```
from collections import defaultdict

class Graph:
    def __init__(self):
        self.graph = defaultdict(list)

    def add_edge(self, u, v, weight):
        self.graph[u].append((v, weight))
        self.graph[v].append((u, weight))

    def prim_mst(self):
        # Initialize data structures
        parent = {}
        key = {}
        mst_set = set()

        # Initialize key values to infinity for all vertices
        for v in self.graph:
            key[v] = float('inf')

        # Start with the first vertex
        key[self.graph.keys()[0]] = 0
        parent[self.graph.keys()[0]] = None

        # MST has V-1 vertices
        for _ in range(len(self.graph) - 1):
            # Find the vertex with the minimum key value
            min_key = float('inf')
            min_vertex = None
            for v in self.graph:
                if key[v] < min_key and v not in mst_set:
                    min_key = key[v]
                    min_vertex = v

            mst_set.add(min_vertex)
```

```
            # Update key values and parent pointers of adjacent vertices
            for neighbor, weight in self.graph[min_vertex]:
                if neighbor not in mst_set and weight < key[neighbor]:
                    parent[neighbor] = min_vertex
                    key[neighbor] = weight

        # Print the MST
        self.print_mst(parent)

    def print_mst(self, parent):
        print("Edge \tWeight")
        for v in self.graph:
            if parent[v]:
                print(parent[v], "-", v, "\t", self.get_weight(parent[v], v))

    def get_weight(self, u, v):
        for neighbor, weight in self.graph[u]:
            if neighbor == v:
                return weight

# Example usage
g = Graph()
g.add_edge('A', 'B', 2)
g.add_edge('A', 'C', 3)
g.add_edge('B', 'C', 1)
g.add_edge('B', 'D', 4)
g.add_edge('C', 'D', 5)

g.prim_mst()
```