DIJKSTRA'S ALGORITHM

From point A to B! Foundations of Shortest Path Algorithms

ALGORITHM AUTHOR'S BACKGROUND

Bertrand Meyer(2009) noted, "The revolution in views of programming started by Dijkstra's iconoclasm led to a movement known as structured programming, which advocated a systematic, rational approach to program construction.

Structured programming is the basis for all that has been done since in programming methodology, including object-oriented programming."

EDSGER W. DIJKSTRA

- Born May 1930 – August 2002
- Netherlands
- Awarded Turing Award
- For contributions "of lasting and major technical importance to the computer field"
- Leiden University,
- (B.S., M.S.)
- University of Amsterdam,
- (Ph.D.)

ALGO HISTORY

Dijkstra thought about the shortest path problem when working at the Mathematical Center in Amsterdam in 1956 as a programmer to demonstrate the capabilities of a new computer called ARMAC.[6]

His objective was to choose both a problem and a solution (that would be produced by computer) that non-computing people could understand. He designed the shortest path algorithm and later implemented it for ARMAC for a slightly simplified transportation map of 64 cities in the Netherlands (64, so that 6 bits would be sufficient to encode the city number).[2]

A year later, he came across another problem from hardware engineers working on the institute's next computer: minimize the amount of wire needed to connect the pins on the back panel of the machine. As a solution, he re-discovered the algorithm known as Prim's minimal spanning tree algorithm (known earlier to Jarník, and also rediscovered by Prim).[7][8] Dijkstra published the algorithm in 1959, two years after Prim and 29 years after Jarník.

STEP 1

Mark all nodes unvisited.

Create a set of all the unvisited nodes called the *unvisited set*.

STEP 2

- Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes.

- Set the initial node as current.

STEP 3

- For the current node, consider all of its unvisited neighbors and calculate their *tentative* distances through the current node.
- Compare the newly calculated *tentative* distance to the current assigned value and assign the smaller one.
- For example, if the current node *A* is marked with a distance of 6, and the edge connecting it with a neighbor *B* has length 2, then the distance to *B* through *A* will be 6 + 2 = 8.
- If B was previously marked with a distance greater than 8 then change it to 8. Otherwise, keep the current value.

STEP 4

- When we are done considering all of the neighbors of the current node, mark the current node as visited and remove it from the *unvisited set*.
- A visited node will never be checked again.

STEP 5

- Move to the next unvisited node with the smallest tentative distances.
- Repeat the above steps which check neighbors and mark visited.

