PES UNIVERSITY

Department of Computer science and Engineering

PES UNIVERSITY

UE19CS202: Data Structures and its Applications (4-0-0-4-4)

Applications of BFS and DFS

Abstract

Applications of BFS, Applications of DFS, Implementation of Connectivity of graph in directed and undirected graphs.

Dr.Sandesh and Saritha

Sandesh_bj@pes.edu

Saritha.k@pes.edu

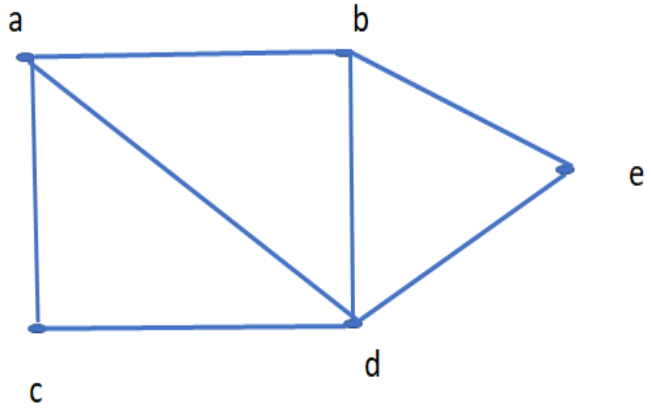
Applications of BFS and DFS:

Connectivity of graph

A graph is said to be connected if there is a path between every pair of vertex.. A graph with multiple disconnected vertices and edges is said to be disconnected. To check connectivity of a graph, we traverse all nodes using graph traversal algorithm like BFS and DFS. On completion of traversal, if there is any node, which is not visited, then the graph is disconnected.

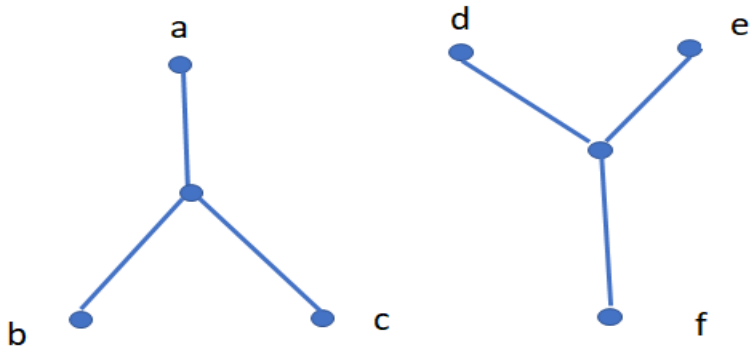
Connected graph:

Below graph is an example for connected graph since it is possible to traverse from one vertex to any other vertex. For example, one can traverse from vertex c to vertex d using the path c-d-e. Similarly we can traverse from vertex a to e using the path a-c-d-e



Disconnected

The graph shown below is an example for disconnected graph because there exist no path from b to e.



- To check whether a graph is connected or not, we traverse the graph using either bfs traversal or dfs traversal method
- After the traversal if there is at-least one node which is not marked as visited then that graph is disconnected graph

Procedure to check the whether a graph is connected or not using adjacency matrix

- Read the adjacency matrix .
- Create a visited [] array. Start DFS/BFS traversal method from any arbitrary vertex and mark the visited vertices in the visited [] array.
- Once DFS/BFS is completed check the visited [] array. If there is at-least one vertex which is marked as unvisited then the graph is disconnected otherwise it is connected.

Connectivity of undirected graph using DFS

We choose one vertex as an arbitrary node and traverse from that node.

Algorithm:

connected (G)

Input – undirected graph.

