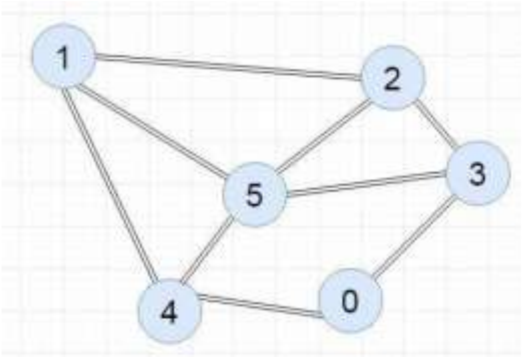
EX.NO : 10(A)	**REPRESENTATION OF GRAPH**
DATE :	

PROGRAM STATEMENT:

To write A C++ Program to represent the Adjacency Matrix.

.



ALGORITHM:

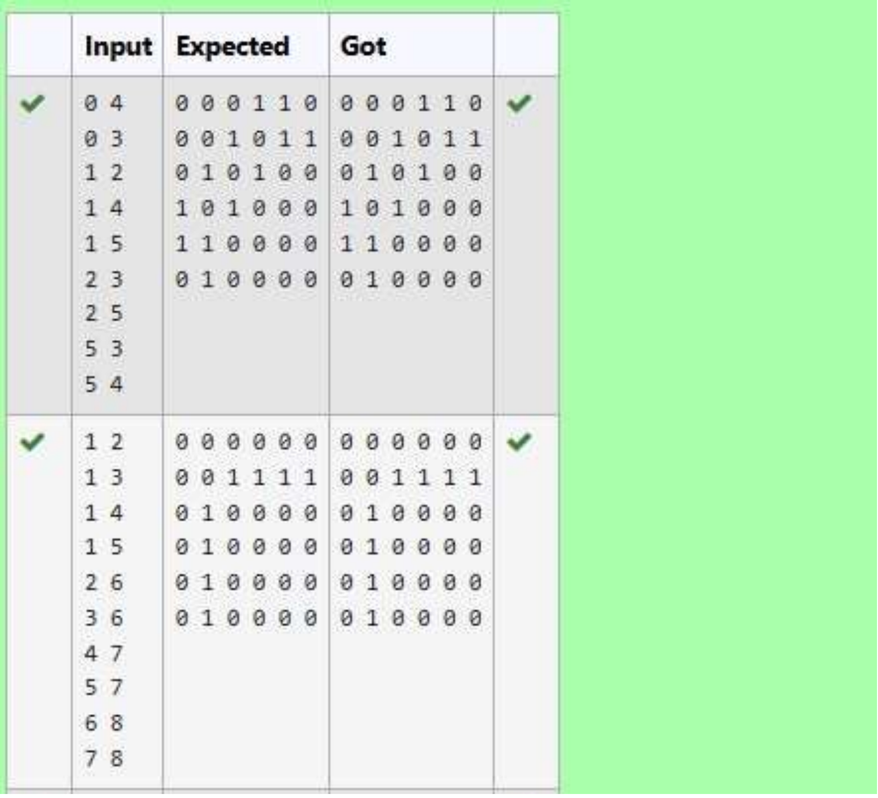
1. Start the program.
2. Define a graph matrix: Create a 2D array vertArr to represent the graph, initialized to 0.
3. Add edges: The add_edge function sets the corresponding matrix entries to 1 for both directions (undirected graph).
4. Input edges: Use a loop to take 6 pairs of inputs representing edges.
5. Display adjacency matrix: The displayMatrix function prints the adjacency matrix, showing connections between vertices.
6. End the program.

PROGRAM:

```
#include<iostream>
usingnamespacestd;
int vertArr[20][20];
int count = 0;
voiddisplayMatrix(int v)
{
  int i, j;
  for(i = 0; i< v; i++)
  {
    for(j = 0; j< v; j++)
    {
    cout << vertArr[i][j] << " ";
    }
```

```
         cout << endl;
      }
    }
    void add_edge(int u, int v)
    {
         vertArr[u][v] = 1;
         vertArr[v][u] = 1;
    }
    int main()
    { int v= 6,a,b;
    for(int i=0;i<6;i++)
      { cin>>a>>b;
         add_edge(a, b);
    }
      displayMatrix(v);
      return 0;
      }
```