STEP 6

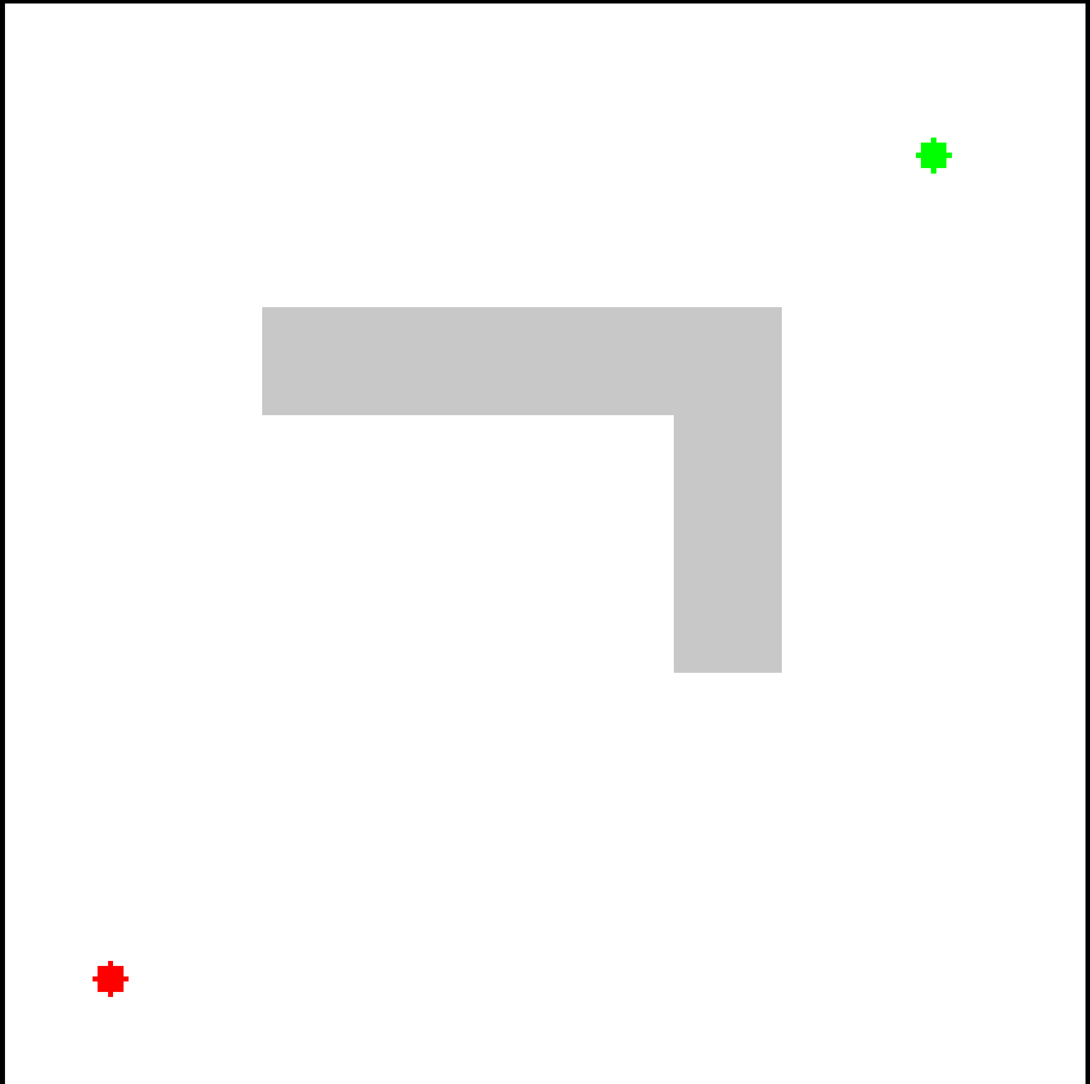
- If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the *unvisited set* is infinity (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), then stop.

- The algorithm has finished.

STEP 7

- Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new "current node", and go back to step 3.

DIJKSTRA'S ALGORITHM AT WORK



PSEUDOCODE OVERVIEW

In the following algorithm, the code u ← vertex in Q with min dist[u], searches for the vertex *u* in the vertex set *Q* that has the least dist[*u*] value.

length(*u*, *v*) returns the length of the edge joining (i.e. the distance between) the two neighbor-nodes *u* and *v*.

