Homework 2: Route Finding

109550135 范恩宇

Part I. Implementation :

↓ ↓ Code and explanation of “ BFS ” ↓ ↓

```python
import csv
edgeFile = 'edges.csv'

def bfs(start, end):
    # Begin your code (Part 1)

    """
    First , build "graph" with data in the csv file . While queue is not empty ,
    put nodes that are linked to vertices and not visited on queue's top . Also
    in the while loop , record parents of vertex for tracing . When the route
    encounters "end" , return the path , distance and number of visited nodes.
    """

    graph = {}
    with open(edgeFile, newline='') as f:
        rows = csv.DictReader(f)
        for line in rows:
            tmp = []
            if int(line['start']) in graph:
                tmp.extend(graph[int(line['start'])])
                tmp.append((int(line['end']),float(line['distance'])))
                graph[int(line['start'])] = tmp
            else:
                tmp.append((int(line['end']),float(line['distance'])))
                graph[int(line['start'])] = tmp

    visited = [start]
    queue = [start]
    parent = {}
    dist_dict = {}
    num_visited = 0
```

```python
    while queue :
        node = queue[0]
        queue.pop(0)

        if node in graph:
            vertice = graph[node]
            for i in vertice:
                if i[0] not in visited:
                    queue.append(i[0])
                    visited.append(i[0])
                    parent[i[0]] = node
                    dist_dict[i[0]] = i[1]

                    if i[0] == end:

                        dist = 0
                        path = []
                        tgt = end

                        while tgt != start :
                            dist += dist_dict[tgt]
                            path.append(tgt)
                            tgt = parent[tgt]

                        path.append(start)
                        num_visited = len(visited)
                        return path,dist,num_visited
    # End your code (Part 1)
```

↓ ↓ Code and explanation of “ DFS ” ↓ ↓

```python
import csv
edgeFile = 'edges.csv'

def dfs(start, end):
    # Begin your code (Part 2)
    """
    First , build "graph" with data in the csv file . While stack is not empty ,
    put nodes that are near to vertices and not visited on stack's top ,
    however , stack pops out the last node . Also in the while loop , record
    parents of vertex for tracing . When the route encounters "end" , return
    the path , distance and number of visited nodes.
    """
    graph = {}
    with open(edgeFile, newline='') as f:
        rows = csv.DictReader(f)
        for line in rows:
            tmp = []
            if int(line['start']) in graph:
                tmp.extend(graph[int(line['start'])])
                tmp.append((int(line['end']),float(line['distance'])))
                graph[int(line['start'])] = tmp
            else:
                tmp.append((int(line['end']),float(line['distance'])))
                graph[int(line['start'])] = tmp

    visited = [start]
    stack = [start]
    parent = {}
    dist_dict = {}
    num_visited = 0

    while stack:
        node = stack.pop()
        visited.append(node)

        if node in graph:
            vertices = graph[node]
            for i in vertices:
                if i[0] not in visited:
                    parent[i[0]] = node
                    dist_dict[i[0]] = i[1]
                    stack.append(i[0])
                if i[0] == end:
                    dist = 0
                    path = []
                    tgt = end

                    while tgt != start :
                        dist += dist_dict[tgt]
                        path.append(parent[tgt])
                        tgt = parent[tgt]

                    path.append(start)
                    num_visited = len(visited)

                    return path,dist,num_visited

    # End your code (Part 2)
```

↓ ↓ Code and explanation of “ UCS ” ↓ ↓

```python
import csv
from queue import PriorityQueue
edgeFile = 'edges.csv'

def ucs(start, end):
    # Begin your code (Part 3)
    """
    First , build "graph" with data in the csv file ,then put "start " into
    priority queue "pq". While "pq" is not empty , put the node near the
    vertex into "pq" and record distance between the node mentioned before
    and "start" into "dist_dict" and replace the value if receiving  smaller one.
    When the route encounters "end" , return the path , distance and number
    of visited nodes.
    """
    graph = {}
    with open(edgeFile, newline='') as f:
        rows = csv.DictReader(f)
        for line in rows:
            tmp = []
            if int(line['start']) in graph:
                tmp.extend(graph[int(line['start'])])
                tmp.append((int(line['end']),float(line['distance'])))
                graph[int(line['start'])] = tmp
            else:
                tmp.append((int(line['end']),float(line['distance'])))
                graph[int(line['start'])] = tmp


    visited = [start]
    dis_dict = {start : 0}
    parent = {}
    pq = PriorityQueue()
    pq.put((0,start))
```

```python
    pq.put((0,start))
    num_visited = 0

    while not pq.empty():
        tmp = []
        node = pq.get()

        if node[1] in graph:
            tmp.extend(graph[node[1]])
            for i in tmp:
                if i[0] not in visited or dis_dict[i[0]] > node[0] + i[1]:
                    if i[0] == end:
                        dist = 0
                        path = []
                        tgt = end
                        parent[tgt] = node[1]

                        while tgt != start :
                            path.append(parent[tgt])
                            tgt = parent[tgt]

                        dist = node[0] + i[1]
                        path.append(start)
                        num_visited = len(visited)

                        return path,dist,num_visited

                    else:
                        visited.append(i[0])
                        dis_dict[i[0]] = node[0] + i[1]
                        parent[i[0]] = node[1]
                        pq.put((node[0]+i[1],i[0]))
    # End your code (Part 3)
```

