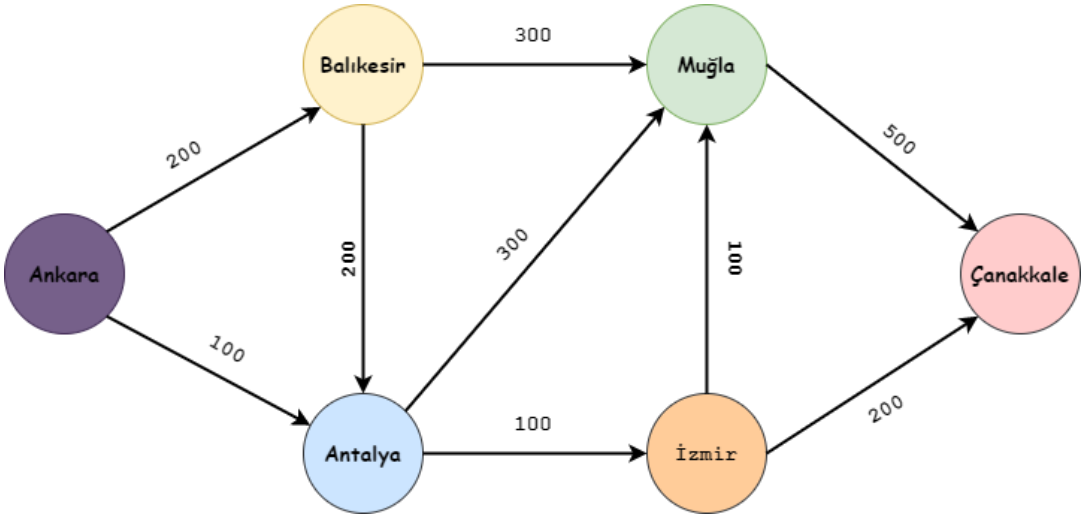
GMT203_1 DATA STRUCTURES&ALGORITHMS

FINAL PROJECT

22.01.2021

Small Navigation (Road Map) With Dijkstra Algorithm



CONTENTS OF REPORT

- **Part I - What is the Dijkstra Algorithm?**
- **Part II - Purpose and Use Cases of Dijkstra**
- **Part III - Dijkstra's Working Logic**
- **Part IV - Description of our project, purpose and why we chose it?**
- **Part V – Analysis of the Project**

Abdulsamet TOPTAŞ 21905024

PART I – WHAT IS THE "DIJKSTRA ALGORITHM"?

Let's talk about the "Dijkstra algorithm". This algorithm is named after Dr. Edsger W. Dijkstra. This is an algorithm that let us to find the shortest path between any two vertices of a graph. The shortest path is found assuming that none of the lengths are negative. It was designed by Dutch physicist Edsger Dijkstra in 1956, when he thought about how he might calculate the shortest route from Rotterdam to Groningen. You might be.deceived by the popularity of the Dijkstra algorithm and consider it a perfect algorithm, but the Dijkstra algorithm is an intuitive algorithm. So it doesn't guarentee the best results, however, it is a very effective algorithm.

PART II – PURPOSE AND USE CASES OF DIJKSTRA

As we mentioned above, you can find the shortest path from a node (called "source node") to all other nodes in the graph, producing a shortest-path tree. This algorithm is used in GPS devices to find the shortest path between the current location and the destination. Application of Dijkstra's algorithm has been used widely in several different aspects of engineering and it has several real-world use cases, some of which are as follows:

1. **Digital Mapping Services in Google Maps:** Many times we have tried to find the distance in G-Maps, from one city to another, or from your location to the nearest desired location. There encounters the Shortest Path Algorithm, as there are various routes/paths connecting them but it has to show the minimum distance, so Dijkstra's Algorithm is used to find the minimum distance between two locations along the path.
2. **Social Networking Applications:** In many applications you might have seen the app suggests the list of friends that a particular user may know. How do you think many social media companies implement this feature efficiently, especially when the system has over a billion users. The standard Dijkstra algorithm can be applied using the shortest path between users measured through handshakes or connections among them. When the social networking graph is very small, it uses standard Dijkstra's algorithm along with some other features to find the shortest paths, and however, when the graph is becoming bigger and bigger, the standard algorithm takes a few several seconds to count and alternate advanced algorithms are used.

