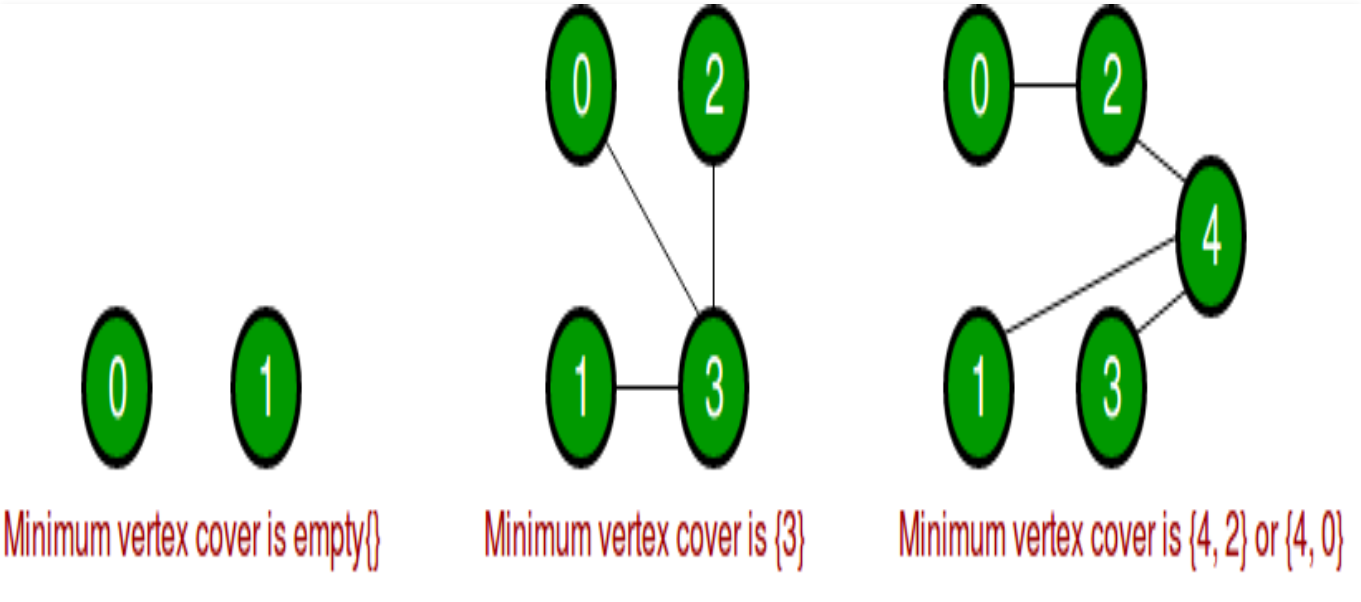
Vertex Cover Problem | Set 1 (Introduction and Approximate Algorithm)

Difficulty Level : Medium ● Last Updated : 04 Nov, 2020

A vertex cover of an undirected graph is a subset of its vertices such that for every edge

Related Articles

The following are some examples.

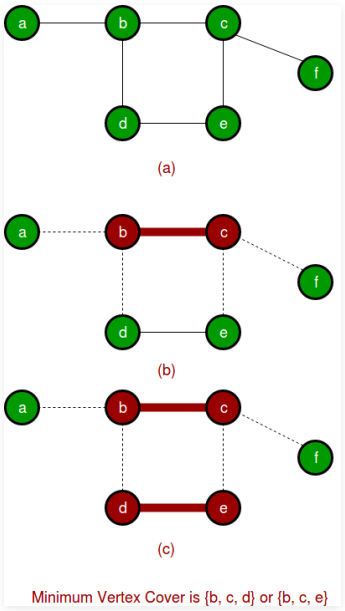


Vertex Cover Problem is a known NP Complete problem, i.e., there is no polynomial-time solution for this unless P = NP. There are approximate polynomial-time algorithms to solve the problem though. Following is a simple approximate algorithm adapted from

Approximate Algorithm for Vertex Cover:

```
1) Initialize the result as {}
2) Consider a set of all edges in given graph.  Let the set be E.
3) Do following while E is not empty
...a) Pick an arbitrary edge (u, v) from set E and add 'u' and 'v' to resu
...b) Remove all edges from E which are either incident on u or v.
4) Return result
```

Below diagram to show the execution of the above approximate algorithm:



How well the above algorithm perform?

