**DELHI TECHNOLOGICAL UNIVERSITY**

****

**PROJECT REPORT**

**Application to Navigate Through the Delhi Metro System**

**IT-205 Discrete Structures**

*By:*

*Akaash Nidhiss 2K19/IT/008*

*Anasuya Mithra 2K19/IT/018*

**DECLARATION**

I, (Anasuya Mithra), Roll No – 2K19/IT/018, and my Project Partner, (Akaash Nidhiss), Roll No - 2K19/IT/008, students of B.Tech. (INFORMATION TECHNOLOGY), hereby declare that the project Dissertation titled “Application to Navigate Through the Delhi Metro System” which is submitted by us to the Department of INFORMATION TECHNOLOGY, Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of the degree of Bachelor of Technology, is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any Degree, Diploma Associateship, Fellowship or other similar title or recognition.

Place: Delhi Anasuya Mithra & Akaash Nidhiss

Date: 01-12-2020

**ACKNOWLEDGEMENT**

We would like to convey our heartfelt thanks to our supervisor Mrs.Swati Sharda for her ingenious ideas, tremendous help and cooperation.

We are extremely grateful to our friends who gave valuable suggestions and guidance for completion of my project. The cooperation and healthy criticism came handy and useful with them.

Finally, we would like to thank the university for giving us the opportunity to create such an innovative project, that will help us in our future endeavours.

**INTRODUCTION**

The Delhi Metro system is used by millions of people everyday, connecting together an expanse as huge as Delhi and making it easy to navigate, for all people of every class. As a result the Delhi Metro network consists of Eleven colour-coded regular lines serving 285 stations with a total length of 389 kilometres, a mind boggling number of stations to traverse without getting lost in its enormity.

The proposed project is an effort to append convenience and flexibility to National/International tourists and everyday Delhiites by providing essential information related to the Delhi Metro System, the cheapest and easiest mode of transport around Delhi. It has been designed with a very user centric approach, with many functionalities that keep user-friendliness in mind.

***FUNCTIONALITIES:***

* Allows the user to enter the metro station they begin their journey from and the station they wish to arrive at. The application will output the shortest route tobe taken to reach the destination within the shortest time frame possible.
* Shows all the stations the user has to pass through, including stations where they have to switch between metro lines at. The colours of each line are also shown in the GUI to increase user-friendliness.
* Displays the total time and cost that would be required for the journey.
* A route map of the Delhi Metro system can be accessed on the clicking of the ‘route map’ tab. This tab will display all the lines on the delhi metro system, highlighting the route that the user has to take for their journey. This map is geocoded, and hence the positioning of the stations are a small scale replica of the actual delhi metro system.
* A map of the Delhi Metro system as a whole can be accessed on another page, to show all the 11 lines and 285 stations in the system, for the user’s convenience to decide their journey.
* A map comparing the top five shortest routes to the route suggested as the shortest path by Dijkstra’s Algorithm is also displayed on a separate page.

***PROGRAMMING LANGUAGE AND LIBRARIES USED:***

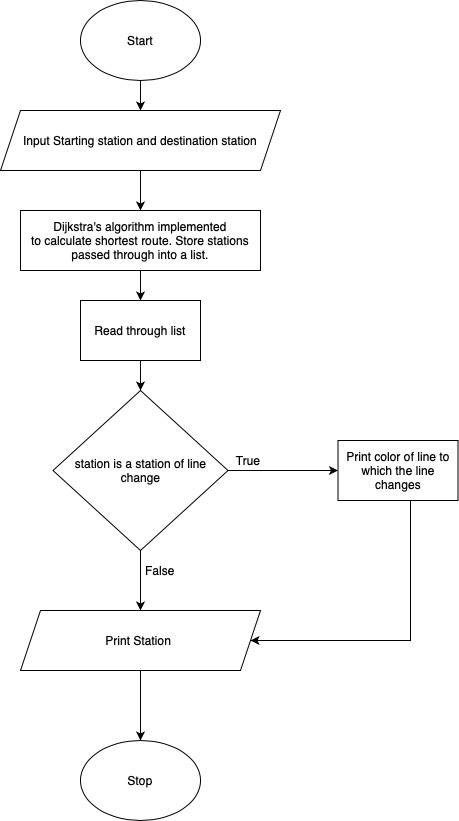
Python Programming language was used to implement this project. The libraries used are as follows:

* PyQt5 for GUI Implementation
* Matplotlib and NetworkX for Route Map and Map Implementation

***DISCRETE STRUCTURES USED:***

* Graphs
* Dijkstra’s Algorithm for Shortest path

**FLOWCHART**



**CODE SNIPPETS**

***Back.py***

Contains the backend working of the project. Dijkstra’s Algorithm is implemented here to calculate the shortest path between two stations, calculate the total time for the journey, the number of stations, station changes and cost.

*CODE:*

* Graph Class:

class Graph(object):

def \_\_init\_\_(self):

self.nodes = {}

self.edges = {}

self.distances = {}

self.key\_to\_name = {}

self.path = []

self.path\_time = 0

self.path\_len = 0

self.path\_changes = 0

self.all\_paths = {}

self.all\_paths\_wts = {}

def add\_vertex(self, value, name):

ntx\_graph.add\_node(value)

self.nodes[value] = name

self.key\_to\_name[name] = value

def add\_edge(self, from\_node, to\_node, distance):

self.\_add\_edge(from\_node, to\_node, distance)

self.\_add\_edge(to\_node, from\_node, distance)

def \_add\_edge(self, from\_node, to\_node, distance):

ntx\_graph.add\_edge(from\_node, to\_node, weight=distance)

self.edges.setdefault(from\_node, [])

self.edges[from\_node].append(to\_node)

self.distances[(from\_node, to\_node)] = distance

* Implementing Dijkstra’s algorithm and obtaining route, utilizing this function

''' djikstra function definition '''

def dijkstra(graph, initial\_node):

visited = {initial\_node: 0}

current\_node = initial\_node

path = {}

unvisited = set(graph.nodes)

while unvisited:

min\_node = None

for node in unvisited:

if node in visited:

if min\_node is None:

min\_node = node

elif visited[node] < visited[min\_node]:

min\_node = node

if min\_node is None:

break

unvisited.remove(min\_node)

cur\_wt = visited[min\_node]

for edge in graph.edges[min\_node]:

wt = cur\_wt + graph.distances[(min\_node, edge)]

if edge not in visited or wt < visited[edge]:

visited[edge] = wt

path[edge] = min\_node

return visited, path

''' recursive function to get route '''

def route(graph, x, y):

distances, paths = dijkstra(graph, x)

route = [y]

while y != x:

route.append(paths[y])

y = paths[y]

route.reverse()

return route

* Printing Route

def print\_route(graph, x, y):

r = route(graph, x, y)

g.path\_len = len(r)

u = int(x)

line\_changes = {}

key\_to\_line\_changes = {}

i = 0

prmpt = list()

for y in r[1:]:

if x >=0 and x <= 27:

if x == 11 and (y == 98 or y == 96):

c = 3

key\_to\_line\_changes[i] = c

elif x == 14 and (y == 40 or y == 38):

c = 2

key\_to\_line\_changes[i] = c

elif x == 14 and (y == 204):

c = 8

key\_to\_line\_changes[i] = c

elif x == 19 and (y == 169):

c = 6

key\_to\_line\_changes[i] = c

elif x == 22 and (y == 67 or y == 69):

c = 3

key\_to\_line\_changes[i] = c

else:

c = 1

key\_to\_line\_changes[i] = c

elif x >= 28 and x <= 64:

if x == 33 and (y == 67 or y == 65):

c = 3

key\_to\_line\_changes[i] = c

elif x == 42 and (y == 163):

c = 5

key\_to\_line\_changes[i] = c

elif x == 49 and (y == 81 or y == 79):

c = 3

key\_to\_line\_changes[i] = c

elif x == 43 and (y == 131 or y == 133):

c = 4

key\_to\_line\_changes[i] = c

elif x == 45 and (y == 209 or y == 211):

c = 8

key\_to\_line\_changes[i] = c

elif x == 52 and (y == 247 or y == 249):

c = 9

key\_to\_line\_changes[i] = c

elif x == 61 and (y == 196 or y == 198):

c = 7

key\_to\_line\_changes[i] = c

else:

c = 2

key\_to\_line\_changes[i] = c

.

