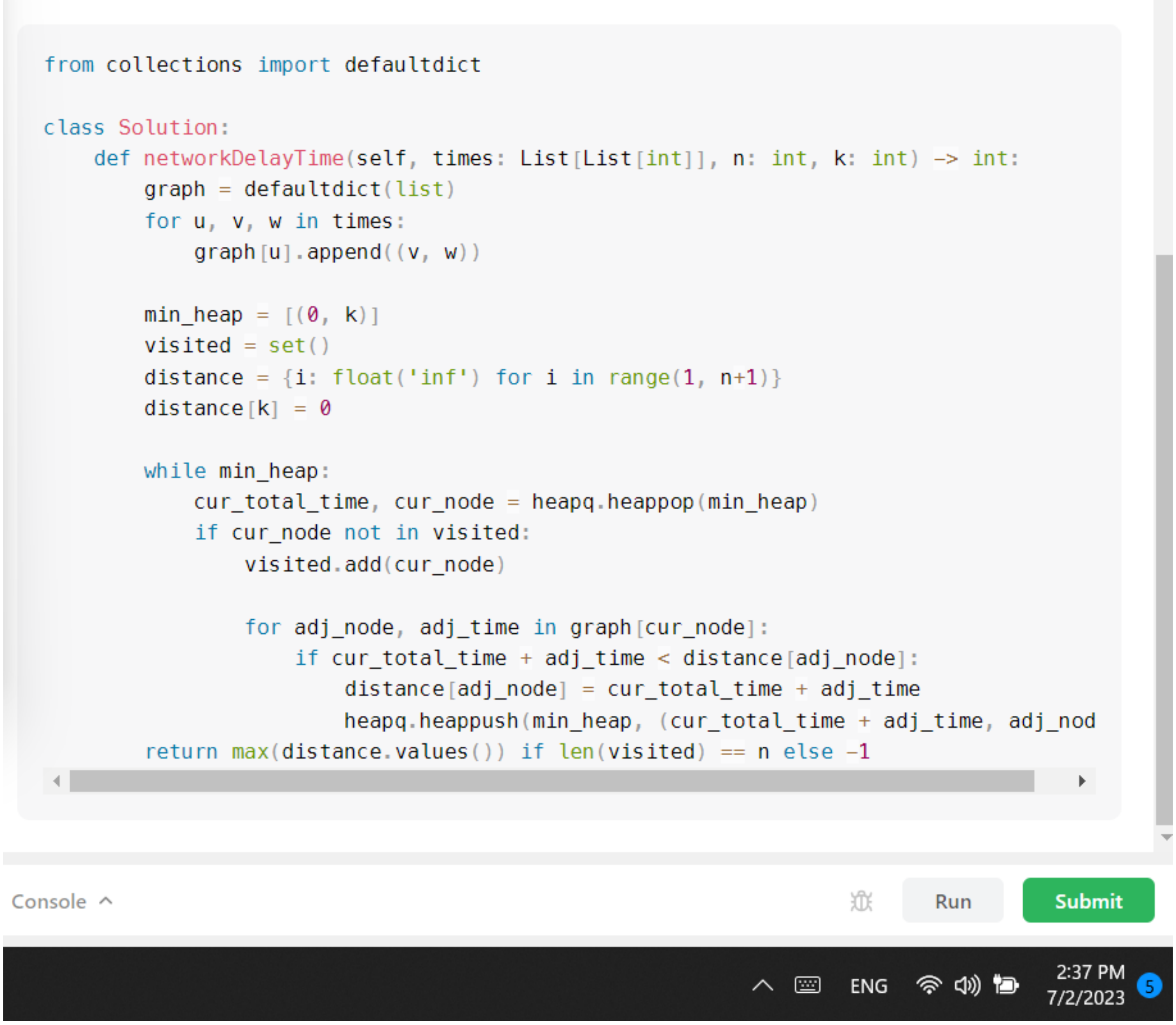
Step 1: Solve this problem manually

Vertex (accumulated path)	**Initial ()**	**Step1 A (A)**	**Step2 B (A, B)**	**Step3 D (A, D)**	**Step4 C (A, B, C)**
	Next Step A	**Next Step B**	**Next Step D**	**Next Step C**	**Next Step E**
A	0	0	~~0~~	~~0~~	~~0~~
B	∞	1	1	~~1~~	~~1~~
C	∞	∞	3	3	3
D	∞	2	2	2	~~2~~
E	∞	∞	∞	5	5

