AEL-4

Graph Traversal - 2

Floyd-Warshall Algorithm

The Floyd Warshall Algorithm is an all pair shortest path algorithm unlike Dijkstra and Bellman Ford which are single source shortest path algorithms. This algorithm works for both the directed and undirected weighted graphs. But, it does not work for the graphs with negative cycles (where the sum of the edges in a cycle is negative). It follows Dynamic Programming approach to check every possible path going via every possible node in order to calculate shortest distance between every pair of nodes.

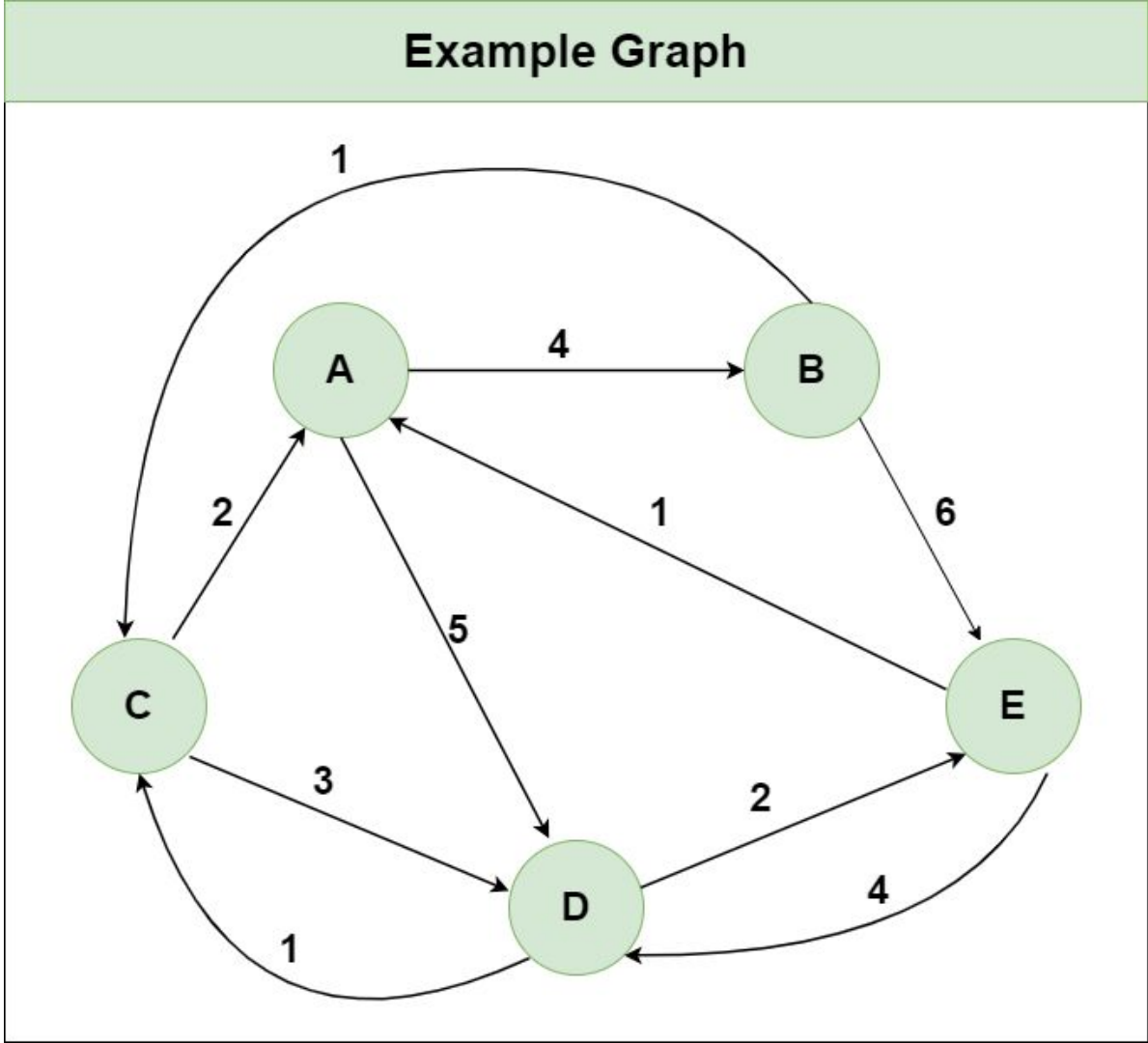
Obviously the shortest path between i to j will have some k number of intermediate nodes. The idea behind floyd warshall algorithm is to treat each and every vertex from 1 to N as an intermediate node one by one.

