# Algorithms

## Lecture 6
## Minimum Spanning Tree

A. S. M. Sanwar Hosen

**Email:** sanwar@wsu.ac.kr

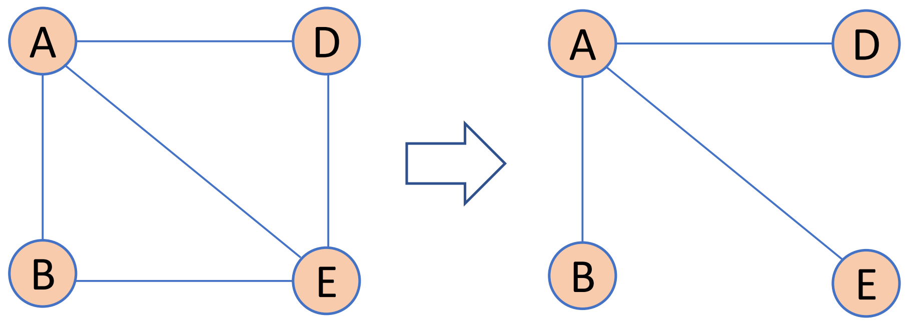
**Date:** 13 April, 2023

## Minimum Spanning Tree

- **Spanning Tree:** It is a subgraph of an undirected connected graph, which includes all the vertices of the graph with a minimum possible number of edges. There is always a path from a vertex to any other vertex.

  The maximum number of spanning trees with $n$ vertices is $= n^{n-2}$

  **Example:**

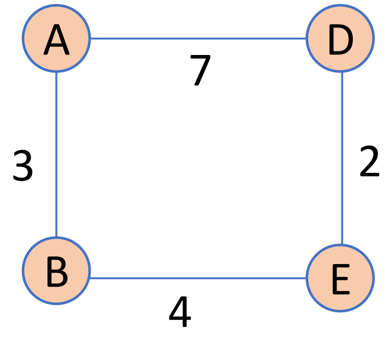


**Undirected Graph** **Spanning Tree**

## Minimum Spanning Tree

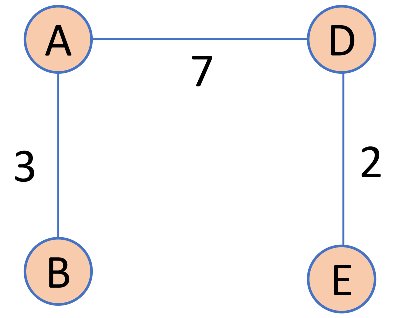
- **Minimum Spanning Tree (MST):** A minimum spanning tree is a spanning tree in which the sum of the weight of the edges is as minimum as possible.

Example:

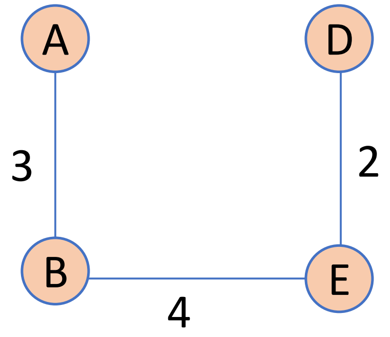


**Weighted Undirected Graph**

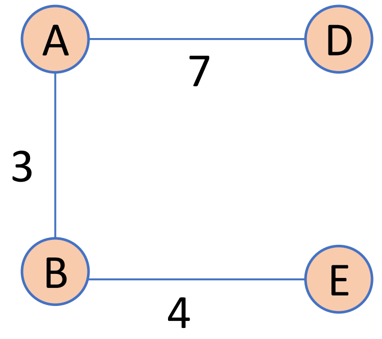
Possible spanning trees of the given graph are:



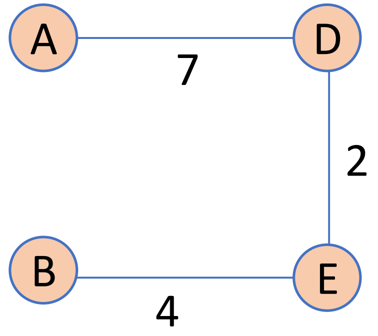
Sum = 12



Sum = 9

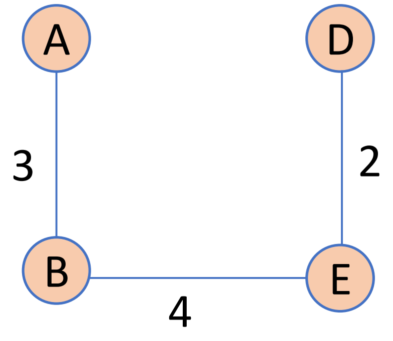


Sum = 14



Sum = 13

The MST of the above subgraphs is:



Sum = 9

## Minimum Spanning Tree

- **Minimum Spanning Tree Algorithms:** Find the minimum spanning tree. Most commonly used algorithms are:

✓ Kruskal’s Algorithm

✓ Prim’s Algorithm

## Minimum Spanning Tree

- **Kruskal's Algorithm:** It creates the spanning tree by adding edges one by one into a growing spanning tree. It uses greedy technique as in each iteration it finds an edge which has minimum weight and add it to the growing spanning tree.

✓ **Basic Steps of Kruskal's Algorithm:**

Step 1: Sort all the edges in the graph in non-decreasing order of their weight.

Step 2: Create a forest of trees where each vertex is a separate tree.

Step 3: Selects the edge with the minimum weight and add it to the spanning tree. If the edge connects two different trees, then merge the two trees into a single tree.

Step 4: Find the minimum weight edge in the set of edges.

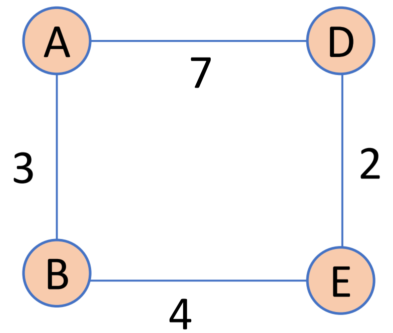
Step 5: Repeat step 3 until there is only one tree left.

✓ **Requirements of Kruskal's Algorithm:** The graph should be weighted and undirected.

## Minimum Spanning Tree

### ❑ Kruskal's Algorithm

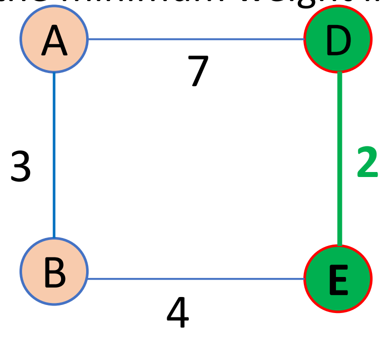
**Example:** Consider the following weighted undirected graph. Find the MST by using the Kruskal's algorithm:



**Weighted Undirected Graph**

**Step 1:**

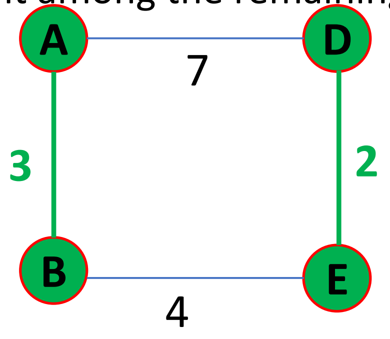
- Start with a weighted undirected graph.
- Choose an edge with the minimum weight in the graph (**D, E**) that does not create a cycle.



**Growing MST**

**Step 2:**

- Choose an edge with the minimum weight among the remaining edges in the graph (**A, B**) that does not create a cycle.