3. **Telephone Network:** As we know, in a telephone network, each line has a bandwidth, ‘b’. The bandwidth of the transmission line is the highest frequency that that line can support. Generally, if the frequency of the signal is higher in a certain line, the signal is reduced by that line. Bandwidth represents the amount of information that can be transmitted by the line. If we imagine a city to be a graph, the vertices represent the switching stations, and the edges represent the transmission lines and the weight of the edges represents ‘b’. So as you can see it can fall into the category of shortest distance problem, for which the Dijkstra is can be used.
4. **IP routing to find Open shortest Path First:** Open Shortest Path First (OSPF) is a link-state routing protocol that is used to find the best path between the source and the destination router using its own Shortest Path First. The algorithm provides the shortest cost path from the source router to other routers in the network.
5. **Flighting Agenda:** For example, If a person needs software for making an agenda of flights for customers. The agent has access to a database with all airports and flights. Besides the flight number, origin airport, and destination, the flights have departure and arrival time. Specifically, the agent wants to determine the earliest arrival time for the destination given an origin airport and start time. There this algorithm comes into use.
6. **Designate file server:** To designate a file server in a LAN(local area network), Dijkstra’s algorithm can be used. Consider that an infinite amount of time is required for transmitting files from one computer to another computer. Therefore to minimize the number of “hops” from the file server to every other computer on the network the idea is to use Dijkstra’s algorithm to minimize the shortest path between the networks resulting in the minimum number of hops.
7. **Robotic Path:** Nowadays, drones and robots have come into existence, some of which are manual, some automated. The drones/robots which are automated and are used to deliver the packages to a specific location or used for a task are loaded with this algorithm module so that when the source and destination is known, the robot/drone moves in the ordered direction by following the shortest path to keep delivering the package in a minimum amount of time.

PART III –DIJKSTRA’S WORKING LOGIC

The algorithm steps as following:

1. The starting point is permanently resolved on the current set of nodes and the set of adjacent nodes accessible to this node is detected.
2. The shortest path from the current set of nodes to the detected set of accessible nodes is found and stored. Stops if target node is reached otherwise go to step 3.
3. The accessible node for the shortest path selected is included in the current cluster.
4. Accessible nodes of the current cluster are found again and Go again to step 2.

EXAMPLE

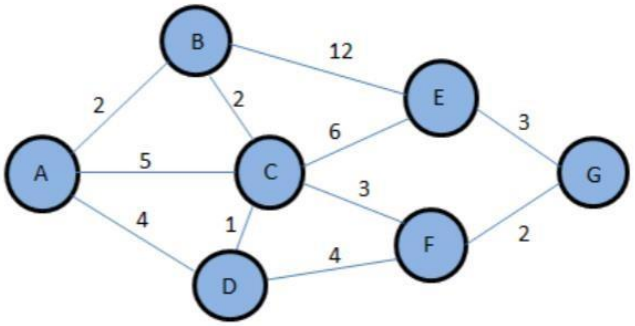


Figure 1 - Graph used for the explanation of Dijkstra's algorithm

Using the network given in Figure 1, the shortest path will be found with the Dijkstra algorithm. In this example, the goal is to find the shortest path from node A to node G.

