HW2 report

Part I. Implementation (6%):

- Part 1

```
    '''
    line 17 ~ 26 : Reads the file into Edges, every start node have a list, inside are end node and distance, and let start
    node, end node to be integer
    line 28 ~ 31 : Prepare for bfs, use list to simulate queue to store node we want to visit, visited stores node have been
    visited, previous_node stores current node's parent node and the distance to parent node
    line 32 ~ 54 : Implementation of bfs, let non-visited node in queue, until end is visited
    line 44 ~ 50 : Calculate path and distance, from end to start to rebuild the path, and start should be taken careful
    (line 48 ~ 49), and reverse the path
    '''
    Edges = {}
    with open(edgeFile, newline = '') as Edgefile:
        edges = csv.reader(Edgefile)
        for edge in edges:
            if edge[0] == 'start':
                continue
            edge[0], edge[1] = int(edge[0]), int(edge[1])
            if edge[0] not in Edges:
                Edges[edge[0]] = []
            Edges[edge[0]].append([edge[1], float(edge[2])])

    queue = [start]
    visited = set()
    visited.add(start)
    previous_node = {}
    while len(queue) > 0:
        now_node = queue.pop(0)
        if now_node not in Edges:
            continue
            for next_node in Edges[now_node]:
                if next_node[0] not in visited:
                    previous_node[next_node[0]] = [now_node, next_node[1]]
                    if next_node[0] == end:
                        i = end
                        path = [end]
                        dis = 0.0

                        while previous_node[i][0] != start:
                            dis += previous_node[i][1]
                            path.append(previous_node[i][0])
                            i = previous_node[i][0]
                        dis += previous_node[i][1]
                        path.append(start)
                        path.reverse()

                        return path, dis, len(visited)
                    visited.add(next_node[0])
                    queue.append(next_node[0])
```

- Part 2

```
    '''
    line 17 ~ 26 : Reads the file into Edges, every start node have a list, inside are end node and distance, and let start
    node, end node to be integer
    line 28 ~ 31 : Prepare for dfs, use list to simulate stack to store node we want to visit, visited stores node have been
    visited, previous_node stores current node's parent node and the distance to parent node
    line 32 ~ 54 : Implementation of dfs, let non-visited node in stack, until end is visited
    line 44 ~ 50 : Calculate path and distance, from end to start to rebuild the path, and start should be taken careful
    (line 48 ~ 49), and reverse the path
    '''
    Edges = {}
    with open(edgeFile, newline = '') as Edgefile:
        edges = csv.reader(Edgefile)
        for edge in edges:
            if edge[0] == 'start':
                continue
            edge[0], edge[1] = int(edge[0]), int(edge[1])
            if edge[0] not in Edges:
                Edges[edge[0]] = []
            Edges[edge[0]].append([edge[1], float(edge[2])])

    stack = [start]
    visited = set()
    visited.add(start)
    previous_node = {}
    while len(stack) > 0:
        now_node = stack.pop()
        if now_node not in Edges:
            continue
        for next_node in Edges[now_node]:
            if next_node[0] not in visited:
                previous_node[next_node[0]] = [now_node, next_node[1]]
                if next_node[0] == end:
                    i = end
                    path = [end]
                    dis = 0.0

                    while previous_node[i][0] != start:
                        dis += previous_node[i][1]
                        path.append(previous_node[i][0])
                        i = previous_node[i][0]
                    dis += previous_node[i][1]
                    path.append(start)
                    path.reverse()

                    return path, dis, len(visited)
                visited.add(next_node[0])
                stack.append(next_node[0])
```

