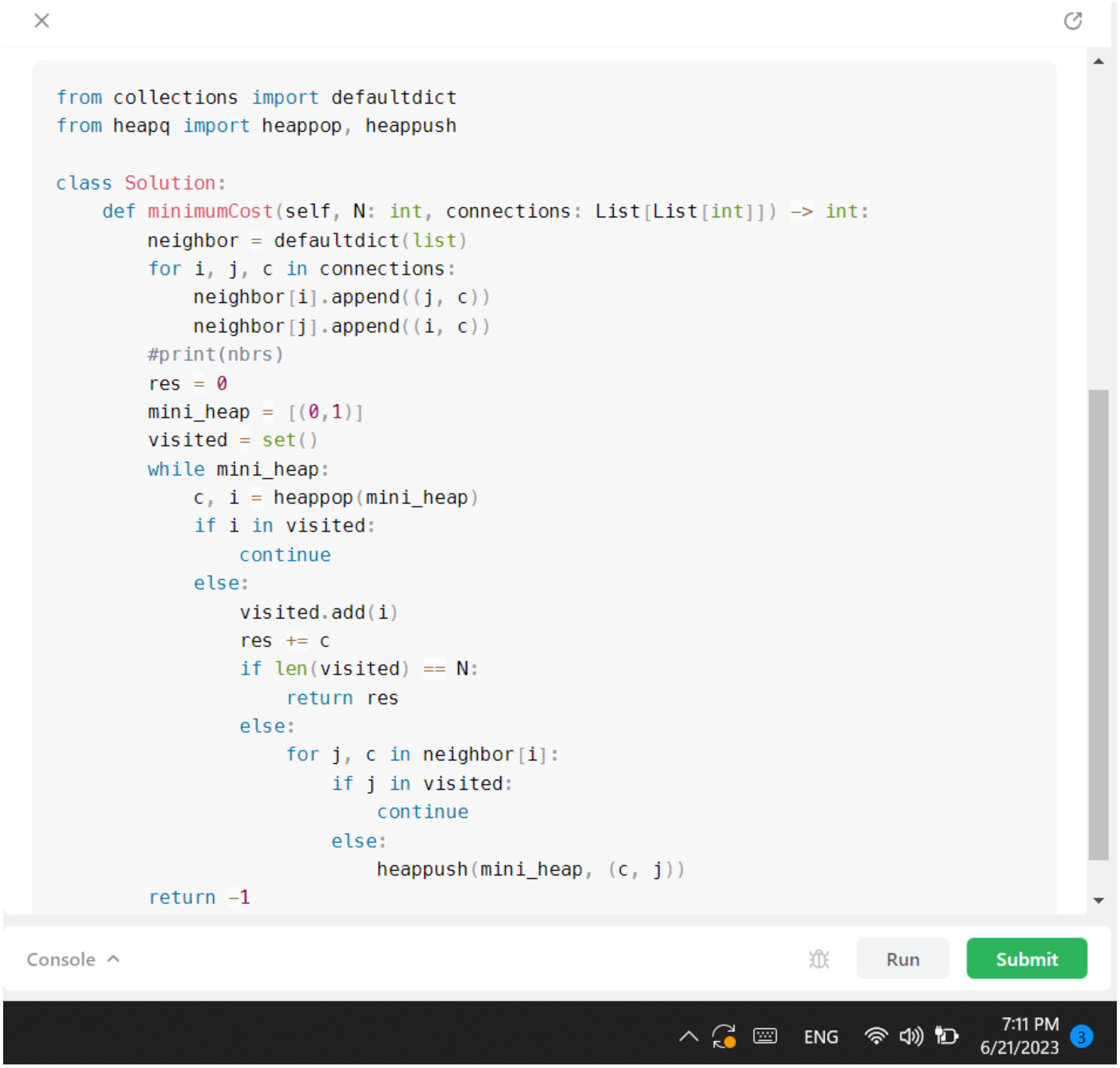
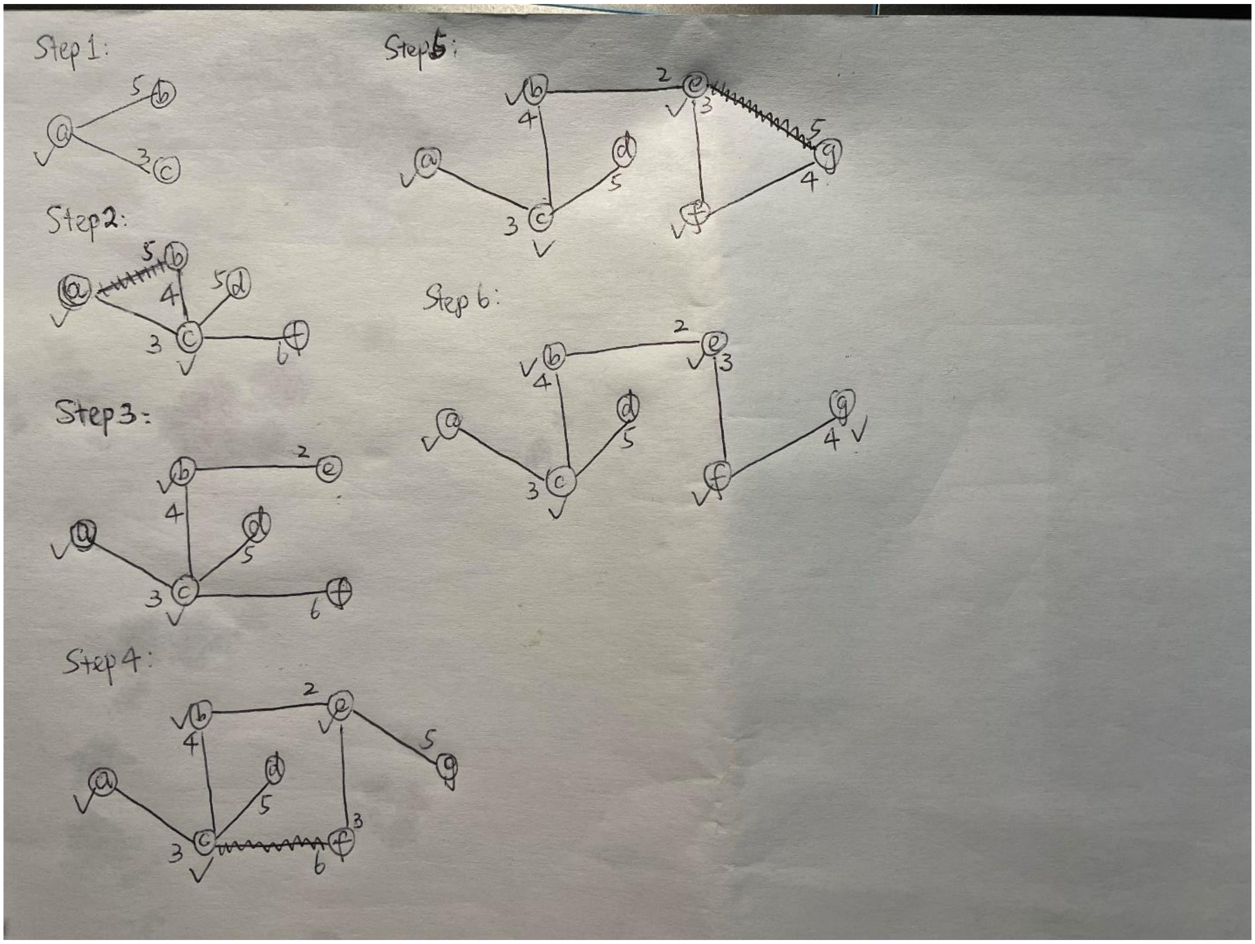
Step 1: Study 1135 Connecting Cities With Minimum Cost (local copy) – Medium

