DSA Lab
Name: Siddhi Parekh
Reg No.: 221071047
Batch: C
SY Comps

Experiment No.: 9

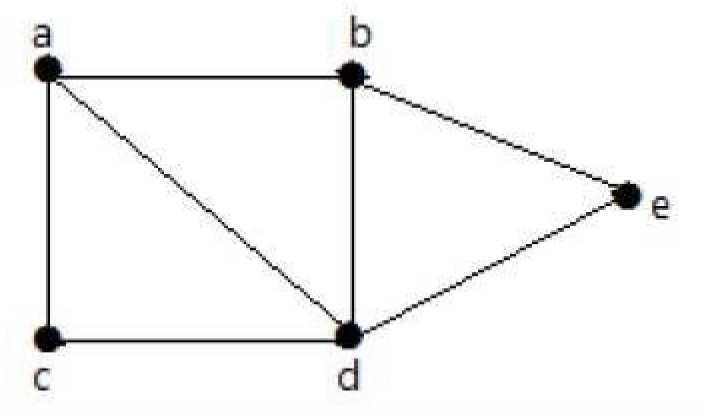
AIM:
Graph using an adjacency list and check whether the created graph connected or not.
THEORY:
A Graph is a non-linear data structure consisting of vertices and edges. The vertices are sometimes also referred to as nodes and the edges are lines or arcs that connect any two nodes in the graph. More formally a Graph is composed of a set of vertices(V) and a set of edges(E). The graph is denoted by G(E, V).

Components of a Graph

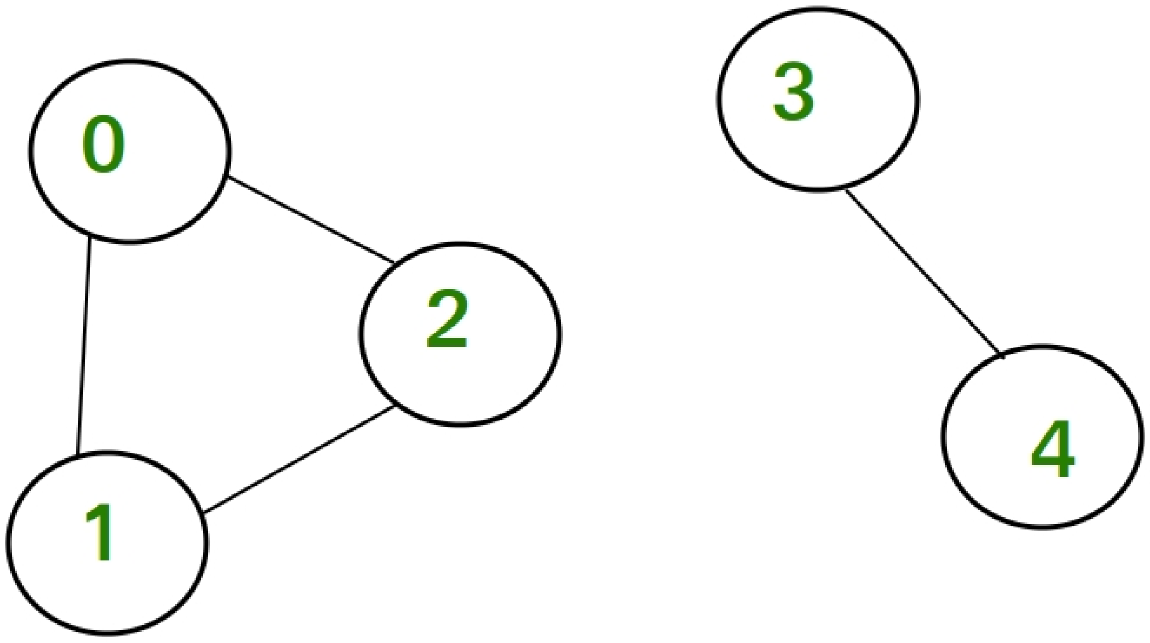
- Vertices: Vertices are the fundamental units of the graph. Sometimes, vertices are also known as vertex or nodes. Every node/vertex can be labeled or unlabelled.
- Edges: Edges are drawn or used to connect two nodes of the graph. It can be ordered pair of nodes in a directed graph. Edges can connect any two nodes in any possible way.

Connected Graph:

A connected graph is one where there is a path between every pair of vertices. In contrast, a disconnected graph has at least two vertices without a connecting path.



Here we can traverse any vertex of the graph from any vertex.

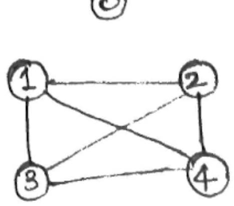


Here there is no connection between left graph and right graph so its not connected graph.

ALGORITHM:

1. Start.
2. In DFS function color all vertex as WHITE as no vertex visited.
3. Now start from starting vertex and color it GRAY as it discovered now.
4. Keep traversing in depth wise and if found vertex is coloured WHITE that means it is not visite hence call that vertex recursively.
5. While backtracking mark all vertex as BLACK as they discovered
6. Now in isConnected function create count variable and initialize it to 0
7. Check for every vertex by calling DFS function
8. If in one traversal all vertex all covered then graph is connected and count=1
9. Else we require more than one traversal for covering all vertex then count>1
10. If count=1 return Graph is connected
11. Else Graph is not connected.