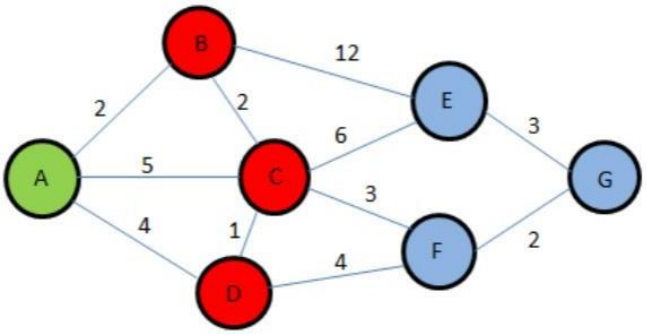


Figure 2

In Figure 2, as mentioned in Step 1, node A has been permanently added to the current set of nodes. The set of neighbor nodes that can be accessed from node A is defined as "B, C and D".

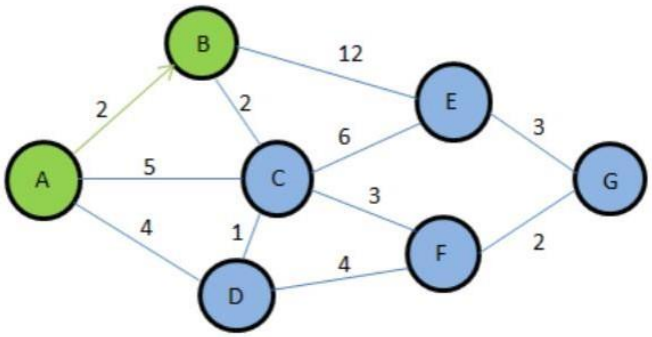


Figure 3

In Figure 3, node B in the set of accessible nodes detected in step 1 is stored because it is the closest to node A. Node B is included in the current set of nodes.

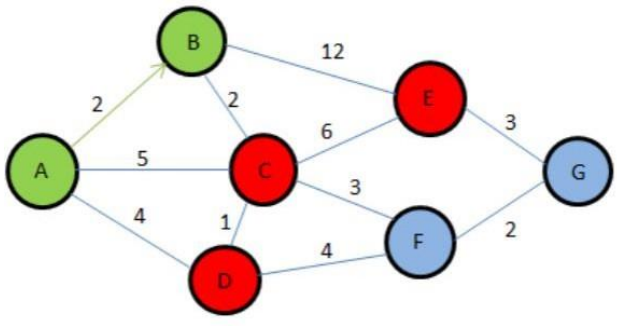


Figure 4

Accessible nodes from nodes A and B in the current nodes set are given in Figure 4. These are "C, D and E" nodes.

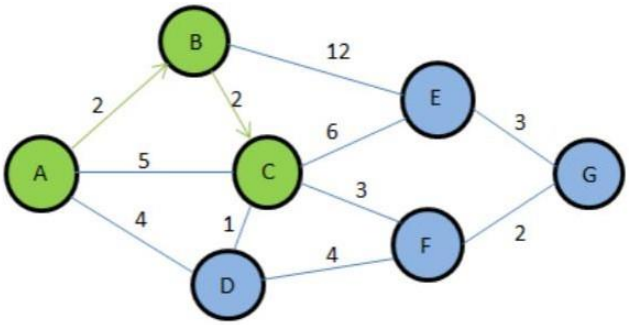


Figure 5

Shortest path from start to set of accessible nodes from current set of nodes is between B-C nodes. Node C is included in the current set of nodes.

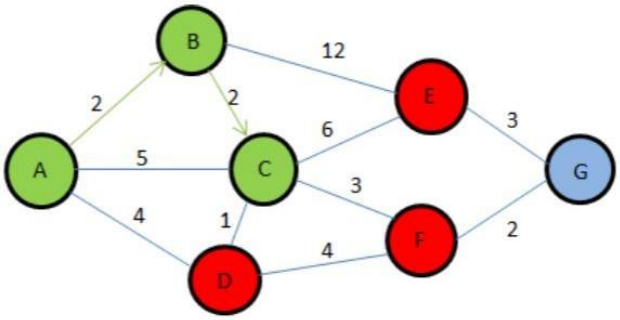


Figure 6

In Figure 6, the current set of nodes consists of "A, B and C", the accessible set of nodes consists of "D, E and F"