.

.

// laying out colours for initial nodes is done for the rest of the possible nodes

else:

print("error in laying out colours for initial node")

line\_changes.setdefault(c,[])

d = graph.distances.get((x, y))

g.path\_time = g.path\_time + d

if g.nodes[x] is not 'blank':

prmpt.append('{}'.format(g.nodes[x]))

try:

if u >=0 and u <=27:

if x == 11 and (y == 98 or y == 96):

print("\nLine change: RED TO PINK\n")

u = 98

c = 3

placeholder = 0

key\_to\_line\_changes[i] = c

g.path\_changes = g.path\_changes + 1

if x == 14 and (y == 40 or y == 38):

print("\nLine change: RED TO YELLOW\n")

u = 40

c = 2

placeholder = 0

key\_to\_line\_changes[i] = c

g.path\_changes = g.path\_changes + 1

if x == 14 and (y == 204):

print("\nLine change: RED TO VIOLET\n")

u = 204

c = 8

placeholder = 0

key\_to\_line\_changes[i] = c

g.path\_changes = g.path\_changes + 1

if x == 19 and (y == 169):

print("\nLine change: RED TO GREEN\n")

u = 169

c = 6

placeholder = 0

key\_to\_line\_changes[i] = c

g.path\_changes = g.path\_changes + 1

if x == 22 and (y == 67 or y == 69):

print("\nLine change: RED TO PINK\n")

u = 69

c = 3

placeholder = 0

key\_to\_line\_changes[i] = c

g.path\_changes = g.path\_changes + 1

elif u >=28 and u <=64:

if x == 14 and (y == 13 or y == 15):

print("\nLine change: YELLOW TO RED\n")

u = 15

c = 1

placeholder = 0

key\_to\_line\_changes[i] = c

g.path\_changes = g.path\_changes + 1

if x == 33 and (y == 67 or y == 65):

print("\nLine change: YELLOW TO PINK\n")

u = 67

c = 3

placeholder = 0

key\_to\_line\_changes[i] = c

g.path\_changes = g.path\_changes + 1

if x == 14 and (y == 204):

print("\nLine change: YELLOW TO VIOLET\n")

u = 204

c = 8

placeholder = 0

key\_to\_line\_changes[i] = c

g.path\_changes = g.path\_changes + 1

if x == 42 and (y == 163):

print("\nLine change: YELLOW TO ORANGE\n")

u = 163

c = 5

placeholder = 0

key\_to\_line\_changes[i] = c

g.path\_changes = g.path\_changes + 1

.

.

// calculating line changes for the rest of the 11 metro lines is done

i = i + 1

key\_to\_line\_changes[i] = c

except:

print("Error in print function in back.py")

x = y

prmpt.append('{}'.format(g.nodes[y]))

g.path.extend(prmpt) #*adds prompt to the path list*

return prmpt, key\_to\_line\_changes

***MainFunctions.py***

Functions for the GUI are implemented here, including dynamically assigning labels to show the list of stations and creating the route map, comparing the shortest route with other shortest routes, etc.

*CODE:*

* Dynamically assigning labels to the stations

def printPathsLabels\_staions(self,start\_key,dest\_key):

