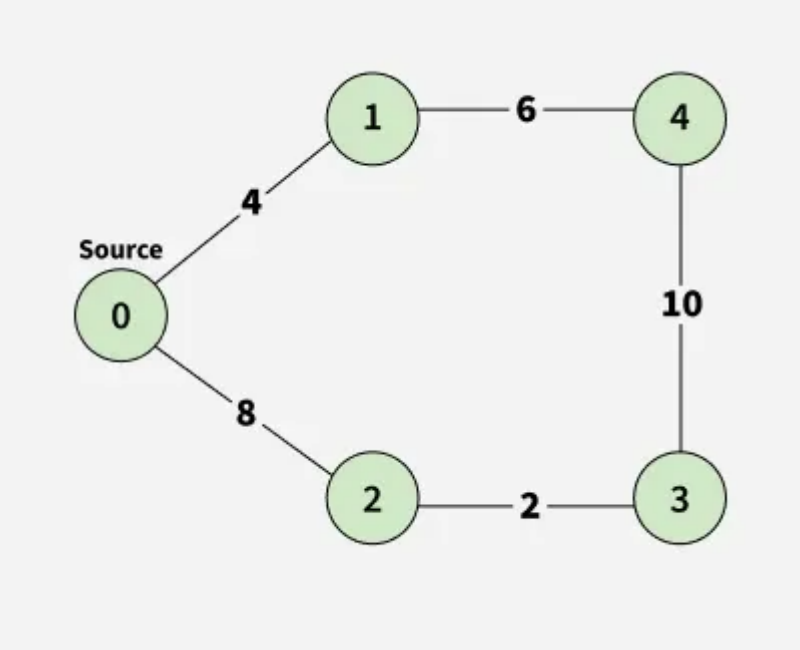
Dijkstra's Algorithm to find Shortest Paths from a Source to all

November 25, 2012

Given a weighted undirected graph represented as an **edge** list and a source vertex `src`, find the shortest path distances from the source vertex to all other vertices in the graph. The graph contains`v`vertices, numbered from `0` to `V - 1`.

Note: The given graph does not contain any negative edge.

Examples:

> **Input:** src = 0, V = 5, edges[][] = [[0, 1, 4], [0, 2, 8], [1, 4, 6], [2, 3, 2], [3, 4, 10]]
>
> 
>
>
> Graph with 5 node
>
> **Output:** 0 4 8 10 10
> **Explanation:** Shortest Paths:
> 0 to 1 = 4. 0 → 1
> 0 to 2 = 8. 0 → 2
> 0 to 3 = 10. 0 → 2 → 3
> 0 to 4 = 10. 0 → 1 → 4

Dijkstra’s Algorithm using Min Heap - O(E*logV) Time and O(V) Space

> In Dijkstra's Algorithm, the goal is to find the shortest distance from a given source node to all other nodes in the graph. As the source node is the starting point, its distance is initialized to zero. From there, we iteratively pick the unprocessed node with the minimum distance from the source, this is where a min-heap (priority queue) or a set is typically used for efficiency. For each picked node u, we update the distance to its neighbors v using the formula: **dist[v] = dist[u] + weight[u][v],** but only if this new path offers a shorter distance than the current known one. This process continues until all nodes have been processed.

Step-by-Step Implementation

1. Set `dist[source]=0` and all other distances as `infinity`.
2. Push the source node into the min heap as a pair `<distance, node>` → i.e., `<0, source>`.
3. Pop the top element (node with the smallest distance) from the min heap.
    1. For each adjacent neighbor of the current node:
    2. Calculate the distance using the formula:
       `dist[v] = dist[u] + weight[u][v]`
       If this new distance is shorter than the current `dist[v]`, update it.
       Push the updated pair `<dist[v], v>` into the min heap
4. **Repeat** step 3 until the min heap is empty.
5. Return the distance array, which holds the shortest distance from the source to all nodes.

Time Complexity: O(E*logV), Where E is the number of edges and V is the number of vertices.
Auxiliary Space: O(V), Where V is the number of vertices, We do not count the adjacency list in auxiliary space as it is necessary for representing the input graph.