Shri Ramdeobaba College of Engineering &Management, Nagpur

Project on "To Find Shortest Path Between Cities Using python"

This project report is submitted to

Shree Ramdeobaba College of Engineering &Management, Nagpur

In partial fulfillment of the requirement

For the award of the degree

Of

Bachelor of Engineering in Electrical Engineering

By

Avishkar Rajurkar (23)

Devesh Mishra (24)

(**VI Sem. B.E. (EE)**

Year: 2021-2022

Under the guidance of

Dr. Gaurav Goyal

ACKNOWLEDGEMENT

We take this opportunity to express a deep sense of gratitude towards our guide Professor **Dr. Gaurav Goyal** for providing excellent guidance, encouragement, and inspiration throughout the project work. Without her valuable guidance, this work would never have been a successful one.

We would like to thank **Dr. Sanjay Bodhke**, Head of the department of Electrical Engineering.
We would also like to thank all our classmates for their valuable suggestions and helpful discussions and our thanks goes to all the people who have supported us to complete work directly or indirectly.

Finally, we are extremely grateful to our parents for their love, prayers, care and sacrifices for educating and preparing us for our future.

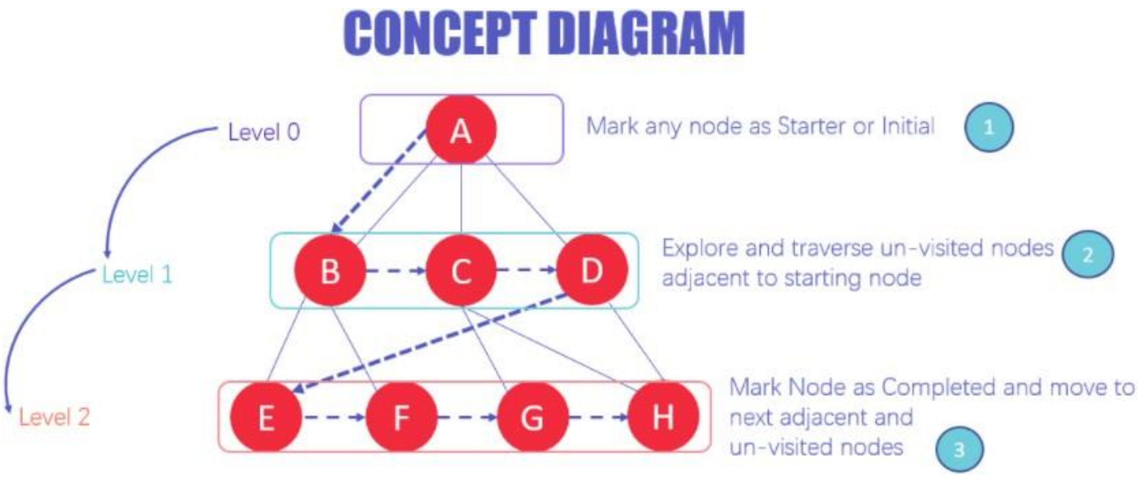
Certificate

This is to certify that **Avishkar Rajurkar, Devesh Mishra** of Electrical Engineering (VI Sem) from Shree Ramdeobaba College of Engineering & Management Nagpur has completed project work during academic session 2021-2022 and there work has been satisfactory.

