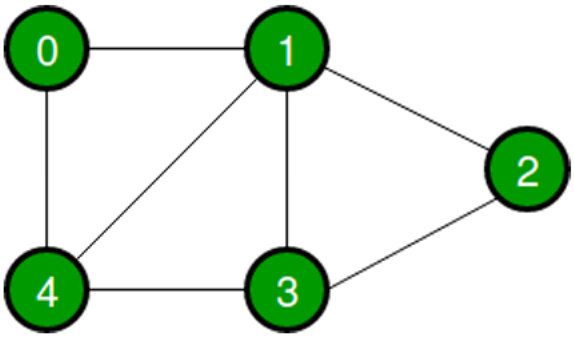
LAB 5

TASK:

Write a program that will implement the following graph in python.

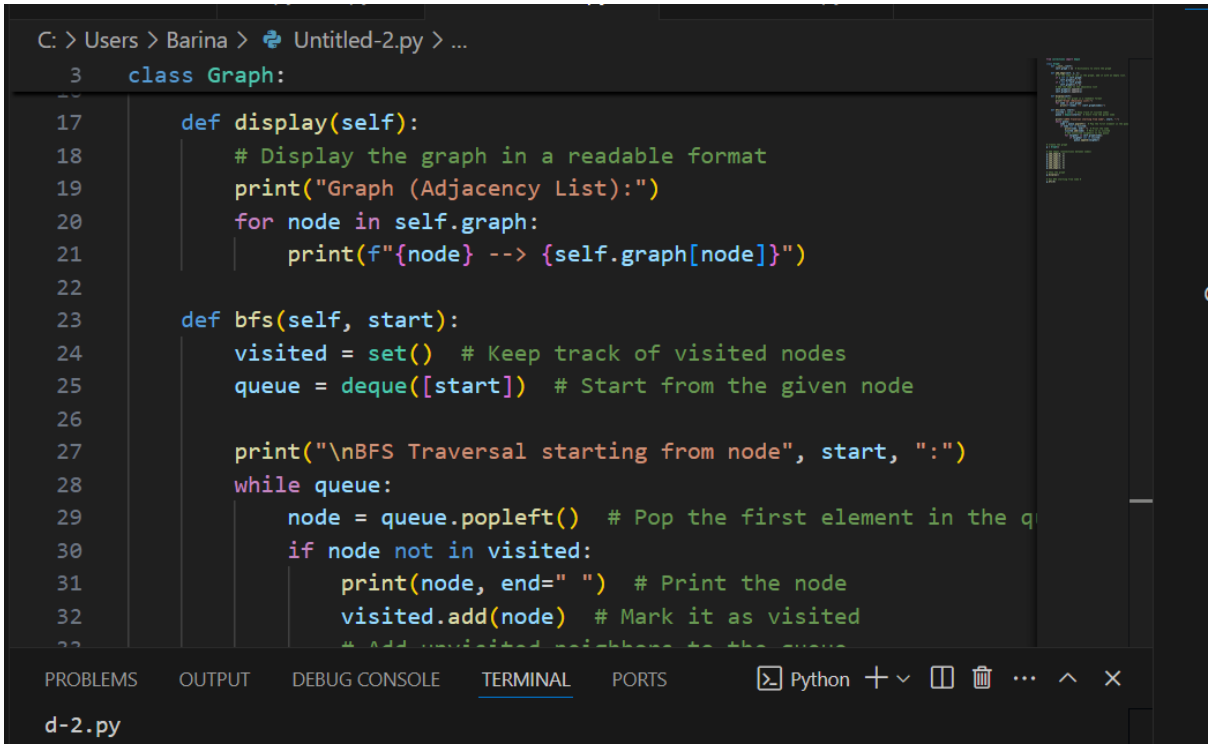


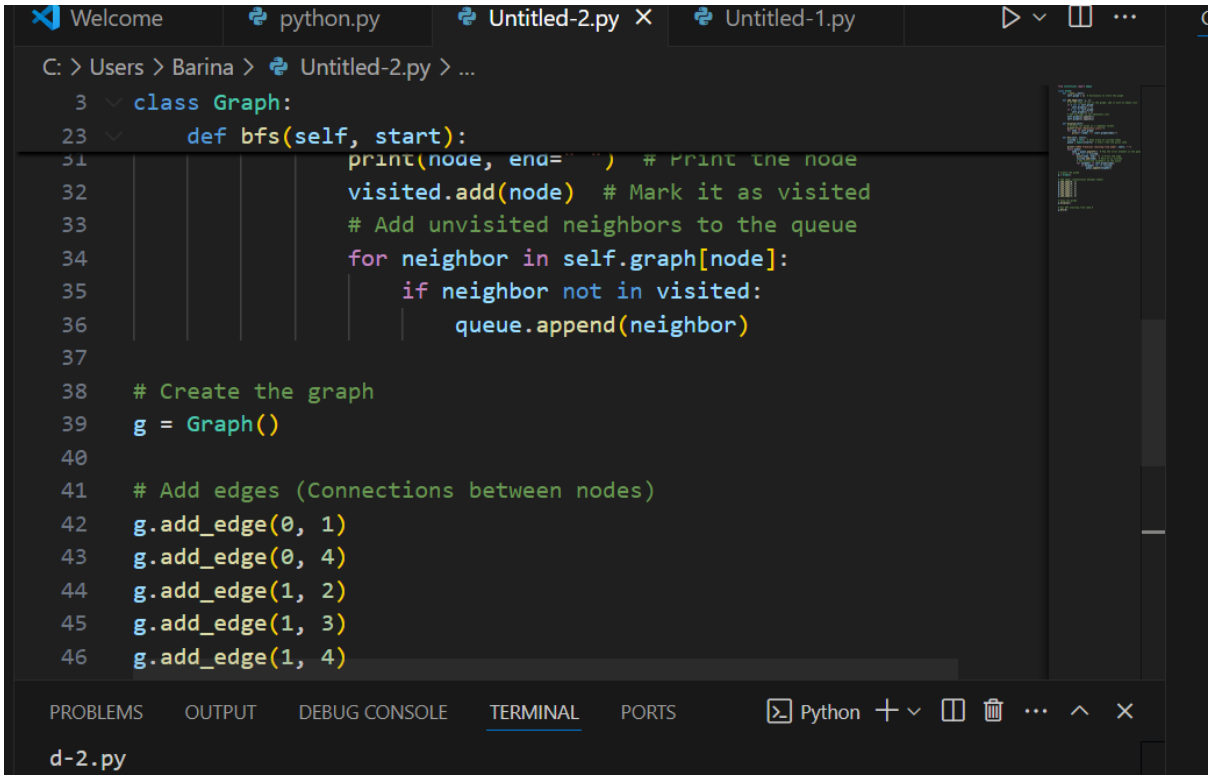
Welcome | python.py | Untitled-2.py | Untitled-1.py

