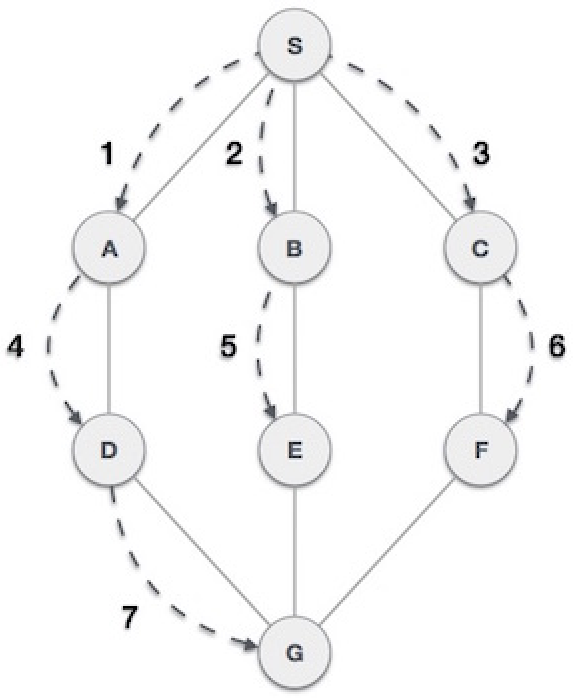
Breadth First Search (BFS) Algorithm

Breadth First Search (BFS) Algorithm

Breadth First Search (BFS) algorithm traverses a graph in a breadthward motion to search a graph data structure for a node that meets a set of criteria. It uses a queue to remember the next vertex to start a search, when a dead end occurs in any iteration.

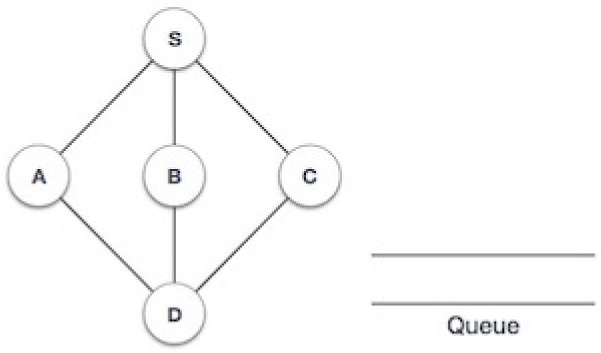
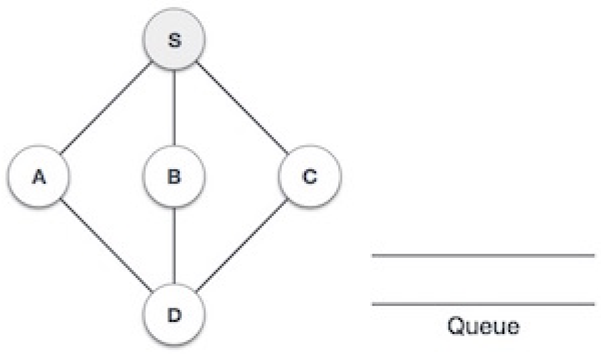
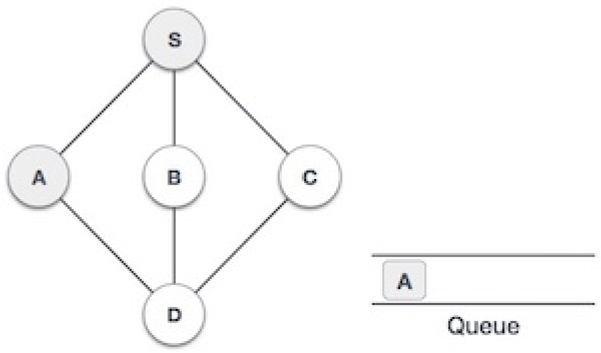
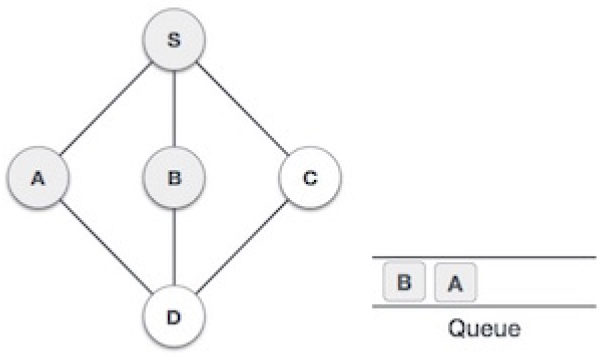
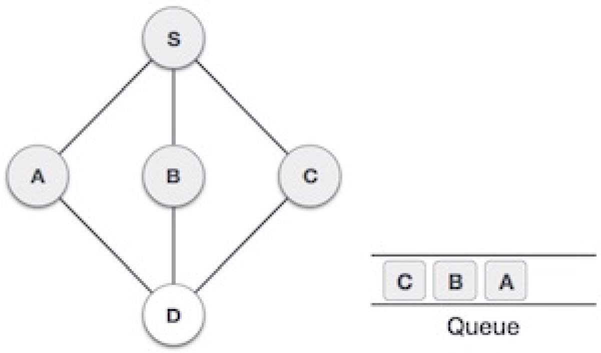
Breadth First Search (BFS) algorithm starts at the tree root and explores all nodes at the present depth prior to moving on to the nodes at the next depth level.