- Part 3

```
'''
line 21 ~ 30 : Reads the file into Edges, every start node have a list, inside are end node and distance, and let start node, end node to be integer
line 32 ~ 36 : Prepare for ucs, use list to simulate priority queue to store node we want to visit, visited stores node have been visited, previous_node stores current node's parent node, dis_to_start stores the distance from current node to start node
line 37 ~ 70 : Implementation of ucs, let non-visited node in queue, until end is visited. In order to simulate priority queue, I sort the list by the distance from current node to start
line 44 ~ 48 : Calculate path, from end to start to rebuild the path, and start should be taken careful (line 47), and reverse the path
line 55 ~ 69 : Check next node whether in queue already, if it's not in queue, just add it, if in queue already, update the dis_to_start and previous_node
'''
Edges = {}
with open(edgeFile, newline = '') as Edgefile:
    edges = csv.reader(Edgefile)
    for edge in edges:
        if edge[0] == 'start':
            continue
        edge[0], edge[1] = int(edge[0]), int(edge[1])
        if edge[0] not in Edges:
            Edges[edge[0]] = []
        Edges[edge[0]].append([edge[1], float(edge[2])])
    queue = [[0, start]]
    visited = set()
    previous_node = {}
    dis_to_start = {}
    dis_to_start[start] = 0
    while len(queue) > 0:
        now_node = queue.pop(0)
        visited.add(now_node[1])
        if now_node[1] == end:
            i = end
            path = [end]

            while previous_node[i] != start:
                path.append(previous_node[i])
                i = previous_node[i]
            path.append(start)
            path.reverse()

            return path, dis_to_start[end], len(visited)

        if now_node[1] not in Edges:
            continue
```

```
    for next_node in Edges[now_node[1]]:
        if next_node[0] not in visited:
            flag = False
            for node in queue:
                if node[1] == next_node[0]:
                    flag = True
                    if dis_to_start[next_node[0]] > next_node[1] + now_node[0]:
                        dis_to_start[next_node[0]] = next_node[1] + now_node[0]
                        previous_node[next_node[0]] = now_node[1]
                        node[0] = dis_to_start[next_node[0]]
                        break
            if flag:
                continue
            previous_node[next_node[0]] = now_node[1]
            dis_to_start[next_node[0]] = now_node[0] + next_node[1]
            queue.append([dis_to_start[next_node[0]], next_node[0]])
    queue.sort()
```

- Part 4

```
'''
line 23 ~ 32 : Reads the file into Edges, every start node have a list, inside are end node and distance, and let start
node, end node to be integer
line 34 ~ 46 : Reads the file into heuristic_value, every node stores its distance to end node
line 48 ~ 53 : Prepare for astar, use list to simulate priority queue to store node we want to visit, visited stores node
have been visited, previous_node stores current node's parent node, dis_to_start stores the distance from current node
to start node
line 54 ~ 81 : Implementation of astar, let non-visited node in queue, until end is visited. In order to simulate priority
queue, I sort the list by the distance from current node to start plus heuristic value
line 60 ~ 64 : Calculate path, from end to start to rebuild the path, and start should be taken careful
(line 63), and reverse the path
line 71 ~ 79 : Check next node whether in queue and visited already, if it's not in queue and not visited, just add it,
else update the dis_to_start and previous_node
'''
Edges = {}
with open(edgeFile, newline = '') as Edgefile:
    edges = csv.reader(Edgefile)
    for edge in edges:
        if edge[0] == 'start':
            continue
        edge[0], edge[1] = int(edge[0]), int(edge[1])
        if edge[0] not in Edges:
            Edges[edge[0]] = []
        Edges[edge[0]].append([edge[1], float(edge[2])])
```

```
    heuristic_value = {}
    with open(heuristicFile, newline = '') as Heuristic:
        heuristics = csv.reader(Heuristic)
        for heuristic in heuristics:
            if heuristic[0] == 'node':
                continue
            heuristic[0] = int(heuristic[0])
            if end == 1079387396:
                heuristic_value[heuristic[0]] = float(heuristic[1])
            elif end == 1737223506:
                heuristic_value[heuristic[0]] = float(heuristic[2])
            else:
                heuristic_value[heuristic[0]] = float(heuristic[3])

    open_list = [[heuristic_value[start], start]]
    previous_node = {}
    dis_to_start = {}
    visited = set()
    visited.add(start)
    dis_to_start[start] = 0
while len(open_list) > 0:
    now_node = open_list.pop(0)
    if now_node[1] == end:
        i = end
        path = [end]

        while previous_node[i] != start:
            path.append(previous_node[i])
            i = previous_node[i]
        path.append(start)
        path.reverse()

        return path, dis_to_start[end], len(visited)

    if now_node[1] not in Edges:
        continue
    for next_node in Edges[now_node[1]]:
        if next_node[0] not in visited:
            if next_node[0] not in open_list:
                previous_node[next_node[0]] = now_node[1]
                dis_to_start[next_node[0]] = dis_to_start[now_node[1]] + next_node[1]
                open_list.append([dis_to_start[next_node[0]] + heuristic_value[next_node[0]], next_node[0]])
        else:
            if dis_to_start[next_node[0]] > dis_to_start[now_node[1]] + next_node[1]:
                dis_to_start[next_node[0]] = dis_to_start[now_node[1]] + next_node[1]
                previous_node[next_node[0]] = now_node[1]
        visited.add(next_node[0])
    open_list.sort()
```