It can be proved that the above approximate algorithm never finds a vertex cover whose size is more than twice the size of the minimum possible vertex cover (Refer this for proof)

Implementation:

C++

```
// Program to print Vertex Cover of a given undirected graph
#include<iostream>
#include <list>
using namespace std;

// This class represents a undirected graph using adjacency list
class Graph
{
    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)
```

```
            // (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(7);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 3);
    g.addEdge(3, 4);
    g.addEdge(4, 5);
    g.addEdge(5, 6);

    g.printVertexCover();

    return 0;
}
```

Java

```
// Java Program to print Vertex
// Cover of a given undirected graph
import java.io.*;
```

```
// graph using adjacency list
class Graph
{
    private int V;   // No. of vertices

    // Array  of lists for Adjacency List Representation
    private LinkedList<Integer> adj[];

    // Constructor
    Graph(int v)
    {
        V = v;
        adj = new LinkedList[v];
        for (int i=0; i<v; ++i)
            adj[i] = new LinkedList();
    }

    //Function to add an edge into the graph
    void addEdge(int v, int w)
    {
        adj[v].add(w);  // Add w to v's list.
        adj[w].add(v);  //Graph is undirected
    }

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

        Iterator<Integer> 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.
                i = adj[u].iterator();
                while (i.hasNext())
                {
```

```
                        // 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 j=0; j<V; j++)
            if (visited[j])
              System.out.print(j+" ");
    }

    // Driver method
    public static void main(String args[])
    {
        // Create a graph given in the above diagram
        Graph g = new Graph(7);
        g.addEdge(0, 1);
        g.addEdge(0, 2);
        g.addEdge(1, 3);
        g.addEdge(3, 4);
        g.addEdge(4, 5);
        g.addEdge(5, 6);

        g.printVertexCover();
    }
}

// This code is contributed by Aakash Hasija
```

Python3

```
# Python3 program to print Vertex Cover
# of a given undirected graph
from collections import defaultdict

# This class represents a directed graph
# using adjacency list representation
class Graph:
```

```
        # Default dictionary to store graph
        self.graph = defaultdict(list)

    # Function to add an edge to graph
    def addEdge(self, u, v):
        self.graph[u].append(v)

    # The function to print vertex cover
    def printVertexCover(self):

        # Initialize all vertices as not visited.
        visited = [False] * (self.V)

        # Consider all edges one by one
        for u in range(self.V):

            # An edge is only picked when
            # both visited[u] and visited[v]
            # are false
            if not visited[u]:

                # 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 v in self.graph[u]:
                    if not visited[v]:

                        # 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 j in range(self.V):
            if visited[j]:
                print(j, end = ' ')

        print()

# Driver code
```

```
g.addEdge(0, 2)
g.addEdge(1, 3)
g.addEdge(3, 4)
g.addEdge(4, 5)
g.addEdge(5, 6)

g.printVertexCover()

# This code is contributed by Prateek Gupta
```

C#

```
// C# Program to print Vertex
// Cover of a given undirected
// graph
using System;
using System.Collections.Generic;

// This class represents an
// undirected graph using
// adjacency list
class Graph{

// No. of vertices
public int V;

// Array of lists for
// Adjacency List Representation
public List<int> []adj;

// Constructor
public Graph(int v)
{
  V = v;
  adj = new List<int>[v];

  for (int i = 0; i < v; ++i)
    adj[i] = new List<int>();
}

//Function to add an edge
// into the graph
void addEdge(int v, int w)
{
   // Add w to v's list.
```

```
}

// The function to print
// vertex cover
void printVertexCover()
{
  // Initialize all vertices
  // as not visited.
  bool []visited = new bool[V];

  // 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.
      foreach(int i in adj[u])
      {
        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 j = 0; j < V; j++)
    if (visited[j])
```

```
public static void Main(String []args)
{
  // Create a graph given in
  // the above diagram
  Graph g = new Graph(7);
  g.addEdge(0, 1);
  g.addEdge(0, 2);
  g.addEdge(1, 3);
  g.addEdge(3, 4);
  g.addEdge(4, 5);
  g.addEdge(5, 6);

  g.printVertexCover();
}
}

// This code is contributed by gauravrajput1
```

Output:

```
0 1 3 4 5 6
```

The Time Complexity of the above algorithm is O(V + E).

Exact Algorithms:

Although the problem is NP complete, it can be solved in polynomial time for the following types of graphs.

1) Bipartite Graph

2) Tree Graph

The problem to check whether there is a vertex cover of size smaller than or equal to a given number k can also be solved in polynomial time if k is bounded by O(LogV) (Refer this)

We will soon be discussing exact algorithms for vertex cover.

This article is contributed by **Shubham**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Previous Next

RECOMMENDED ARTICLES

Page : 1 2 3

01 **Set Cover Problem | Set 1 (Greedy Approximate Algorithm)**
27, Mar 15

02 **K Centers Problem | Set 1 (Greedy Approximate Algorithm)**
26, Mar 15

03 **Vertex Cover Problem | Set 2 (Dynamic Programming Solution for Tree)**
28, Feb 15

04 **Travelling Salesman Problem | Set 2 (Approximate using MST)**
04, Nov 13

05 **Exact Cover Problem and Algorithm X | Set 1**
12, Jul 17

06 **Exact Cover Problem and Algorithm X | Set 2 (Implementation with DLX)**
29, Jul 17

07 **Finding minimum vertex cover size of a graph using binary search**
05, Jul 16

08 **Proof that vertex cover is NP complete**
03, Aug 18

Article Contributed By :

GeeksforGeeks

Current difficulty : Medium

Easy | Normal | Medium | Hard | Expert

Improved By : PrateekGupta10, GauravRajput1

Article Tags : NPHard, Graph

Practice Tags : Graph

Improve Article | Report Issue

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

GeeksforGeeks

5th Floor, A-118,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org