Dijakstra’s Algorithm

https://www.one-tab.com/page/vdsg9a85Rj-k1QRyG7Hiqw

With Dijkstra's Algorithm, you can find the shortest path between nodes in a graph. Particularly, you can **find the shortest path from a node (called the "source node") to all other nodes in the graph**, producing a shortest-path tree.

This algorithm is used in GPS devices to find the shortest path between the current location and the destination. It has broad applications in industry, specially in domains that require modeling networks.

- Dijkstra's Algorithm basically starts at the node that you choose (the source node) and it analyzes the graph to find the shortest path between that node and all the other nodes in the graph.
- The algorithm keeps track of the currently known shortest distance from each node to the source node and it updates these values if it finds a shorter path.
- Once the algorithm has found the shortest path between the source node and another node, that node is marked as "visited" and added to the path.
- The process continues until all the nodes in the graph have been added to the path. This way, we have a path that connects the source node to all other nodes following the shortest path possible to reach each node.

if($d(u) + c(u, v) < d(v)$),

$d(v) = d(u) + c(u, v)$

Time complexity = E logV

```
function Dijkstra(Graph, source):
for each vertex v in Graph: // Initialization
dist[v] := infinity // initial distance from source to vertex v is set to infinite
previous[v] := undefined  // Previous node in optimal path from source
dist[source] := 0 // Distance from source to source
Q := the set of all nodes in Graph  // all nodes in the graph are unoptimized - thus are in Q
while Q is not empty: // main loop
u := node in Q with smallest dist[ ]
remove u from Q
for each neighbor v of u: // where v has not yet been removed from Q.
alt := dist[u] + dist_between(u, v)
if alt < dist[v]  // Relax (u,v)
dist[v] := alt
previous[v] := u
return previous[ ]
```