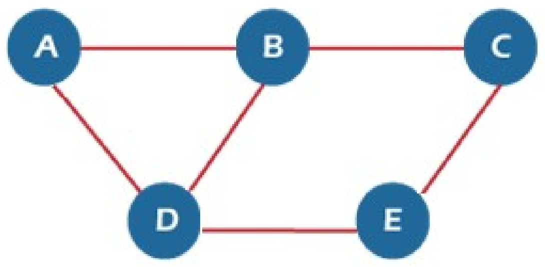
Graph Data Structure:-

- Graph is a non-linear data structure.
- It is a collection of nodes or vertex.
- It is a network of nodes.
- It is a connection of edge.



Basic Terminology of Graph Data Structure:-

Node/vertex/vertices:-Every individual elements in graph is known as node/vertex/vertices.

Example:-ABCDE is nodes.

Edge:-It is used to connect two nodes.

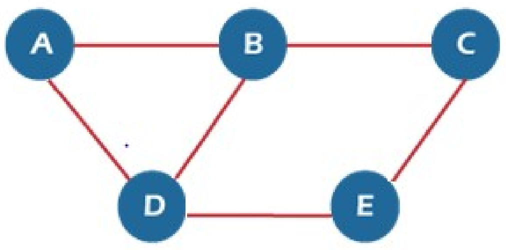
Degree of node:-The number of edges connected to any individual node is known as the degree of that node.

Weight:-It is a non-negative number assigned to the edge. It is also called length.

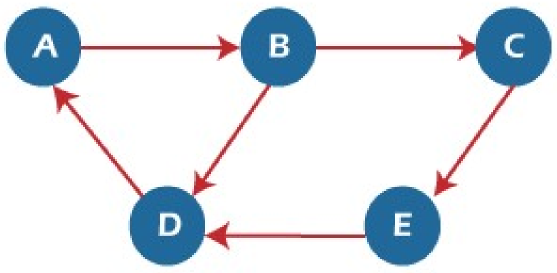
The most important Graph Data Structure types are.

- Undirected Graph.
- Directed Graph.
- Weighted Graph.
- Connected Graph.
- Disconnected Graph.

Undirected Graph:-In this graph there is no any fixed direction to move from one node to another node.OR we can say moving from one node to another node is unidirectional.

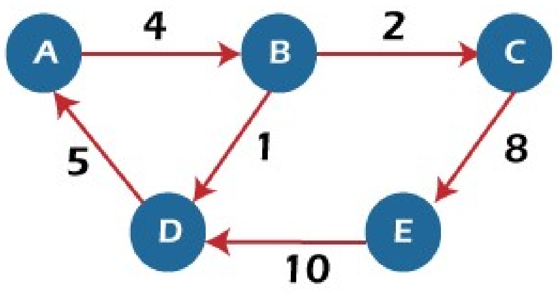


Directed Graph:- In this graph we move from one node to another node with given direction. OR we can say moving from one node to another node is unidirectional.



Directed Graph

Weighted Graph:-In this graph every edge weight is assigned.



weighted Directed Graph

Represent of Graph:-There are two ways to represent a graph.

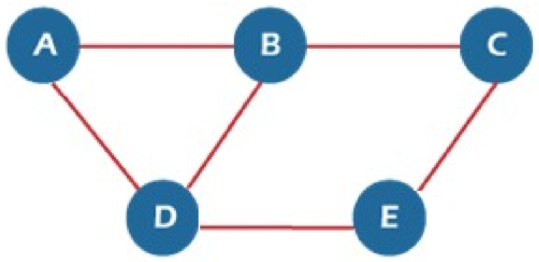
- Adjacency Matrix(2D Array).
- Adjacency List(List of List).

Adjacency Matrix(2D Array):-In this approach, we represent a graph in the form of the 2D matrix where rows and columns denote vertices(node).

```
Example:-
Suppose
arr[i][j]=1;
Where 1(one) indicates that there is one direct edge between vertex i
to vertex j.

And arr[i][j]=0;
Where 0(zero) indicates that there is no direct edge between vertex i
to vertex j.
```

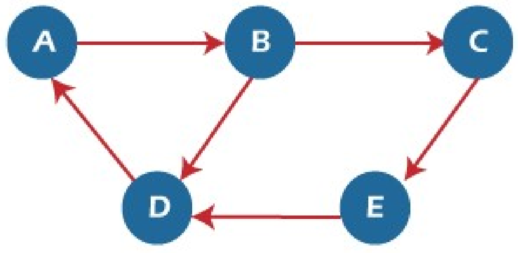
Adjacency Matrix(2D Array) representation of the given Graph.



Undirected Graph

$$\begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & 0 & 1 & 0 & 1 & 0 \\ B & 1 & 0 & 1 & 1 & 0 \\ C & 0 & 1 & 0 & 0 & 1 \\ D & 1 & 1 & 0 & 0 & 1 \\ E & 0 & 0 & 1 & 1 & 0 \end{array}$$

Adjacency Matrix

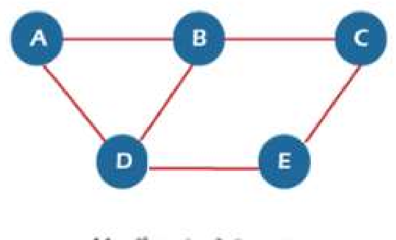


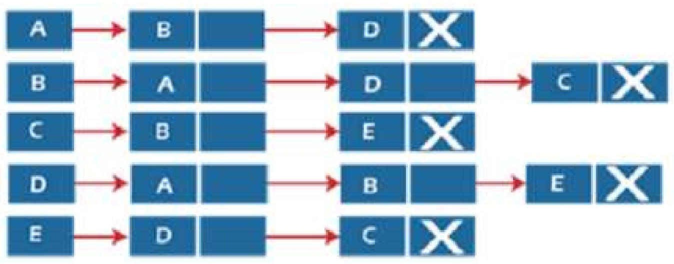
Directed Graph

$$\begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & 0 & 1 & 0 & 0 & 0 \\ B & 0 & 0 & 1 & 1 & 0 \\ C & 0 & 0 & 0 & 0 & 1 \\ D & 1 & 0 & 0 & 0 & 0 \\ E & 0 & 0 & 0 & 1 & 0 \end{array}$$

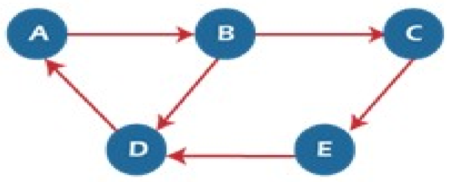
Adjacency Matrix

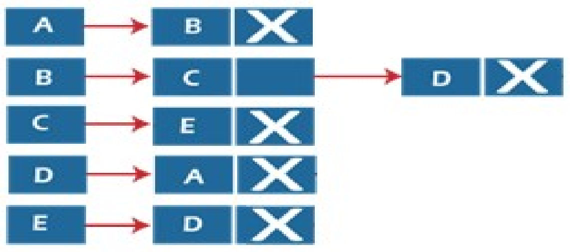
Adjacency List(List of List):-
In this approach we used LinkedList to represent graph.


Undirected Graph



Adjacency List


Directed Graph



Adjacency List

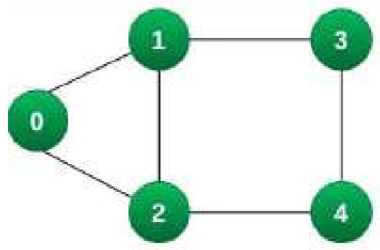
Traverse the Data from Graph:-There are two approach to traverse the data from Graph.

- BFS(Breadth First Search).
- DFS(Depth First Search).

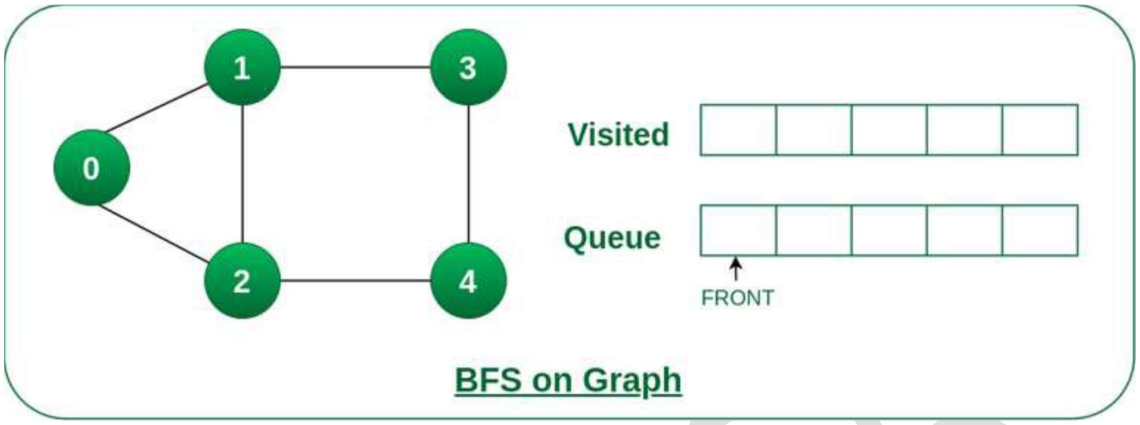
BFS(Breadth First Search):-

BFS stands for *Breadth First Search*. It is also known as level order traversal. The Queue data structure is used for the Breadth First Search traversal. When we use the BFS algorithm for the traversal in a graph, we can consider any node as a root node.

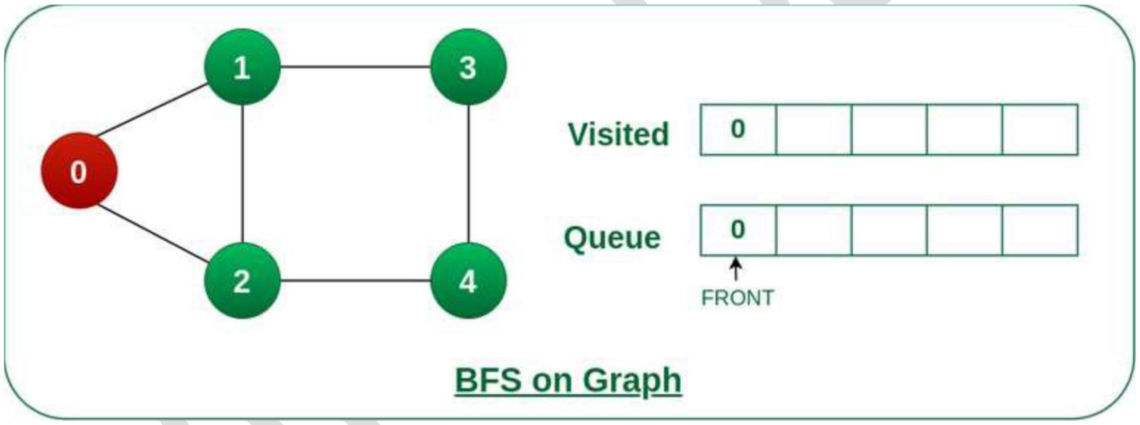
Perform BFS(Breadth First Search) traversal on given graph:-



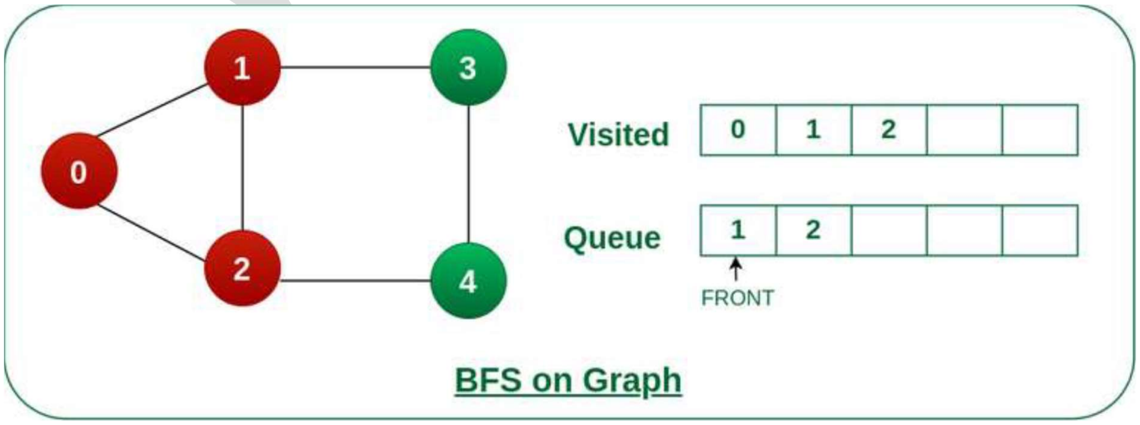
Step1:- Initially queue and visited arrays are empty.



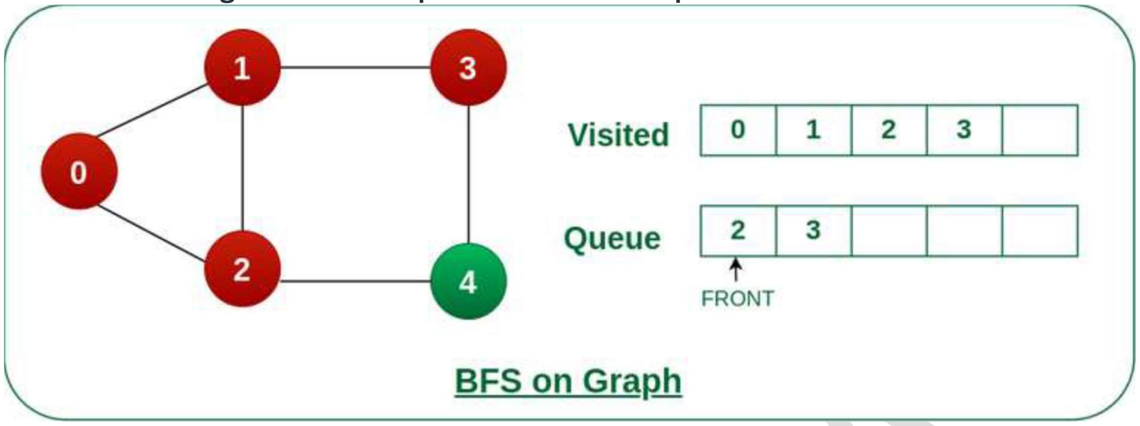
Step 2:- Push node 0 into queue and mark it visited.



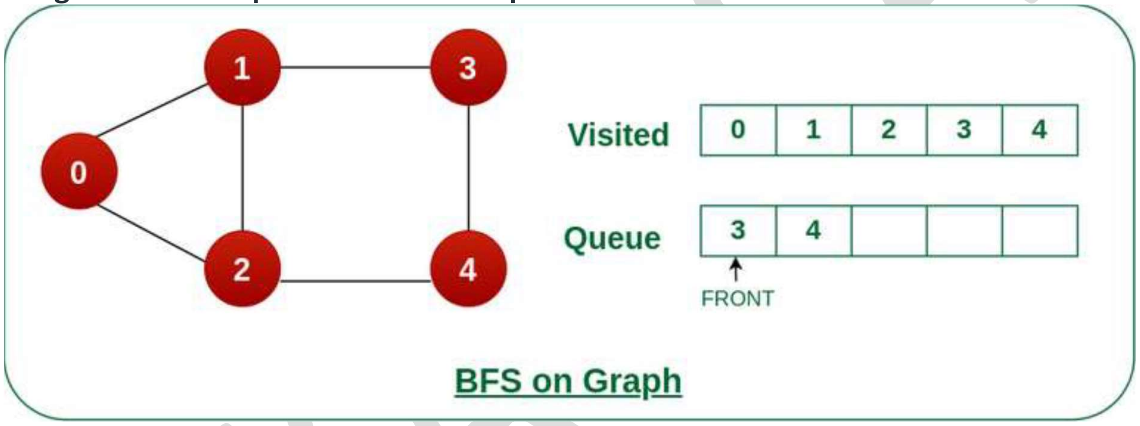
Step 3:- Remove node 0 from the front of queue and visit the unvisited neighbours and push them into queue.



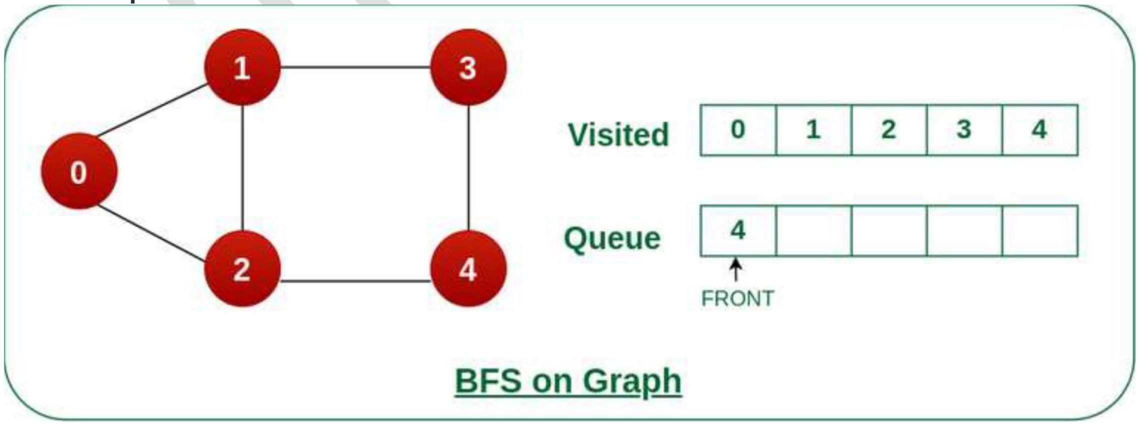
Step 4:- Remove node 1 from the front of queue and visit the unvisited neighbours and push them into queue.



Step 5:- Remove node 2 from the front of queue and visit the unvisited neighbours and push them into queue.

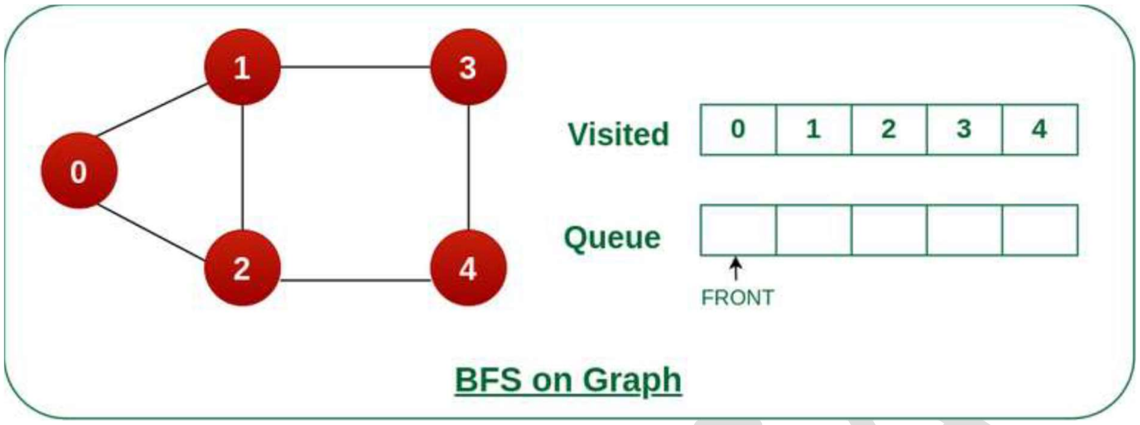


Step 6:- Remove node 3 from the front of queue and visit the unvisited neighbours and push them into queue. As we can see that every neighbour of node 3 is visited, so move to the next node that are in the front of the queue.



Steps 7:- Remove node 4 from the front of queue and visit the unvisited neighbours and push them into queue. As we can see that every neighbour

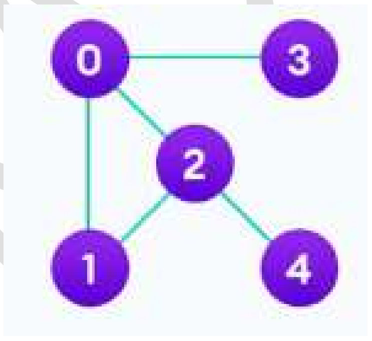
of node 4 are visited, so move to the next node that is in the front of the queue.



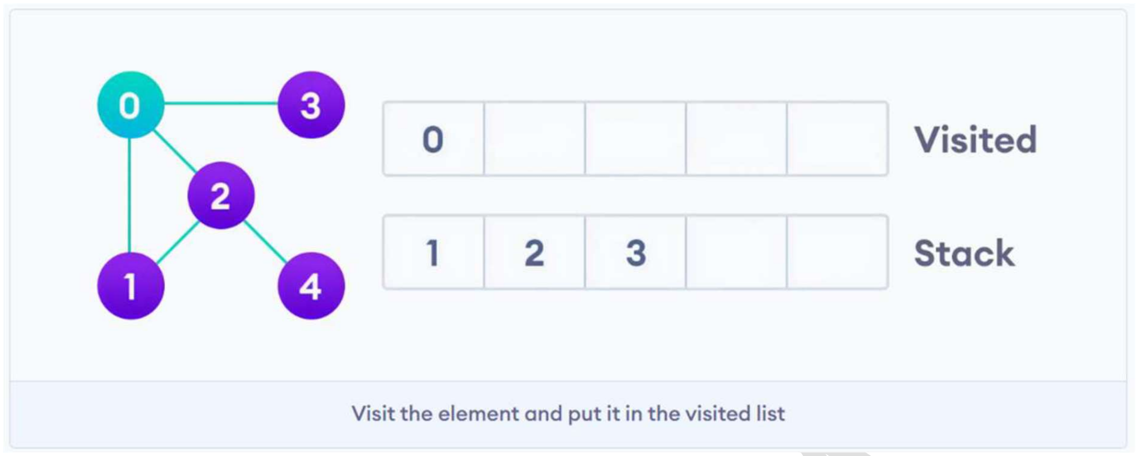
DFS(Depth First Search):-

DFS stands for Depth First Search. In DFS traversal, the stack data structure is used, which works on the LIFO (Last In First Out) principle. In DFS, traversing can be started from any node, or we can say that any node can be considered as a root node until the root node is not mentioned in the problem.

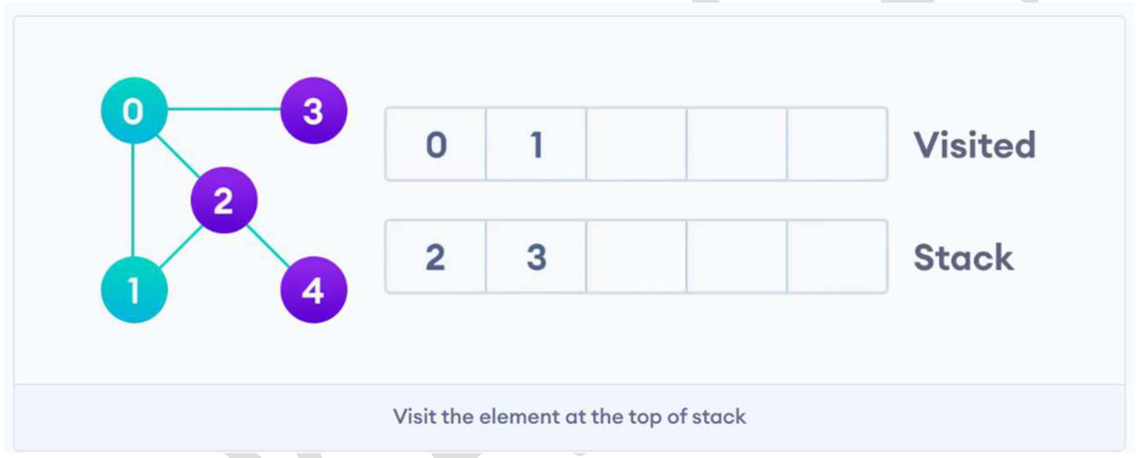
Perform DFS(Depth First Search) traversal on given graph:-



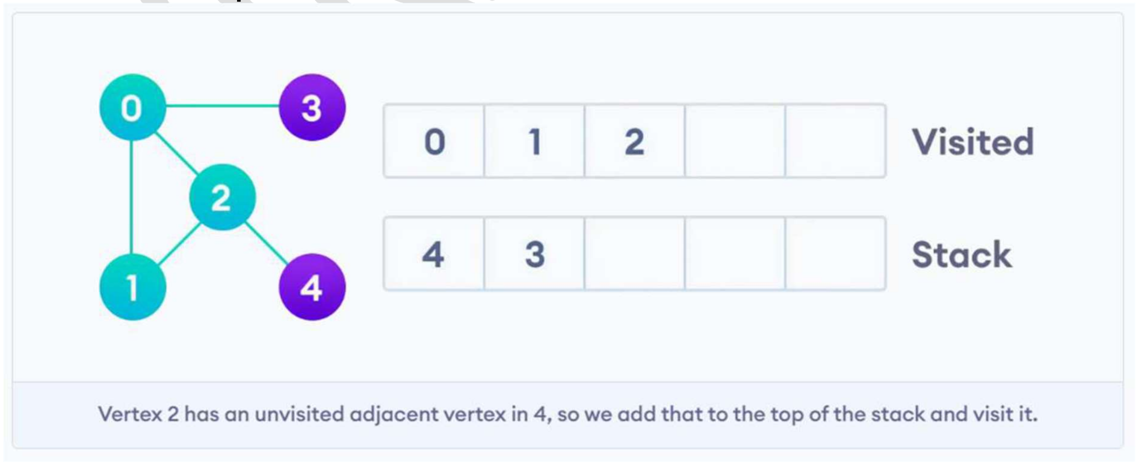
Step 1:- We start from vertex 0, the DFS algorithm starts by putting it in the Visited list and putting all its adjacent vertices in the stack.

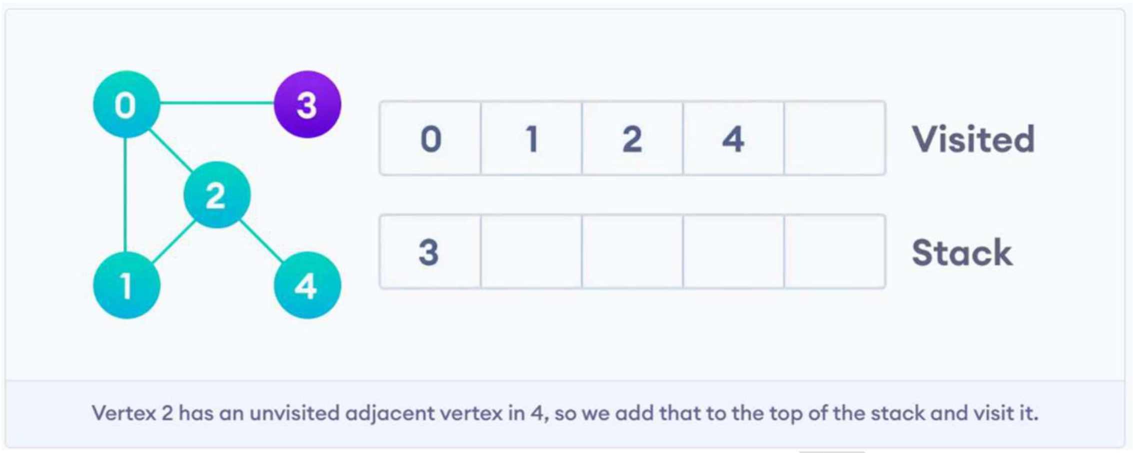


Step 2:- Next, we visit the element at the top of stack i.e. 1 and go to its adjacent nodes. Since 0 has already been visited, we visit 2 instead.



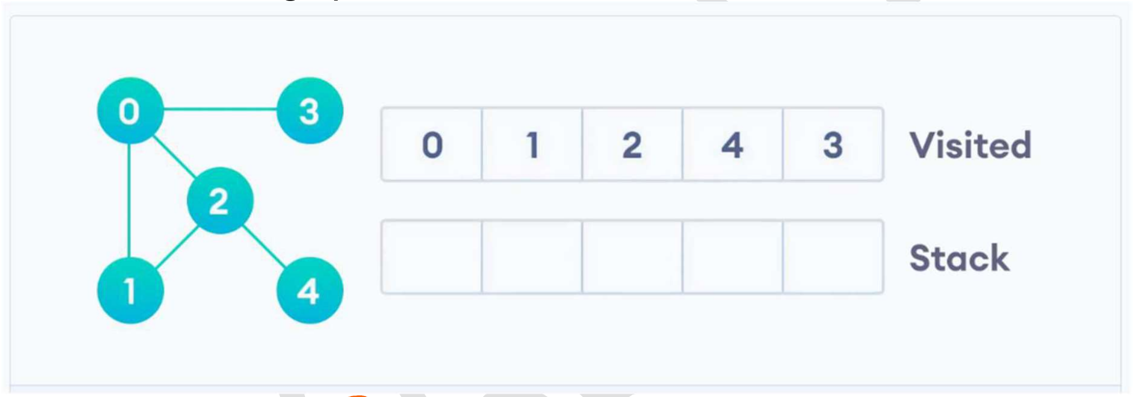
Step 3:- Vertex 2 has an unvisited adjacent vertex in 4, so we add that to the top of the stack and visit it.

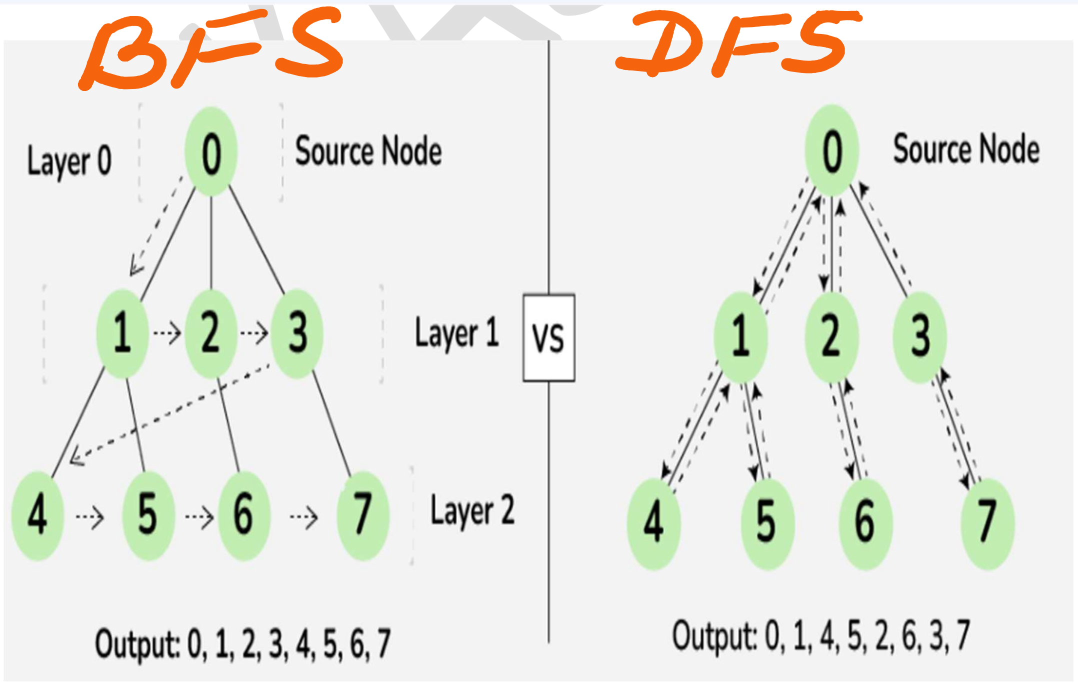

0
3
2
1
4
0
1
2
Visited
4
3
Stack
Vertex 2 has an unvisited adjacent vertex in 4, so we add that to the top of the stack and visit it.



Vertex 2 has an unvisited adjacent vertex in 4, so we add that to the top of the stack and visit it.

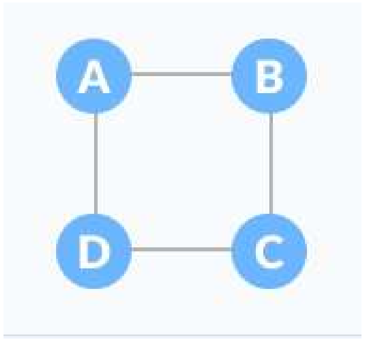
Step 4:- After we visit the last element 3, it doesn't have any unvisited adjacent nodes, so we have completed the Depth First Traversal of the graph.



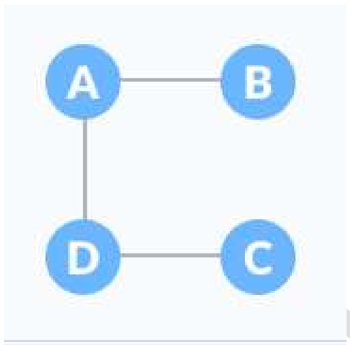


Spanning Tree:- A Spanning tree is a subset of graph G, which has all the vertices covered with the minimum possible number of edges. Hence Spanning tree does not have a cycle and it cannot be disconnected.

Example:-



Normal graph



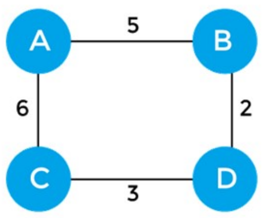
A spanning tree

Suppose we have n vertices in the graph then n^{n-2} number of Spanning tree will create.
In the above graph 4 vertices are available the possible number of Spanning tree are $4^{4-2} = 16$.

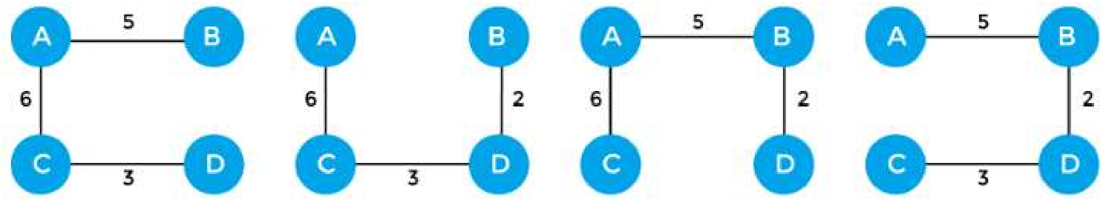
Application area of the Spanning tree:-

- Cluster Analysis.
- Civil Network Planning.
- Computer network routing protocol.

Minimum Spanning Tree:- A minimum spanning tree is a spanning tree in which the sum of the weight of the edge is the minimum possible.



Weighted graph



Sum = 14 Minimum spanning tree - 1
Sum = 11 Minimum spanning tree - 2
Sum = 13 Minimum spanning tree - 3
Sum = 10 Minimum spanning tree - 4

There are two algorithms available to find the minimum spanning tree.

1. Prim’s algorithm.
2. Kruskal's Algorithm

Prim’s algorithm:-By using this algorithm we find a minimum spanning tree.
This algorithm takes a graph as input and finds the minimum spanning tree with the following case.

- Create a tree that includes all vertices.
- Created tree weight should be minimum among all possible trees that can be formed from the graph.
- We start from one vertex and keep adding edges with the lowest weight until we reach our goals.

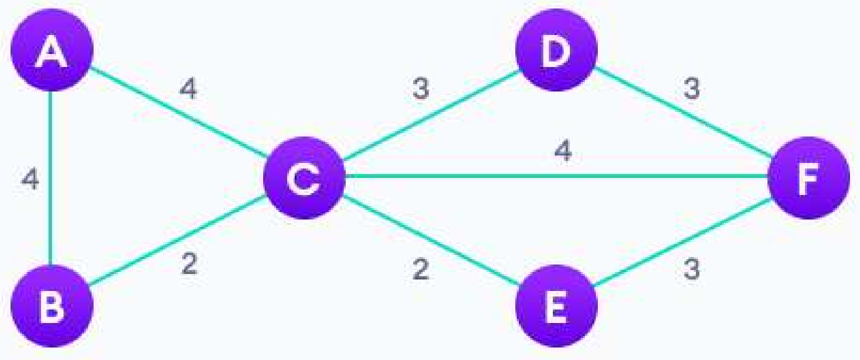
The steps for implementing Prim’s algorithm are as follows:-

Step:-1 Initialize the minimum spanning tree with a vertex choose at random.

Step:-2 Find all the edges that connect the tree to new vertices, find the minimum, and add it to the tree.

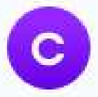
Step:-3 Keep repeating step 2 until we get a minimum spanning tree.

Example of Prim’s Algorithm.



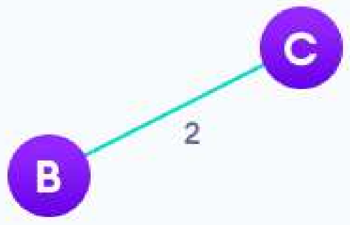
Step: 1

Start with a weighted graph



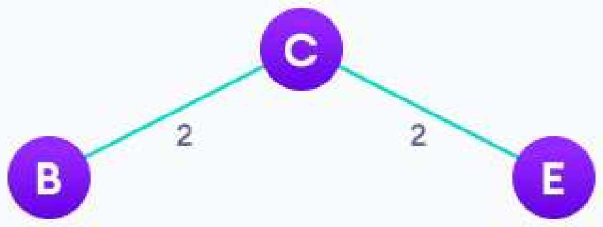
Step: 2

Choose a vertex



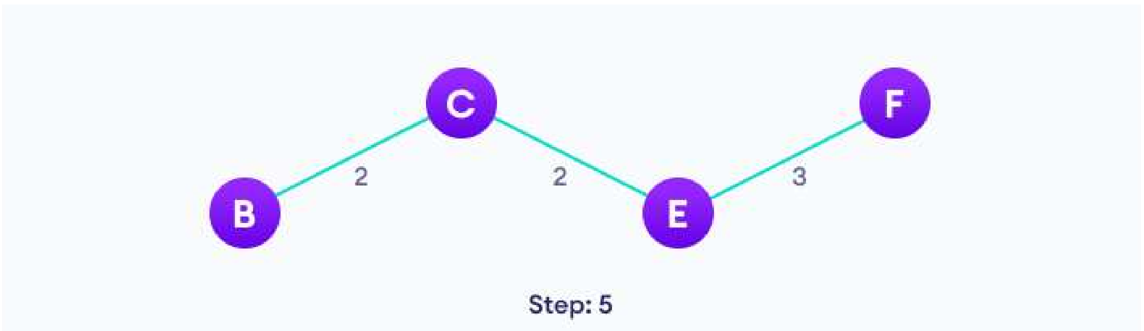
Step: 3

Choose the shortest edge from this vertex and add it

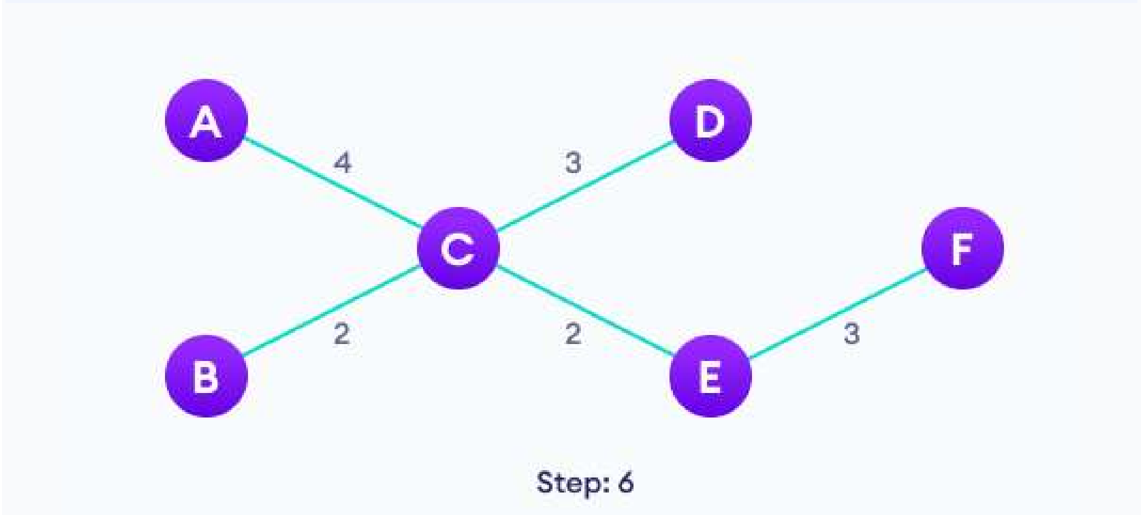


Step: 4

Choose the nearest vertex not yet in the solution



Choose the nearest edge not yet in the solution, if there are multiple choices, choose one at random



Repeat until you have a spanning tree

Kruskel's Algorithm:- By using this algorithm we find a minimum spanning tree.
This algorithm takes a graph as input and finds the minimum spanning tree with the following case.

- Create a tree that includes all vertices.
- Created tree weight should be minimum among all possible trees that can be formed from the graph.
- We start from one vertex and keep adding edges with the lowest weight until we reach our goals.

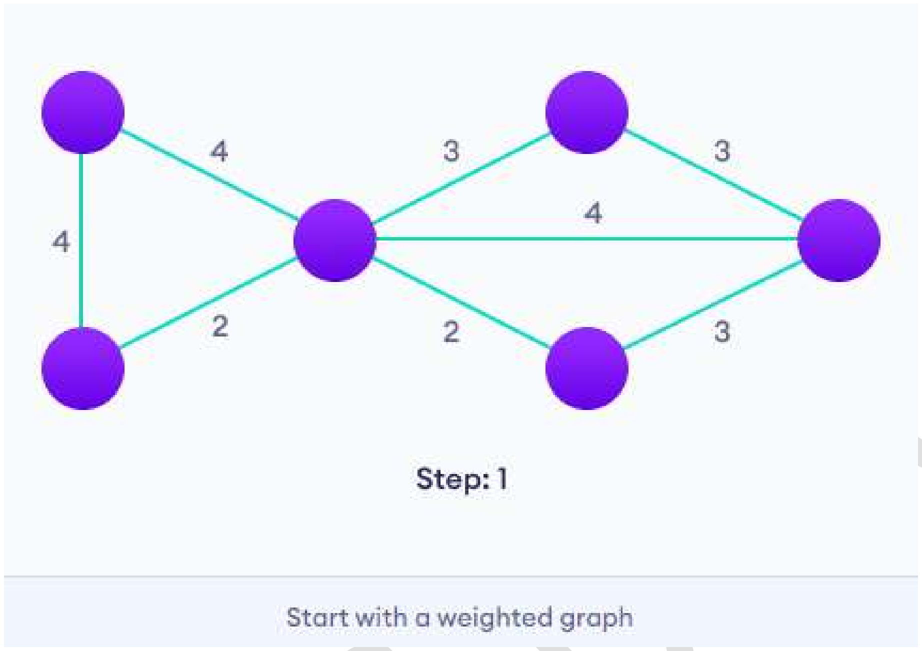
The steps for implementing Kruskal's algorithm are as follows:-

Step:-1 Sort all the edges from low weight to high

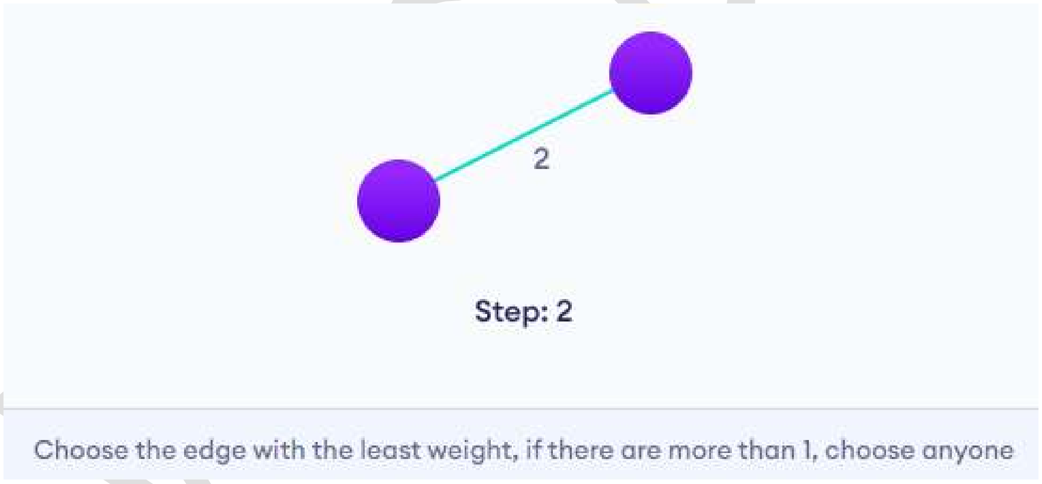
Step:-2 Take the edge with the lowest weight and add it to the spanning tree. If adding the edge created a cycle, then reject this edge.

Step:-3 Keep adding edges until we reach all vertices.

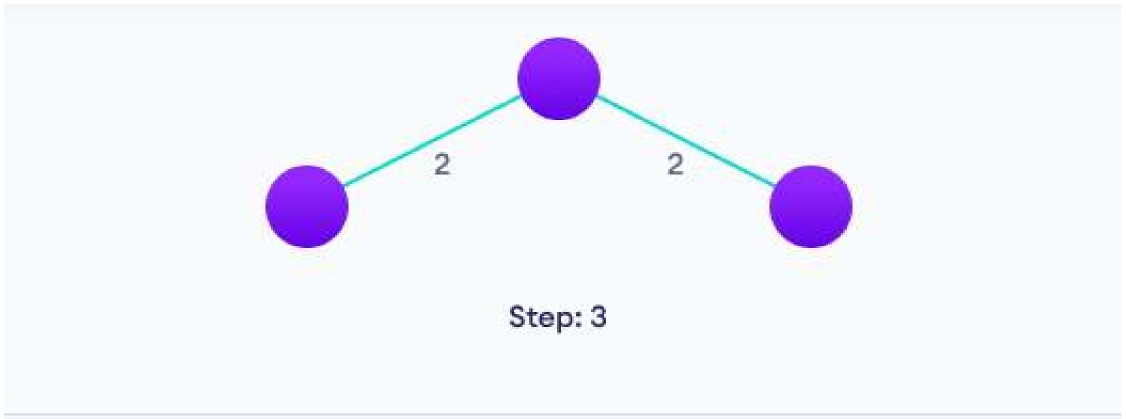
Example of Kruskal's algorithm:-



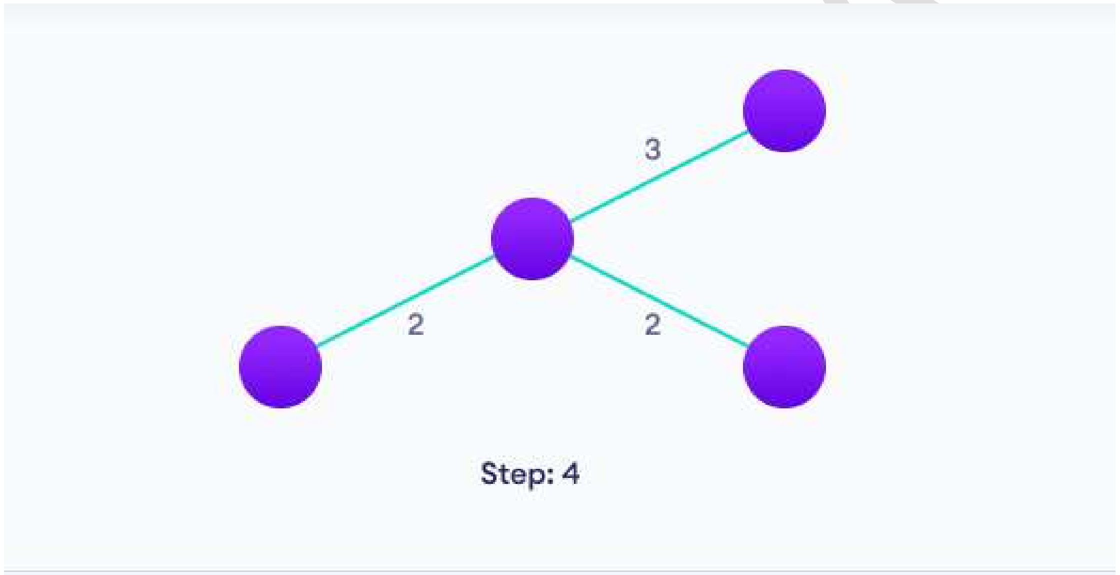
Start with a weighted graph



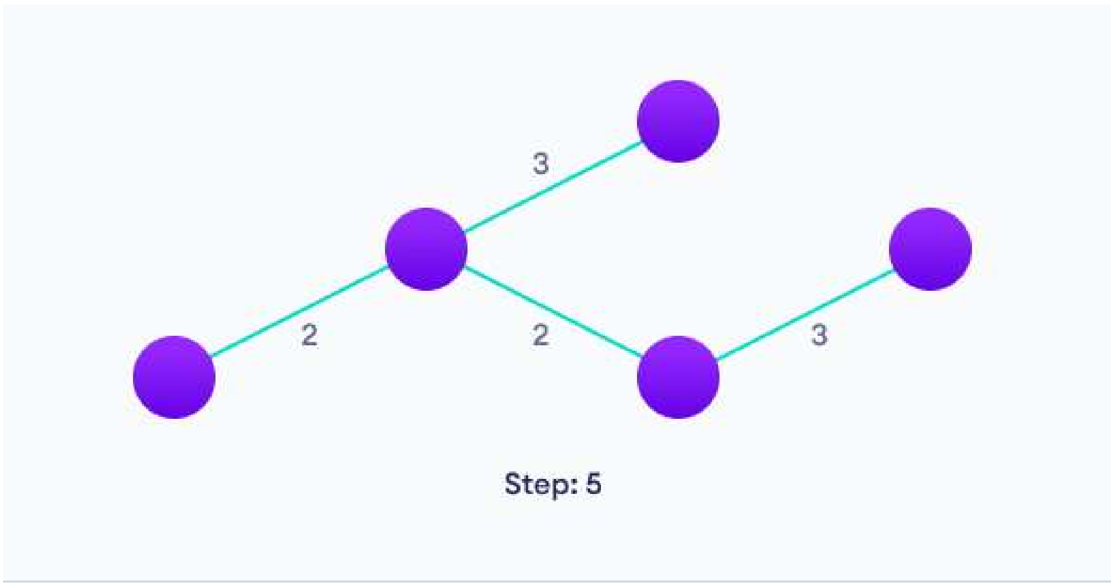
Choose the edge with the least weight, if there are more than 1, choose anyone



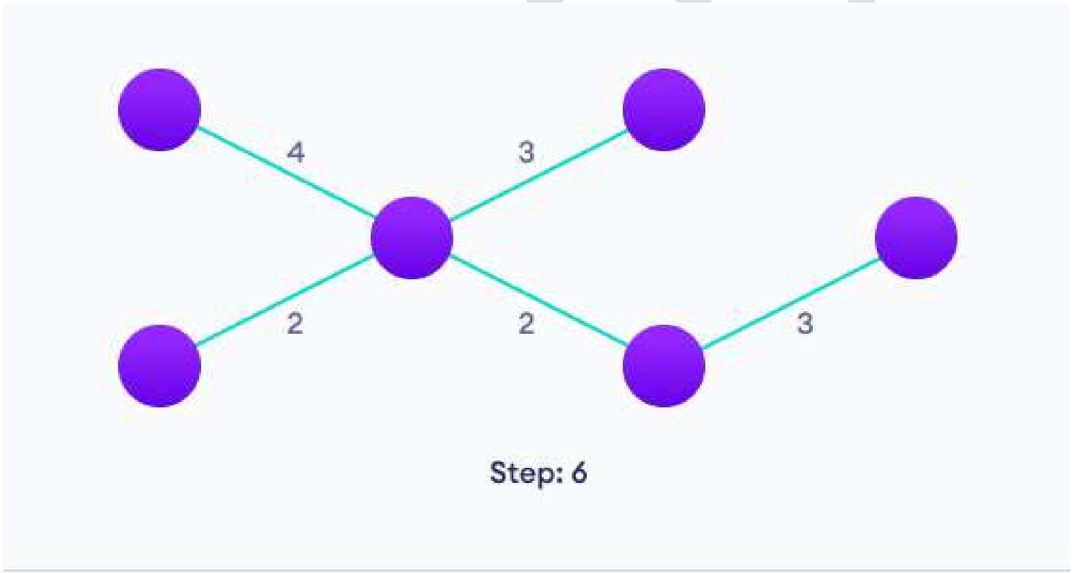
Choose the next shortest edge and add it



Choose the next shortest edge that doesn't create a cycle and add it



Choose the next shortest edge that doesn't create a cycle and add it



Repeat until you have a spanning tree

Shortest Path Graph:-

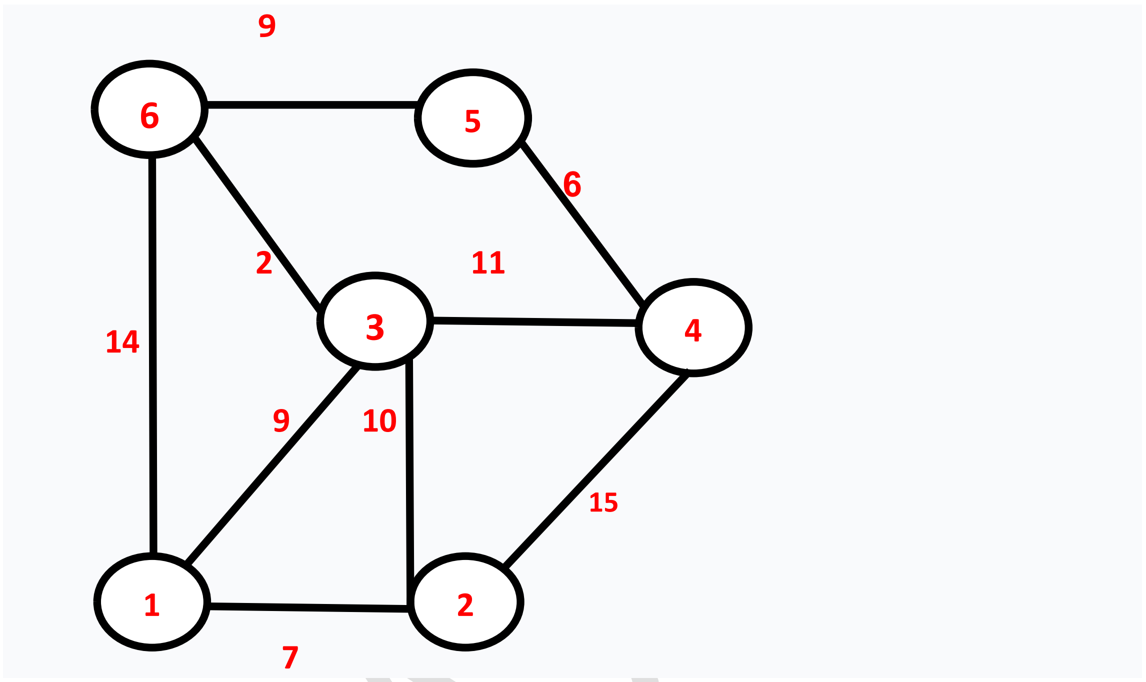
In the given graph we find the shortest path from the source vertices to all the given vertices (the source can be any one vertex if it is not mentioned).

There are two algorithms to find the shortest path in graph:-

1. **Dijkstra's Algorithm**
2. **Bellman Ford's Algorithm**

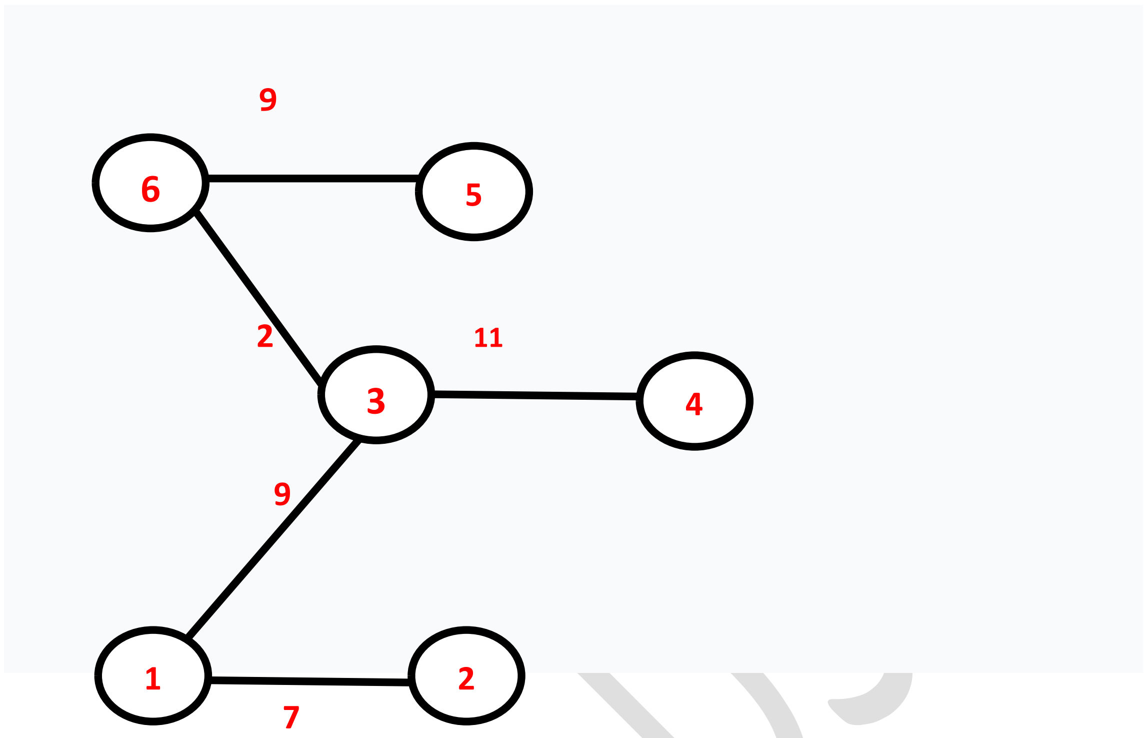
Dijkstra's Algorithm:- Dijkstra's Algorithm is a single source shortest path algorithm, here single source means that only one source is given and we have to find the shortest path from the source vertex to all the vertices of a weighted graph.

Example of Dijkstra's algorithm:-

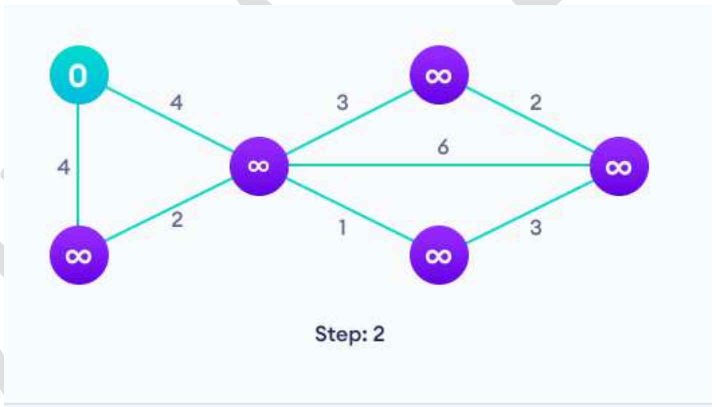


SOURCE **DESTINATION**

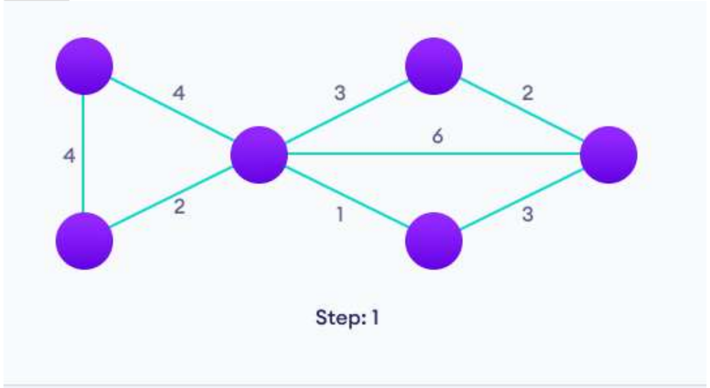
	2	3	4	5	6
	∞	∞	∞	∞	∞
1	7	9	∞	∞	14
1 2	7	9	22	∞	14
1 2 3	7	9	20	∞	11
1 2 3 6	7	9	20	20	11
1 2 3 6 4	7	9	20	20	11
1 2 3 6 4 5	7	9	20	20	11



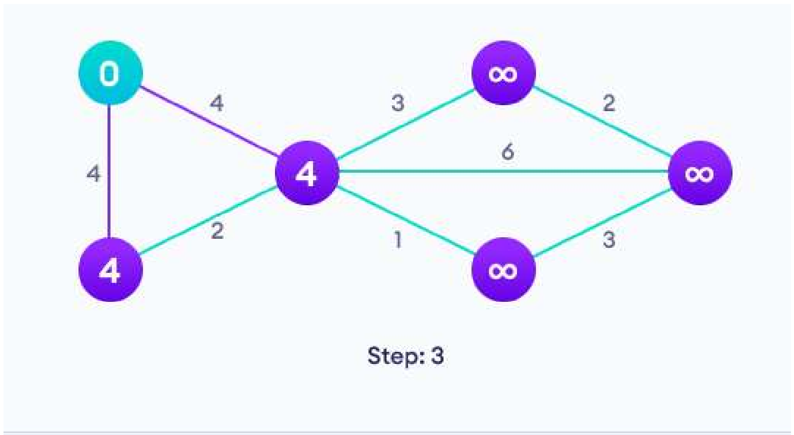
EXAMPLE:-2



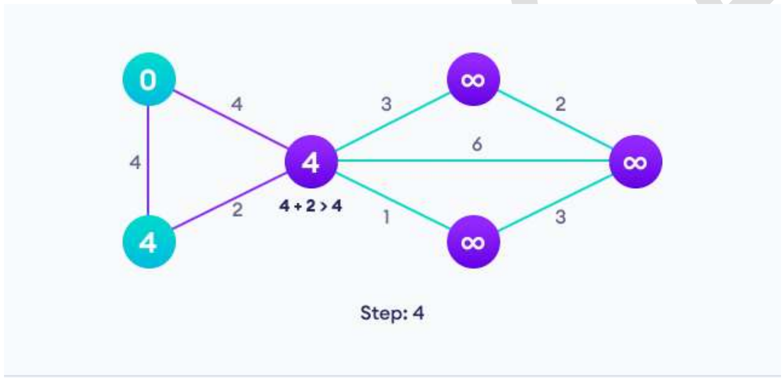
Choose a starting vertex and assign infinity path values to all other devices



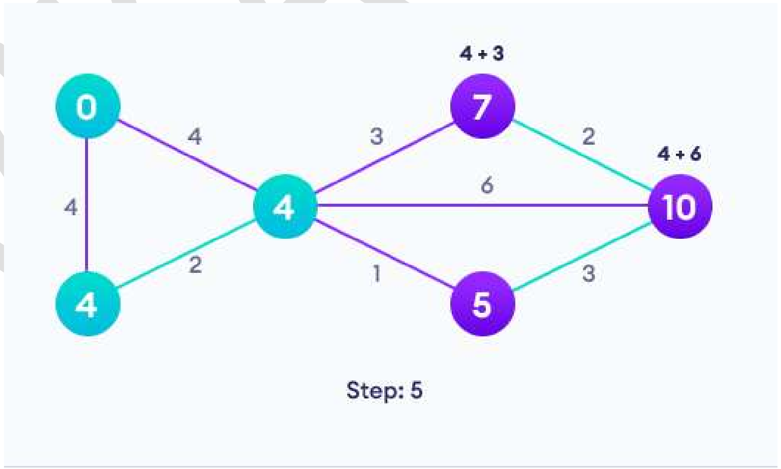
Start with a weighted graph



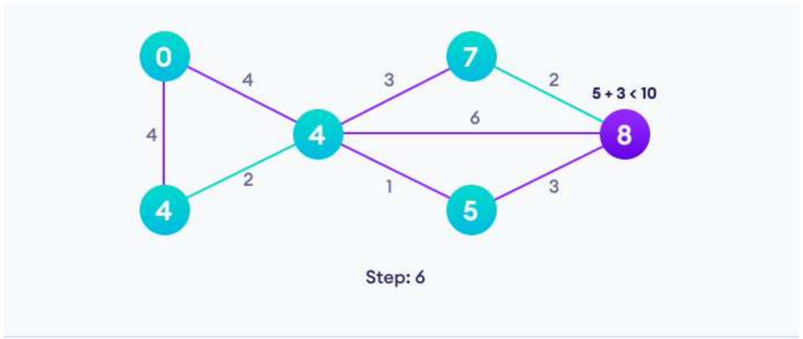
Go to each vertex and update its path length



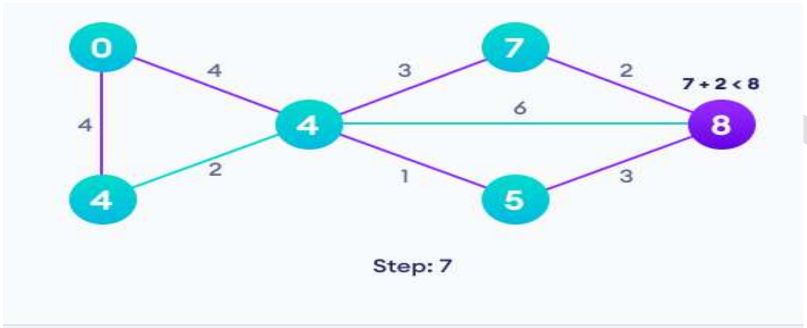
If the path length of the adjacent vertex is lesser than new path length, don't update it



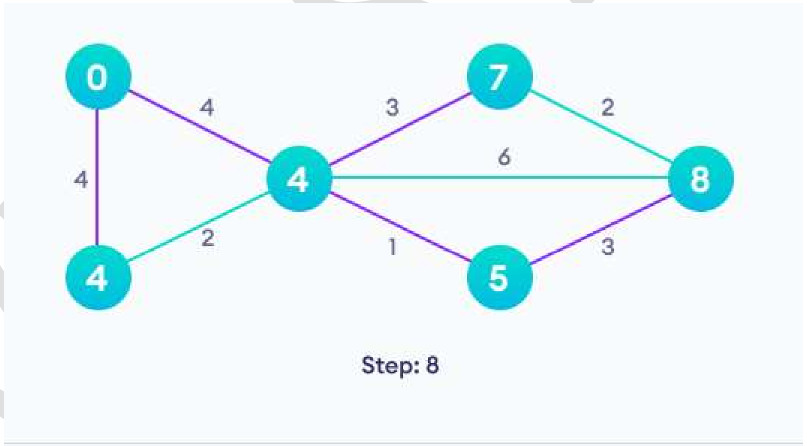
Avoid updating path lengths of already visited vertices



After each iteration, we pick the unvisited vertex with the least path length. So we choose 5 before 7



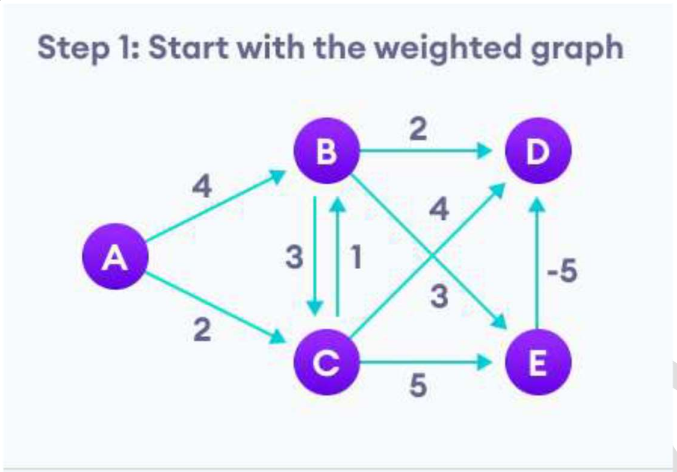
Notice how the rightmost vertex has its path length updated twice



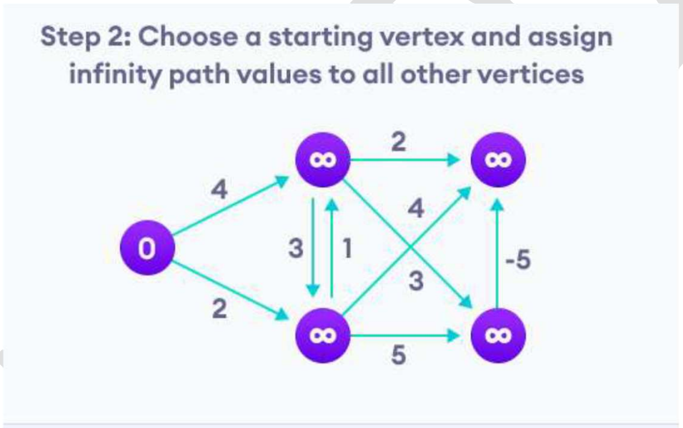
Repeat until all the vertices have been visited

Bellman Ford's Algorithm:- Bellman Ford's Algorithm is single source shortest path algorithm, this algorithm is used to find the shortest distance from the single source vertex to all other vertices of a weighted graph.
There are some problems in Dijkastra's algorithm, suppose the graph contains a negative value weight then Dijkastra's algorithm does confirm whether produces result is correct or not, To overcome these

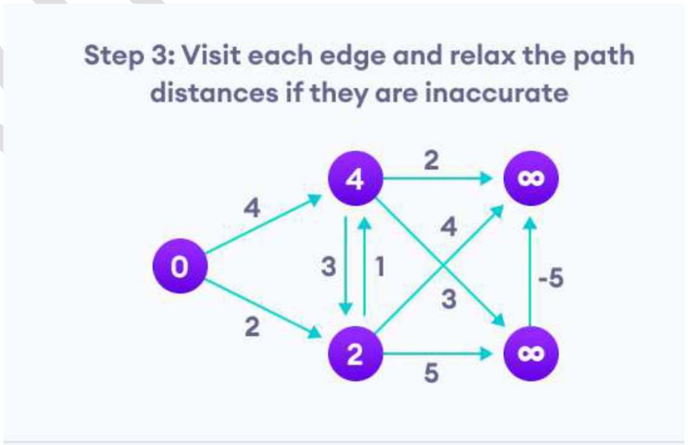
problems we used Bellman Ford's Algorithm, this algorithm give the guarantees the correct result even if a weighted graph contains negative weight value.



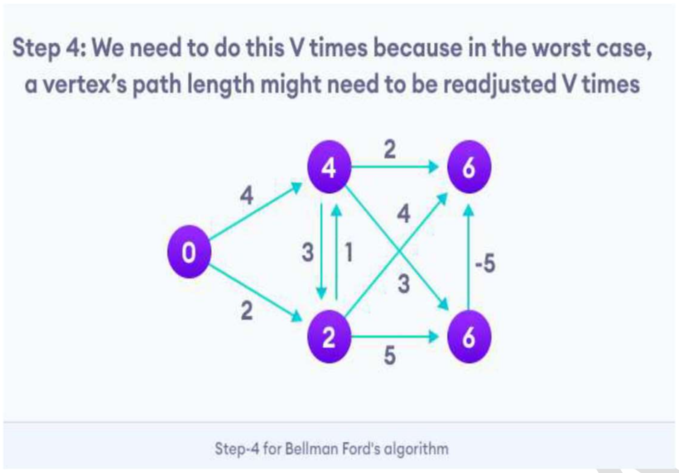
Step-1 for Bellman Ford's algorithm


Step 2: Choose a starting vertex and assign infinity path values to all other vertices
0
∞
∞
∞
∞
4
2
2
3
1
4
3
-5
5

Step-2 for Bellman Ford's algorithm

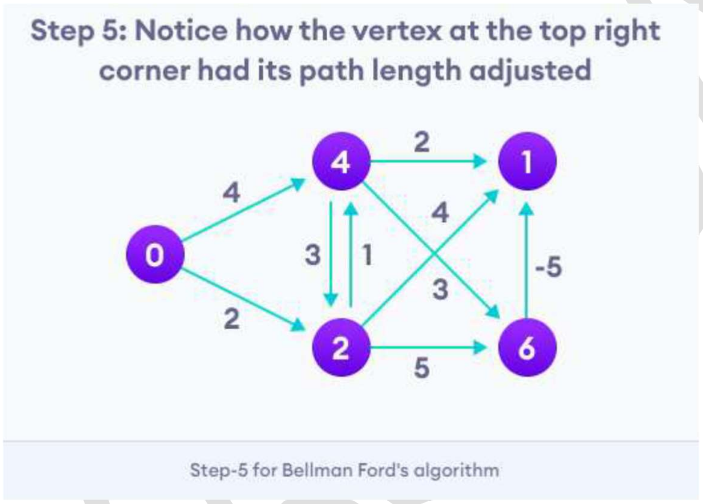

Step 3: Visit each edge and relax the path distances if they are inaccurate
0
4
2
∞
∞
4
2
2
3
1
4
3
-5
5

Step-3 for Bellman Ford's algorithm



Step 4: We need to do this V times because in the worst case, a vertex's path length might need to be readjusted V times

Step-4 for Bellman Ford's algorithm



Step 5: Notice how the vertex at the top right corner had its path length adjusted

Step-5 for Bellman Ford's algorithm