# Report

## Simulating Router with Shortest Path Algorithm

This Python code implements the Bellman-Ford algorithm to find the shortest path routing in a directed graph. The graph is represented as a dictionary of nodes and their weighted edges. The code computes the shortest path routing from a given source node to all other nodes in the graph, considering edge weights.

**Graph Definition:** The code defines the graph using a dictionary where each node is a key, and the associated value is another dictionary containing the neighboring nodes as keys and their edge weights as values. This graph represents a network with nodes A through G and weighted edges between them.

**Bellman-Ford Algorithm:** The Bellman-Ford algorithm is employed to find the shortest path routing. This algorithm is known for its ability to handle graphs with negative edge weights and detect negative-weight cycles. The algorithm works in several steps:

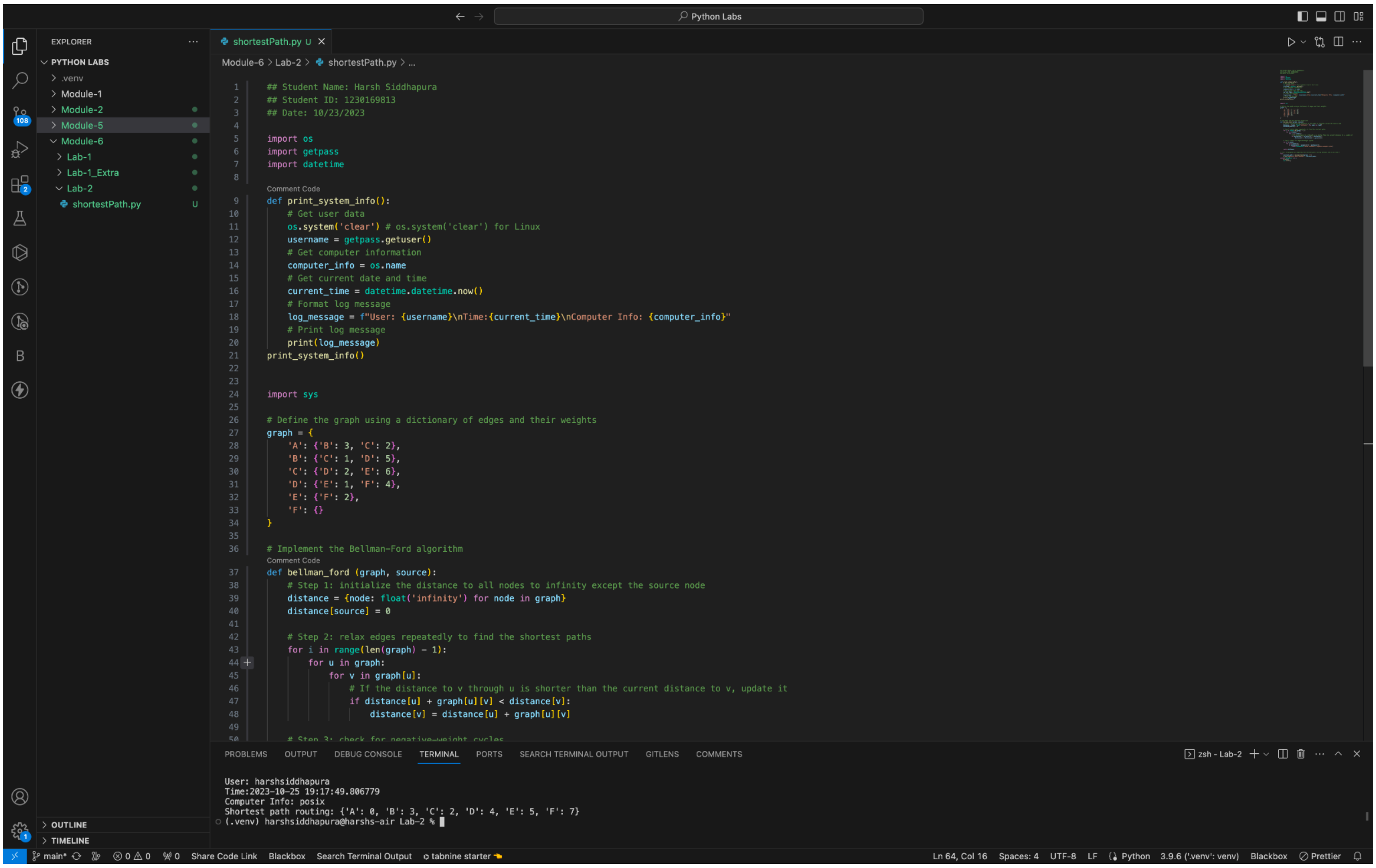
- Initialization: It starts by initializing the distance to all nodes as infinity, except for the source node, which has a distance of 0. This step prepares the distance dictionary to keep track of the shortest distances from the source node to all other nodes.
- Relaxation: The code then enters a loop that iterates a maximum of (number of nodes - 1) times. In each iteration, it goes through all nodes and their neighbors, comparing the distance through the current node to the recorded distance to the neighbor node. If the distance through the current node is shorter, it updates the distance.
- Negative-Weight Cycles: After completing the relaxation step, the code checks for negative-weight cycles. It again goes through all nodes and their neighbors, and if it finds a shorter distance through the current node to the neighbor, it raises an exception indicating that the graph contains a negative-weight cycle.

