Question-1(15 Marks): Apply Dijkstras Algorithm For The Following Graph And Find The Shortest Path From

Question-1(15 Marks): Apply Dijkstras Algorithm For The Following Graph And Find The Shortest Path From

---

Introduction

Dijkstra's Algorithm is a fundamental technique in graph theory and computer science used to determine the shortest paths from a single source vertex to all other vertices in a weighted graph with non-negative edge weights. Its efficiency and simplicity make it a preferred method for network routing, geographical mapping, and various optimization problems. In this article, we will explore how to apply Dijkstra's Algorithm step-by-step to a given graph and calculate the shortest path from a specified source node to all other nodes.

Understanding Dijkstra’s Algorithm

What is Dijkstra’s Algorithm?

Dijkstra's Algorithm, developed by Edsger Dijkstra in 1956, systematically explores the shortest paths in a graph. It works by iteratively selecting the closest unvisited vertex, updating the shortest paths to neighboring vertices, and marking vertices as visited once their shortest path is confirmed.

Key Concepts

- Weighted Graph: A graph where edges have associated weights (costs, distances, etc.).
- Source Vertex: The starting point from which shortest paths are calculated.
- Distance Table: Maintains the current shortest distance from the source to each vertex.
- Visited Set: Vertices for which the shortest path has been finalized.

Algorithm Steps

1. Initialization:
- Set the distance to the source vertex as 0.
- Set the distance to all other vertices as infinity.
- Mark all vertices as unvisited.

2. Selection:
- Pick the unvisited vertex with the smallest tentative distance.

3. Update:
- For each unvisited neighbor of the current vertex, calculate the tentative distance.
- If this distance is less than the current known distance, update it.

4. Mark as Visited:
- Once processed, mark the current vertex as visited.

5. Repeat:
- Continue until all vertices are visited or the shortest path to the target is determined.

Applying Dijkstra’s Algorithm to the Given Graph

Step 1: Graph Representation

Suppose the problem provides a graph with nodes labeled A, B, C, D, E, and F, with edges and weights as follows:

Edge	Weight
A - B	4
A - C	2
B - C	1
B - D	5
C - D	8
C - E	10
D - E	2
D - F	6
E - F	3

Note: For illustration, assume the source node is A.

Step 2: Initialization

Vertex	Initial Distance	Status	Comment
A	0	Unvisited	Source node, distance initialized to 0
B	∞	Unvisited	Unknown at the start
C	∞	Unvisited	Unknown at the start
D	∞	Unvisited	Unknown at the start
E	∞	Unvisited	Unknown at the start
F	∞	Unvisited	Unknown at the start

Step 3: Iteration 1 - Starting from Node A

- Current vertex: A (distance 0)
- Neighbors: B (4), C (2)

Update tentative distances:

- B: min(∞, 0 + 4) = 4
- C: min(∞, 0 + 2) = 2

Mark A as visited.

Vertex	Distance	Status
A	0	Visited
B	4	Unvisited
C	2	Unvisited
D	∞	Unvisited
E	∞	Unvisited
F	∞	Unvisited

Step 4: Iteration 2 - Select the Closest Unvisited Vertex

- The next closest is C (distance 2).

Process node C:

- Neighbors: B (already visited), D (8), E (10)
- Update distances:

- D: min(∞, 2 + 8) = 10
- E: min(∞, 2 + 10) = 12

Update table:

Vertex	Distance	Status
A	0	Visited
B	4	Unvisited
C	2	Visited
D	10	Unvisited
E	12	Unvisited
F	∞	Unvisited

Step 5: Iteration 3 - Next Vertex: B (distance 4)

Process node B:

- Neighbors: A (visited), C (visited), D (5)

Update D:

- D: min(10, 4 + 5) = 9

Updated table:

Vertex	Distance	Status

A	0	Visited
B	4	Visited
C	2	Visited
D	9	Unvisited
E	12	Unvisited
F	∞	Unvisited

Step 6: Iteration 4 - Next Vertex: D (distance 9)

Process node D:

- Neighbors: B (visited), C (visited), E (2), F (6)

Update:

- E: min(12, 9 + 2) = 11
- F: min(∞, 9 + 6) = 15

Updated table:

Vertex	Distance	Status
A	0	Visited
B	4	Visited
C	2	Visited
D	9	Visited
E	11	Unvisited
F	15	Unvisited

Step 7: Iteration 5 - Next Vertex: E (distance 11)

Process node E:

- Neighbors: C (visited), D (visited), F (3)

Update:

- F: min(15, 11 + 3) = 14

Update table:

Vertex	Distance	Status
A	0	Visited
B	4	Visited
C	2	Visited
D	9	Visited
E	11	Visited
F	14	Unvisited

Step 8: Final Iteration - Next Vertex: F (distance 14)

Process node F:

- Neighbors: D (visited), E (visited)

No further updates are needed.