Signature of	Signature of
Head of Department	Project Guide

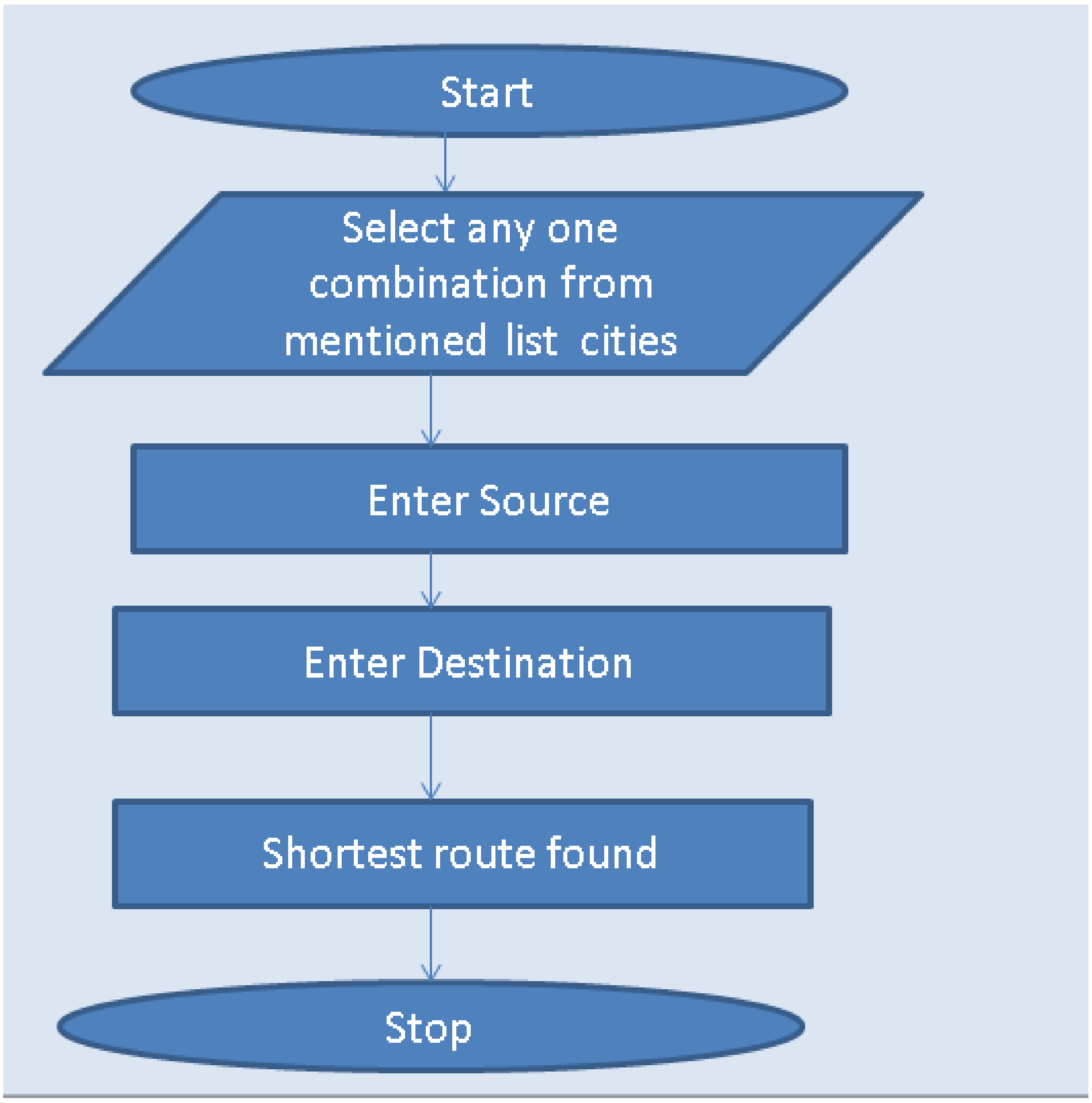
Introduction

“In the era of technology, people are forced to be a technologically aware person. This rapid development has given many benefits to the human kind especially for getting information. Shortest Path problems are inevitable in road network applications such as city emergency handling and drive guiding system, in where the optimal routings have to be found. As the traffic condition among a city change from time to time and there are usually a huge amount of request occur at any moment, it needs to quickly find the solution. Therefore, the efficiency of the algorithm is very important Some approaches take advantage of preprocessing that compute results before demanding. These results are saved in memory and could be used directly when a new request comes up. This can be inapplicable if the devices have limited memory and external storage. This project aims only to find shortest path between the cities”.


CONCEPT DIAGRAM
Level 0
Level 1
Level 2
A
B
C
D
E
F
G
H
Mark any node as Starter or Initial
1
Explore and traverse un-visited nodes adjacent to starting node
2
Mark Node as Completed and move to next adjacent and un-visited nodes
3

“This project focuses on finding shortest path among cities by repeatedly combining the start nodes nearest neighbour to implement Dijkstra algorithm. The successful implementation in finding the shortest path on the real problem is a good point therefore the project can be developed to solve transportation network problem”.

FLOWCHART


Start
Select any one combination from mentioned list cities
Enter Source
Enter Destination
Shortest route found
Stop

THEORY

In this project the path finding algorithm is implemented which basically, works on the principle of making a graph node and using weight distance to update the unknow distance between two nodes and thus finding the shortest path by comparing the distance.

In order to understand the algorithm principle, we have to understand how this code implements on a graph for finding the distance it uses the uninformed search technique which is used to find most optimistic and minimum path for the provided node.

Now let us understand how does it creates the path. In this type of algorithm in order to find path it explores all the node once it is provided with source and destination it will search all possible paths and once done with all exploration it will provide with the shortest path solution.

Let us understand the concept of shortest path and search algorithm techniques

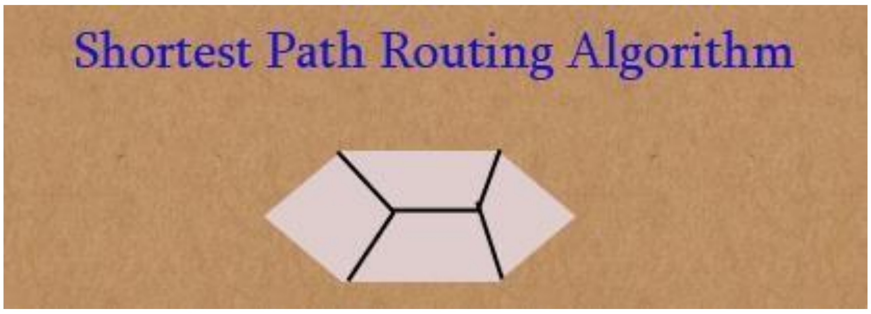
What is path finding algorithms?

Path finding algorithms build on top of graph search algorithms and explore routes between nodes, starting at one node and traversing through relationships until the destination has been reached. These algorithms find the cheapest path in terms of the number of hops or weight. Weights can be anything measured, such as time, distance, capacity, or cost.