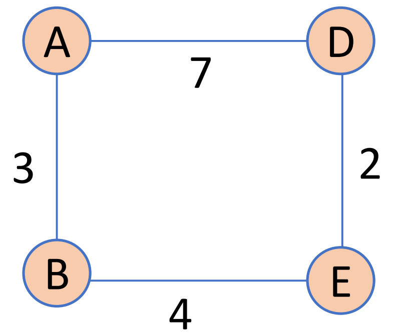


**Growing MST**

## Minimum Spanning Tree

### ❑ Kruskal's Algorithm

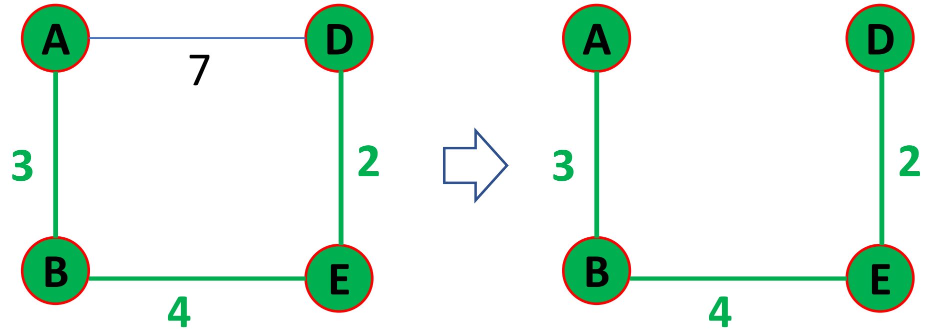
**Example:** Consider the following weighted undirected graph. Find the MST by using the Kruskal's algorithm:



**Weighted Undirected Graph**

**Step 3:**

- Choose an edge with the minimum weight among the remaining edges in the graph (**A, B**) that does not create a cycle.



**Growing MST** **MST**

## Minimum Spanning Tree

❑ **Prim's Algorithm:** It creates the spanning tree by adding vertices one by one into a growing spanning tree. It uses greedy technique as in each iteration it finds an edge with the minimum weight  to the chosen vertex and add it to the growing spanning tree.

✓ **Basic Steps of Prim's Algorithm:**

Step 1: Choose a starting vertex from the graph arbitrary.

Step 2: Create a set of visited vertices and add the starting vertex to it.

Step 3: Create a set of edges that connect the visited vertices to the unvisited vertices.

Step 4: Find the minimum weight edge in the set of edges.

Step 5: Add the minimum weight edge to the spanning tree and mark the adjacent vertex as visited.

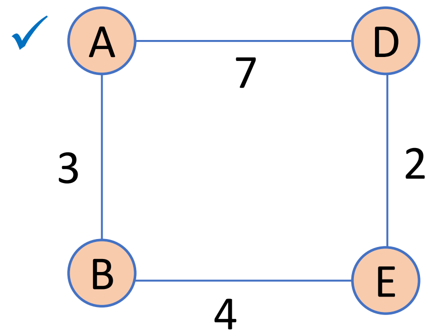
Step 6: Repeat steps 3 to 5 until all the vertices are visited.

✓ **Requirements of Prim's Algorithm:** The graph should be weighted and undirected.

## Minimum Spanning Tree

### ❑ Prim's Algorithm

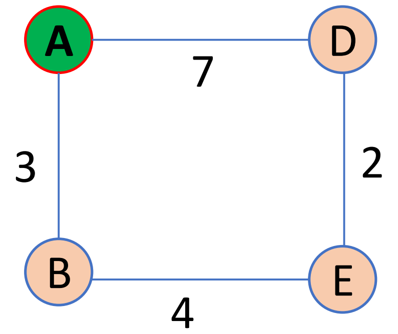
**Example:** Consider the following weighted undirected graph. Find the MST by using the Prim's algorithm:



**Weighted Undirected Graph**

**Step 1:**

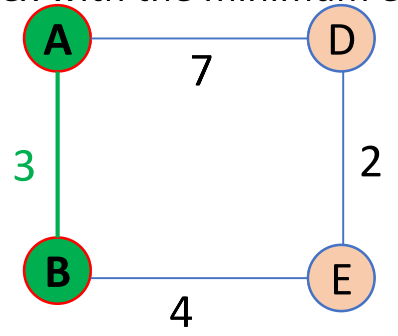
- Start with a weighted undirected graph and an arbitrary vertex (**A**).
- Mark it as visited and add it to the growing MST.



**Growing MST**

**Step 2:**

- Choose a new vertex (**B**) adjacent to the visited one (**A**) based on the greedy approach (selects a vertex with the minimum edge) that does not create a cycle.

