NAME:	Gitanjali Gangurde
UID:	2021300034
BRANCH:	Computer engineering
BATCH:	B
SUBJECT:	DAA
EXPT NO:	9
AIM:	Approximate solution for vertex cover problem.
THEORY:	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. Naive Approach: Consider all the subset of vertices one by one and find out whether it covers all edges of the graph. For eg. in a graph consisting only 3 vertices the set consisting of the combination of vertices are:{0,1,2,{0,1},{0,2},{1,2},{0,1,2}} . Using each element of this set check whether these vertices cover all the edges of the graph. Hence update the optimal answer. And hence print the subset having minimum number of vertices which also covers all the edges of the graph.

PROGRAM:	`#include <iostream>`
	`#include <list>`
	`using namespace std;`
	
	`// This class represents a undirected graph using adjacency list`
	`class Graph`
	`{`

```cpp
    int V;          // No. of vertices
    list<int> *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->V = V;
    adj = new list<int>[V];
}

void Graph::addEdge(int v, int w)
{
    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
```

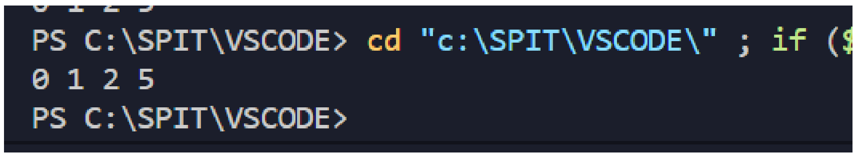
```
            // (u, v) from remaining edges.
            for (i = adj[u].begin(); i != adj[u].end(); ++i)
            {
                int v = *i;
                if (visited[v] == false)
                {
                    // Add the vertices (u, v) to the result
set.
                    // We 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()
{
    // Create a graph given in the above diagram
    Graph g(6);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(0, 3);
    g.addEdge(1, 3);
    g.addEdge(0, 4);
    g.addEdge(2, 5);

    g.printVertexCover();

    return 0;
}
```

RESULT:	 ```PS C:\SPIT\VSCODE> cd "c:\SPIT\VSCODE\" ; if ($``` ```0 1 2 5``` ```PS C:\SPIT\VSCODE>```
CONCLUSION:	From this experiment, I learnt the concept of approximate value and how I can apply that approach in solving the vertex cover problem.