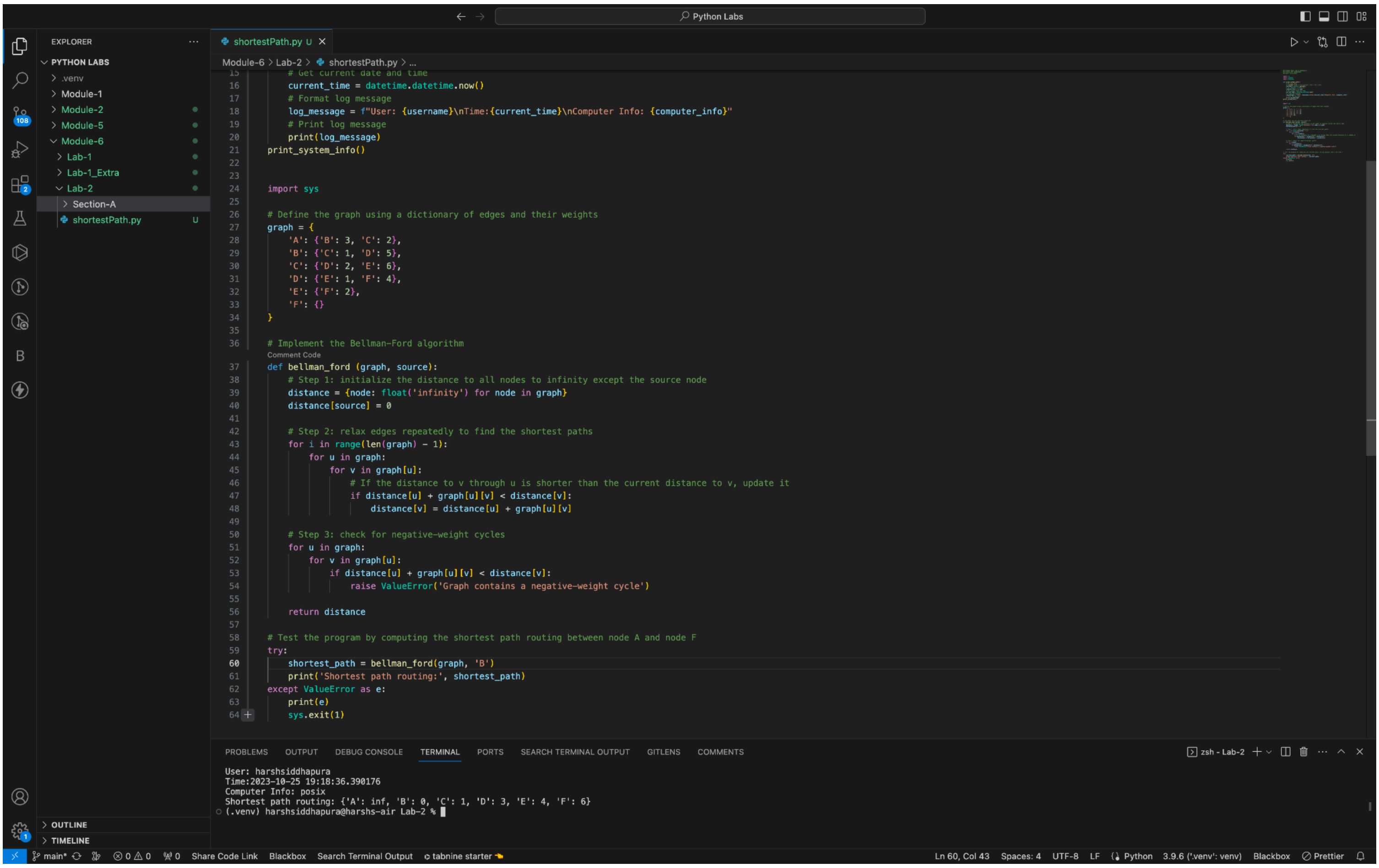
**Conclusion:** In conclusion, this Python code offers a well-structured and effective implementation of the Bellman-Ford algorithm for finding the shortest path routing in a directed graph. This algorithm is particularly valuable because it can accommodate graphs with negative edge weights and can detect the presence of negative-weight cycles. Here are some key points to emphasize:

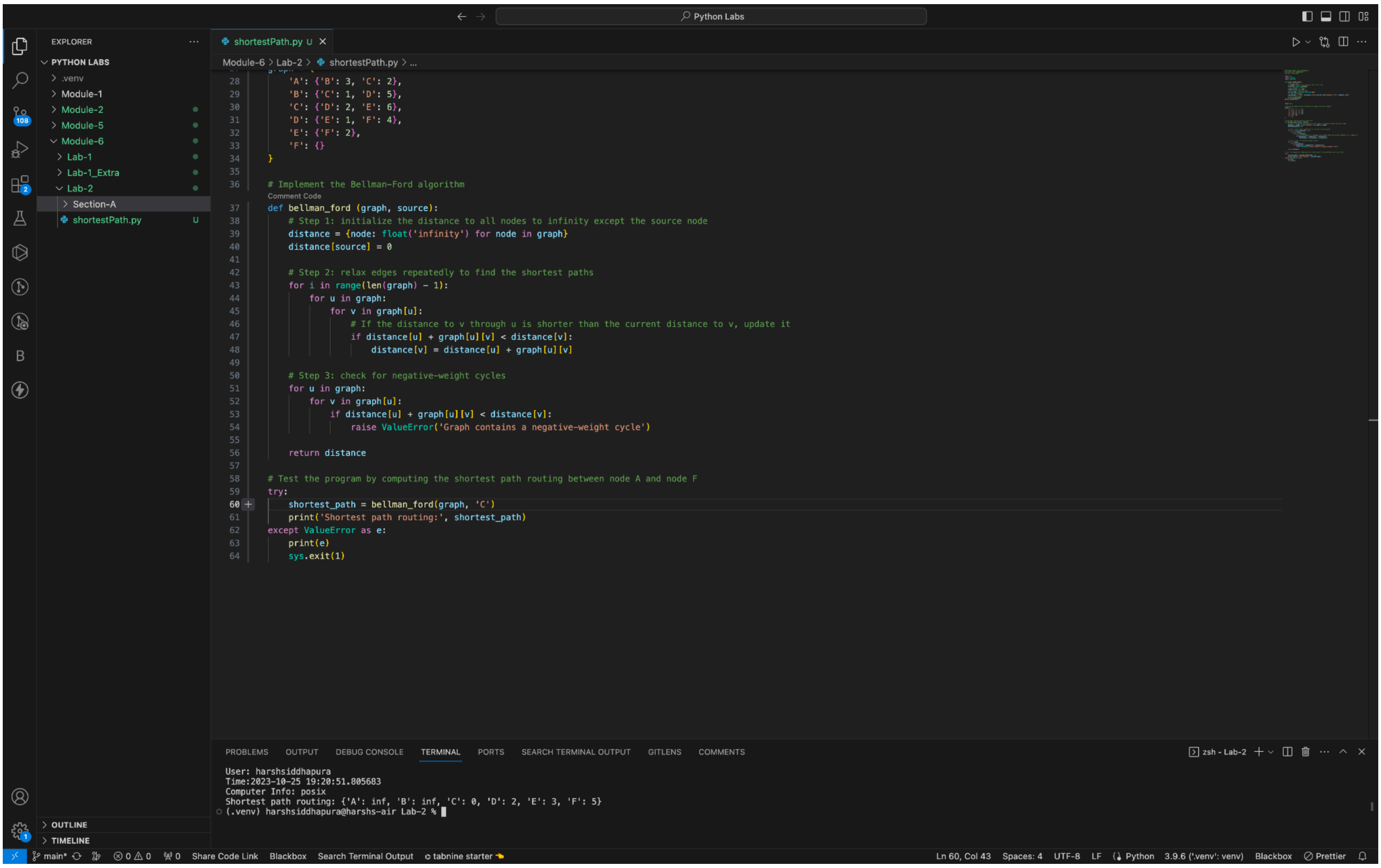
- Graph Representation: The code defines the network graph as a dictionary, making it easy to represent nodes, their connections, and edge weights. This clear representation simplifies the process of defining and analyzing complex network structures.
- Bellman-Ford Algorithm: The core of the code is the Bellman-Ford algorithm. It performs three crucial steps: initialization, relaxation, and negative-weight cycle detection. This ensures that it can find the shortest path from a source node to all other nodes, handling various edge weights.
- Negative-Weight Cycles: The code incorporates a check for negative-weight cycles. If such cycles are detected during the computation, the code raises a ValueError, indicating the presence of these cycles. This is a valuable feature for network analysis as it helps identify problematic situations.
- Testing and Source Selection: The code is tested with a specific source node ('C') to demonstrate the computation of the shortest path routing. The choice of 'C' as the source node showcases the flexibility of the code in selecting the starting point for routing analysis.

