Dijkstra’s: Shortest path **from one node to all other nodes** in a *weighted* graph, directed or undirected, **non-negative** weights.

Shortest path in terms of number of edges may not be the shortest in terms of total cost.

If we only want the cheapest path between one pair of nodes, just pick one of them as the starting point and run Dijkstra's algorithm. Finding the cheapest path to *all* nodes *includes* finding the cheapest path to the other node in the pair.

But isn't Dijkstra's algorithm overkill if we only care about one pair of nodes? Actually no, because we'll still need to consider other nodes in the graph to make sure we've found the lowest-cost weighted path.

For finding the shortest path on a graph, using BFS and DFS can be very impractical. **They assume that the weight between all nodes is the same**, whereas the actual weight/cost could vary. This is where Dijkstra comes into play!

Dijkstra’s algorithm finds a **shortest path tree** from a single source node.

Algo:

- Initialize three values: *dist* array (from *s* to all nodes), queue *Q* of unvisited nodes (all nodes in the beginning), set *S* of visited nodes (empty to start with)
- While *Q* is not empty, pop the node *v*, that is not already in *S*, from *Q* with the smallest *dist* (*v*). In the first run, source node *s* will be chosen because *dist*(*s*) was initialized to 0. In the next run, the next node with the smallest *dist* value is chosen.
- Add node *v* to *S*, to indicate that *v* has been visited
- Update *dist* values of adjacent nodes of the current node *v* as follows: for each new adjacent node *u*,
    - if *dist* (*v*) + *weight*(*u*,*v*) < *dist* (*u*), there is a new minimal distance found for *u*, so update *dist* (*u*) to the new minimal distance value;
    - otherwise, no updates are made to *dist* (*u*).

Pseudocode:

```
function Dijkstra(Graph, source):
    // Initializations
    dist[source]  := 0     // Distance from source to source is set to 0
    for each vertex v in Graph:
        if v ≠ source
            dist[v]  := infinity
        add v to Q                              // All nodes initially in Q

    while Q is not empty:                       // The main loop
        v := vertex in Q with min dist[v]       // v = source in 1st iteration
        remove v from Q

        for each neighbor u of v:               // where u is still present in Q
            alt := dist[v] + length(v, u)
            if alt < dist[u]:                   // A shorter path to u found
                dist[u]  := alt                 // Update distance of u

    return dist[]
end function
```

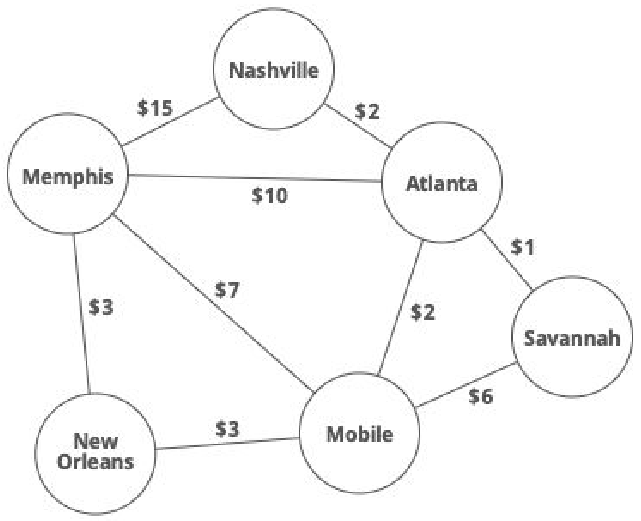
- Time Complexity: O(V^2) but with min-priority queue it drops down to O(V+ElogV). Expensive for shortest paths between **all pairs** of nodes. Floyd-Warshall is a better algorithm for that scenario.
- Other Shortest Path algorithms: BFS, Bellman Ford, Floyd-Warshall, Prim's MST
- If you only need the path between two specific nodes, you can stop the algorithm as soon as you mark your second node as visited.
- Sometimes, there are several minimum paths between two nodes (different paths with the same weights). Ties can be broken arbitrarily.
- If you finish the algorithm because there are no unvisited nodes left but there are nodes which minimum distance is still infinity, those nodes don't have any valid path to the original node.

Applications:

- Google Maps: Suppose we want to go from A to B in the shortest possible way. We know some roads are heavily congested and difficult to use. In Dijkstra's algorithm, this means the edge has a large weight--the shortest path tree found by the algorithm will try to avoid edges with larger weights.

- IP Routing

- Airlines - Lots of airlines like to route through large hubs and might not have much service directly between two smaller cities.



Helpful links:

Explanation with airlines example, Python code using heapq module, and discussion of time & space complexity: https://www.interviewcake.com/concept/java/dijkstras-algorithm

MCQs on Dijkstra

Some interesting shortest path questions

Implementations:

https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/

https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-using-set-in-stl/

https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-using-priority_queue-stl/

https://www.geeksforgeeks.org/widest-path-problem-practical-application-of-dijkstras-algorith
m

https://www.geeksforgeeks.org/printing-paths-dijkstras-shortest-path-algorithm/