МІНІСТЕРСТВО ОСВІТИ І НАУКИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»



ЛАБОРАТОРНА РОБОТА № 4

“Алгоритми і структури данних”

**Виконав:**

ст. гр. ІР-21

Патана Д. А.

Львів 2021

**Завдання**

Побудувати напрямлений та зважений граф та знайти середню відстань від заданої вершини S до інших вершин за допомогою алгоритму Дейкстри.

**Код**

**dijkstra.py**

import heapq

class Edge:

    def \_\_init\_\_(self, end, weight=0):

        if weight < 0:

            raise ValueError

        self.end = end

        self.weight = weight

class Vertice:

    def \_\_init\_\_(self, id, value=None):

        self.id = id

        self.value = value

        self.edges = {}

    def add\_edge(self, edge\_end, weight):

        self.edges[edge\_end.id] = Edge(edge\_end, weight)

    def \_\_lt\_\_(self, other):

        return self.id < other.id

class Graph:

    def \_\_init\_\_(self):

        self.vertices = {}

    def add\_vertice(self, vertice):

        self.vertices[vertice.id] = vertice

    def dijkstra\_algorithm(self, start\_vertice\_id):

        distances = {vertice\_id: float("inf")

                     for vertice\_id in self.vertices.keys()}

        distances[start\_vertice\_id] = 0

        visited = {vertice\_id: False for vertice\_id in self.vertices.keys()}

        visited[start\_vertice\_id] = True

        min\_heap = []

        heapq.heappush(min\_heap, (0, self.vertices[start\_vertice\_id]))

        while min\_heap:

            cur\_vertice = heapq.heappop(min\_heap)[1]

            for edge in cur\_vertice.edges.values():

                if distances[cur\_vertice.id] + edge.weight < distances[edge.end.id]:

                    distances[edge.end.id] = distances[cur\_vertice.id] + \

                        edge.weight

                if not visited[edge.end.id]:

                    visited[edge.end.id] = True

                    heapq.heappush(

                        min\_heap, (distances[edge.end.id], edge.end))

        return distances

if \_\_name\_\_ == "\_\_main\_\_":

    graph = Graph()

    input\_file = open("./input.txt", "r")

    number\_of\_edges, start\_vertice\_id = input\_file.readline().split()

    number\_of\_edges = int(number\_of\_edges)

    for \_ in range(number\_of\_edges):

        edge\_start, edge\_end, edge\_weight = input\_file.readline().split()

        edge\_weight = int(edge\_weight)

        if edge\_start not in graph.vertices.keys():

            graph.add\_vertice(Vertice(edge\_start))

        if edge\_end not in graph.vertices.keys():

            graph.add\_vertice(Vertice(edge\_end))

        graph.vertices[edge\_start].add\_edge(

            graph.vertices[edge\_end], edge\_weight)

    distances = [distance for distance in graph.dijkstra\_algorithm(

        start\_vertice\_id).values() if distance != float("inf")]

    print(sum(distances)/(len(distances)-1))

**test.py**

from dijkstra import Graph, Vertice

import unittest

class TestDijkstraAlgorithm(unittest.TestCase):

    def setUp(self):

        a = Vertice("A")

        b = Vertice("B")

        c = Vertice("C")

        d = Vertice("D")

        e = Vertice("E")

        f = Vertice("F")

        g = Vertice("G")

        h = Vertice("H")

        i = Vertice("I")

        a.add\_edge(b, 4)

        a.add\_edge(h, 8)

        b.add\_edge(a, 4)

        b.add\_edge(c, 8)

        b.add\_edge(h, 11)

        c.add\_edge(b, 8)

        c.add\_edge(d, 7)

        c.add\_edge(f, 4)

        c.add\_edge(i, 2)

        d.add\_edge(c, 7)

        d.add\_edge(e, 9)

        d.add\_edge(f, 14)

        e.add\_edge(d, 9)

        e.add\_edge(f, 10)

        f.add\_edge(c, 4)

        f.add\_edge(d, 14)

        f.add\_edge(e, 10)

        f.add\_edge(g, 2)

        g.add\_edge(f, 2)

        g.add\_edge(h, 1)

