Assignment 2

Dijkstra's Algorithm

Computer Networks

Name	ID
Omar Mohamed Diaaeldin Ibrahim Elsayed	1802932
Ahmed Magdy Fahmy Mohamed Ahmed	1805862
Ahmed Magdi Mostafa Hosni	1808714
Ahmed Ayman Abd-Alaziz Sharf	1806171
Yousef Adel Ismail Mohammed Shalaby	1802267

Assignment 2

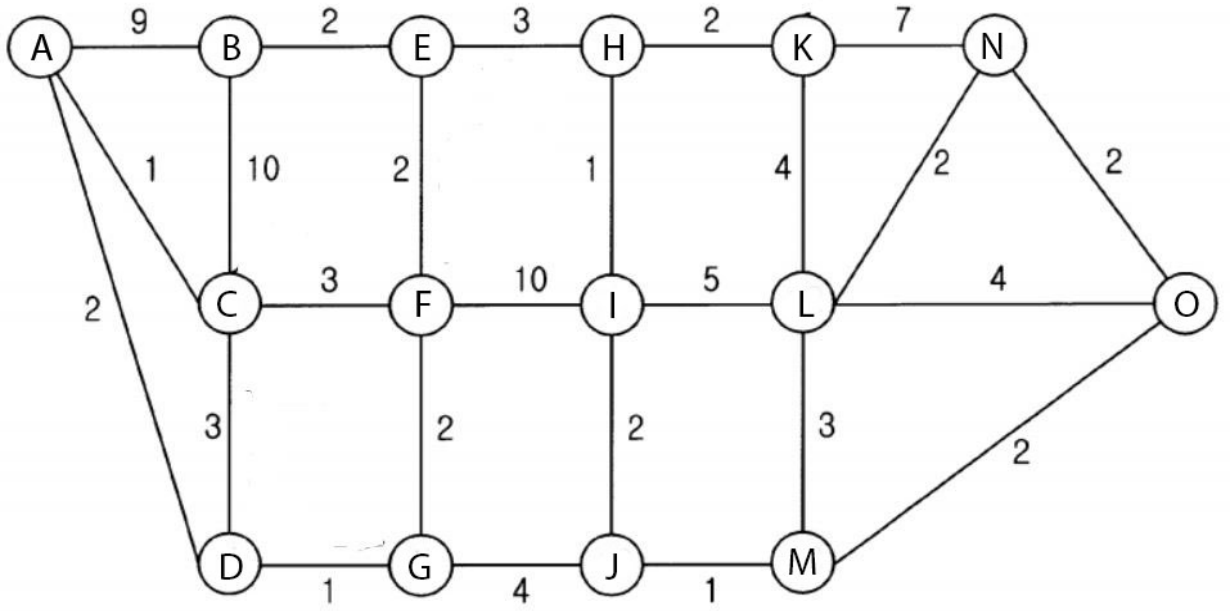
Dijkstra's Algorithm Code (Input Example is Hard Coded):

```
'''
            Dijkstra's Algorithm

Dijkstra's algorithm find the shortest path between any two nodes
by checking the neighboring nodes and calculating the cost
(distace of previous node + distance to the neighboring node)
then comparing with each others and choosing the minimum value and append
this minimum value in a list and this list will include the shortest path
nodes.

This steps are repeated till the terminal node is reached.
'''

import sys

def dijkstra(nodes_graph,source,destination):

    # A function that takes nodes parameter (which describes how nodes are connected)
    # source parameter is the first terminal that we want to start from
    # destination parameter is the final terminal that we want to reach

    # We set the cost of all terminal to infinty ( max number to compare with to get
    # the minimum when the node is not visited yet)
    # create a dictionary to hold nodes data

    inf = sys.maxsize

    node_data = {}
    unvisited = nodes_graph.copy()
    path = []
    track_previous = {}

    for node in nodes_graph:
        node_data[node] = inf

    # source terminal cost must be zero

    node_data[source] = 0

    while unvisited:
        min_distance_node = None

        for node in unvisited:
            if min_distance_node is None:
                min_distance_node = node
            elif node_data[node] < node_data[min_distance_node]:
                min_distance_node = node

        path_options = nodes_graph[min_distance_node].items()

        for child_node, weight in path_options:
            if ((weight + node_data[min_distance_node]) < node_data[child_node]):
                node_data[child_node] = weight + node_data[min_distance_node]
                track_previous[child_node] = min_distance_node

        unvisited.pop(min_distance_node)
```

```
    currentNode = destination

    while currentNode != source:
        try:
            path.insert(0, currentNode)
            currentNode = track_previous[currentNode]

        except KeyError:
            print("Path is not reachable\n")
            break

    path.insert(0, source)

    # printing the shortest distance and the shortest path
    if node_data[destination] != inf:
        print("\n")
        print("Shortest Distance to the Final Node: " + str(node_data[destination]))
        print("Shortest Path: " + str(path))
        print("\n")



if __name__ == "__main__":

    nodes_graph = {
        'A': {'B':9, 'C':1, 'D':2},
        'B': {'A':9, 'C':10, 'E':2},
        'C': {'A':1, 'B':10, 'D':3, 'F':3},
        'D': {'A':2, 'C':3, 'G':1},
        'E': {'B':2, 'F':2, 'H':3},
        'F': {'C':3, 'E':2, 'G':2, 'I':10},
        'G': {'D':1, 'F':2, 'J':4},
        'H': {'E':3, 'I':1, 'K':2},
        'I': {'F':10, 'H':1, 'L':5, 'J':2},
        'J': {'G':4, 'I':2, 'M':1},
        'K': {'H':2, 'N':7, 'L':4},
        'L': {'I':5, 'K':4, 'N':2, 'M':3},
        'M': {'J':1, 'L':3, 'O':2},
        'N': {'K':7, 'L':2, 'O':2},
        'O': {'N':2, 'L':4, 'M':2}
        }


    char = "Y"

    while(char == "Y"):

        source = input("Enter The name of the source node: ").title()
        destination = input("Enter The name of the destination node: ").title()

        dijkstra(nodes_graph,source,destination)

        char = input("Do you want to try the path between 2 different nodes? y/n: ").title()
        print("\n")
```

