Practical Work number 3:

Problem number 2, Manea Robert-Petrisor Group 914.

Dijkstra's algorithm is a popular graph traversal algorithm used to find the shortest path between two vertices in a graph with non-negative edge weights. Here's a short summary of the algorithm that uses the Dijkstra's algorithm backwards:

```
def backwardsDijkstra(self, s, t):
    q = PriorityQueue()
    dist = {}
    prev = {}
    q.put((0, t))
    dist[t] = 0
    found = False
    while not q.empty() and not found:
        _, x = q.get()  # Extract the vertex 'x' from the priority queue
        if x == s:
            found = True
            break
        for y in self.__Outbound[x]:  # Traverse outbound neighbors instead
of inbound
            if y not in dist.keys() or dist[x] + self.__Costs[(x, y)] <
dist[y]:
                dist[y] = dist[x] + self.__Costs[(x, y)]
                q.put((dist[y], y))  # Reversed the order of distance and
vertex in the priority queue
                prev[y] = x
    if not found:
        raise ValueError("No lowest cost walk exists between the given
vertices.")
    # Reconstruct the lowest cost walk using 'prev'
    path = [s]
    current = s
    while current != t:
        current = prev[current]
        path.append(current)
    return dist[s], list(reversed(path))
```

1. The algorithm takes a graph, a source vertex *s*, and a target vertex *t* as inputs.
2. It initializes a priority queue (q), a distance dictionary (*dist*), and a previous vertex dictionary (*prev*).
3. The algorithm starts from the target vertex and iteratively explores the graph backwards.
4. It updates the distances and previous vertices based on the shortest paths from the target vertex.
5. The algorithm stops when the source vertex is reached or when there are no more vertices to explore.
6. If the source vertex is reached, it reconstructs the lowest cost walk from the source to the target using the *prev* dictionary.

7. If the source vertex is not reached, it indicates that no lowest cost walk exists between the source and target vertices.
8. The algorithm returns the shortest distance from the target vertex to the source vertex and the lowest cost walk (if it exists).

In summary, the algorithm finds the shortest paths from the target vertex to other vertices in the graph, keeping track of the distances and previous vertices. It stops when the source vertex is reached or when there are no more vertices to explore. The algorithm then provides the shortest distance and, if available, the lowest cost walk from the source to the target vertex.

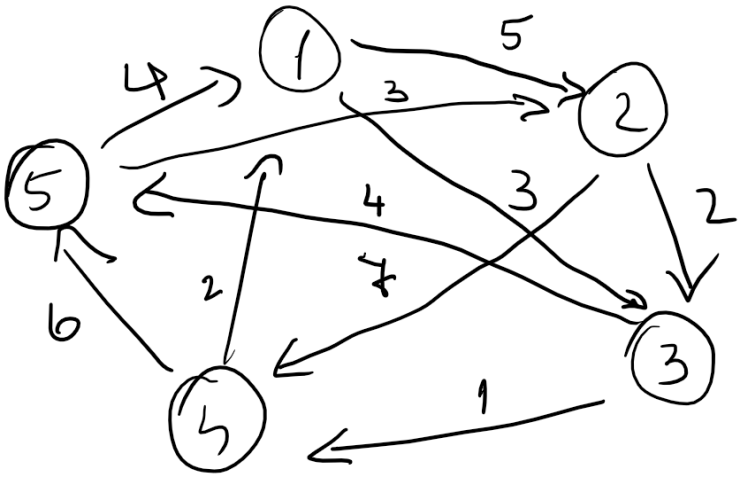
Manual Executions:

Graph 1 :

- has a walk

Vertices : 1, 2, 3, 4, 5

Edges : (1, 2, 5), (1, 3, 3), (2, 3, 2)
(2, 4, 4), (3, 4, 1), (3, 5, 4),
(4, 5, 6), (4, 1, 2), (5, 1, 4), (5, 2, 3)



s (source vertex) = 1
t (target vertex = 4
q (priority queue) = Empty
dist (dictionary) = {}
prev (dictionary) = { }
found (bool) = False

Iteration	q	dist	prev
1	[(0,4)]	{4:0}	{4:None}
2	[(0,4),(2,1)]	{4:0, 1:2}	{4:None, 1:4}
3	[(0,4),(2,1),(3,3)]	{4:0, 1:2, 3:3}	{4:None, 1:4, 3:1}
4	[(0,4),(2,1),(3,3), (5,5)]	{4:0, 1:2, 3:3, 5:5}	{4:None, 1:4, 3:1, 5:3}
5	[(0,4),(2,1),(3,3), (5,5),(6,2)]	{4:0, 1:2, 3:3, 5:5, 2:6}	{4:None, 1:4, 3:1, 5:3, 2:5}
6	[(0,4),(2,1),(3,3), (5,5),(6,2),(8,4)]	{4:0, 1:2, 3:3, 5:5, 2:6, 4:8}	{4:None, 1:4, 3:1, 5:3, 2:5, 4:2}
7	[(0,4),(2,1),(3,3), (5,5),(6,2),(8,4), (9,5)]	{4:0, 1:2, 3:3, 5:5, 2:6, 4:8, 5:9}	{4:None, 1:4, 3:1, 5:3, 2:5, 4:2, 5:4}

Since 'e' is (4) is source vertex => found is set to True.

Final result : dist [1] : 2
path : [1, 4]

Graph 2
- Bar no walls

Vertices : 1, 2, 3, 4, 5

Edges : (1, 2, 5), (1, 3, 3), (2, 3, 2), (2, 4, 7), (3, 4, 1), (3, 5, 4), (4, 5, 6), (4, 1, 2), (5, 1, 4), (5, 2, 3)

s (source vertex) = 1
t (target vertex = 5
q (priority queue) = Empty
dist (dictionary) = {}
prev (dictionary) = {}
found (bool) = False

Iteration	q	dist	prev
1	[(0,5)]	{5:0}	{5:None}
2	[(0,5),(4,1)]	{5:0, 1:4}	{5:None, 1:5}
3	[(0,5),(4,1),(5,2)]	{5:0, 1:4, 2:5}	{5:None, 1:5, 2:1}
4	[(0,5),(4,1),(5,2),(9,3)]	{5:0, 1:4, 2:5, 3:9}	{5:None, 1:5, 2:1, 3:2}
5	[(0,5),(4,1),(5,2),(9,3),(13,4)]	{5:0, 1:4, 2:5, 3:9, 4:13}	{5:None, 1:5, 2:1, 3:2, 4:3}
6	[(0,5),(4,1),(5,2),(9,3),(13,4),(15,1)]	{5:0, 1:4, 2:5, 3:9, 4:13, 1:15}	{5:None, 1:5, 2:1, 3:2, 4:3, 1:4}

't' is now 5, source vertex → found is set to True

Since there is no walk, the ValueError is raised