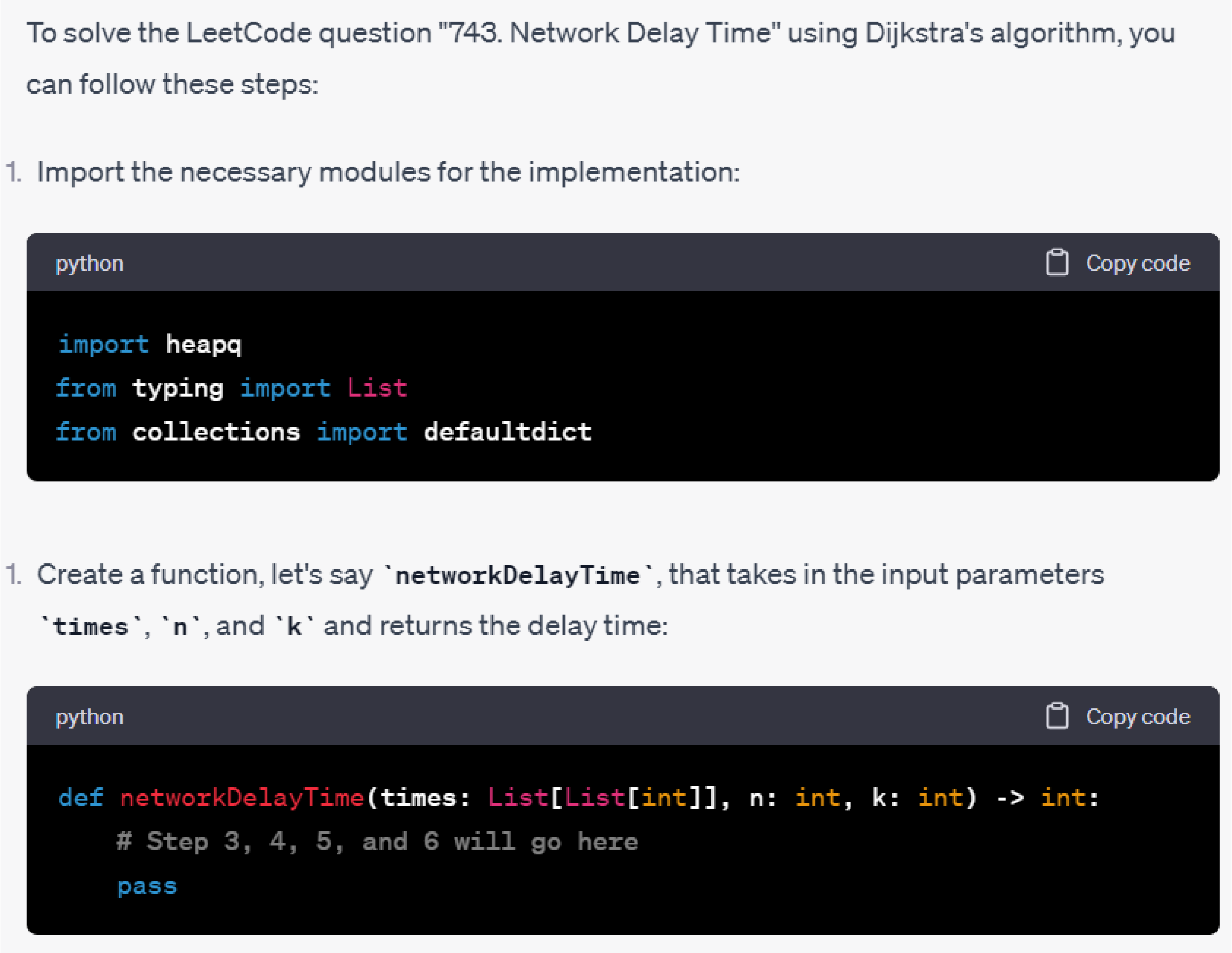
The shortest path is A-B-D-C-E and the minimum time it takes for all the nodes to receive the signal is 5.

Step 2: Study 743. Network Delay Time 50.3 (local copy)- Medium



```
from collections import defaultdict

class Solution:
    def networkDelayTime(self, times: List[List[int]], n: int, k: int) -> int:
        graph = defaultdict(list)
        for u, v, w in times:
            graph[u].append((v, w))

        min_heap = [(0, k)]
        visited = set()
        distance = {i: float('inf') for i in range(1, n+1)}
        distance[k] = 0

        while min_heap:
            cur_total_time, cur_node = heapq.heappop(min_heap)
            if cur_node not in visited:
                visited.add(cur_node)

                for adj_node, adj_time in graph[cur_node]:
                    if cur_total_time + adj_time < distance[adj_node]:
                        distance[adj_node] = cur_total_time + adj_time
                        heapq.heappush(min_heap, (cur_total_time + adj_time, adj_nod
        return max(distance.values()) if len(visited) == n else -1
```

Step 3: Ask ChatGPT: "How to use Dijkstra's Algorithm to implement a Python code to solve the LeetCode question: '743. Network Delay Time', with the test data: 'Input: times = [[2,1,1],[2,3,1],[3,4,1]], n = 4, k = 2; Output: 2'"