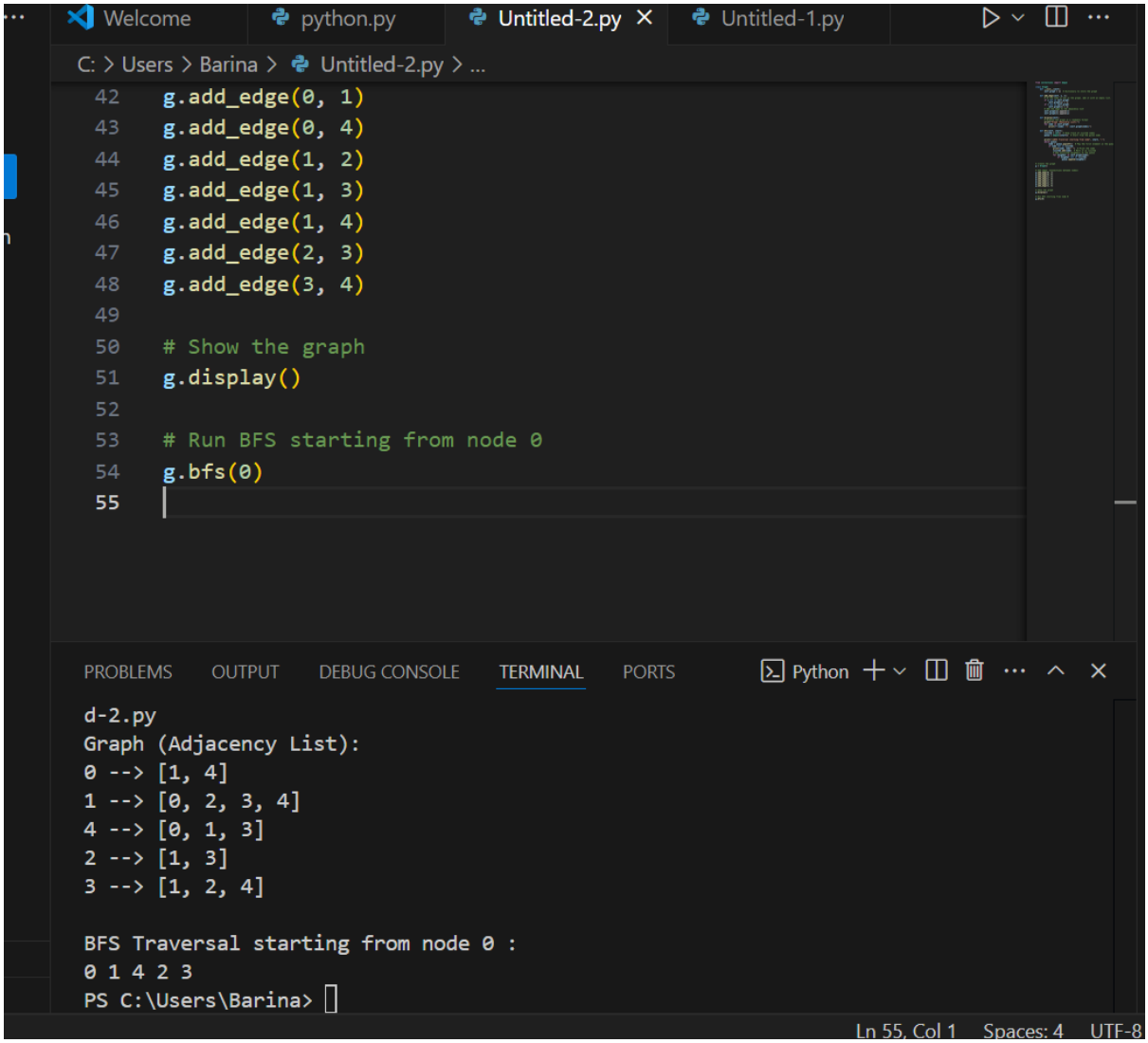
C: > Users > Barina > Untitled-2.py > ...

```python
from collections import deque

class Graph:
    def __init__(self):
        self.graph = {}  # Dictionary to store the graph

    def add_edge(self, u, v):
        # If the node is not in the graph, add it with an empty list
        if u not in self.graph:
            self.graph[u] = []
        if v not in self.graph:
            self.graph[v] = []
        # Add the edge to the adjacency list
        self.graph[u].append(v)
        self.graph[v].append(u)

    def display(self):
        # Display the graph in a readable format
```

PROBLEMS | OUTPUT | DEBUG CONSOLE | TERMINAL | PORTS | Python


C: > Users > Barina > Untitled-2.py > ...
3 class Graph:
17 def display(self):
18 # Display the graph in a readable format
19 print("Graph (Adjacency List):")
20 for node in self.graph:
21 print(f"{node} --> {self.graph[node]}")
22
23 def bfs(self, start):
24 visited = set() # Keep track of visited nodes
25 queue = deque([start]) # Start from the given node
26
27 print("\nBFS Traversal starting from node", start, ":")
28 while queue:
29 node = queue.popleft() # Pop the first element in the q
30 if node not in visited:
31 print(node, end=" ") # Print the node
32 visited.add(node) # Mark it as visited
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Python
d-2.py


Welcome
python.py
Untitled-2.py
Untitled-1.py
C: > Users > Barina > Untitled-2.py > ...
3 class Graph:
23 def bfs(self, start):
32 visited.add(node) # Mark it as visited
33 # Add unvisited neighbors to the queue
34 for neighbor in self.graph[node]:
35 if neighbor not in visited:
36 queue.append(neighbor)
37
38 # Create the graph
39 g = Graph()
40
41 # Add edges (Connections between nodes)
42 g.add_edge(0, 1)
43 g.add_edge(0, 4)
44 g.add_edge(1, 2)
45 g.add_edge(1, 3)
46 g.add_edge(1, 4)
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Python
d-2.py



TASK 2:

Implementation of BFS in c++.