↓ ↓ Code and explanation of “ A* ” ↓ ↓

```python
import csv
from queue import PriorityQueue
edgeFile = 'edges.csv'
heuristicFile = 'heuristic.csv'

def astar(start, end):
    # Begin your code (Part 4)
    """
    First , build "graph" and "heuri " with data in 2 csv files ,then put
    "start " into priority queue "pq" . While "pq" is not empty , put
    the node near the vertex into "pq" and record the distance between the
    node mentioned before and "start" into "dist_dict" and replace the
    value if receiving  smaller one . In addition , sum the distance and
    the "heuri" to compare . When the route encounters "end" , return the
    path , distance and number of visited nodes.

    """
    graph = {}
    heuri = {}

    with open(edgeFile, newline='') as f:
        rows = csv.DictReader(f)
        for line in rows:
            tmp = []
            if int(line['start']) in graph:
                tmp.extend(graph[int(line['start'])])
                tmp.append((int(line['end']),float(line['distance'])))
                graph[int(line['start'])] = tmp
            else:
                tmp.append((int(line['end']),float(line['distance'])))
                graph[int(line['start'])] = tmp
```

```python
    with open(heuristicFile, newline='') as f2:
        rows = csv.DictReader(f2)
        for line in rows:
            heuri[int(line['node'])] = float(line[str(end)])

    visited = [start]
    dist_dict = {start : 0}
    parent = {}
    pq = PriorityQueue()
    pq.put((0,start))
    num_visited = 0
```

```python
    while not pq.empty():
        tmp = []
        node = pq.get()

        if node[1] in graph:
            tmp.extend(graph[node[1]])

            for i in tmp:
                if i[0] not in visited or dist_dict[i[0]] > dist_dict[node[1]] + i[1]:
                    if i[0] == end:
                        dist = 0
                        path = []
                        tgt = end
                        parent[tgt] = node[1]

                        while tgt != start :
                            path.append(parent[tgt])
                            tgt = parent[tgt]

                        dist = dist_dict[node[1]] + i[1]
                        path.append(start)
                        num_visited = len(visited)

                        return path,dist,num_visited

                    else:
                        visited.append(i[0])
                        dist_dict[i[0]] = dist_dict[node[1]] + i[1]
                        parent[i[0]] = node[1]
                        pq.put((heuri[i[0]] + dist_dict[node[1]]+i[1],i[0]))

    # End your code (Part 4)
```

↓ ↓ Code and explanation of “ A*(time) ” ↓ ↓

```python
import csv
from queue import PriorityQueue
edgeFile = 'edges.csv'
heuristicFile = 'heuristic.csv'

def astar_time(start, end):

    # Begin your code (Part 6)
    """
    Basically the same thing like "astar.py" , but I used the biggest
    speed limit to divide "heuri" . In addition , I divide each distance
    by speed limit to get the time , not distance .
    """

    max_lmt = 0
    graph = {}
    heuri = {}
    with open(edgeFile, newline='') as file:
        rows = csv.DictReader(file)
        for line in rows:
            tmp = []
            if int(line['start']) in graph:
                tmp.extend(graph[int(line['start'])])
                tmp.append((int(line['end']),float(line['distance']),float(line['speed limit'])))
                max_lmt = max(max_lmt, float(line['speed limit']))
                graph[int(line['start'])]=tmp
            else:
                tmp.append((int(line['end']),float(line['distance']),float(line['speed limit'])))
                max_lmt = max(max_lmt, float(line['speed limit']))
                graph[int(line['start'])] = tmp

    with open(heuristicFile, newline='') as f2:
        rows = csv.DictReader(f2)
        for line in rows:
            heuri[int(line['node'])] = float(line[str(end)])

    visited = [start]
    sec_dict = {start : 0}
    parent = {}
    pq = PriorityQueue()
    pq.put((0,start))
    num_visited = 0

    while not pq.empty():
        tmp = []
        node = pq.get()

        if node[1] in graph:
            tmp.extend(graph[node[1]])

            for i in tmp:
                if i[0] not in visited or sec_dict[i[0]] > sec_dict[node[1]] + i[1]/i[2]:
                    if i[0] == end:
                        sec = 0
                        path = []
                        tgt = end
                        parent[tgt] = node[1]

                        while tgt != start :
                            path.append(parent[tgt])
                            tgt = parent[tgt]

                        sec = sec_dict[node[1]] + i[1]/i[2]
                        path.append(start)
                        num_visited = len(visited)

                        return path,sec,num_visited
                    else:
                        visited.append(i[0])
                        sec_dict[i[0]] = sec_dict[node[1]] + i[1]/i[2]
                        parent[i[0]] = node[1]
                        pq.put((heuri[i[0]]/max_lmt + sec_dict[node[1]] + i[1]/i[2] , i[0]))

    # End your code (Part 6)
```