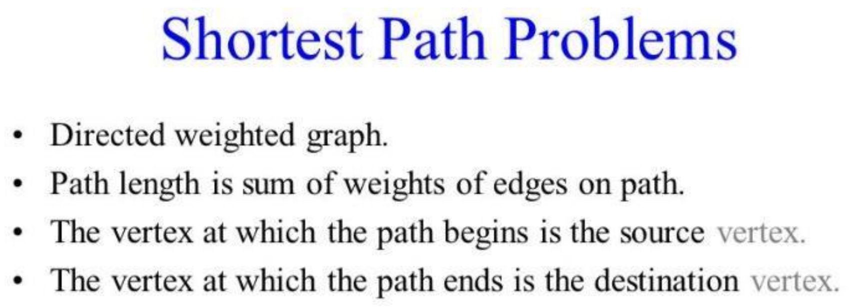
Shortest Path

The Shortest Path algorithm calculates the shortest (weighted) path between a pair of nodes. Shortest path is considered to be one of the classical graph problems and has been researched as far back as the 19th century. It has the following use cases:

- Finding directions between physical locations. This is the most common usage, and web mapping tools such as Google Maps use the shortest path algorithm, or a variant of it, to provide driving directions.
- Social networks can use the algorithm to find the degrees of separation between people. For example, when you view someone's profile on LinkedIn, it will indicate how many people separate you in the connections graph, as well as listing your mutual connections.

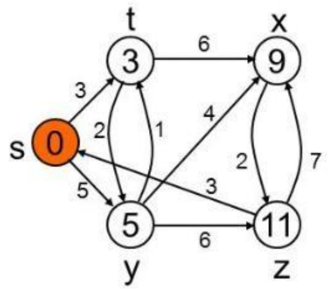


Single Source Shortest Paths



Shortest Path Problems

- Directed weighted graph.
- Path length is sum of weights of edges on path.
- The vertex at which the path begins is the source vertex.
- The vertex at which the path ends is the destination vertex.



In a shortest- paths problem, we are given a weighted, directed graphs G = (V, E), with weight function w: E → R mapping edges to real-valued weights. The weight of path p = (v0,v1,..... vk) is the total of the weights of its constituent edges:

Single Source Shortest Paths

We define the shortest - path weight from u to v by δ(u,v) = min (w (p): u→v), if there is a path from u to v, and δ(u,v)= ∞, otherwise.

The shortest path from vertex s to vertex t is then defined as any path p with weight w (p) = δ(s,t).

The breadth-first- search algorithm is the shortest path algorithm that works on unweighted graphs, that is, graphs in which each edge can be considered to have unit weight.

In a Single Source Shortest Paths Problem, we are given a Graph G = (V, E), we want to find the shortest path from a given source vertex s ∈ V to every vertex v ∈ V.

Variants:

There are some variants of the shortest path problem.

Single- destination shortest - paths problem: Find the shortest path to a given destination vertex t from every vertex v. By shift the direction of each edge in the graph, we can shorten this problem to a single source problem.

Single - pair shortest - path problem: Find the shortest path from u to v for given vertices u and v. If we determine the single - source problem with source vertex u, we clarify this problem also. Furthermore, no algorithms for this problem are known that run asymptotically faster than the best single - source algorithms in the worst case.

All - pairs shortest - paths problem: Find the shortest path from u to v for every pair of vertices u and v. Running a single - source algorithm once from each vertex can clarify this problem; but it can generally be solved faster, and its structure is of interest in the own right.

The most important algorithms for solving shortest path problems are:

Breadth-first search and depth-first search- This refer to different search orders; for depth-first search, instances can be found where their naive implementation does not find an optimal solution, or does not terminate.

Dijkstra's algorithm-It solves the Single-Source Shortest Path problem if all edge weights are greater than or equal to zero. Without worsening the runtime complexity, this algorithm can in fact compute the shortest paths from a given start point s to all other nodes.

The Bellman-Ford algorithm -It also solves the Single-Source Shortest Paths problem, but in contrast to Dijkstra's algorithm, edge weights may be negative.

The Floyd-Warshall algorithm -It solves the all--Pairs Shortest Paths problem**.**

The A* algorithm- It solves the Single-Source Shortest Path problem for nonnegative edge costs.

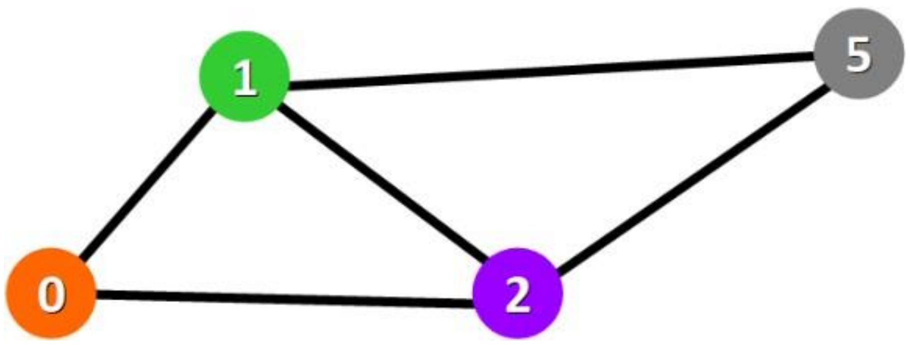
For this project we have used and implemented DJIKSTRA'S ALGORITHM for finding the search of optimal path.
Before we learn about djikstra we need to understand the basic concept of graph

GRAPH

Basic Concepts

Graphs are data structures used to represent "connections" between pairs of elements.

- These elements are called **nodes**. They represent real-life objects, persons, or entities.
- The connections between nodes are called **edges**.