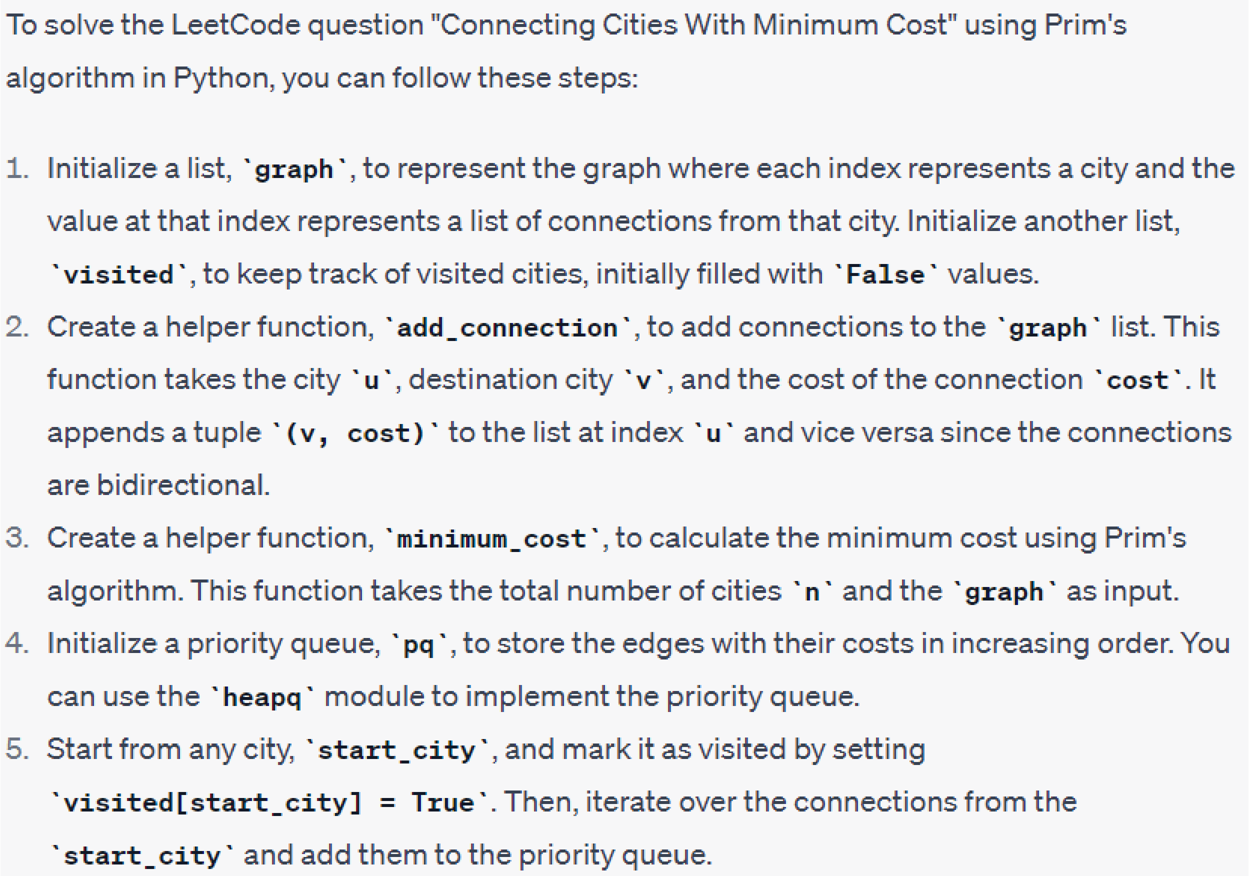


```
from collections import defaultdict
from heapq import heappop, heappush

class Solution:
    def minimumCost(self, N: int, connections: List[List[int]]) -> int:
        neighbor = defaultdict(list)
        for i, j, c in connections:
            neighbor[i].append((j, c))
            neighbor[j].append((i, c))
        #print(nbrs)
        res = 0
        mini_heap = [(0,1)]
        visited = set()
        while mini_heap:
            c, i = heappop(mini_heap)
            if i in visited:
                continue
            else:
                visited.add(i)
                res += c
                if len(visited) == N:
                    return res
                else:
                    for j, c in neighbor[i]:
                        if j in visited:
                            continue
                        else:
                            heappush(mini_heap, (c, j))
        return -1
```

