# BFS, DFS, Floyd Warshall

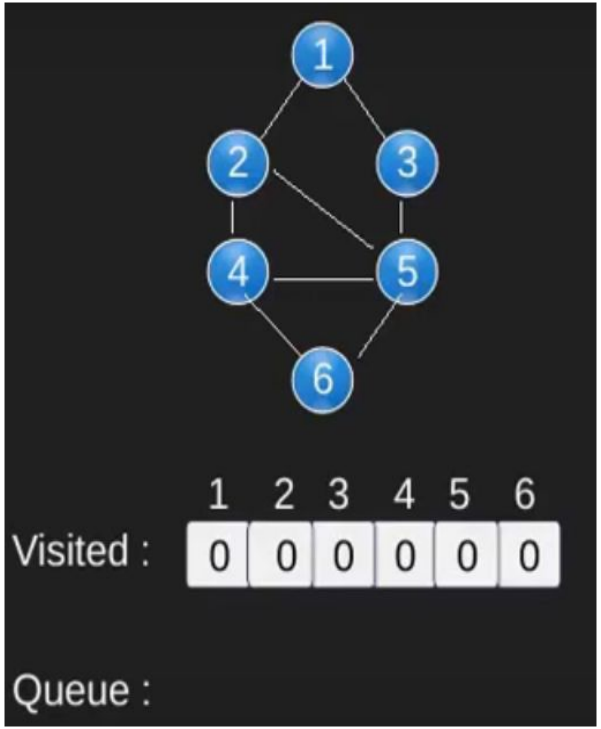
## Breath First Search

BFS is a graph traversing algorithm that uses queue data structure.

Steps for traversing :

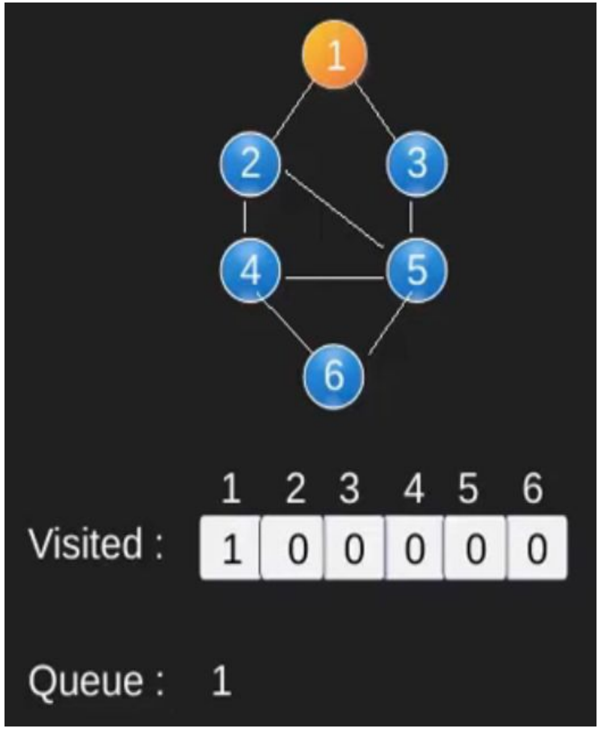
1. Start with a root node
2. Mark the node as visited
3. Insert the root node in the queue
4. Dequeue the front node from the queue and enqueue the adjacent nodes of the front node

Repeat enquing and dequing the nodes from the queue and make them visited. Each node is visited just once. So the visitited array is needed.