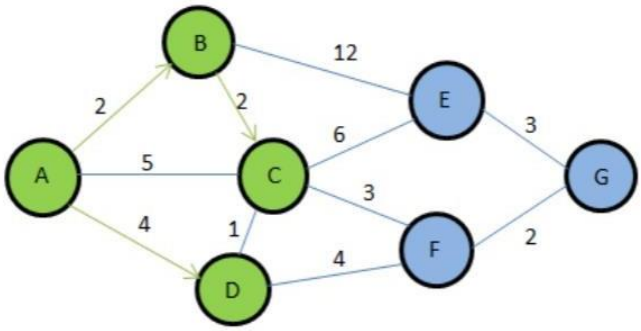


Figure 7

In Figure 7, the shortest path from the starting node A to the set of accessible nodes is between A-D. Node D has been added to the current set of nodes.

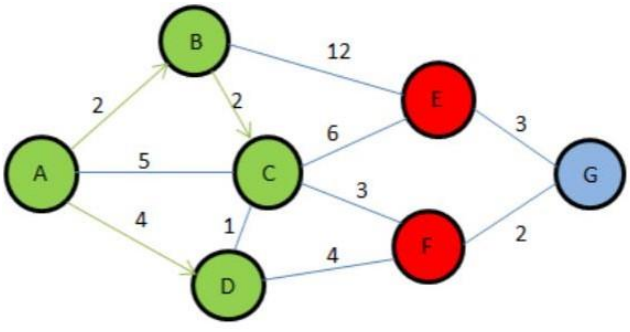


Figure 8

The set of current nodes consists of "A, B, C and D", the set of accessible nodes consists of "E and F".

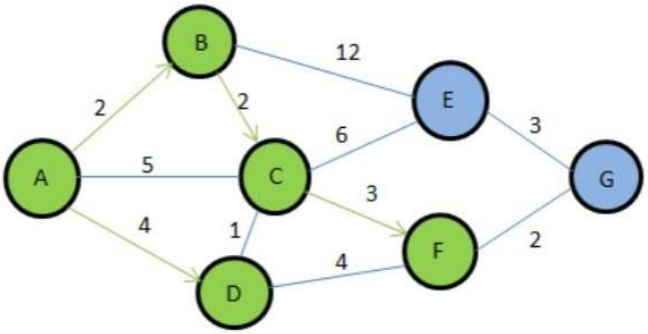


Figure 9

Node F has been added to the current set of nodes.

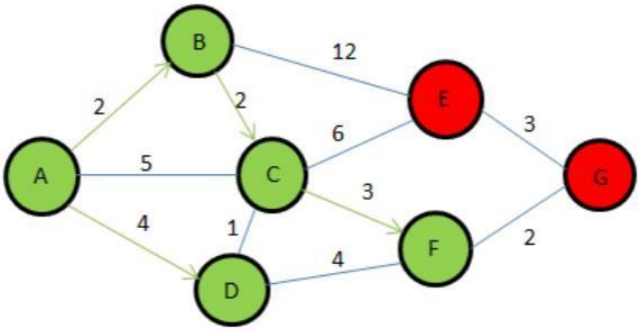


Figure 10

The set of current nodes consists of "A, B, C, D and F", the set of accessible nodes consists of "E and G".

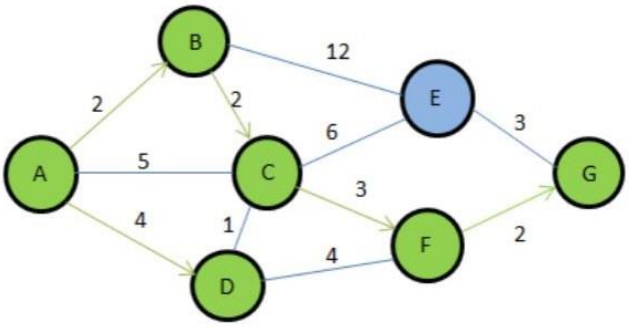


Figure 11

Shortest path from the current set of nodes to the set of accessible nodes is between F-G node. Node G has been added to the current set of nodes. The algorithm was terminated because the target node G was reached.