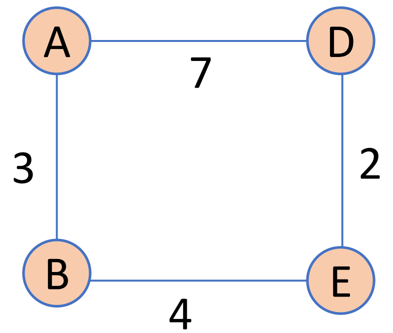


**Growing MST**

## Minimum Spanning Tree

### Prim's Algorithm

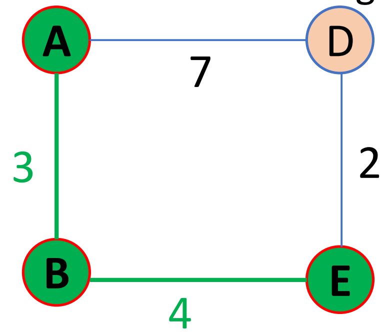
**Example:** Consider the following weighted undirected graph. Find the MST by using the Prim's algorithm:



**Weighted Undirected Graph**

**Step 3:**

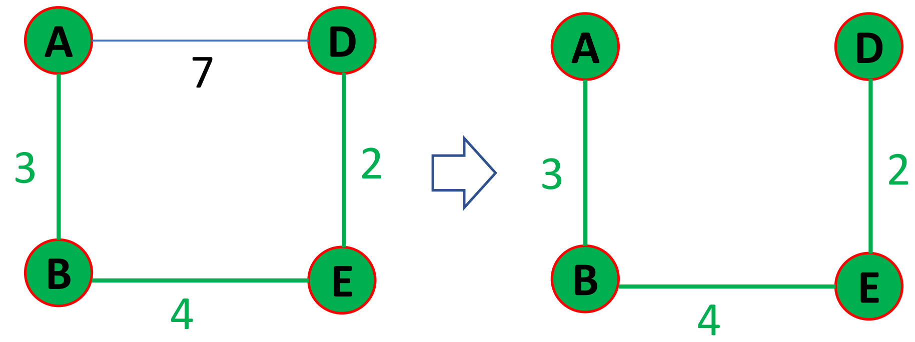
- Choose a new vertex (**E**) adjacent to the visited one (**B**) based on the greedy approach (selects a vertex with the minimum edge) that does not create a cycle.
-



**Growing MST**

**Step 4:**

- Choose a new vertex (**D**) adjacent to the visited one (**E**) based on the greedy approach (selects a vertex with the minimum edge) that does not create a cycle.



**Growing MST** **MST**

## Minimum Spanning Tree

### ❑ Time and Space Complexity

| Algorithm | Time Complexity | Space Complexity |
|---|---|---|
| Kruskal’s | $O(E \log E) = O(n \log n)$ | $O(V + E) = O(n)$ |
| Prim’s | $O(E \log V) = O(n \log n)$ | $O(V + E) = O(n)$ |

Where, $E$ is the number of edges and $V$ is the number vertices in the graph.

### ❑ Comparison

- ✓ Kruskal’s algorithm is a greedy algorithm that selects edges in increasing order of weight and adds them to the MST if they do not create a cycle
- ✓ Prim’s algorithm is also a greedy algorithm that starts from an arbitrary vertex and grows the MST by selecting the edge with the minimum weight that connects to a vertex already in the MST.
- ✓ Kruskal’s algorithm requires extra memory for sorting the edges and keeping track of which vertices are in which sets.
- ✓ Prim’s algorithm requires extra memory for maintaining the priority queue of edges.
- ✓ Kruskal's algorithm is better suited for graphs with a large number of edges and sparse graphs since it does not depend on the starting vertex.
- ✓ Prim’s algorithm is better suited for graphs with a large number of vertices and dense graphs since it depends on the starting vertex.

## Applications of Minimum Spanning Tree

❑ The common uses of MST are as follows:

✓ **Network Design:** MST can be used to design efficient and cost-effective networks, such as communication networks, transportation networks, and power grids.

✓ **Clustering:** It is used in clustering data points into groups based on their similarity, which is useful in machine learning, data analysis, and pattern recognition.

✓ **Image Segmentation:** It can be used for image segmentation, where it is used to separate an image into different regions based on their similarity.

✓ **Circuit Design:** MST is an efficient technique in designing efficient circuit layouts, where it is used to connect different components in a circuit while minimizing the total length of the wires.