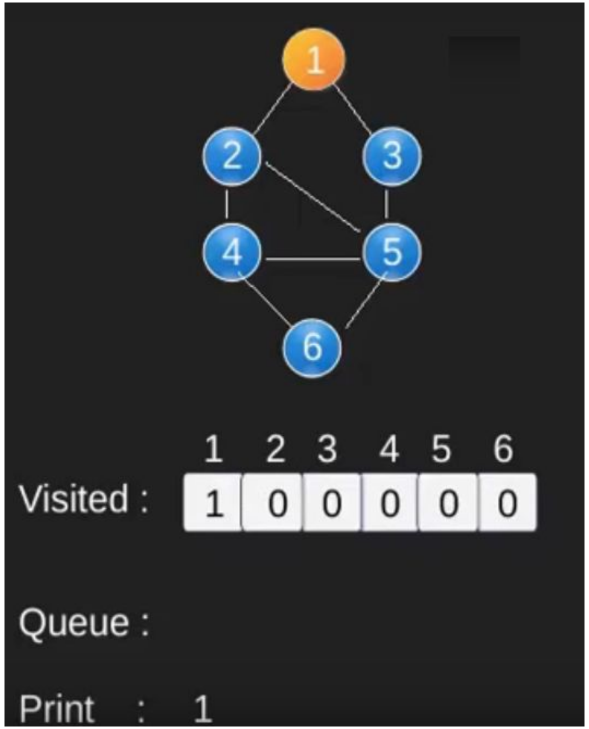
| Visited : | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | 0 |

Queue :



| Visited : | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| | 1 | 0 | 0 | 0 | 0 | 0 |

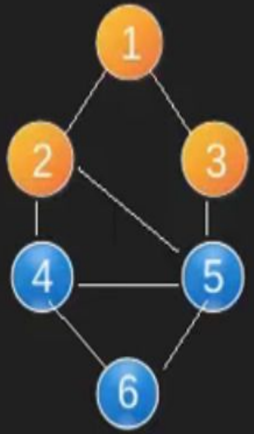
Queue : 1



| Visited : | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| | 1 | 0 | 0 | 0 | 0 | 0 |

Queue :

Print : 1

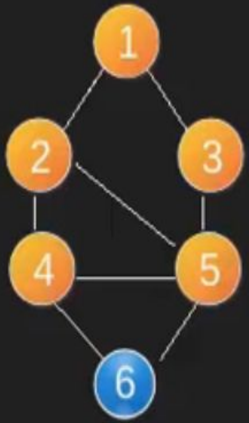


| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Visited : | 1 | 1 | 1 | 0 | 0 | 0 |

Queue : 2 3

After removing 1 from queue and printing it, we enqueue its non-visited adjacent Nodes

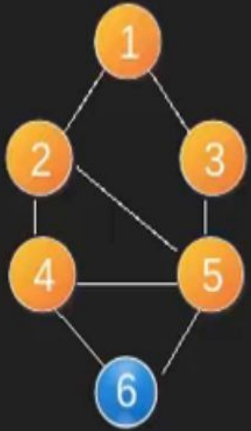
Print : 1



| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Visited : | 1 | 1 | 1 | 1 | 1 | 0 |

Queue : 3 4 5

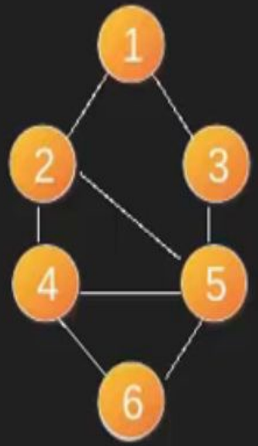
Print : 1 2



| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Visited : | 1 | 1 | 1 | 1 | 1 | 0 |

Queue : 5

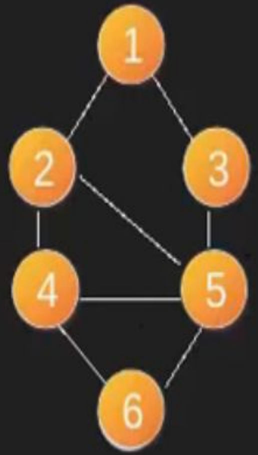
Print : 1 2 3 4



| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Visited : | 1 | 1 | 1 | 1 | 1 | 1 |

