Session 10

Graph Data Structure

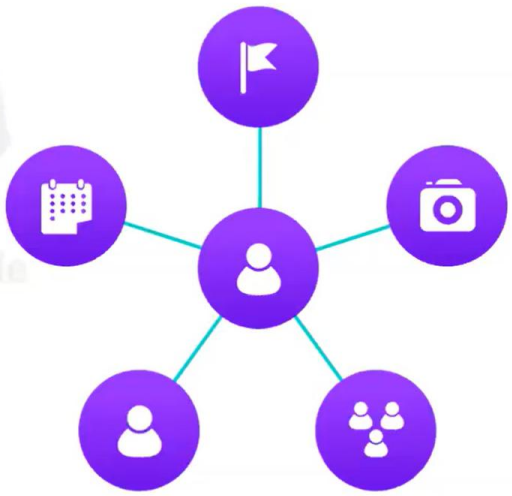
PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

## Graph Data Structure

A graph data structure is a collection of nodes that have data and are connected to other nodes.

Let's try to understand this through an example. On Facebook, everything is a node. That includes User, Photo, Album, Event, Group, Page, Comment, Story, Video, Link, Note...anything that has data is a node.

Every relationship is an edge from one node to another. Whether you post a photo, join a group, like a page, etc., a new edge is created for that relationship.





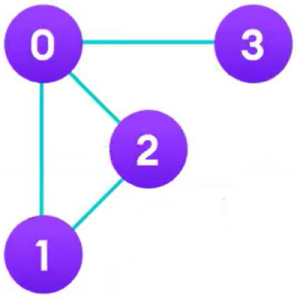


PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

## Graph Data Structure

More precisely, a graph is a data structure (V, E) that consists of

- A collection of vertices V
- A collection of edges E, represented as ordered pairs of vertices (u,v)




In the graph,

V = {0, 1, 2, 3}
E = {(0,1), (0,2), (0,3), (1,2)}
G = {V, E}

## Graph Data Structure

### Graph Terminology

**Adjacency:** A vertex is said to be adjacent to another vertex if there is an edge connecting them. Vertices 2 and 3 are not adjacent because there is no edge between them.

**Path:** A sequence of edges that allows you to go from vertex A to vertex B is called a path. 0-1, 1-2 and 0-2 are paths from vertex 0 to vertex 2.

**Directed Graph:** A graph in which an edge (u,v) doesn't necessarily mean that there is an edge (v, u) as well. The edges in such a graph are represented by arrows to show the direction of the edge.



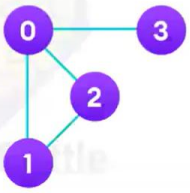




## Graph Data Structure

### Graph Representation

Graphs are commonly represented in two ways:




| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 2 | 1 | 1 | 0 | 0 |
| 3 | 1 | 0 | 0 | 0 |

### 1. Adjacency Matrix

An adjacency matrix is a 2D array of V x V vertices. Each row and column represent a vertex.

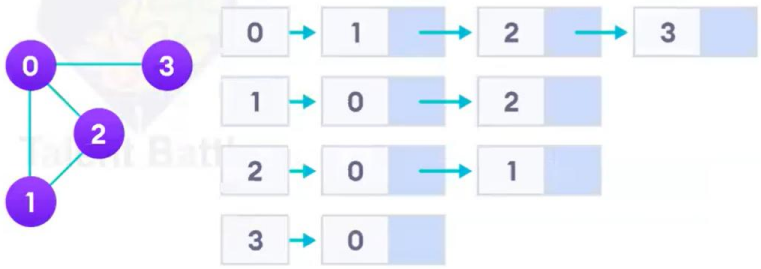
If the value of any element a[i][j] is 1, it represents that there is an edge connecting vertex i and vertex j.

## Graph Data Structure

**2. Adjacency List**

An adjacency list represents a graph as an array of linked lists.

The index of the array represents a vertex and each element in its linked list represents the other vertices that form an edge with the vertex.