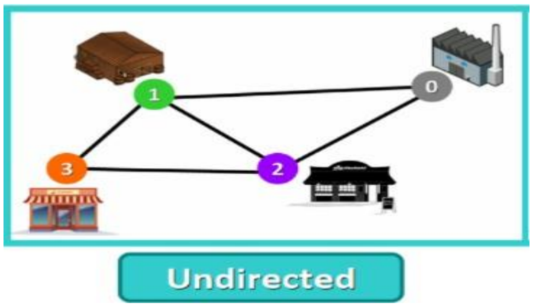
APPLICATIONS

Graphs are directly applicable to real-world scenarios. For example, we could use graphs to model a transportation network where nodes would represent facilities that send or receive products and edges would represent roads or paths that connect them

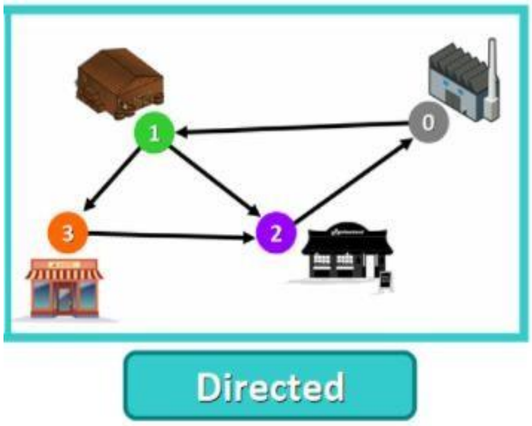
Types of Graphs

Graphs can be:

- **Undirected:** if for every pair of connected nodes, you can go from one node to the other in both directions.

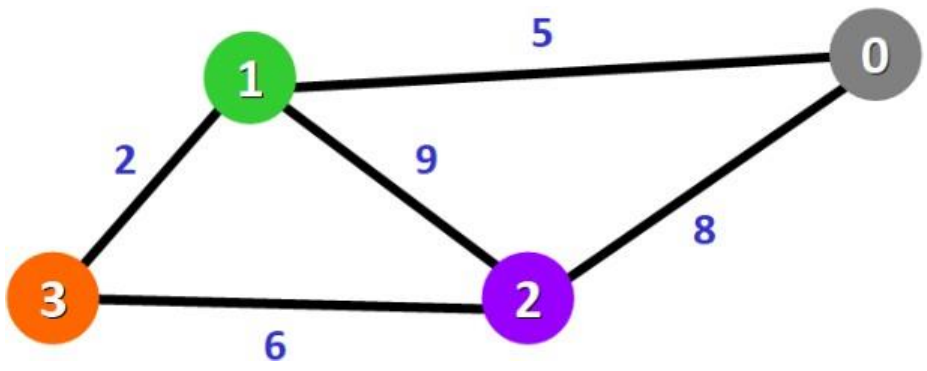


Directed: if for every pair of connected nodes, you can only go from one node to another in a specific direction. We use arrows instead of simple lines to represent directed edges.



Weighted Graphs

A **weight graph** is a graph whose edges have a "weight" or "cost". The weight of an edge can represent distance, time, or anything that models the "connection" between the pair of nodes it connects.
For example, in the weighted graph below you can see a blue number next to each edge.
This number is used to represent the weight of the corresponding edge

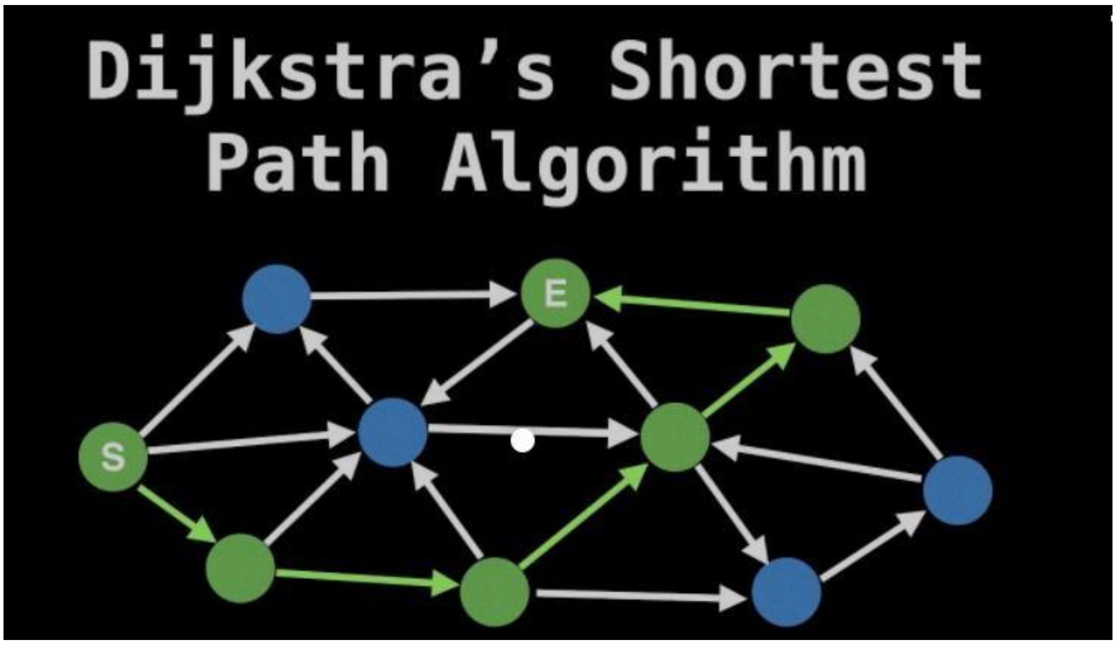


Let us learn more about the djikstra’s algorithm and it’s application

DJIKSTRA'S ALGORITHM

It was conceived by computer scientist **Edsger W. Dijkstra** in 1956 and published three years later.

Dijkstra algorithm is a single-source shortest path algorithm. Here, single-source means that only one source is given, and we have to find the shortest path from the source to all the nodes.



THE NODE COMBINATION BASED ON DIJKSTRA ALGORITHM

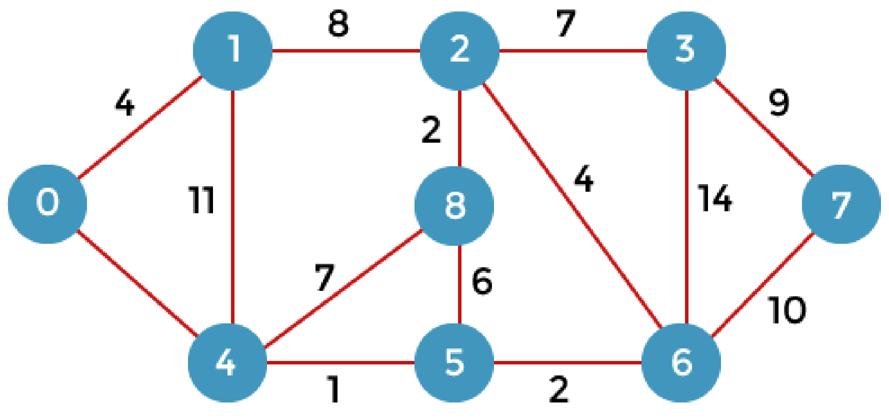
Basic idea of the node combination is to combine nodes instead of maintaining the labelling sets in Dijkstra algorithm. The node combination method finds the shortest paths iteratively by finding the nearest neighbour of the start node, combining that node with the start and updates the edge weights connected to the nearest node.

The steps of the node combination method are as follows:

The steps of the node combination method are as follows:

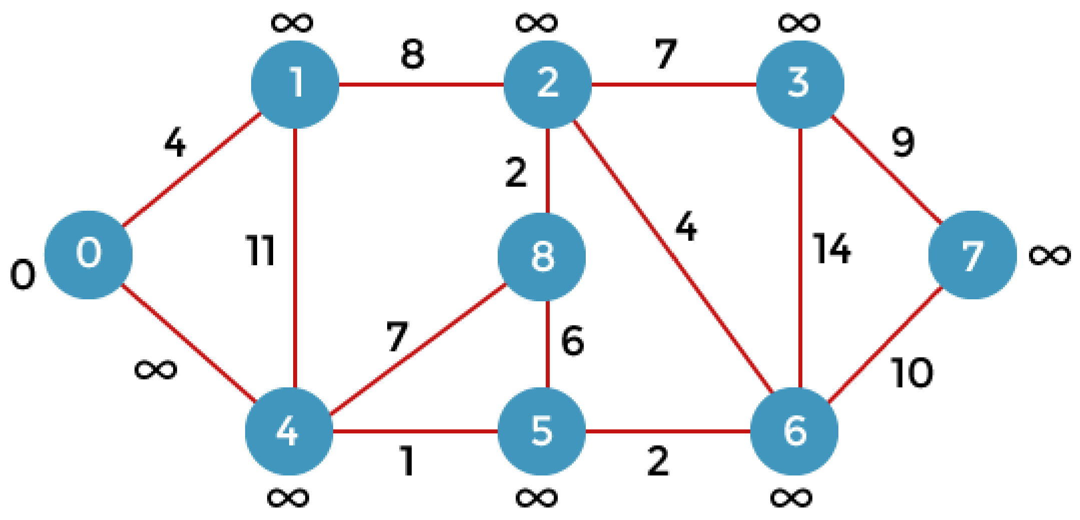
- Determine the start node
- Find the nearest node by looking at the smallest weights that are connected with the start node
- Update the edge weights that are connected with the start node. If there are two or more weights then choose the smallest one
- Repeat the steps until only two nodes remains.

Let's understand the working of Dijkstra's algorithm.
Consider the given undirected graph below .



First, we have to consider any vertex as a source vertex. Suppose we consider vertex 0 as a source vertex.

Here we assume that 0 as a source vertex, and distance to all the other vertices is infinity. Initially, we do not know the distances. First, we will find out the vertices which are directly connected to the vertex 0. As we can observe in the above graph that two vertices are directly connected to vertex 0.



Let's assume that the vertex 0 is represented by 'x' and the vertex 1 is represented by 'y'. The distance between the vertices can be calculated by using the below formula:

d(x, y) = d(x) + c(x, y) < d(y)

= **(0 + 4)** < ∞

= **4** < ∞

Since 4<∞ so we will update d(v) from ∞ to 4.

Therefore, we come to the conclusion that the formula for calculating the distance between the vertices:

{if(d(u) + c(u, v) < d(v))

d(v) = d(u) +c(u, v) }

Now we consider vertex 0 same as 'x' and vertex 4 as 'y'.

d(x, y) = d(x) + c(x, y) < d(y)

= **(0 + 8)** < ∞

= **8** < ∞

Therefore, the value of d(y) is 8. We replace the infinity value of vertices 1 and 4 with the values 4 and 8 respectively. Now, we have found the shortest path from the vertex 0 to 1 and 0 to 4. Therefore, vertex 0 is selected. Now, we will compare all the vertices except the vertex 0. Since vertex 1 has the lowest value, i.e., 4; therefore, vertex 1 is selected.

Since vertex 1 is selected, so we consider the path from 1 to 2, and 1 to 4. We will not consider the path from 1 to 0 as the vertex 0 is already selected.

First, we calculate the distance between the vertex 1 and 2. Consider the vertex 1 as 'x', and the vertex 2 as 'y'.

d(x, y) = d(x) + c(x, y) < d(y)

= **(4 + 8)** < ∞

= **12** < ∞

Since 12<∞ so we will update d(2) from ∞ to 12.

Now, we calculate the distance between the vertex 1 and vertex 4. Consider the vertex 1 as 'x' and the vertex 4 as 'y'.

d(x, y) = d(x) + c(x, y) < d(y)

= (4 + 11) < 8

= 15 < 8

Since 15 is not less than 8, we will not update the value d(4) from 8 to 12.

Till now, two nodes have been selected, i.e., 0 and 1. Now we have to compare the nodes except the node 0 and 1. The node 4 has the minimum distance, i.e., 8. Therefore, vertex 4 is selected.

Since vertex 4 is selected, so we will consider all the direct paths from the vertex 4. The direct paths from vertex 4 are 4 to 0, 4 to 1, 4 to 8, and 4 to 5. Since the vertices 0 and 1 have already been selected so we will not consider the vertices 0 and 1. We will consider only two vertices, i.e., 8 and 5.

First, we consider the vertex 8. First, we calculate the distance between the vertex 4 and 8. Consider the vertex 4 as 'x', and the vertex 8 as 'y'.

d(x, y) = d(x) + c(x, y) < d(y)

= (8 + 7) < ∞

= **15** < ∞

Since 15 is less than the infinity so we update d(8) from infinity to 15.

Now, we consider the vertex 5. First, we calculate the distance between the vertex 4 and 5. Consider the vertex 4 as 'x', and the vertex 5 as 'y'.

d(x, y) = d(x) + c(x, y) < d(y)

= **(8 + 1)** < ∞

= **9** < ∞

Since 5 is less than the infinity, we update d(5) from infinity to 9.

Till now, three nodes have been selected, i.e., 0, 1, and 4. Now we have to compare the nodes except the nodes 0, 1 and 4. The node 5 has the minimum value, i.e., 9. Therefore, vertex 5 is selected.

Since the vertex 5 is selected, so we will consider all the direct paths from vertex 5. The direct paths from vertex 5 are 5 to 8, and 5 to 6.

First, we consider the vertex 8. First, we calculate the distance between the vertex 5 and 8. Consider the vertex 5 as 'x', and the vertex 8 as 'y'.

d(x, y) = d(x) + c(x, y) < d(y)

= (9 + 15) < 15

= 24 < 15

Since 24 is not less than 15 so we will not update the value d(8) from 15 to 24.

Now, we consider the vertex 6. First, we calculate the distance between the vertex 5 and 6. Consider the vertex 5 as 'x', and the vertex 6 as 'y'.

d(x, y) = d(x) + c(x, y) < d(y)

= **(9 + 2)** < ∞

= **11** < ∞

Since 11 is less than infinity, we update d(6) from infinity to 11.

Till now, nodes 0, 1, 4 and 5 have been selected. We will compare the nodes except the selected nodes. The node 6 has the lowest value as compared to other nodes. Therefore, vertex 6 is selected.

Since vertex 6 is selected, we consider all the direct paths from vertex 6. The direct paths from vertex 6 are 6 to 2, 6 to 3, and 6 to 7.

First, we consider the vertex 2. Consider the vertex 6 as 'x', and the vertex 2 as 'y'. d(x, y) = d(x) + c(x, y) < d(y)

= (11 + 4) < 12

= 15 < 12

Since 15 is not less than 12, we will not update d(2) from 12 to 15

Now we consider the vertex 3. Consider the vertex 6 as 'x', and the vertex 3 as 'y'.

d(x, y) = d(x) + c(x, y) < d(y)

= (11 + 14) < ∞

= 25 < ∞

Since 25 is less than ∞, so we will update d(3) from ∞ to 25.

Now we consider the vertex 7. Consider the vertex 6 as 'x', and the vertex 7 as 'y'.

d(x, y) = d(x) + c(x, y) < d(y)

= (11 + 10) < ∞

= 22 < ∞

Since 22 is less than ∞ so, we will update d(7) from ∞ to 22.

Till now, nodes 0, 1, 4, 5, and 6 have been selected. Now we have to compare all the unvisited nodes, i.e., 2, 3, 7, and 8. Since node 2 has the minimum value, i.e., 12 among all the other unvisited nodes. Therefore, node 2 is selected.

Since node 2 is selected, so we consider all the direct paths from node 2. The direct paths from node 2 are 2 to 8, 2 to 6, and 2 to 3.

First, we consider the vertex 8. Consider the vertex 2 as 'x' and 8 as 'y'.

d(x, y) = d(x) + c(x, y) < d(y)

= (12 + 2) < 15

= 14 < 15

Since 14 is less than 15, we will update d(8) from 15 to 14.

Now, we consider the vertex 6. Consider the vertex 2 as 'x' and 6 as 'y'.

d(x, y) = d(x) + c(x, y) < d(y)

= (12 + 4) < 11

= 16 < 11

Since 16 is not less than 11 so we will not update d(6) from 11 to 16.

Now, we consider the vertex 3. Consider the vertex 2 as 'x' and 3 as 'y'.

d(x, y) = d(x) + c(x, y) < d(y)

= (12 + 7) < 25

= 19 < 25

Since 19 is less than 25, we will update d(3) from 25 to 19.