Pseudo code

```
For k = 0 to n – 1
For i = 0 to n – 1
For j = 0 to n – 1
Distance[i, j] = min(Distance[i, j], Distance[i, k] + Distance[k, j])

where i = source Node, j = Destination Node, k = Intermediate Node
```

Illustration



Continued

Step1: Initializing Distance[ ][ ] using the Input Graph

	A	B	C	D	E
A	0	4	∞	5	∞
B	∞	0	1	∞	6
C	2	∞	0	3	∞
D	∞	∞	1	0	2
E	1	∞	∞	4	0

Step 2: Using Node A as the Intermediate node

Distance[i][j] = min (Distance[i][j], Distance[i][A] + Distance[A][j])

	A	B	C	D	E
A	0	4	∞	5	∞
B	∞	?	?	?	?
C	2	?	?	?	?
D	∞	?	?	?	?
E	1	?	?	?	?

	A	B	C	D	E
A	0	4	∞	5	∞
B	∞	0	1	∞	6
C	2	6	0	3	12
D	∞	∞	1	0	2
E	1	5	∞	4	0

Step 3: Using Node B as the Intermediate node

Distance[i][j] = min (Distance[i][j], Distance[i][B] + Distance[B][j])

	A	B	C	D	E
A	?	4	?	?	?
B	∞	0	1	∞	6
C	?	6	?	?	?
D	?	∞	?	?	?
E	?	5	?	?	?

	A	B	C	D	E
A	0	4	5	5	10
B	∞	0	1	∞	6
C	2	6	0	3	12
D	∞	∞	1	0	2
E	1	5	6	4	0

Step 4: Using Node C as the Intermediate node

Distance[i][j] = min (Distance[i][j], Distance[i][C] + Distance[C][j])

	A	B	C	D	E
A	?	?	5	?	?
B	?	?	1	?	?
C	2	6	0	3	12
D	?	?	1	?	?
E	?	?	6	?	?

	A	B	C	D	E
A	0	4	5	5	10
B	3	0	1	4	6
C	2	6	0	3	12
D	3	7	1	0	2
E	1	5	6	4	0

Step 5: Using Node D as the Intermediate node

Distance[i][j] = min (Distance[i][j], Distance[i][D] + Distance[D][j])

	A	B	C	D	E
A	?	?	?	5	?
B	?	?	?	4	?
C	?	?	?	3	?
D	3	7	1	0	2
E	?	?	?	4	?

	A	B	C	D	E
A	0	4	5	5	7
B	3	0	1	4	6
C	2	6	0	3	5
D	3	7	1	0	2
E	1	5	5	4	0

Step 6: Using Node E as the Intermediate node

Distance[i][j] = min (Distance[i][j], Distance[i][E] + Distance[E][j])

	A	B	C	D	E
A	?	?	?	?	7
B	?	?	?	?	6
C	?	?	?	?	5
D	?	?	?	?	2
E	1	5	5	4	0

	A	B	C	D	E
A	0	4	5	5	7
B	3	0	1	4	6
C	2	6	0	3	5
D	3	7	1	0	2
E	1	5	5	4	0

Step 7: Return Distance[ ][ ] matrix as the result

	A	B	C	D	E
A	0	4	5	5	7
B	3	0	1	4	6
C	2	6	0	3	5
D	3	7	1	0	2
E	1	5	5	4	0

Important Facts about Floyd-Warshall

No matter how many edges are there in the graph the Floyd Warshall Algorithm runs for O(V3) times therefore it is best suited for Dense graphs. That's why Floyd-Warshall Algorithm better for Dense Graphs and not for Sparse Graphs.

- State some differences between Dijkstra's and floyd-warshall.
- State some difference between bellman-ford and floyd-warshall.

Minimum Spanning Tree

A **spanning tree** is defined as a tree-like subgraph of a connected, undirected graph that includes all the vertices of the graph. Or, to say in Layman's words, it is a subset of the edges of the graph that forms a tree (**acyclic**) where every node of the graph is a part of the tree.

The minimum spanning tree has all the properties of a spanning tree with an added constraint of having the minimum possible weights among all possible spanning trees. Like a spanning tree, there can also be many possible MSTs for a graph.

