DATA STRUCTURES AND ITS APPLICATIONS

Graphs

Saritha

Department of Computer Science & Engineering

DATA STRUCTURES AND ITS APPLICATIONS

Connectivity of the graph

Saritha

Department of Computer Science & Engineering

Applications of BFS

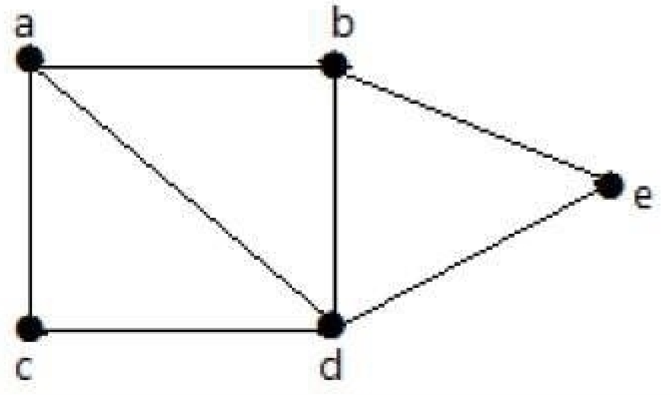
Application of BFS

- Finding the shortest path
- Social Networking websites like twitter, Facebook etc.
- GPS Navigation system
- Web crawlers
- Finding a path in network
- In Networking to broadcast the packets

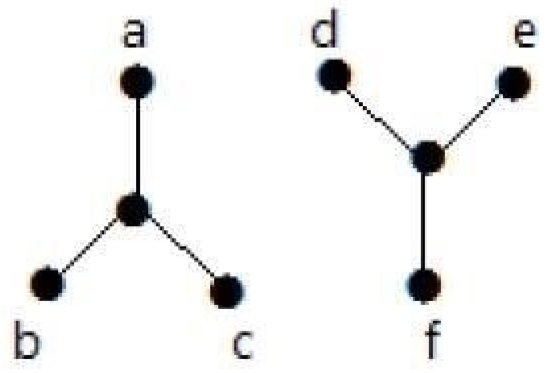
Connectivity of graph

- Connectivity refers to connection between two or more nodes or things

Connected graph



Disconnected graph

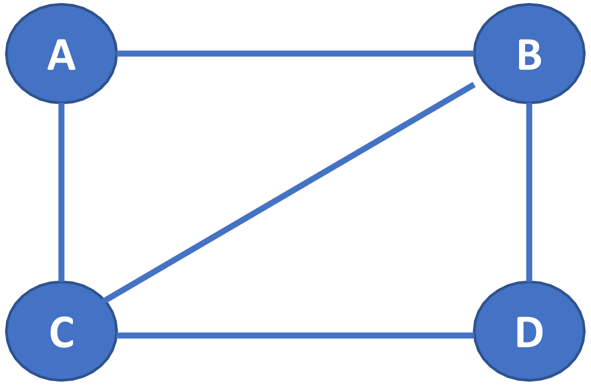


Connectivity of the graph

- A graph is connected if there is a path between every pair of vertices.
- The graph that is not connected is called disconnected graph.
- In connected graph there is no unreachable vertex.
- In the area of Information Technology the connectivity refers to internet connectivity through which various devices are connected to global network

Connectedness in the Direct graph using adjacency matrix

- 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
- For example: Below graph is connected



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

Connectedness in the Direct graph using adjacency matrix

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 the graph using Adjacency Matrix

Function to read adjacency matrix:

```
void read_ad_matr(int graph[][],int n)
{
   int i,j;
   for(i=0;i<n;i++)
    {
      for(j=0;j<n;j++)
        {
          scanf(“%d”, &graph[i][j]);//reading the values into two dimensional array
         }
    }
}
```

Connectivity of the graph using Adjacency Matrix

Function to traverse the graph using DFS

```
void traverse(int u, int visited[])
{
    visited[u] = 1; //mark v as visited
    for(int v = 0; v<n; v++)
   {
       if(graph[u][v])
       {
         if(!visited[v])
         traverse(v, visited);
      }
   }
}
```

Function to traverse the graph using bfs:

```
void traverse(int s, int visited[],int n,int graph[][])
 {
   int f,r,v,q[10];
   f=0,r=-1;
   q[++r]=s;
   visited[s]=1;// mark the nodes as visited
   while(f<=r)
   {
      s=q[f++];
      for(v=0;v<n;v++)
     {
      if(graph[s][v]==1) //adjacent nodes
       {
```

Connectivity of the graph using Adjacency Matrix

```
    if(visited[v]==0) //nodes not visited
     {
        visited[v]=1; //mark as visited
        q[++r]=v;
     }
}
 }
   }
}
```

Connectivity of the graph using Adjacency Matrix

Function to check whether the graph is connected or not

```
int isConnected()
{
    int vis[NODE];   //for all vertex u as start point, check whether all nodes are visible or not
  for(int u=0; u < NODE; u++)
     {
        for(int i = 0; i<NODE; i++)
            vis[i] = 0; //initialize as no node is visited
        traverse(u, vis); // any traversal method either bfs or dfs
         for(int i = 0; i<NODE; i++)
         {
            if(!vis[i]) //if there is a node, not visited by traversal, graph is not connected
            return 0;
          }
      }
   return 1;}
```

THANK YOU

Saritha

Department of Computer Science & Engineering

Saritha.k@pes.edu

9844668963