Final Shortest Distances from Source A

Vertex	Shortest Distance from A	Path
A	0	A
B	4	A → B
C	2	A → C
D	9	A → C → D or A → B → D (via 4+5)
E	11	A → C → E
F	14	A → C → D → F or other paths

Visual Representation of the Shortest Paths

To better understand the results, here's a visual summary:

```
2
A ------ C
| | \
4| | \10
| | \
B D --- F
| | /
5 8| /
| |/
E -------|
10
```

From the diagram, the shortest paths are:

- A → C: 2
- A → B: 4
- A → C → D: 2 + 8 = 10 (but D has a shortest distance of 9, so the actual shortest path is A → B → D with total 4 + 5 = 9)
- A → C → E: 2 + 10 = 12 (but shortest is via A → C → E with total 11, so path is A → C → E)
- A → C → D → F: 2 + 8 + 6 = 16, but shortest is A → C → D → F with total 14 via A → C → D → F.

Conclusion

Applying Dijkstra's Algorithm involves initializing the graph, selecting the closest unvisited vertex, updating neighboring vertices' tentative distances, and marking vertices as visited. Through systematic iteration

Frequently Asked Questions

What is Dijkstra's Algorithm and how does it work for finding the shortest path in a graph?

Dijkstra's Algorithm is a greedy algorithm used to find the shortest path from a source vertex to all other vertices in a weighted graph with non-negative edge weights. It works by repeatedly selecting the vertex with the smallest tentative distance, updating the distances to its neighbors, and marking vertices as visited until all vertices have been processed.

How do you initialize the distances and previous nodes before applying Dijkstra's Algorithm?

Initially, set the distance to the source node as zero and all other nodes as infinity. Also, set the previous node for each vertex as null. This setup prepares the algorithm to iteratively improve the shortest path estimates.

What data structures are most suitable for implementing Dijkstra's Algorithm efficiently?

A min-priority queue or a min-heap is most suitable, as it allows efficient extraction of the vertex with the smallest tentative distance. Additionally, adjacency lists are commonly used to represent the graph for efficient traversal.

How do you handle the case when there are multiple shortest paths in a graph using Dijkstra's Algorithm?

Dijkstra's Algorithm finds one shortest path by updating predecessor nodes during relaxation. To find multiple shortest paths, you can modify the algorithm to store all predecessor nodes when multiple paths have the same shortest distance, then backtrack to list all possible shortest routes.

Can Dijkstra's Algorithm be applied to graphs with negative edge weights? Why or why not?

No, Dijkstra's Algorithm cannot be applied to graphs with negative edge weights because it assumes that once a node's shortest distance is finalized, it cannot be improved. Negative weights can violate this assumption, potentially leading to incorrect results. For

graphs with negative weights, algorithms like Bellman-Ford are used.

What are the steps involved in applying Dijkstra's Algorithm to the given graph in the question?

The steps include: 1) Initialize distances and previous nodes; 2) Select the unvisited node with the smallest tentative distance; 3) Update the distance values of its neighbors; 4) Mark the node as visited; 5) Repeat until all nodes are visited or the destination is reached; 6) Trace back from the destination to find the shortest path.

How do you interpret the results after applying Dijkstra's Algorithm to find the shortest path?

The results include the shortest distance from the source to each vertex, stored in the distance array, and the shortest path itself can be reconstructed by backtracking through the predecessor nodes. This helps identify the optimal route and total cost.

Additional Resources

Applying Dijkstra’s Algorithm to Determine the Shortest Path in a Weighted Graph

---

Introduction

In the realm of graph theory and network analysis, finding the most efficient path between nodes is a task of profound significance. Whether in transportation networks, communication systems, or logistics planning, the ability to determine the shortest path optimally influences decision-making and resource allocation. Among the algorithms designed for this purpose, Dijkstra’s Algorithm stands out as one of the most widely used and effective techniques for solving single-source shortest path problems in graphs with non-negative edge weights.

This article provides a comprehensive, investigative exploration of applying Dijkstra’s Algorithm to a specific weighted graph. The goal is to elucidate the step-by-step process of the algorithm, analyze its computational characteristics, and demonstrate how it derives the shortest path from a designated source node to all other nodes in the graph.

Fundamental Concepts of Dijkstra’s Algorithm

Background and Principles

Developed by Edsger W. Dijkstra in 1959, Dijkstra’s Algorithm is a greedy method for solving the single-source shortest path problem in graphs with non-negative weights. It systematically explores neighboring nodes, updating the shortest distances as it progresses, until the shortest paths to all nodes are finalized.

Core Assumptions and Constraints

- Non-negative Edge Weights: The algorithm assumes all edge weights are zero or positive. Negative weights can lead to incorrect results.
- Single Source: The starting node (source) is specified from which shortest paths are calculated.
- Graph Type: The graph can be directed or undirected, provided the weights are non-negative.

The Graph Under Consideration

To thoroughly illustrate the application of Dijkstra’s Algorithm, we consider a hypothetical weighted graph with nodes labeled A through F and weighted edges as follows:

Edge	Weight
A - B	4
A - C	2
B - C	1
B - D	5
C - D	8
C - E	10
D - E	2
D - F	6
E - F	3

Graph Visualization:

```
A
/ \
4 2
/ \
B-------C
| \ | \
5 1 8 10
| \ | \
D-----E--F
2 3 6
```

Note: The above diagram simplifies the structure, with nodes and weighted edges, to provide clarity on the algorithm's operations.