from collections import deque, defaultdict

This class represents a directed graph using adjacency list representation

class Graph:

  def _init_(self, V):

    self.V = V # Number of vertices

    self.adj = defaultdict(list) # Dictionary to store adjacency lists

  # Function to add an edge to the graph

```
    def addEdge(self, v, w):
        self.adj[v].append(w)

    # Function to perform BFS traversal from a given source 's'
    def BFS(self, s):
        # Mark all the vertices as not visited
        visited = [False] * self.V

        # Create a queue for BFS
        queue = deque()

        # Mark the current node as visited and enqueue it
        visited[s] = True
        queue.append(s)

        while queue:
            # Dequeue a vertex from queue and print it
            s = queue.popleft()
            print(s, end=" ")

            # Get all adjacent vertices of the dequeued vertex s
            # If an adjacent has not been visited, mark it visited and enqueue it
            for i in self.adj[s]:
                if not visited[i]:
                    visited[i] = True
                    queue.append(i)

# Driver program to test the methods of the Graph class
```

```
if _name_ == "_main_":
    # Create a graph with 4 vertices
    g = Graph(4)
    g.addEdge(0, 1)
    g.addEdge(0, 2)
    g.addEdge(1, 2)
    g.addEdge(2, 0)
    g.addEdge(2, 3)
    g.addEdge(3, 3)

    print("Following is Breadth First Traversal (starting from vertex 2):")
    g.BFS(2)
```

IN Python:

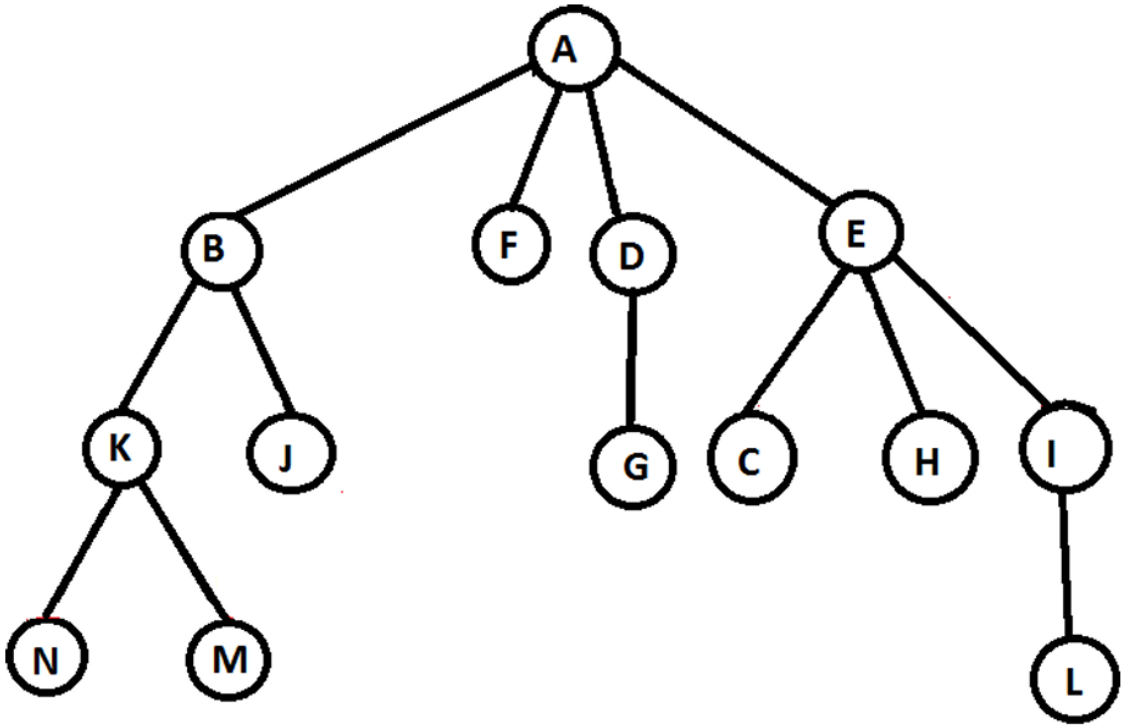
```
from collections import defaultdict, deque

class Graph:
    def _init_(self, V):
        self.V = V  # Number of vertices
        self.adj = defaultdict(list)  # Dictionary to store adjacency list

    def addEdge(self, v, w):
        self.adj[v].append(w)  # Add w to v's list

    def BFS(self, s):
        visited = [False] * self.V  # Mark all vertices as not visited
        queue = deque()  # Create a queue for BFS

        visited[s] = True  # Mark the source node as visited
```

```
        queue.append(s)

        while queue:
            s = queue.popleft()  # Dequeue a vertex
            print(s, end=" ")

            # Get all adjacent vertices
            for i in self.adj[s]:
                if not visited[i]:
                    visited[i] = True
                    queue.append(i)

# Driver code to test the Graph class
g = Graph(4)
g.addEdge(0, 1)
g.addEdge(0, 2)
g.addEdge(1, 2)
g.addEdge(2, 0)
g.addEdge(2, 3)
g.addEdge(3, 3)

print("Following is Breadth First Traversal (starting from vertex 2):")
g.BFS(2)
```