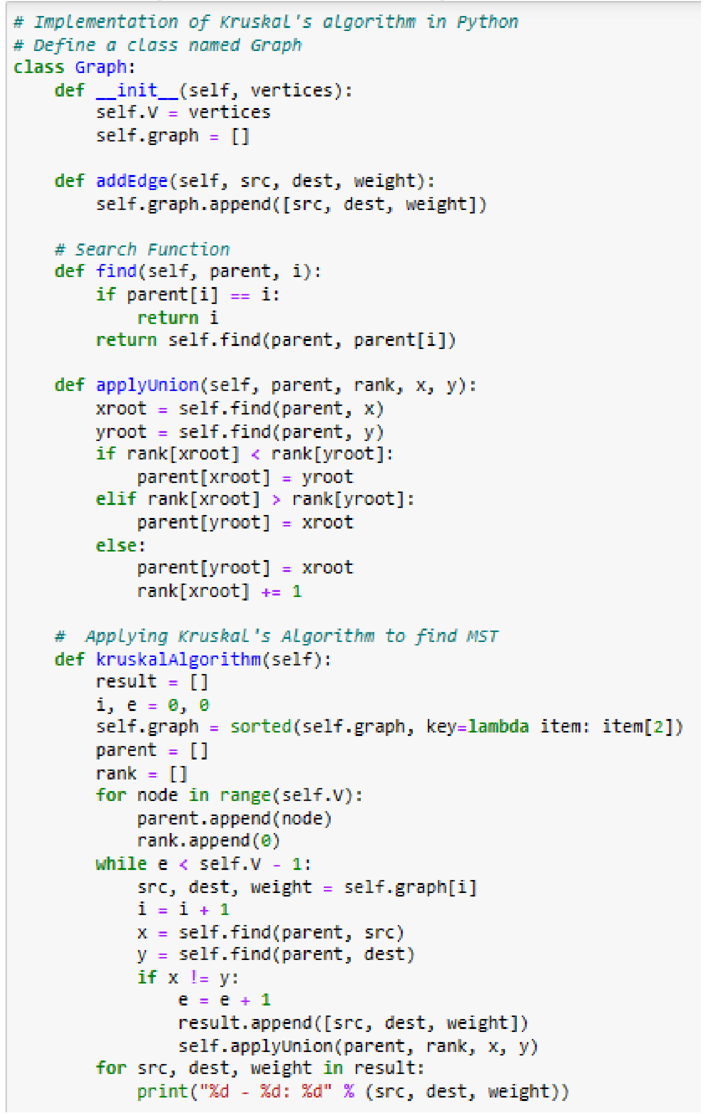
✓ **Approximation Algorithms:** MST can be used to develop approximation algorithms for various optimization problems, such as the traveling salesman problem, facility location problem, and many others.

✓ **Resource Allocation:** To minimize overall cost of a system, MST is used to allocate resources, such as bandwidth or storage, among different entities in a system.

✓ **Phylogenetic Trees:** MST can be used to construct phylogenetic trees, which represent the evolutionary relationships among different species based on their genetic data.