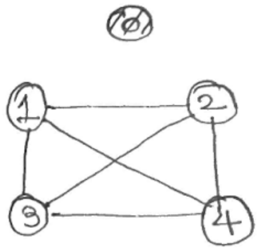
Example :

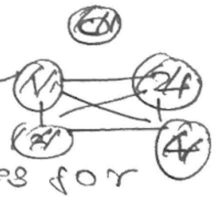


At first
count = 0.

Loop start from i=0
vertex '0' is not visited so it
enters loop as color '0' == white
and calls DFSvisited('0').
As '0' is not connected to
any other vertex marked '0' as
Black and proceeds by increamenting
count as count = 1.

for i = 1
vertex '1' is not visited so
it calls DFSvisited('1')
as 1st adjecent is 2
2st adjecent is 3
and 3 has no more vertex to explore which
is not visited
hence vertex 3 = Black as discovered
similery vertex 1, 2 and 4 also coloured Black
as all vertex visited
and count increamented , count = 2





for i=2 two is coloured Black hence goes for
next iteration
for i=3 '3' is coloured Black hence goes for
next iteration
for i=4 '4' is also coloured Black hence
go for next iteration.
~~As i=4 , i is not~~ smaller

loop breaks as all iterations has completed
Now, count = 2
as count > 1
hence graph is not connected.

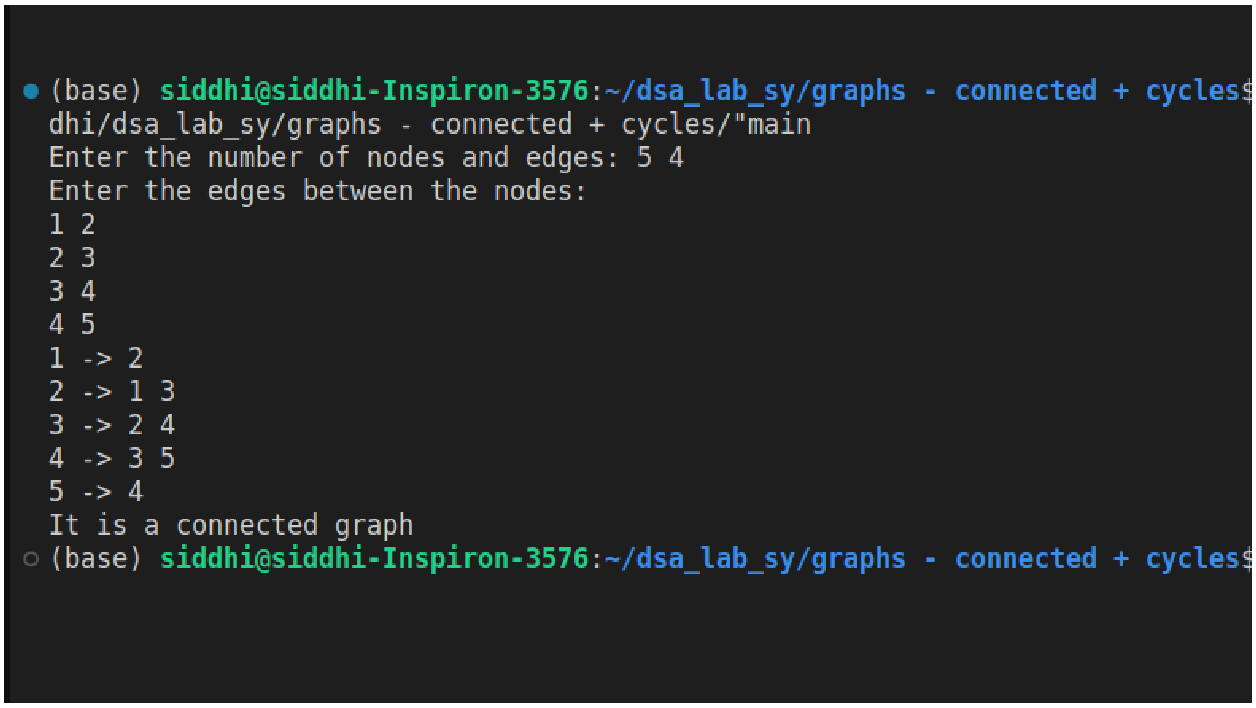
CONCLUSION:

In this experiment we have understood to implement a graph using an adjacency list and also to check that the graph is connected or not.