Part II. Results & Analysis :

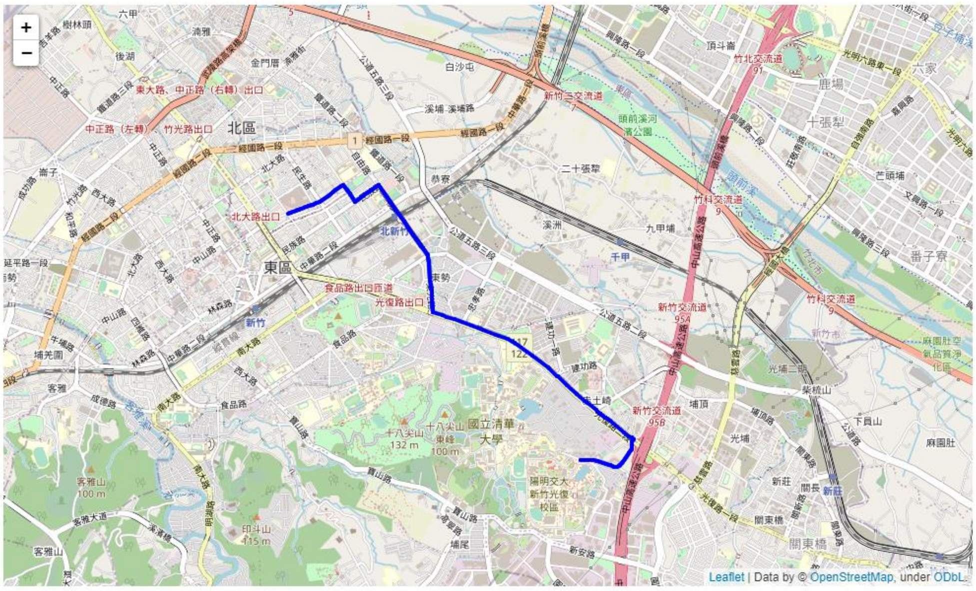
<u>Results:</u>

<u>Test 1 :</u>

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

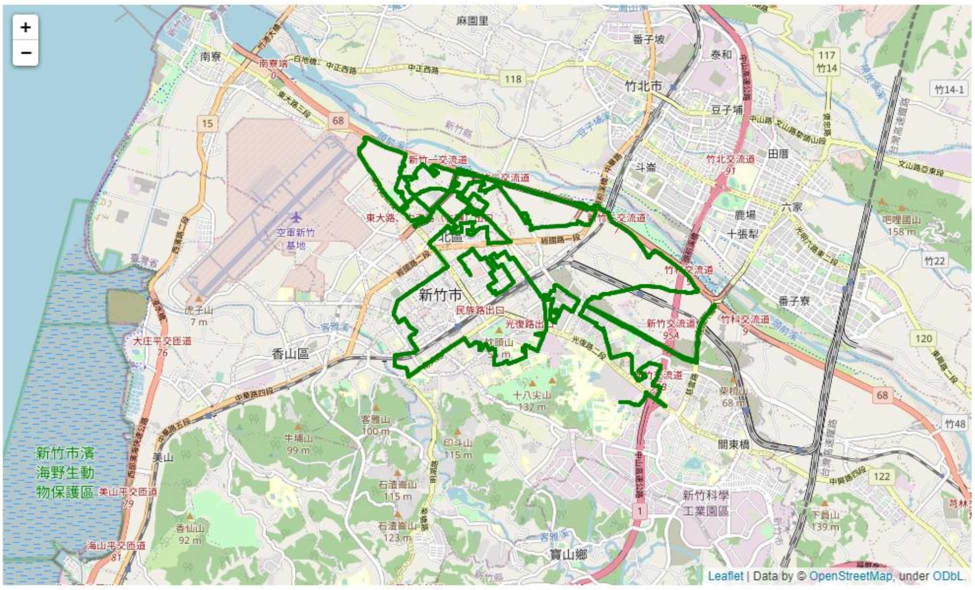
BFS:

```
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: 4274
```