# Minimum Spanning Tree Implementation in Python (1)

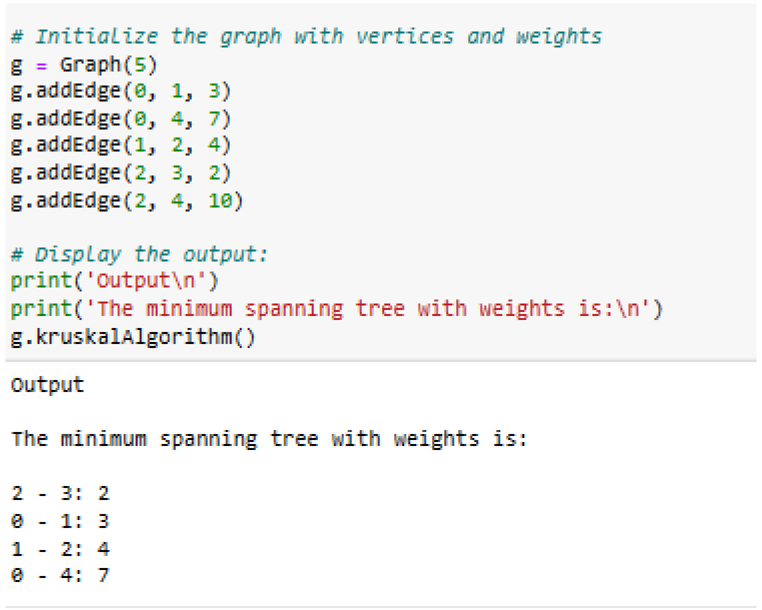
- Finding MST by using Kruskal's Algorithm



```python
# Implementation of Kruskal's algorithm in Python
# Define a class named Graph
class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = []

    def addEdge(self, src, dest, weight):
        self.graph.append([src, dest, weight])

    # Search Function
    def find(self, parent, i):
        if parent[i] == i:
            return i
        return self.find(parent, parent[i])

    def applyUnion(self, parent, rank, x, y):
        xroot = self.find(parent, x)
        yroot = self.find(parent, y)
        if rank[xroot] < rank[yroot]:
            parent[xroot] = yroot
        elif rank[xroot] > rank[yroot]:
            parent[yroot] = xroot
        else:
            parent[yroot] = xroot
            rank[xroot] += 1

    #  Applying Kruskal's Algorithm to find MST
    def kruskalAlgorithm(self):
        result = []
        i, e = 0, 0
        self.graph = sorted(self.graph, key=lambda item: item[2])
        parent = []
        rank = []
        for node in range(self.V):
            parent.append(node)
            rank.append(0)
        while e < self.V - 1:
            src, dest, weight = self.graph[i]
            i = i + 1
            x = self.find(parent, src)
            y = self.find(parent, dest)
            if x != y:
                e = e + 1
                result.append([src, dest, weight])
                self.applyUnion(parent, rank, x, y)
        for src, dest, weight in result:
            print("%d - %d: %d" % (src, dest, weight))
```

**Algorithm 1:** Kruskal's algorithm to find MST

```
KRUSKAL(G):
A = Ø
For each vertex v ∈ G.V:
    MAKE-SET(v)
For each edge (u, v) ∈ G.E ordered by increasing order by weight(u, v):
    if FIND-SET(u) ≠ FIND-SET(v):
    A = A ∪ {(u, v)}
    UNION(u, v)
return A
```

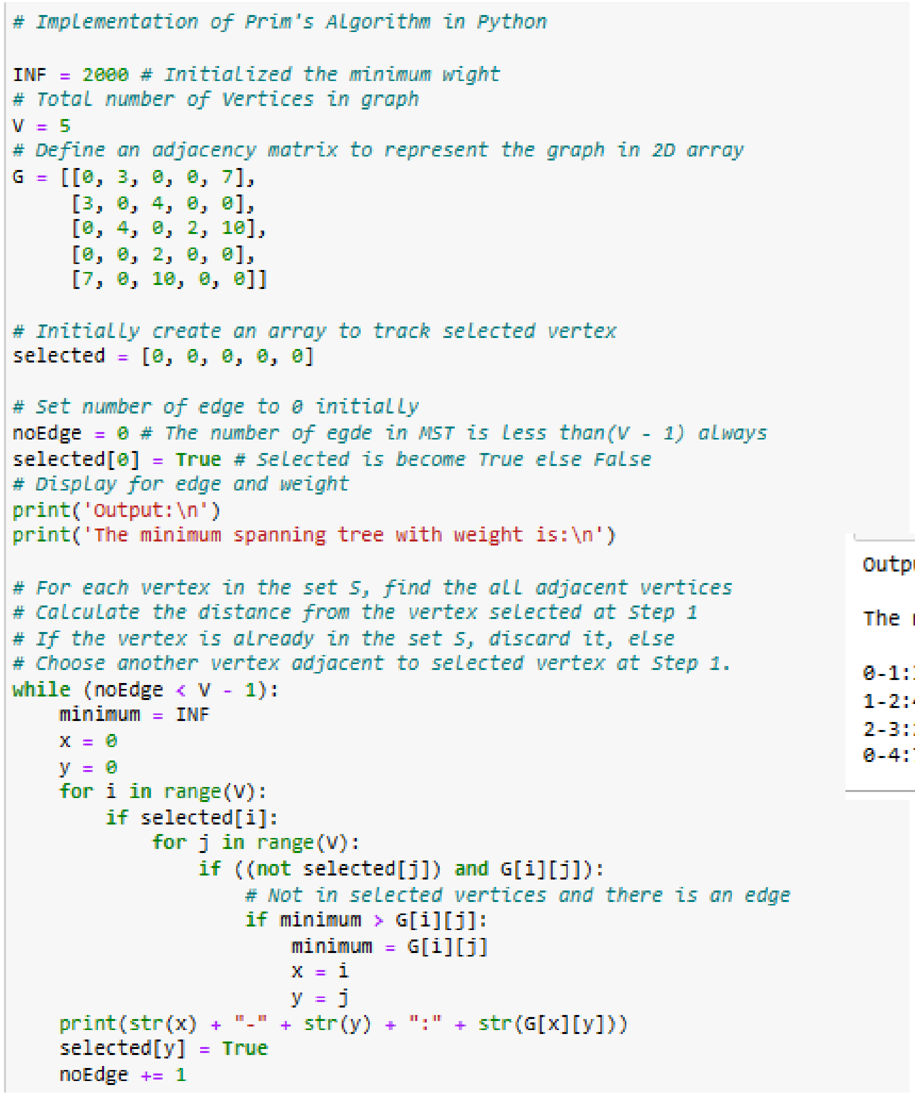


```python
# Initialize the graph with vertices and weights
g = Graph(5)
g.addEdge(0, 1, 3)
g.addEdge(0, 4, 7)
g.addEdge(1, 2, 4)
g.addEdge(2, 3, 2)
g.addEdge(2, 4, 10)

# Display the output:
print('Output\n')
print('The minimum spanning tree with weights is:\n')
g.kruskalAlgorithm()
```