        g.add\_edge(i, 6)

        h.add\_edge(a, 8)

        h.add\_edge(b, 11)

        h.add\_edge(g, 1)

        h.add\_edge(i, 7)

        i.add\_edge(c, 2)

        i.add\_edge(g, 6)

        i.add\_edge(h, 7)

        vertices = [a, b, c, d, e, f, g, h, i]

        self.graph = Graph()

        for vertice in vertices:

            self.graph.add\_vertice(vertice)

    def test\_vertice\_a(self):

        projected\_result = {"A": 0, "B": 4, "C": 12,

                            "D": 19, "E": 21, "F": 11, "G": 9, "H": 8, "I": 14}

        self.assertDictEqual(

            self.graph.dijkstra\_algorithm("A"), projected\_result)

    def test\_vertice\_b(self):

        projected\_result = {"A": 4, "B": 0, "C": 8, "D": 15,

                            "E": 22, "F": 12, "G": 12, "H": 11, "I": 10}

        self.assertDictEqual(

            self.graph.dijkstra\_algorithm("B"), projected\_result)

    def test\_vertice\_c(self):

        projected\_result = {"A": 12, "B": 8, "C": 0,

                            "D": 7, "E": 14, "F": 4, "G": 6, "H": 7, "I": 2}

        self.assertDictEqual(

            self.graph.dijkstra\_algorithm("C"), projected\_result)

    def test\_vertice\_d(self):

        projected\_result = {"A": 19, "B": 15, "C": 7,

                            "D": 0, "E": 9, "F": 11, "G": 13, "H": 14, "I": 9}

        self.assertDictEqual(

            self.graph.dijkstra\_algorithm("D"), projected\_result)

    def test\_vertice\_e(self):

        projected\_result = {"A": 21, "B": 22, "C": 14,

                            "D": 9, "E": 0, "F": 10, "G": 12, "H": 13, "I": 16}

        self.assertDictEqual(

            self.graph.dijkstra\_algorithm("E"), projected\_result)

    def test\_vertice\_f(self):

        projected\_result = {"A": 11, "B": 12, "C": 4,

                            "D": 11, "E": 10, "F": 0, "G": 2, "H": 3, "I": 6}

        self.assertDictEqual(

            self.graph.dijkstra\_algorithm("F"), projected\_result)

    def test\_vertice\_g(self):

        projected\_result = {"A": 9, "B": 12, "C": 6,

                            "D": 13, "E": 12, "F": 2, "G": 0, "H": 1, "I": 6}

        self.assertDictEqual(

            self.graph.dijkstra\_algorithm("G"), projected\_result)

    def test\_vertice\_h(self):

        projected\_result = {"A": 8, "B": 11, "C": 7,

                            "D": 14, "E": 13, "F": 3, "G": 1, "H": 0, "I": 7}

        self.assertDictEqual(

            self.graph.dijkstra\_algorithm("H"), projected\_result)

    def test\_vertice\_i(self):

        projected\_result = {"A": 14, "B": 10, "C": 2,

                            "D": 9, "E": 16, "F": 6, "G": 6, "H": 7, "I": 0}

        self.assertDictEqual(

            self.graph.dijkstra\_algorithm("I"), projected\_result)