Test Cases:

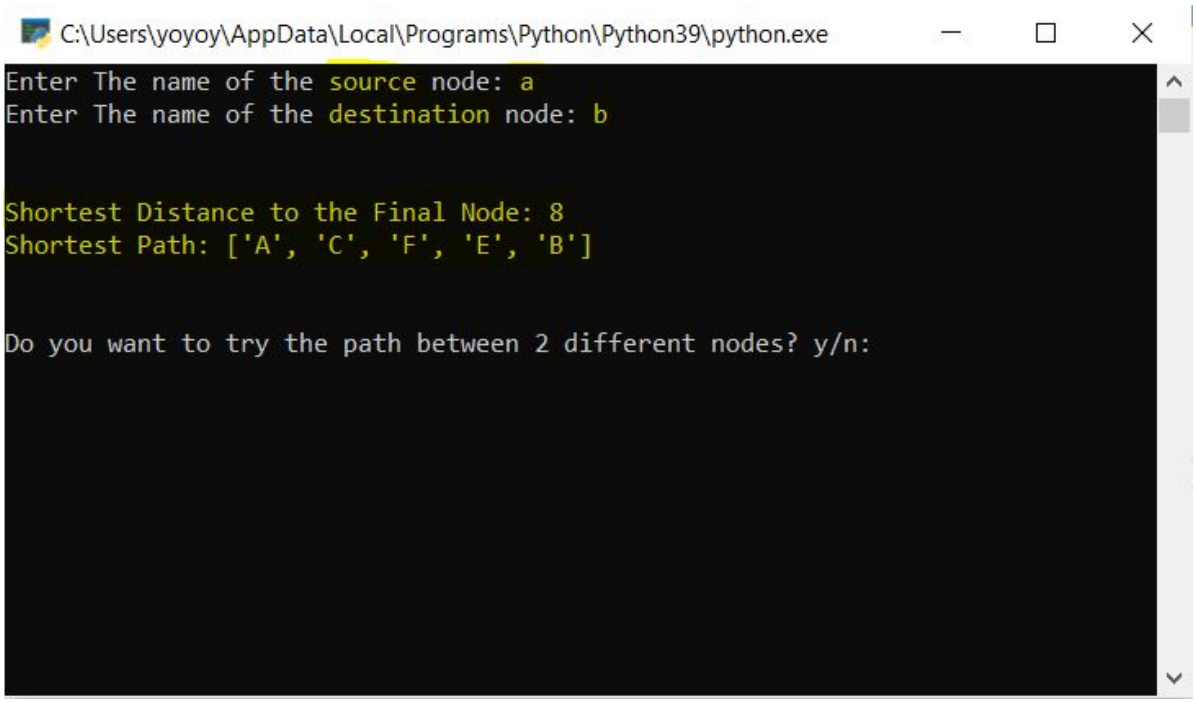
Example:



Test Case 1:

Source Node: A

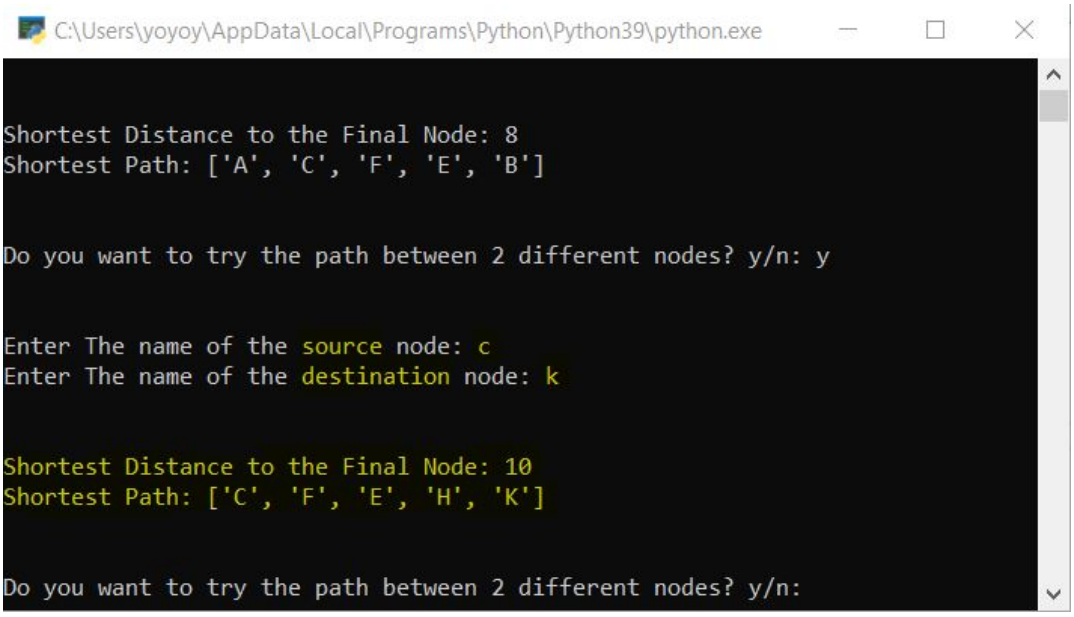
Destination Node: B



Test Case 2:

Source Node: C

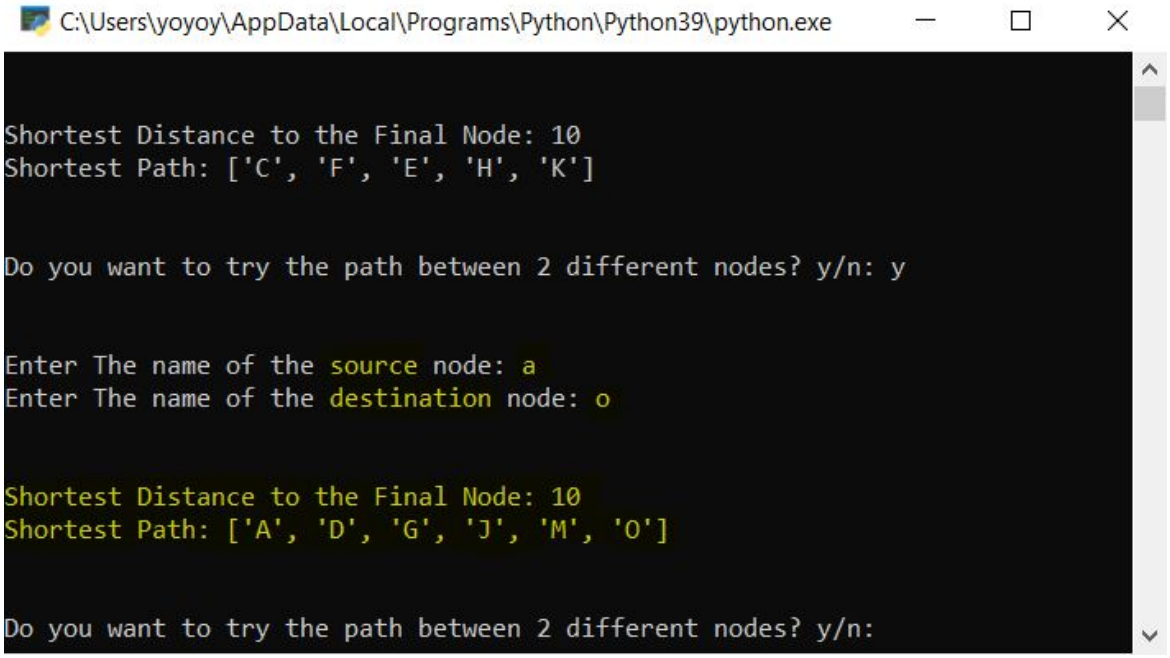
Destination Node: K



Test Case 3:

Source Node: A

Destination Node: O



Dijkstra's Algorithm Code (User should input number of Nodes and Distances between Nodes Manually):

Code:

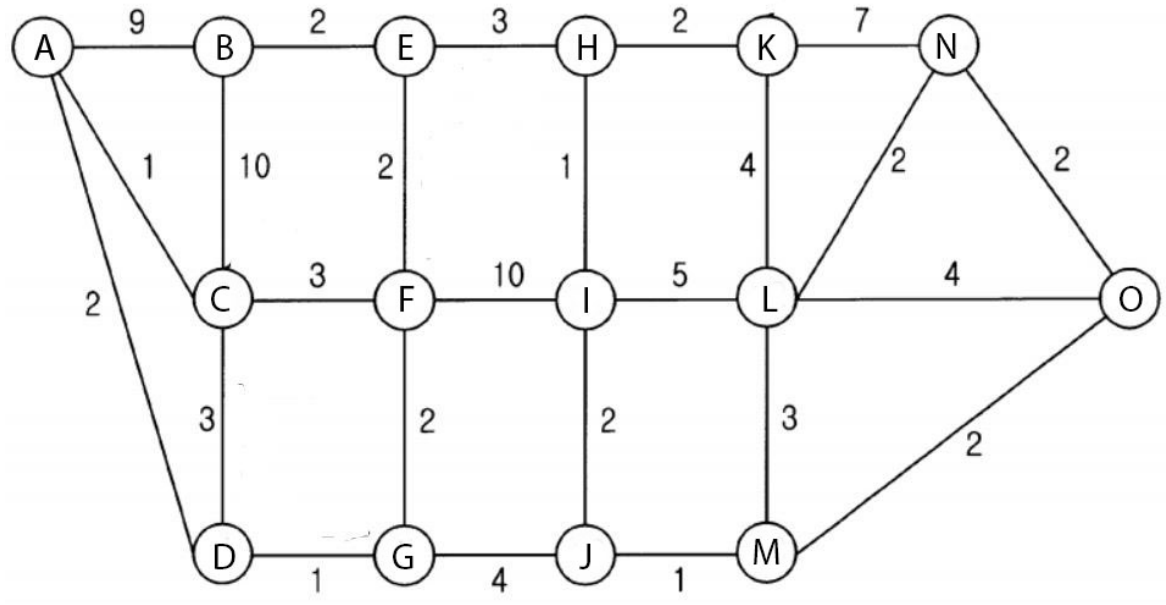
```
'''
            Dijkstra's Algorithm

Dijkstra's algorithm find the shortest path between any two nodes
by checking the neighboring nodes and calculating the cost
(distace of previous node + distance to the neighboring node)
then comparing with each others and choosing the minimum value and append
this minimum value in a list and this list will include the shortest path
nodes.

This steps are repeated till the terminal node is reached.
'''

import sys
import time

def dijkstra(n_nodes,nodes_graph,source,destination):

    # A function that takes nodes parameter (which describes how nodes are connected)
    # source parameter is the first terminal that we want to start from
    # destination parameter is the final terminal that we want to reach

    # We set the cost of all terminal to infinty ( max number to compare with to get
    # the minimum when the node is not visited yet)
    # create a dictionary to hold nodes data

    inf = sys.maxsize

    node_data = {}
    unvisited = nodes_graph.copy()
    path = []
    track_previous = {}

    for node in nodes_graph:
        node_data[node] = inf

    # source terminal cost must be zero

    node_data[source] = 0

    while unvisited:
        min_distance_node = None

        for node in unvisited:
            if min_distance_node is None:
                min_distance_node = node
            elif node_data[node] < node_data[min_distance_node]:
                min_distance_node = node

        path_options = nodes_graph[min_distance_node].items()

        for child_node, weight in path_options:
            if ((weight + node_data[min_distance_node]) < node_data[child_node]):
                node_data[child_node] = weight + node_data[min_distance_node]
                track_previous[child_node] = min_distance_node
```

