Problem Solving C++ Lab
Week 6

Q1]

```
def treasure_hunt(graph, start, end):
    def dfs(current, path):
        if current == end:
            paths.append(path)
            return
        visited[current] = True
        for neighbor in graph[current]:
            if not visited[neighbor]:
                dfs(neighbor, path + [neighbor])
        visited[current] = False

    paths = []
    visited = [False] * len(graph)
    dfs(start, [start])
    return paths

graph1 = [[1,2], [0,3], [0,3], [1,2]]
start1 = 0
end1 = 3
result1 = treasure_hunt(graph1, start1, end1)
print(result1)
```

Q2]

```
from collections import deque

def message_spread(graph, source):
    queue = deque([source])
    visited = set([source])
    order = []
    while queue:
        current = queue.popleft()
        order.append(current)

        for neighbor in graph[current]:
            if neighbor not in visited:
                visited.add(neighbor)
                queue.append(neighbor)

    return order

graph2 = [[1,2], [0,2], [0,1,3], [2]]
source2 = 0
result2 = message_spread(graph2, source2)
print(result2)
```

Q3]

```
def time_travel_paradox(graph):
    def dfs(current, visited, stack):
        visited[current] = True
        stack[current] = True

        for neighbor in graph[current]:
            if not visited[neighbor]:
                if dfs(neighbor, visited, stack):
                    return True
            elif stack[neighbor]:
                return True

        stack[current] = False
        return False
    num_events = len(graph)
    visited = [False] * num_events
    stack = [False] * num_events

    for event in range(num_events):
        if not visited[event]:
            if dfs(event, visited, stack):
                return False

    return True


graph3 = [[1], [2], []]
result3 = time_travel_paradox(graph3)
print(result3)
```

Q4]

```
def explorers_mystic_land(graph, start, end):
    def dfs(current, path):
        if current == end:
            paths.append(path)
            return
        for neighbor in graph[current]:
            dfs(neighbor, path + [neighbor])

    paths = []
    dfs(start, [start])
    return paths


graph4 = [[1,2], [3], [3], []]
```

```
start4 = 0
end4 = 3
result4 = explorers_mystic_land(graph4, start4, end4)
print(result4)
```

Q5]

```
import heapq

def shortest_path_to_enlightenment(graph, source, destination):
    pq = [(0, source)]
    distances = {vertex: float('infinity') for vertex in graph}
    distances[source] = 0

    while pq:
        current_distance, current_vertex = heapq.heappop(pq)

        if current_distance > distances[current_vertex]:
            continue

        for neighbor, weight in graph[current_vertex]:
            distance = current_distance + weight

            if distance < distances[neighbor]:
                distances[neighbor] = distance
                heapq.heappush(pq, (distance, neighbor))

    return distances[destination]

graph5 = [[1, 2, 24], [0, 2, 5], [0, 1, 6]]
source5 = 0
destination5 = 2
```

```
result5 = shortest_path_to_enlightenment(graph5, source5,
destination5)
print(result5)
```

Q6]

```
import heapq

def rebuilding_kingdom(graph):
    pq = [(0, 0)]
    visited = set()
    total_cost = 0

    while pq:
        cost, current_vertex = heapq.heappop(pq)

        if current_vertex in visited:
            continue

        visited.add(current_vertex)
        total_cost += cost

        for neighbor, road_cost in graph[current_vertex]:
            if neighbor not in visited:
                heapq.heappush(pq, (road_cost, neighbor))

    return total_cost

graph6 = [[1, 2, 3], [0, 2, 2], [0, 1, 1]]
result6 = rebuilding_kingdom(graph6)
print(result6)
```