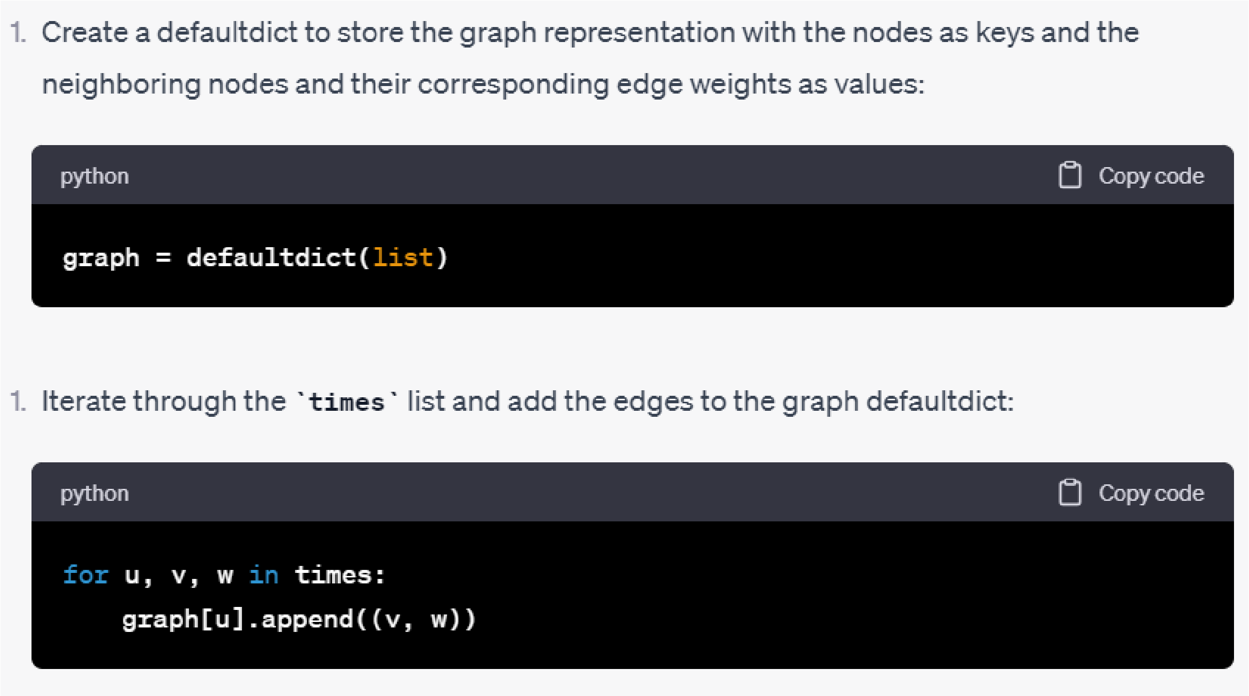
To solve the LeetCode question "743. Network Delay Time" using Dijkstra's algorithm, you can follow these steps:

1. Import the necessary modules for the implementation:

```python
import heapq
from typing import List
from collections import defaultdict
```

1. Create a function, let's say `networkDelayTime`, that takes in the input parameters `times`, `n`, and `k` and returns the delay time:

```python
def networkDelayTime(times: List[List[int]], n: int, k: int) -> int:
    # Step 3, 4, 5, and 6 will go here
    pass
```