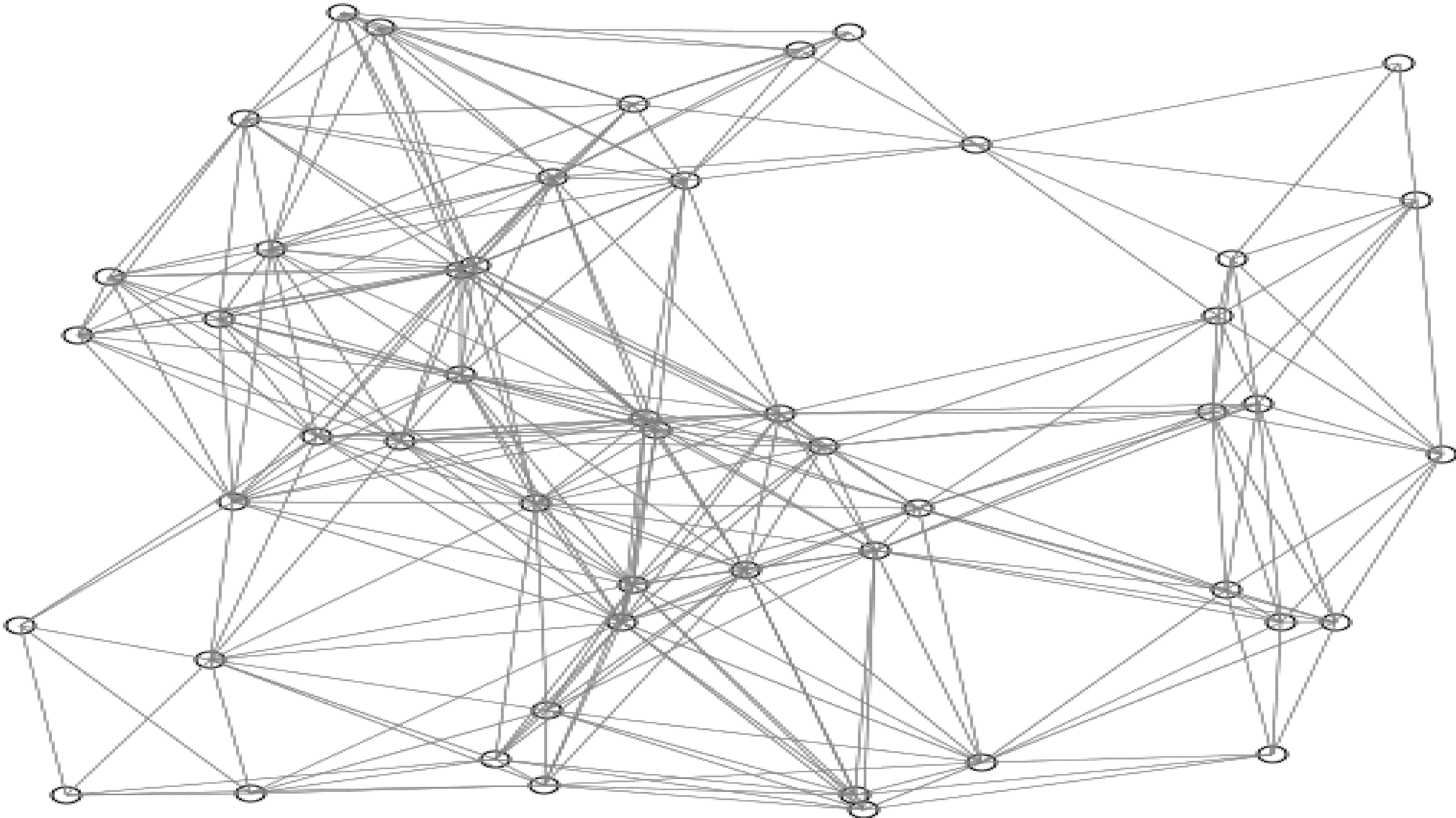
The variable *alt* on line 18 is the length of the path from the root node to the neighbor node *v* if it were to go through *u*.

If this path is shorter than the current shortest path recorded for *v*, that current path is replaced with this *alt* path.

The previous array is populated with a pointer to the "next-hop" node on the source graph to get the shortest route to the source.