Queue : 5 6

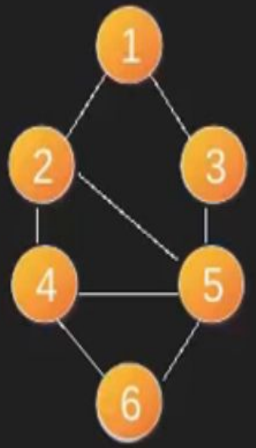
Print : 1 2 3 4



| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Visited : | 1 | 1 | 1 | 1 | 1 | 1 |

Queue : 6

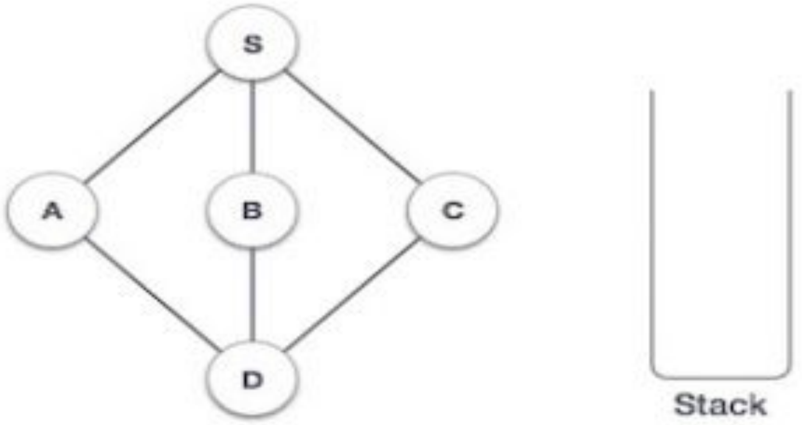
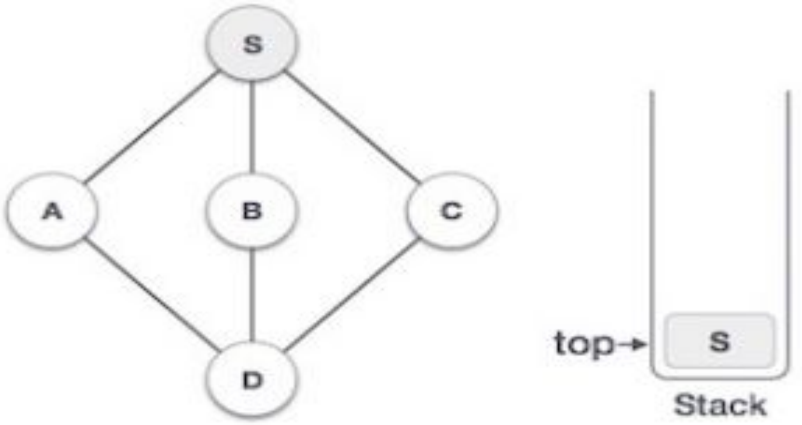
Print : 1 2 3 4 5



| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Visited : | 1 | 1 | 1 | 1 | 1 | 1 |

Queue :

