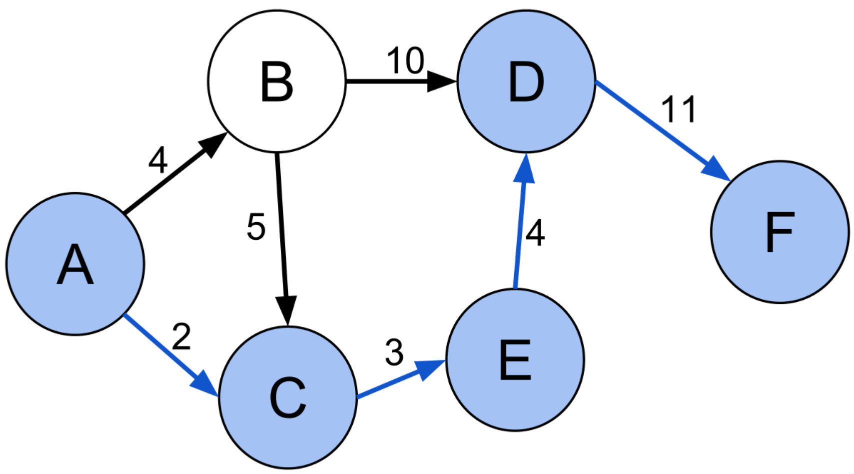
Lecture 19: Finding Shortest Path

Date: 10/24/2023

Shortest Path

A shortest path from **vertex s** to **vertex t** in an edge-weighted graph is a directed path from s to t with the property that no other such path has a lower weight.
⇒ One of the most intuitive use cases of graphs is to find the shortest path from one node to another.



reference: https://en.wikipedia.org/wiki/Shortest_path_problem

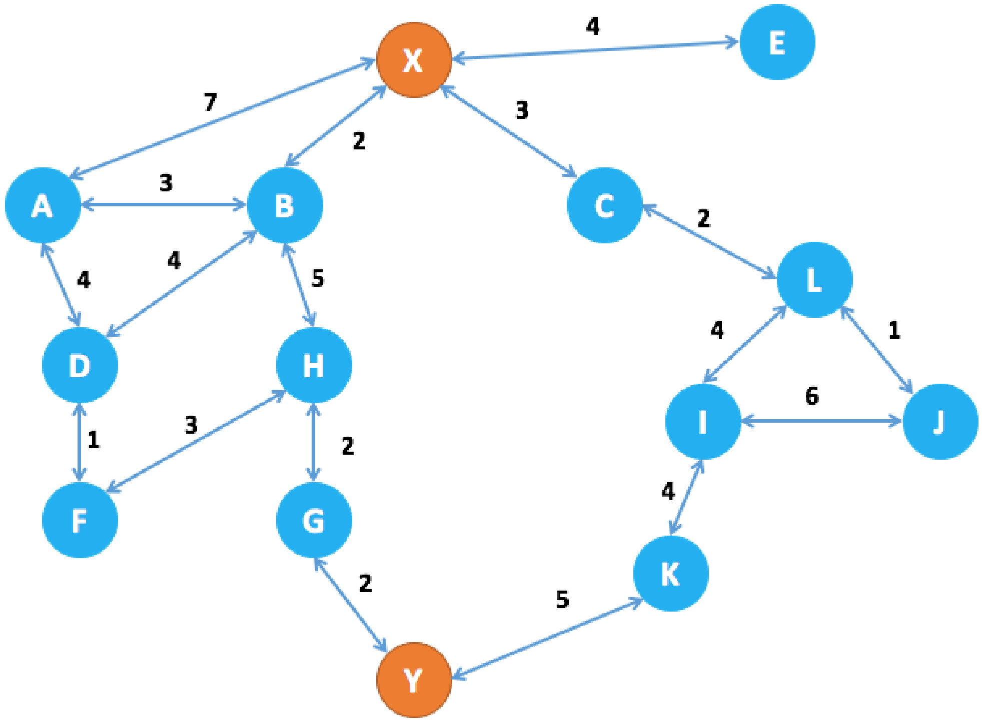
Applications

application	vertex	edge
map	intersection	road
network	router	connection
schedule	job	precedence constraint
arbitrage	currency	exchange rate