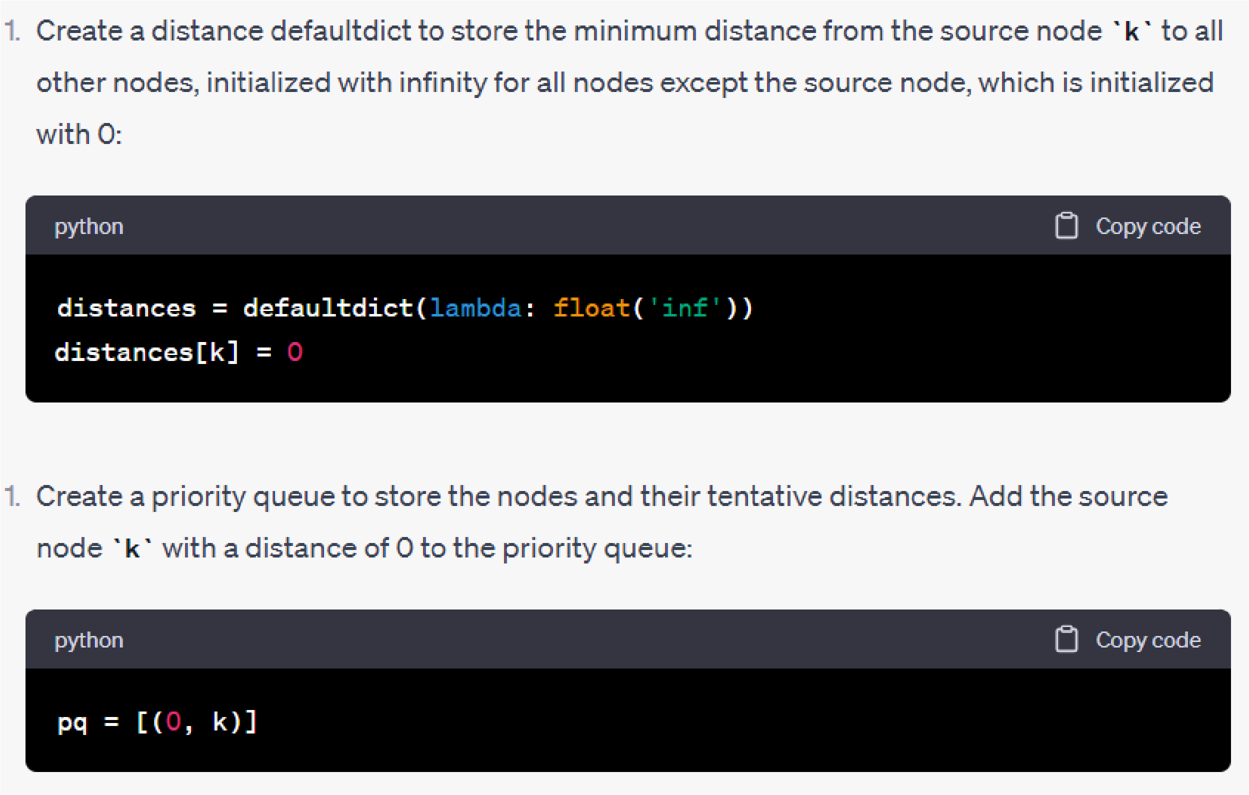


1. Create a defaultdict to store the graph representation with the nodes as keys and the neighboring nodes and their corresponding edge weights as values:

```python
graph = defaultdict(list)
```

1. Iterate through the `times` list and add the edges to the graph defaultdict:

```python
for u, v, w in times:
    graph[u].append((v, w))
```