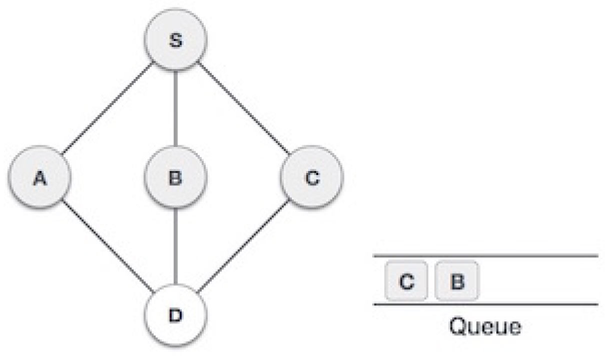
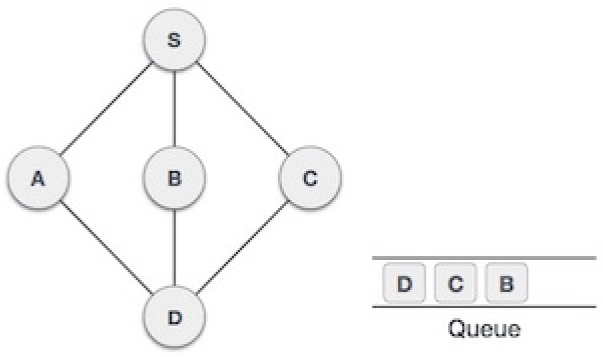


As in the example given above, BFS algorithm traverses from A to B to E to F first then to C and G lastly to D. It employs the following rules.

- **Rule 1** – Visit the adjacent unvisited vertex. Mark it as visited. Display it. Insert it in a queue.
- **Rule 2** – If no adjacent vertex is found, remove the first vertex from the queue.
- **Rule 3** – Repeat Rule 1 and Rule 2 until the queue is empty.

Step	Traversal	Description

1		Initialize the queue.
2		We start from visiting **S** (starting node), and mark it as visited.
3		We then see an unvisited adjacent node from **S**. In this example, we have three nodes but alphabetically we choose **A**, mark it as visited and enqueue it.
4		Next, the unvisited adjacent node from **S** is **B**. We mark it as visited and enqueue it.
5		Next, the unvisited adjacent node from **S** is **C**. We mark it as visited and enqueue it.

6		Now, **S** is left with no unvisited adjacent nodes. So, we dequeue and find **A**.
7		From **A** we have **D** as unvisited adjacent node. We mark it as visited and enqueue it.

At this stage, we are left with no unmarked (unvisited) nodes. But as per the algorithm we keep on dequeuing in order to get all unvisited nodes. When the queue gets emptied, the program is over.

Example

Following are the implementations of Breadth First Search (BFS) Algorithm in various programming languages −

C | C++ | Java | Python

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define MAX 5
struct Vertex {
   char label;
   bool visited;
};
//queue variables
int queue[MAX];
int rear = -1;
int front = 0;
int queueItemCount = 0;
//graph variables
```

```
//array of vertices
struct Vertex* lstVertices[MAX];
//adjacency matrix
int adjMatrix[MAX][MAX];
//vertex count
int vertexCount = 0;
//queue functions
void insert(int data) {
   queue[++rear] = data;
   queueItemCount++;
}
int removeData() {
   queueItemCount--;
   return queue[front++];
}
bool isQueueEmpty() {
   return queueItemCount == 0;
}
//graph functions
//add vertex to the vertex list
void addVertex(char label) {
   struct Vertex* vertex = (struct Vertex*) malloc(sizeof(struct Vertex))
   vertex->label = label;
   vertex->visited = false;
   lstVertices[vertexCount++] = vertex;
}
//add edge to edge array
void addEdge(int start,int end) {
   adjMatrix[start][end] = 1;
   adjMatrix[end][start] = 1;
}
//display the vertex
void displayVertex(int vertexIndex) {
   printf("%c ",lstVertices[vertexIndex]->label);
}
//get the adjacent unvisited vertex
int getAdjUnvisitedVertex(int vertexIndex) {
   int i;

   for(i = 0; i<vertexCount; i++) {
      if(adjMatrix[vertexIndex][i] == 1 && lstVertices[i]->visited == fal
         return i;
   }
   return -1;
}
void breadthFirstSearch() {
   int i;
   //mark first node as visited
   lstVertices[0]->visited = true;
   //display the vertex
   displayVertex(0);
   //insert vertex index in queue
   insert(0);
   int unvisitedVertex;
   while(!isQueueEmpty()) {
      //get the unvisited vertex of vertex which is at front of the queue
      int tempVertex = removeData();
      //no adjacent vertex found
      while((unvisitedVertex = getAdjUnvisitedVertex(tempVertex)) != -1)
         lstVertices[unvisitedVertex]->visited = true;
         displayVertex(unvisitedVertex);
         insert(unvisitedVertex);
      }
   }
   //queue is empty, search is complete, reset the visited flag
   for(i = 0;i<vertexCount;i++) {
      lstVertices[i]->visited = false;
   }
}
int main() {
   int i, j;

   for(i = 0; i<MAX; i++) { // set adjacency
      for(j = 0; j<MAX; j++) // matrix to 0
         adjMatrix[i][j] = 0;
   }
   addVertex('S');   // 0
   addVertex('A');   // 1
   addVertex('B');   // 2
   addVertex('C');   // 3
   addVertex('D');   // 4
   addEdge(0, 1);    // S - A
   addEdge(0, 2);    // S - B
   addEdge(0, 3);    // S - C
```

```
    addEdge(1, 4);    // A - D
    addEdge(2, 4);    // B - D
    addEdge(3, 4);    // C - D
    printf("\nBreadth First Search: ");
    breadthFirstSearch();
    return 0;
}
```

Output

```
Breadth First Search: S A B C D
```

Click to check C implementation of Breadth First Search (BFS) Algorithm

Complexity of BFS Algorithm

Time 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.

Space Complexity

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