Djikstra.py

```
def buildPath(self, data: dict, start: int, end: int):
    path = []

    current = start
    path.append(start)

    while end != current:
        current = data[current]
        path.append(current)

    return path


def dijkstra(self, start, end):
    # Initialise the priority que, the dict for the next vertex in the path and one for the cost for each path from
    # the end vertex to the corresponding vertex
    pq = queue.PriorityQueue()
    next = {}
    dist = {}

    # Add in the priority que the end vertex and the cost 0 since we start from it
    pq.put((0, end))

    # Add the vertex we start from in the dict responsible for the dist of the paths
    dist[end] = 0

    # Initialise found with false
    found = False

    # We loop until the priority que is empty (case in which every possible path was checked)
    # or we found a path to the start vertex, case in which it is the shortest possible path
    # because at each step we choose the cheapest path when we decide which vertex to visit next
    while not pq.empty() and not found:

        # Get from the priority que the next in line element
        current = pq.get()

        # Iterate trough all it's inbound neighbors(since we go in a backwards order)
        for neighbor in self.getInbound(current[1]):

            # If the neighbor is not in inside the dist dict we add it and it's corresponding cost
            # or we found a cheaper way to reach the same node we change the cost to the new cheaper one
            # and update the path to the node as well
            if neighbor not in dist.keys() or \
                    dist[current[1]] + self.retrieve_info(neighbor, current[1]) < dist[neighbor]:
                dist[neighbor] = dist[current[1]] + self.retrieve_info(neighbor, current[1])
                pq.put((dist[neighbor], neighbor))
                next[neighbor] = current[1]

        # If the start vertex was reached we set the found flag to true
        if current[1] == start:
            found = True

    # If start is in the next dictionary it means there is a path to it
    # else it means there is no path to the start node so we return False in the UI
    # and a corresponding message will be printed
    if start not in next.keys():
        return False, 0, 0
    else:
        return True, self.buildPath(next, start, end), dist[start]
```