**Dijkstra's Algorithm: Finding the Shortest Path**

**Dijkstra's Algorithm** is a fundamental algorithm used to find the shortest path from a starting vertex to every other vertex in a graph. It is a **greedy algorithm**, meaning that it builds the shortest path one step at a time by always choosing the closest vertex that hasn't yet been visited.

**Steps of Dijkstra's Algorithm**

Let's walk through the general steps of Dijkstra's Algorithm:

1. **Initialization**:
   * Start from a given vertex (say vertex A).
   * Assign a tentative distance value to every vertex. Set the starting vertex's distance to 0 and all other vertices to infinity.
   * Mark all vertices as unvisited. The starting vertex is considered the current vertex.
2. **Expand Current Vertex**:
   * For each unvisited neighbor of the current vertex, calculate the tentative distance through the current vertex.
   * If this distance is less than the currently known distance to that vertex, update it.
3. **Select the Next Vertex**:
   * After considering all neighbors, mark the current vertex as visited (i.e., "known").
   * Choose the unvisited vertex with the smallest tentative distance and set it as the new current vertex.
4. **Repeat**:
   * Continue the process for the unvisited vertex with the smallest distance until all vertices have been visited.

**Example of Dijkstra’s Algorithm**

Let’s use the following undirected, weighted graph as an example:

* **Vertices**: A, B, C, D, E
* **Edges**:
  + A → B (7), A → E (3)
  + B → C (6), B → D (2)
  + C → D (4)
  + D → E (2), E → B (1)

The goal is to find the shortest distance from **A** to every other vertex.

**Step-by-Step Walkthrough**

**Initial State**

Start by setting up a table to track the shortest distances and previous vertices for each vertex. We initialize all vertices to infinity (∞) except the starting vertex A, which is set to 0.

| **Vertex** | **Shortest Distance from A** | **Previous Vertex** | **Known** |
| --- | --- | --- | --- |
| **A** | 0 | - | False |
| **B** | ∞ | - | False |
| **C** | ∞ | - | False |
| **D** | ∞ | - | False |
| **E** | ∞ | - | False |

**Expand A**

From vertex A, we can reach B and E:

* Distance to B: 7
* Distance to E: 3

We update the table with the new distances:

| **Vertex** | **Shortest Distance from A** | **Previous Vertex** | **Known** |
| --- | --- | --- | --- |
| **A** | 0 | - | True |
| **B** | 7 | A | False |
| **C** | ∞ | - | False |
| **D** | ∞ | - | False |
| **E** | 3 | A | False |

**Expand E**

Next, the vertex with the smallest distance is **E** (distance 3). From E, we can reach B and D:

* Distance to B: 3 + 1 = 4 (update, since 4 < 7)
* Distance to D: 3 + 2 = 5 (update, since 5 < ∞)

| **Vertex** | **Shortest Distance from A** | **Previous Vertex** | **Known** |
| --- | --- | --- | --- |
| **A** | 0 | - | True |
| **B** | 4 | E | False |
| **C** | ∞ | - | False |
| **D** | 5 | E | False |
| **E** | 3 | A | True |

**Expand B**

The next smallest distance is **B** (distance 4). From B, we can reach C and D:

* Distance to C: 4 + 6 = 10 (update, since 10 < ∞)
* Distance to D: 4 + 2 = 6 (no update, since 6 > 5)

| **Vertex** | **Shortest Distance from A** | **Previous Vertex** | **Known** |
| --- | --- | --- | --- |
| **A** | 0 | - | True |
| **B** | 4 | E | True |
| **C** | 10 | B | False |
| **D** | 5 | E | False |
| **E** | 3 | A | True |

**Expand D**

The next smallest distance is **D** (distance 5). From D, we can reach C:

* Distance to C: 5 + 4 = 9 (update, since 9 < 10)

| **Vertex** | **Shortest Distance from A** | **Previous Vertex** | **Known** |
| --- | --- | --- | --- |
| **A** | 0 | - | True |
| **B** | 4 | E | True |
| **C** | 9 | D | False |
| **D** | 5 | E | True |
| **E** | 3 | A | True |

**Expand C**

Finally, we expand **C**. Since C has no unvisited neighbors, we mark it as visited:

| **Vertex** | **Shortest Distance from A** | **Previous Vertex** | **Known** |
| --- | --- | --- | --- |
| **A** | 0 | - | True |
| **B** | 4 | E | True |
| **C** | 9 | D | True |
| **D** | 5 | E | True |
| **E** | 3 | A | True |

**Final Table**

This final table shows the shortest distances from A to all other vertices, and the "Previous Vertex" column can be used to reconstruct the shortest path from A to any other vertex.

**Finding the Shortest Path**

If we want to find the shortest path from A to C:

* The shortest distance is 9.
* To find the path, we work backwards from C:
  + C’s previous vertex is D.
  + D’s previous vertex is E.
  + E’s previous vertex is A.

Thus, the shortest path from A to C is:

* **A → E → D → C**

**Quiz: Dijkstra's Algorithm**

1. **What type of algorithm is Dijkstra’s Algorithm?**
   * a) Dynamic programming
   * b) Backtracking
   * c) Greedy algorithm
   * d) Divide and conquer

**Answer**: c) Greedy algorithm

1. **What is the time complexity of Dijkstra’s Algorithm when implemented with a priority queue?**
   * a) O(V2)O(V^2)O(V2)
   * b) O(E+V)O(E + V)O(E+V)
   * c) O(Elog⁡V)O(E \log V)O(ElogV)
   * d) O(Vlog⁡V)O(V \log V)O(VlogV)

**Answer**: c) O(Elog⁡V)O(E \log V)O(ElogV)

1. **Which of the following graphs would Dijkstra’s Algorithm fail to find the correct shortest path for?**
   * a) Undirected graph
   * b) Weighted graph
   * c) Directed graph
   * d) Graph with negative weight edges

**Answer**: d) Graph with negative weight edges

1. **What does Dijkstra's algorithm use to keep track of the next vertex to expand?**
   * a) Stack
   * b) Queue
   * c) Priority queue (min-heap)
   * d) Linked list

**Answer**: c) Priority queue (min-heap)

1. **Which vertex is chosen next in Dijkstra’s Algorithm?**
   * a) The vertex with the fewest edges
   * b) The vertex closest to the start vertex that hasn’t been processed yet
   * c) The vertex furthest from the start vertex
   * d) A random unvisited vertex

**Answer**: b) The vertex closest to the start vertex that hasn’t been processed yet

This overview covers the working of **Dijkstra’s Algorithm** along with an example, the steps to reconstruct the shortest path, and a quiz to test your understanding.