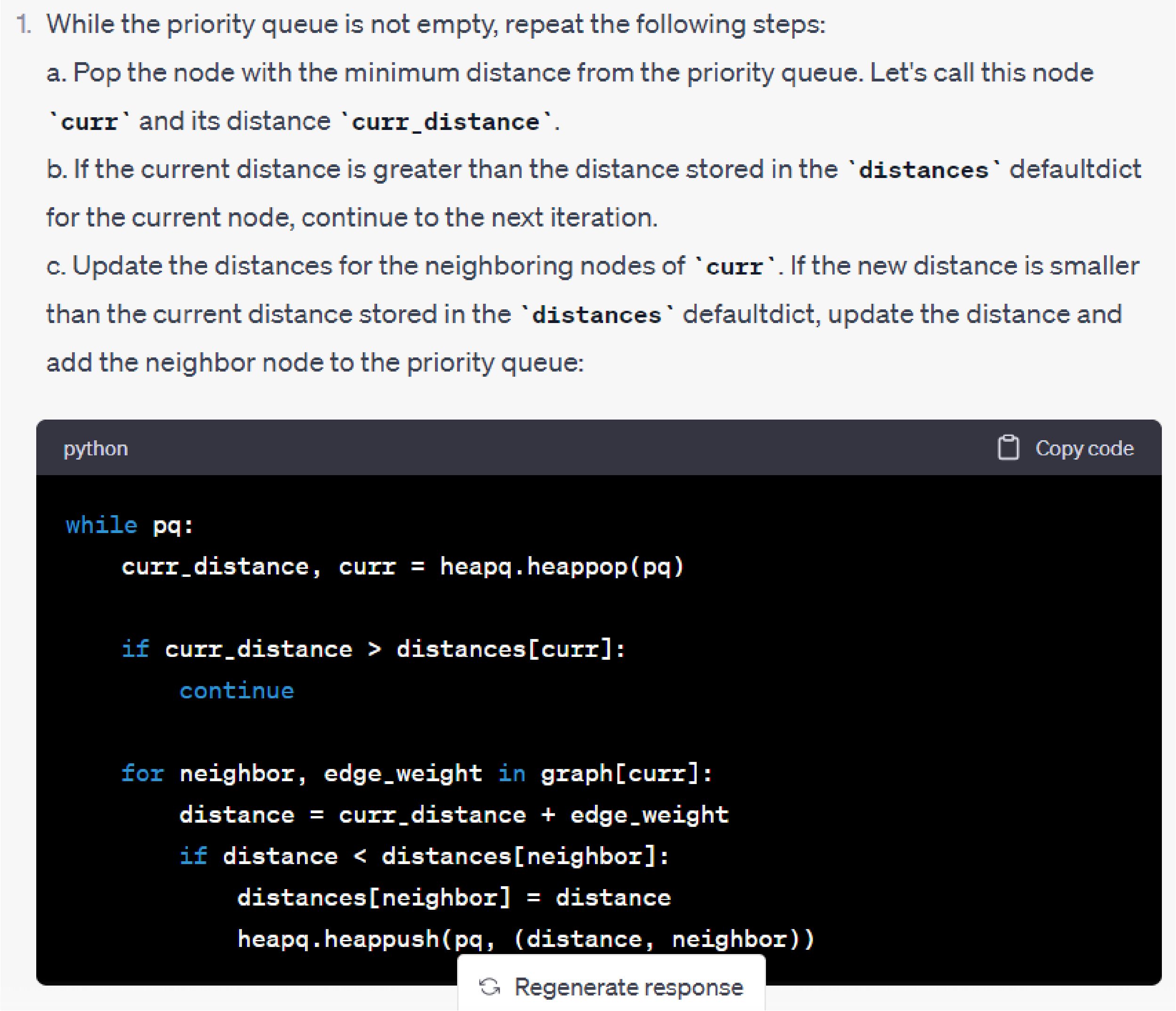


1. Create a distance defaultdict to store the minimum distance from the source node `k` to all other nodes, initialized with infinity for all nodes except the source node, which is initialized with 0:

```python
distances = defaultdict(lambda: float('inf'))
distances[k] = 0
```

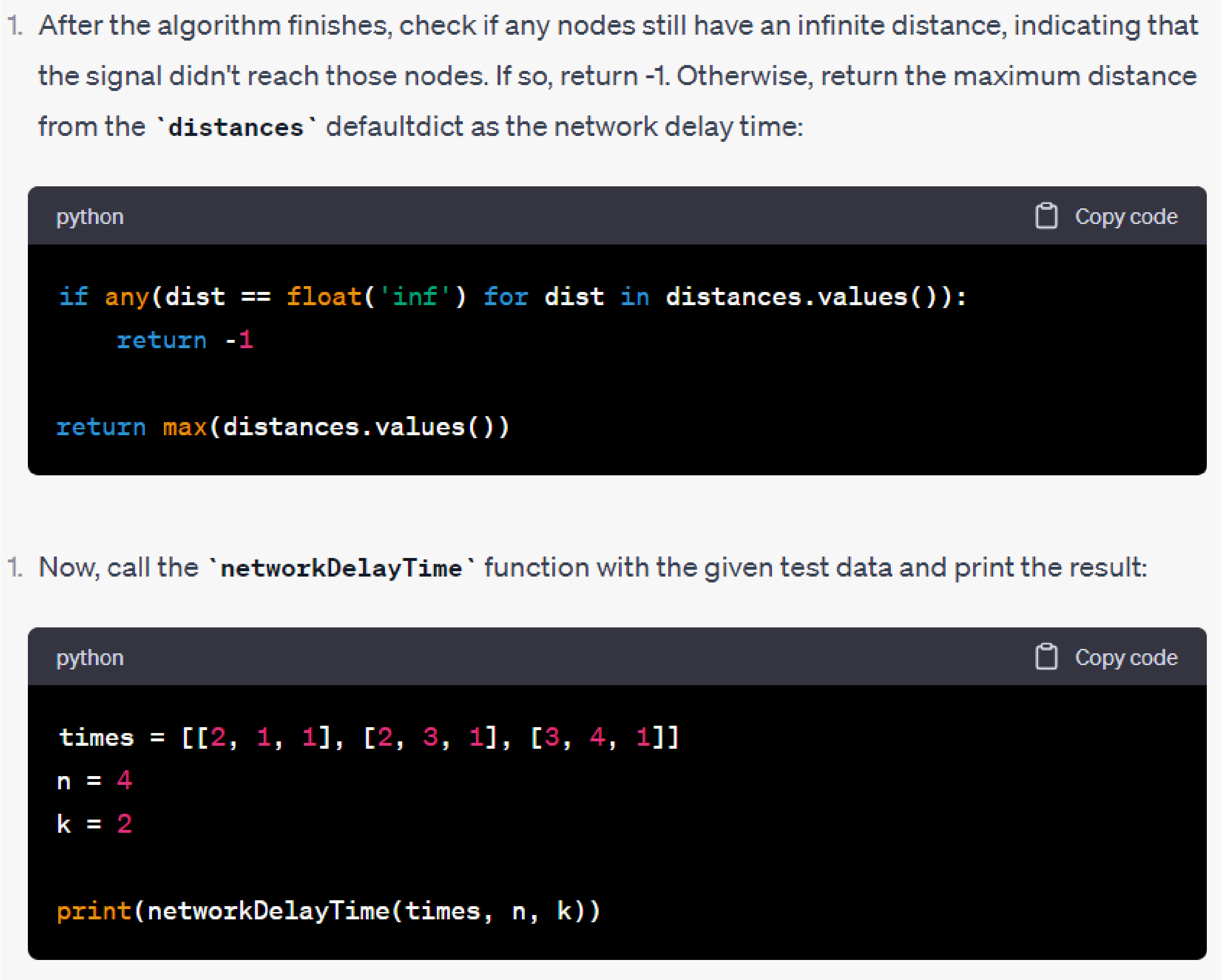
1. Create a priority queue to store the nodes and their tentative distances. Add the source node `k` with a distance of 0 to the priority queue:

```python
pq = [(0, k)]
```