- Part 6

```
'''
line 26 ~ 37 : Reads the file into Edges, every start node have a list, inside are end node and time to next node, and
let start node, end node to be integer, and store the maximum speed of all speed
line 39 ~ 53 : Reads the file into heuristic_value, every node stores (its distance to end node / maximum speed) to be
the new heuristic value. Because its direct distance to end node must less than the distance through edges, and its speed
must less than maximum speed. Time = distance / speed, so this is an admissible heuristic function because it doesn't
overestimate the cost of the minimum cost path from a node to the end node.
line 55 ~ 60 : Prepare for astar, use list to simulate priority queue to store node we want to visit, visited stores node
have been visited, previous_node stores current node's parent node, time_to_start stores the time from current node
to start node
line 61 ~ 88 : Implementation of astar, let non-visited node in queue, until end is visited. In order to simulate priority
queue, I sort the list by the time from current node to start plus heuristic value
line 67 ~ 71 : Calculate path, from end to start to rebuild the path, and start should be taken careful
(line 70), and reverse the path
line 78 ~ 86 : Check next node whether in queue and visited already, if it's not in queue and not visited, just add it,
else update the time_to_start and previous_node
'''
Edges = {}
max_speed = 0
with open(edgeFile, newline = '') as Edgefile:
    edges = csv.reader(Edgefile)
    for edge in edges:
        if edge[0] == 'start':
            continue
        edge[0], edge[1] = int(edge[0]), int(edge[1])
        if edge[0] not in Edges:
            Edges[edge[0]] = []
        Edges[edge[0]].append([edge[1], float(edge[2]) / (float(edge[3]) * 5 / 18)])
        max_speed = max(max_speed, float(edge[3]) * 5 / 18)
    heuristic_value = {}
    with open(heuristicFile, newline = '') as Heuristic:
        heuristics = csv.reader(Heuristic)
        for heuristic in heuristics:
            if heuristic[0] == 'node':
                continue
            heuristic[0] = int(heuristic[0])
            if end == 1079387396:
                heuristic_value[heuristic[0]] = float(heuristic[1])
            elif end == 1737223506:
                heuristic_value[heuristic[0]] = float(heuristic[2])
            else:
                heuristic_value[heuristic[0]] = float(heuristic[3])

            heuristic_value[heuristic[0]] /= max_speed

    open_list = [[heuristic_value[start], start]]
    previous_node = {}
    time_to_start = {}
    visited = set()
    visited.add(start)
    time_to_start[start] = 0
```

```python
while len(open_list) > 0:
    now_node = open_list.pop(0)
    if now_node[1] == end:
        i = end
        path = [end]

        while previous_node[i] != start:
            path.append(previous_node[i])
            i = previous_node[i]
        path.append(start)
        path.reverse()

        return path, time_to_start[end], len(visited)

    if now_node[1] not in Edges:
        continue
    for next_node in Edges[now_node[1]]:
        if next_node[0] not in visited:
            if next_node[0] not in open_list:
                previous_node[next_node[0]] = now_node[1]
                time_to_start[next_node[0]] = time_to_start[now_node[1]] + next_node[1]
                open_list.append([time_to_start[next_node[0]] + heuristic_value[next_node[0]], next_node[0]])
            else:
                if time_to_start[next_node[0]] > time_to_start[now_node[1]] + next_node[1]:
                    time_to_start[next_node[0]] = time_to_start[now_node[1]] + next_node[1]
                    previous_node[next_node[0]] = now_node[1]
            visited.add(next_node[0])
    open_list.sort()
```

Part II. Results & Analysis (12%):