Step-by-Step Application of Dijkstra’s Algorithm

Initialization

- Source Node: A
- Distances: Set initial distances to infinity for all nodes except the source, which is zero.

Node	Distance from A	Previous Node
A	0	-
B	∞	-
C	∞	-
D	∞	-
E	∞	-
F	∞	-

- Unvisited Nodes: All nodes are initially unvisited.

Iterative Process

Step 1: Select the unvisited node with the smallest tentative distance.

- Current node: A (distance 0).

Update neighbors:

- B: 0 + 4 = 4 → Update distance to B: 4
- C: 0 + 2 = 2 → Update distance to C: 2

Updated table:

Node	Distance	Previous
A	0	-
B	4	A
C	2	A
D	∞	-
E	∞	-
F	∞	-

Mark A as visited.

Step 2: Next node with smallest tentative distance: C (2).

Update neighbors:

- B: current distance 4, via C: 2 + 1 = 3 → update to 3 (better than 4)
- D: current ∞, via C: 2 + 8 = 10 → update to 10
- E: current ∞, via C: 2 + 10 = 12 → update to 12

Updated table:

| Node | Distance | Previous |

|-------|------------|----------|
A	0	-
B	3	C
C	2	A
D	10	C
E	12	C
F	∞	-

Mark C as visited.

Step 3: Next node: B (3).

Update neighbors:

- D: current 10, via B: 3 + 5 = 8 → update to 8
- F: current ∞, via B: 3 + 6 = 9 → update to 9

Updated table:

Node	Distance	Previous
A	0	-
B	3	C
C	2	A
D	8	B
E	12	C
F	9	B

Mark B as visited.

Step 4: Next node: D (8).

Update neighbors:

- E: current 12, via D: 8 + 2 = 10 → update to 10
- F: current 9, via D: 8 + 6 = 14 → no update since 9 < 14

Updated table:

Node	Distance	Previous
A	0	-
B	3	C
C	2	A
D	8	B
E	10	D
F	9	B

Mark D as visited.

Step 5: Next node: F (9).

Update neighbors:

- E: current 10, via F: 9 + 3 = 12 → no update since 10 < 12

Mark F as visited.

Step 6: Last node: E (10).

No further updates needed.

Final shortest path distances from A:

Node	Shortest Distance	Path
A	0	-
B	3	A → C → B
C	2	A → C
D	8	A → C → B → D
E	10	A → C → B → D → E
F	9	A → C → B → F

Extracting the Shortest Paths

Using the 'Previous' node information, the shortest paths can be reconstructed:

- To B: A → C → B, with total weight 3.
- To C: A → C, weight 2.
- To D: A → C → B → D, weight 8.
- To E: A → C → B → D → E, weight 10.
- To F: A → C → B → F, weight 9.

Computational Analysis and Optimization

Time Complexity

The efficiency of Dijkstra’s Algorithm depends on the data structures used:

- Using a simple array or list: $O(V^2)$, where V is the number of vertices.
- Using a min-priority queue (heap): $O((V + E) \log V)$, where E is the number of edges.

In practical applications, especially with sparse graphs, a min-heap implementation yields significant performance benefits.

Limitations and Considerations

- Negative Edge Weights: Dijkstra’s Algorithm cannot handle negative weights. For graphs with negative edges, Bellman-Ford algorithm is preferred.
- Graph Size: Large graphs may require optimized data structures or parallel processing.

Real-World Applications and Implications

The clarity and efficiency of Dijkstra’s Algorithm make it invaluable across diverse fields:

- Navigation Systems: Calculating shortest routes in GPS applications.
- Network Routing: Determining optimal data paths in communication networks.
- Supply Chain Management: Optimizing logistics and delivery routes.
- Robotics: Path planning in obstacle-laden environments.

Its robustness, coupled with computational efficiency, ensures its continued relevance in both theoretical research and practical implementations.

Conclusion

Applying Dijkstra’s Algorithm to a weighted graph provides a systematic approach to identifying the shortest paths from a source node to all other nodes. Through an iterative process of selecting the nearest unvisited node, updating neighboring distances, and recording optimal routes, the algorithm guarantees the shortest paths in graphs with non-negative weights.

This detailed exploration underscores not only the step-by-step procedure but also emphasizes the importance of understanding underlying data structures,

Question 1 15 Marks Apply Dijkstras Algorithm For The Following Graph And Find The Shortest Path From

Question 1 15 Marks Apply Dijkstras Algorithm For The Following Graph And Find The Shortest Path From

Related Articles

- Question 1 Of 10What Information Can The Reader Learn From Characterization?A. The Central Conflict The
- Q C A System Consists Of Three Particles, Each Of Mass 5.00g , Located At The Corners Of An

Back to Home