''' main function to dynamically allocate labels and assign colours to them '''

global i

key\_to\_line\_changes = {}

station\_name = list()

station\_name, key\_to\_line\_changes = print\_route(g, start\_key, dest\_key)

self.ui.scrollArea\_2.setAlignment(Qt.AlignCenter)

for i in range(len(station\_name)):

station = QtWidgets.QLabel(self.ui.scrollArea\_2)

self.ui.verticalLayout\_15.addWidget(station)

station.setMinimumSize(QtCore.QSize(230, 40))

station.setMaximumSize(QtCore.QSize(230, 40))

station.setAlignment(Qt.AlignCenter)

try:

station.setText(station\_name[i])

if key\_to\_line\_changes[i] == 1:

station.setStyleSheet("QLabel{ margin-left: 10px; border-radius: 20px; border: 1px solid rgb(217,11,52); background: rgb(35,35,35); color: rgb(217,11,52);} ")

if key\_to\_line\_changes[i] == 2:

station.setStyleSheet("QLabel{margin-left: 10px; border-radius: 20px;border: 1px solid rgb(217,176,11); background: rgb(35,35,35);color: rgb(217,176,11);}")

if key\_to\_line\_changes[i] == 3:

station.setStyleSheet("QLabel{margin-left: 10px; border-radius: 20px;border: 1px solid rgb(255,0,101); background: rgb(35,35,35);color: rgb(255,0,101);}}")

if key\_to\_line\_changes[i] == 4:

station.setStyleSheet("QLabel{ margin-left: 10px; border-radius: 20px; border: 1px solid rgb(73,11,217); background: rgb(35,35,35); color: rgb(73,11,217);} ")

if key\_to\_line\_changes[i] == 5:

station.setStyleSheet("QLabel{margin-left: 10px; border-radius: 20px;border: 1px solid rgb(217,73,11); background: rgb(35,35,35);color: rgb(217,73,11);}")

if key\_to\_line\_changes[i] == 6:

station.setStyleSheet("QLabel{margin-left: 10px; border-radius: 20px;border: 1px solid rgb(52,217,11); background: rgb(35,35,35);color: rgb(52,217,11);}")

if key\_to\_line\_changes[i] == 7:

station.setStyleSheet("QLabel{ margin-left: 10px; border-radius: 20px; border: 1px solid rgb(255,105,97); background: rgb(35,35,35); color: rgb(255,105,97);} ")

if key\_to\_line\_changes[i] == 8:

station.setStyleSheet("QLabel{margin-left: 10px; border-radius: 20px;border: 1px solid rgb(176,11,217); background: rgb(35,35,35);color: rgb(176,11,217);}")

if key\_to\_line\_changes[i] == 9:

station.setStyleSheet("QLabel{margin-left: 10px; border-radius: 20px;border: 1px solid rgb(217,11,155); background: rgb(35,35,35);color: rgb(217,11,155);}")

if key\_to\_line\_changes[i] == 10:

station.setStyleSheet("QLabel{ margin-left: 10px; border-radius: 20px; border: 1px solid rgb(11,217,176); background: rgb(35,35,35); color: rgb(11,217,176);} ")

if key\_to\_line\_changes[i] == 11:

station.setStyleSheet("QLabel{margin-left: 10px; border-radius: 20px;border: 1px solid rgb(166,166,166); background: rgb(35,35,35);color: rgb(166,166,166);}")

except:

print("key to line change dict error")

* Setting the Trip Information

def setInfo(self):

''' setting trip info '''

paths\_time = int(g.path\_time)/2

strTime = "<strong>{}</strong> mins to reach".format(int(paths\_time))

self.ui.label\_time.setText(strTime)

paths\_line = int(g.path\_len) - 1

strLen = "<strong>{}</strong> stops".format(int(paths\_line))

font = QtGui.QFont()

font.setFamily("Segoe UI")

font.setPointSize(20)

font.setWeight(75)

self.ui.label\_stops.setFont(font)

self.ui.label\_stops.setText(strLen)

paths\_changes = int(g.path\_changes)/2

strChanges = "<strong>{}</strong> changes".format(int(paths\_changes))

self.ui.label\_changes.setText(strChanges)

if paths\_time >= 0 and paths\_time < 5:

self.ui.label\_cost.setFont(font)

self.ui.label\_cost.setText("Rs. <strong>10</strong>")

elif paths\_time >=5 and paths\_time < 11:

self.ui.label\_cost.setFont(font)

self.ui.label\_cost.setText("Rs. <strong>20</strong>")

elif paths\_time >=11 and paths\_time < 25:

self.ui.label\_cost.setFont(font)

self.ui.label\_cost.setText("Rs. <strong>30</strong>")

elif paths\_time >=25 and paths\_time <43:

self.ui.label\_cost.setFont(font)

self.ui.label\_cost.setText("Rs. <strong>40</strong>")

elif paths\_time >=43 and paths\_time <64:

self.ui.label\_cost.setFont(font)

self.ui.label\_cost.setText("Rs. <strong>50</strong>")

else:

self.ui.label\_cost.setFont(font)

self.ui.label\_cost.setText("Rs. <strong>60</strong>")