Step 2: Use Prim approach to find Minimum Spanning Tree of this question

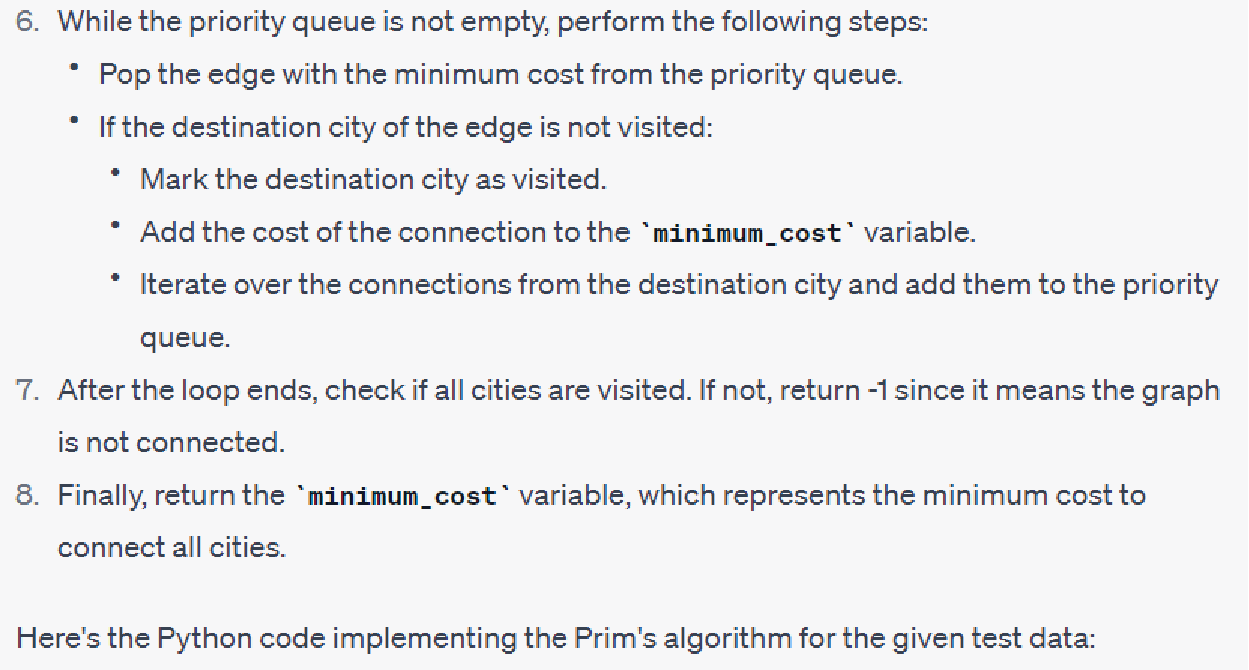


Step 3: Ask ChatGPT: "How to use Prim approach to implement a Python code to solve the LeetCode question: '1135 Connecting Cities With Minimum Cost', with the test data: 'Input: n = 3, connections = [[1, 2, 5],[1,3,6],[2,3,1]], Output: 6'"