You are given the following tree whose starting node is A and Goal node is G

TASK 3:

You are given the following tree whose starting node is **A** and Goal node is **G**



You are required to implement the following graph in python and then apply the following search(Stop the search when goal is achieved)

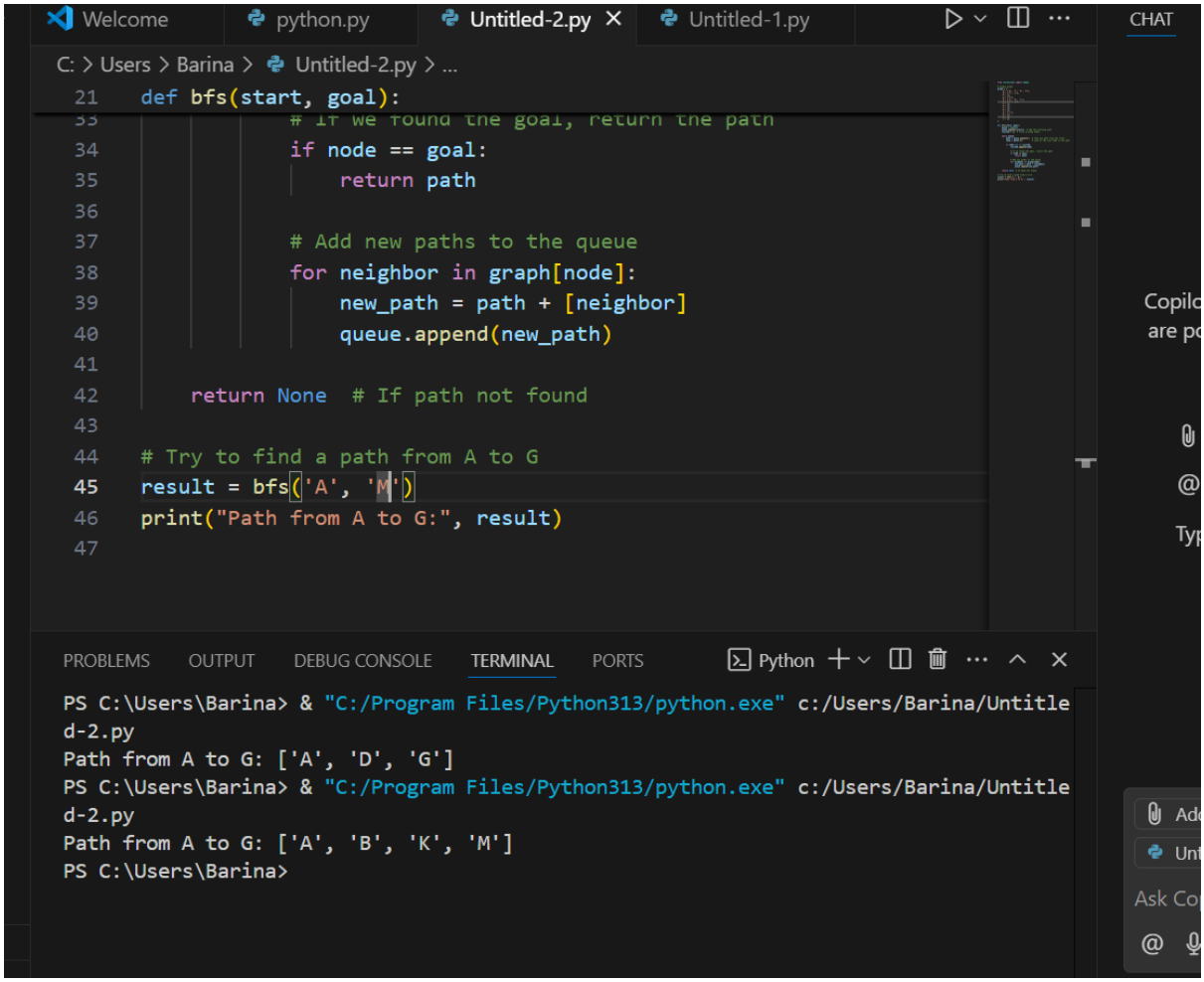
a) **Breadth First Search using Queue.**

