Report on Dijkastra's Shortest Path

Avinash Shanker
ID: 1001668570
NET ID: AXS8570

Algorithm Implementation:

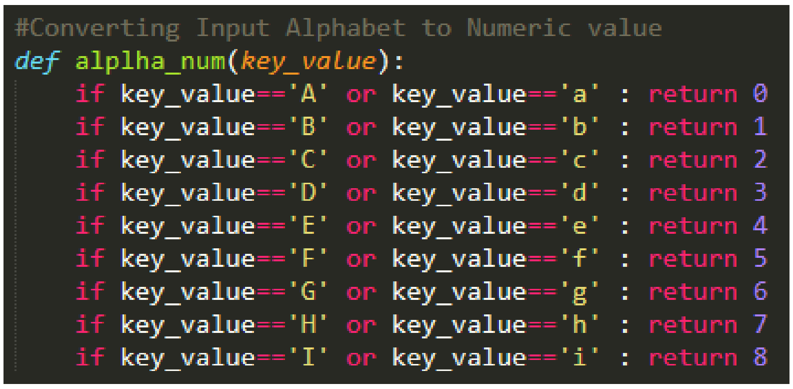
Note test output given by TA is printed in the end

I have implemented my algorithm as multiple function definitons in the following main *Steps*:

1. Obtain input from in the form of Alphabet nodes.
2. Convert human input from eg A -> 0 in the form of a python LIST.
3. To create adjacency matrix and store in TXT file format.
4. To print adjacency Matrix.
5. Disjkstra's function to obtain the shortest path from any node by reading ajacency matrix from TXT file.
6. Function to print the shortest path from any node.
7. My code works by taking input in the **form of a text file**.

Step-1 and _Step-2_ code snippets:

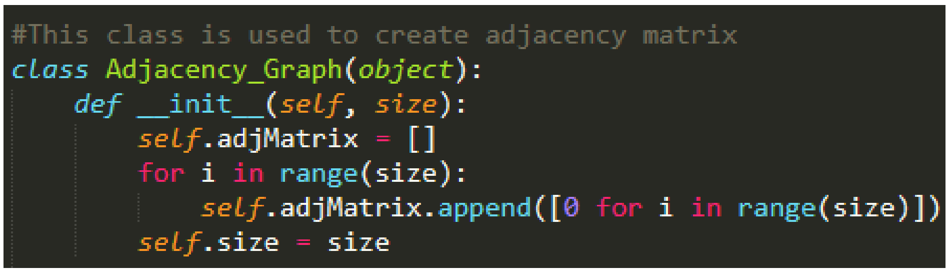
In the below snippet we obtain alphabetic input from user and map it to numeric values



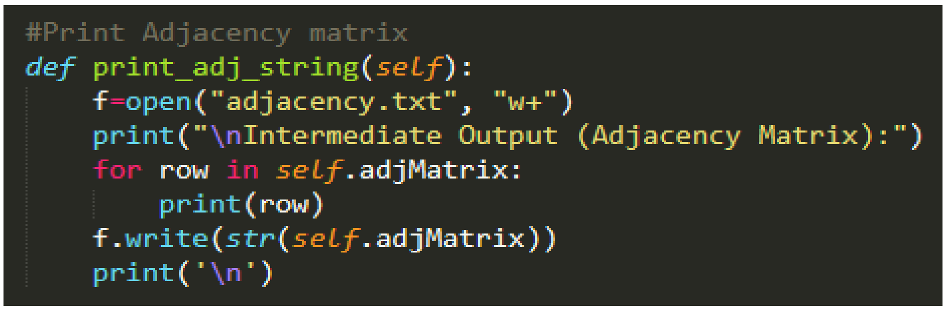
```
#Converting Input Alphabet to Numeric value
def alplha_num(key_value):
    if key_value=='A' or key_value=='a' : return 0
    if key_value=='B' or key_value=='b' : return 1
    if key_value=='C' or key_value=='c' : return 2
    if key_value=='D' or key_value=='d' : return 3
    if key_value=='E' or key_value=='e' : return 4
    if key_value=='F' or key_value=='f' : return 5
    if key_value=='G' or key_value=='g' : return 6
    if key_value=='H' or key_value=='h' : return 7
    if key_value=='I' or key_value=='i' : return 8
```