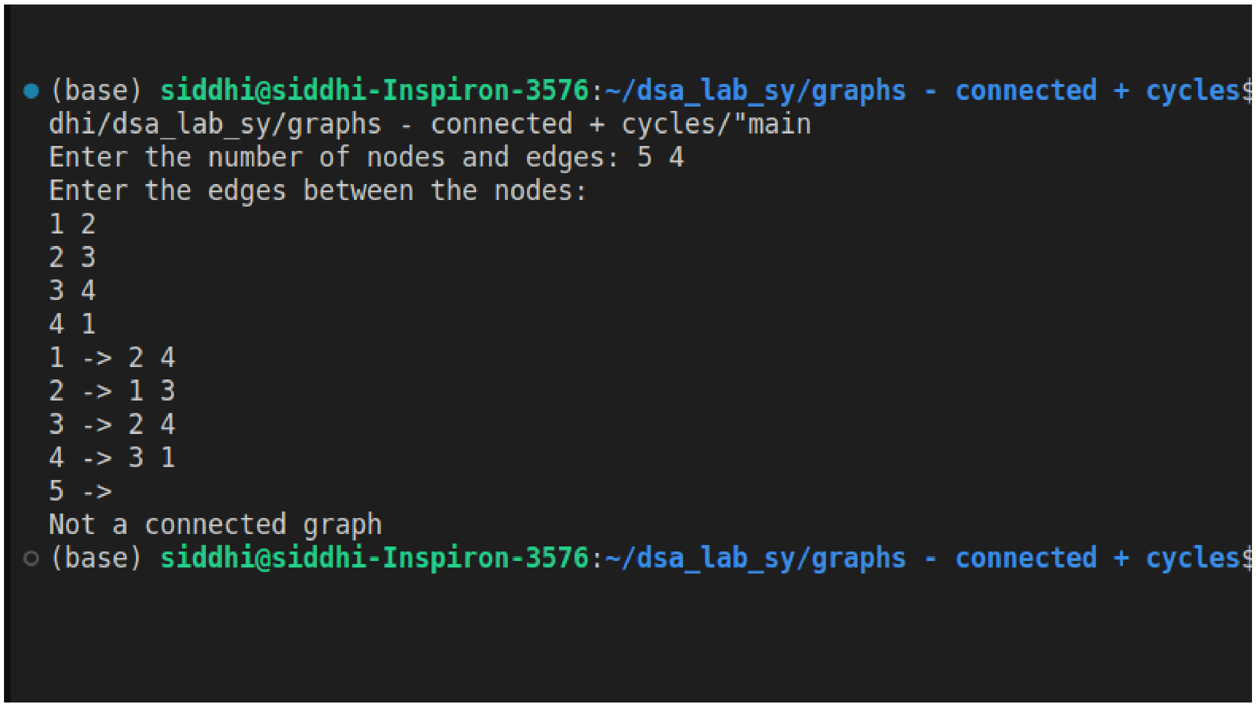
CODE:

```
#include <iostream>
#include <vector>

using namespace std;

void createList(vector<int> adjList[], int e)
{
        cout << "Enter the edges between the nodes:" << endl;
        for (int i = 1; i <= e; i++)
        {
        int u, v;
        cin >> u >> v;
        adjList[v].push_back(u);
        adjList[u].push_back(v);
        }
}

void display(vector<int> adj[], int n)
{
        for (int i = 1; i <= n; i++)
        {
        cout << i << " -> ";
        for (auto x : adj[i])
        {
        cout << x << " ";
        }

        cout << endl;
        }
}

void dfs_visit(vector<int> adj[], int a, vector<bool> &visited)
{
        visited[a] = true;

        for (auto x : adj[a])
        {
        if (!visited[x])
        {
        dfs_visit(adj, x, visited);
        }
        }
}

void dfs_check(vector<int> adj[], int n)
{
        vector<bool> visited(n + 1, false);
        int counter = 0;

        for (int i = 1; i <= n; i++)
        {
        if (!visited[i])
        {
        counter++;
        dfs_visit(adj, i, visited);
        }
        }

        if (counter == 1)
        {
        cout << "It is a connected graph" << endl;
        }
        else
        {
        cout << "Not a connected graph" << endl;
        }
}

int main()
{
        int n, e;
        cout << "Enter the number of nodes and edges: ";
        cin >> n >> e;

        vector<int> adjList[n + 1];

        createList(adjList, e);
        display(adjList, n);

        dfs_check(adjList, n);
        return 0;
}
```

OUTPUT:



```
● (base) siddhi@siddhi-Inspiron-3576:~/dsa_lab_sy/graphs - connected + cycles$
  dhi/dsa_lab_sy/graphs - connected + cycles/"main
  Enter the number of nodes and edges: 5 4
  Enter the edges between the nodes:
  1 2
  2 3
  3 4
  4 5
  1 -> 2
  2 -> 1 3
  3 -> 2 4
  4 -> 3 5
  5 -> 4
  It is a connected graph
○ (base) siddhi@siddhi-Inspiron-3576:~/dsa_lab_sy/graphs - connected + cycles$
```



```
● (base) siddhi@siddhi-Inspiron-3576:~/dsa_lab_sy/graphs - connected + cycles$
  dhi/dsa_lab_sy/graphs - connected + cycles/"main
  Enter the number of nodes and edges: 5 4
  Enter the edges between the nodes:
  1 2
  2 3
  3 4
  4 1
  1 -> 2 4
  2 -> 1 3
  3 -> 2 4
  4 -> 3 1
  5 ->
  Not a connected graph
○ (base) siddhi@siddhi-Inspiron-3576:~/dsa_lab_sy/graphs - connected + cycles$
```