1. While the priority queue is not empty, repeat the following steps:
   a. Pop the node with the minimum distance from the priority queue. Let's call this node `curr` and its distance `curr_distance`.
   b. If the current distance is greater than the distance stored in the `distances` defaultdict for the current node, continue to the next iteration.
   c. Update the distances for the neighboring nodes of `curr`. If the new distance is smaller than the current distance stored in the `distances` defaultdict, update the distance and add the neighbor node to the priority queue:

```python
while pq:
    curr_distance, curr = heapq.heappop(pq)

    if curr_distance > distances[curr]:
        continue

    for neighbor, edge_weight in graph[curr]:
        distance = curr_distance + edge_weight
        if distance < distances[neighbor]:
            distances[neighbor] = distance
            heapq.heappush(pq, (distance, neighbor))
```

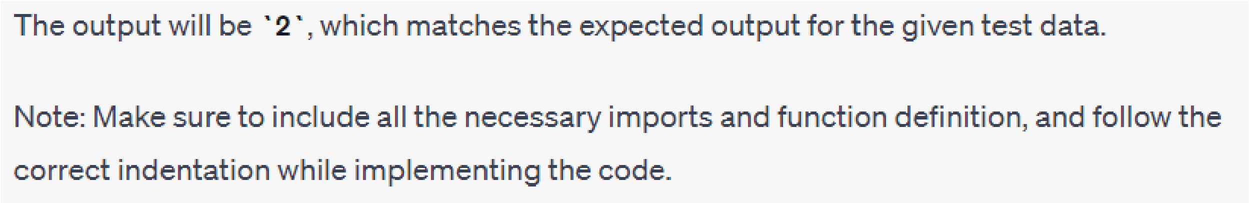


1. After the algorithm finishes, check if any nodes still have an infinite distance, indicating that the signal didn't reach those nodes. If so, return -1. Otherwise, return the maximum distance from the `distances` defaultdict as the network delay time:

```python
if any(dist == float('inf') for dist in distances.values()):
    return -1

return max(distances.values())
```

1. Now, call the `networkDelayTime` function with the given test data and print the result:

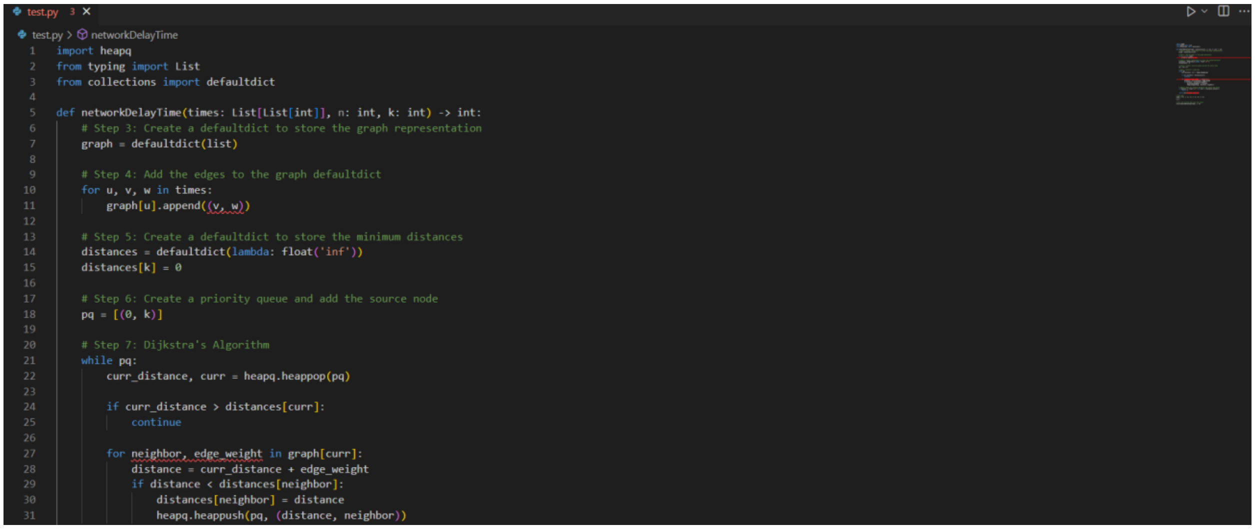
```python
times = [[2, 1, 1], [2, 3, 1], [3, 4, 1]]
n = 4
k = 2

print(networkDelayTime(times, n, k))
```



