Sardar Patel Institute of Technology

Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058, India

(Autonomous College Affiliated to University of Mumbai)

Experiment No.	9
Aim	Implement the vertex-cover problem
Name	Aarush Kinhikar
UID No.	2021300063
Class & Division	SE Comps A, Batch D

Theory:

The vertex cover problem is a classic optimization problem in graph theory that involves finding the smallest possible set of vertices in a graph such that each edge in the graph is incident to at least one vertex in the set. In other words, a vertex cover is a set of vertices that covers all the edges in the graph.

The vertex cover problem is known to be an NP-hard problem, which means that it is difficult to find an exact solution to the problem in a reasonable amount of time for large graphs. However, there are many algorithms and heuristics that can be used to find approximate solutions to the problem.

The vertex cover problem has numerous practical applications in areas such as computer networking, social network analysis, and operations research. For example, in a computer network, the vertices might represent computers and the edges might represent connections between them, and finding a vertex cover can help to identify a minimum set of computers that must be monitored or secured to ensure the overall security and reliability of the network.

Algorithm:

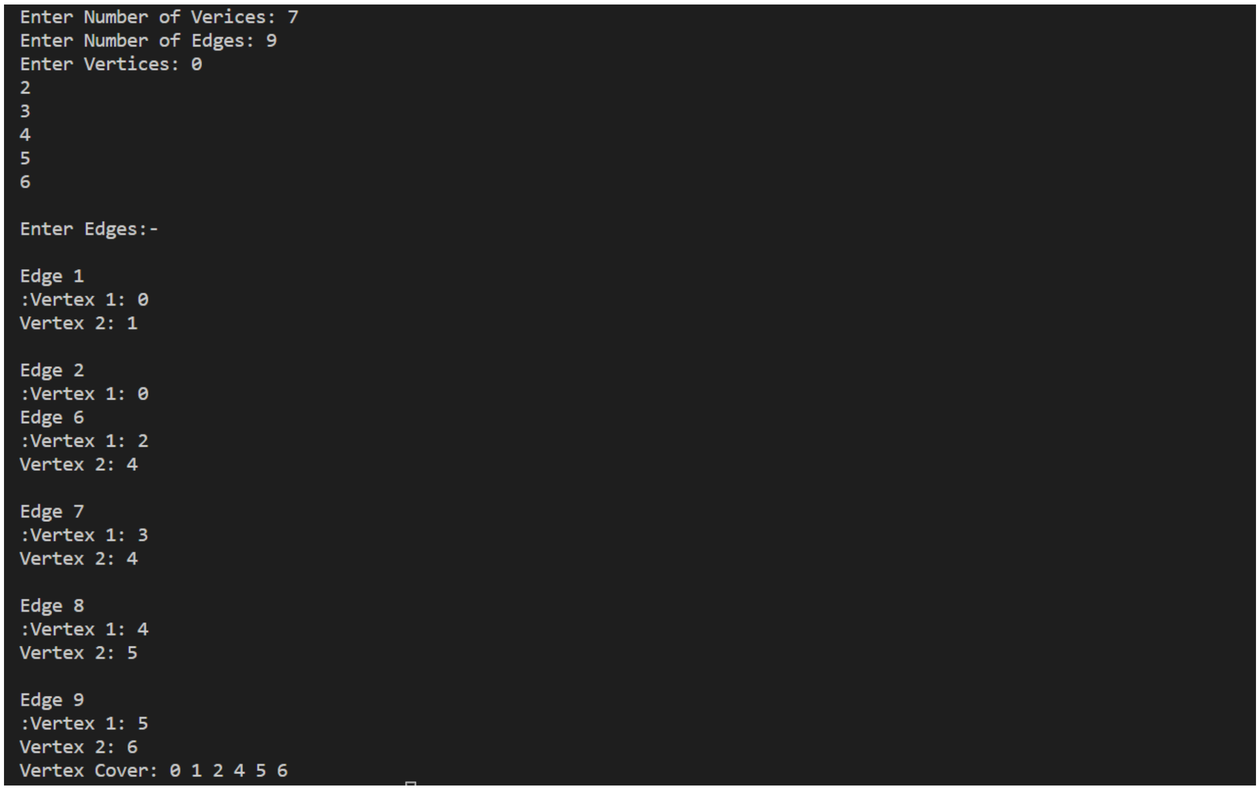
1. Initialize the vertex cover set to be empty.
2. While there are still uncovered edges in the graph:
3. Choose an arbitrary uncovered edge (u, v).
4. Add both vertices u and v to the vertex cover set.
5. Return the vertex cover set.

Program:

```
#include <stdio.h>
#include <stdlib.h>

typedef struct Edge {
    int u, v;
} Edge;

int vertex_cover[100];

void add_edge(int u, int v, Edge edges[], int* edge_count) {
    Edge e = {u, v};
    edges[(*edge_count)++] = e;
}

void print_vertex_cover(int vertex_count) {
    printf("Vertex Cover: ");
    for (int i = 0; i < vertex_count; i++) {
        if (vertex_cover[i]) {
            printf("%d ", i);
        }
    }
    printf("\n");
}

void vertex_cover_greedy(int vertex_count, Edge edges[], int edge_count) {
    for (int i = 0; i < edge_count; i++) {
        Edge e = edges[i];
        if (!vertex_cover[e.u] && !vertex_cover[e.v]) {
            vertex_cover[e.u] = 1;
            vertex_cover[e.v] = 1;
        }
    }
}

int main() {
```

```c
    int i,j,vertex_count,edge_count;

    printf("Enter Number of Verices: ");
    scanf("%d",&vertex_count);
    printf("Enter Number of Edges: ");
    scanf("%d",&edge_count);

    int vertices[vertex_count];
    printf("Enter Vertices: ");
    for(i=0;i<vertex_count;++i){
        scanf("%d",&vertices[i]);
    }

    Edge edges[vertex_count * vertex_count];
    int edge_index = 0,v1,v2;

    printf("\nEnter Edges:-\n");
    for(i=0;i<edge_count;++i){
        printf("\nEdge %d\n:",(i+1));
        printf("Vertex 1: ");
        scanf("%d",&v1);
        printf("Vertex 2: ");
        scanf("%d",&v2);
        add_edge(v1, v2, edges, &edge_index);
    }

    vertex_cover_greedy(vertex_count, edges, edge_count);
    print_vertex_cover(vertex_count);

    return 0;
}
```

Results:



```
Enter Number of Verices: 7
Enter Number of Edges: 9
Enter Vertices: 0
2
3
4
5
6

Enter Edges:-

Edge 1
:Vertex 1: 0
Vertex 2: 1

Edge 2
:Vertex 1: 0
Edge 6
:Vertex 1: 2
Vertex 2: 4

Edge 7
:Vertex 1: 3
Vertex 2: 4

Edge 8
:Vertex 1: 4
Vertex 2: 5

Edge 9
:Vertex 1: 5
Vertex 2: 6
Vertex Cover: 0 1 2 4 5 6
```

Conclusion:

the vertex cover experiment has provided valuable insights into the theory and practical implementation of the vertex cover problem. Through the use of algorithms and simulations, we were able to explore the efficiency and effectiveness of different approaches to finding the minimum vertex cover in a given graph. The results of the experiment demonstrate that while there are several algorithms that can be used to solve the vertex cover problem, each approach has its strengths and weaknesses. For example, the greedy algorithm provides a fast solution but may not always produce the optimal result, while the brute-force algorithm is accurate but can be very slow for large graphs.