**AA\_LAB-10\_Assignment**

**CE\_054**

**Aim :-** Find out the Minimum vertex cover using different method of Approximation.

Program :-

1. Approximation Algorithm for Vertex Cover using Greedy strategy (Select a vertex with the highest degree).

Code :-

// Author : Dhruv B Kakadiya

#include <stdio.h>

#include <stdlib.h>

int no\_of\_nodes;

int minimum\_cover(int \*\*adj\_matrix, int deg\_matrix[], int sol[]);

int find\_max\_index\_graph(int deg\_matrix[]);

int main()

{

    int nodes, edges, \_, \_\_, u\_, v\_, w\_, max\_flow;

    printf("enter number of nodes and edges\n");

    scanf("%d %d", &nodes, &edges);

    no\_of\_nodes = nodes;

    int \*adj\_matrix[nodes], deg\_matrix[nodes], sol[nodes];

    for (\_ = 0; \_ < nodes; \_++)

    {

        adj\_matrix[\_] = (int \*)calloc(nodes, sizeof(int));

        deg\_matrix[\_] = 0;

    }

    printf("enter edges in form u v w ;such that 0<= u, 0<=v \n");

    for (\_ = 0; \_ < edges ; \_++)

    {

        scanf("%d %d %d", &u\_, &v\_, &w\_);

        /\*undirected graph\*/

        adj\_matrix[u\_][v\_] = w\_;

        adj\_matrix[v\_][u\_] = w\_;

        deg\_matrix[u\_]++;

        deg\_matrix[v\_]++;

    }

    int solLen = minimum\_cover(adj\_matrix, deg\_matrix, sol);

    printf("Solution is:\n");

    for (\_ = 0; \_ < solLen; \_++)

    {

        printf("%d ", sol[\_]);

    }

    return 0;

}

int find\_max\_index\_graph(int deg\_matrix[])

{

    int \_, max\_deg, index\_max\_degree;

    max\_deg = deg\_matrix[0];

    for (\_ = 1; \_ < no\_of\_nodes; \_++)

    {

        if (max\_deg < deg\_matrix[\_])

        {

            max\_deg = deg\_matrix[\_];

            index\_max\_degree = \_;

        }

    }

    return index\_max\_degree;

}

int minimum\_cover(int \*\*adj\_matrix, int deg\_matrix[], int sol[])

{

    int \_, \_\_, index\_max\_degree, max\_deg, v\_, u\_;

    int solTop = 0;

    index\_max\_degree = find\_max\_index\_graph(deg\_matrix);

    max\_deg = deg\_matrix[index\_max\_degree];

    while (max\_deg > 0)

    {

        sol[solTop] = index\_max\_degree;

        solTop++;

        u\_ = index\_max\_degree;

        deg\_matrix[u\_] -= max\_deg;

        for (\_ = 0; \_ < no\_of\_nodes; \_++)

        {

            v\_ = \_;

            if (adj\_matrix[u\_][v\_] > 0)

            {

                adj\_matrix[u\_][v\_] = 0;

                adj\_matrix[v\_][u\_] = 0;

                deg\_matrix[v\_]--;

            }

        }

        /\*finding next max degree\*/

        index\_max\_degree = find\_max\_index\_graph(deg\_matrix);

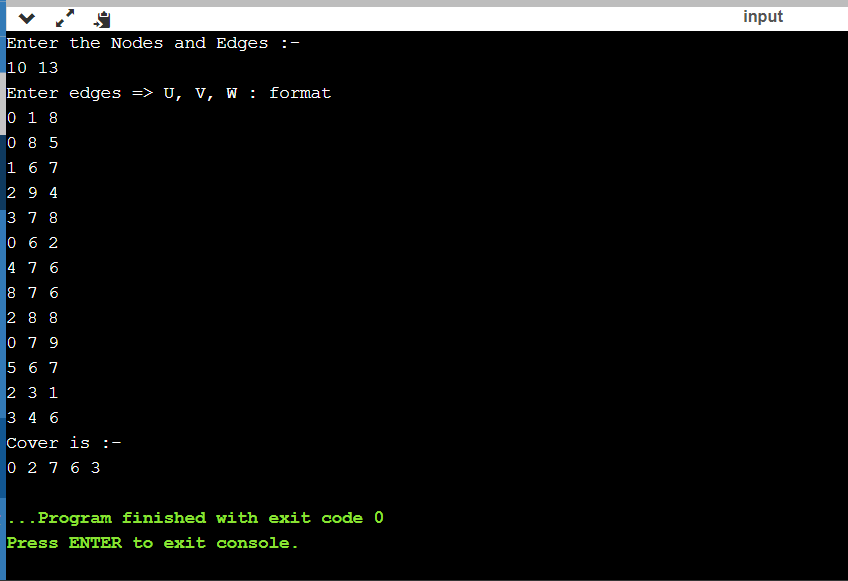
        max\_deg = deg\_matrix[index\_max\_degree];

    }

    return solTop;

}

Output :-



1. Approximation Algorithm for Vertex Cover which Randomly selects edges.

(Algorithm which has Approximation Factor 2)

Code :-

#Author : Dhruv B Kakadiya

def find\_max(matrix):

    maximum\_degree = 0

    maximum\_Node = 0

    for i in range(len(matrix)):

        countD = matrix[i].count(1)

        if countD > maximum\_degree:

            maximum\_degree = countD

            maximum\_Node = i

    return maximum\_degree, maximum\_Node

def removeEdge(matrix, n):

    for i in range(len(matrix)):

        for j in range(len(matrix)):

            if i == n or j == n:

                matrix[i][j] = 0

def anyEdgeLeft(matrix):

    left = False

    for i in range(len(matrix)):

        if matrix[i].count(1) > 0:

            left = True

    return left

if \_\_name\_\_ == "\_\_main\_\_":

    n = int(input("Enter number of vertex :- \n"))

    print("Enter matrix :- \n")

    matrix = []

    maximum\_degree = 0

    maximum\_Node = -1

    Ran\_approx = []

    for i in range(n):

        temp = list(map(int, input().split()))

        matrix.append(temp)

    while anyEdgeLeft(matrix):

        maximum\_degree, maximum\_Node = find\_max(matrix)

        Ran\_approx.append(maximum\_Node)

        removeEdge(matrix, maximum\_Node)

    print("Min vertex cover:", Ran\_approx)

Output :-

