**SVKM’s NMIMS**

**Mukesh Patel School of Technology Management & Engineering**

**Computer Engineering Department**

Program: B.Tech. Sem V

**Course: Design and Analysis of Algorithms**

**List of Experiments**

w.e.f. 1st Jul 2020

**Faculty:** Abhay Kolhe.

LAB Manual

**Experiment No.06**

PART B

|  |  |
| --- | --- |
| Roll No. B032 | Name: Naman Garg |
| Class : Btech CS B | Batch : B2 |
| Date of Experiment: 11-08-2020 | Date of Submission |
| Grade : | Time of Submission: |
| Date of Grading: |  |

**B.1 Software Code written by student:**

# Naman Garg

# B032

# aim: Implementation of Greedy Technique Algorithm Design.

#Write a program to implement Kruskal’s Algorithm for finding Minimum Spanning Tree (MST)

def find(i):

    while sel\_matrix[i] != i:

        i = sel\_matrix[i]

    return i

# Does union of i and j. It returns

# false if i and j are already in same

# set.

def unionfinder(i, j):

    a = find(i)

    b = find(j)

    sel\_matrix[a] = b

# Finds MST using Kruskal's algorithm

def krushkals(cost):

    mincost = 0  # Cost of min MST

    # Initialize sets of disjoint sets

    for i in range(v):

        sel\_matrix[i] = i

    # Include minimum weight edges one by one

    edge\_count = 0

    while edge\_count < v - 1:

        min = 999999        # inf high value for min initally

        a = -1

        b = -1

        for i in range(v):

            for j in range(v):

                if find(i) != find(j) and cost[i][j] < min:

                    min = cost[i][j]

                    a = i

                    b = j

        unionfinder(a, b)

        print(f'Edge {edge\_count}:({a}, {b}) cost:{min}')

        edge\_count += 1

        mincost += min

    print(f"Minimum cost= {mincost}")

if \_\_name\_\_ == "\_\_main\_\_":

        # Taking input from user

    v = int(input("no of vertices"))

    G = []

    # Input weights

    for i in range(v):

        G.append(

            list(map(int, input(f'vertex {i+1}: ').split())))

    print("Input Matrix")

    for i in range(v):

        for j in range(v):

            print(f"{G[i][j]}", end=" ")

        print("")

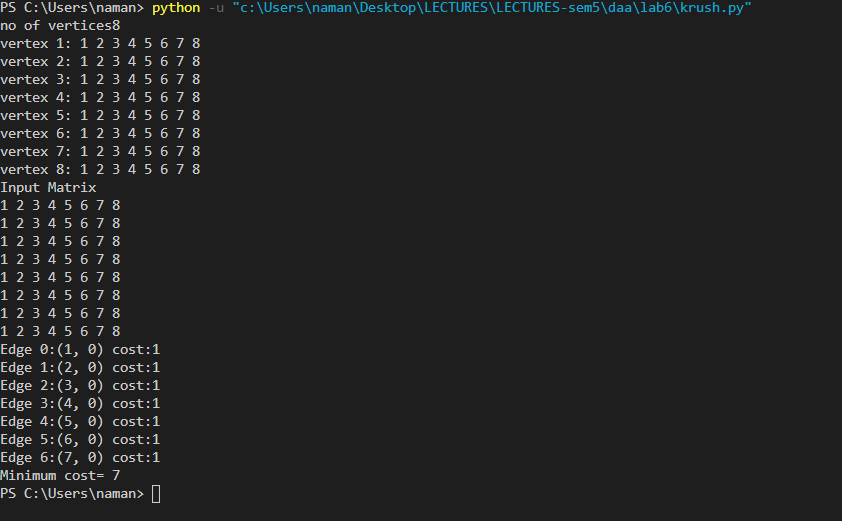
    sel\_matrix = [0 for i in range(v)]

    no\_edges = 0

    # Find set of vertex i

    krushkals(G)

**B.2 Input and Output:**

**B.3 Observations and learning:**

For a disconnected graph, a minimum spanning forest is composed of a minimum spanning tree for each connected component.) It is a greedy algorithm in graph theory as in each step it adds the next lowest-weight edge that will not form a cycle to the minimum spanning forest. Thus, I learned another algorithm other than prim’s for minimum spanning tree.