* Plotting Route Map

def plot(self,start\_key,dest\_key):

''' plot the route map '''

plt.style.use('fivethirtyeight')

G = nx.read\_graphml("coordinates.graphml")

names, key\_to\_line\_changes = print\_route(g, start\_key, dest\_key)

edge\_x = []

edge\_y = []

edge\_path\_x = []

edge\_path\_y = []

node\_path\_x = []

node\_path\_y = []

.

.

// all other necessary lists are intialised as seen above

values = list()

for name in names:

values.append(g.key\_to\_name[name])

self.figure.clear()

# *create an axis*

ax = self.figure.add\_subplot(111)

for edge in G.edges():

if int(edge[0]) in values and int(edge[1]) in values:

x0 = G.nodes[edge[0]]['x']

y0 = G.nodes[edge[0]]['y']

x1 = G.nodes[edge[1]]['x']

y1 = G.nodes[edge[1]]['y']

edge\_path\_x.append(x0)

edge\_path\_x.append(x1)

edge\_path\_x.append(None)

edge\_path\_y.append(y0)

edge\_path\_y.append(y1)

edge\_path\_y.append(None)

if int(edge[1]) >= 0 and int(edge[1]) <= 27:

ax.plot(edge\_path\_x,edge\_path\_y, "#ff0130", lw=2)

edge\_path\_x = []

edge\_path\_y = []

elif int(edge[1]) >= 28 and int(edge[1]) <= 64:

ax.plot(edge\_path\_x,edge\_path\_y, "#f9c70c", lw=2)

edge\_path\_x = []

edge\_path\_y = []

elif int(edge[1]) >= 65 and int(edge[1]) <= 102:

ax.plot(edge\_path\_x,edge\_path\_y, "#ff3ba7", lw=2)

edge\_path\_x = []

edge\_path\_y = []

elif int(edge[1]) >= 103 and int(edge[1]) <= 153:

ax.plot(edge\_path\_x,edge\_path\_y, "#001fd7", lw=2)

edge\_path\_x = []

edge\_path\_y = []

elif int(edge[1]) >= 154 and int(edge[1]) <= 161:

ax.plot(edge\_path\_x,edge\_path\_y, "#001fd7", lw=2)

edge\_path\_x = []

edge\_path\_y = []

elif int(edge[1]) >= 162 and int(edge[1]) <= 167:

ax.plot(edge\_path\_x,edge\_path\_y, "#ff7e1c", lw=2)

edge\_path\_x = []

edge\_path\_y = []

elif int(edge[1]) >= 168 and int(edge[1]) <= 188:

ax.plot(edge\_path\_x,edge\_path\_y, "#8fff2d", lw=2)

edge\_path\_x = []

edge\_path\_y = []

elif int(edge[1]) >= 189 and int(edge[1]) <= 191:

ax.plot(edge\_path\_x,edge\_path\_y, "#8fff2d", lw=2)

edge\_path\_x = []

edge\_path\_y = []

elif int(edge[1]) >= 192 and int(edge[1]) <= 202:

ax.plot(edge\_path\_x,edge\_path\_y, "black", lw=2)

edge\_path\_x = []

edge\_path\_y = []

elif int(edge[1]) >= 203 and int(edge[1]) <= 236:

ax.plot(edge\_path\_x,edge\_path\_y, "#9c05d7", lw=2)

edge\_path\_x = []

edge\_path\_y = []

elif int(edge[1]) >= 237 and int(edge[1]) <= 261:

ax.plot(edge\_path\_x,edge\_path\_y, "#cf1f6e", lw=2)

edge\_path\_x = []