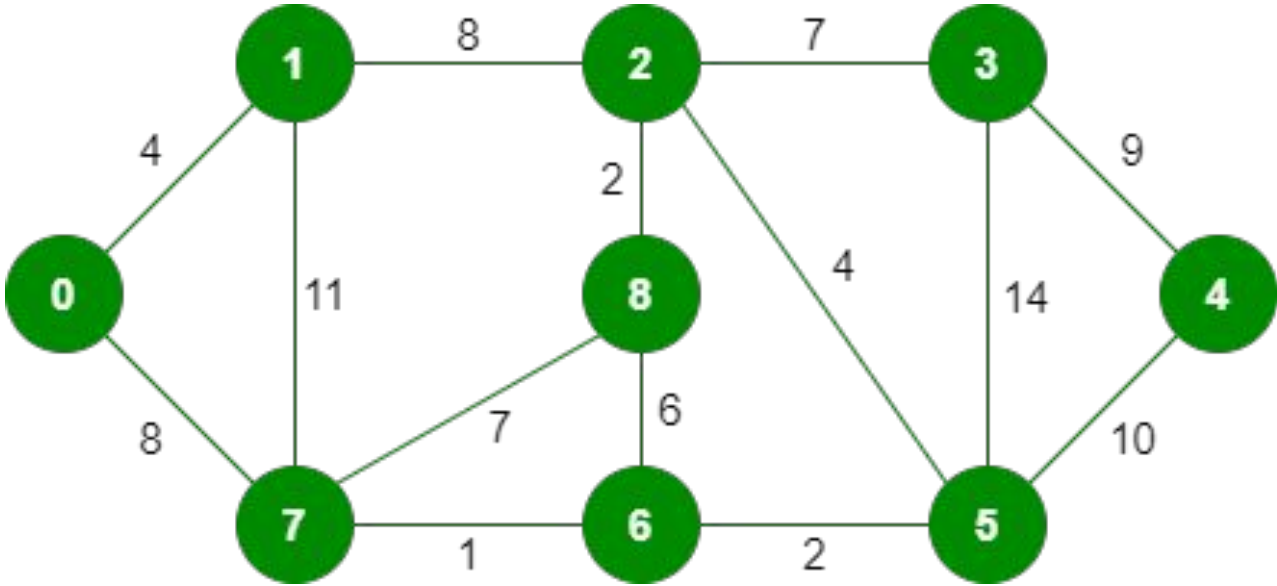
Properties of a Spanning Tree:

The spanning tree holds the **below-mentioned principles**:

- The number of vertices (**V**) in the graph and the spanning tree is the same.
- There is a fixed number of edges in the spanning tree which is equal to one less than the total number of vertices (**E = V-1**).
- The spanning tree should not be **disconnected**, as in there should only be a single source of component, not more than that.
- The spanning tree should be **acyclic,** which means there would not be any cycle in the tree.
- The total cost (or weight) of the spanning tree is defined as the sum of the edge weights of all the edges of the spanning tree.
- There can be many possible spanning trees for a graph.

Kruskal MST Algorithm

1. Sort all the edges in non-decreasing order of their weight.
2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If the cycle is not formed, include this edge. Else, discard it.
3. Repeat step#2 until there are (V-1) edges in the spanning tree.



Union-Find Algorithm

Step 2 of the previous slide uses union-find algorithm to detect cycles. Has two operation Union and Find.

```
#include<bits/stdc++.h>
using namespace std;

int find(int i)

{

    // If i is the parent of itself
    if (parent[i] == i) {

        // Then i is the representative of
        // this set
        return i;
    }
    else {

        // Else if i is not the parent of
        // itself, then i is not the
        // representative of his set. So we
        // recursively call Find on its parent
        return find(parent[i]);
    }
}

// The code is contributed by Nidhi goel
```

Inefficient approach could take O(n) in worst case.

```
#include <bits/stdc++.h>
using namespace std;

void union(int i, int j) {

    // Find the representatives
    // (or the root nodes) for the set
    // that includes i
    int irep = this.Find(i),

    // And do the same for the set
    // that includes j
    int jrep = this.Find(j);

    // Make the parent of i's representative
    // be j's  representative effectively
    // moving all of i's set into j's set)
    this.Parent[irep] = jrep;
}
```

Almost Constant Approach

Path Compression (Used to improve find()):

The idea of path compression is to make the found root as parent of x so that we don’t have to traverse all intermediate nodes again.

Union by Rank (Modifications to union()):

Rank is same as height if path compression is not used. With path compression, rank can be more than the actual height.

Now recall that in the Union operation, it doesn’t matter which of the two trees is moved under the other. Now what we want to do is minimize the height of the resulting tree.

Cycle detection In Kruskal:

If parent of two node is same it means there is a cycle so we avoid that edge.

```
def find(self, i):

    root = self.parent[i]

    if self.parent[root] != root:
        self.parent[i] = self.find(root)
        return self.parent[i]

    return root
```

```
def unionSets(self, x, y):
    xRoot = self.find(x)
    yRoot = self.find(y)

    if xRoot == yRoot:
        return

    # Union by Rank
    if self.rank[xRoot] < self.rank[yRoot]:
        self.parent[xRoot] = yRoot
    elif self.rank[yRoot] < self.rank[xRoot]:
        self.parent[yRoot] = xRoot
    else:
        self.parent[yRoot] = xRoot
        self.rank[xRoot] += 1
```

Prim’s Algorithm

Step 1: Determine an arbitrary vertex as the starting vertex of the MST.

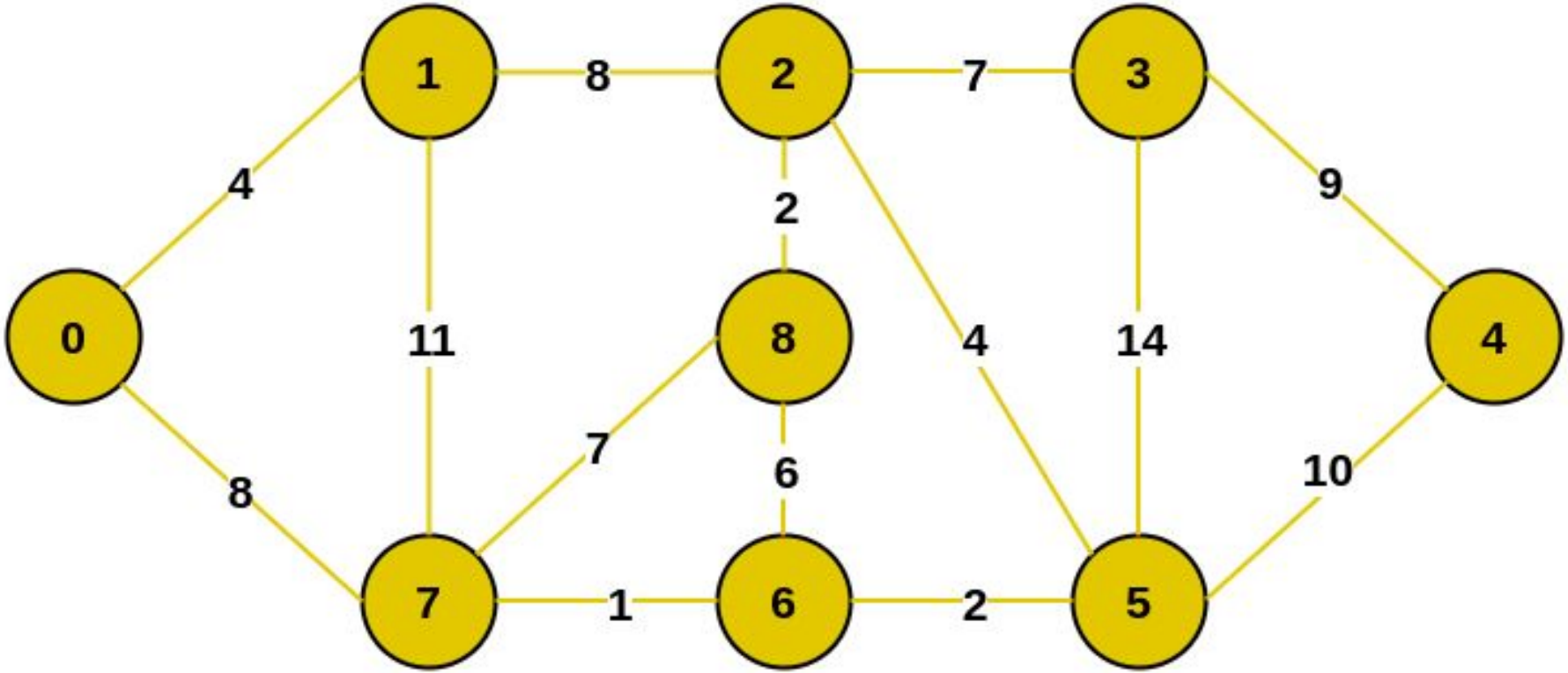
Step 2: Follow steps 3 to 5 till there are vertices that are not included in the MST (known as fringe vertex).

Step 3: Find edges connecting any tree vertex with the fringe vertices.

Step 4: Find the minimum among these edges.

Step 5: Add the chosen edge to the MST if it does not form any cycle.

Step 6: Return the MST and exit



Example of a Graph