}")
143:             if j >= 20:
144:                 print(end = " ")
145:             else:
146:                 print(end = " ")
147:         print()
148:
149:     def MakeWeightMatrix(self):
150:         self._weightMatrix = MakeMatrix(self._nodes)
151:         for i in range(self._nodes):
152:             for j in range(self._nodes):
153:                 if i != j:
154:                     if self._amatrix[i][j] == 0:
155:                         self._weightMatrix[i][j] = math.inf
156:                     else:
157:                         self._weightMatrix[i][j] = random.randint(1, 5)
158:
159:     # Вывод матрицы весов
160:     def showWeightMatrix(self):
161:         if self._weightMatrix == None:
162:             self.MakeWeightMatrix()
163:         print("Матрица весов: ")
164:         for i in range(self._nodes * 2 + 1):
165:             for j in range(self._nodes * 2 + 1):
166:                 if j % 2 == 1:
167:                     print(end = " ") # |
168:                 elif i % 2 == 1:
169:                     print(end = " ") # -
170:                 elif (i == 0 and j != 0) or (j == 0 and i != 0):
171:                     print(end = f"{chr(AINDEX + (i + j) // 2 - 1)}")
172:                 elif i // 2 > 0 and j // 2 > 0:
173:                     if self._weightMatrix[i // 2 - 1][j // 2 - 1] == math.inf:
174:                         print(end = "\b")
175:                     print(end = f"{self._weightMatrix[i // 2 - 1][j // 2 - 1]}")
176:                 else:
177:                     print(end = " ")
178:             print()
179:
180:     def DijkstraAlgorithm(self):
181:         self._dijkstraMatrix = list()
182:         for i in range(self._nodes):
183:             self._dijkstraMatrix.append(MakeMatrix(self._nodes))
184:
185:         for l in range(self._nodes):
186:             mask = ''
187:             for k in range(self._nodes):
188:                 mask += chr(AINDEX + k)
189:
190:             currentNode = l
191:             for i in range(self._nodes):
192:                 #print(currentNode)
193:                 if i == 0:
194:                     for j in range(self._nodes):
195:                         self._dijkstraMatrix[l][i][j] =
self._weightMatrix[currentNode][j]
196:                 else:

```

```

197:         for j in range(self._nodes):
198:             if self._dijkstraMatrix[l][i-1][j] > self._dijkstraMatrix[l][i
- 1][currentNode] + self._weightMatrix[currentNode][j]:
199:                 self._dijkstraMatrix[l][i][j] = self._dijkstraMatrix[l][i -
1][currentNode] + self._weightMatrix[currentNode][j]
200:             else:
201:                 self._dijkstraMatrix[l][i][j] = self._dijkstraMatrix[l][i-
1][j]
202:
203:         temp = ''
204:         for k in range(len(mask)):
205:             temp += mask[k] if mask[k] != chr(currentNode + AINDEX) else ''
206:         mask = temp
207:
208:         minimum = math.inf
209:         for j in range(self._nodes):
210:             if chr(AINDEX + j) in mask:
211:                 if minimum > self._dijkstraMatrix[l][i][j]:
212:                     minimum = self._dijkstraMatrix[l][i][j]
213:                     currentNode = j
214:
215:     def ShowDijkstra(self):
216:         for l in range(self._nodes):
217:             print(f"Матрица Дейкстры из вершины {chr(l + AINDEX)}: ")
218:             for i in range(self._nodes * 2 + 1):
219:                 for j in range(self._nodes * 2 + 1):
220:                     if j % 2 == 1:
221:                         print(end = " ") # |
222:                     elif i % 2 == 1:
223:                         print(end = " ") # -
224:                     elif (i == 0 and j != 0) or (j == 0 and i != 0):
225:                         print(end = f"{chr(AINDEX + (i + j) // 2 - 1)}")
226:                     elif i // 2 > 0 and j // 2 > 0:
227:                         if self._dijkstraMatrix[i // 2 - 1][j // 2 - 1] == math.inf:
228:                             print(end = "\b")
229:                         print(end = f"{self._dijkstraMatrix[l][i // 2 - 1][j // 2 -
1]}")
230:                     else:
231:                         print(end = " ")
232:                 print()
233:
234:
235:
236: def main():
237:     menu = "Выберите режим работы программы (можно выбрать несколько):\n"
238:     menu += "Enter - Обычный режим\n1 - Полный граф\n2 - Кратные ребра\n3 - Наличие
петель (могут быть кратными)\nq - Выход\n"
239:
240:     mode = input(menu)
241:
242:     while ('q' not in mode):
243:         NodeNumber = int(input("Введите количество вершин графа: "))
244:         graph = Graph(NodeNumber, '1' in mode, '2' in mode, '3' in mode)
245:         graph.setRandomMatrix()
246:         graph.showAdjacencyMatrix()
247:         print()
248:         graph.showIncidenceMatrix()
249:         graph.MakeWeightMatrix()
250:         graph.showWeightMatrix()
251:         graph.DijkstraAlgorithm()
252:         graph.ShowDijkstra()
253:         graph.showGraph()
254:         if os.system("clear"): os.system("cls")
255:         mode = input(menu)
256:         if os.system("clear"): os.system("cls")
257:
258: if __name__ == "__main__":
259:     main()

```

4. Заключение

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