## Graph Data Structure

**Graph Operations**

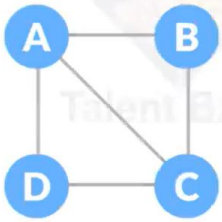
The most common graph operations are:

- Check if the element is present in the graph
- Graph Traversal
- Add elements(vertex, edges) to graph
- Finding the path from one vertex to another

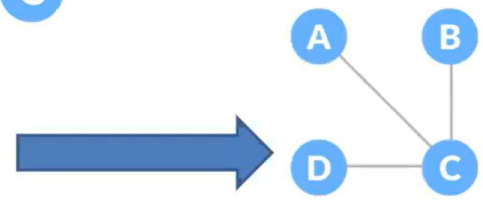
## Graph Data Structure — Spanning Tree and Minimum Spanning Tree

Before we learn about spanning trees, we need to understand two graphs: undirected graphs and connected graphs.

An **undirected graph** is a graph in which the edges do not point in any direction (ie. the edges are bidirectional).




A **connected graph** is a graph in which there is always a path from a vertex to any other vertex.










## Graph Data Structure

### Spanning tree

A spanning tree is a sub-graph of an undirected connected graph, which includes all the vertices of the graph with a minimum possible number of edges. If a vertex is missed, then it is not a spanning tree.

The edges may or may not have weights assigned to them.

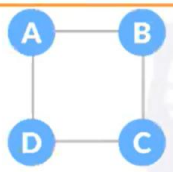
The total number of spanning trees with n vertices that can be created from a complete graph is equal to n^(n-2).

If we have n = 4, the maximum number of possible spanning trees is equal to 4^(4-2) = 16. Thus, 16 spanning trees can be formed from a complete graph with 4 vertices.

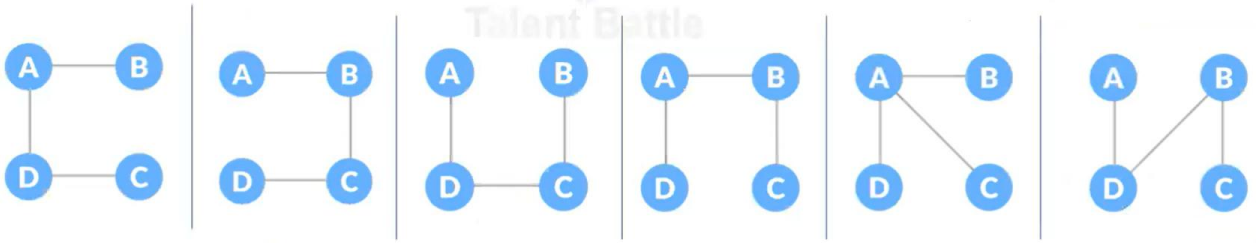
## Graph Data Structure

### Example of a Spanning Tree

Let the original graph be:




Some of the possible spanning trees that can be created from the above graph are:









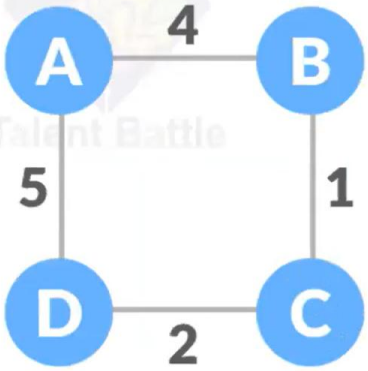

## Graph Data Structure

### Minimum Spanning Tree

A minimum spanning tree is a spanning tree in which the sum of the weight of the edges is as minimum as possible.

### Example of a Spanning Tree

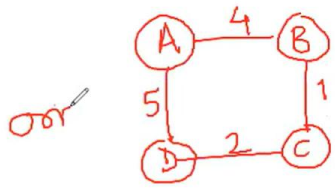
The initial graph is:



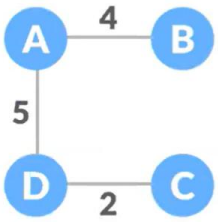

Weighted graph

## Graph Data Structure

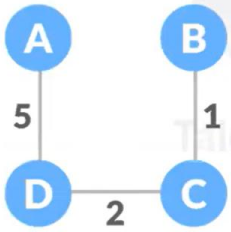
**The possible spanning trees from the graph are:**



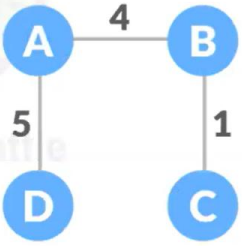

or



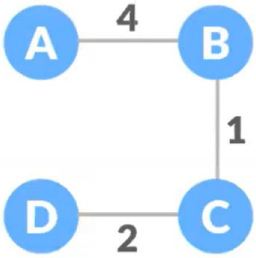

sum = 11




sum = 8




sum = 10




sum = 7







## Graph Data Structure

### Spanning Tree Applications

- Computer Network Routing Protocol
- Cluster Analysis
- Civil Network Planning

### Minimum Spanning tree Applications

- To find paths in the map
- To design networks like telecommunication networks, water supply networks, and electrical grids.