Step-3 and Step-4 code snippets:

In the below code we are writing the input as ajacency List into a file for dijkastra function to read and process.



```
#This class is used to create adjacency matrix
class Adjacency_Graph(object):
    def __init__(self, size):
        self.adjMatrix = []
        for i in range(size):
            self.adjMatrix.append([0 for i in range(size)])
        self.size = size
```



```
#Print Adjacency matrix
def print_adj_string(self):
    f=open("adjacency.txt", "w+")
    print("\nIntermediate Output (Adjacency Matrix):")
    for row in self.adjMatrix:
        print(row)
    f.write(str(self.adjMatrix))
    print('\n')
```

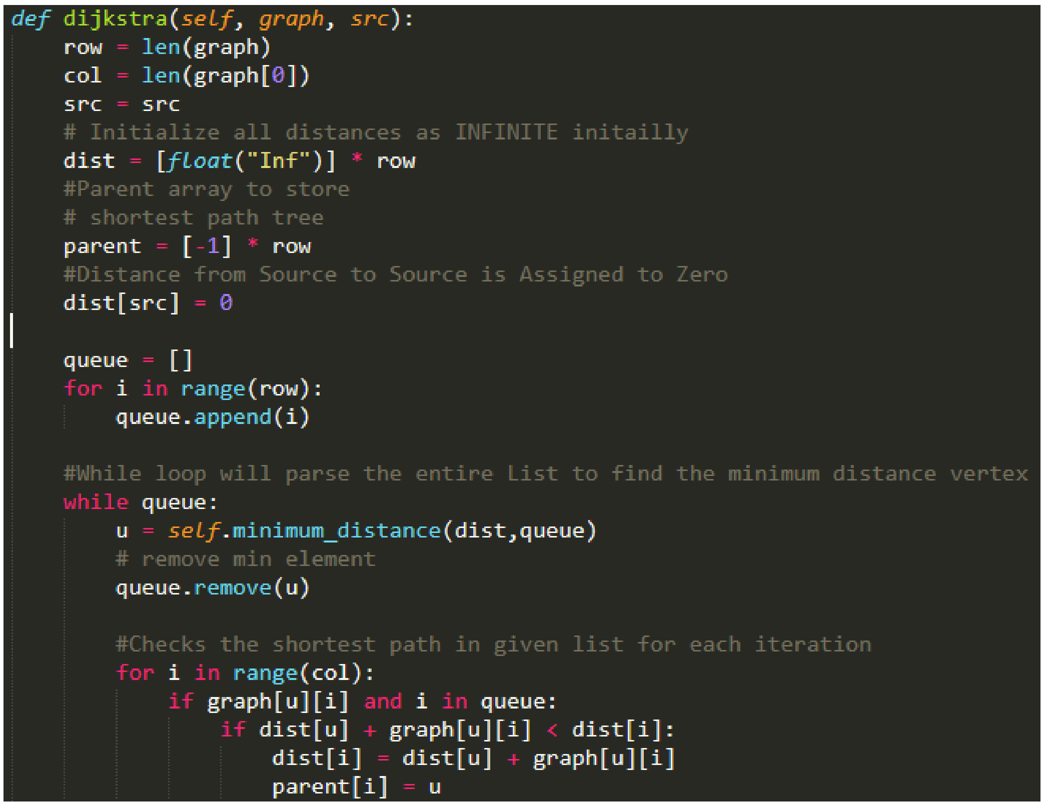
Step 5 Dijkastra's Algorithm:

I have implemented code in the from of undirected graph format as induced from the sample example given in the requirement. i.e. if there is path from A->B them its same as B->A as well.