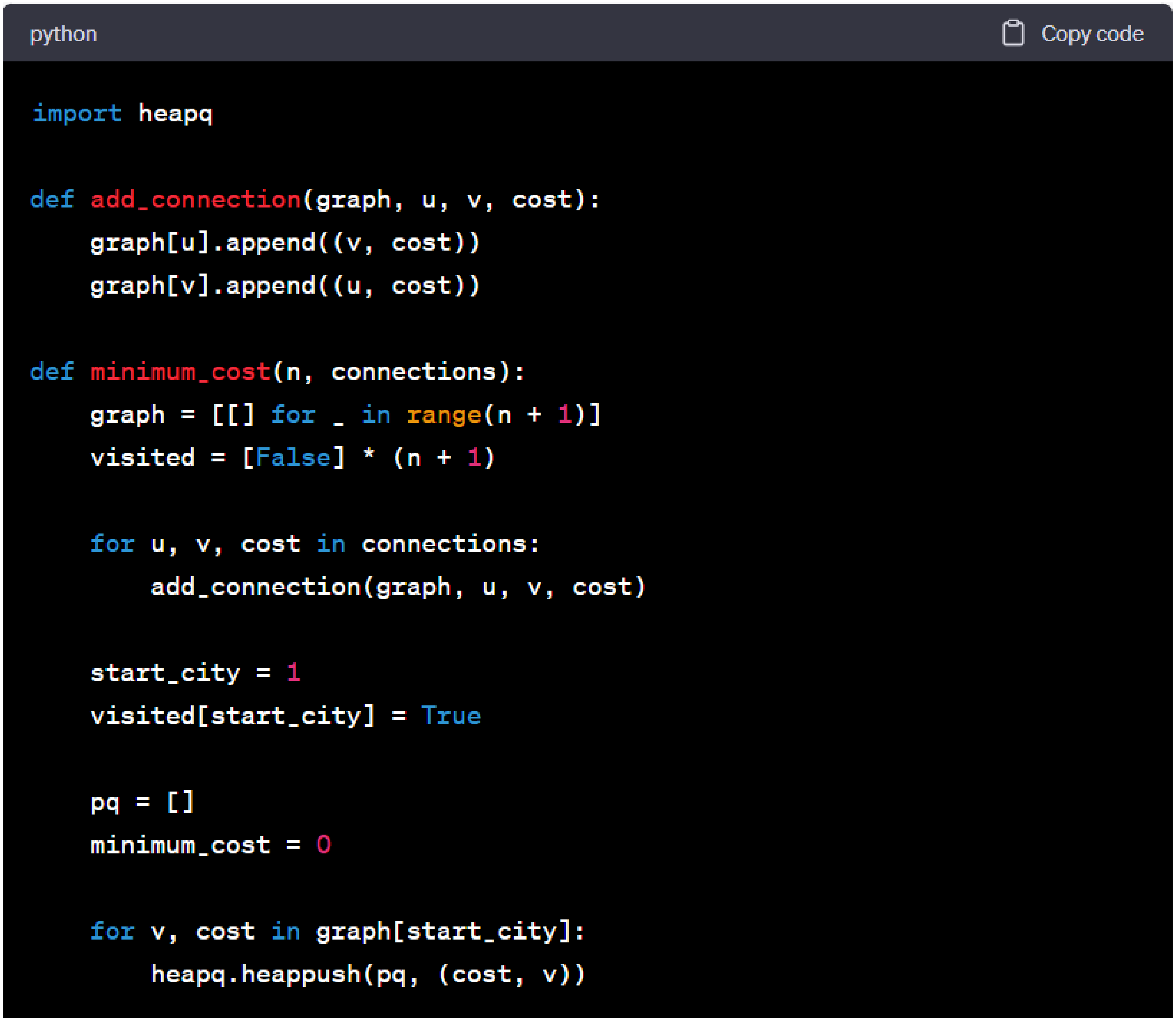
To solve the LeetCode question "Connecting Cities With Minimum Cost" using Prim's algorithm in Python, you can follow these steps:

1. Initialize a list, `graph`, to represent the graph where each index represents a city and the value at that index represents a list of connections from that city. Initialize another list, `visited`, to keep track of visited cities, initially filled with `False` values.
2. Create a helper function, `add_connection`, to add connections to the `graph` list. This function takes the city `u`, destination city `v`, and the cost of the connection `cost`. It appends a tuple `(v, cost)` to the list at index `u` and vice versa since the connections are bidirectional.
3. Create a helper function, `minimum_cost`, to calculate the minimum cost using Prim's algorithm. This function takes the total number of cities `n` and the `graph` as input.
4. Initialize a priority queue, `pq`, to store the edges with their costs in increasing order. You can use the `heapq` module to implement the priority queue.
5. Start from any city, `start_city`, and mark it as visited by setting `visited[start_city] = True`. Then, iterate over the connections from the `start_city` and add them to the priority queue.