## Graph Data Structure

## Strongly Connected Components

- A strongly connected component is the portion of a directed graph in which there is a path from each vertex to another vertex.

- **It is applicable only on a directed graph.**

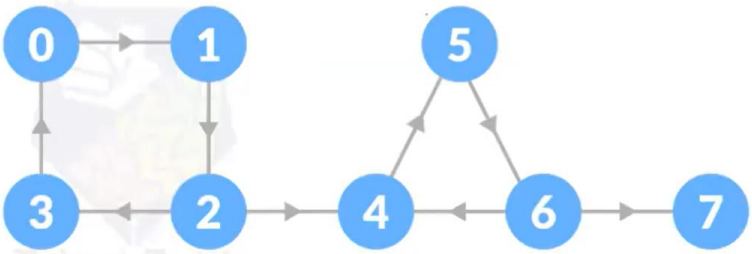




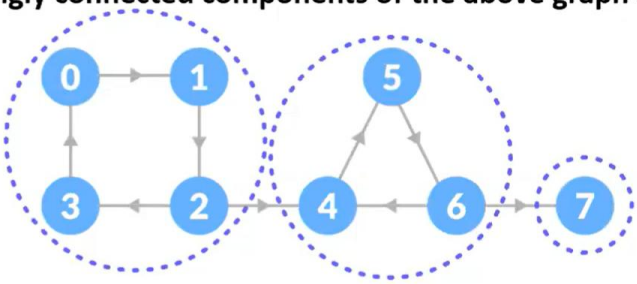


## Graph Data Structure

**Let us take the graph:**




**The strongly connected components of the above graph are:**



PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

## Graph Data Structure

### Strongly Connected Components Applications

- Vehicle routing applications
- Maps





PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

## Graph Data Structure

### Depth First Search (DFS)

Depth first Search or Depth first traversal is a recursive algorithm for searching all the vertices of a graph or tree data structure. Traversal means visiting all the nodes of a graph.

**Depth First Search Algorithm**

A standard DFS implementation puts each vertex of the graph into one of two categories:

- **Visited**
- **Not Visited**

The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.

**The DFS algorithm works as follows:**

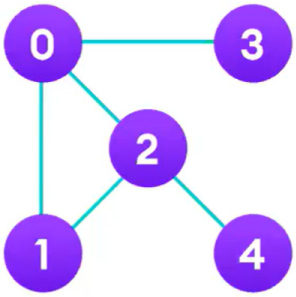
1. Start by putting any one of the graph's vertices on top of a stack.
2. Take the top item of the stack and add it to the visited list.
3. Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the top of the stack.
4. Keep repeating steps 2 and 3 until the stack is empty.

PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

## Graph Data Structure

## Depth First Search Example

Let's see how the Depth First Search algorithm works with an example. We use an undirected graph with 5 vertices.




| | | | | |
|---|---|---|---|---|
| | | | | |

Visited

| | | | | |
|---|---|---|---|---|
| | | | | |

Stack

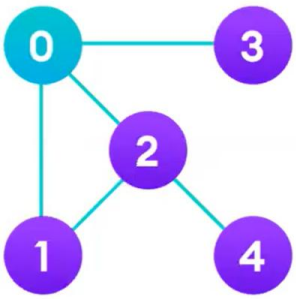




PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

## Graph Data Structure

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.




| | | | | |
|---|---|---|---|---|
| 0 | | | | |

Visited

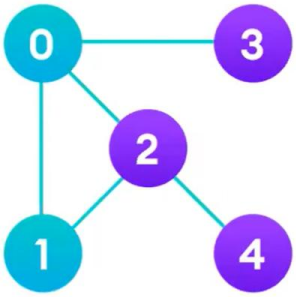
| | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | | |

Stack

PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

## Graph Data Structure

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.




