This Python code implements Prim's algorithm to find the Minimum Spanning Tree (MST) of a weighted, undirected graph using the NetworkX library. Here's a brief overview of the key components:

1. **Imports**: The code imports the necessary libraries: **networkx** for graph manipulation and **matplotlib.pyplot** for visualization.
2. **Utility Function (minDistance)**: This function finds the vertex with the minimum edge weight that has not yet been included in the MST. It iterates through all vertices and returns the index of the vertex with the smallest distance.
3. **Prim's Algorithm Function (prims)**: This function implements Prim's algorithm:
    - It initializes the distance array (**dist**), parent array (**parent**), and a set to track included vertices (**mstSet**).
    - It sets the distance of the starting vertex to zero and iteratively picks the vertex with the minimum distance that hasn't been included in the MST.
    - It updates the distances of adjacent vertices and records the parent of each vertex in the MST.
    - Finally, it draws the edges of the MST in red.
4. **Graph Creation Function (CreateGraph)**: This function reads a weighted adjacency matrix from an input file (**input.txt**) and constructs a graph using NetworkX. It adds edges with their corresponding weights to the graph.
5. **Graph Drawing Function (DrawGraph)**: This function visualizes the graph using Matplotlib. It displays the nodes, edges, and their weights.
6. **Main Function**: The script's entry point creates the graph, draws it, applies Prim's algorithm to find the MST, and then displays the resulting graph with the MST highlighted.

Usage

To use this code, you need to have an **input.txt** file containing the adjacency matrix of the graph. The first line should specify the number of vertices, followed by the rows of the matrix. The code will visualize the graph and highlight the edges of the MST.

Code Defines Description:

1- def minDistance(dist, mstSet, V):

#utility function that returns the minimum egde weight node

#assigning largest numeric value to min

2- def prims(G, pos):

#function that performs prims algorithm on the graph G

# V denotes the number of vertices in G

# dist[i] will hold the minimum weight edge value of node i to be included in MST

# parent[i] will hold the vertex connected to i, in the MST edge

# mstSet[i] will hold true if vertex i is included in the MST

#initially, for every node, dist[] is set to maximum value and mstSet[] is set to False

#starting vertex is itself the root, and hence has no parent

#pick the minimum distance vertex from the set of vertices

#update the vertices adjacent to the picked vertex

#ignore the parent of the starting node

3- def CreateGraph():

#takes input from the file and creates a weighted graph
#Adds egdes along with their weights to the graph

