1. 205. Given a graph with weights and a potential Minimum Spanning Tree (MST), verify if the given MST is unique. If it is not unique, provide another possible MST.

Program:

```
from collections import defaultdict
import heapq

# Helper function to find the root of a set
def find(parent, x):
    if parent[x] != x:
        parent[x] = find(parent, parent[x])
    return parent[x]

# Helper function to union two sets
def union(parent, rank, x, y):
    rootX = find(parent, x)
    rootY = find(parent, y)
    if rootX != rootY:
        if rank[rootX] > rank[rootY]:
            parent[rootY] = rootX
        elif rank[rootX] < rank[rootY]:
            parent[rootX] = rootY
        else:
            parent[rootY] = rootX
            rank[rootX] += 1

# Kruskal's algorithm to find the MST
def kruskal(n, edges):
    parent = list(range(n))
    rank = [0] * n
    mst_weight = 0
    mst_edges = []

    # Sort edges by weight
    edges.sort(key=lambda x: x[2])

    for u, v, weight in edges:
        if find(parent, u) != find(parent, v):
            union(parent, rank, u, v)
            mst_weight += weight
            mst_edges.append((u, v, weight))

    return mst_weight, mst_edges
```

```
# Function to check if the given MST is unique
def is_unique_mst(n, edges, given_mst):
    # Calculate MST using Kruskal's algorithm
    mst_weight, mst_edges = kruskal(n, edges)

    # Check if the weight of the given MST is the same
    given_mst_weight = sum(weight for u, v, weight in given_mst)

    if given_mst_weight != mst_weight:
        return False, None

    # Check if the given MST is indeed the MST
    if set((min(u, v), max(u, v)) for u, v, _ in given_mst) != set((min(u, v),
max(u, v)) for u, v, _ in mst_edges):
        return False, None

    # Find all MSTs and check for another one
    all_edges = set((min(u, v), max(u, v)) for u, v, _ in edges)
    mst_edges_set = set((min(u, v), max(u, v)) for u, v, _ in mst_edges)

    # Check if there is another MST
    alternative_mst_edges = []
    for (u, v, weight) in edges:
        if (min(u, v), max(u, v)) not in mst_edges_set:
            alternative_mst_edges.append((u, v, weight))

    # Run Kruskal's again on the remaining edges to check for another MST
    _, alt_mst_edges = kruskal(n, alternative_mst_edges + given_mst)
    alternative_mst_edges_set = set((min(u, v), max(u, v)) for u, v, _ in
alt_mst_edges)

    if alternative_mst_edges_set != set(mst_edges_set):
        return False, alt_mst_edges
    else:
        return True, None

# Test Case 1
n = 4
edges = [(0, 1, 10), (0, 2, 6), (0, 3, 5), (1, 3, 15), (2, 3, 4)]
given_mst = [(2, 3, 4), (0, 3, 5), (0, 1, 10)]

is_unique, alternative_mst = is_unique_mst(n, edges, given_mst)
```

```
print(f"Is the given MST unique? {is_unique}")
if not is_unique and alternative_mst:
    print(f"Another possible MST: {alternative_mst}")
```

out put:

```
Is the given MST unique? True


=== Code Execution Successful ===
```

time complexity:O(mlogn)