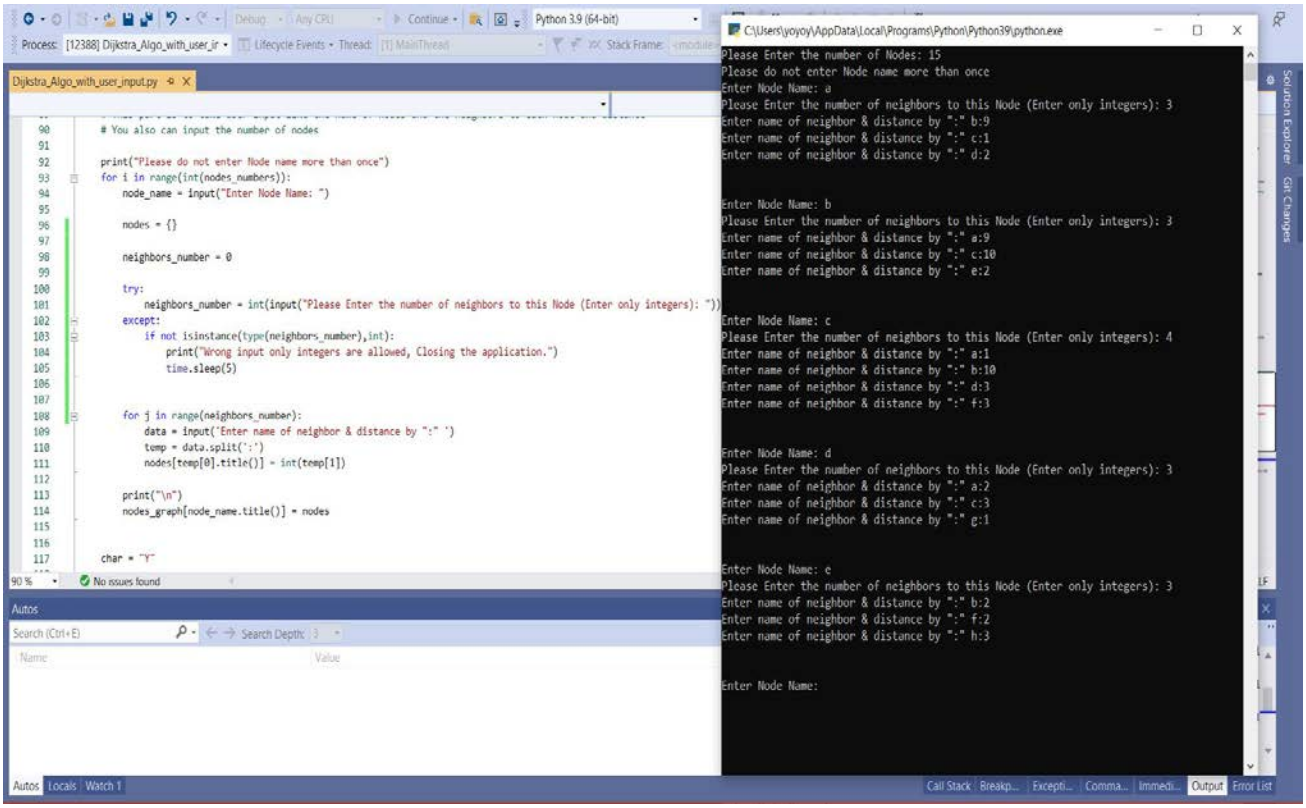
```
        unvisited.pop(min_distance_node)


    currentNode = destination

    while currentNode != source:
        try:
            path.insert(0, currentNode)
            currentNode = track_previous[currentNode]

        except KeyError:
            print("Path is not reachable\n")
            break

    path.insert(0, source)

    # printing the shortest distance and the shortest path
    if node_data[destination] != inf:
        print("\n")
        print("Shortest Distance to the Final Node: " + str(node_data[destination]))
        print("Shortest Path: " + str(path))
        print("\n")



if __name__ == "__main__":

    nodes_graph = {}

    nodes_numbers = input("Please Enter the number of Nodes: ")

    # This part is to take user input like the name of nodes and the neighbors to each node
and distance
    # You also can input the number of nodes

    print("Please do not enter Node name more than once")
    for i in range(int(nodes_numbers)):
        node_name = input("Enter Node Name: ")

        nodes = {}

        neighbors_number = 0

        try:
            neighbors_number = int(input("Please Enter the number of neighbors to this Node
(Enter only integers): "))
        except:
            if not isinstance(type(neighbors_number),int):
                print("Wrong input only integers are allowed, Closing the application.")
                time.sleep(5)


        for j in range(neighbors_number):
            data = input('Enter name of neighbor & distance by ":" ')
            temp = data.split(':')
            nodes[temp[0].title()] = int(temp[1])

        print("\n")
        nodes_graph[node_name.title()] = nodes


    char = "Y"
```

```
while(char == "Y"):
    source = input("Enter The name of the source node: ").title()
    destination = input("Enter The name of the destination node: ").title()

    dijkstra(nodes_numbers,graph,source,destination)

    char = input("Do you want to try the path between 2 different nodes? y/n: ").title()
    print("\n")
```