In the program, I have two sets, first set contains vertices which are included in the shortest path tree and the second set includes the vertices that are to be included in the shortest path yet.

Each time when I find a vertice in the Second that has minimun disrtabce from source include it in the shortest path. For every adjacent vertex v, if sum of distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.

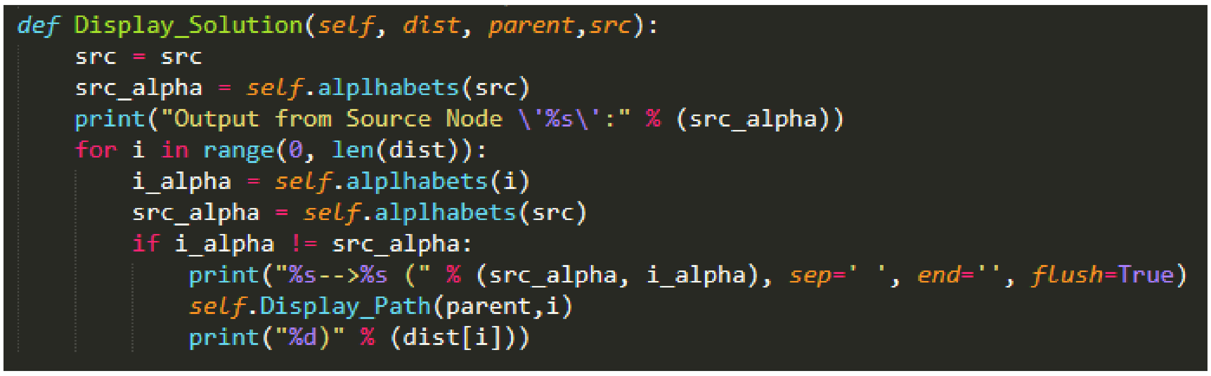
I getting the inputs in the from of python lists from a text files, so incase the inputs are too huge we can read the input from a text file instead of manually entering.



```
def dijkstra(self, graph, src):
    row = len(graph)
    col = len(graph[0])
    src = src
    # Initialize all distances as INFINITE initailly
    dist = [float("Inf")] * row
    #Parent array to store
    # shortest path tree
    parent = [-1] * row
    #Distance from Source to Source is Assigned to Zero
    dist[src] = 0

    queue = []
    for i in range(row):
        queue.append(i)

    #While loop will parse the entire List to find the minimum distance vertex
    while queue:
        u = self.minimum_distance(dist,queue)
        # remove min element
        queue.remove(u)

        #Checks the shortest path in given list for each iteration
        for i in range(col):
            if graph[u][i] and i in queue:
                if dist[u] + graph[u][i] < dist[i]:
                    dist[i] = dist[u] + graph[u][i]
                    parent[i] = u
```

Step 6 to print the shortest path:

The idea is to create a separate array parent[]. Value of parent[v] for a vertex v stores parent vertex of v in shortest path tree. Parent of root (or source vertex) is -1. Whenever we find shorter path through a vertex u, we make u as parent of current vertex. Once we have parent array constructed, we can print path using below recursive function.



```
def Display_Solution(self, dist, parent,src):
    src = src
    src_alpha = self.alplhabets(src)
    print("Output from Source Node \'%s\':" % (src_alpha))
    for i in range(0, len(dist)):
        i_alpha = self.alplhabets(i)
        src_alpha = self.alplhabets(src)
        if i_alpha != src_alpha:
            print("%s-->%s (" % (src_alpha, i_alpha), sep=' ', end='', flush=True)
            self.Display_Path(parent,i)
            print("%d)" % (dist[i]))
```

Time Complexity of Algorithm:

Time Complexity of the algorithm is implemtned is Big O(n2). Where ‘n’ represents the number of vertices given and also input graph must given in the form of a adjacency list format.