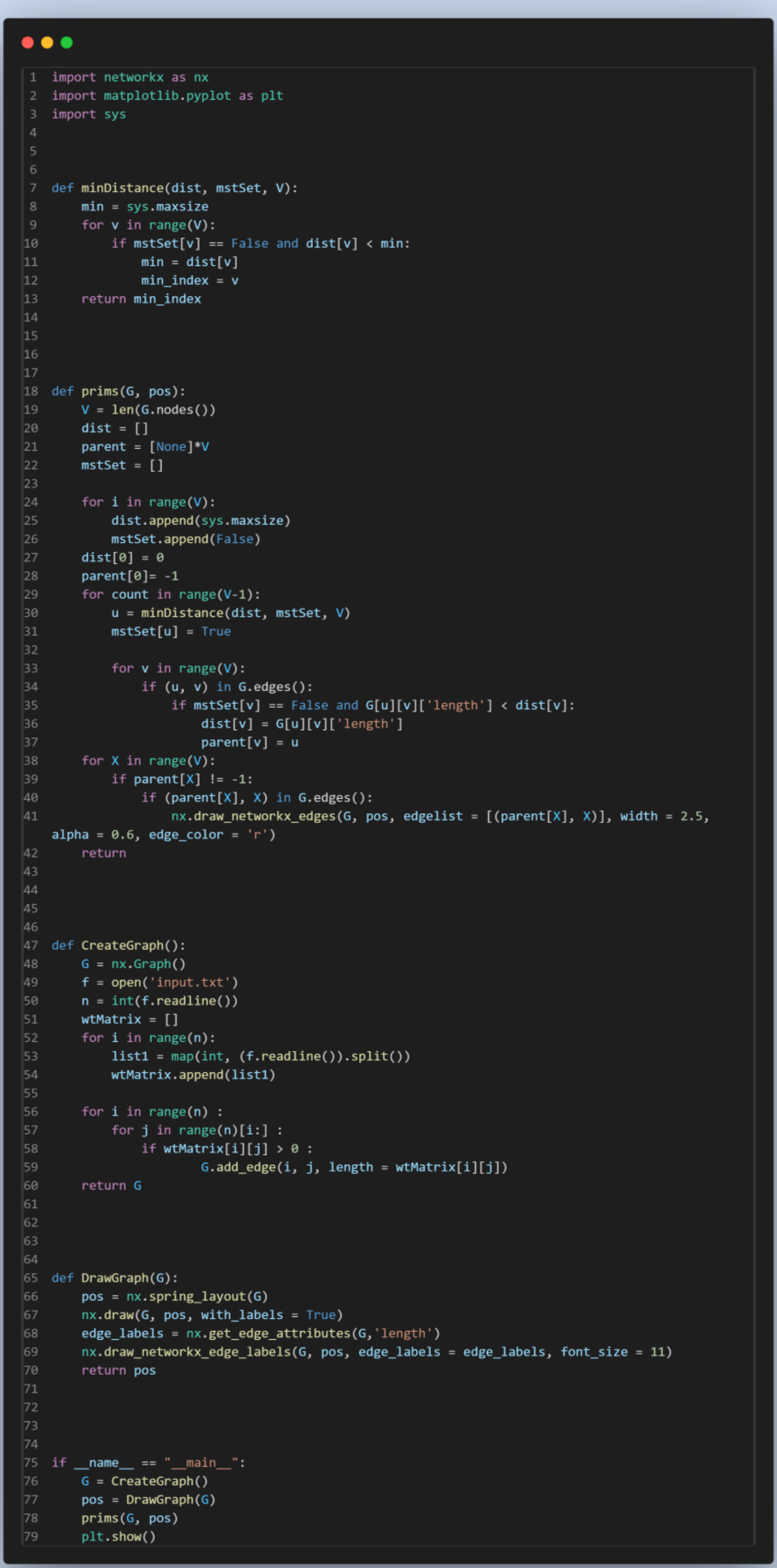
4- def DrawGraph(G):

#draws the graph and displays the weights on the edges

#with_labels=true is to show the node number in the output graph

#prints weight on all the edges

A Snap of The Code



```python
import networkx as nx
import matplotlib.pyplot as plt
import sys



def minDistance(dist, mstSet, V):
    min = sys.maxsize
    for v in range(V):
        if mstSet[v] == False and dist[v] < min:
            min = dist[v]
            min_index = v
    return min_index




def prims(G, pos):
    V = len(G.nodes())
    dist = []
    parent = [None]*V
    mstSet = []

    for i in range(V):
        dist.append(sys.maxsize)
        mstSet.append(False)
    dist[0] = 0
    parent[0]= -1
    for count in range(V-1):
        u = minDistance(dist, mstSet, V)
        mstSet[u] = True

        for v in range(V):
            if (u, v) in G.edges():
                if mstSet[v] == False and G[u][v]['length'] < dist[v]:
                    dist[v] = G[u][v]['length']
                    parent[v] = u
    for X in range(V):
        if parent[X] != -1:
            if (parent[X], X) in G.edges():
                nx.draw_networkx_edges(G, pos, edgelist = [(parent[X], X)], width = 2.5, alpha = 0.6, edge_color = 'r')
    return




def CreateGraph():
    G = nx.Graph()
    f = open('input.txt')
    n = int(f.readline())
    wtMatrix = []
    for i in range(n):
        list1 = map(int, (f.readline()).split())
        wtMatrix.append(list1)

    for i in range(n) :
        for j in range(n)[i:] :
            if wtMatrix[i][j] > 0 :
                    G.add_edge(i, j, length = wtMatrix[i][j])
    return G




def DrawGraph(G):
    pos = nx.spring_layout(G)
    nx.draw(G, pos, with_labels = True)
    edge_labels = nx.get_edge_attributes(G,'length')
    nx.draw_networkx_edge_labels(G, pos, edge_labels = edge_labels, font_size = 11)
    return pos




if __name__ == "__main__":
    G = CreateGraph()
    pos = DrawGraph(G)
    prims(G, pos)
    plt.show()
```