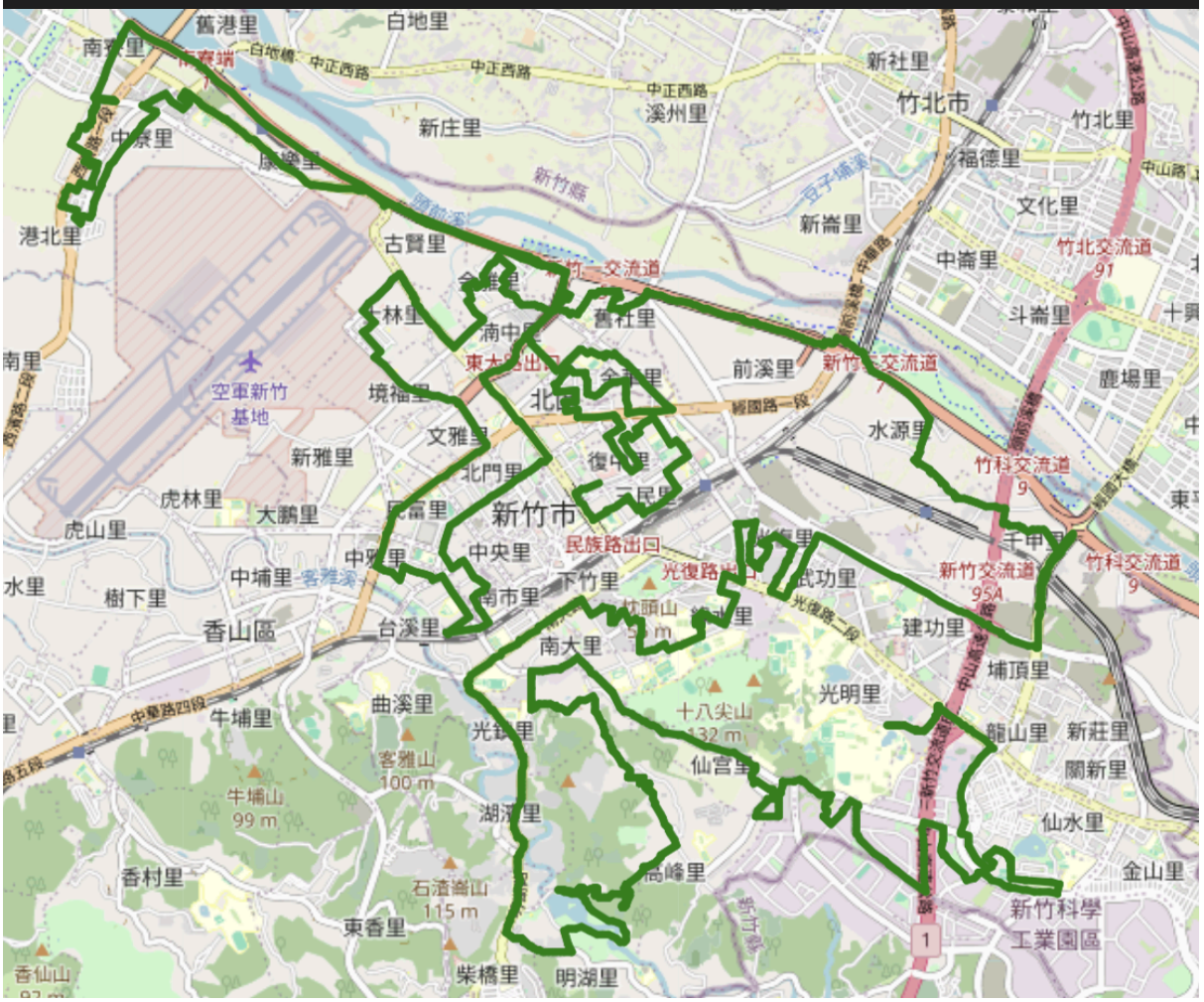
Test 1: From National Yang Ming Chiao Tung University (ID: 2270143902) to Big City Shopping Mall (ID: 1079387396)

- BFS

```
The number of nodes in the path found by BFS: 88
Total distance of path found by BFS: 4978.881999999998 m
The number of visited nodes in BFS: 4273
```

- DFS(stack)

```
The number of nodes in the path found by DFS: 1718
Total distance of path found by DFS: 75504.3150000001 m
The number of visited nodes in DFS: 5235
```



- UCS

```
The number of nodes in the path found by UCS: 89
Total distance of path found by UCS: 4367.881 m
The number of visited nodes in UCS: 5086
```

- A*

```
The number of nodes in the path found by A* search: 89
Total distance of path found by A* search: 4367.881 m
The number of visited nodes in A* search: 306
```

- A*(time)

```
The number of nodes in the path found by A* search: 89
Total second of path found by A* search: 320.87823163083164 s
The number of visited nodes in A* search: 2006
```

Test 2: **From Hsinchu Zoo (ID: 426882161) to COSTCO Hsinchu Store (ID: 1737223506)**

- BFS

```
The number of nodes in the path found by BFS: 60
Total distance of path found by BFS: 4215.521000000001 m
The number of visited nodes in BFS: 4606
```

- DFS(stack)

```
The number of nodes in the path found by DFS: 930
Total distance of path found by DFS: 38752.307999999895 m
The number of visited nodes in DFS: 9615
```

- UCS

```
The number of nodes in the path found by UCS: 63
Total distance of path found by UCS: 4101.84 m
The number of visited nodes in UCS: 7213
```

- A*

```
The number of nodes in the path found by A* search: 63
Total distance of path found by A* search: 4101.84 m
The number of visited nodes in A* search: 1361
```

- A*(time)

```
The number of nodes in the path found by A* search: 63
Total second of path found by A* search: 304.44366343603014 s
The number of visited nodes in A* search: 3067
```

Test 3：From National Experimental High School At Hsinchu Science Park (ID: 1718165260) to Nanliao Fighing Port (ID: 8513026827)

- BFS

```
The number of nodes in the path found by BFS: 183
Total distance of path found by BFS: 15442.394999999995 m
The number of visited nodes in BFS: 11241
```

- DFS(stack)

```
The number of nodes in the path found by DFS: 900
Total distance of path found by DFS: 39219.993000000024 m
The number of visited nodes in DFS: 2493
```

- UCS

```
The number of nodes in the path found by UCS: 288
Total distance of path found by UCS: 14212.412999999997 m
The number of visited nodes in UCS: 11926
```

- A*

```
The number of nodes in the path found by A* search: 288
Total distance of path found by A* search: 14212.412999999997 m
The number of visited nodes in A* search: 7198
```

- A*(time)

```
The number of nodes in the path found by A* search: 209
Total second of path found by A* search: 779.527922836848 s
The number of visited nodes in A* search: 8613
```

Observation and Analysis :

- UCS and A* have the optimal result of the minimum distance path from start to end, but A* is more efficient due to the number of visited node are less.
- BFS has better performance than DFS, but in test3, DFS has fewer visited nodes than BFS. I think that is because DFS depends on adjacent list more than BFS.
- My heuristic function is admissible because it doesn't overestimate the cost of the minimum cost path from a node to an end node. My method is dividing the direct distance from the current node to the end node by the maximum speed, so the cost must be less than the actual cost.
- In test1 and test2, A*(distance) and A*(time) has the same result path, but they are different in test3. I think A*(time) has a more valid result because it considers the speed limit of each edge. But A*(distance) doesn't, its result may not be the optimal path in reality.

Part III. Question Answering (12%):

1. **Please describe a problem you encountered and how you solved it.**
   When I finished the BFS and DFS and wanted to draw the path on the map, but the path didn't appear, only the map. After hours of debugging, I found that if node id is stored in string format, it wouldn't appear on the map. So I finally converted all node id into integer, and the path appeared.
2. **Besides speed limit and distance, could you please come up with another attribute that is essential for route finding in the real world? Please explain the rationale**
   I think the traffic light is also an important attribute in route finding in the real world. Because you can't just always not stop along the path, it must have some intersection. You need to wait for the traffic light to be green. So I think if we have that kind of information or data, the optimal route must be more accurate in the real world.
3. **As mentioned in the introduction, a navigation system involves mapping, localization, and route finding. Please suggest possible solutions for mapping and localization components?**
   For mapping, we can collect the pictures obtained by satellites, and utilize some techniques to identify each object in the pictures, such as roads, buildings. So that we can construct a map.
   For localization, we can use GPS to know the current position of the smartphone or other vehicles with the user.
4. **The estimated time of arrival (ETA) is one of the features of Uber Eats. To provide accurate estimates for users, Uber Eats needs to dynamically update ETA based on their mechanism. Please define a dynamic heuristic equation for ETA and explain the rationale of your design. Hint: You can consider meal prep time, delivery priority, multiple orders, etc.**
   ETA = meal prep time + actual delivery time + some other delivery time.
   Meal prep time is just the time for preparing the meal you order. Actual delivery time is the time the food delivery driver spends from his/her position to your assigned position. Some other delivery time is determined by the multiple orders and delivery priority. If there is only your order, then this time is 0. If there are multiple orders, it will consider the delivery priority. People who have higher priority will receive his/her meal first. If your priority is low, you need to wait longer for your meal. Otherwise, you can have faster delivery with high priority