**input.txt**

    300 5

37 175 5

55 35 1

15 56 4

136 192 3

49 108 4

126 25 9

100 79 3

48 189 8

125 143 8

181 60 1

199 128 1

131 64 10

134 74 0

147 5 2

76 195 5

98 42 3

60 120 10

196 66 9

116 157 8

111 112 1

109 99 0

42 123 3

70 72 1

128 132 6

197 82 5

110 22 8

129 197 9

172 4 1

47 7 3

31 35 2

189 84 10

55 4 10

89 185 3

24 161 7

12 115 3

190 164 5

169 63 2

141 26 1

54 21 10

17 23 5

158 150 8

158 46 8

27 196 0

140 40 10

89 3 5

25 116 2

60 168 8

144 7 5

144 60 8

32 27 1

173 159 4

189 190 10

88 163 2

124 195 1

62 113 5

38 99 5

42 3 10

9 186 6

39 77 8

189 166 6

1 32 1

155 41 1

176 194 8

114 178 9

129 123 4

29 137 10

159 78 0

187 102 9

197 115 10

166 98 1

7 30 2

34 67 4

127 83 0

52 167 0

192 17 4

13 181 1

27 77 0

193 171 8

143 193 7

20 56 8

79 21 10

42 105 0

17 37 7

148 22 3

92 57 5

132 123 8

6 181 4

54 186 6

128 65 4

65 118 3

22 99 9

115 121 9

187 27 6

151 24 1

142 74 1

137 175 5

128 72 2

187 173 2

108 185 7

127 160 4

30 90 2

191 11 7

121 170 8

87 112 0

148 165 6

185 48 2

46 178 0

70 198 2

165 27 6

33 55 9

51 106 7

77 41 2

152 149 7

101 72 3

195 199 7

103 153 4

153 66 6

127 141 3

91 79 1

23 186 0

169 125 2

10 36 5

137 132 10

182 160 3

186 116 0

65 182 1

94 110 8

69 54 1

153 198 2

197 148 0

123 125 9

148 79 6

56 143 2

144 192 5

3 171 5

146 61 7

21 182 6

115 111 2

105 6 7

40 130 5

106 10 4

113 78 8

147 98 0

163 63 3

39 18 3

161 10 9

196 118 0

155 114 1

20 70 8

16 163 8

38 183 10

34 106 10

112 38 2

48 129 10

86 82 5

70 164 10

32 167 9

75 15 5

128 3 9

68 177 2

62 70 8

118 46 1

106 110 10

170 54 7

165 110 8

66 109 9

7 98 10

186 105 3

14 77 5

7 178 4

190 158 5

4 178 1

177 61 2

177 9 0

179 85 5

160 198 9

112 70 8

156 30 1

88 197 10

126 47 9

161 25 6

178 158 7

174 193 3

11 0 1

59 29 2

29 110 10

2 47 2

138 157 9

80 130 9

13 118 9

0 43 8

74 86 4

192 177 6

164 76 2

187 25 7

103 88 10

107 192 10

113 96 7

121 52 10

117 167 5

80 176 3

195 69 3

65 164 7

199 120 3

11 77 8

81 40 7

76 172 9

65 157 4

111 95 5

57 88 3

27 108 10

90 158 5

187 79 7

106 64 10

137 94 3

179 159 5

64 21 10

110 128 4

159 100 5

67 103 1

106 141 10

66 52 1

170 96 2

102 60 10

109 103 6

183 177 10

118 168 9

64 196 1

97 4 2

63 151 8

109 87 0

162 150 7

192 129 4

53 20 0

188 145 2

186 95 10

104 188 3

162 168 9

84 137 7

1 96 2

46 147 10

150 93 0

138 184 5

102 112 6

130 85 8

87 94 7

72 26 4

154 48 5

93 18 8

185 134 2

16 43 3

44 177 2

130 53 6

147 104 3

102 86 8

155 185 9

179 190 8

178 117 1

168 95 3

194 9 0

142 123 10

198 147 1

101 166 1

136 94 7

126 124 8

161 80 6

155 3 4

98 83 7

54 130 2

93 12 3

137 142 1

71 153 3

96 73 9

150 2 9

84 173 6

140 143 6

62 15 10

49 125 9

124 84 1

112 75 0

148 153 2

147 181 9

30 173 9

91 130 4

87 4 0

113 106 8

37 97 6

113 2 6

100 131 2

124 5 8

50 140 5

184 11 8

84 1 3

92 115 4

7 100 10

170 10 9

125 92 5

178 32 2

165 112 9

5 32 0

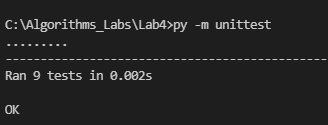
**Github**

<https://github.com/Danyaruko/Algorithms_Labs/pull/4>

**Результат виконання**

****

**Результат виконання тестів**

****

**Висновок**

Виконавши цю лабораторну роботу, я познайомився з графами та реалізацією алгоритму пошуку найкоротшого шляху у графі зі зваженими та напрямленими ребрами.