

<b>NAME:</b>	Harshal Chawan
<b>UID:</b>	2021300019
<b>SUBJECT</b>	Design and Analysis of Algorithm
<b>EXPERIMENT NO :</b>	09
<b>DATE OF PERFORMANCE</b>	17/04/2023
<b>DATE OF SUBMISSION</b>	23/04/2023
<b>AIM:</b>	To implement Approximation algorithms (The vertex-cover problem).
<b>PROBLEM STATEMENT 1:</b>	Approximation algorithms (The vertex-cover problem)
<b>ALGORITHM and THEORY:</b>	<p><b>THEORY :</b></p> <p>A vertex cover of an undirected graph is a subset of its vertices such that for every edge (u, v) of the graph, either 'u' or 'v' is in the vertex cover. Although the name is Vertex Cover, the set covers all edges of the given graph. Given an undirected graph, the vertex cover problem is to find minimum size vertex cover.</p> <p>Vertex Cover Problem is a known NP Complete problem, i.e., there is no polynomial-time solution for this unless <math>P = NP</math>. There are approximate polynomial-time algorithms to solve the problem though. Following is a simple approximate algorithm adapted from CLRS book.</p>

	<p><b>ALGORITHM :</b></p> <ol style="list-style-type: none"> <li>1) Initialize the result as { }</li> <li>2) Consider a set of all edges in given graph. Let the set be E.</li> <li>3) Do following while E is not empty.             <ol style="list-style-type: none"> <li>...a) Pick an arbitrary edge (u, v) from set E and add 'u' and 'v' to result.</li> <li>...b) Remove all edges from E which are either incident on u or v.</li> </ol> </li> <li>4) Return result.</li> </ol>
<p><b>PROGRAM:</b></p>	<pre>// Program to print Vertex Cover of a given undirected graph #include&lt;iostream&gt; #include &lt;list&gt; using namespace std;  // This class represents a undirected graph using adjacency list class Graph {     int V; // No. of vertices     list&lt;int&gt; *adj; // Pointer to an array containing adjacency lists public:     Graph(int V); // Constructor     void addEdge(int v, int w); // function to add an edge to graph     void printVertexCover(); // prints vertex cover };  Graph::Graph(int V) {     this-&gt;V = V;     adj = new list&lt;int&gt;[V]; }  void Graph::addEdge(int v, int w) { </pre>

```

adj[v].push_back(w); // Add w to v's list.
adj[w].push_back(v); // Since the graph is undirected
}

// The function to print vertex cover
void Graph::printVertexCover()
{
    // Initialize all vertices as not visited.
    bool visited[V];
    for (int i=0; i<V; i++)
        visited[i] = false;

    list<int>::iterator i;

    // Consider all edges one by one
    for (int u=0; u<V; u++)
    {
        // An edge is only picked when both visited[u] and
visited[v]
        // are false
        if (visited[u] == false)
        {
            // Go through all adjacents of u and pick the first not
            // yet visited vertex (We are basically picking an
edge
            // (u, v) from remaining edges.
            for (i= adj[u].begin(); i != adj[u].end(); ++i)
            {
                int v = *i;
                if (visited[v] == false)
                {
                    // Add the vertices (u, v) to the result
set.

                    // We make the vertex u and v visited so