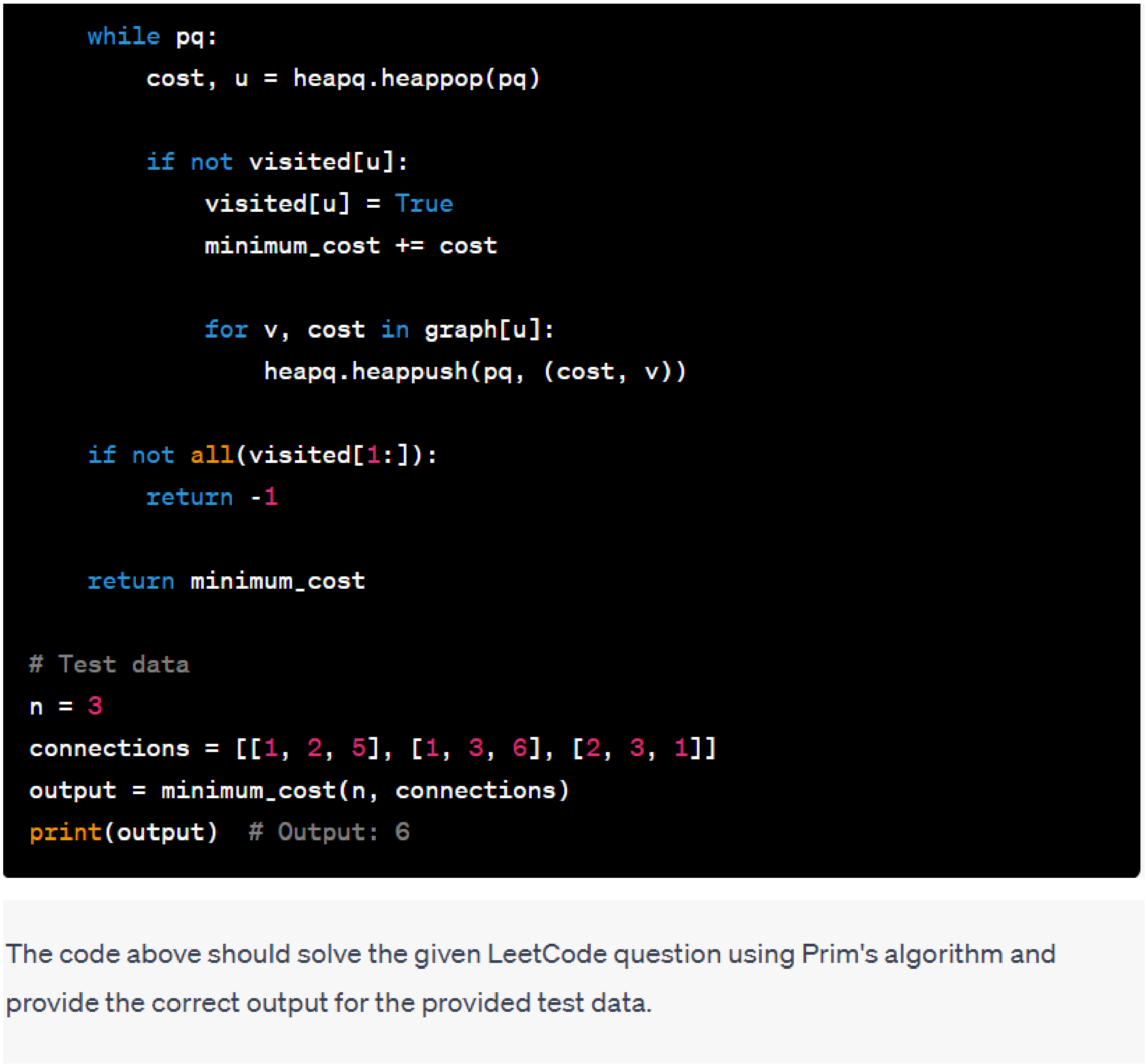


6. While the priority queue is not empty, perform the following steps:
   - Pop the edge with the minimum cost from the priority queue.
   - If the destination city of the edge is not visited:
     - Mark the destination city as visited.
     - Add the cost of the connection to the `minimum_cost` variable.
     - Iterate over the connections from the destination city and add them to the priority queue.
7. After the loop ends, check if all cities are visited. If not, return -1 since it means the graph is not connected.
8. Finally, return the `minimum_cost` variable, which represents the minimum cost to connect all cities.

Here's the Python code implementing the Prim's algorithm for the given test data:



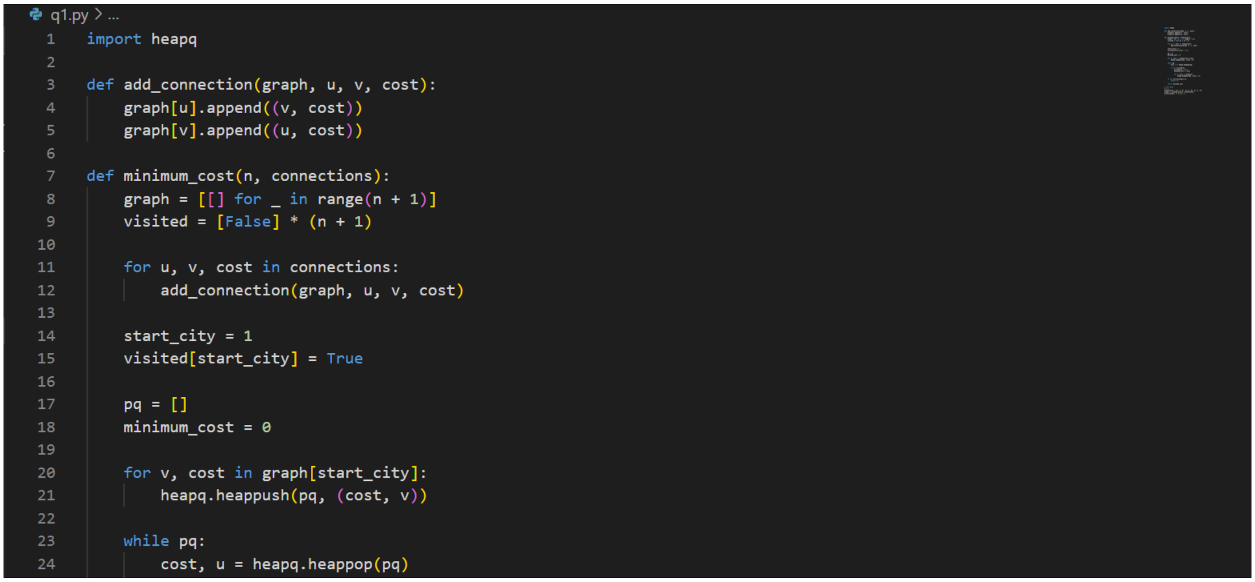
```python
import heapq

def add_connection(graph, u, v, cost):
    graph[u].append((v, cost))
    graph[v].append((u, cost))

def minimum_cost(n, connections):
    graph = [[] for _ in range(n + 1)]
    visited = [False] * (n + 1)

    for u, v, cost in connections:
        add_connection(graph, u, v, cost)

    start_city = 1
    visited[start_city] = True

    pq = []
    minimum_cost = 0

    for v, cost in graph[start_city]:
        heapq.heappush(pq, (cost, v))
```



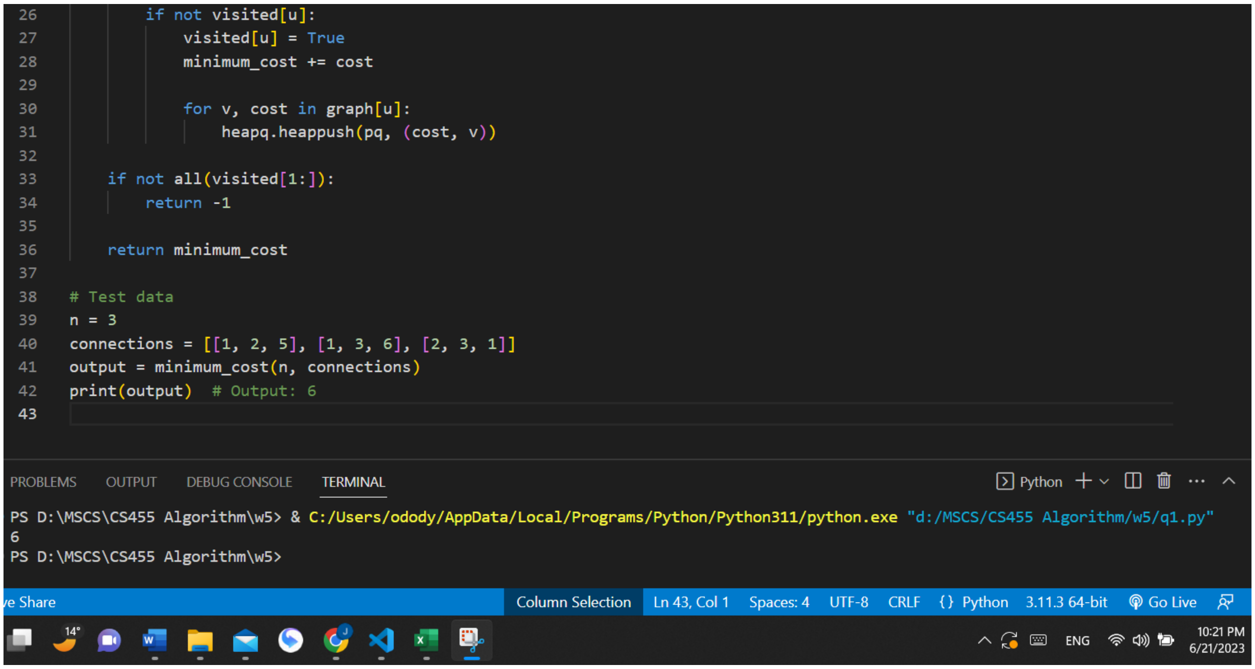
```
    while pq:
        cost, u = heapq.heappop(pq)

        if not visited[u]:
            visited[u] = True
            minimum_cost += cost

            for v, cost in graph[u]:
                heapq.heappush(pq, (cost, v))

    if not all(visited[1:]):
        return -1

    return minimum_cost

# Test data
n = 3
connections = [[1, 2, 5], [1, 3, 6], [2, 3, 1]]
output = minimum_cost(n, connections)
print(output)  # Output: 6
```

The code above should solve the given LeetCode question using Prim's algorithm and provide the correct output for the provided test data.

Step 4: Implement the Python code created by ChatGPT



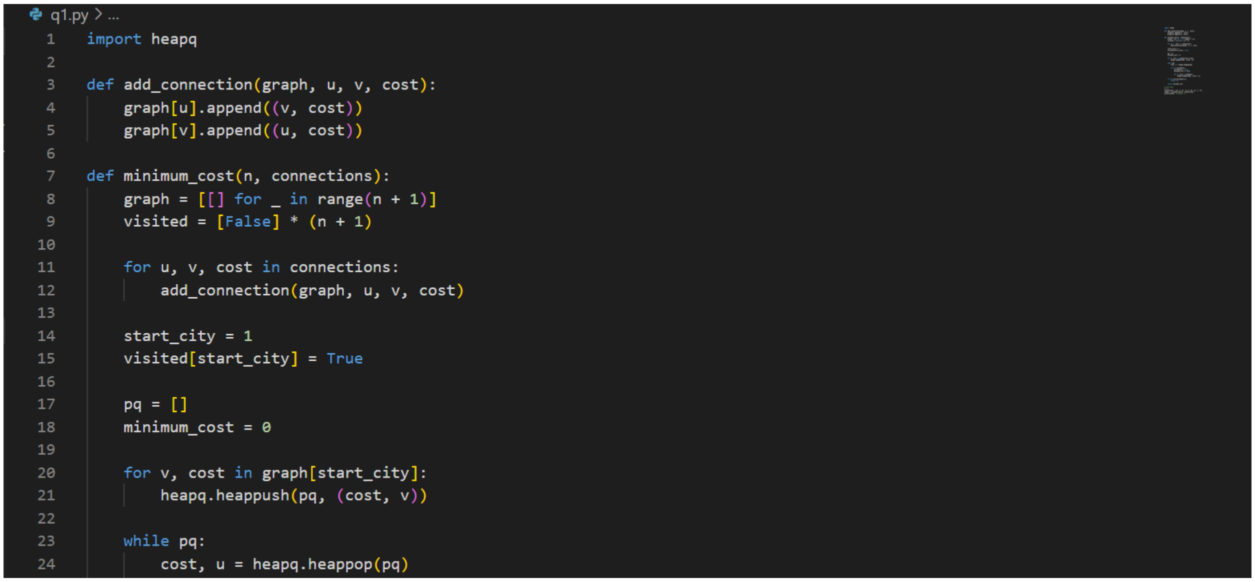
```python
import heapq

def add_connection(graph, u, v, cost):
    graph[u].append((v, cost))
    graph[v].append((u, cost))

def minimum_cost(n, connections):
    graph = [[] for _ in range(n + 1)]
    visited = [False] * (n + 1)

    for u, v, cost in connections:
        add_connection(graph, u, v, cost)

    start_city = 1
    visited[start_city] = True

    pq = []
    minimum_cost = 0

    for v, cost in graph[start_city]:
        heapq.heappush(pq, (cost, v))

    while pq:
        cost, u = heapq.heappop(pq)
```



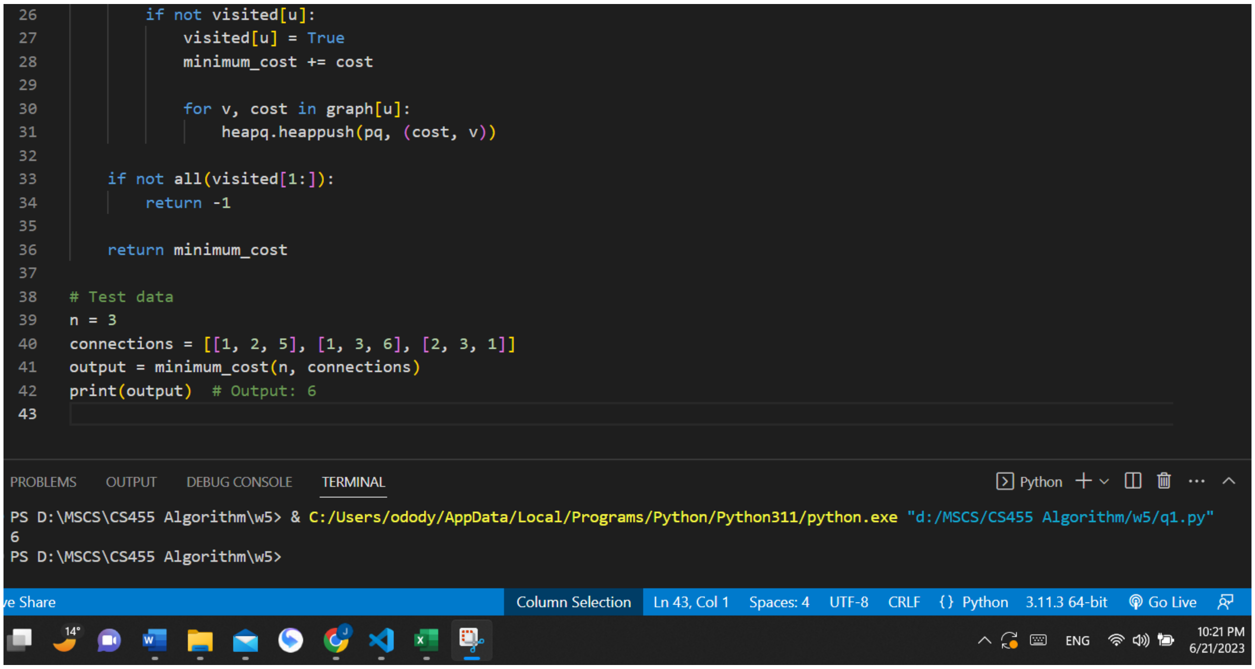
```python
        if not visited[u]:
            visited[u] = True
            minimum_cost += cost

            for v, cost in graph[u]:
                heapq.heappush(pq, (cost, v))

    if not all(visited[1:]):
        return -1

    return minimum_cost

# Test data
n = 3
connections = [[1, 2, 5], [1, 3, 6], [2, 3, 1]]
output = minimum_cost(n, connections)
print(output)  # Output: 6

```

```
PS D:\MSCS\CS455 Algorithm\w5> & C:/Users/odody/AppData/Local/Programs/Python/Python311/python.exe "d:/MSCS/CS455 Algorithm/w5/q1.py"
6
PS D:\MSCS\CS455 Algorithm\w5>
```

Step 5: Test the Python code with all the test cases provided by 1135 Connecting Cities With Minimum Cost (local copy)

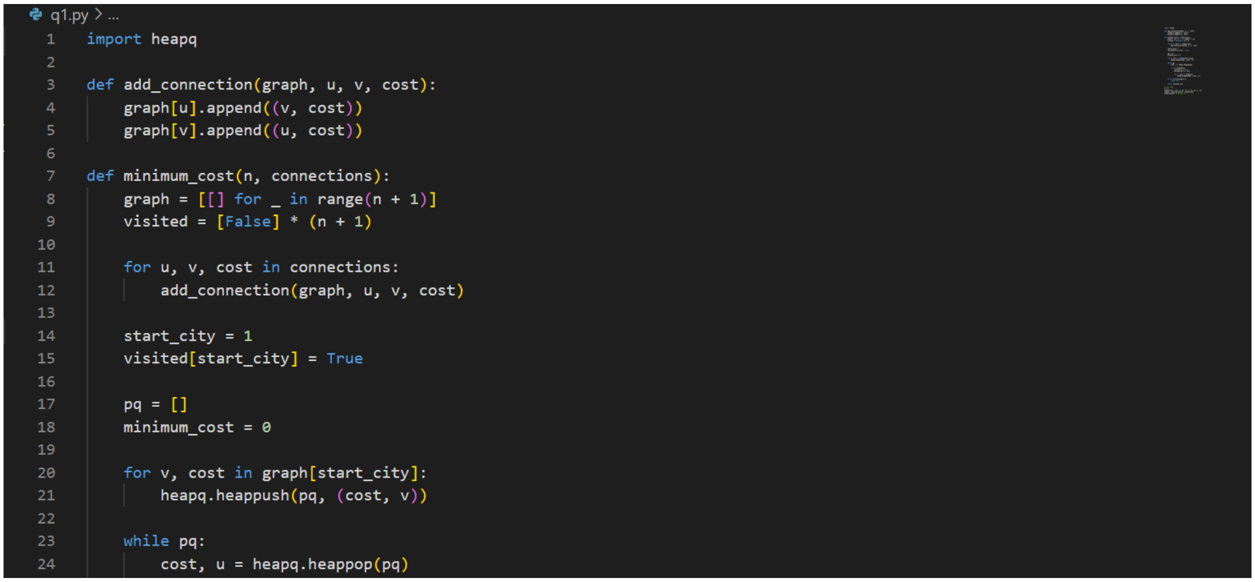


```python
import heapq

def add_connection(graph, u, v, cost):
    graph[u].append((v, cost))
    graph[v].append((u, cost))

def minimum_cost(n, connections):
    graph = [[] for _ in range(n + 1)]
    visited = [False] * (n + 1)

    for u, v, cost in connections:
        add_connection(graph, u, v, cost)

    start_city = 1
    visited[start_city] = True

    pq = []
    minimum_cost = 0

    for v, cost in graph[start_city]:
        heapq.heappush(pq, (cost, v))

    while pq:
        cost, u = heapq.heappop(pq)
```