| 0 | 1 | | | | Visited |
|---|---|---|---|---|---|
| 2 | 3 | | | | Stack |

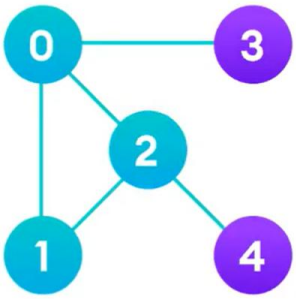




PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

## Graph Data Structure

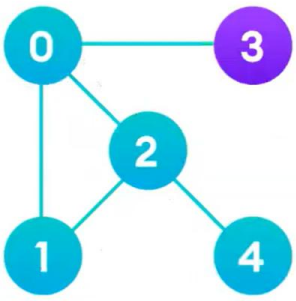
Vertex 2 has an unvisited adjacent vertex in 4, so we add that to the top of the stack and visit it.




| 0 | 1 | 2 | | | Visited |
|---|---|---|---|---|---|
| 4 | 3 | | | | Stack |

PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

## Graph Data Structure




| 0 | 1 | 2 | 4 | | Visited |
|---|---|---|---|---|---|

| 3 | | | | | Stack |
|---|---|---|---|---|---|

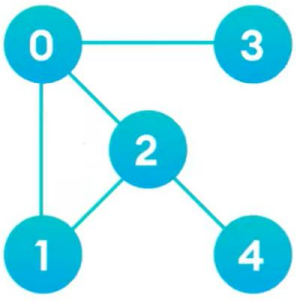




PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

## Graph Data Structure

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.




| 0 | 1 | 2 | 4 | 3 | Visited |
|---|---|---|---|---|---|

| | | | | | Stack |
|---|---|---|---|---|---|

## Graph Data Structure

### Complexity of Depth First Search

The **time complexity** of the DFS algorithm is represented in the form of **O(V + E)**, where V is the number of nodes and E is the number of edges.

The **space complexity** of the algorithm is **O(V).**

### Application of DFS Algorithm

1. For finding the path
2. To test if the graph is bipartite
3. For finding the strongly connected components of a graph
4. For detecting cycles in a graph







## Graph Data Structure

### Breadth first search

Traversal means visiting all the nodes of a graph. Breadth First Traversal or Breadth First Search is a recursive algorithm for searching all the vertices of a graph or tree data structure.

PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

## Graph Data Structure

### BFS algorithm

A standard BFS implementation puts each vertex of the graph into one of two categories:
1.Visited
2.Not Visited

The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.
The algorithm works as follows:
1.Start by putting any one of the graph's vertices at the back of a queue.
2.Take the front item of the queue and add it to the visited list.
3.Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the back of the queue.
4.Keep repeating steps 2 and 3 until the queue is empty.

The graph might have two different disconnected parts so to make sure that we cover every vertex, we can also run the BFS algorithm on every node



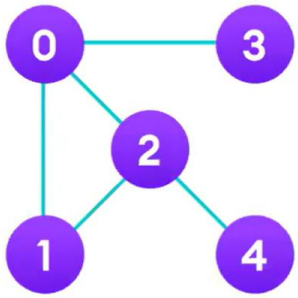


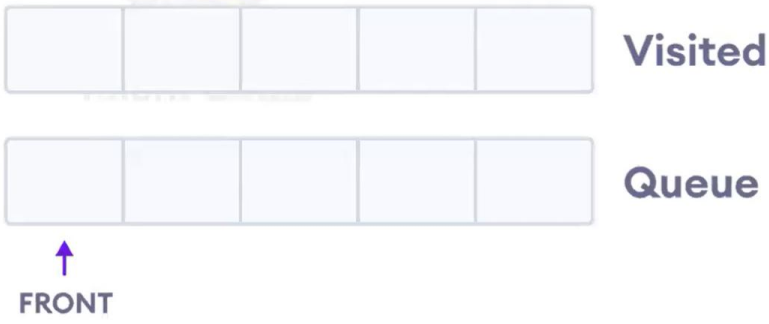
PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

## Graph Data Structure

### BFS example

Let's see how the Breadth First Search algorithm works with an example. We use an undirected graph with 5 vertices.