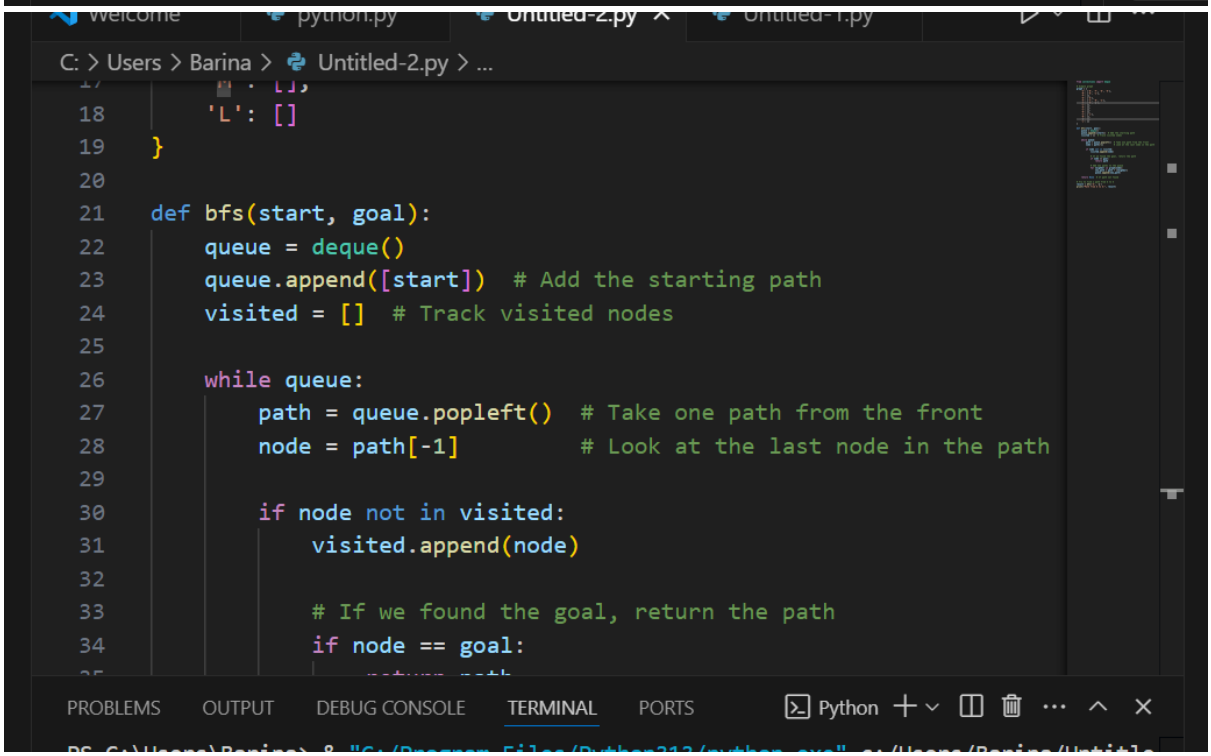
C: > Users > Barina > Untitled-2.py > ...

```
1   from collections import deque
2
3   # Simple graph
4   graph = {
5       'A': ['B', 'F', 'D', 'E'],
6       'B': ['K', 'J'],
7       'F': [],
8       'D': ['G'],
9       'E': ['C', 'H', 'I'],
10      'K': ['N', 'M'],
11      'J': [],
12      'G': [],
13      'C': [],
14      'H': [],
15      'I': ['L'],
16      'N': [],
17      'M': [],
18      'L': []
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python

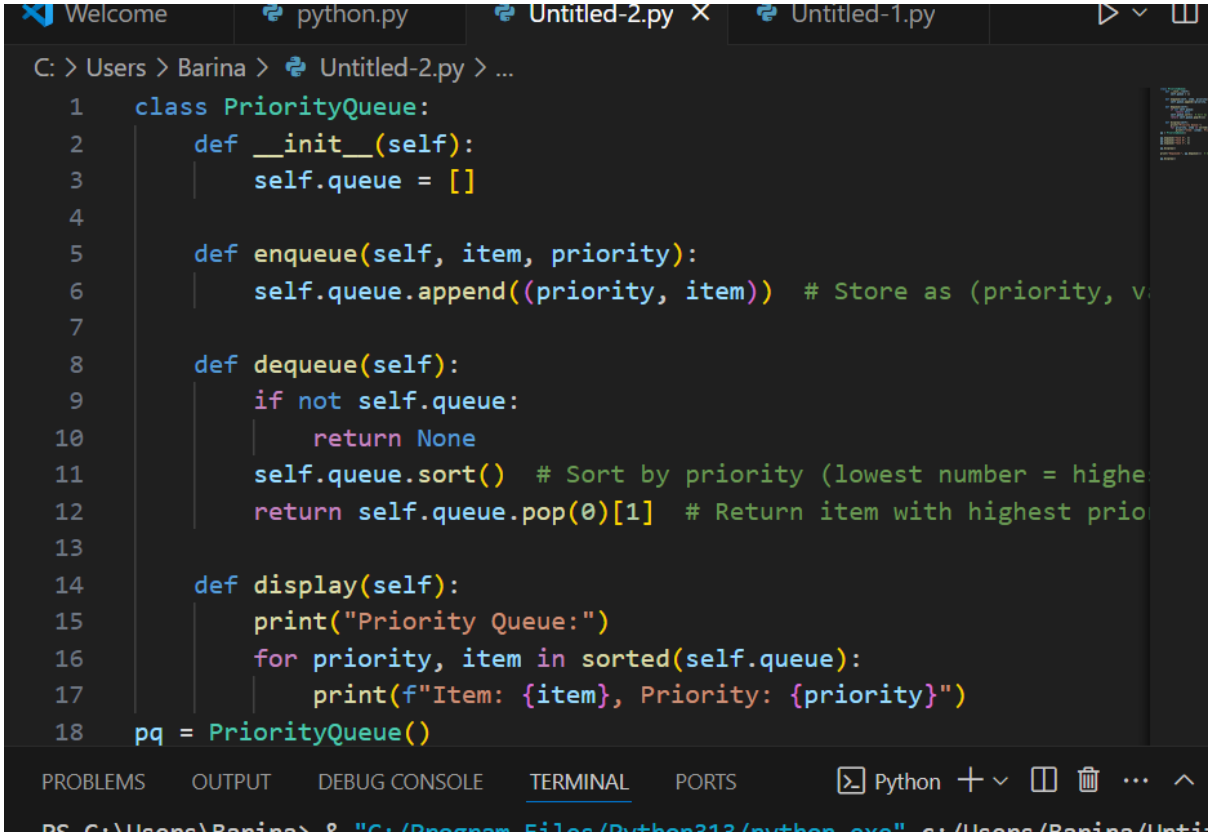
```
PS C:\Users\Barina> & "C:/Program Files/Python313/python.exe" c:/Users/Barina/Untitle
```

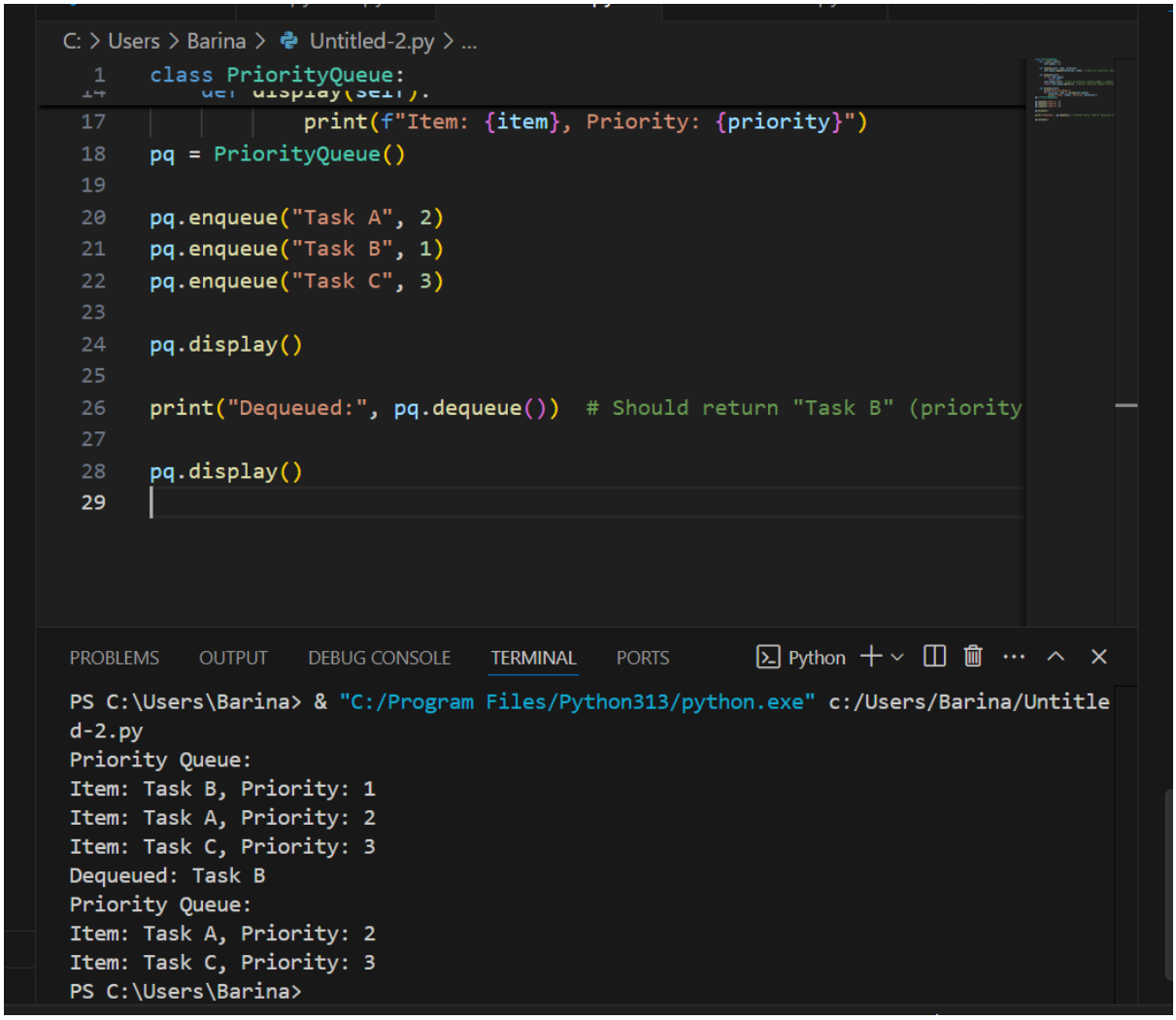

Welcome
python.py
Untitled-2.py
Untitled-1.py
CHAT
C: > Users > Barina > Untitled-2.py > ...
21 def bfs(start, goal):
34 if node == goal:
35 return path
37 # Add new paths to the queue