Input Example:

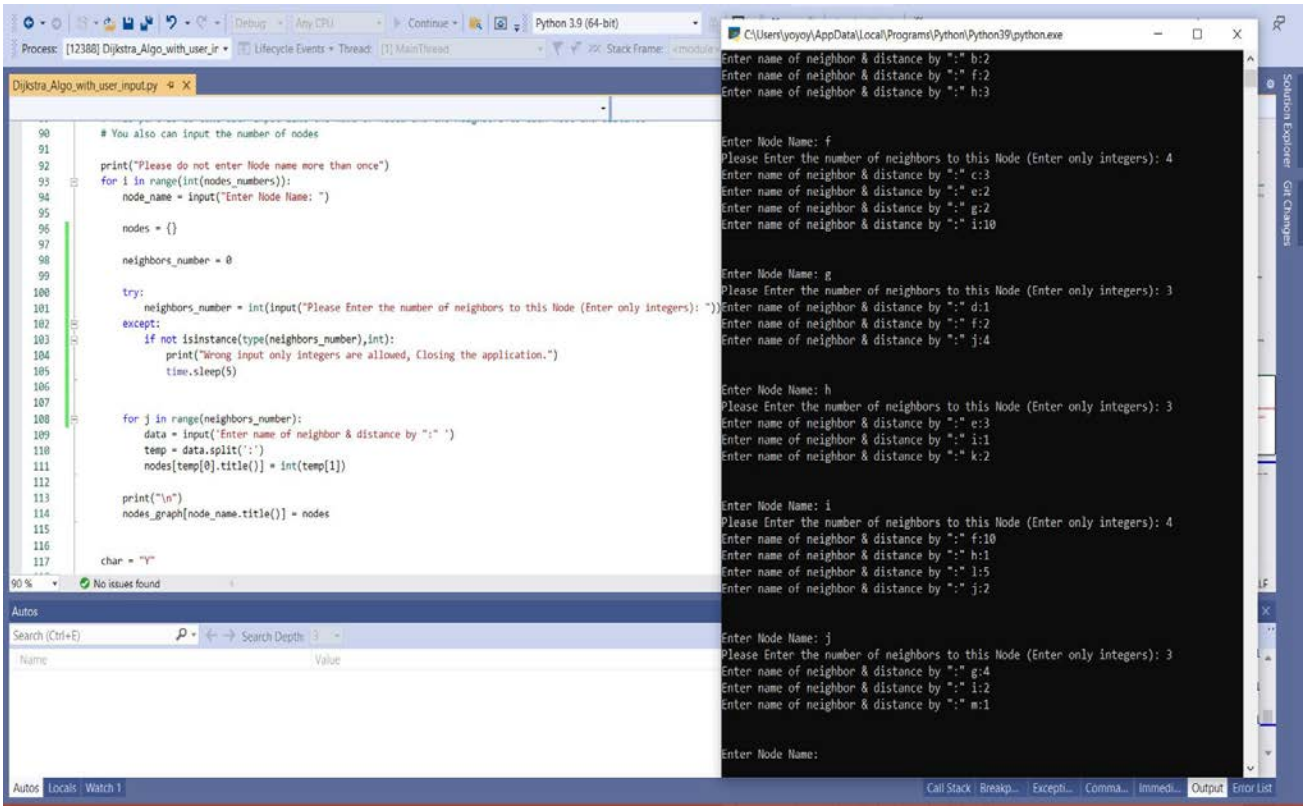
We will enter the same Node Examples Manually





