19.Connected Components

Intorduction: Connected component (graph theory), a set of vertices in a graph that are linked to each other by paths.

(or)

Connected component (topology), a maximal subset of a topological space that cannot be covered by the union of two disjoint open sets.

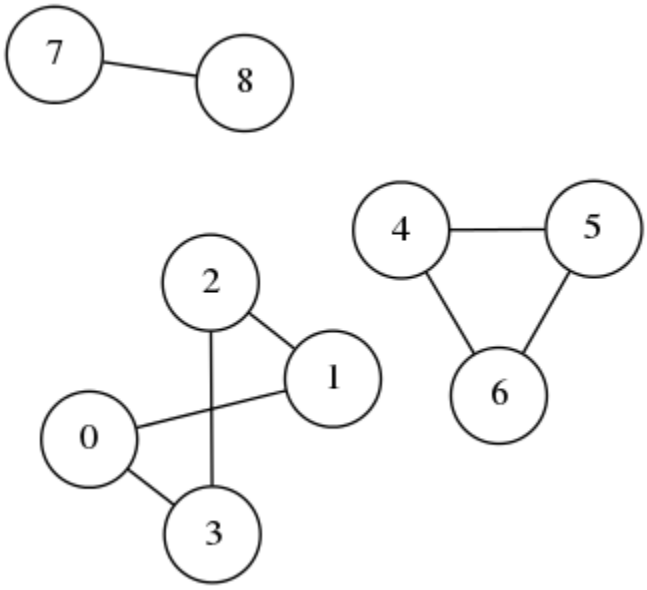
(or)

Connected component is a subgraph in which any two vertices are connected to each other by paths, and which is connected to no additional vertices in the supergraph

Definition: A graph G is said to be **connected** if there exist a path between every pair of vertices.
A graph is G said to be **disconnected** , if it doesn't contain at least two connected vertices.

Undirected graphs

A connected graph has only one connected component, which is the graph itself, while unconnected graphs have more than one component. For example, the graph shown below has three components, (0, 1, 2, 3), (4, 5, 6), and (7, 8).



In an undirected graph, a vertex v is reachable from a vertex u if there is a path from u to v. In this definition, a single vertex is counted as a path of length zero, and the same vertex may occur more than once within a path. Reachability is an equivalence relation, since: It is reflexive, symmetric and transitive.

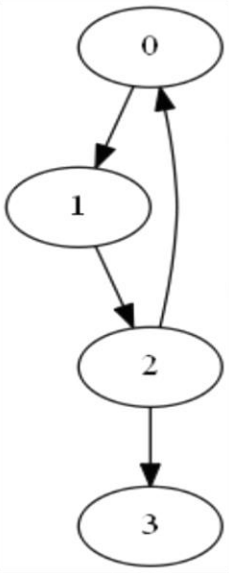
Directed graphs In case of directed graphs an additional distinction has to be made:

- As before a graph is said to be disconnected if it doesn't exist a path from a start vertex to any other vertex. Not even ignoring the edge directions .
- A directed graph is said to be weakly connected if ignoring all the directions (i.e. just considering the graph as an undirected one) results in a connected graph.

If the graph isn't weakly connected, weak connected components can be found as in the case for the undirected graph, i.e. by just marking every node encountered from a starting vertex as being in the same component.

- A directed graph is said to be strongly connected if from any node there's always a path to reach any other one. This is the criterion city roads should be designed with.

An example is the following graph



The connected components of a graph can be found using either a depth-first search (DFS), or a breadth-first search (BFS)

To find Connected components in directed graph We have the following algorithms

- Kosaraju's algorithm for strongly connected components.
- Tarjan's Algorithm to find Strongly Connected Components

Problem: Find N.of connected components in a given Graph?

Finding connected components for an undirected graph is an easier task. We simple need to do either BFS or DFS starting from every unvisited vertex, and we get all strongly connected components.

Algorithm to find n.of connected components in given graph:

Input: Graph -G,N.of nodes -N
Output: n.of connected components, DFS traversal

```
Connected Component (G,N){
     for each vertex in  v in G.V
          do flag[v] ← -1
     for  each vertex in G.V
          do if(flag[v]== -1)
               DFS(v,flag)
               count++
     return count

}
DFS(v,flag){
     print v
     flag[v]←1
     for each adjacent node u of v
          do if (flag[u]==-1)
               DFS(u,flag)
     }
```

Approach:

The idea is to use a variable count to store the number of connected components and do the following steps:

1. Initialize all vertices as unvisited.
2. For all the vertices check if a vertex has not been visited, then perform DFS on that vertex and increment the variable count by 1.

Time complexity Analysis

- Time complexity of above solution is O(V + E) as it does simple DFS for given graph.
- If we use any other data structure(fibanaccoi heap or min heap) We can reduce Time complexity O(V+ ElogV) or O(V+VlogE)

Applications

1.Spaning Trees
2.Clustering
3.Fraud detection
4.Entity Resolution
5. Component labeling is commonly used in image processing, to join neighboring pixels into connected regions which are the shapes in the image