Output – Returns True if the graph is connected otherwise False.

```
Begin
  define visited array
    for all nodes u in the G, do
      mark all nodes unvisited
      traversal (u, visited)
      if unvisited , then
```

```
        return false
    done
  return true
End
```

traversal(u, visited)

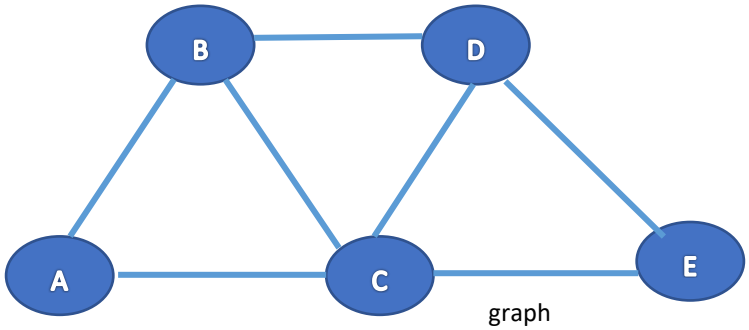
Input – Any arbitrary node u as the start node and the visited node to mark which node is visited

Output: Traverse all connected vertices.

```
Begin
  mark u as visited
  for all nodes v, if it is adjacent with u, do
    if v is not visited, then
      traversal(v, visited)
  done
End
```

Output: For the below graph is connected

To check whether the below graph is connected we give the adjacency matrix for the below graph as input.



graph

	A	B	C	D	E
A	0	1	1	0	0
B	1	0	1	1	0
C	1	1	0	1	1
D	0	1	1	0	1
E	0	0	1	1	0

Connectivity of directed graph using DFS

We choose one vertex as an arbitrary node and traverse from that node.

Algorithm:

connected (G)

Input – directed graph.

Output – Returns True if the graph is connected otherwise False.

Begin

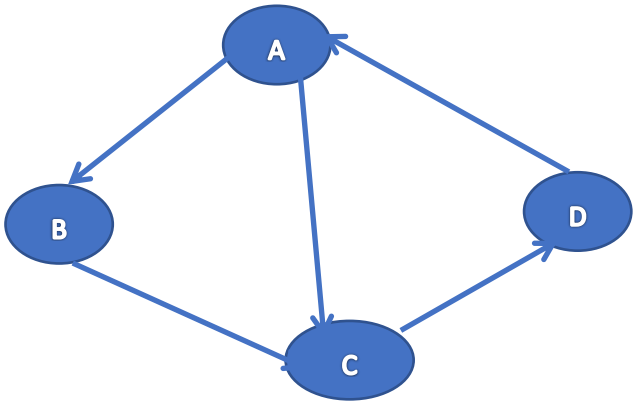
```
  define visited array
    for all nodes u in the G, do
      mark all nodes unvisited
      traversal (u, visited)
      if unvisited , then
        return false
    done
  return true
End
```

traversal(u, visited)

Input – Any arbitrary node u as the start node and the visited node to mark which node is visited

Output: Traverse all connected vertices.

```
Begin
  mark u as visited
  for all nodes v, if it is adjacent with u, do
    if v is not visited, then
      traversal(v, visited)
  done
End
```



graph

Adjacency matrix

	A	**B**	**C**	**D**
A	0	1	1	0
B	0	0	1	0
C	0	0	0	1
D	1	0	0	0

Output for the above directed graph is connected graph

Connectivity of directed graph using BFS

Algorithm:

traversal(x, visited)

Input: the arbitrary node x as start node

Output: Traverse all connected vertices.

Begin

```
  make x as visited
  insert x into a queue Que
  until the Que is not empty, do
  u = node taken out from queue
  for each vertex v of the graph G, do
    if the u and v are connected, then
      if u is not visited, then
        make u as visited
      insert u into the queue Que.
    done
  done
End
```

connected(G)