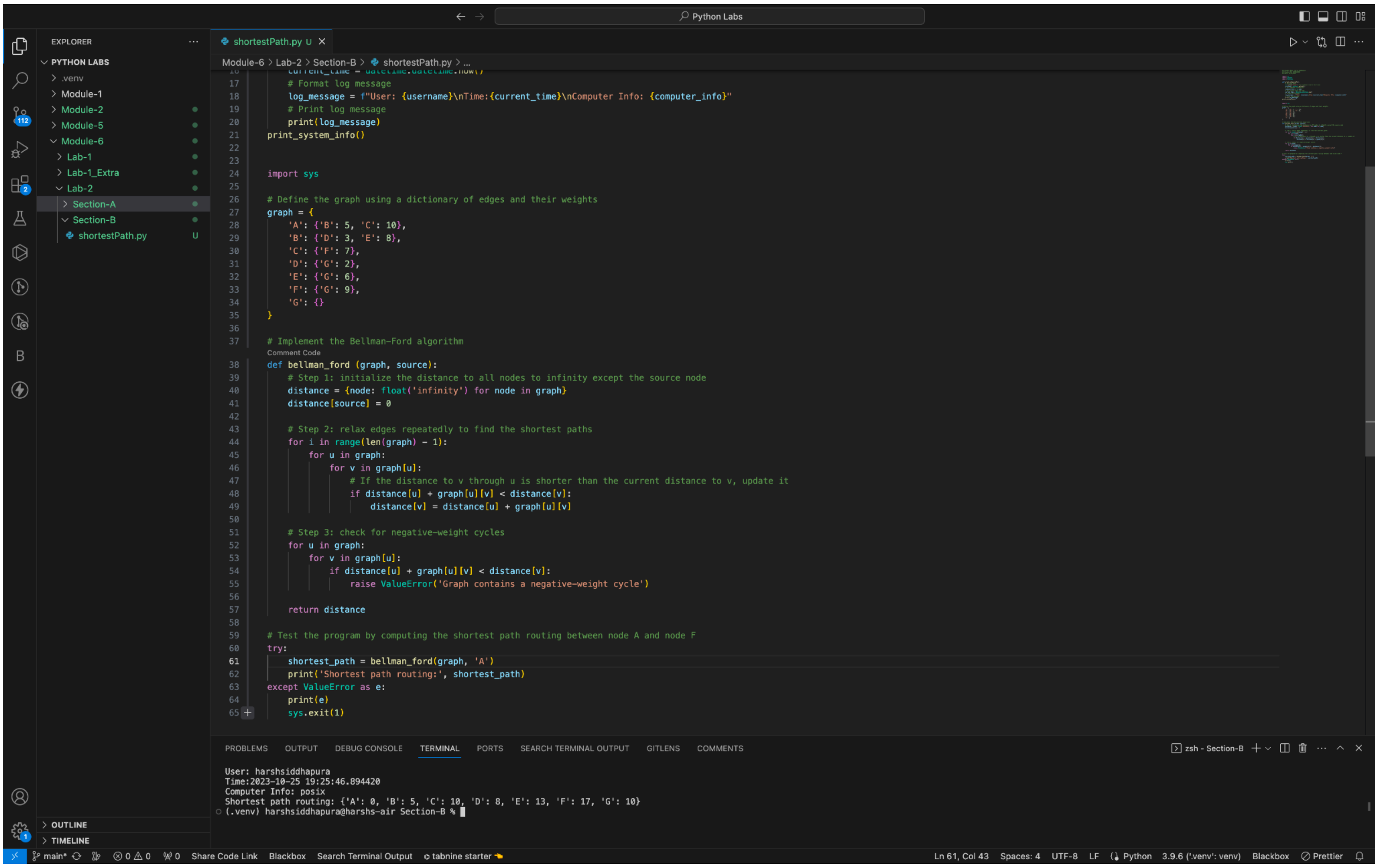
In summary, this Python code is a valuable resource for network engineers, researchers, and anyone working with graph-based routing problems. It provides a clear and reliable implementation of the Bellman-Ford algorithm and can be a foundation for more complex network analysis and optimization tasks. Its ability to handle negative-weight cycles and detect them ensures the reliability and correctness of routing solutions.