Print : 1 2 3 4 5 6

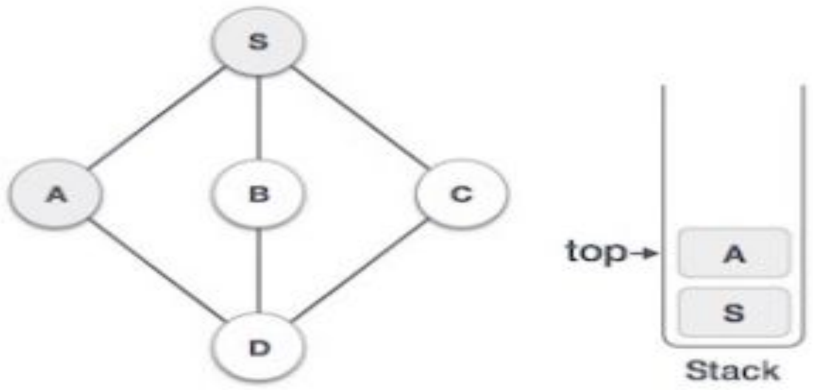
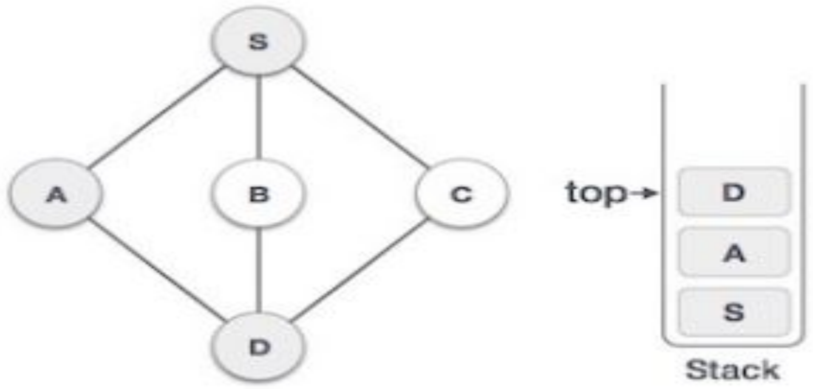
| | | |
|---|---|---|
| 1 |  | Initialize the stack. |
| 2 |  | Mark **S** as visited and put it onto the stack. Explore any unvisited adjacent node from **S**. We have three nodes and we can pick any of them. For this example, we shall take the node in an alphabetical order. |

## Depth First Search

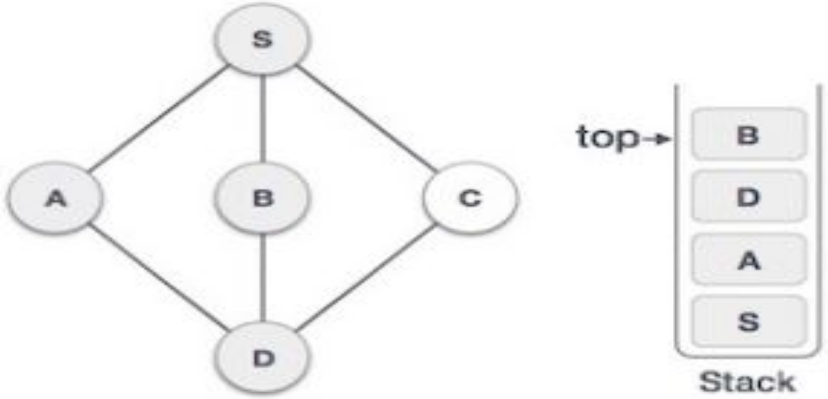
DFS is a graph traversing algorithm that uses stack data structure.

Rules for traversing :

- Rule 1 − Visit the adjacent unvisited vertex. Mark it as visited. Display it. Push it in a stack.
- Rule 2 − If no adjacent vertex is found, pop up a vertex from the stack. (It will pop up all the vertices from the stack, which do not have adjacent vertices.)
- Rule 3 − Repeat Rule 1 and Rule 2 until the stack is empty.