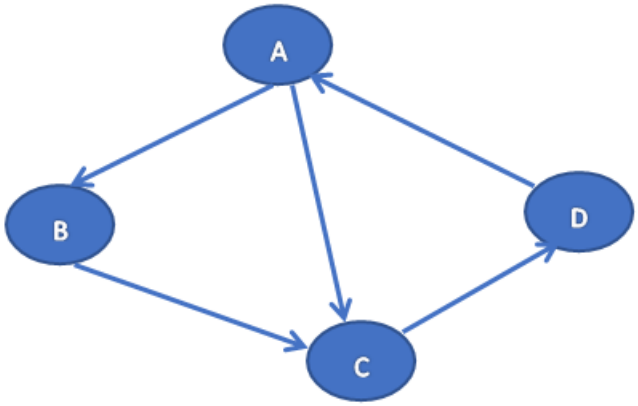
Input – The directed graph.

Output – True if the graph is connected otherwise returns False.

```
Begin
  define visited array
  for all nodes u in the G, do
    mark all nodes as unvisited
  traversal(u, visited)
  if  unvisited , then
    return false
  done
  return true
```

End

We input the adjacency matrix for the below graph as and the The output for the above directed graph is connected.



	A	B	C	D
A	0	1	1	0
B	0	0	1	0
C	0	0	0	1
D	1	0	0	0

Connectivity of undirected graph using BFS

Algorithm:

traversal(x, visited)

Input: The arbitrary node x as start node and the visited node to mark which node is visited.

Output: Traverse all connected vertices.

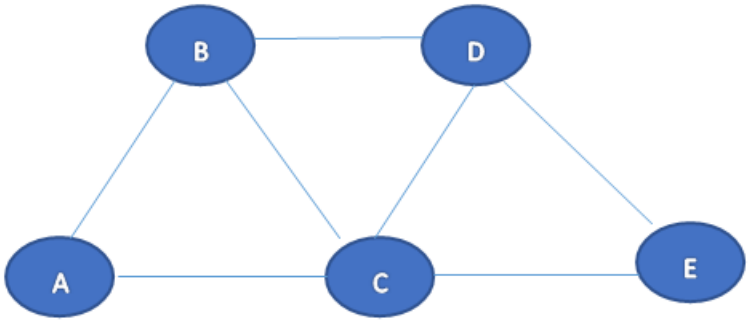
```
Begin
  make x as visited
  insert x into a queue Que
  until the Que is not empty, do
  u = node that is taken out from the queue
  for each vertex v of the graph G, do
    if the u and v are connected, then
      if u is not visited, then
        make u as visited
      insert u into the queue Que.
    done
  done
End
```

Connected(G)

Input – The directed graph.

Output – True if the graph is connected otherwise returns False.

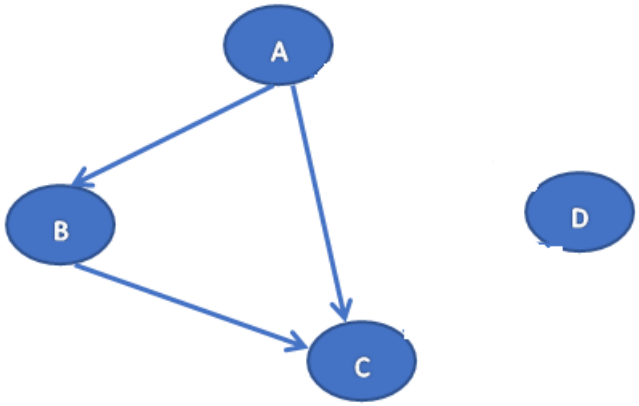
```
Begin
  define visited array
  for all nodes u in the G, do
    mark all nodes as unvisited
  traversal(u, visited)
  if  unvisited , then
    return false
  done
  return true
End
```



graph

	A	B	C	D	E
A	0	1	1	0	0
B	1	0	1	1	0
C	1	1	0	1	1
D	0	1	1	0	1
E	0	0	1	1	0

The output for the below undirected graph using BFS is connected.



graph

	A	B	C	D
A	0	1	1	0
B	0	0	1	0
C	0	0	0	0
D	0	0	0	0

The output for the above graph is disconnected.