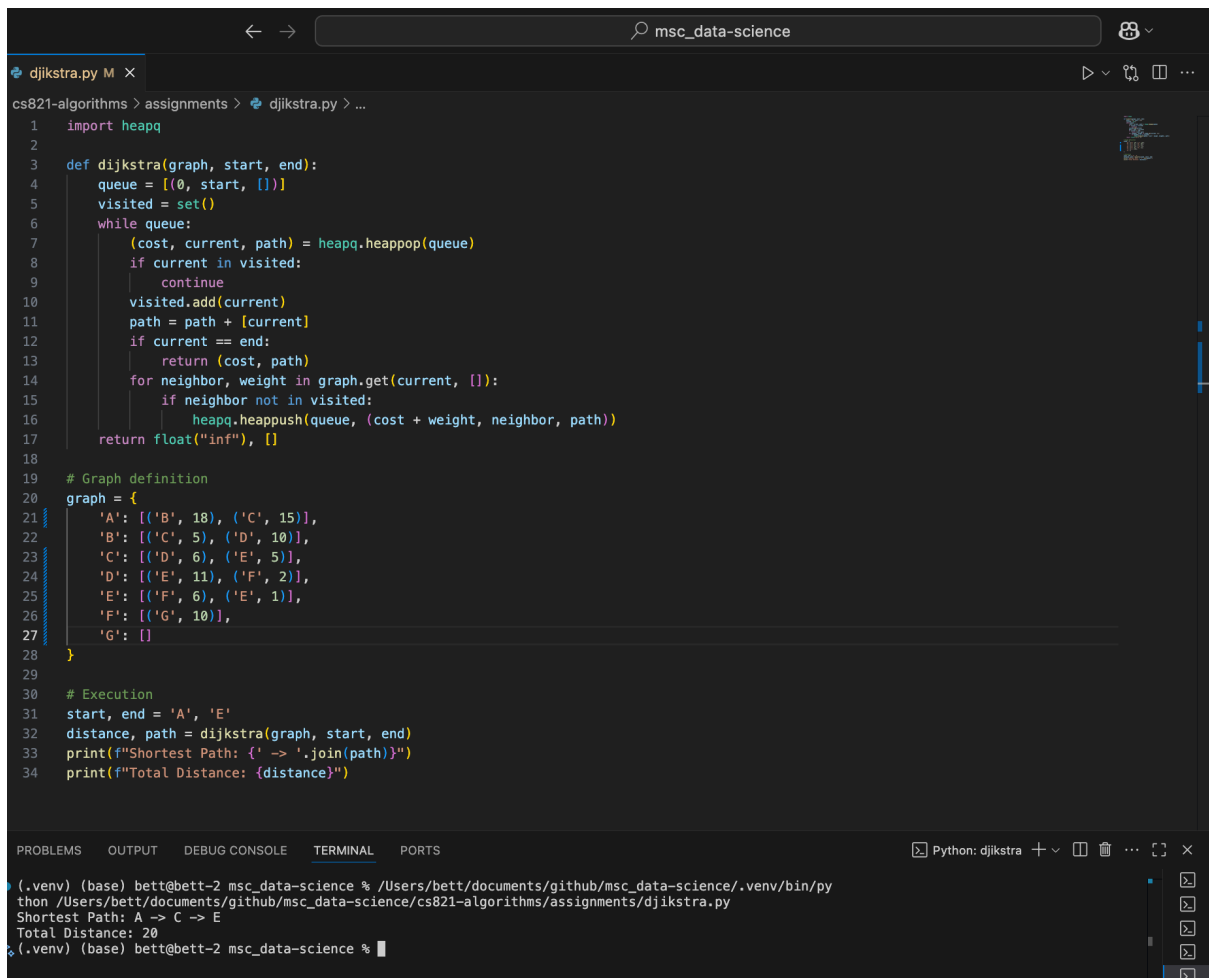


Dijkstra's algorithm uses a **greedy strategy** by always selecting the node with the smallest tentative distance from the source (A) and updating distances to its neighbors. Here's how it works step-by-step:

- Set distance to A = 0, others (B, C, D, E, F, G) = infinity.
- Priority queue: [(A, 0)], visited set: empty.
- Paths: Track predecessors to reconstruct the path.



```
1 import heapq
2
3 def dijkstra(graph, start, end):
4     queue = [(0, start, [])]
5     visited = set()
6     while queue:
7         (cost, current, path) = heapq.heappop(queue)
8         if current in visited:
9             continue
10        visited.add(current)
11        path = path + [current]
12        if current == end:
13            return (cost, path)
14        for neighbor, weight in graph.get(current, []):
15            if neighbor not in visited:
16                heapq.heappush(queue, (cost + weight, neighbor, path))
17    return float("inf"), []
18
19 # Graph definition
20 graph = {
21     'A': [('B', 18), ('C', 15)],
22     'B': [('C', 5), ('D', 10)],
23     'C': [('D', 6), ('E', 5)],
24     'D': [('E', 11), ('F', 2)],
25     'E': [('F', 6), ('G', 1)],
26     'F': [('G', 10)],
27     'G': []
28 }
29
30 # Execution
31 start, end = 'A', 'E'
32 distance, path = dijkstra(graph, start, end)
33 print(f"Shortest Path: {' -> '.join(path)}")
34 print(f"Total Distance: {distance}")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
(.venv) (base) bett@bett-2 msc_data-science % /Users/bett/documents/github/msc_data-science/.venv/bin/python /Users/bett/documents/github/msc_data-science/cs821-algorithms/assignments/dijkstra.py
Shortest Path: A -> C -> E
Total Distance: 20
(.venv) (base) bett@bett-2 msc_data-science %
```