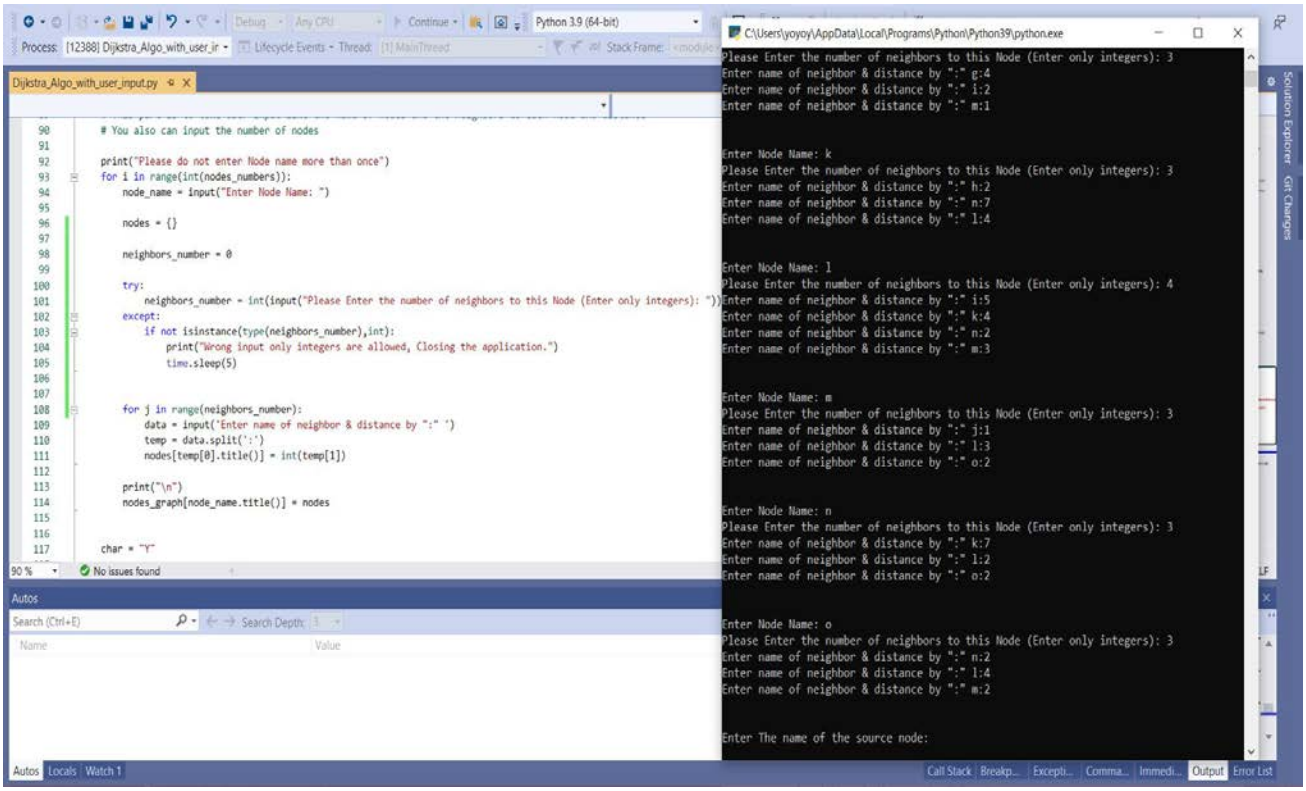
```
# You also can input the number of nodes

print("Please do not enter Node name more than once")
for i in range(int(nodes_numbers)):
    node_name = input("Enter Node Name: ")

    nodes = {}

    neighbors_number = 0

    try:
        neighbors_number = int(input("Please Enter the number of neighbors to this Node (Enter only integers): "))
    except:
        if not isinstance(type(neighbors_number),int):
            print("Wrong input only integers are allowed, Closing the application.")
            time.sleep(5)


    for j in range(neighbors_number):
        data = input('Enter name of neighbor & distance by ":" ')
        temp = data.split(':')
        nodes[temp[0].title()] = int(temp[1])

    print("\n")
    nodes_graph[node_name.title()] = nodes


char = "Y"
```

```
Please Enter the number of Nodes: 15
Please do not enter Node name more than once
Enter Node Name: a
Please Enter the number of neighbors to this Node (Enter only integers): 3
Enter name of neighbor & distance by ":" b:9
Enter name of neighbor & distance by ":" c:1
Enter name of neighbor & distance by ":" d:2


Enter Node Name: b
Please Enter the number of neighbors to this Node (Enter only integers): 3
Enter name of neighbor & distance by ":" a:9
Enter name of neighbor & distance by ":" c:10
Enter name of neighbor & distance by ":" e:2


Enter Node Name: c
Please Enter the number of neighbors to this Node (Enter only integers): 4
Enter name of neighbor & distance by ":" a:1
Enter name of neighbor & distance by ":" b:10
Enter name of neighbor & distance by ":" d:3
Enter name of neighbor & distance by ":" f:3


Enter Node Name: d
Please Enter the number of neighbors to this Node (Enter only integers): 3
Enter name of neighbor & distance by ":" a:2
Enter name of neighbor & distance by ":" c:3
Enter name of neighbor & distance by ":" g:1


Enter Node Name: e
Please Enter the number of neighbors to this Node (Enter only integers): 3
Enter name of neighbor & distance by ":" b:2
Enter name of neighbor & distance by ":" f:2
Enter name of neighbor & distance by ":" h:3


Enter Node Name:
```



```
# You also can input the number of nodes

print("Please do not enter Node name more than once")
for i in range(int(nodes_numbers)):
    node_name = input("Enter Node Name: ")

    nodes = {}

    neighbors_number = 0

    try:
        neighbors_number = int(input("Please Enter the number of neighbors to this Node (Enter only integers): "))
    except:
        if not isinstance(type(neighbors_number),int):
            print("Wrong input only integers are allowed, Closing the application.")
            time.sleep(5)


    for j in range(neighbors_number):
        data = input('Enter name of neighbor & distance by ":" ')
        temp = data.split(':')
        nodes[temp[0].title()] = int(temp[1])

    print("\n")
    nodes_graph[node_name.title()] = nodes


char = "Y"
```