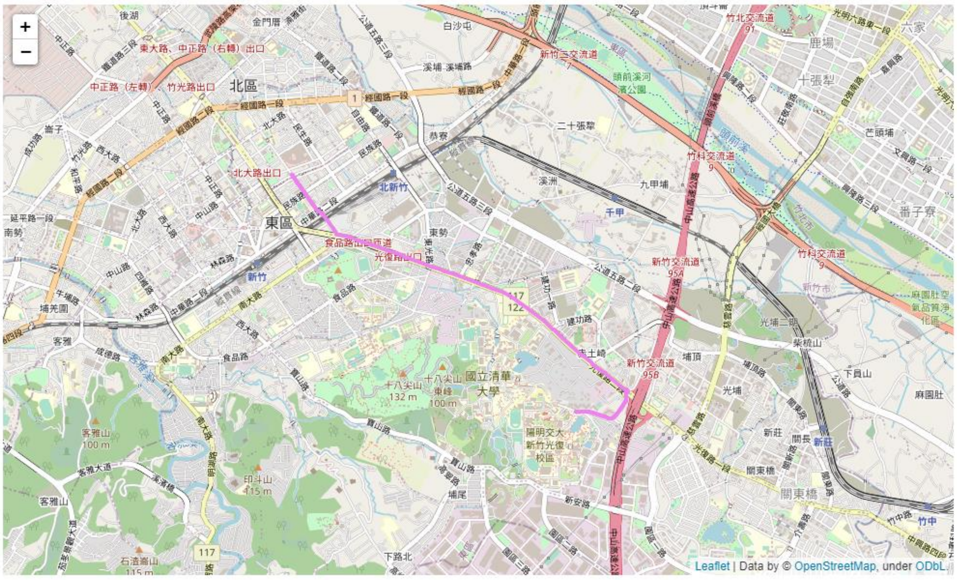
DFS(stack):

```
The number of nodes in the path found by DFS: 1232
Total distance of path found by DFS: 57208.987 m
The number of visited nodes in DFS: 4381
```



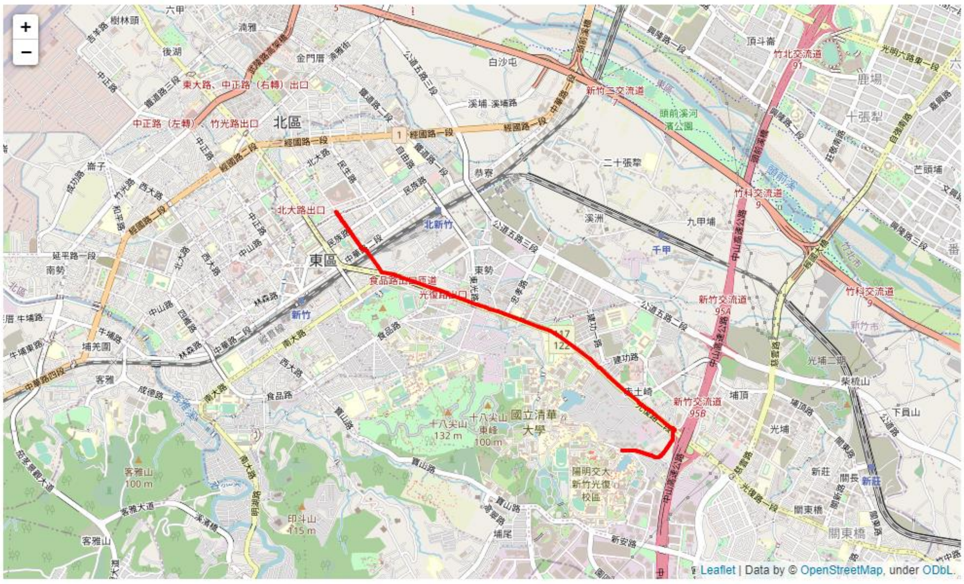
UCS:

```
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: 5222
```