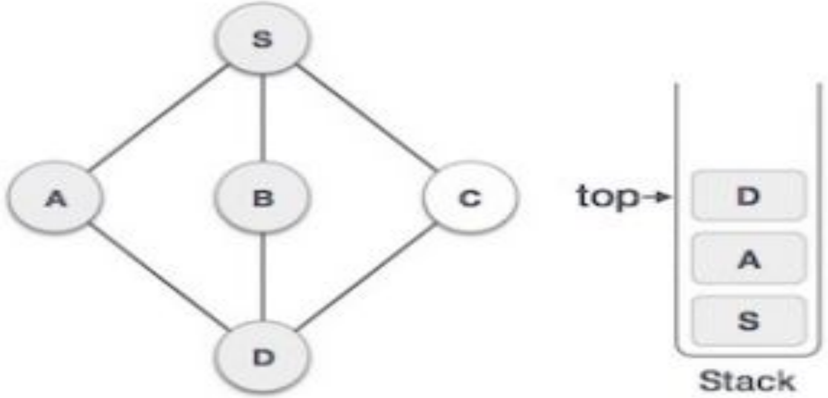
| | | |
|---|---|---|
| 3 |  | Mark **A** as visited and put it onto the stack. Explore any unvisited adjacent node from A. Both **S** and **D** are adjacent to **A** but we are concerned for unvisited nodes only. |
| 4 |  | Visit **D** and mark it as visited and put onto the stack. Here, we have **B** and **C** nodes, which are adjacent to **D** and both are unvisited. However, we shall again choose in an alphabetical order. |

5



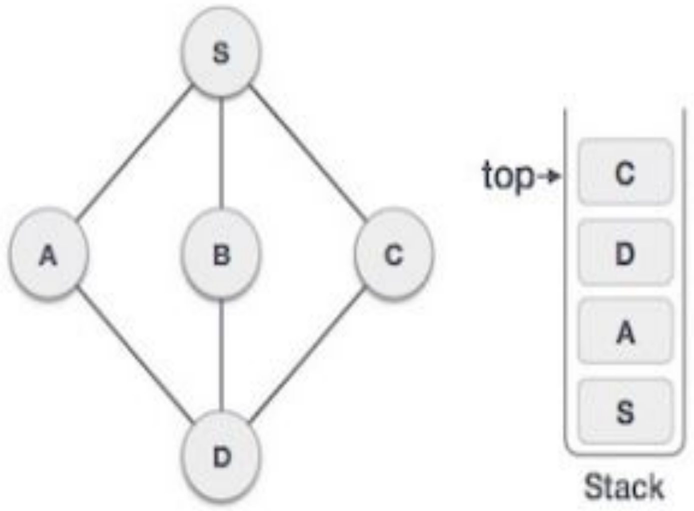
We choose **B**, mark it as visited and put onto the stack. Here **B** does not have any unvisited adjacent node. So, we pop **B** from the stack.

6



We check the stack top for return to the previous node and check if it has any unvisited nodes. Here, we find **D** to be on the top of the stack.

7

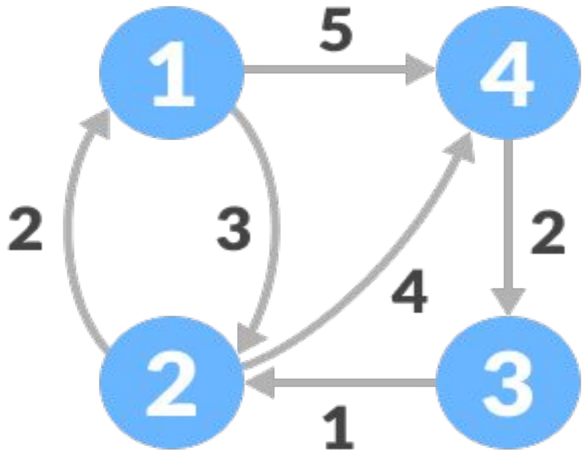


Only unvisited adjacent node is from **D** is **C** now. So we visit **C**, mark it as visited and put it onto the stack.

As C does not have any unvisited adjacent node so we keep popping the stack until we find a node that has an unvisited adjacent node. In this case, there's none and we keep popping until the stack is empty.

## Floyd Warshall

Floyd-Warshall Algorithm is an algorithm for finding the shortest path between all the pairs of vertices in a weighted graph. 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).