Typical shortest-paths applications

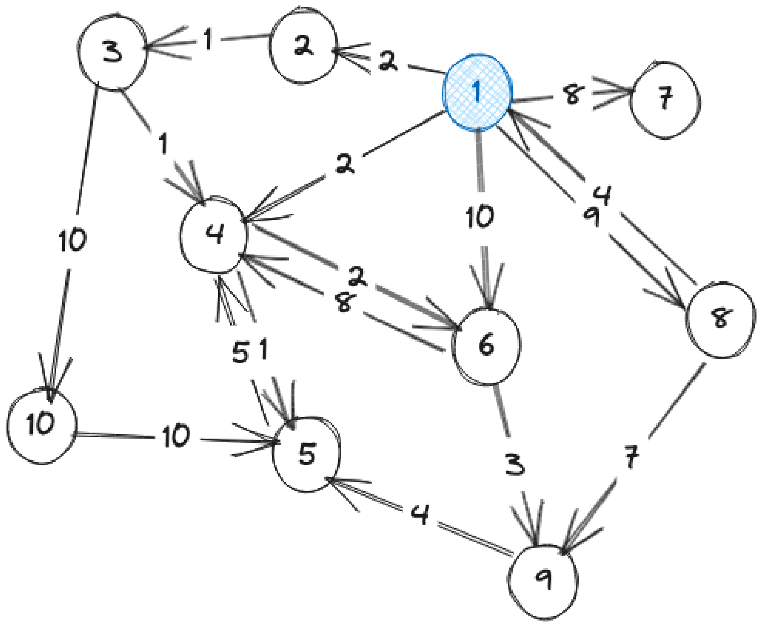
Types

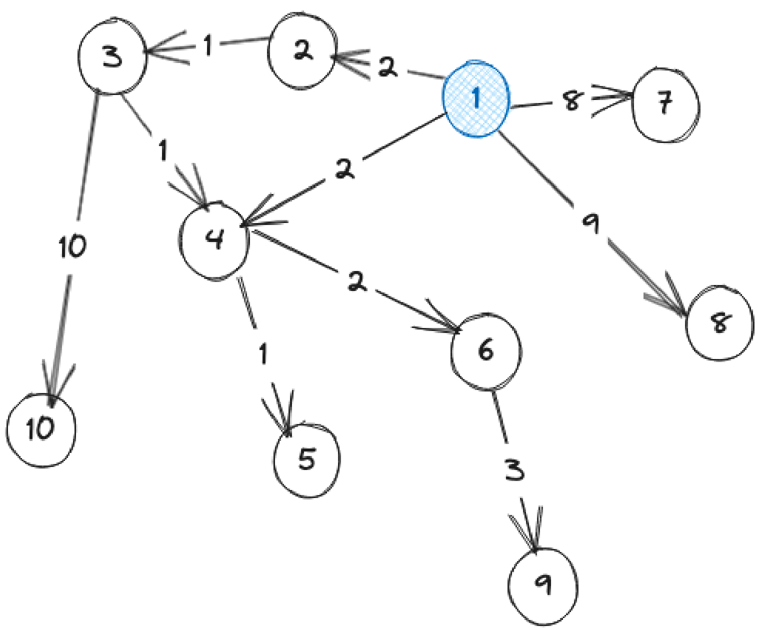
⇒ Single Source Single Sink: find path between source and sink



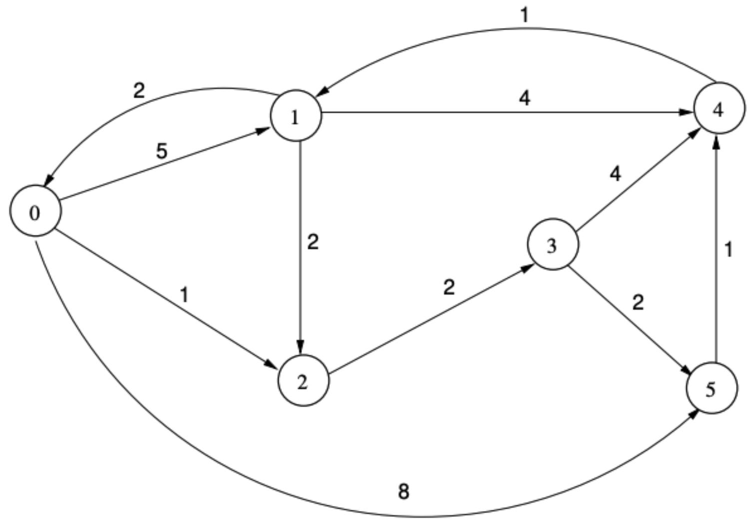
reference:
https://benalexkeen.com/implementing-djikstras-shortest-path-algorithm-with-python/