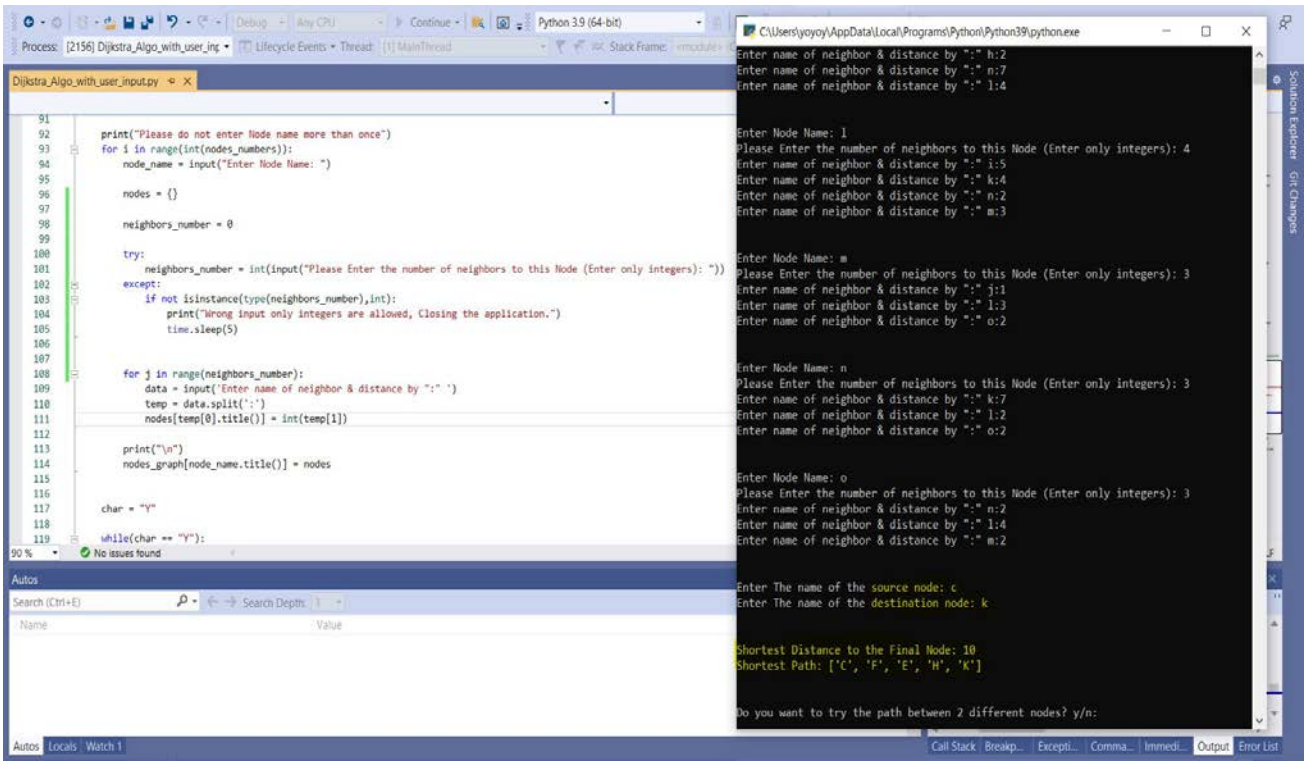
```
Enter name of neighbor & distance by ":" b:2
Enter name of neighbor & distance by ":" f:2
Enter name of neighbor & distance by ":" h:3


Enter Node Name: f
Please Enter the number of neighbors to this Node (Enter only integers): 4
Enter name of neighbor & distance by ":" c:3
Enter name of neighbor & distance by ":" e:2
Enter name of neighbor & distance by ":" g:2
Enter name of neighbor & distance by ":" i:10


Enter Node Name: g
Please Enter the number of neighbors to this Node (Enter only integers): 3
Enter name of neighbor & distance by ":" d:1
Enter name of neighbor & distance by ":" f:2
Enter name of neighbor & distance by ":" j:4


Enter Node Name: h
Please Enter the number of neighbors to this Node (Enter only integers): 3
Enter name of neighbor & distance by ":" e:3
Enter name of neighbor & distance by ":" i:1
Enter name of neighbor & distance by ":" k:2


Enter Node Name: i
Please Enter the number of neighbors to this Node (Enter only integers): 4
Enter name of neighbor & distance by ":" f:10
Enter name of neighbor & distance by ":" h:1
Enter name of neighbor & distance by ":" l:5
Enter name of neighbor & distance by ":" j:2


Enter Node Name: j
Please Enter the number of neighbors to this Node (Enter only integers): 3
Enter name of neighbor & distance by ":" g:4
Enter name of neighbor & distance by ":" i:2
Enter name of neighbor & distance by ":" m:1


Enter Node Name:
```



