Dijkstra's Algorithm

CCE414-Artificial Intelligence

Team members

1)Ahmed Mohamed Fawzy. G01
ID:221903175

2) Passant El-Tonsy Ali. G01
ID:221903220

3)Saif Emad El-Den Abdel-Kareem. G02
ID:231903756

4) Mohamed Emad Fawzy. G02
ID:231903548

Under Supervision

—

Prof.Dr. Lamiaa Elrefaei.

Date Due	**May 7th ,2024.**
Date Handed in	**May 4th ,2024.**

ABSTRACT

The optimal path selection is a practical problem, and the application frequency in daily life is very high, which has practical research significance. The Dijkstra algorithm is one of the most classical algorithms in the algorithm of optimal path selection. It is also the theoretical basis of many experimental designs when solving the optimal path problem.

Dijkstra's algorithm was developed by Edsger W. Dijkstra in 1956. Dijkstra was a mathematician who greatly contributed to the field of computer science and is considered one of its founding members. He initially used it to help develop a transportation map of 64 cities in the Netherlands. Dijkstra's algorithm had one other iteration which utilized a minimum spanning tree instead of a graph. This version of the algorithm was identical to an algorithm developed by the mathematician Jarnik, who also worked in the field of computer science. Dijkstra's algorithm has been studied a great deal and has also led to other more specialized algorithms.

ACKNOWLEDGEMENT

I would like to extend my heartfelt appreciation to the individuals whose work has greatly contributed to the completion of this project. Their dedication and expertise have been invaluable in shaping the outcomes presented in this report.

We're also indebted to **Prof.Dr. Lamiaa El-Refaei** and **Eng.Mohamed Rehan** for generously sharing their expertise and providing invaluable guidance throughout the duration of this project. Their depth of knowledge and willingness to offer assistance have been instrumental in overcoming challenges and achieving our objectives.

Additionally, we extend our appreciation to the authors of the various studies, reports, and publications cited in this work. Their groundbreaking research laid the foundation for our understanding and served as a source of inspiration for our endeavours.

Lastly, we wish to thank our Parents, mentors, advisors, and supervisors for their continuous encouragement and guidance throughout this journey. Their unwavering support and mentorship have been instrumental in shaping our growth and development as researchers.

Table of Content

1- Introduction:

1.1- Directed weighted graph:

This type of graphs has edges which have a direction associated with them. This means that the relationship between nodes is one-way, indicating a flow or directionality from one node (the source) to another (the destination) and each edge in it has an associated weight or cost. This weight represents a value assigned to the edge, which could represent various things depending on the context. For example, in a transportation network, it might represent the distance between two locations, the time it takes to travel between them, or the cost associated with the journey.

Directed graph can be defined as an ordered pair G: = (V,E) with V is a set, whose elements are called vertices or nodes and E is a set of ordered pairs of vertices, called directed edges, arcs, or arrows. Directed graphs are also known as digraph.

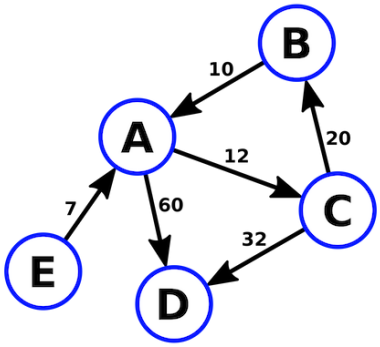

Figure1-Directed Weighted Graph

1.2- Single- Source shortest path:

Dijkstra's algorithm is used to find the shortest route between two vertices, or nodes, on a graph it is also called single-source shortest path.

Let G= {V, E} be a directed weighted graph with V having the set of vertices. The special vertex s in V, where s is the source and let for any edge e in E, Edge Cost(e) be the length of edge e. Important constraint is that all the weights in the graph should be non-negative

2- Explanation of Dijkstra's Algorithm:

This algorithm uses a greedy approach which is a problem-solving method that makes locally optimal choices at each step with the hope of finding a global optimum. They are characterized by their tendency to choose the best immediate option without necessarily considering the long-term consequences that's why it's greedy.

Dijkstra's algorithm works by solving the subproblem k, which computes the shortest path from the source to vertices among the k closest vertices to the source.

For the Dijkstra's algorithm to work it should be directed- weighted graph and the edges should be non-negative. If the edges are negative, then the actual shortest path cannot be obtained.

At the k th round, there will be a set called Frontier of k vertices that will consist of the vertices closest to the source and the vertices that lie outside frontier are computed and put into New Frontier.

The shortest distance obtained is maintained in sDist[w]. It holds the estimate of the distance from s to w.
Dijkstra's algorithm finds the next closest vertex by maintaining the New Frontier vertices in a priority-min queue.

The algorithm works by keeping the shortest distance of vertex v from the source in an array called sDist. The shortest distance of the source to itself is zero. sDist for all other vertices is set to infinity to indicate that those vertices are not yet processed.

After the algorithm finishes the processing of the vertices sDist will have the shortest distance of vertex w to s. two sets are maintained Frontier and new Frontier which helps in the processing of the algorithm.

Frontier has k vertices which are closest to the source, will have already computed shortest distances to these vertices, for paths restricted up to k vertices. The vertices that reside outside of Frontier is put in a set called New Frontier.

3- Pseudo code:

```
dijkstraShortestPath(G graph, V start)
  Set known; Map edgeTo, distTo;
  initialize distTo with all nodes mapped to ∞, except start to 0

  while (there are unknown vertices):
    let u be the closest unknown vertex
    known.add(u)
    for each edge (u,v) to unknown v with weight w:
      oldDist = distTo.get(v)      // previous best path to v
      newDist = distTo.get(u) + w  // what if we went through u?
      if (newDist < oldDist):
        distTo.put(v, newDist)
        edgeTo.put(v, u)
        update distance in list of unknown vertices
```

Figure 2- Dijkstra's Pseudo Code

- **Set "known" is similar to "visited" in BFS, it contains nodes which are finalized.**
- Initialize distTo with all nodes mapped to ∞, except start to 0.
- **Map "edgeTo" is to store edges between vertices.**
- while loop is designed to update best so far.

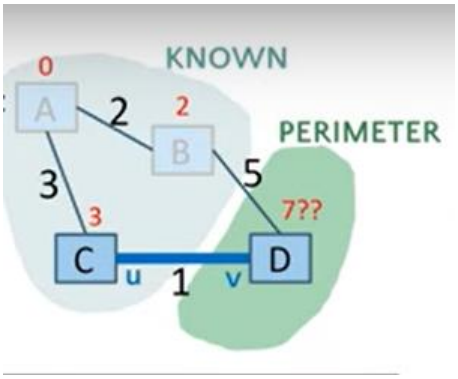

Figure 3-Simple Graph

- Suppose we already visited B, distTo[D] = 7
- Now considering edge (C, D):
  - `oldDist = 7`
  - `newDist = 3 + 1`
  - That's better! Update distTo[D], edgeTo[D]

4- Examples:

4.1- Example1 on directed graph.

Find the shortest path between node (A) and node (F) in this weighted directed graph.

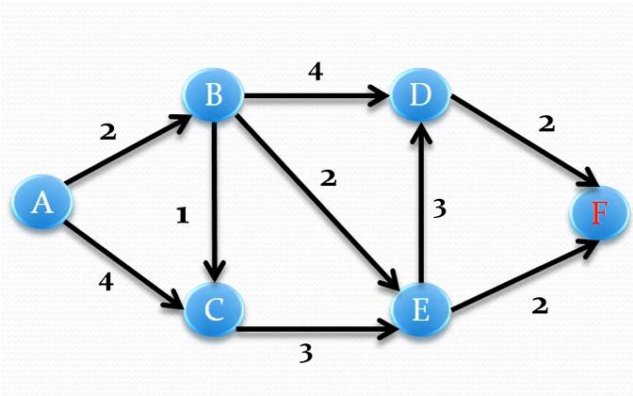

Figure 4-Example (1)

Solution:

- Make sure there's no negative edges.
- build a table for all vertices as shown in figure.

Q <= V	A	B	C	D	E	F

Figure 5-Table of Vertices

- set distance to source vertex to zero and other vertices to infinity.
- If distance of current vertex+ distance adjacent edge is less than distance of adjacent, then update distance with the new value.

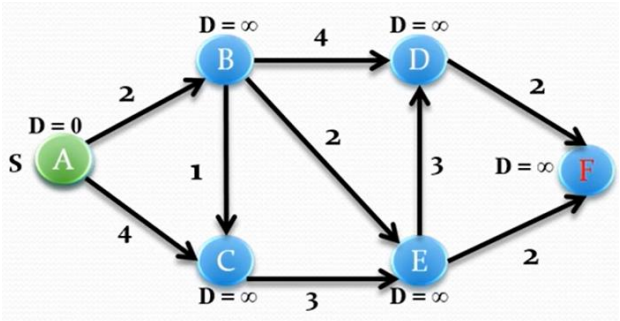

Figure 6-Node A to itself

- Assign values to tables according to the following figures:
  -Vertex (A):

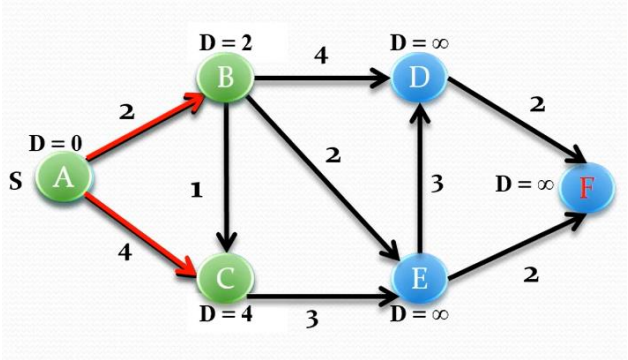

Figure 7-Expanding A

Q <= V	A	B	C	D	E	F
A	0^A	2^A	4^A	∞^A	∞^A	∞^A

Figure 8-Table for A Node

-Vertex (B) which has the least edge value:

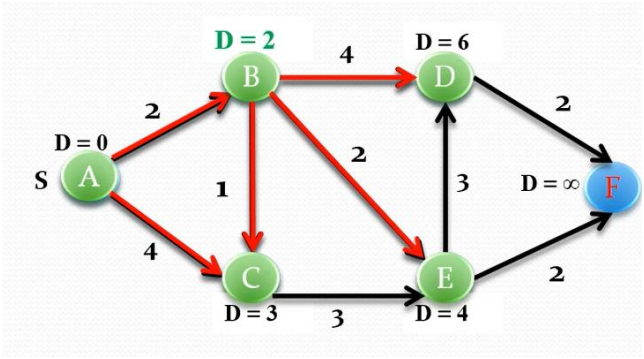

Figure 9-Expanding B

Q <= V	A	B	C	D	E	F
A	0^A	2^A	4^A	∞^A	∞^A	∞^A
B	0^A	2^A	3^B	6^B	4^B	∞^A

Figure 10-Table for A&B Node

-Go to vertex (C) then continue tell you fill the whole table:

Q <= V	A	B	C	D	E	F
A	0^A	2^A	4^A	∞^A	∞^A	∞^A
B	0^A	2^A	3^B	6^B	4^B	∞^A
C	0^A	2^A	3^B	6^B	4^B	∞^A
E	0^A	2^A	3^B	6^B	4^B	6^E
D	0^A	2^A	3^B	6^B	4^B	6^E
F	0^A	2^A	3^B	6^B	4^B	6^E

Figure 11-Complete Matrix

-The shortest path from (A) to (F) is: ABEF

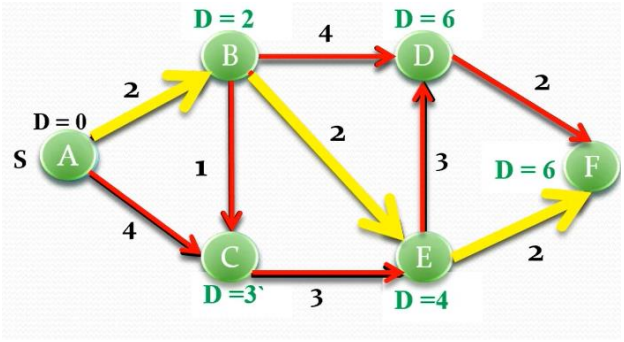

Figure 12-Shortest Path

4.2- Example2 on undirected graph:

Build an Adjacency matrix to find the shortest path between node (A) and other nodes (vertices).

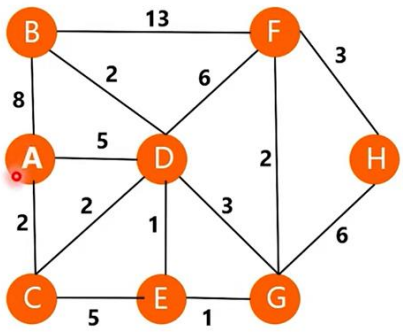

Figure 13-Example 2

Solution:

- **First you have to build an adjacency matrix by assigning (-) or (0) to visited vertices and between each node and itself.**
- **Assign (∞) for nodes which don't have direct path.**

	A	B	C	D	E	F	G	H
A	-	8A	2A	5A	∞	∞	∞	∞
C	-	8A	-	4C	7C	∞	∞	∞
D	-	6D	-	-	5D	10D	7D	∞
E	-	6D	-	-	-	10D	6E	∞
B	-	-	-	-	-	10D	6E	∞
G	-	-	-	-	-	8G	-	12G
F	-	-	-	-	-	-	-	11F
H	-	-	-	-	-	-	-	11F

Figure 14-Adjacency Matrix

- **Orange values represent what are the shortest path between (A) and this node E.g. the shortest path between (A) and (F) is: [F←G←E←D←C←A] with cost 8.**

5- Evaluation:

5.1- Completeness:

YES*

- **Dijkstra's algorithm fails to find the shortest paths when negative weights exist.**

5.2- Optimality:

YES*

- **Dijkstra's algorithm guarantees finding the shortest path from a single source vertex to all other vertices in the graph, provided the graph has non-negative edge weights.**

5.3- Time Complexity:

- **it has a time complexity of O(V^2) where V is the number of vertices.**

- **With a more optimized implementation using a priority queue, it can achieve O((V+E) logV) complexity, where E is the number of edges.**
- **This best-case scenario occurs when using an optimized data structure like a Fibonacci heap for implementing the priority queue.**

5.4- Space Complexity:

- **O(V) to O(E + V).**
- **The auxiliary space complexity of Dijkstra's algorithm primarily depends on the data structures used for implementation, particularly the priority queue for managing vertices with their associated distances.**

6- Applications in Real World:

- **Pathfinding in Video Games and Robotics:**

  Dijkstra's algorithm is utilized in video game development and robotics for pathfinding purposes. It enables game characters or robots to navigate through a complex environment, avoiding obstacles and finding the shortest path to reach their targets.

- **Network Routing Protocols:**

  Dijkstra's algorithm is used in network routing protocols, such as RIP, OSPF, and BGP, to calculate the best route between two nodes.

  RIP is based on the distance vector-based strategy, so we consider the entire structure as a graph where nodes are the routers, and the links are the networks.

  In a routing table, the first column is the destination, or we can say that it is a network address.

The cost metric is the number of hops to reach the destination. The number of hops available in a network would be the cost. The hop count is the number of networks required to reach the destination.

In RIP, infinity is defined as 16, which means that the RIP is useful for smaller networks or small autonomous systems. The maximum number of hops that RIP can contain is 15 hops, i.e., it should not have more than 15 hops as 16 is infinity.

The next column contains the address of the router to which the packet is to be sent to reach the destination.

- **Navigation Systems:**

  GPS devices and mapping applications use Dijkstra's algorithm to find the shortest path between two locations. Whether you're driving, walking, or using public transport, these systems calculate the most efficient route based on factors like distance, traffic conditions, and travel time.

7- References:

[1]-https://study.com/academy/lesson/weighted-graphs-implementation-dijkstra-algorithm.html#:~:text=A%20weighted%20graph%20refers%20to%20one%20where%20weights%20are%20assigned,directional%20and%20have%20no%20arrows.

[2]- https://www.youtube.com/watch?v=C9yOKXsGzSE

[3]- https://testbook.com/gate/dijkstra-algorithm-notes

[4]- https://www.analyticssteps.com/blogs/how-dijkstras-algorithm-used-real-world

[5]-https://www.linkedin.com/pulse/ospf-open-short-path-first-routing-protocol-using-dijkstra-nalla/

[6]-https://www.geeksforgeeks.org/time-and-space-complexity-of-dijkstras-algorithm/

Name	Contribution Percentage	Signature
Ahmed Mohamed Fawzy.	**25% of Report.** **25% of Presentation.**	
Passant El-Tonsy Ali.	**25% of Report.** **25% of Presentation.**	
Saif Emad El-Deen Abd-Elkareem.	**25% of Report.** **25% of Presentation.**	
Mohamed Emad Fawzy.	**25% of Report.** **25% of Presentation.**	