OUTPUT :



	Input	Expected	Got	
✓	0 4 0 3 1 2 1 4 1 5 2 3 2 5 5 3 5 4	0 0 0 1 1 0 0 0 1 0 1 1 0 1 0 1 0 0 1 0 1 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0	0 0 0 1 1 0 0 0 1 0 1 1 0 1 0 1 0 0 1 0 1 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0	✓
✓	1 2 1 3 1 4 1 5 2 6 3 6 4 7 5 7 6 8 7 8	0 0 0 0 0 0 0 0 1 1 1 1 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0	0 0 0 0 0 0 0 0 1 1 1 1 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0	✓

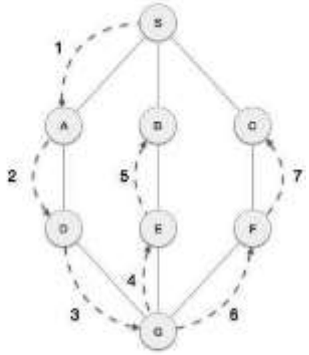
RESULT:

Thus, the C++ program to represent the Adjacency Matrix is created successfully.

EX.NO : 10(B)	**BREADTH FIRST SEARCH-BFS**
DATE :	

PROGRAM STATEMENT:

To write A CPP Program to implement BFS using vectors and queue.
.



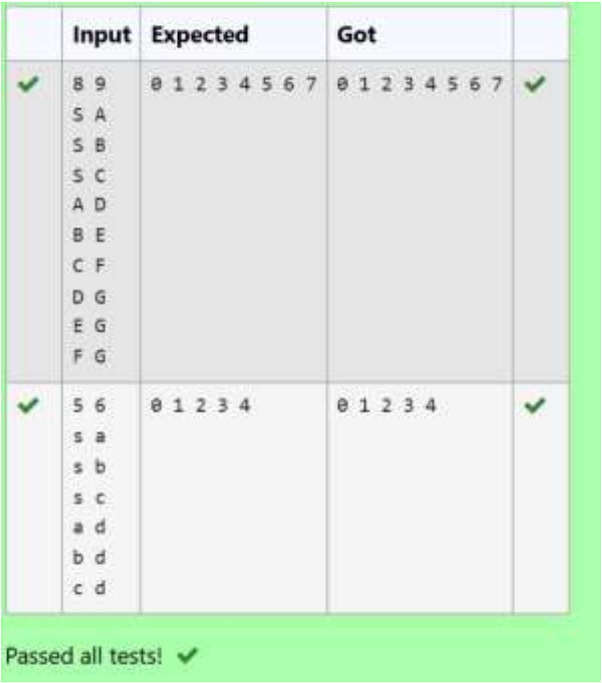
ALGORITHM:

1. Start the program.
2. Define graph structure: Create a 2D vector `g` to store the adjacency list and a vector `v` to track visited nodes.
3. Edge addition: Define the `edge` function to add an undirected edge between nodes by updating the adjacency list.
4. BFS traversal: Implement BFS using a queue. Start from a node, mark it visited, and explore its neighbors in the queue.
5. Input edges and nodes: Read the number of nodes and edges from the user, followed by pairs of nodes representing edges. Store these in the adjacency list.
6. End the program: Traverse the graph, calling BFS on unvisited nodes, and output the BFS traversal order.

PROGRAM:

```
#include <bits/stdc++.h>
using namespace std;
vector<bool> v;
vector<vector<char> > g;
void edge(int a, int b)
{
g[a].pb(b);
}
void bfs(int u)
{
queue<char> q; q.push(u);
v[u] =true; while(!q.empty())
{
int f=q.front();
q.pop();
cout << f << " ";
        for (auto i= g[f].begin(); i != g[f].end(); i++)
        {
                if (!v[*i])
                {
                           q.push(*i); v[*i] =true;
                             }
                  }
       }
}
int main()
{
       int  n,  e;  cin
       >> n >> e;

       v.assign(n, false);
       g.assign(n, vector<char>());

       char a, b;
       for (int i= 0; i< e; i++)
       {
       cin >> a >> b;
       edge(a, b);
       }
       for (int i= 0; i< n; i++)
       {
        if (!v[i])
             bfs(i);
       } return 0;
}
```

OUTPUT :



	Input	Expected	Got	
✓	8 9 S A S B S C A D B E C F D G E G F G	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7	✓
✓	5 6 s a s b s c a d b d c d	0 1 2 3 4	0 1 2 3 4	✓

Passed all tests! ✓

RESULT:

Thus, the C++ program to implement BFS using vectors and queue is created successfully.