edge\_path\_y = []

elif int(edge[1]) >= 262 and int(edge[1]) <= 282:

ax.plot(edge\_path\_x,edge\_path\_y, "#17fffb", lw=2)

edge\_path\_x = []

edge\_path\_y = []

elif int(edge[1]) >= 283 and int(edge[1]) <= 285:

ax.plot(edge\_path\_x,edge\_path\_y, "grey", lw=2)

edge\_path\_x = []

edge\_path\_y = []

else:

x0 = G.nodes[edge[0]]['x']

y0 = G.nodes[edge[0]]['y']

x1 = G.nodes[edge[1]]['x']

y1 = G.nodes[edge[1]]['y']

edge\_x.append(x0)

edge\_x.append(x1)

edge\_x.append(None)

edge\_y.append(y0)

edge\_y.append(y1)

edge\_y.append(None)

for node in G.nodes():

if int(node) in values:

x = G.nodes[node]['x']

y = G.nodes[node]['y']

if int(node) >= 0 and int(node) <= 27:

ax.scatter(x, y, s=30,c = "#ff0130")

elif int(node) >= 28 and int(node) <= 64:

ax.scatter(x, y, s=30,c = "#f9c70c")

elif int(node) >= 65 and int(node) <= 102:

ax.scatter(x, y, s=30,c = "#ff3ba7")

elif int(node) >= 103 and int(node) <= 153:

ax.scatter(x, y, s=30,c = "#001fd7")

elif int(node) >= 154 and int(node) <= 161:

ax.scatter(x, y, s=30,c = "#001fd7")

elif int(node) >= 162 and int(node) <= 167:

ax.scatter(x, y, s=30,c = "#ff7e1c")

elif int(node) >= 168 and int(node) <= 188:

ax.scatter(x, y, s=30,c = "#8fff2d")

elif int(node) >= 189 and int(node) <= 191:

ax.scatter(x, y, s=30,c = "#8fff2d")

elif int(node) >= 192 and int(node) <= 202:

ax.scatter(x, y, s=30,c = "black")

elif int(node) >= 203 and int(node) <= 236:

ax.scatter(x, y, s=30,c = "#9c05d7")

elif int(node) >= 237 and int(node) <= 261:

ax.scatter(x, y, s=30,c = "#cf1f6e")

elif int(node) >= 262 and int(node) <= 282:

ax.scatter(x, y, s=30,c = "#17fffb")

elif int(node) >= 283 and int(node) <= 285:

ax.scatter(x, y, s=30,c = "grey")

else:

print("error")

ax.text(x,y,G.nodes[node]['label'],fontsize=6)

elif int(node) >= 0 and int(node) <= 27:

x = G.nodes[node]['x']

y = G.nodes[node]['y']

node\_red\_x.append(x)

node\_red\_y.append(y)

elif int(node) >= 28 and int(node) <= 64:

x = G.nodes[node]['x']

y = G.nodes[node]['y']

node\_yellow\_x.append(x)

node\_yellow\_y.append(y)

elif int(node) >= 65 and int(node) <= 102:

x = G.nodes[node]['x']

y = G.nodes[node]['y']

node\_pink\_x.append(x)

node\_pink\_y.append(y)

.

.

// x and y coordinates of the rest of the 11 lines and nodes are added similarly

# *plot data*

ax.scatter(node\_red\_x, node\_red\_y, s=10,c = "#ff0130", alpha=0.3)

ax.scatter(node\_yellow\_x, node\_yellow\_y, s=10,c = "#f9c70c", alpha=0.6)

ax.scatter(node\_pink\_x, node\_pink\_y, s=10,c = "#ff3ba7", alpha=0.6)

ax.scatter(node\_blue\_x, node\_blue\_y, s=10,c = "#001fd7", alpha=0.3)

ax.scatter(node\_orange\_x, node\_orange\_y, s=10,c = "#ff7e1c", alpha=0.6)

ax.scatter(node\_green\_x, node\_green\_y, s=10,c = "#8fff2d", alpha=0.3)