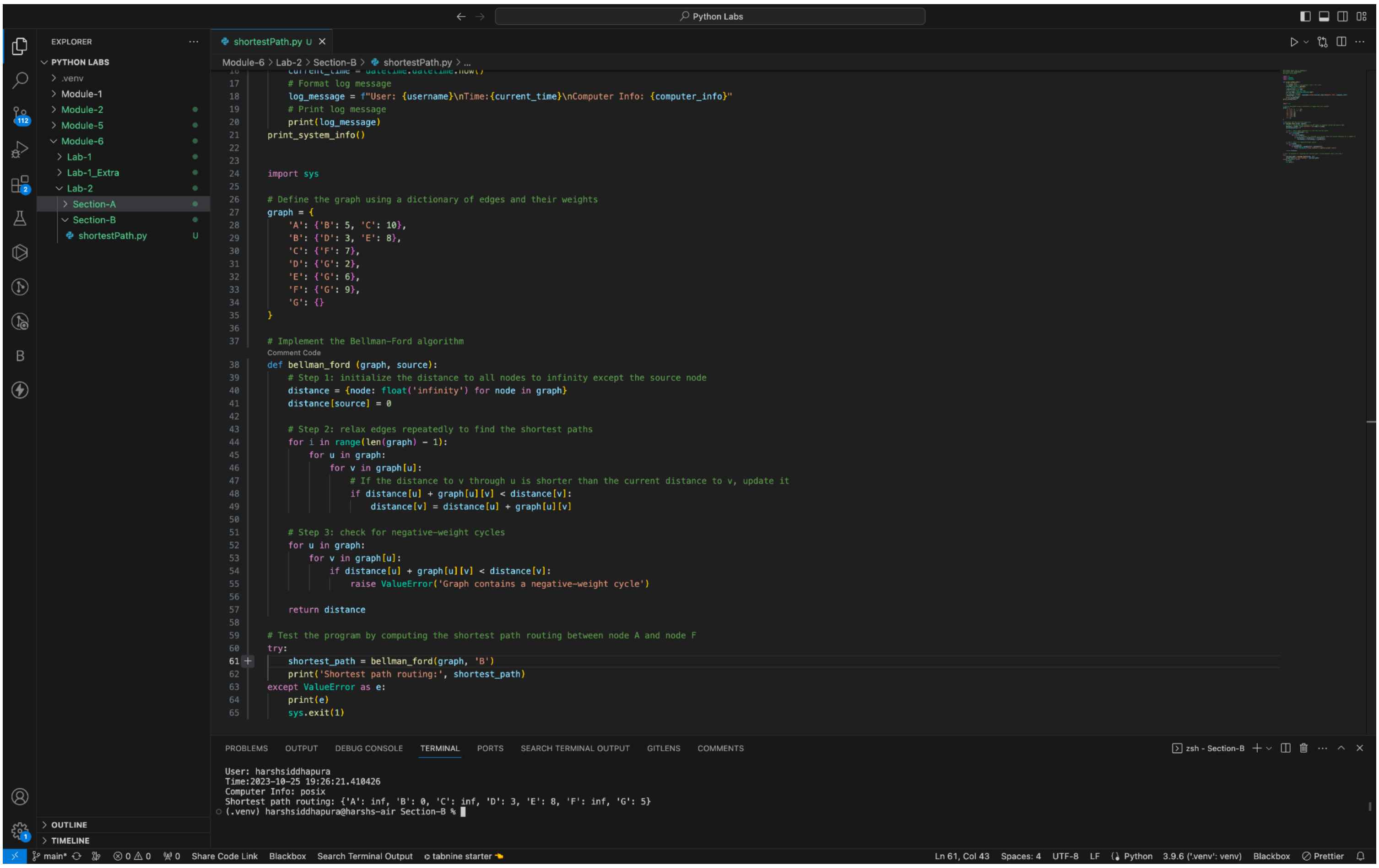
# Screenshot

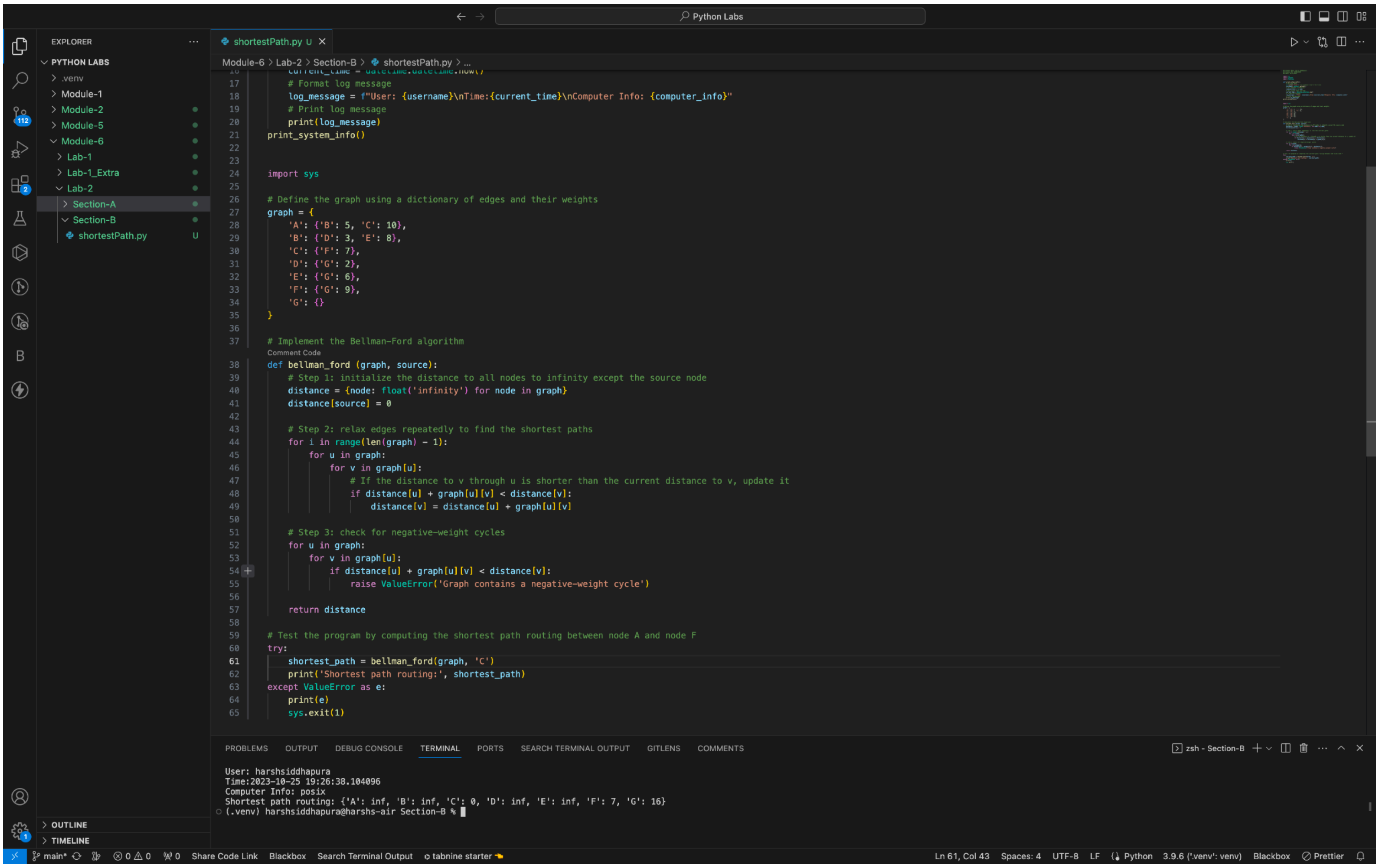

Python Labs
EXPLORER
PYTHON LABS
.venv
Module-1
Module-2
Module-5
Module-6
Lab-1
Lab-1_Extra
Lab-2
shortestPath.py
shortestPath.py U
Module-6 > Lab-2 > shortestPath.py > ...
## Student Name: Harsh Siddhapura
## Student ID: 1230169813
## Date: 10/23/2023
import os
import getpass
import datetime
Comment Code
def print_system_info():
# Get user data
os.system('clear') # os.system('clear') for Linux
username = getpass.getuser()
# Get computer information
computer_info = os.name
# Get current date and time
current_time = datetime.datetime.now()
# Format log message
log_message = f"User: {username}\nTime:{current_time}\nComputer Info: {computer_info}"
# Print log message
print(log_message)
print_system_info()
import sys
# Define the graph using a dictionary of edges and their weights
graph = {
'A': {'B': 3, 'C': 2},
'B': {'C': 1, 'D': 5},
'C': {'D': 2, 'E': 6},
'D': {'E': 1, 'F': 4},
'E': {'F': 2},
'F': {}
}
# Implement the Bellman-Ford algorithm
Comment Code
def bellman_ford (graph, source):
# Step 1: initialize the distance to all nodes to infinity except the source node
distance = {node: float('infinity') for node in graph}
distance[source] = 0
# Step 2: relax edges repeatedly to find the shortest paths
for i in range(len(graph) - 1):
for u in graph:
for v in graph[u]:
# If the distance to v through u is shorter than the current distance to v, update it
if distance[u] + graph[u][v] < distance[v]:
distance[v] = distance[u] + graph[u][v]
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SEARCH TERMINAL OUTPUT GITLENS COMMENTS
zsh - Lab-2
User: harshsiddhapura
Time:2023-10-25 19:17:49.806779
Computer Info: posix
Shortest path routing: {'A': 0, 'B': 3, 'C': 2, 'D': 4, 'E': 5, 'F': 7}
(.venv) harshsiddhapura@harshs-air Lab-2 %
OUTLINE
TIMELINE
main* Share Code Link Blackbox Search Terminal Output tabnine starter
Ln 64, Col 16 Spaces: 4 UTF-8 LF Python 3.9.6 ('.venv': venv) Blackbox Prettier