Till now, nodes 0, 1, 2, 4, 5, and 6 have been selected. We compare all the unvisited nodes, i.e., 3, 7, and 8. Among nodes 3, 7, and 8, node 8 has the minimum value. The nodes which are directly connected to node 8 are 2, 4, and 5. Since all the directly connected nodes are selected so we will not consider any node for the updation.

The unvisited nodes are 3 and 7. Among the nodes 3 and 7, node 3 has the minimum value, i.e., 19. Therefore, the node 3 is selected. The nodes which are directly connected to the node 3 are 2, 6, and 7. Since the nodes 2 and 6 have been selected so we will consider these two nodes.

Now, we consider the vertex 7. Consider the vertex 3 as 'x' and 7 as 'y'.

d(x, y) = d(x) + c(x, y) < d(y)

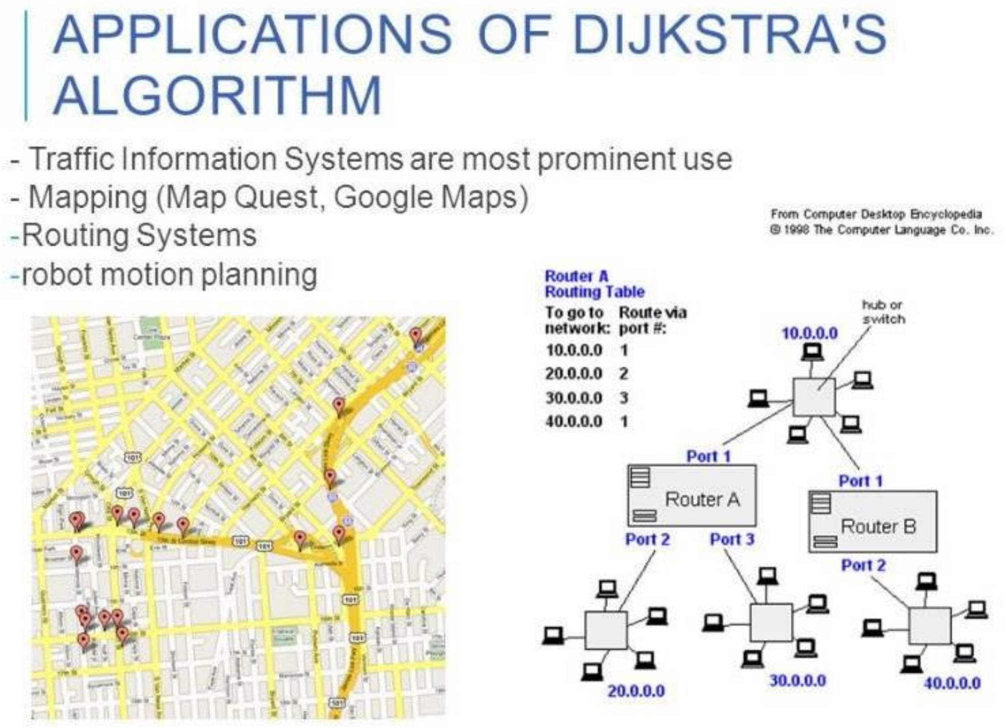
= (19 + 9) < 21

= 28 < 21

Since 28 is not less than 21, so we will not update d(7) from 28 to 21.

With this example we can understand how the optimal solution can be calculated to find the path

APPLICATION



APPLICATIONS OF DIJKSTRA'S ALGORITHM

- Traffic Information Systems are most prominent use
- Mapping (Map Quest, Google Maps)
-Routing Systems
-robot motion planning

From Computer Desktop Encyclopedia
© 1998 The Computer Language Co. Inc.

Router A
Routing Table

To go to network:	Route via port #:
10.0.0.0	1
20.0.0.0	2
30.0.0.0	3
40.0.0.0	1

hub or switch
10.0.0.0
Port 1
Router A
Port 2
Port 3
Port 1
Router B
Port 2
20.0.0.0
30.0.0.0
40.0.0.0

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. Consider India as a graph and represent a city/place with a vertex and the route between two cities/places as an edge, then by using this algorithm, the shortest routes between any two cities/places or from one city/place to another city/place can be calculated.

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.

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.

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.

Advantages

Many benefits can be obtained when knowing the shortest or fastest route in a travel route, as an example, can **save travel time, power, less fuel consumption, and the density of vehicles on certain segments can decompose**

ADVANTAGE OF LOW TIME COMPLEXITY

The main advantage of Dijkstra's algorithm is its considerably low complexity, which is almost linear. However, when working with negative weights, Dijkstra's algorithm can't be used.

Also, when working with dense graphs, where E is close to V^2, if we need to calculate the shortest path between any pair of nodes, using Dijkstra's algorithm is not a good option.

The reason for this is that Dijkstra's time complexity is O(V + E \cdot log(V)). Since E equals almost V^2, the complexity becomes O(V + V^2 \cdot log(V)).

When we need to calculate the shortest path between every pair of nodes, we'll need to call Dijkstra's algorithm, starting from each node inside the graph. Therefore, the total complexity will become O(V^2 + V^3 \cdot log(V)).

The reason why this is not a good enough complexity is that the same can be calculated using the Floyd-Warshall algorithm, which has a time complexity of O(V^3). Hence, it can give the same result with lower complexity

LIMITATIONS

When working with graphs that have negative weights, Dijkstra's algorithm fails to calculate the shortest paths correctly. The reason is that Path(u, v) might be negative, which will make it possible to reach u from v at a lower cost. Therefore, we can't prove the optimality of choosing the node that has the lowest cost.