38 for neighbor in graph[node]:
39 new_path = path + [neighbor]
40 queue.append(new_path)
42 return None # If path not found
44 # Try to find a path from A to G
45 result = bfs('A', 'M')
46 print("Path from A to G:", result)
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Python
PS C:\Users\Barina> & "C:/Program Files/Python313/python.exe" c:/Users/Barina/Untitled-2.py
Path from A to G: ['A', 'D', 'G']
PS C:\Users\Barina> & "C:/Program Files/Python313/python.exe" c:/Users/Barina/Untitled-2.py
Path from A to G: ['A', 'B', 'K', 'M']
PS C:\Users\Barina>


C: > Users > Barina > Untitled-2.py > ...
18 'L': []
19 }
21 def bfs(start, goal):
22 queue = deque()
23 queue.append([start]) # Add the starting path
24 visited = [] # Track visited nodes
26 while queue:
27 path = queue.popleft() # Take one path from the front
28 node = path[-1] # Look at the last node in the path
30 if node not in visited:
31 visited.append(node)
33 # If we found the goal, return the path
34 if node == goal:
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Python

Task3:

Implement priority Queue.




C: > Users > Barina > Untitled-2.py > ...
1 class PriorityQueue:
17 print(f"Item: {item}, Priority: {priority}")
18 pq = PriorityQueue()
19
20 pq.enqueue("Task A", 2)
21 pq.enqueue("Task B", 1)
22 pq.enqueue("Task C", 3)
23
24 pq.display()
25
26 print("Dequeued:", pq.dequeue()) # Should return "Task B" (priority
27
28 pq.display()
29
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python
PS C:\Users\Barina> & "C:/Program Files/Python313/python.exe" c:/Users/Barina/Untitled-2.py
Priority Queue:
Item: Task B, Priority: 1
Item: Task A, Priority: 2
Item: Task C, Priority: 3
Dequeued: Task B
Priority Queue:
Item: Task A, Priority: 2
Item: Task C, Priority: 3
PS C:\Users\Barina>