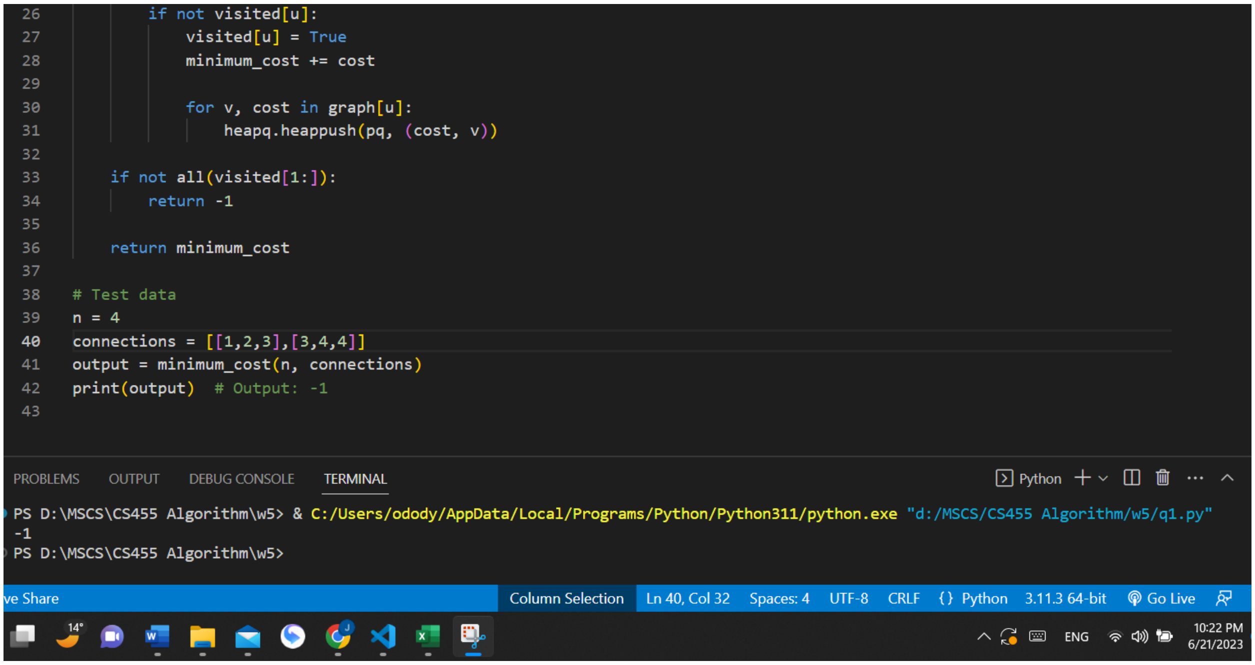


```python
        if not visited[u]:
            visited[u] = True
            minimum_cost += cost

            for v, cost in graph[u]:
                heapq.heappush(pq, (cost, v))

    if not all(visited[1:]):
        return -1

    return minimum_cost

# Test data
n = 3
connections = [[1, 2, 5], [1, 3, 6], [2, 3, 1]]
output = minimum_cost(n, connections)
print(output)  # Output: 6

```

```
PS D:\MSCS\CS455 Algorithm\w5> & C:/Users/ododу/AppData/Local/Programs/Python/Python311/python.exe "d:/MSCS/CS455 Algorithm/w5/q1.py"
6
PS D:\MSCS\CS455 Algorithm\w5>
```



```python
import heapq

def add_connection(graph, u, v, cost):
    graph[u].append((v, cost))
    graph[v].append((u, cost))

def minimum_cost(n, connections):
    graph = [[] for _ in range(n + 1)]
    visited = [False] * (n + 1)

    for u, v, cost in connections:
        add_connection(graph, u, v, cost)

    start_city = 1
    visited[start_city] = True

    pq = []
    minimum_cost = 0

    for v, cost in graph[start_city]:
        heapq.heappush(pq, (cost, v))

    while pq:
        cost, u = heapq.heappop(pq)
```



```python
        if not visited[u]:
            visited[u] = True
            minimum_cost += cost

            for v, cost in graph[u]:
                heapq.heappush(pq, (cost, v))

    if not all(visited[1:]):
        return -1

    return minimum_cost

# Test data
n = 4
connections = [[1,2,3],[3,4,4]]
output = minimum_cost(n, connections)
print(output)  # Output: -1

```

```
PS D:\MSCS\CS455 Algorithm\w5> & C:/Users/odody/AppData/Local/Programs/Python/Python311/python.exe "d:/MSCS/CS455 Algorithm/w5/q1.py"
-1
PS D:\MSCS\CS455 Algorithm\w5>
```