FUTURE SCOPE IN ELECTRIC VEHICLE

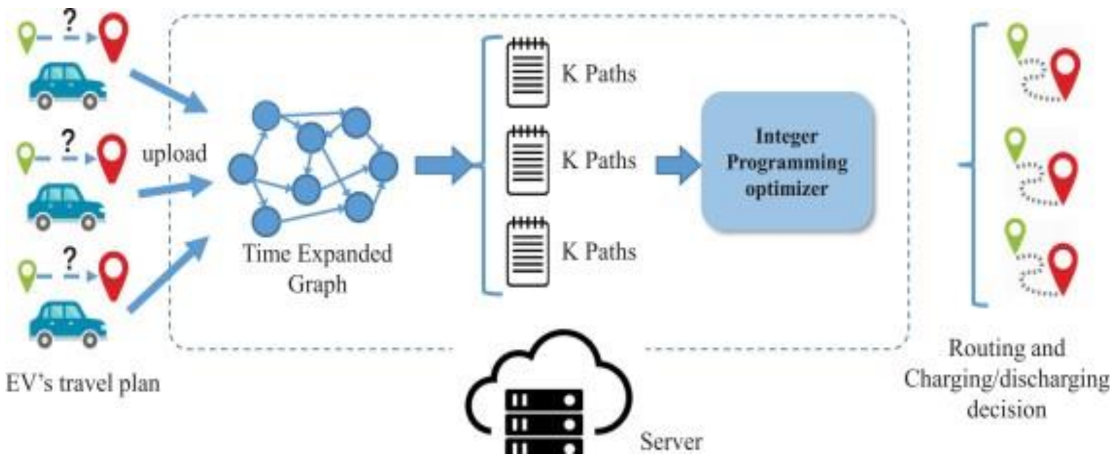
In recent years, the importance of electric vehicles as a fossil fuel independent alternative to conventional vehicles has increased steadily. However, their limited battery capacity, which leads to comparatively small cruising ranges, still presents a major obstacle towards a more widespread use. While the number of charging stations for electric vehicles is increasing, they are still fairly rare compared to conventional gas stations. Furthermore, the process of recharging the battery is time-consuming and may take several hours. This makes it especially important to plan long-distance routes with charging stops beforehand, presenting a new challenge for routing algorithms. The constraints imposed by an electric vehicle's limited battery capacity require special algorithmic handling. In particular, route planning algorithms must ensure that the vehicle never runs out of energy. Furthermore, electric vehicles can recharge their battery when driving downhill; this effect is called recuperation. However, recuperation is only possible as long as the battery does not exceed its maximum capacity. Therefore, routing algorithms for electric vehicles must keep track of the battery's state of charge to ensure that it always remains within these constraints. Previous research on the topic of electric vehicle route planning has focused mostly on finding time- or energy-optimal routes while taking battery constraints into account. Research on incorporating charging stations has been limited so far, mainly because small battery capacities and long charging times make long-distance routes impractical. However, with the advent of more powerful charging stations and vehicles with larger batteries, charging stops are becoming increasingly feasible. For the problem of finding energy-optimal routes, only battery swapping stations (BSS) have been considered as a method of recharging. These are stations where the entire battery is replaced with a new, fully charged battery. However, battery swapping stations are currently not widely available. So far, route planning algorithms that take commonly available types of charging stations into account exist only for the time-optimal electric vehicle routing problem. In this thesis, we adapt existing approaches to develop algorithms that find energy-optimal routes and can handle a variety of charging station types.

BATTERY CHARGING PLANNING IN EV's FOR ROUTING OPTIMAL PATH

In today's market the countries are still adapting the electric vehicles usage Steadily and needs a lot of improvement but most importantly one should consider charging station on each fuel station and issue of resolving it as the capacity of EV's vary differently. With the help of algorithm, we can improve its searching method by considering the charging capacity and discharging capacity of each vehicle and charging port available on each route by providing the mathematical conditions and giving all this condition as input variable we can help to reduce the routing problem by finding optimal energy path solution on basis of condition of each EV and charging station.

Targeted optimal-path problem for electric vehicles

The prototypical problem that we study is to identify the targeted optimal path for an EV from its origin to its destination. The EV may charge once or more times at CSs. Consequently, any solution path is composed of several segments, namely, subpaths. All the subpaths are limited by the battery level. The optimal path refers to the path that corresponds to the lowest time cost for both basic travel and charging operations. The targeted optimal path is identified via the proposed travel-cost method.


upload
Time Expanded Graph
K Paths
K Paths
K Paths
Integer Programming optimizer
EV's travel plan
Server
Routing and Charging/discharging decision

- Graphs are used to model connections between objects, people, or entities. They have two main elements: nodes and edges. Nodes represent objects and edges represent the connections between these objects.
- Dijkstra's Algorithm finds the shortest path between a given node (which is called the "source node") and all other nodes in a graph.
- This algorithm uses the weights of the edges to find the path that minimizes the total distance (weight) between the source node and all other nodes.
- Djikstra’s algorithm can be improved and can bring revolution in entire EV industry for finding optimal path thus saving fuels and increasing electric vehicle efficiency

WITH THIS WE END THE REPORT OF OUR MINI PROJECT

WE HOPE THAT YOU HAVE UNDERSTAND THE IMPORTANCE OF THIS ALGORITHM AND ADVANCEMNT IN IOT & AI

THANK YOU ONCE AGAIN