**GPS SHORTEST DISTANCE ROUTE**

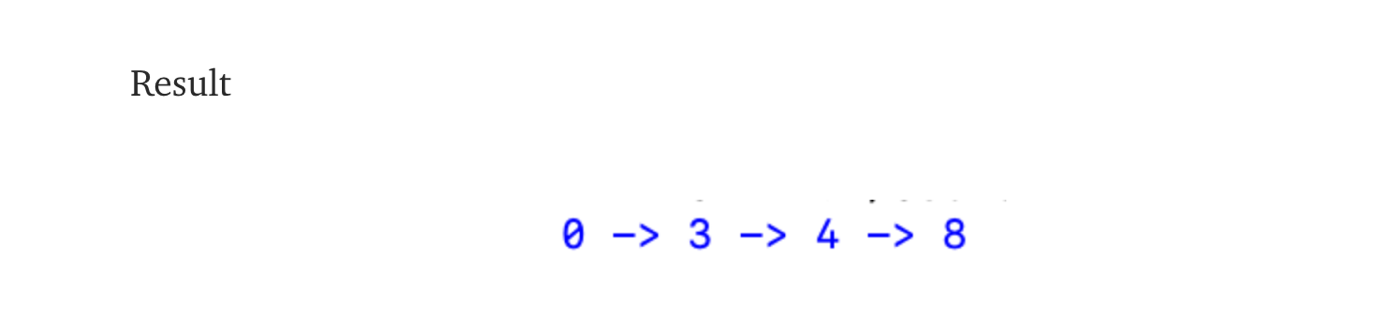
**by PYTHON**

***CODE:***

|  |
| --- |
|  |
|  |
| from queue import PriorityQueue |
|  |
| class DijkstraSP: |
|  |
| edgeTo=[] |
| distTo=[] |
| pq = PriorityQueue() # put the weight in first element and vertex in the second element |
|  |
| def \_\_init\_\_(self,G,root): |
| self.root=root |
| self.edgeTo=[] |
| self.distTo=[] |
| self.pq = PriorityQueue() # put the weight in first element and vertex in the second element |
|  |
| self.edgeTo=list(range(0,G.V)) # default all edges to itself |
| self.distTo=[float('inf')] \* G.V # default all value in distTo to be inf |
|  |
| self.distTo[root] = 0 # started vertex |
|  |
| self.pq.put((0,root)) |
|  |
| while(self.pq.empty() is False): |
|  |
| v = self.pq.get()[1] # get next vertex |
| adjacent = G.adj[v] # get all edges start from v |
| for e in adjacent: |
| self.relax(e) |
|  |
|  |
| def relax(self,e): |
| v = e.From() |
| w = e.To() |
|  |
| if(self.distTo[w] > self.distTo[v] + e.Weight()): |
| self.distTo[w] = self.distTo[v]+e.Weight() |
| self.edgeTo[w] = v |
| self.pq.put((self.distTo[w],w)) |
|  |
| def shortestpath(self,destination): # print out the shortest path |
| path=[] |
| path.append(destination) |
|  |
| while(destination != self.edgeTo[destination]): |
|  |
| destination=self.edgeTo[destination] |
| path.append(destination) |
|  |
| if(path[-1]!=self.root): |
| print("There is no connection between "+ str(self.root )+" and "+str(destination)) |
| else: |
| res="" |
| for i in range(0,len(path)): |
| if(i==len(path)-1): |
| res+=str(path[len(path)-1-i]) |
| else: |
| res+=str(path[len(path)-1-i])+" -> " |
| print(res) |

**OUTPUT EXAMPLE:**

|  |
| --- |
| # create graph |
|  |  |
|  | testgraph=EdgeWeightedDigraph(9) |
|  |  |
|  | alledges=[] |
|  |  |
|  | alledges.append(DirectedEdge(0,1,3)) |
|  | alledges.append(DirectedEdge(0,2,2)) |
|  | alledges.append(DirectedEdge(0,3,3)) |
|  | alledges.append(DirectedEdge(1,5,4)) |
|  | alledges.append(DirectedEdge(5,3,2)) |
|  | alledges.append(DirectedEdge(3,7,3)) |
|  | alledges.append(DirectedEdge(2,7,5)) |
|  | alledges.append(DirectedEdge(3,4,7)) |
|  | alledges.append(DirectedEdge(4,8,1)) |
|  | alledges.append(DirectedEdge(5,4,5)) |
|  | alledges.append(DirectedEdge(4,7,6)) |
|  | alledges.append(DirectedEdge(7,8,10)) |
|  | alledges.append(DirectedEdge(5,6,3)) |
|  | alledges.append(DirectedEdge(6,8,3)) |
|  |  |
|  | for e in alledges: |
|  | testgraph.addEdge(e) |
|  |  |
|  | test = D(testgraph,0) #set 0 as root |
|  | test.shortestpath(8) |



**DETECTING SPAM EMAILS BY PYTHON**

***CODE:***

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn import svm

spam = pd.read\_csv('C:\\Users\\nethm\\Downloads\\spam.csv')

spam = pd.read\_csv('spam.csv')

train\_test\_split()

z = spam['EmailText']

y = spam["Label"]

z\_train, z\_test,y\_train, y\_test = train\_test\_split(z,y,test\_size = 0.2)

cv = CountVectorizer()

features = cv.fit\_transform(z\_train)

model = svm.SVC()

model.fit(features,y\_train)

features\_test = cv.transform(z\_test)

print(model.score(features\_test,y\_test))

features\_test = cv.transform(z\_test)

print("Accuracy: {}".format(model.score(features\_test,y\_test)))

df\_train = pd.read\_csv("training.csv")

df\_test = pd.read\_csv("test.csv")

df\_test.sample(5)