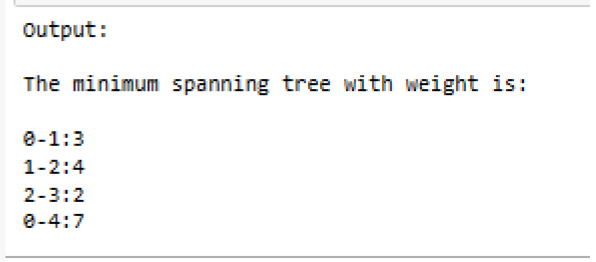
```
Output

The minimum spanning tree with weights is:

2 - 3: 2
0 - 1: 3
1 - 2: 4
0 - 4: 7
```

## Minimum Spanning Tree Implementation in Python (2)

- Finding MST by using Prim's Algorithm



```python
# Implementation of Prim's Algorithm in Python

INF = 2000 # Initialized the minimum wight
# Total number of Vertices in graph
V = 5
# Define an adjacency matrix to represent the graph in 2D array
G = [[0, 3, 0, 0, 7],
     [3, 0, 4, 0, 0],
     [0, 4, 0, 2, 10],
     [0, 0, 2, 0, 0],
     [7, 0, 10, 0, 0]]

# Initially create an array to track selected vertex
selected = [0, 0, 0, 0, 0]

# Set number of edge to 0 initially
noEdge = 0 # The number of egde in MST is less than(V - 1) always
selected[0] = True # Selected is become True else False
# Display for edge and weight
print('Output:\n')
print('The minimum spanning tree with weight is:\n')

# For each vertex in the set S, find the all adjacent vertices
# Calculate the distance from the vertex selected at Step 1
# If the vertex is already in the set S, discard it, else
# Choose another vertex adjacent to selected vertex at Step 1.
while (noEdge < V - 1):
    minimum = INF
    x = 0
    y = 0
    for i in range(V):
        if selected[i]:
            for j in range(V):
                if ((not selected[j]) and G[i][j]):
                    # Not in selected vertices and there is an edge
                    if minimum > G[i][j]:
                        minimum = G[i][j]
                        x = i
                        y = j
    print(str(x) + "-" + str(y) + ":" + str(G[x][y]))
    selected[y] = True
    noEdge += 1
```



```
Output:

The minimum spanning tree with weight is:

0-1:3
1-2:4
2-3:2
0-4:7
```

**Algorithm 2:** Prim's algorithm to find MST

```
T = Ø;
U = { 1 };
while (U ≠ V)
    let (u, v) be the lowest cost edge such that u ∈ U and v ∈ V - U;
    T = T ∪ {(u, v)}
    U = U ∪ {v}
```