ax.scatter(node\_rapid\_x, node\_rapid\_y, s=10,c = "black", alpha=0.3)

ax.scatter(node\_violet\_x, node\_violet\_y, s=10,c = "#9c05d7", alpha=0.3)

ax.scatter(node\_magenta\_x, node\_magenta\_y, s=10,c = "#cf1f6e", alpha=0.3)

ax.scatter(node\_aqua\_x, node\_aqua\_y, s=10,c = "#17fffb", alpha=0.3)

ax.scatter(node\_grey\_x, node\_grey\_y, s=10,c = "grey", alpha=0.3)

ax.plot(edge\_x,edge\_y,alpha=0.3)

red\_patch = mpatches.Patch(color='#ff0130', label='Red line')

yellow\_patch = mpatches.Patch(color='#f9c70c', label='Yellow line')

pink\_patch = mpatches.Patch(color='#ff3ba7', label='Pink line')

blue\_patch = mpatches.Patch(color='#001fd7', label='Blue line')

orange\_patch = mpatches.Patch(color='#ff7e1c', label='Orange line')

green\_patch = mpatches.Patch(color='#8fff2d', label='Green line')

rapid\_patch = mpatches.Patch(color='black', label='Rapid line')

violet\_patch = mpatches.Patch(color='#9c05d7', label='Violet line')

magenta\_patch = mpatches.Patch(color='#cf1f6e', label='Magenta line')

cyan\_patch = mpatches.Patch(color='#17fffb', label='Aqua line')

grey\_patch = mpatches.Patch(color='grey', label='Grey line')

ax.legend(handles=[red\_patch,yellow\_patch,pink\_patch,blue\_patch,orange\_patch,green\_patch,rapid\_patch,violet\_patch,magenta\_patch,cyan\_patch,grey\_patch],prop={'size': 7},loc='upper right', borderaxespad=0.)

ax.axes.get\_xaxis().set\_visible(False)

ax.axes.get\_yaxis().set\_visible(False)

ax.set\_facecolor('#dfffff')

# *refresh canvas*

self.canvas.draw()

* Plotting the map to compare shortest path with 5 other paths

def plot\_compare\_graph(self,start\_key,dest\_key): #*?*

''' plot the compare graph on page 3 '''

#*plt.style.use('Solarize\_Light2')*

plt.style.use('fivethirtyeight')

G = nx.read\_graphml("coordinates.graphml")

names, key\_to\_line\_changes = print\_route(g, start\_key, dest\_key)

edge\_x = []

edge\_y = []

edge\_path\_x = []

edge\_path\_y = []

.

.

// all other necessary lists are intialised as seen above

values = list()

ind = 0

for name in names:

values.append(g.key\_to\_name[name])

x = extraFunctions()

for path in nx.shortest\_simple\_paths(ntx\_graph, source=start\_key, target=dest\_key, weight='weight'):

if ind<7:

wts = x.all\_path\_length(ntx\_graph,path,'weight')

g.all\_paths[ind] = path

g.all\_paths\_wts[ind] = wts

ind = ind + 1

else:

break

for edge in G.edges():

if int(edge[0]) in values and int(edge[1]) in values:

x0 = G.nodes[edge[0]]['x']

y0 = G.nodes[edge[0]]['y']

x1 = G.nodes[edge[1]]['x']

y1 = G.nodes[edge[1]]['y']

edge\_path\_x.append(x0)

edge\_path\_x.append(x1)

edge\_path\_x.append(None)

edge\_path\_y.append(y0)

edge\_path\_y.append(y1)

edge\_path\_y.append(None)

elif int(edge[0]) in g.all\_paths[1] and int(edge[1]) in g.all\_paths[1]:

x0 = G.nodes[edge[0]]['x']

y0 = G.nodes[edge[0]]['y']

x1 = G.nodes[edge[1]]['x']

y1 = G.nodes[edge[1]]['y']

edge\_one\_x.append(x0)

edge\_one\_x.append(x1)

edge\_one\_x.append(None)

edge\_one\_y.append(y0)

edge\_one\_y.append(y1)

edge\_one\_y.append(None)

elif int(edge[0]) in g.all\_paths[2] and int(edge[1]) in g.all\_paths[2]:

x0 = G.nodes[edge[0]]['x']

y0 = G.nodes[edge[0]]['y']

x1 = G.nodes[edge[1]]['x']

y1 = G.nodes[edge[1]]['y']

edge\_two\_x.append(x0)

edge\_two\_x.append(x1)

edge\_two\_x.append(None)

edge\_two\_y.append(y0)

edge\_two\_y.append(y1)

edge\_two\_y.append(None)

.