**B.4 Conclusion:**

Thus I implemented krushkal’s algorithm for minimum spanning tree and compares the answer with prims, got the same answer.

**B.5 Question of Curiosity**

Q.1 Identify & discuss in detail the real life applications Kruskal’s Algorithm.

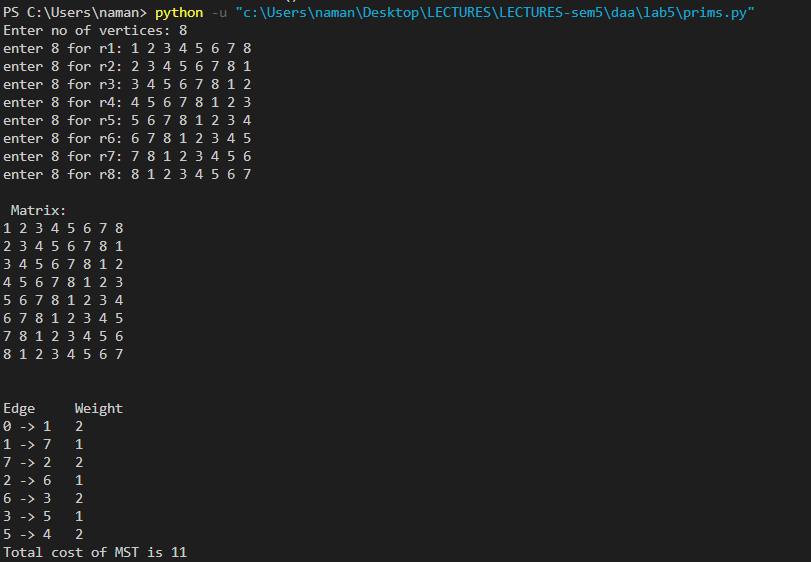
The best use for Kruskal algorithm would be finding out the shortest path for laying down telephone or cable wires. In this way, the telephone or the cable company saves huge amount on the cost of wires and at the same time, the redundancy of path from which information travels decreases and hence much less noise.

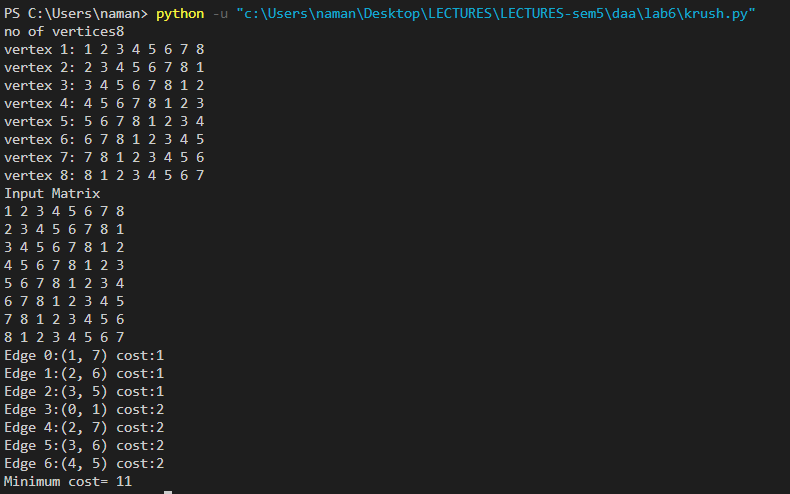
Q.2 Compare the performance of Kruskal’s and Prim’s algorithm.

Kruskal’s algorithm runs faster in sparse graphs.

Prim’s algorithm runs faster in dense graphs.

Compare the output of prims and kruskhals for the same graph,,





Q.3 Discuss the time and space complexity of Kruskal’s algorithm.

Space complexity of O(n) as we use another matrix to store the output of the same size,

And the time complexity is O(logV), V being the number of vertices.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*