Real Life Example:

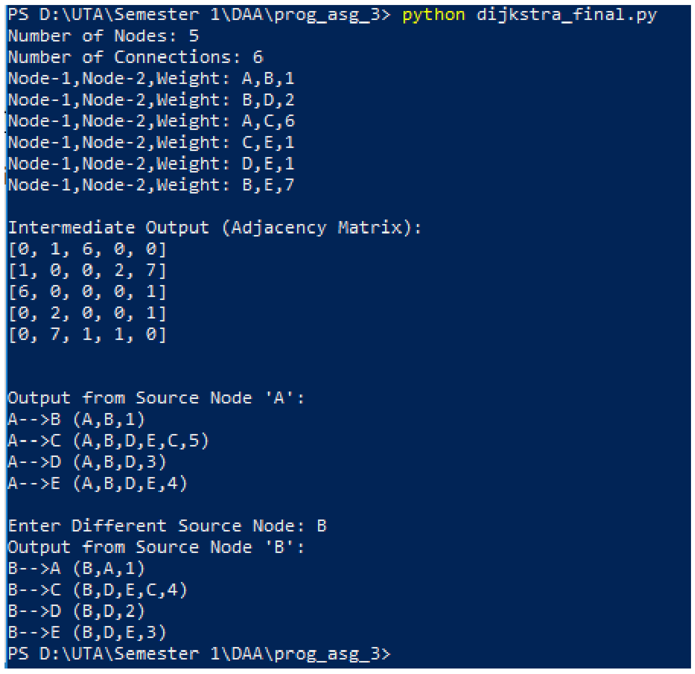
Problem of determining the estimated time of arrival (ETA) at the destination port for a ship located at sea. This problem is formulated as a shortest path problem with obstacles, where the obstacles are modelled by polygons representing the coastlines.

Link: https://www.jstor.org/stable/254011?seq=1#page_scan_tab_contents

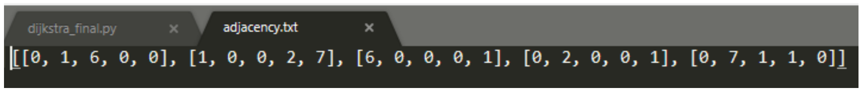
How to use and test output:

1. Navigate to path where code is placed
2. In command prompt if **Python3** is already installed, use the below line call the code
   > python dijkstra_final.py
3. Enter the Number of Nodes and its connection with enter key after each input
4. Enter Nodes separated by comma two at a time with the weight
5. Use th below sample visual output as guideline incase if any issues.
6. This can also be used by passing the adjacency list from a text file

Code Working Correctly for Sample Given:

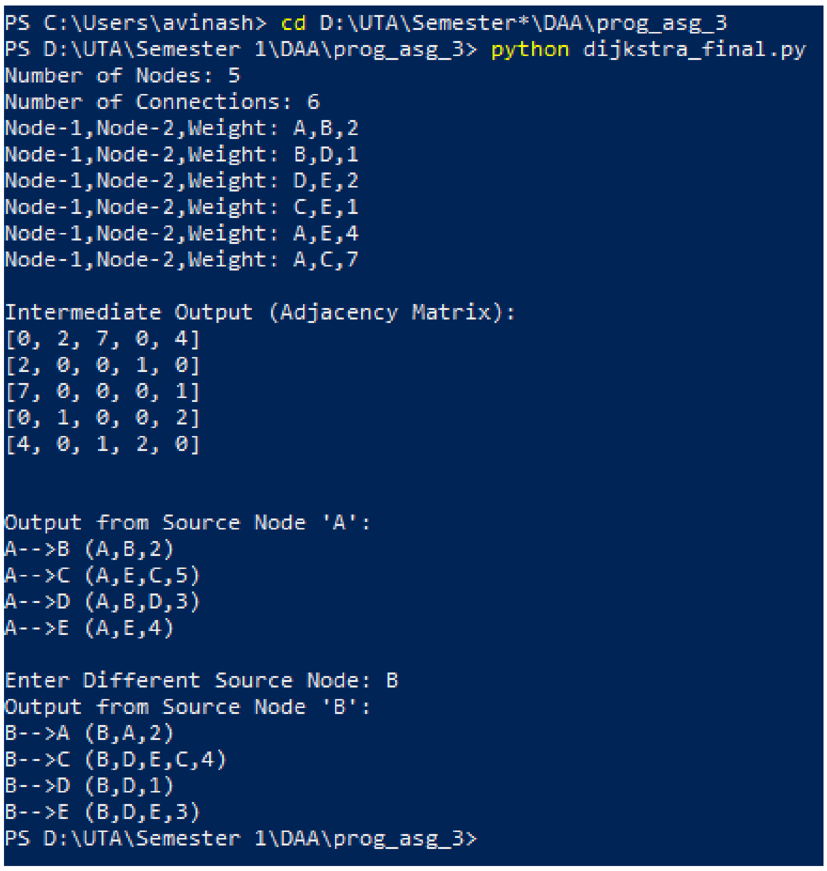


```
PS D:\UTA\Semester 1\DAA\prog_asg_3> python dijkstra_final.py
Number of Nodes: 5
Number of Connections: 6
Node-1,Node-2,Weight: A,B,1
Node-1,Node-2,Weight: B,D,2
Node-1,Node-2,Weight: A,C,6
Node-1,Node-2,Weight: C,E,1
Node-1,Node-2,Weight: D,E,1
Node-1,Node-2,Weight: B,E,7

Intermediate Output (Adjacency Matrix):
[0, 1, 6, 0, 0]
[1, 0, 0, 2, 7]
[6, 0, 0, 0, 1]
[0, 2, 0, 0, 1]
[0, 7, 1, 1, 0]


Output from Source Node 'A':
A-->B (A,B,1)
A-->C (A,B,D,E,C,5)
A-->D (A,B,D,3)
A-->E (A,B,D,E,4)

Enter Different Source Node: B
Output from Source Node 'B':
B-->A (B,A,1)
B-->C (B,D,E,C,4)
B-->D (B,D,2)
B-->E (B,D,E,3)
PS D:\UTA\Semester 1\DAA\prog_asg_3>
```



```
[[0, 1, 6, 0, 0], [1, 0, 0, 2, 7], [6, 0, 0, 0, 1], [0, 2, 0, 0, 1], [0, 7, 1, 1, 0]]
```

Output of the Test inputs Given which is correctly printed:



```
PS C:\Users\avinash> cd D:\UTA\Semester*\DAA\prog_asg_3
PS D:\UTA\Semester 1\DAA\prog_asg_3> python dijkstra_final.py
Number of Nodes: 5
Number of Connections: 6
Node-1,Node-2,Weight: A,B,2
Node-1,Node-2,Weight: B,D,1
Node-1,Node-2,Weight: D,E,2
Node-1,Node-2,Weight: C,E,1
Node-1,Node-2,Weight: A,E,4
Node-1,Node-2,Weight: A,C,7

Intermediate Output (Adjacency Matrix):
[0, 2, 7, 0, 4]
[2, 0, 0, 1, 0]
[7, 0, 0, 0, 1]
[0, 1, 0, 0, 2]
[4, 0, 1, 2, 0]


Output from Source Node 'A':
A-->B (A,B,2)
A-->C (A,E,C,5)
A-->D (A,B,D,3)
A-->E (A,E,4)

Enter Different Source Node: B
Output from Source Node 'B':
B-->A (B,A,2)
B-->C (B,D,E,C,4)
B-->D (B,D,1)
B-->E (B,D,E,3)
PS D:\UTA\Semester 1\DAA\prog_asg_3>
```

References:

1. *https://www.geeksforgeeks.org/printing-paths-dijkstras-shortest-path-algorithm/*
2. *https://www.geeksforgeeks.org/dijkstras-algorithm-for-adjacency-list-representation-greedy-algo-8/*
3. *https://www.jstor.org/stable/254011?seq=1#page_scan_tab_contents*
4. *https://www.programiz.com/dsa/graph-adjacency-matrix*
5. *https://gist.github.com/econchick/4666413*