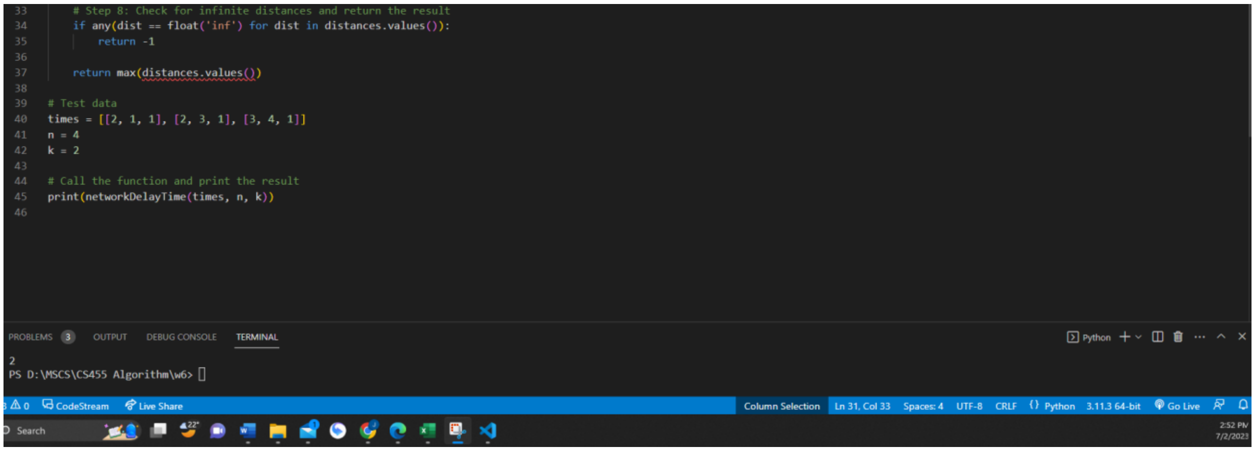
The output will be `2`, which matches the expected output for the given test data.

Note: Make sure to include all the necessary imports and function definition, and follow the correct indentation while implementing the code.

Step 4: Implement the Python code created by ChatGPT



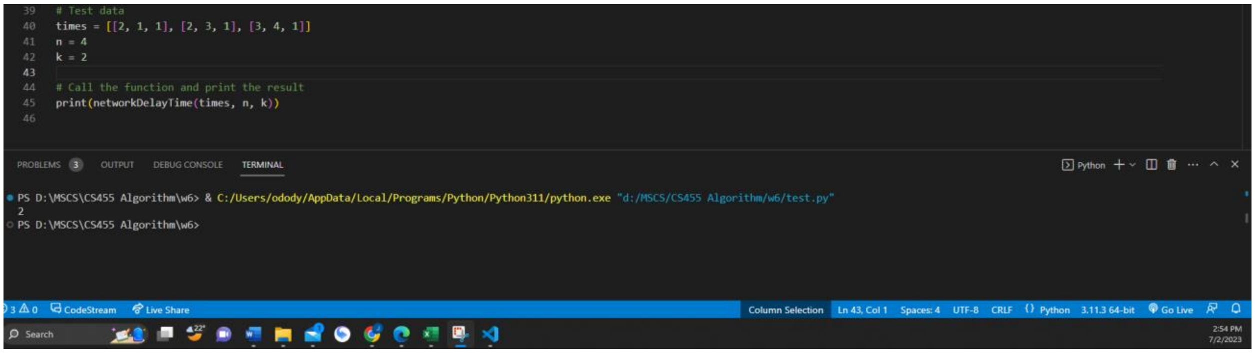
```python
import heapq
from typing import List
from collections import defaultdict

def networkDelayTime(times: List[List[int]], n: int, k: int) -> int:
    # Step 3: Create a defaultdict to store the graph representation
    graph = defaultdict(list)

    # Step 4: Add the edges to the graph defaultdict
    for u, v, w in times:
        graph[u].append((v, w))

    # Step 5: Create a defaultdict to store the minimum distances
    distances = defaultdict(lambda: float('inf'))
    distances[k] = 0

    # Step 6: Create a priority queue and add the source node
    pq = [(0, k)]

    # Step 7: Dijkstra's Algorithm
    while pq:
        curr_distance, curr = heapq.heappop(pq)

        if curr_distance > distances[curr]:
            continue

        for neighbor, edge_weight in graph[curr]:
            distance = curr_distance + edge_weight
            if distance < distances[neighbor]:
                distances[neighbor] = distance
                heapq.heappush(pq, (distance, neighbor))
```



```python
    # Step 8: Check for infinite distances and return the result
    if any(dist == float('inf') for dist in distances.values()):
        return -1

    return max(distances.values())

# Test data
times = [[2, 1, 1], [2, 3, 1], [3, 4, 1]]
n = 4
k = 2

# Call the function and print the result
print(networkDelayTime(times, n, k))
```