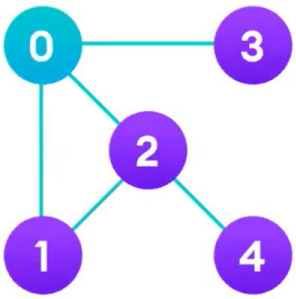


PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

## Graph Data Structure

We start from vertex 0, the BFS algorithm starts by putting it in the Visited list and putting all its adjacent vertices in the stack.




| 0 | | | | |
|---|---|---|---|---|

Visited

| 1 | 2 | 3 | | |
|---|---|---|---|---|

Queue

FRONT

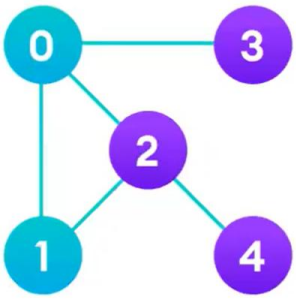




PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

## Graph Data Structure

Next, we visit the element at the front of queue i.e. 1 and go to its adjacent nodes. Since 0 has already been visited, we visit 2 instead.




| 0 | 1 | | | |
|---|---|---|---|---|

Visited

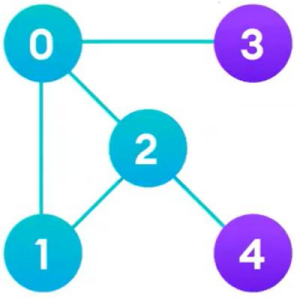
| 2 | 3 | | | |
|---|---|---|---|---|

Queue

FRONT

## Graph Data Structure

Vertex 2 has an unvisited adjacent vertex in 4, so we add that to the back of the queue and visit 3, which is at the front of the queue.




| 0 | 1 | 2 | | | Visited |
|---|---|---|---|---|---|
| 3 | 4 | | | | Queue |

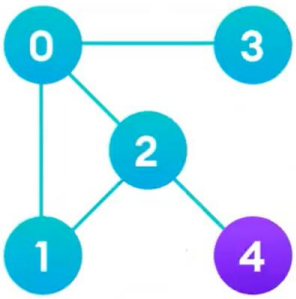
FRONT







## Graph Data Structure



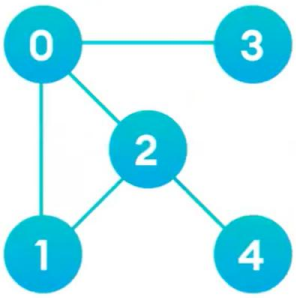

| 0 | 1 | 2 | 3 | | Visited |
|---|---|---|---|---|---|
| 4 | | | | | Queue |

FRONT

PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

## Graph Data Structure

Only 4 remains in the queue since the only adjacent node of 3 i.e. 0 is already visited. We visit it.




| 0 | 1 | 2 | 3 | 4 | Visited |
|---|---|---|---|---|---|

| | | | | | Queue |
|---|---|---|---|---|---|

FRONT





PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

## Graph Data Structure

**BFS Algorithm Complexity**

The **time complexity** of the BFS algorithm is represented in the form of **O(V + E)**, where V is the number of nodes and E is the number of edges.

The **space complexity** of the algorithm is **O(V).**

**BFS Algorithm Applications:**

- To build index by search index
- For GPS navigation
- Path finding algorithms
- In Ford-Fulkerson algorithm to find maximum flow in a network
- Cycle detection in an undirected graph
- In minimum spanning tree

## Graph Data Structure

## Bellman Ford's Algorithm

Bellman Ford algorithm helps us find the shortest path from a vertex to all other vertices of a weighted graph.

It is similar to Dijkstra's algorithm but it can work with graphs in which edges can have negative weights.







## Graph Data Structure

### How Bellman Ford's algorithm works

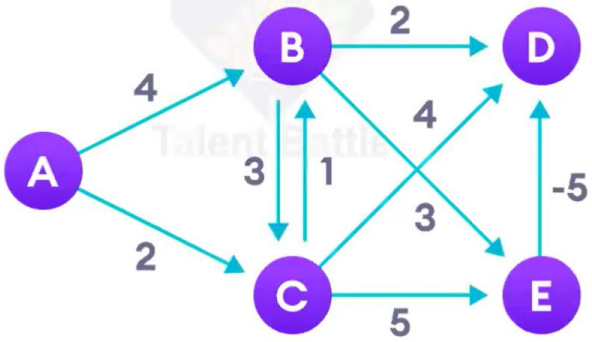
Bellman Ford algorithm works by overestimating the length of the path from the starting vertex to all other vertices.