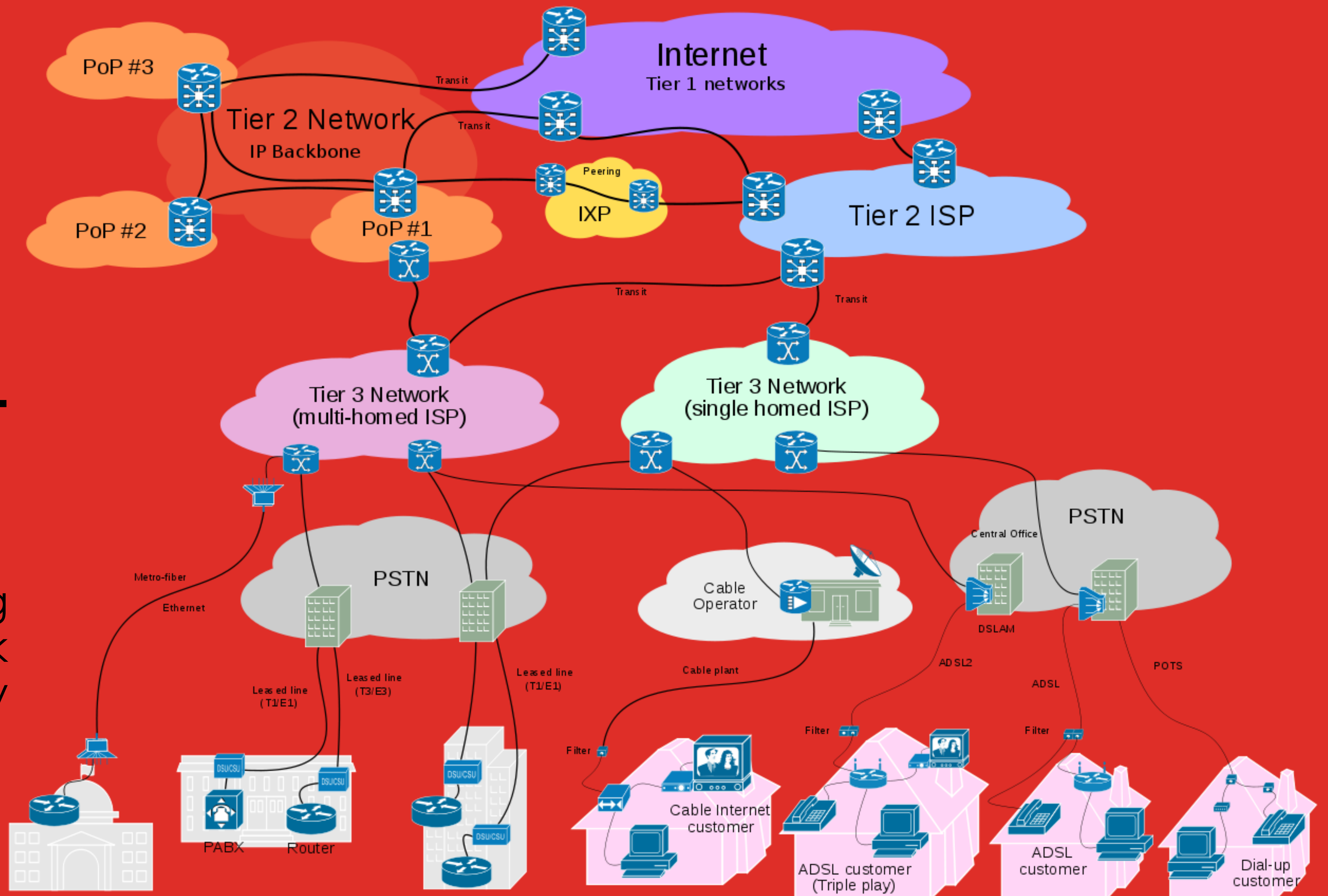
PSEUDOCODE

```
function Dijkstra(Graph, source):

    create vertex set Q

    for each vertex v in Graph:             // Initialization
        dist[v] ← INFINITY                  // Unknown distance from source to v
        prev[v] ← UNDEFINED                 // Previous node in optimal path from source
        add v to Q                          // All nodes initially in Q (unvisited nodes)

    dist[source] ← 0                        // Distance from source to source

    while Q is not empty:
        u ← vertex in Q with min dist[u]    // Node with the least distance
                                            // will be selected first
        remove u from Q

        for each neighbor v of u:           // where v is still in Q.
            alt ← dist[u] + length(u, v)
            if alt < dist[v]:               // A shorter path to v has been found
                dist[v] ← alt
                prev[v] ← u

    return dist[], prev[]
```



TECH IMPACT

Shortest distance routing network traffic in a network // decreased latency



ECONOMIC IMPACT

01

Increased efficiency, eased traffic flow

02

Time saving

03

Saves on fuel costs for transport of goods

POSSIBLE DRAWBACKS

Spams nodes, blindly searches for the best route (no prediction until end)

May waste runtime or resources depending on the use case

Cannot handle negative inputs for edges

Above can lead to acyclic graphs and the inability to identify the shortest path

ADDITIONAL RESOURCES/SOURCES – LINKS TO OTHER WORKS OF DIJKSTRA

https://www.geeksforgeeks.org/greedy-algorithms-set-6-dijkstras-shortest-path-algorithm/

https://www.youtube.com/watch?v=gdmfOwyQlcI

https://www.youtube.com/watch?v=GazC3A4OQTE

https://www3.cs.stonybrook.edu/~skiena/combinatorica/animations/dijkstra.html

FIN