Python Labs
EXPLORER
PYTHON LABS
.venv
Module-1
Module-2
Module-5
Module-6
Lab-1
Lab-1_Extra
Lab-2
Section-A
shortestPath.py
shortestPath.py U
Module-6 > Lab-2 > shortestPath.py > ...
current_time = datetime.datetime.now()
# Format log message
log_message = f"User: {username}\nTime:{current_time}\nComputer Info: {computer_info}"
# Print log message
print(log_message)
print_system_info()
import sys
# Define the graph using a dictionary of edges and their weights
graph = {
'A': {'B': 3, 'C': 2},
'B': {'C': 1, 'D': 5},
'C': {'D': 2, 'E': 6},
'D': {'E': 1, 'F': 4},
'E': {'F': 2},
'F': {}
}
# Implement the Bellman-Ford algorithm
Comment Code
def bellman_ford (graph, source):
# Step 1: initialize the distance to all nodes to infinity except the source node
distance = {node: float('infinity') for node in graph}
distance[source] = 0
# Step 2: relax edges repeatedly to find the shortest paths
for i in range(len(graph) - 1):
for u in graph:
for v in graph[u]:
# If the distance to v through u is shorter than the current distance to v, update it
if distance[u] + graph[u][v] < distance[v]:
distance[v] = distance[u] + graph[u][v]
# Step 3: check for negative-weight cycles
for u in graph:
for v in graph[u]:
if distance[u] + graph[u][v] < distance[v]:
raise ValueError('Graph contains a negative-weight cycle')
return distance
# Test the program by computing the shortest path routing between node A and node F
try:
shortest_path = bellman_ford(graph, 'B')
print('Shortest path routing:', shortest_path)
except ValueError as e:
print(e)
sys.exit(1)
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SEARCH TERMINAL OUTPUT GITLENS COMMENTS
zsh - Lab-2
User: harshsiddhapura
Time:2023-10-25 19:18:36.390176
Computer Info: posix
Shortest path routing: {'A': inf, 'B': 0, 'C': 1, 'D': 3, 'E': 4, 'F': 6}
(.venv) harshsiddhapura@harshs-air Lab-2 %
OUTLINE
TIMELINE
main* Share Code Link Blackbox Search Terminal Output tabnine starter
Ln 60, Col 43 Spaces: 4 UTF-8 LF Python 3.9.6 ('.venv': venv) Blackbox Prettier


Python Labs
shortestPath.py
Module-6 > Lab-2 > shortestPath.py > ...
'A': {'B': 3, 'C': 2},
'B': {'C': 1, 'D': 5},
'C': {'D': 2, 'E': 6},
'D': {'E': 1, 'F': 4},
'E': {'F': 2},
'F': {}
}
# Implement the Bellman-Ford algorithm
Comment Code
def bellman_ford (graph, source):
# Step 1: initialize the distance to all nodes to infinity except the source node
distance = {node: float('infinity') for node in graph}
distance[source] = 0
# Step 2: relax edges repeatedly to find the shortest paths
for i in range(len(graph) - 1):
for u in graph:
for v in graph[u]:
# If the distance to v through u is shorter than the current distance to v, update it
if distance[u] + graph[u][v] < distance[v]:
distance[v] = distance[u] + graph[u][v]
# Step 3: check for negative-weight cycles
for u in graph:
for v in graph[u]:
if distance[u] + graph[u][v] < distance[v]:
raise ValueError('Graph contains a negative-weight cycle')
return distance
# Test the program by computing the shortest path routing between node A and node F
try:
shortest_path = bellman_ford(graph, 'C')
print('Shortest path routing:', shortest_path)
except ValueError as e:
print(e)
sys.exit(1)
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SEARCH TERMINAL OUTPUT GITLENS COMMENTS
User: harshsiddhapura
Time:2023-10-25 19:20:51.805683
Computer Info: posix
Shortest path routing: {'A': inf, 'B': inf, 'C': 0, 'D': 2, 'E': 3, 'F': 5}
(.venv) harshsiddhapura@harshs-air Lab-2 %
Share Code Link Blackbox Search Terminal Output tabnine starter
Ln 60, Col 43 Spaces: 4 UTF-8 LF Python 3.9.6 ('.venv': venv) Blackbox Prettier