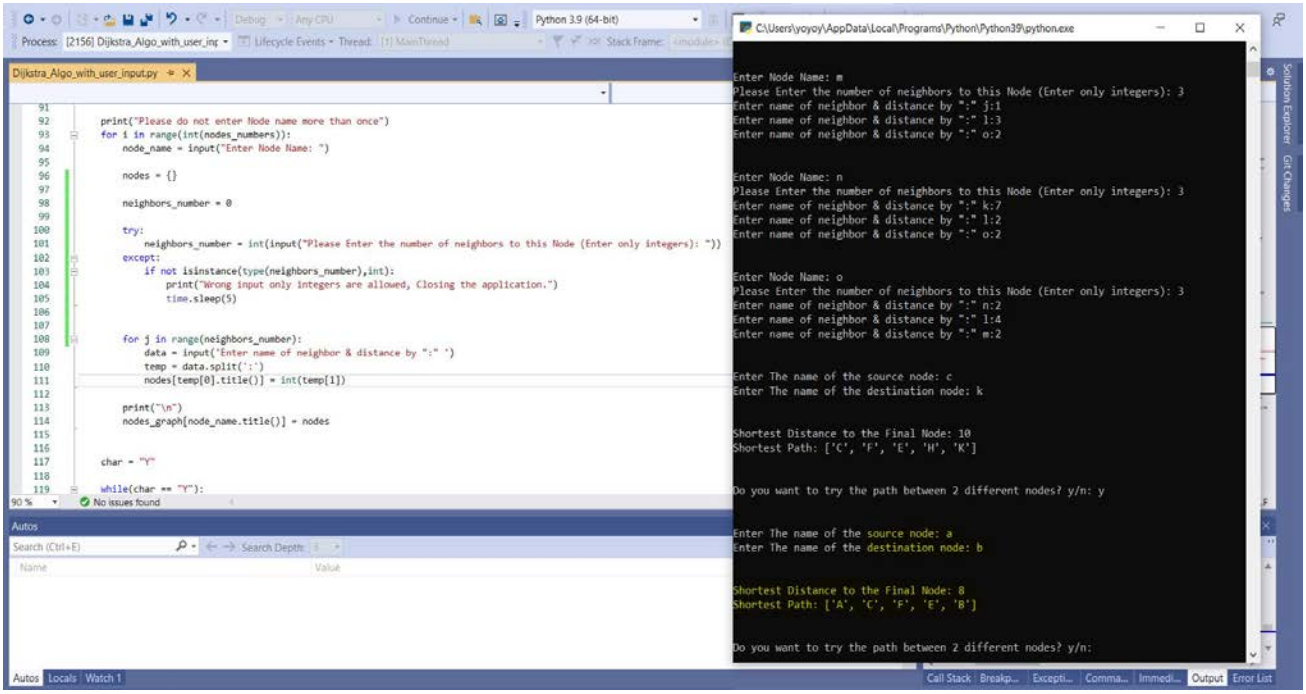
Test Cases:

Test Case 1:

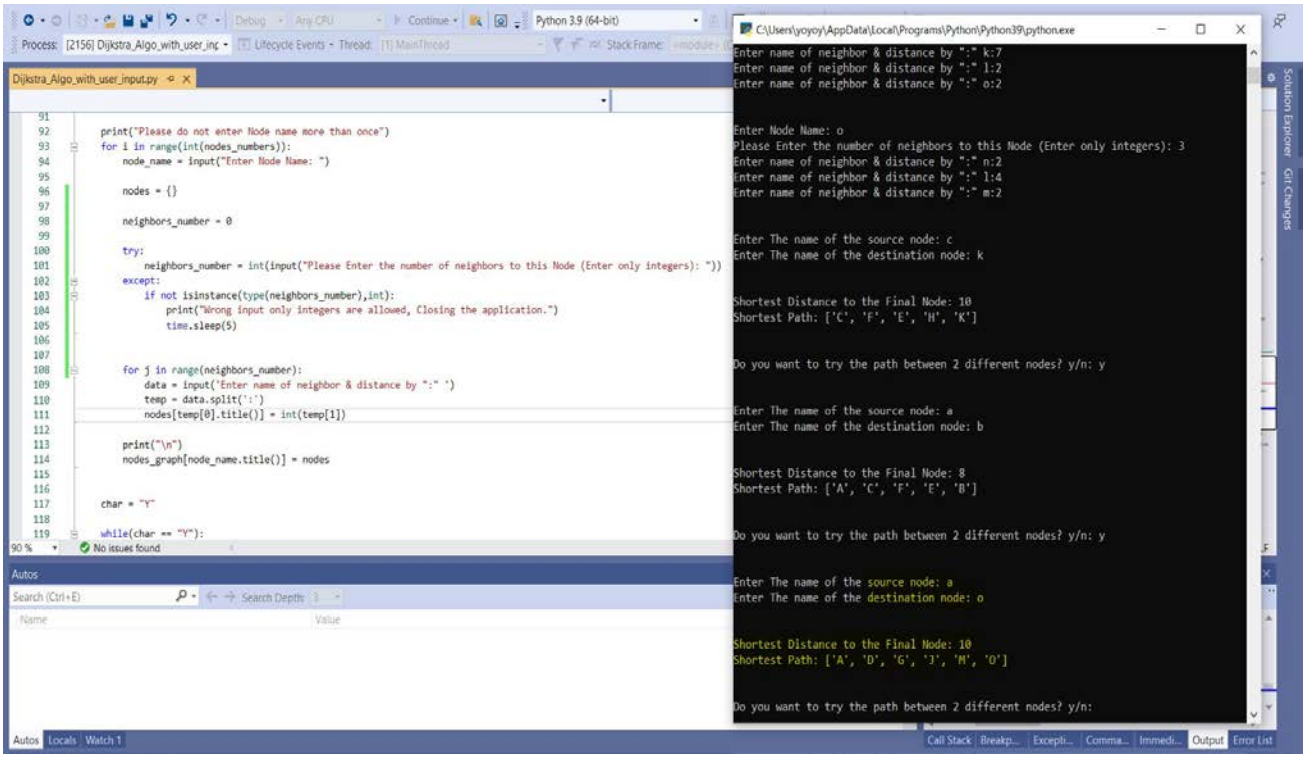
Source: C
Destination: K



Test Case 2:
Source: A
Destination: B



Test Case 3:
Source: A
Destination: O



Closing the application: