Analysis of Algorithm

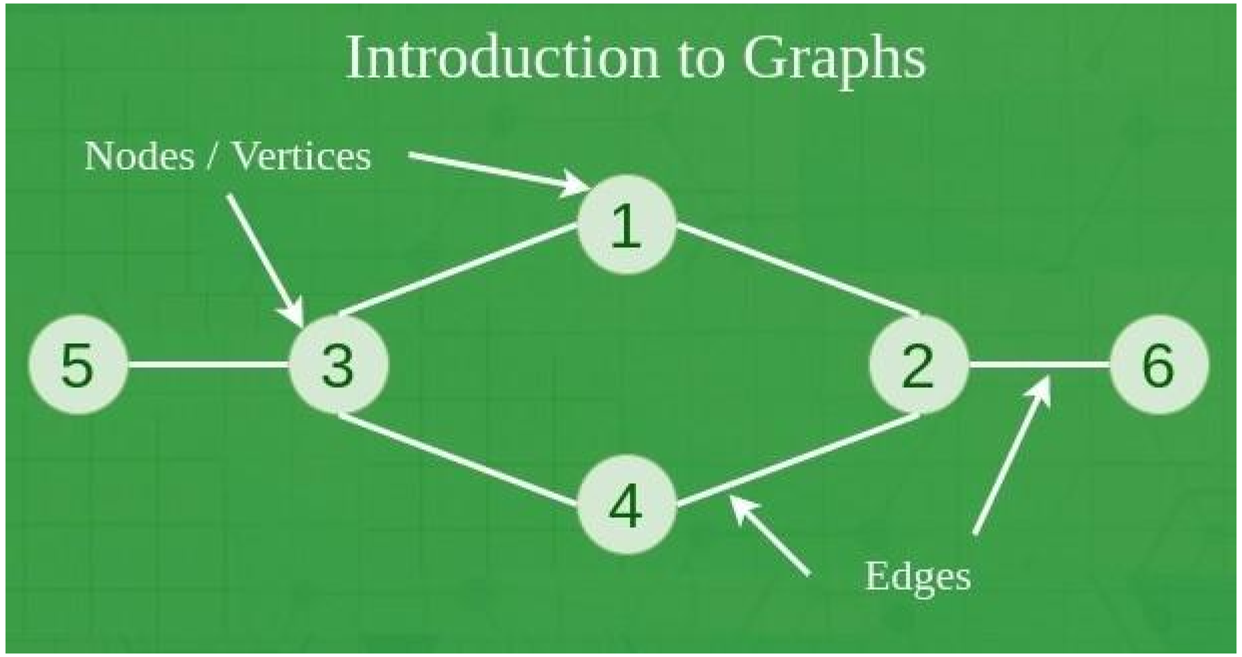
Amypo Technologies Pvt Ltd

Concepts:

- Graph Algorithm
- Shortest Path Algorithm
- Dijkstra Algorithm
- Floyd Warshall Algorithm

Graph Algorithm

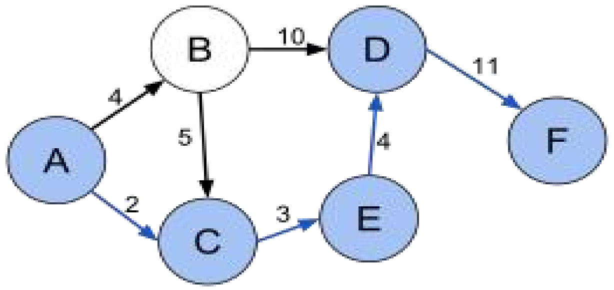
A Graph Data Structure is a collection of nodes connected by edges. It’s used to represent relationships between different entities. Graph algorithms are methods used to manipulate and analyze graphs, solving various problems like finding the shortest path or detecting cycles.



Shortest Path Algorithm

- In graph theory, the shortest path problem is the problem of finding a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized.

- The problem of finding the shortest path between two intersections on a road map may be modeled as a special case of the shortest path problem in graphs, where the vertices correspond to intersections and the edges correspond to road segments, each weighted by the length of the segment.



Dijkstra Algorithm

The idea is to generate a SPT (shortest path tree) with a given source as a root. Maintain an Adjacency Matrix with two sets,

- one set contains vertices included in the shortest-path tree,
- other set includes vertices not yet included in the shortest-path tree.

At every step of the algorithm, find a vertex that is in the other set (set not yet included) and has a minimum distance from the source.

Algorithm:

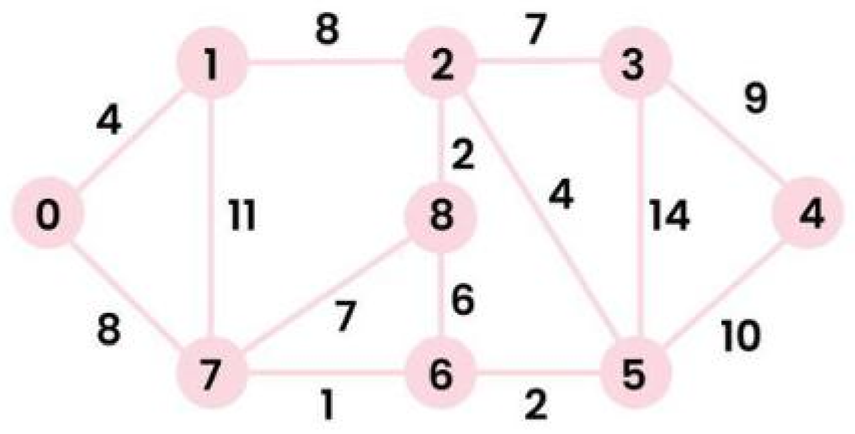
- Create a set sptSet (shortest path tree set) that keeps track of vertices included in the shortest path tree, i.e., whose minimum distance from the source is calculated and finalized. Initially, this set is empty.
- Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign the distance value as 0 for the source vertex so that it is picked first.
- While sptSet doesn't include all vertices
- Pick a vertex u that is not there in sptSet and has a minimum distance value.
- Include u to sptSet.

- Then update the distance value of all adjacent vertices of u.

  - To update the distance values, iterate through all adjacent vertices.
  - For every adjacent vertex v, if the sum of the distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.

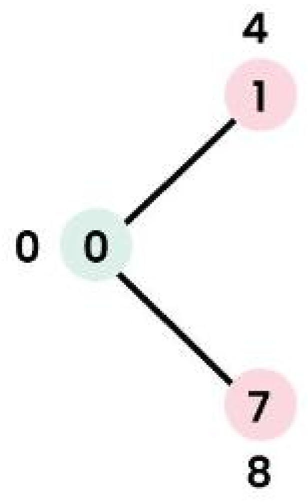
Illustration of Dijkstra Algorithm:

Consider below graph and src = 0



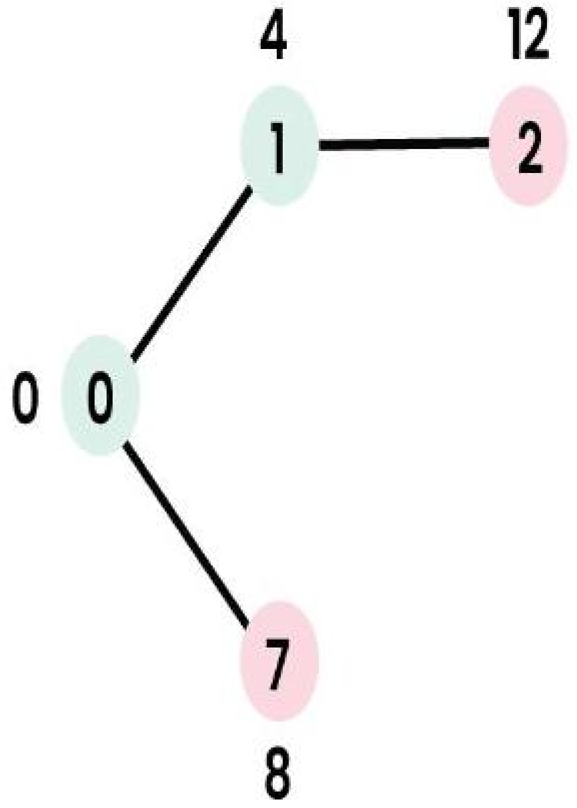
Step 1:

- *The set **sptSet** is initially empty and distances assigned to vertices are {0, INF, INF, INF, INF, INF, INF, INF} where **INF** indicates infinite.*
- *Now pick the vertex with a minimum distance value. The vertex 0 is picked, include it in **sptSet**. So **sptSet** becomes {0}. After including 0 to **sptSet**, update distance values of its adjacent vertices.*
- *Adjacent vertices of 0 are 1 and 7. The distance values of 1 and 7 are updated as 4 and 8.*



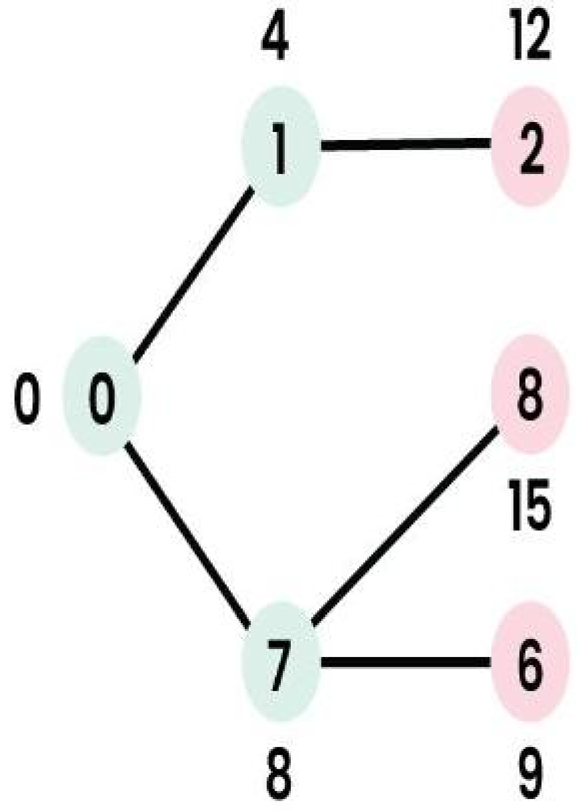
Step 2:

- *Pick the vertex with minimum distance value and not already included in **SPT** (not in **sptSET**). The vertex 1 is picked and added to **sptSet**.*
- *So **sptSet** now becomes {0, 1}. Update the distance values of adjacent vertices of 1.*
- *The distance value of vertex 2 becomes **12**.*



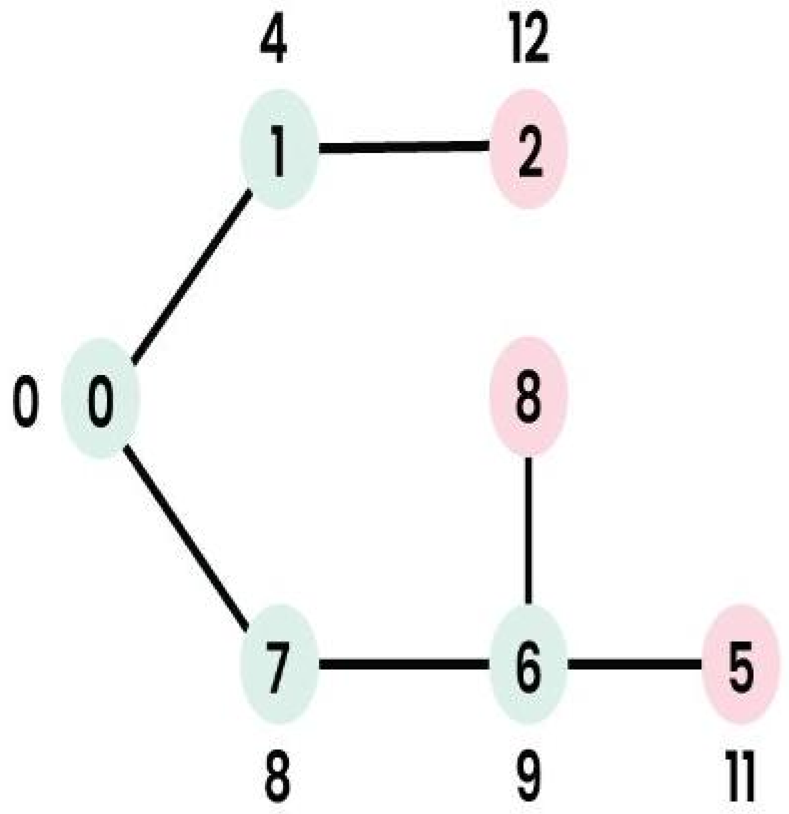
Step 3:

- *Pick the vertex with minimum distance value and not already included in **SPT** (not in **sptSET**). Vertex 7 is picked. So **sptSet** now becomes **{0, 1, 7}**.*
- *Update the distance values of adjacent vertices of 7. The distance value of vertex 6 and 8 becomes finite (**15 and 9** respectively).*

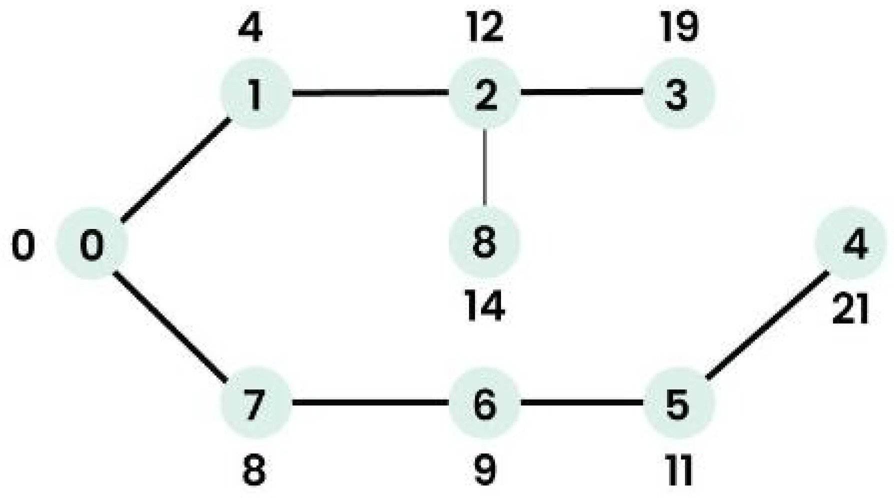


Step 4:

- *Pick the vertex with minimum distance value and not already included in **SPT** (not in **sptSET**). Vertex 6 is picked. So **sptSet** now becomes **{0, 1, 7, 6}**.*
- *Update the distance values of adjacent vertices of 6. The distance value of vertex 5 and 8 are updated.*



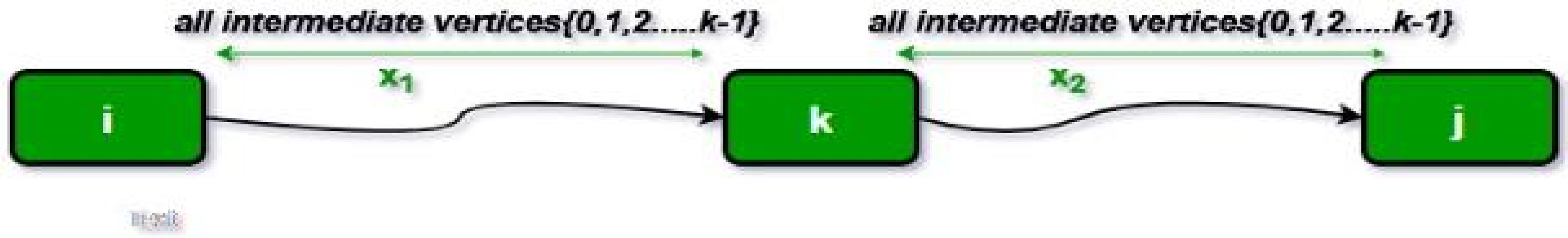
*We repeat the above steps until **sptSet** includes all vertices of the given graph. Finally, we get the following **Shortest Path Tree (SPT)**.*



Floyd Warshall Algorithm

- Suppose we have a graph G[][] with V vertices from 1 to N. Now we have to evaluate a shortestPathMatrix[][] where shortestPathMatrix[i][j] represents the shortest path between vertices i and j.

- 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.

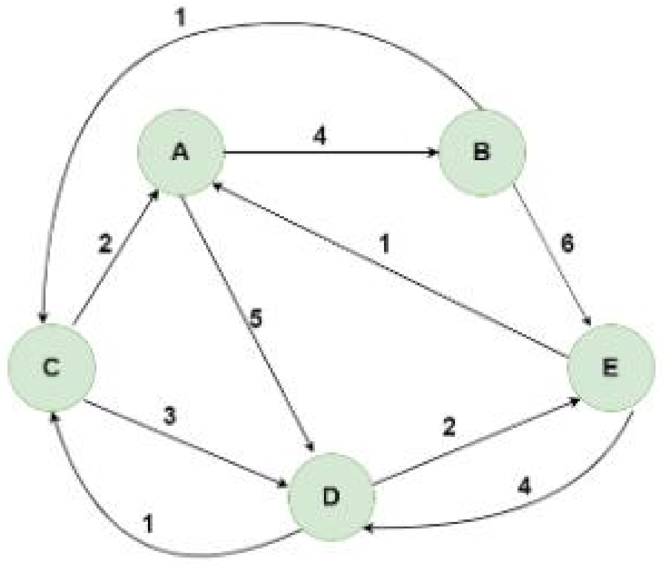

all intermediate vertices{0,1,2.....k-1}
all intermediate vertices{0,1,2.....k-1}
x1
x2
i
k
j

Algorithm:

- Initialize the solution matrix same as the input graph matrix as a first step.
- Then update the solution matrix by considering all vertices as an intermediate vertex.
- The idea is to pick all vertices one by one and updates all shortest paths which include the picked vertex as an intermediate vertex in the shortest path.
- When we pick vertex number k as an intermediate vertex, we already have considered vertices {0, 1, 2, .. k-1} as intermediate vertices.
- For every pair (i, j) of the source and destination vertices respectively, there are two possible cases.

    - k is not an intermediate vertex in shortest path from i to j. We keep the value of dist[i][j] as it is.
    - k is an intermediate vertex in shortest path from i to j. We update the value of dist[i][j] as dist[i][k] + dist[k][j], if dist[i][j] > dist[i][k] + dist[k][j]

Illustration

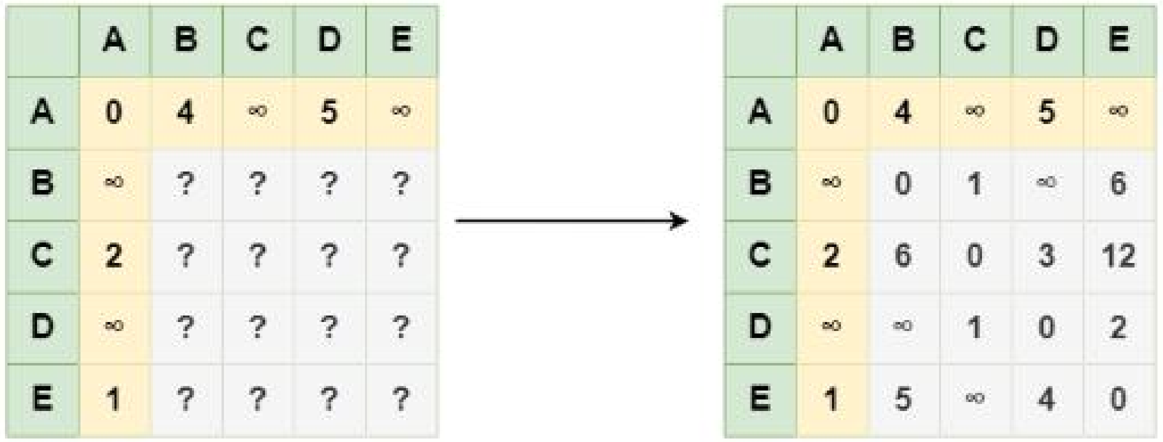


***Step 1**: Initialize the Distance[][] matrix using the input graph such that Distance[i][j]= weight of edge from **i** to **j**, also Distance[i][j] = Infinity if there is no edge from **i** to **j**.*

	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

***<u>Step 2</u>**: Treat node **A** as an intermediate node and calculate the Distance[][] for every {i,j} node pair using the formula:*

*= Distance[i][j] = minimum (Distance[i][j], (Distance from i to **A**) + (Distance from **A** to j))*
*= Distance[i][j] = minimum (Distance[i][j], Distance[i][**A**] + Distance[**A**][j])*



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

$\longrightarrow$

	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**: Treat node **B** as an intermediate node and calculate the Distance[][] for every {i,j} node pair using the formula:*

*= Distance[i][j] = minimum (Distance[i][j], (Distance from i to **B**) + (Distance from **B** to j))*

*= Distance[i][j] = minimum (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: *Treat node **C** as an intermediate node and calculate the Distance[][] for every {i,j} node pair using the formula:*

*= Distance[i][j] = minimum (Distance[i][j], (Distance from i to **C**) + (Distance from **C** to j))*

*= Distance[i][j] = minimum (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: *Treat node **D** as an intermediate node and calculate the Distance[][] for every {i,j} node pair using the formula:*

*= Distance[i][j] = minimum (Distance[i][j], (Distance from i to **D**) + (Distance from **D** to j))*

*= Distance[i][j] = minimum (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: *Treat node* ***E*** *as an intermediate node and calculate the Distance[][] for every {i,j} node pair using the formula:*

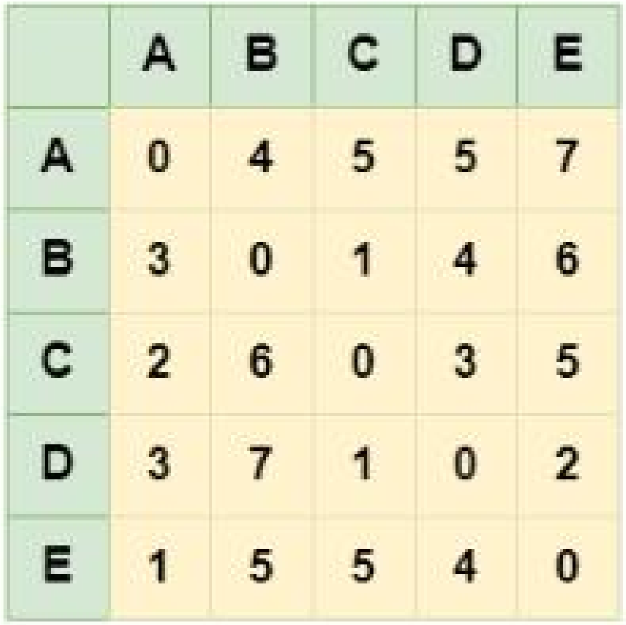
= Distance[i][j] = minimum (Distance[i][j], (Distance from i to ***E****) + (Distance from* ***E*** *to j))*

*= Distance[i][j] = minimum (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**: Since all the nodes have been treated as an intermediate node, we can now return the updated Distance[][] matrix as our answer matrix.*



	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