The edges stored on the way from the starting point, node A, to the targeted node G are indicated by arrows. The shortest path is A-B-C-F-G, the total distance is 9 units. The A-D edge is stored. However, this route was not used because there is no stored edge that can be accessed from node D to node G. The table below contains the current and accessible set of nodes in each step and the stored edge and the total distance information from the starting point to the end of the stored edge:

Current Set of Nodes	Set of Accessible Nodes	Stored Edge	Total Distance
A	B, C, D	(A,B)	2
A, B	C, D, E	(B,C)	2+2=4
A, B, C	D, E, F	(A,D)	4
A, B, C, D	E, F	(C,F)	4+3=7
A, B, C, D, F	E, G	(F,G)	7+2=9

PART IV – DESCRIPTION OUR PROJECT,PURPOSE - WHY WE CHOSE IT?

The project was made to generate a road map showing the shortest way to go from Ankara, an industrial city, to the most famous holiday cities in Turkey for having a pleasant holiday. After settling the ideas, a method to help our aim was investigated and we understood that Dijkstra Algorithm is a method to create the shortest paths.

Why we chose it?

Actually, the main idea we had while selecting this Project was that it seemed more logical and informative to use an algorithm to design a code for us. In addition, we learned that the Dijkstra algorithm is frequently used in devices such as navigation and GPS that are in the fields of Geomatics, and it drew our attention to have a connection with our professions. For all these reasons, we decided to do this project.

PART V – ANALYSIS OF THE PROJECT

```
import sys

cities = ['Ankara','Balıkesir','Muğla','Antalya','İzmir','Çanakkale']

class Graph:

    def ShortCut(self,distance,queue):

        min = sys.maxsize
        min_index = -1

        for i in range(len(distance)):
            if distance[i] < min and i in queue:
                min = distance[i]
                min_index = i
        return min_index
```

We defined the array in the name of cities and took first step of our small navigation (road map) project. Then we created class for a directed chart that uses the adjacent matrix notation and have defined a function to find the point with the minimum dist value of the set in the queue. We reference the minimum valuse and min_index as -1 for the next value and got the minimum value from the distance array, at the same time we made a value in the queue.

```
def shortestPath(self, parent, j):
    if parent[j] == -1 :
        print(cities[j],end='-'),
        return
    self.shortestPath(parent , parent[j])
    print (cities[j],end='-'),

def Solution(self, distance, parent):
    src = 0
    print("============================================================================""\n  \
    Destination\t\tDistance to Destination (KM)\t The Route Being Followed")
    for i in range(1, len(distance)):
        print("\n%s --> %s \t\t\t%s \t\t\t\t" % (cities[src], cities[i], distance[i]),end=''),
        self.shortestPath(parent,i)


def dijkstra(self, graph, source):

    rows = len(graph)
    column = len(graph[0])
    source = cities.index(source)
```

Using the array we created, we reached the shortest path from source to j and printed it. We printed the distance series created for neighboring cities. Using the adjacent matrix representation, we applied Dijkstra's algorithm to initially exit a single point and reach the destination with the shortest path. Then we defined the variables row, column, source in the matrix chart we created.

```
        dist = [sys.maxsize] * rows

        parent = [-1] * rows

        dist[source] = 0

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

        while queue:
            a = self.ShortCut(dist,queue)

            queue.remove(a)
```