Create a matrix $A^1$ of dimension `n*n` where n is the number of vertices. The row and the column are indexed as `i` and `j` respectively. `i` and `j` are the vertices of the graph. Each cell A[i][j] is filled with the distance from the ith vertex to the jth vertex. If there is no path from ith vertex to jth vertex, the cell is left as infinity.

$$A^0 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & \infty & 5 \\ 2 & 2 & 0 & \infty & 4 \\ 3 & \infty & 1 & 0 & \infty \\ 4 & \infty & \infty & 2 & 0 \end{array}$$

matrix `A1` is derived from matrix `A0`. The elements in the first column and the first row are left as they are. The remaining cells are filled in the following way. Let `k` be the intermediate vertex in the shortest path from source to destination. In this step, `k` is the first vertex. `A[i][j]` is filled with `(A[i][k] + A[k][j]) if (A[i][j] > A[i][k] + A[k][j])`. That is, if the direct distance from the source to the destination is greater than the path through the vertex `k`, then the cell is filled with `A[i][k] + A[k][j]`. In this step, k is vertex 1. We calculate the distance from source vertex to destination vertex through this vertex k. For example: For `A1[2, 4]`, the direct distance from vertex 2 to 4 is 4 and the sum of the distance from vertex 2 to 4 through vertex (ie. from vertex 2 to 1 and from vertex 1 to 4) is 7. Since `4 < 7`, `A0[2, 4]` is filled with 4.

$$A^1 = \begin{array}{c|cccc|} & 1 & 2 & 3 & 4 \\ 1 & 0 & 3 & \infty & 5 \\ 2 & 2 & 0 & & \\ 3 & \infty & & 0 & \\ 4 & \infty & & & 0 \end{array} \longrightarrow \begin{array}{c|cccc|} & 1 & 2 & 3 & 4 \\ 1 & 0 & 3 & \infty & 5 \\ 2 & 2 & 0 & 9 & 4 \\ 3 & \infty & 1 & 0 & 8 \\ 4 & \infty & \infty & 2 & 0 \end{array}$$

Calculate the distance from the source vertex to destination vertex through this vertex k

Similarly, `A2` is created using `A3`. The elements in the second column and the second row are left as they are. In this step, `k` is the second vertex (i.e. vertex 2). The remaining steps are the same as in step 2.

$$A^2 = \begin{array}{c|cccc|} & 1 & 2 & 3 & 4 \\ 1 & 0 & 3 & & \\ 2 & 2 & 0 & 9 & 4 \\ 3 & & 1 & 0 & \\ 4 & & \infty & & 0 \end{array} \longrightarrow \begin{array}{c|cccc|} & 1 & 2 & 3 & 4 \\ 1 & 0 & 3 & 9 & 5 \\ 2 & 2 & 0 & 9 & 4 \\ 3 & 3 & 1 & 0 & 5 \\ 4 & \infty & \infty & 2 & 0 \end{array}$$

Calculate the distance from the source vertex to destination vertex through this vertex 2

4. Similarly, `A3` and `A4` is also created.

$$A^3 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & & \infty & \\ 2 & & 0 & 9 & \\ 3 & \infty & 1 & 0 & 8 \\ 4 & & & 2 & 0 \end{array} \longrightarrow \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & 9 & 5 \\ 2 & 2 & 0 & 9 & 4 \\ 3 & 3 & 1 & 0 & 5 \\ 4 & 5 & 3 & 2 & 0 \end{array}$$

Calculate the distance from the source vertex to destination vertex through this vertex 3

$$A^4 = \begin{array}{ccccc} & 1 & 2 & 3 & 4 \\ 1 & 0 & & & 5 \\ 2 & & 0 & & 4 \\ 3 & & & 0 & 5 \\ 4 & 5 & 3 & 2 & 0 \end{array} \longrightarrow \begin{array}{ccccc} & 1 & 2 & 3 & 4 \\ 1 & 0 & 3 & 7 & 5 \\ 2 & 2 & 0 & 6 & 4 \\ 3 & 3 & 1 & 0 & 5 \\ 4 & 5 & 3 & 2 & 0 \end{array}$$

Calculate the distance from the source vertex to destination vertex through this vertex 4.