Python Labs
shortestPath.py
Module-6 > Lab-2 > Section-B > shortestPath.py > ...
# Format log message
log_message = f"User: {username}\nTime:{current_time}\nComputer Info: {computer_info}"
# Print log message
print(log_message)
print_system_info()
import sys
# Define the graph using a dictionary of edges and their weights
graph = {
'A': {'B': 5, 'C': 10},
'B': {'D': 3, 'E': 8},
'C': {'F': 7},
'D': {'G': 2},
'E': {'G': 6},
'F': {'G': 9},
'G': {}
}
# Implement the Bellman-Ford algorithm
Comment Code
def bellman_ford (graph, source):
# Step 1: initialize the distance to all nodes to infinity except the source node
distance = {node: float('infinity') for node in graph}
distance[source] = 0
# Step 2: relax edges repeatedly to find the shortest paths
for i in range(len(graph) - 1):
for u in graph:
for v in graph[u]:
# If the distance to v through u is shorter than the current distance to v, update it
if distance[u] + graph[u][v] < distance[v]:
distance[v] = distance[u] + graph[u][v]
# Step 3: check for negative-weight cycles
for u in graph:
for v in graph[u]:
if distance[u] + graph[u][v] < distance[v]:
raise ValueError('Graph contains a negative-weight cycle')
return distance
# Test the program by computing the shortest path routing between node A and node F
try:
shortest_path = bellman_ford(graph, 'A')
print('Shortest path routing:', shortest_path)
except ValueError as e:
print(e)
sys.exit(1)
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SEARCH TERMINAL OUTPUT GITLENS COMMENTS
User: harshsiddhapura
Time:2023-10-25 19:25:46.894420
Computer Info: posix
Shortest path routing: {'A': 0, 'B': 5, 'C': 10, 'D': 8, 'E': 13, 'F': 17, 'G': 10}
(.venv) harshsiddhapura@harshs-air Section-B %
Share Code Link Blackbox Search Terminal Output tabnine starter
Ln 61, Col 43 Spaces: 4 UTF-8 LF Python 3.9.6 ('.venv': venv) Blackbox Prettier


Python Labs
shortestPath.py U
Module-6 > Lab-2 > Section-B > shortestPath.py > ...
# Format log message
log_message = f"User: {username}\nTime:{current_time}\nComputer Info: {computer_info}"
# Print log message
print(log_message)
print_system_info()
import sys
# Define the graph using a dictionary of edges and their weights
graph = {
'A': {'B': 5, 'C': 10},
'B': {'D': 3, 'E': 8},
'C': {'F': 7},
'D': {'G': 2},
'E': {'G': 6},
'F': {'G': 9},
'G': {}
}
# Implement the Bellman-Ford algorithm
def bellman_ford (graph, source):
# Step 1: initialize the distance to all nodes to infinity except the source node
distance = {node: float('infinity') for node in graph}
distance[source] = 0
# Step 2: relax edges repeatedly to find the shortest paths
for i in range(len(graph) - 1):
for u in graph:
for v in graph[u]:
# If the distance to v through u is shorter than the current distance to v, update it
if distance[u] + graph[u][v] < distance[v]:
distance[v] = distance[u] + graph[u][v]
# Step 3: check for negative-weight cycles
for u in graph:
for v in graph[u]:
if distance[u] + graph[u][v] < distance[v]:
raise ValueError('Graph contains a negative-weight cycle')
return distance
# Test the program by computing the shortest path routing between node A and node F
try:
shortest_path = bellman_ford(graph, 'B')
print('Shortest path routing:', shortest_path)
except ValueError as e:
print(e)
sys.exit(1)
User: harshsiddhapura
Time:2023-10-25 19:26:21.410426
Computer Info: posix
Shortest path routing: {'A': inf, 'B': 0, 'C': inf, 'D': 3, 'E': 8, 'F': inf, 'G': 5}
(.venv) harshsiddhapura@harshs-air Section-B %


Python Labs
shortestPath.py U
Module-6 > Lab-2 > Section-B > shortestPath.py > ...
# Format log message
log_message = f"User: {username}\nTime:{current_time}\nComputer Info: {computer_info}"
# Print log message
print(log_message)
print_system_info()
import sys
# Define the graph using a dictionary of edges and their weights
graph = {
'A': {'B': 5, 'C': 10},
'B': {'D': 3, 'E': 8},
'C': {'F': 7},
'D': {'G': 2},
'E': {'G': 6},
'F': {'G': 9},
'G': {}
}
# Implement the Bellman-Ford algorithm
def bellman_ford (graph, source):
# Step 1: initialize the distance to all nodes to infinity except the source node
distance = {node: float('infinity') for node in graph}
distance[source] = 0
# Step 2: relax edges repeatedly to find the shortest paths
for i in range(len(graph) - 1):
for u in graph:
for v in graph[u]:
# If the distance to v through u is shorter than the current distance to v, update it
if distance[u] + graph[u][v] < distance[v]:
distance[v] = distance[u] + graph[u][v]
# Step 3: check for negative-weight cycles
for u in graph:
for v in graph[u]:
if distance[u] + graph[u][v] < distance[v]:
raise ValueError('Graph contains a negative-weight cycle')
return distance
# Test the program by computing the shortest path routing between node A and node F
try:
shortest_path = bellman_ford(graph, 'C')
print('Shortest path routing:', shortest_path)
except ValueError as e:
print(e)
sys.exit(1)
User: harshsiddhapura
Time:2023-10-25 19:26:38.104096
Computer Info: posix
Shortest path routing: {'A': inf, 'B': inf, 'C': 0, 'D': inf, 'E': inf, 'F': 7, 'G': 16}
(.venv) harshsiddhapura@harshs-air Section-B %