LAB 1

Lab: "Help the Rat Escape the Underground Pipes"

Terry the rat is trapped in a vast underground pipe network. The pipes connect junctions, and each pipe has a travel cost representing how hard it is to crawl through (longer pipes or blocked paths cost more time). Your task is to write a program that helps Terry find a path from his starting junction to the cheese hidden somewhere the network. in

Your program should:

Explore paths where Terry always tries to go deeper into unexplored pipes before checking others.

Explore paths where Terry always checks all nearby pipes before moving deeper.

Find the path with the lowest total travel cost.

Improve the search for the lowest-cost path by using a map showing the straight-line distance from each junction to the cheese as a guide.

Input:

Number of junctions and pipes.

For each pipe: two junctions connected and the travel cost.

Coordinates of each junction (for estimating straight-line distance).

Starting junction and target junction.

Output:

The path Terry takes for each exploration strategy.

The total cost of each path

How many junctions Terry visited in each case.

INPUT

File Edit Format Run Options Window Help

```
import heapq

def ucs(graph, start, goal):
    # Check if the start node is the same as the goal node
    if start == goal:
        return [start], 0

    # Initialize the priority queue with the start node
    queue = [(0, start, [])]
    visited = set()

    while queue:
        # Extract the node with the lowest cost from the queue
        cost, node, path = heapq.heappop(queue)

        # If the node is the goal, return the path and cost
        if node == goal:
            return path + [node], cost

        # Skip if already visited
        if node in visited:
            continue

        # Mark the node as visited
        visited.add(node)

        # Explore the neighbors of the node
        for neighbor, edge_cost in graph.get(node, {}).items():
            if neighbor not in visited:
                new_cost = cost + edge_cost
                heapq.heappush(queue, (new_cost, neighbor, path + [node]))

    # If no path is found, return None
    return None, None
```

```
def main():
    # Get the number of junctions and pipes
    num_junctions = int(input("Enter the number of pipe junctions: "))
    num_pipes = int(input("Enter the number of pipes: "))

    # Initialize the graph
    graph = {}

    # Get the pipe connections and costs
    for _ in range(num_pipes):
        junction1, junction2, cost = input("Enter pipe connection (junction1 junction2 cost): ").split()
        cost = int(cost)

        # Add the pipe connection to the graph
        graph.setdefault(junction1, {})[junction2] = cost
        graph.setdefault(junction2, {})[junction1] = cost  # Assuming the graph is undirected

    # Get the rat's starting position and the location of the cheese
    start_node = input("Enter the rat's starting position: ")
    goal_node = input("Enter the location of the cheese: ")

    # Find the shortest path
    path, cost = ucs(graph, start_node, goal_node)

    if path is not None:
        print("Shortest path to the cheese:", ' -> '.join(path))
        print("Total distance to the cheese:", cost)
    else:
        print("No path to the cheese found.")

if __name__ == "__main__":
    main()
```

OUTPUT

```
>>>
= RESTART: C:/Users/varsh/AppData/Local/Programs/Python/Python313/jhdkghsjdb.py
Enter the number of pipe junctions: 5
Enter the number of pipes: 6
Enter pipe connection (junction1 junction2 cost): a b 2
Enter pipe connection (junction1 junction2 cost): b c 3
Enter pipe connection (junction1 junction2 cost): c d 1
Enter pipe connection (junction1 junction2 cost): d a 4
Enter pipe connection (junction1 junction2 cost): b d 2
Enter pipe connection (junction1 junction2 cost): c a 2
Enter the rat's starting position: a
Enter the location of the cheese: a
Shortest path to the cheese: a
Total distance to the cheese: 0
>>>
= RESTART: C:/Users/varsh/AppData/Local/Programs/Python/Python313/jhdkghsjdb.py
Enter the number of pipe junctions: 3
Enter the number of pipes: 2
Enter pipe connection (junction1 junction2 cost): a b 4
Enter pipe connection (junction1 junction2 cost): b c 3
Enter the rat's starting position: a
Enter the location of the cheese: c
No path to the cheese found.
```

```
= RESTART: C:/Users/varsh/AppData/Local/Programs/Python/Python313/jhdkghsjdb.py
Enter the number of pipe junctions: 4
Enter the number of pipes: 5
Enter pipe connection (junction1 junction2 cost): a b 2
Enter pipe connection (junction1 junction2 cost): b c 1
Enter pipe connection (junction1 junction2 cost): c d 3
Enter pipe connection (junction1 junction2 cost): d e 4
Enter pipe connection (junction1 junction2 cost): e f 3
Enter the rat's starting position: a
Enter the location of the cheese: e
No path to the cheese found.
>>>
= RESTART: C:/Users/varsh/AppData/Local/Programs/Python/Python313/jhdkghsjdb.py
Enter the number of pipe junctions: 6
Enter the number of pipes: 7
Enter pipe connection (junction1 junction2 cost): a b 2
Enter pipe connection (junction1 junction2 cost): a c 3
Enter pipe connection (junction1 junction2 cost): b d 1
Enter pipe connection (junction1 junction2 cost): b e 4
Enter pipe connection (junction1 junction2 cost): c f 5
Enter pipe connection (junction1 junction2 cost): d e 1
Enter pipe connection (junction1 junction2 cost): e f 1
Enter the rat's starting position: a
Enter the location of the cheese: f
Shortest path to the cheese: a -> b -> d -> e -> f
Total distance to the cheese: 5
>>>
```

CODE

```
import heapq
def ucs(graph, start, goal):
    # Check if the start node is the same as the goal node
    if start == goal:
        return [start], 0
 # Initialize the priority queue with the start node
    queue = [(0, start, [])]
    visited = set()
while queue:
        # Extract the node with the lowest cost from the queue
        cost, node, path = heapq.heappop(queue)
 # If the node is the goal, return the path and cost
        if node == goal:
            return path + [node], cost
 # Skip if already visited
        if node in visited:
            continue
 # Mark the node as visited
        visited.add(node)
# Explore the neighbors of the node
        for neighbor, edge_cost in graph.get(node, {}).items():
            if neighbor not in visited:
                new_cost = cost + edge_cost
                heapq.heappush(queue, (new_cost, neighbor, path + [node]))
 # If no path is found, return None
    return None, None
def main():
    # Get the number of junctions and pipes
    num_junctions = int(input("Enter the number of pipe junctions: "))
    num_pipes = int(input("Enter the number of pipes: "))
 # Initialize the graph
    graph = {}
```

```
 # Get the pipe connections and costs
    for _ in range(num_pipes):
        junction1, junction2, cost = input("Enter pipe connection (junction1 junction2 cost): ").split()
        cost = int(cost)
 # Add the pipe connection to the graph
        graph.setdefault(junction1, {})[junction2] = cost
        graph.setdefault(junction2, {})[junction1] = cost  # Assuming the graph is undirected
# Get the rat's starting position and the location of the cheese
    start_node = input("Enter the rat's starting position: ")
    goal_node = input("Enter the location of the cheese: ")
 # Find the shortest path
    path, cost = ucs(graph, start_node, goal_node)
 if path is not None:
        print("Shortest path to the cheese:", ' -> '.join(path))
        print("Total distance to the cheese:", cost)
    else:
        print("No path to the cheese found.")
if __name__ == "__main__":
    main()
```