dist [i] will keep the shortest distance from src to i. We initialize all distances with the largest value and we created a main array to store the shortest path tree. The distance from the source peak itself always has to be 0. We made the queque array to be able to append all destinations. Later we generated a for loop and append all the rows into the array we opened. In order for Dijkstra's algorithm to quickly find the shortest path, we created a "while queue" loop. Next, we chose the minimum point for destination remaining in the queue. Then, we removed min destinations. We refreshed the remote value and main index of adjacent corners of the selected target, considering corners that are in the queue.

```
            for i in range(column):

                if graph[a][i] and i in queue:
                    if dist[a] + graph[a][i] < dist[i]:
                        dist[i] = dist[a] + graph[a][i]
                        parent[i] = a

        self.Solution(dist,parent)

s_p= Graph()

matrix_g = [[0, 200, 500, 100, 0, 0],
        [200, 0, 300, 200, 0, 0],
        [500, 300, 0, 300, 100, 500],
        [100, 200, 300, 0, 100, 0],
        [0, 0, 100, 100, 0, 200],
        [0, 0, 500, 0, 200, 0]]

s_p.dijkstra(matrix_g,str(input("Which City Are You In Right Now?: ")))
```

After that, we stipulated with the if() condition and if dist(i) is in the tail and there is an edge a to i, also if the sum of the path from source to i and a is less than the current value of dist(i) we fulfilled the function. As you see in the picture above, we put neighbor distances to function. We have defined the s_p variable for functions defined in def dijkstra. And we have defined the ways the algorithm will use as a matrix. While printing the result, we let us enter the starting point by using input(). Finally, we printed solution.

Sample output analysis:

Output1:

```
Which City Are You In Right Now?: Ankara
=================================================================
          Destination       Distance to Destination (KM)     The Route Being Followed

Ankara --> Balıkesir            200                 Ankara-Balıkesir-
Ankara --> Muğla            300                 Ankara-Antalya-İzmir-Muğla-
Ankara --> Antalya          100                 Ankara-Antalya-
Ankara --> İzmir            200                 Ankara-Antalya-İzmir-
Ankara --> Çanakkale            400                 Ankara-Antalya-İzmir-Çanakkale-
Process finished with exit code 0
```

When we enter the input as Ankara, there is the shortest distance of the cities we can go to. For example, our goal is to go from Ankara to Çanakkale. We see that the route to be followed to go to Çanakkale is Ankara-Antalya-İzmir-Çanakkale and the total road distance is 400 km.

Output 2:

```
Which City Are You In Right Now?: Antalya
=====================================================================
          Destination       Distance to Destination (KM)     The Route Being Followed

Ankara --> Balıkesir            200                Antalya-Balıkesir-
Ankara --> Muğla            200                Antalya-İzmir-Muğla-
Ankara --> Antalya          0               Antalya-
Ankara --> İzmir            100                Antalya-İzmir-
Ankara --> Çanakkale            300                Antalya-İzmir-Çanakkale-
Process finished with exit code 0
```

Let's say we lost our route in Antalya while we were on this route. In this case, if we create a route from the city we are in, it will recalculate the shortest distance and route of our remaining road to go to Çanakkale.

Output 3:

```
Which City Are You In Right Now?: Muğla
=====================================================================
          Destination       Distance to Destination (KM)     The Route Being Followed

Ankara --> Balıkesir            300                Muğla-Balıkesir-
Ankara --> Muğla            0               Muğla-
Ankara --> Antalya          200                Muğla-İzmir-Antalya-
Ankara --> İzmir            100                Muğla-İzmir-
Ankara --> Çanakkale            300                Muğla-İzmir-Çanakkale-
Process finished with exit code 0
```

In Output 1, we see that the Ankara-Çanakkale route is respectively Ankara-Antalya-İzmir-Çanakkale. If we cannot follow this route and have to create a route from a city outside the route, we can see that the program recalculates the shortest route from the city we are in, regardless of the route, as in the output above.

****

While doing this project, we learned the usage areas and working principle of the Dijkstra algorithm. We can say that our code writing, algorithm interpretation and teamwork skills have improved during the project under construction phase. During this process, we held online meetings at regular intervals and supported each other by screen sharing rather than individual work. As a result, we created a team project.