```

```

that
// all edges from/to them would be
ignored

visited[v] = true;
visited[u] = true;
break;
    }
    }
}

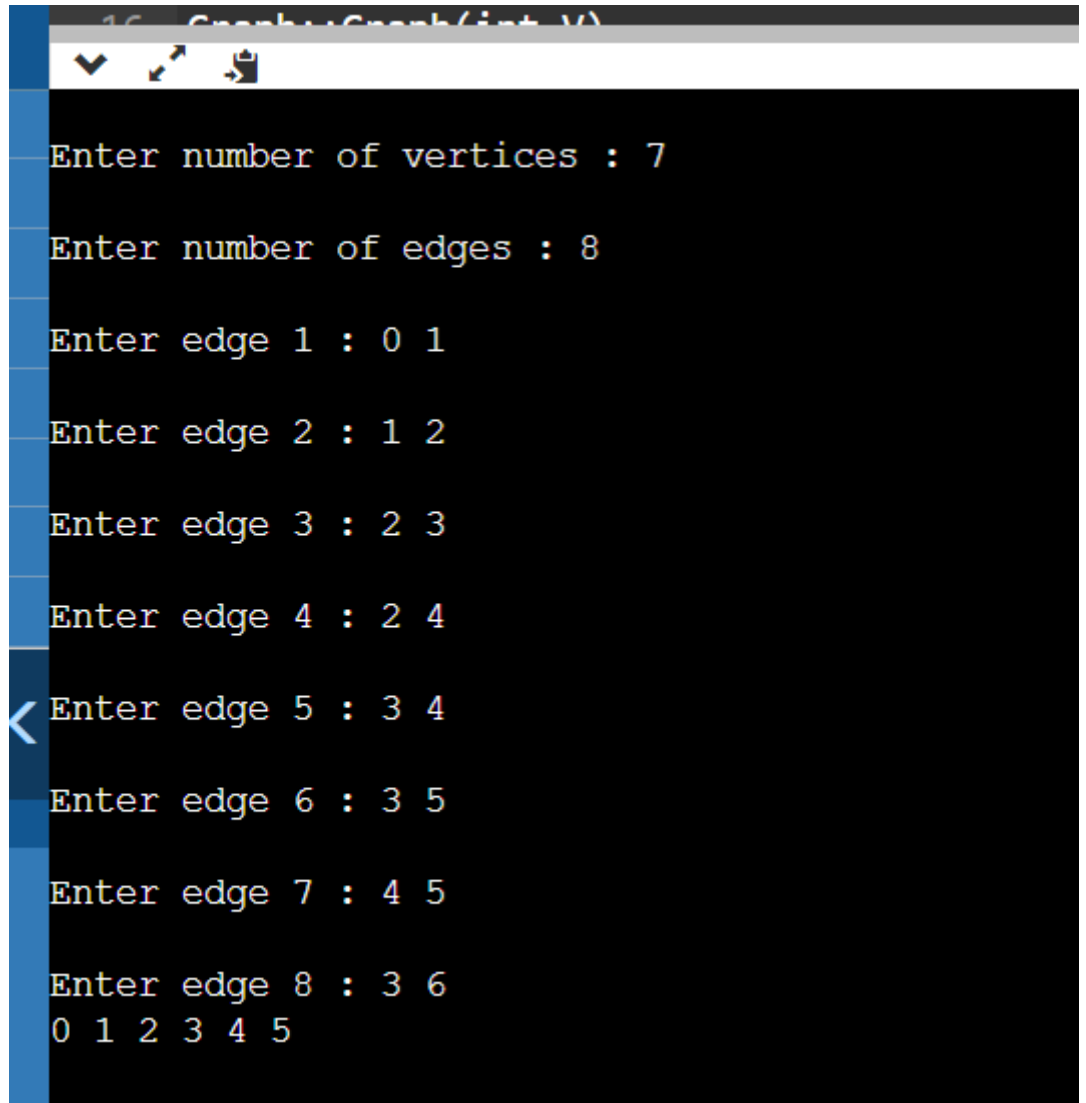
// Print the vertex cover
for (int i=0; i<V; i++)
    if (visited[i])
        cout << i << " ";
}

// Driver program to test methods of graph class
int main()
{
    int n,m,i,a,b;
    cout<<"\nEnter number of vertices : ";
    cin>>n;
    cout<<"\nEnter number of edges : ";
    cin>>m;
    Graph g(n);
    for(i=0;i<m;i++)
    {
        cout<<"\nEnter edge "<<(i+1)<<" : ";
        cin>>a;
        cin>>b;
        g.addEdge(a,b);
    }
}

```

```
g.printVertexCover();  
  
return 0;  
}
```

**OUTPUT:**



```
16 - Graphs - Graph (int V)  
Enter number of vertices : 7  
Enter number of edges : 8  
Enter edge 1 : 0 1  
Enter edge 2 : 1 2  
Enter edge 3 : 2 3  
Enter edge 4 : 2 4  
< Enter edge 5 : 3 4  
Enter edge 6 : 3 5  
Enter edge 7 : 4 5  
Enter edge 8 : 3 6  
0 1 2 3 4 5
```

<b>CONCLUSION:</b>	I have successfully understood and implemented the concept of vertex cover problem through this experiment. I was also able to understand how to find minimum size vertex cover when an undirected graph is given.
--------------------	--