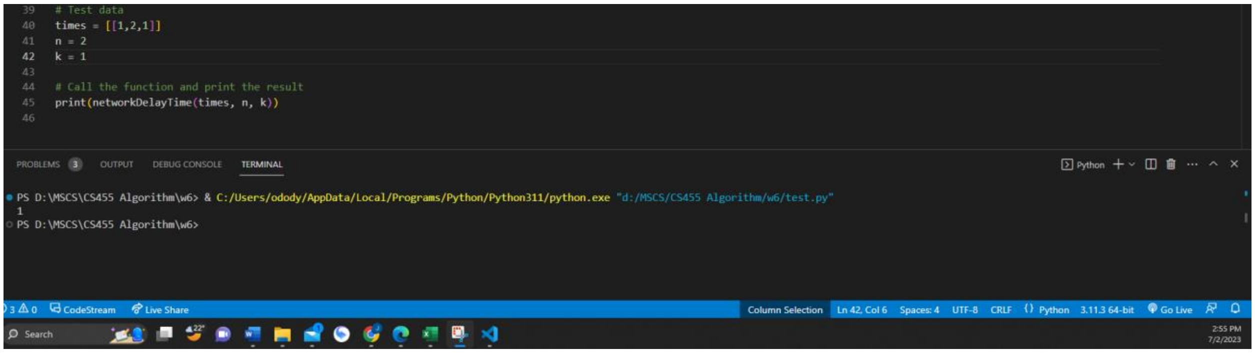
```
2
PS D:\MSCS\CS455 Algorithm\w6> 
```

Step 5: Test the Python code with all the test cases provided by 743. Network Delay Time 50.3 (local copy)- Medium



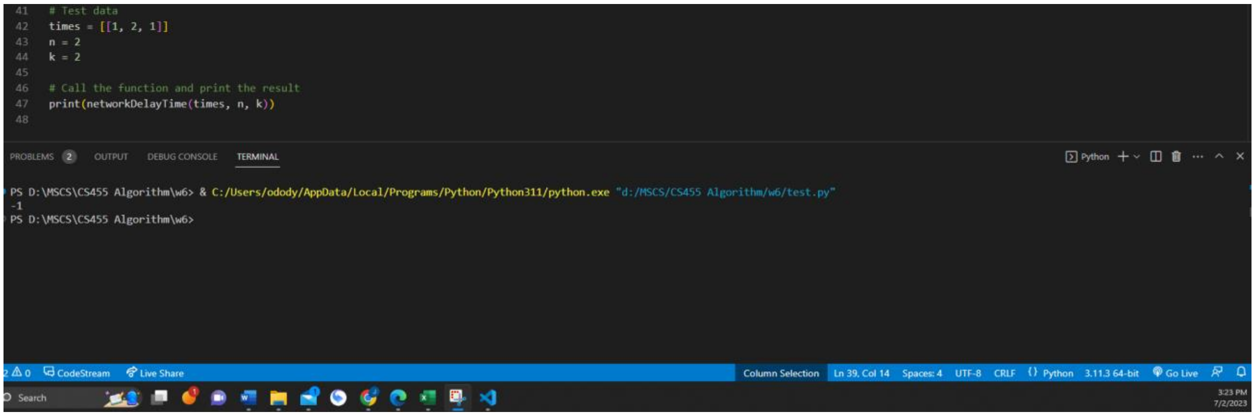
```python
# Test data
times = [[2, 1, 1], [2, 3, 1], [3, 4, 1]]
n = 4
k = 2

# Call the function and print the result
print(networkDelayTime(times, n, k))
```

```
PS D:\MSCS\CS455 Algorithm\w6> & C:/Users/odody/AppData/Local/Programs/Python/Python311/python.exe "d:/MSCS/CS455 Algorithm/w6/test.py"
2
PS D:\MSCS\CS455 Algorithm\w6>
```



```python
# Test data
times = [[1,2,1]]
n = 2
k = 1

# Call the function and print the result
print(networkDelayTime(times, n, k))
```

```
PS D:\MSCS\CS455 Algorithm\w6> & C:/Users/odody/AppData/Local/Programs/Python/Python311/python.exe "d:/MSCS/CS455 Algorithm/w6/test.py"
1
PS D:\MSCS\CS455 Algorithm\w6>
```



```python
# Test data
times = [[1, 2, 1]]
n = 2
k = 2

# Call the function and print the result
print(networkDelayTime(times, n, k))
```

```
PS D:\MSCS\CS455 Algorithm\w6> & C:/Users/odody/AppData/Local/Programs/Python/Python311/python.exe "d:/MSCS/CS455 Algorithm/w6/test.py"
-1
PS D:\MSCS\CS455 Algorithm\w6>
```