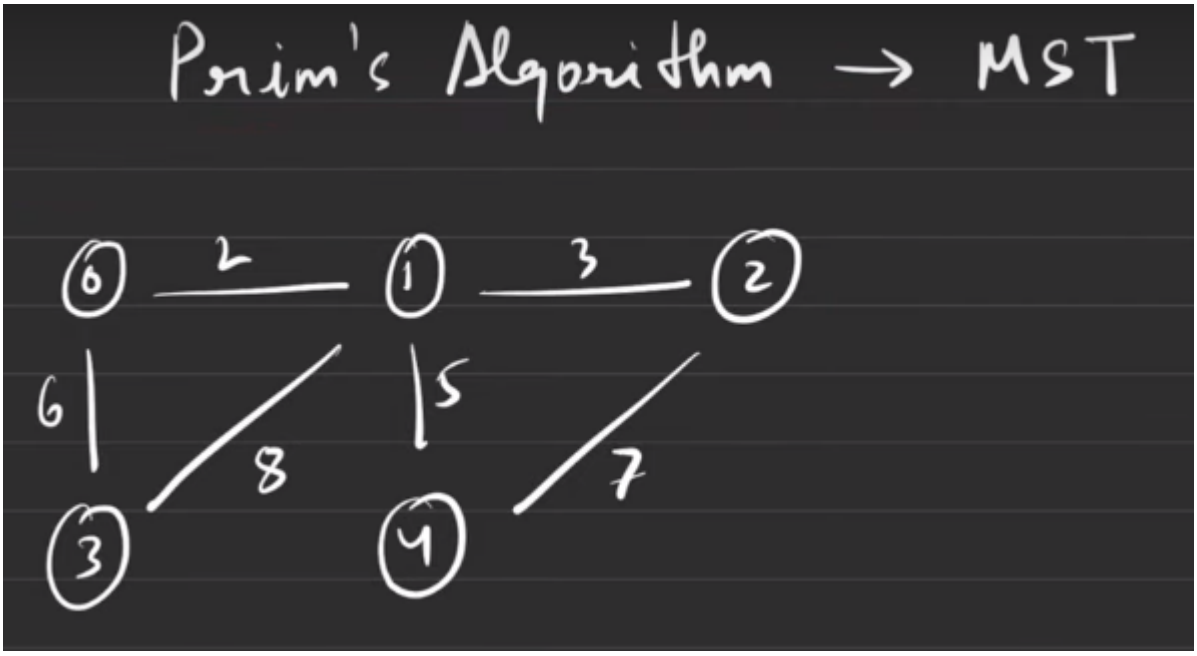
MST Prim's

Sure, I'll provide a dry run for this implementation of Prim's Algorithm on the given graph.

Initial Setup

- Graph:
- 


```
0: [[1, 2], [3, 6]],
1: [[0, 2], [2, 3], [3, 8], [4, 5]],
2: [[1, 3], [4, 7]],
3: [[0, 6], [1, 8]],
4: [[1, 5], [2, 7]],
```

- VISITED_NODE: {0: False, 1: False, 2: False, 3: False, 4: False}
- Priority Queue (q): [(0, 0, -1)] # (weight, node, parent)
- sumMin: 0
- MST: []

Iteration 1:

- Pop (0, 0, -1) from q.

- Current node: 0, Weight: 0, Parent: -1
- Mark node 0 as visited.
- Adjacent list of node 0 : [[1, 2] , [3, 6]]
- Push adjacent nodes to q:
- (weight, node, parent)
    - Push (2, 1, 0)
    - Push (6, 3, 0)

State after iteration 1:

- q: [(2, 1, 0), (6, 3, 0)]
- VISITED_NODE: {0: True, 1: False, 2: False, 3: False, 4: False}
- sumMin: 0
- MST: []

Iteration 2:

- Pop (2, 1, 0) from q.
- Current node: 1, Weight: 2, Parent: 0
- Mark node 1 as visited.
- Add edge (0, 1) to MST and update sumMin: sumMin += 2
- Adjacent list of node 1: 0, 2], [2, 3], [3, 8], [4, 5
    - Remove edge [0, 2] (to prevent re-adding parent node)
- Push adjacent nodes to q:
- (weight node parent)
    - Push (3, 2, 1)
    - Push (8, 3, 1)
    - Push (5, 4, 1)

State after iteration 2:

- q: [(3, 2, 1), (6, 3, 0), (8, 3, 1), (5, 4, 1)]
- VISITED_NODE: {0: True, 1: True, 2: False, 3: False, 4: False}
- sumMin: 2
- MST: [(0, 1)]

Iteration 3:

- Pop (3, 2, 1) from q.
- Current node: 2, Weight: 3, Parent: 1
- Mark node 2 as visited.
- Add edge (1, 2) to MST and update sumMin: sumMin += 3
- Adjacent list of node 2: 1, 3], [4, 7
  - Remove edge [1, 3]
- Push adjacent nodes to q:
  - Push (7, 4, 2)

State after iteration 3:

- q: [(5, 4, 1), (6, 3, 0), (8, 3, 1), (7, 4, 2)]
- VISITED_NODE: {0: True, 1: True, 2: True, 3: False, 4: False}
- sumMin: 5
- MST: [(0, 1), (1, 2)]

Iteration 4:

- Pop (5, 4, 1) from q.
- Current node: 4, Weight: 5, Parent: 1
- Mark node 4 as visited.
- Add edge (1, 4) to MST and update sumMin: sumMin += 5
- Adjacent list of node 4: 1, 5], [2, 7
  - Remove edge [1, 5]
- Push adjacent nodes to q:
  - Push (7, 2, 4) (Not pushed as node 2 is already visited)

State after iteration 4:

- q: [(6, 3, 0), (8, 3, 1), (7, 4, 2)]
- VISITED_NODE: {0: True, 1: True, 2: True, 3: False, 4: True}
- sumMin: 10
- MST: [(0, 1), (1, 2), (1, 4)]

Iteration 5:

- Pop (6, 3, 0) from q.
- Current node: 3, Weight: 6, Parent: 0

- Mark node 3 as visited.
- Add edge (0, 3) to MST and update sumMin: sumMin += 6
- Adjacent list of node 3: 0, 6], [1, 8
  - Remove edge [0, 6]
- Push adjacent nodes to q:
  - Push (8, 1, 3) (Not pushed as node 1 is already visited)

State after iteration 5:

- q: [(7, 4, 2), (8, 3, 1)]
- VISITED_NODE: {0: True, 1: True, 2: True, 3: True, 4: True}
- sumMin: 16
- MST: [(0, 1), (1, 2), (1, 4), (0, 3)]

Iteration 6:

- Pop (7, 4, 2) from q.
- Current node: 4, Weight: 7, Parent: 2
- Node 4 is already visited, so continue.

State after iteration 6:

- q: [(8, 3, 1)]
- VISITED_NODE: {0: True, 1: True, 2: True, 3: True, 4: True}
- sumMin: 16
- MST: [(0, 1), (1, 2), (1, 4), (0, 3)]

Iteration 7:

- Pop (8, 3, 1) from q.
- Current node: 3, Weight: 8, Parent: 1
- Node 3 is already visited, so continue.

State after iteration 7:

- q: []
- VISITED_NODE: {0: True, 1: True, 2: True, 3: True, 4: True}
- sumMin: 16
- MST: [(0, 1), (1, 2), (1, 4), (0, 3)]

Final Result:

- MST: [(0, 1), (1, 2), (1, 4), (0, 3)]
- Minimum Sum: 16

```
import heapq

Graph = {
    0: [[1, 2], [3, 6]],
    1: [[0, 2], [2, 3], [3, 8], [4, 5]],
    2: [[1, 3], [4, 7]],
    3: [[0, 6], [1, 8]],
    4: [[1, 5], [2, 7]],
}
VISITED_NODE = {
    0: False,
    1: False,
    2: False,
    3: False,
    4: False,
}

q = []
# heapq.heappush(q, (wt, Node, Parent))
heapq.heappush(q, (0, 0, -1))
sumMin = 0
MST = []
while len(q) != 0:
    crr = heapq.heappop(q)
    if VISITED_NODE[crr[1]] is True:
        continue
    VISITED_NODE[crr[1]] = True
    adjList = Graph[crr[1]]
    if crr[2] != -1:
        adjList.remove([crr[2], crr[0]])
        sumMin += crr[0]
        MST.append((crr[2], crr[1]))

    for cN, cW in adjList:
        if VISITED_NODE[cN] is not True:
            heapq.heappush(q, (cW, cN, crr[1]))
```

```
print(MST)
print(sumMin)
```