EX.NO : 10(C)	**DEPTH FIRST SEARCH-DFS**
DATE :	

PROGRAM STATEMENT:

To write A C++ Program to implementation DFS using Vector STL

ALGORITHM:

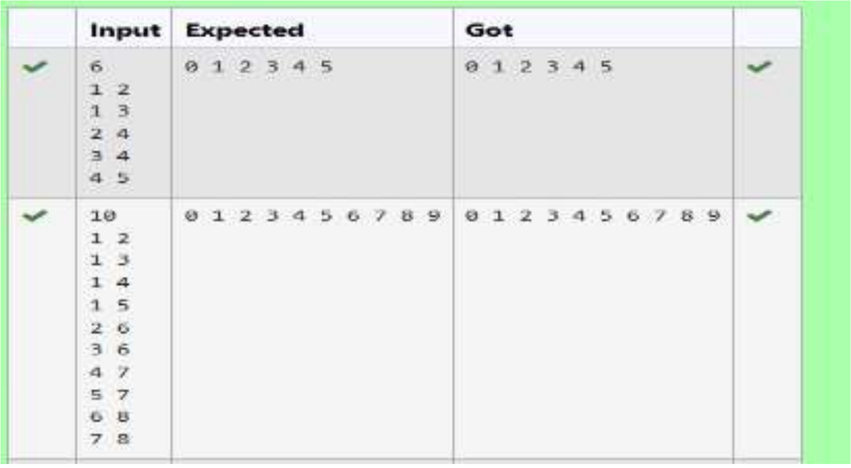
1. Start the program.
2. Define graph structure: Create a 2D vector g to store the adjacency list and a vector v to track visited nodes.
3. Edge addition: Define the addEdge function to add an undirected edge between nodes by updating the adjacency list.
4. DFS visit: Implement the dfsVisit function to visit nodes recursively. Mark the node as visited and explore its neighbors.
5. DFS traversal: Implement the dfs function, iterating over all nodes and calling dfsVisit for unvisited nodes to ensure the entire graph is explored.
6. End the program: Input the number of nodes and edges from the user, then read and store the edges. Perform DFS and output the nodes in the order they are visited.
7. End the program.

PROGRAM:

```
#include <bits/stdc++.h>
using namespace std;
vector< vector <int>> g;
 vector<bool> v;
void addEdge(int a, int b)
{
      g[a].push_back(b);
      g[b].push_back(a);
}
void dfsVisit(int u)
{
v[u] =true; cout << u << " ";
for(auto i= g[u].begin(); i != g[u].end(); i++)
{
 if(!v[*i]) dfsVisit(*i);
}
}
void dfs(int n)
```

```
{ for(int u = 0; u < n; u++)
        {
        if(!v[u])
              dfsVisit(u);
        }
}
int main()
{
        int n, e;
        cin >> n >> e;

        v.assign(n, false);
        g.assign(n, vector<int>());
        int a, b;
        for(int i= 0; i< e; i++)
        {
      cin >> a >> b;
      addEdge(a, b);
        }
        dfs(n);
}
```

OUTPUT :



	Input	Expected	Got	
✓	6 1 2 1 3 2 4 3 4 4 5	0 1 2 3 4 5	0 1 2 3 4 5	✓
✓	10 1 2 1 3 1 4 1 5 2 6 3 6 4 7 5 7 6 8 7 8	0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9	✓

RESULT:

Thus, the C++ program to write C++ Program to implementation DFS using Vector STL is created successfully.