⇒ Shortest-paths Tree: given source, find shortest path to all other nodes





⇒ All Pair: find shortest path from all nodes to all other nodes

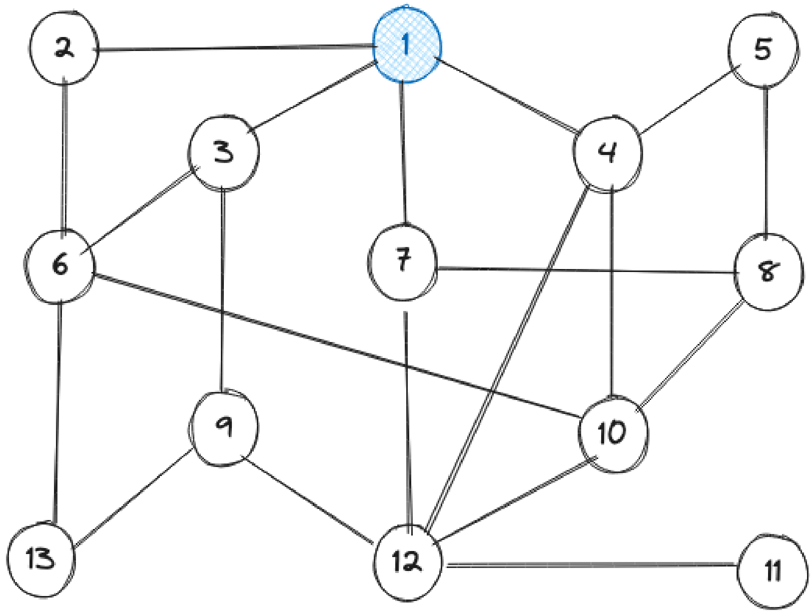


	0	1	2	3	4	5
0	0	5	1	3	6	5
1	2	0	2	4	4	6
2	8	6	0	2	5	4
3	6	4	6	0	3	2
4	3	1	3	5	0	7
5	4	2	4	6	1	0

Table 2:The shortest-paths matrix for the graph in Figure 1

reference: https://codeahoy.com/learn/graphalgorithms/ch8/

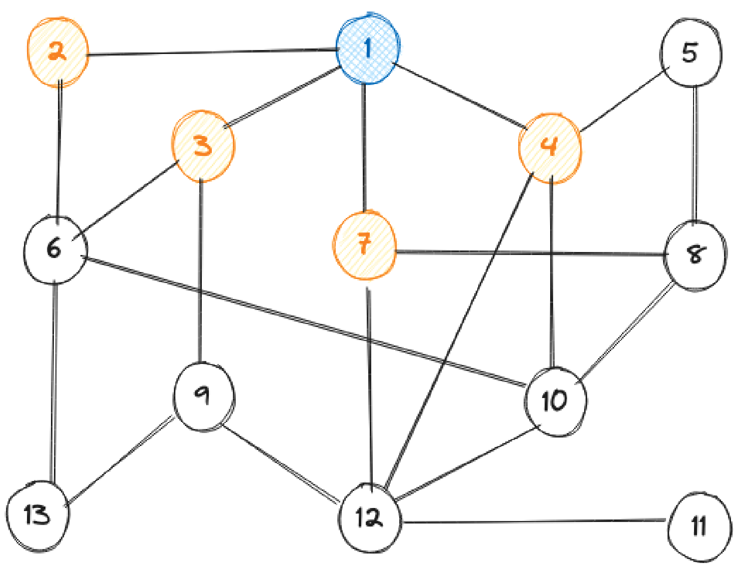
Unweighted Graph Shortest Path



If the graph is unweighted then we can assume each edge weight as one and find the shortest path using that new graph.

Breadth First Traversal

⇒ This traversal starts from the source node and in each iteration finds its first connected node.



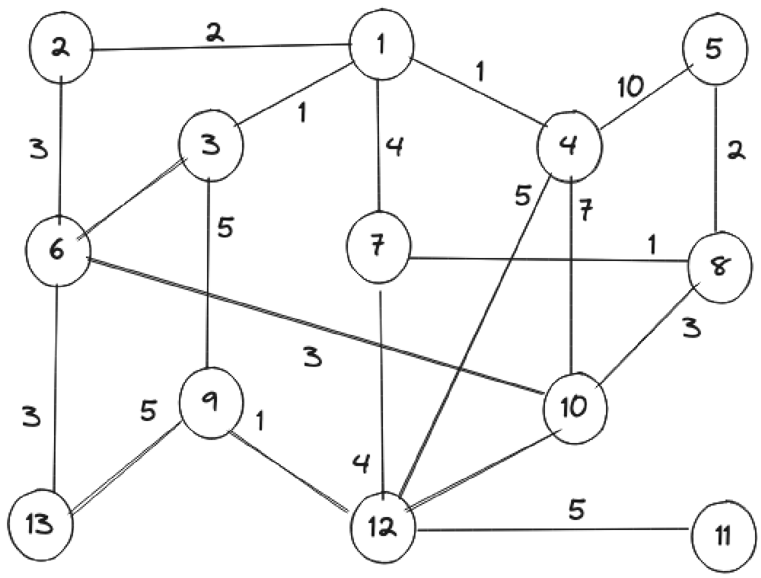
⇒ After the first iteration, we found the shortest path from source node 1 to nodes 2,3,7,4.
⇒ The same way, in each iteration we find shortest paths to their immediately connected nodes.
⇒ At the end of the algorithm, we will have shortest distances for all the nodes from the source node.