Then it iteratively relaxes those estimates by finding new paths that are shorter than the previously overestimated paths.

PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

## Graph Data Structure

### Step 1: Start with the weighted graph



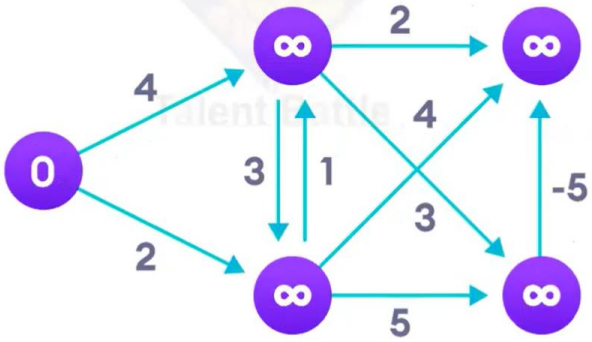





PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

## Graph Data Structure

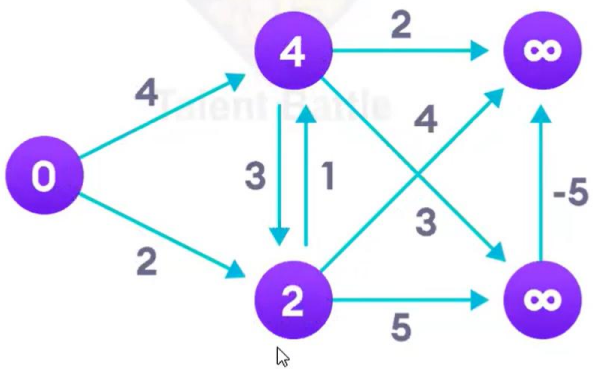
### Step 2: Choose a starting vertex and assign infinity path values to all other vertices



PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

## Graph Data Structure

### Step 3: Visit each edge and relax the path distances if they are inaccurate



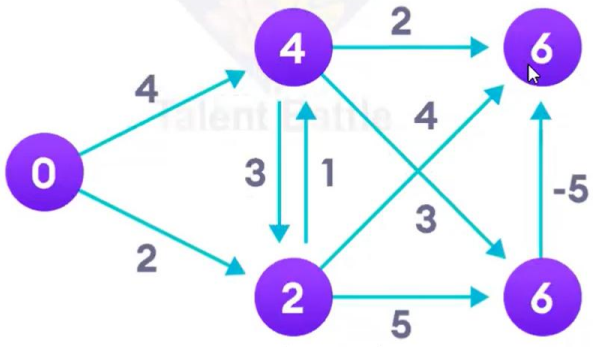





PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

## Graph Data Structure

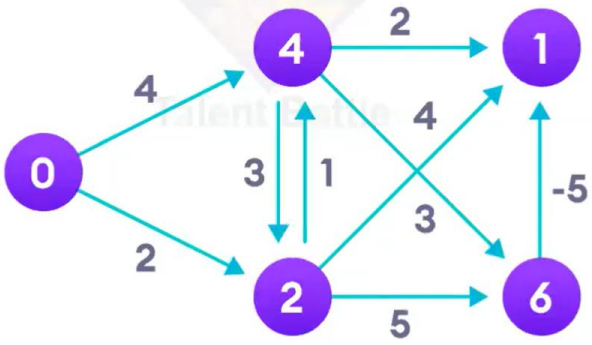
### Step 4: We need to do this V times because in the worst case, a vertex's path length might need to be readjusted V times



PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

## Graph Data Structure

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








PowerPoint Slide Show - [DSA Technical Training] - PowerPoint

## Graph Data Structure

### Step 6: After all the vertices have their path lengths, we check if a negative cycle is present

| | B | C | D | E |
|---|---|---|---|---|
| 0 | ∞ | ∞ | ∞ | ∞ |
| 0 | 4 | 2 | ∞ | ∞ |
| 0 | 3 | 2 | 6 | 6 |
| 0 | 3 | 2 | 1 | 6 |
| 0 | 3 | 2 | 1 | 6 |