.

// similarly values for edge\_three, edge\_four and edge\_five are appended

else:

x0 = G.nodes[edge[0]]['x']

y0 = G.nodes[edge[0]]['y']

x1 = G.nodes[edge[1]]['x']

y1 = G.nodes[edge[1]]['y']

edge\_x.append(x0)

edge\_x.append(x1)

edge\_x.append(None)

edge\_y.append(y0)

edge\_y.append(y1)

edge\_y.append(None)

for node in G.nodes():

if int(node) >= 0 and int(node) <= 27:

x = G.nodes[node]['x']

y = G.nodes[node]['y']

node\_red\_x.append(x)

node\_red\_y.append(y)

elif int(node) >= 28 and int(node) <= 64:

x = G.nodes[node]['x']

y = G.nodes[node]['y']

node\_yellow\_x.append(x)

node\_yellow\_y.append(y)

elif int(node) >= 65 and int(node) <= 102:

x = G.nodes[node]['x']

y = G.nodes[node]['y']

node\_pink\_x.append(x)

node\_pink\_y.append(y)

elif int(node) >= 103 and int(node) <= 153:

x = G.nodes[node]['x']

y = G.nodes[node]['y']

node\_blue\_x.append(x)

node\_blue\_y.append(y)

.

.

// x and y coordinates of the rest of the 11 lines and nodes are added similarly

else:

print("Error in printing nodes graph")

self.figure\_compare.clear()

# *create an axis*

ax\_compare = self.figure\_compare.add\_subplot(111)

# *plot data*

ax\_compare.plot(edge\_x,edge\_y,alpha=0.3)

.

.

// graph is plotted similarly to the route map from the previous function

dijkstra\_legend = "{} mins for Djikstra Path".format(g.all\_paths\_wts[0])

path1\_legend = "{} mins for Path 1".format(g.all\_paths\_wts[1])

path2\_legend = "{} mins for Path 2".format(g.all\_paths\_wts[2])

path3\_legend = "{} mins for Path 3".format(g.all\_paths\_wts[3])

path4\_legend = "{} mins for Path 4".format(g.all\_paths\_wts[4])

path5\_legend = "{} mins for Path 5".format(g.all\_paths\_wts[5])

dijkstra\_patch = mpatches.Patch(color='#ff0130', label=dijkstra\_legend)

path1\_patch = mpatches.Patch(color='black', label=path1\_legend)

path2\_patch = mpatches.Patch(color='yellow', label=path2\_legend)

path3\_patch = mpatches.Patch(color='#8fff2d', label=path3\_legend)

path4\_patch = mpatches.Patch(color='#ff7e1c', label=path4\_legend)

path5\_patch = mpatches.Patch(color='#9c05d7', label=path5\_legend)

ax\_compare.legend(handles=[dijkstra\_patch,path1\_patch,path2\_patch,path3\_patch,path4\_patch,path5\_patch],prop={'size': 7},loc='upper right', borderaxespad=0.)

ax\_compare.axes.get\_xaxis().set\_visible(False)

ax\_compare.axes.get\_yaxis().set\_visible(False)

ax\_compare.set\_facecolor('#dfffff')

ax\_compare.set\_title('Comparing Djikstra Path with Top 5 Shortest Other Paths' , color = '#dfffff')

# *refresh canvas*

self.canvas\_compare.draw()

class extraFunctions:

def all\_path\_length(self,G, nodes, weight):

w = 0

for ind,nd in enumerate(nodes[1:]):

prev = nodes[ind]

w += G[prev][nd][weight]

return w

**RESULT**