Pseudocode

```
function find_shortest_path(graph, source){
    shortest_paths = {}
    level = {source}
    while level is not empty:
        // iterate through all nodes in level
        for node in level:
            // iterate through all connected nodes
            for connected in node->connected:
                // update the shortest distance
                shortest_path[connected] =  shortest_path[node] + 1
                add connected node to new_level

        update level to new level
}
```

Weighted Graph Shortest Path

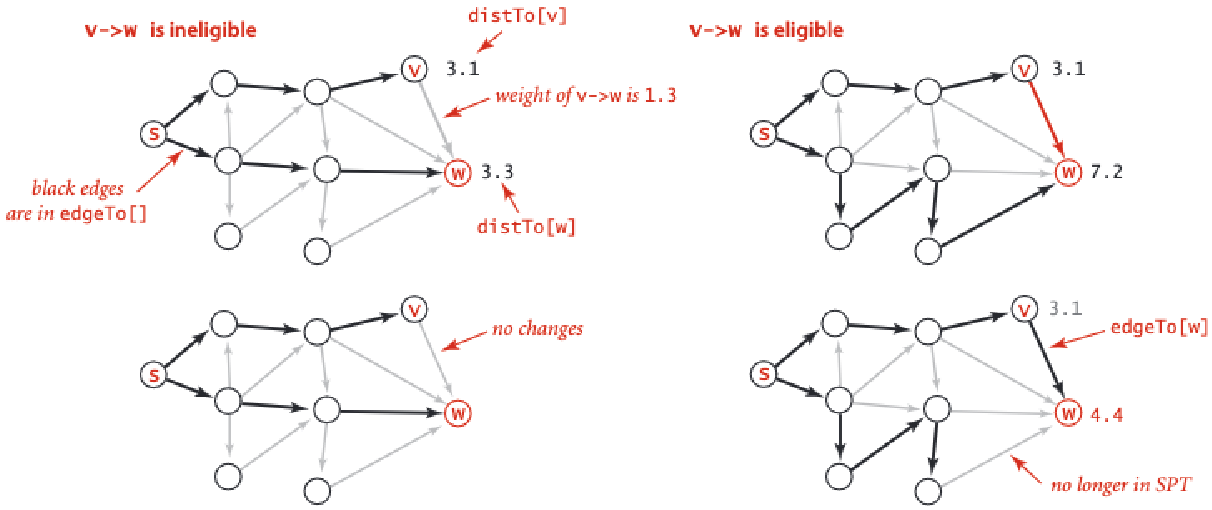


⇒ Using Breadth First Traversal won't work here as it does not take edge weight into consideration.
i.e. From node 1 to node 5, BFS will give 1->4->5 as the shortest path. Which is **not** the shortest. Correct shortest path will be, 1->7->8->5

Relaxation

Edge Relaxation

⇒ To relax an edge e, from v to w, means to figure out if the best way to reach from source to w is from using edge v -> w.



Edge relaxation (two cases)

```
private void relax(DirectedEdge e)
{
   int v = e.from(), w = e.to();
   if (distTo[w] > distTo[v] + e.weight())
   {
       distTo[w] = distTo[v] + e.weight();
       edgeTo[w] = e;
   }
}
```

Vertex Relaxation

⇒ Relax every edge going out from a specific vertex.

```
private void relax(EdgeWeightedDigraph G, int v)
{
   for (DirectedEdge e : G.adj(v))
   {
      int w = e.to();
      if (distTo[w] > distTo[v] + e.weight())
      {
         distTo[w] = distTo[v] + e.weight();
         edgeTo[w] = e;
      }
   }
}
```

Dijkstra's

⇒ Until we don't have every node in the shortest path tree, find the minimum distance node from source and relax it.
⇒ Similar to Prim's algorithm, greedy.

Pseudocode

```
function find_shortest_path(graph, source){
    priority_queue = {}
    initialize all distances to Math.inf except source

    priority_queue.insert(source)
    while priority_queue not empty
        relax_vertex(priority_queue.getMin())
}
```

Problems