EX.NO : 10(D)	**TOPOLOGICAL SORT**
DATE :	

PROGRAM STATEMENT:

To write A C++ Program for Topological Sorting

ALGORITHM:

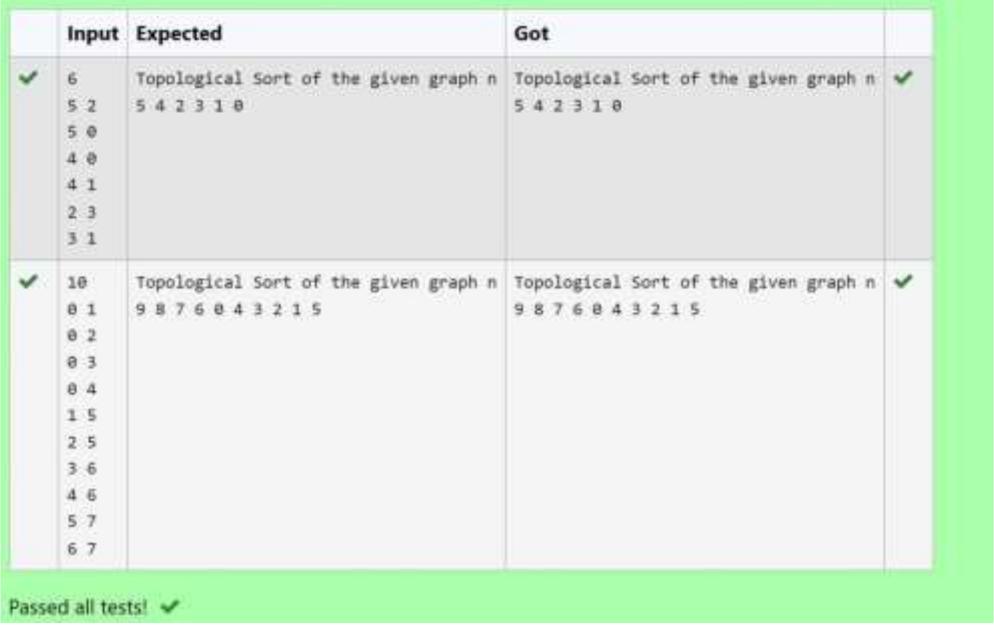
1. Initialize the graph with vertices and edges.
2. Add edges to the adjacency list.
3. Create a visited array and iterate over all vertices.
4. Use DFS to recursively explore unvisited vertices.
5. Push each vertex to the stack after processing its neighbors.
6. Pop and print vertices from the stack to get the topological order.

PROGRAM:

```
#include <iostream>
#include <list>
#include <stack>
using namespace std;
class Graph
{
int V;
list<int>* adj;
void topologicalSortUtil(int v, boolvisited[], stack<int>& Stack);
  public:
     Graph(int V); // Constructor
    voidaddEdge(int v, int w);
voidtopologicalSort();
};
Graph::Graph(int V)
{ this->V = V; adj= new list<int>[V];
}
void Graph::addEdge(int v, int w)
{
 adj[v].push_back(w);
}
void Graph::topologicalSortUtil(int v, boolvisited[], stack<int>& Stack)
{
```

```
visited[v] =true;

    list<int>::iterator i;
    for (i= adj[v].begin(); i != adj[v].end(); ++i)
     if (!visited[*i]) topologicalSortUtil(*i, visited, Stack);
          Stack.push(v);
}
void Graph::topologicalSort()
{

stack<int> Stack; bool* visited=newbool[V];

    for (int i= 0; i< V; i++)
    visited[i] = false;
    for (int i= 0; i< V; i++)
    if(visited[i] == false)
    topologicalSortUtil(i, visited, Stack);

    while (Stack.empty() == false)
    {
    cout << Stack.top() << "";
           Stack.pop();
    }
}
int main()
{       int
  a,b,n;
    cin>>n;
    Graphg(n);
    for(int i=0;i<6;i++)
    {
    cin>>a>>b;
     g.addEdge(a, b);
    }
    cout << "Topological Sort ofthe givengraphn"<<endl;
    g.topologicalSort();
    return 0;
}
```

OUTPUT :



	Input	Expected	Got	
✓	6 5 2 5 0 4 0 4 1 2 3 3 1	Topological Sort of the given graph n 5 4 2 3 1 0	Topological Sort of the given graph n 5 4 2 3 1 0	✓
✓	10 0 1 0 2 0 3 0 4 1 5 2 5 3 6 4 6 5 7 6 7	Topological Sort of the given graph n 9 8 7 6 0 4 3 2 1 5	Topological Sort of the given graph n 9 8 7 6 0 4 3 2 1 5	✓

Passed all tests! ✓

RESULT:

Thus, the C++ program for Topological Sorting STL is created successfully.