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: 317
```



A*(time):

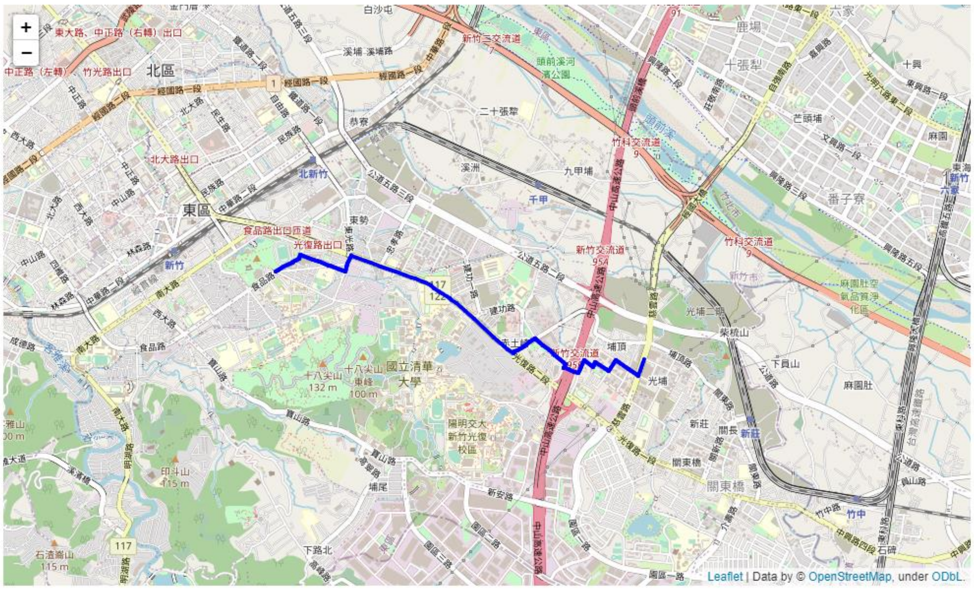
```
The number of nodes in the path found by A* search: 89
Total second of path found by A* search: 89.13284211967543 s
The number of visited nodes in A* search: 2041
```

<u>Test 2 :</u>

from Hsinchu Zoo (ID: 426882161) to COSTCO Hsinchu Store (ID:1737223506)

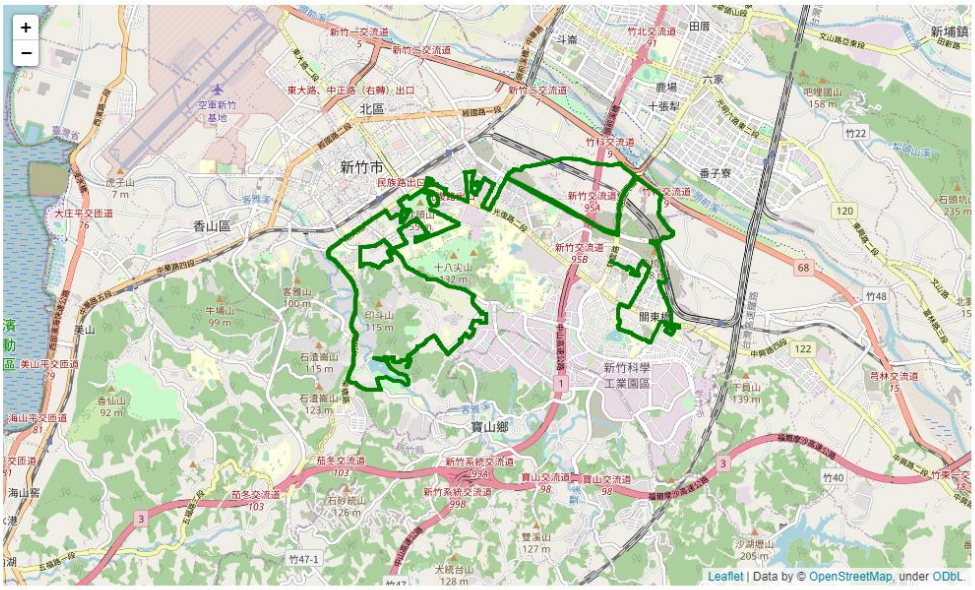
BFS:

```
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: 4607
```



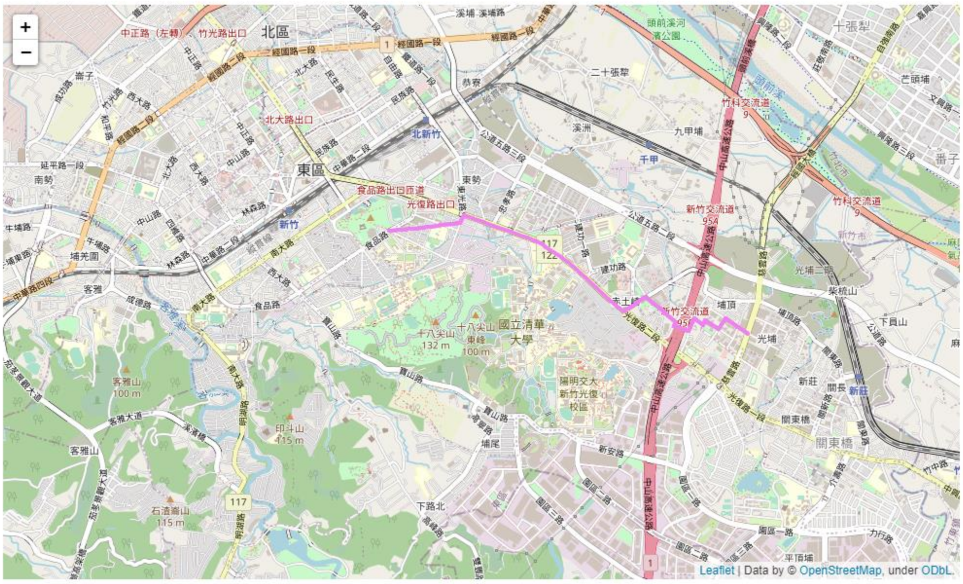
DFS(stack):

```
The number of nodes in the path found by DFS: 998
Total distance of path found by DFS: 41094.657999999916 m
The number of visited nodes in DFS: 8628
```



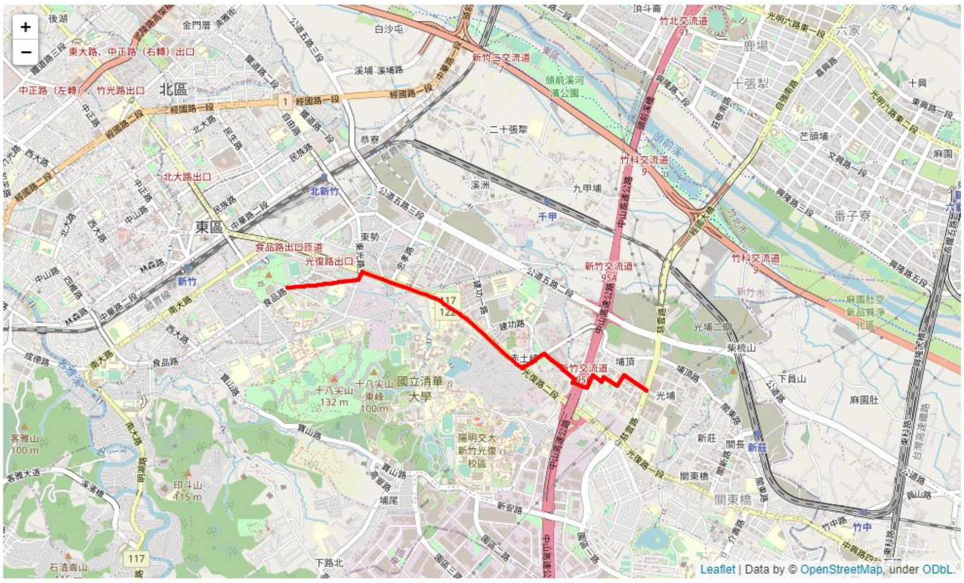
UCS:

```
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: 7129
```



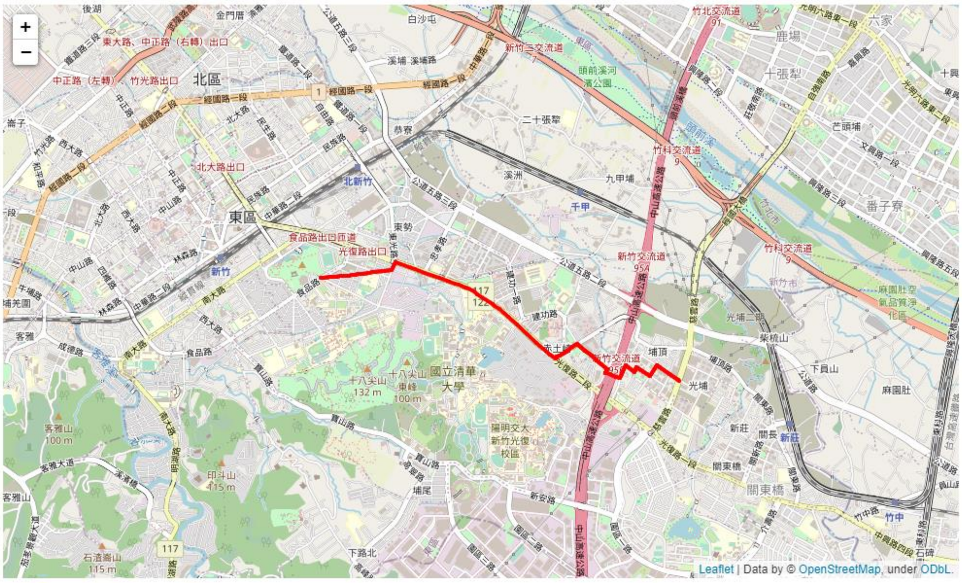
A*:

```
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: 1308
```



A*(time):

```
The number of nodes in the path found by A* search: 63
Total second of path found by A* search: 84.56768428778615 s
The number of visited nodes in A* search: 2845
```

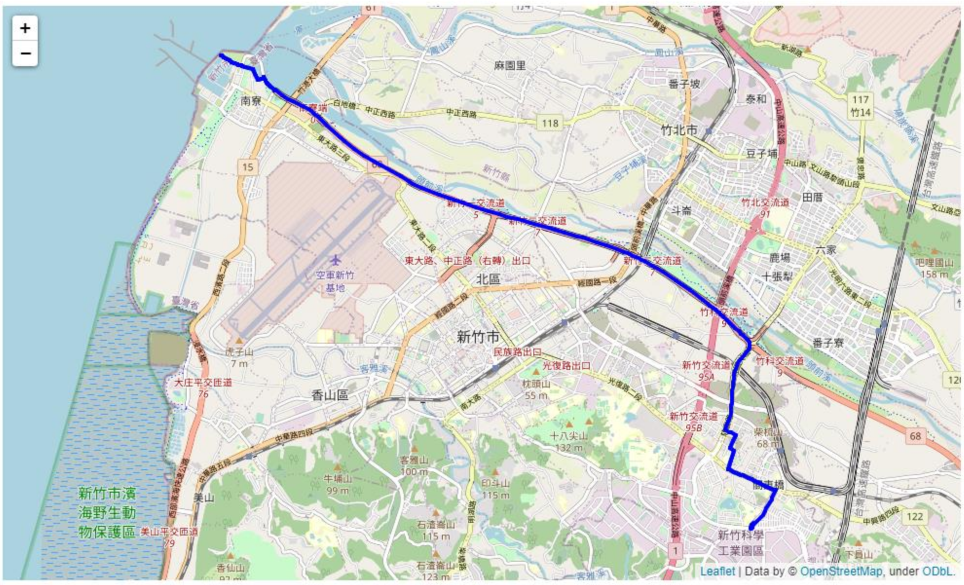


Test 3 :

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

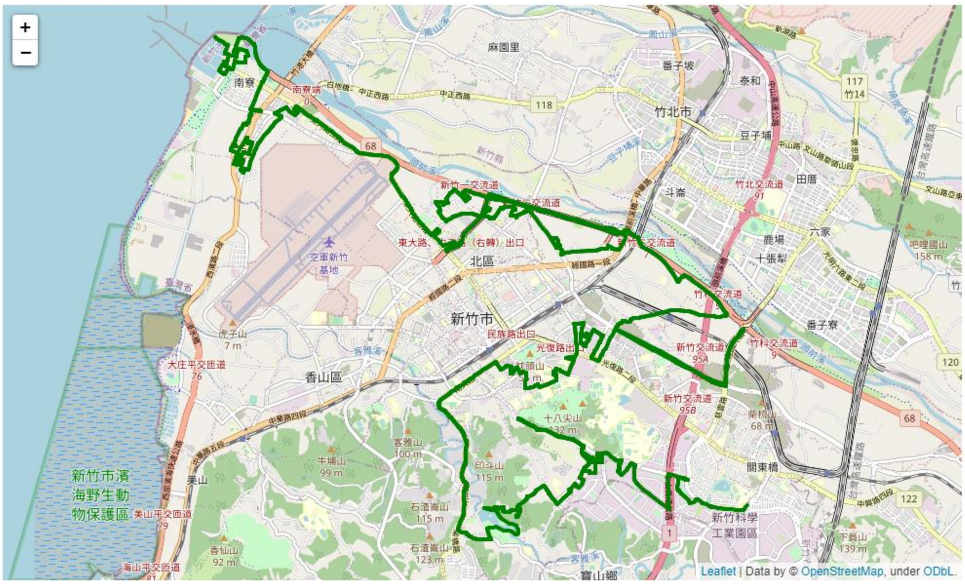
BFS:

```
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: 11242
```



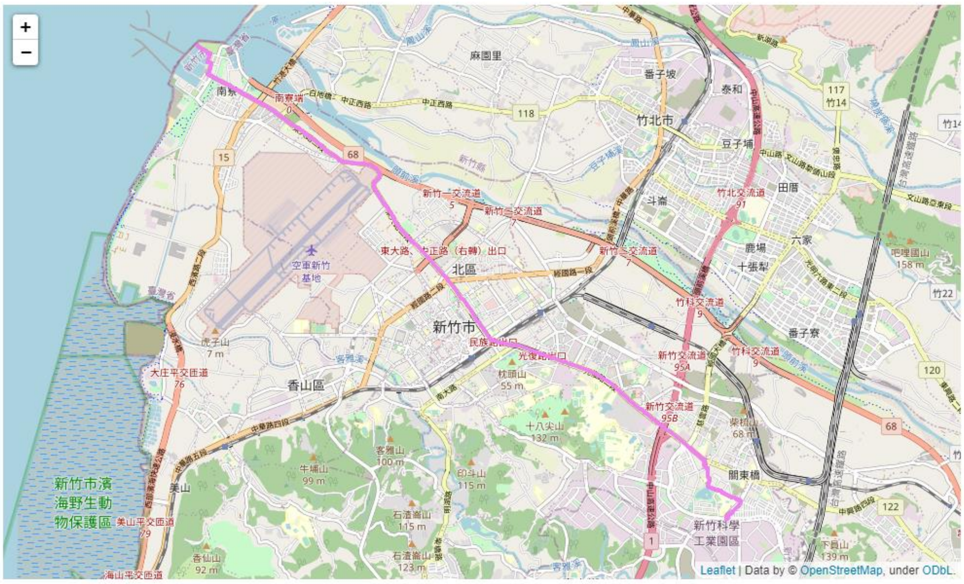
DFS(stack):

```
The number of nodes in the path found by DFS: 1521
Total distance of path found by DFS: 64821.60399999999 m
The number of visited nodes in DFS: 3371
```



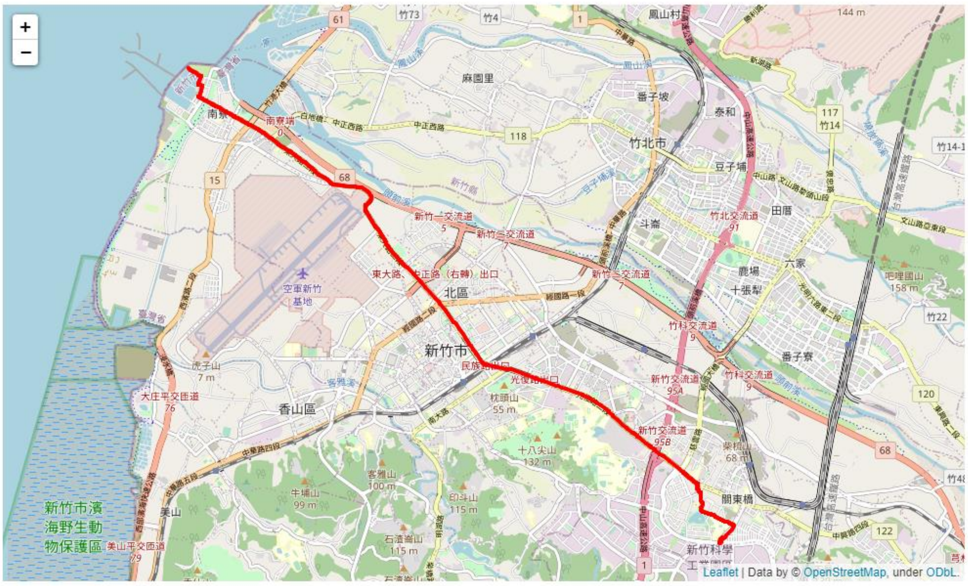
UCS:

```
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: 12316
```



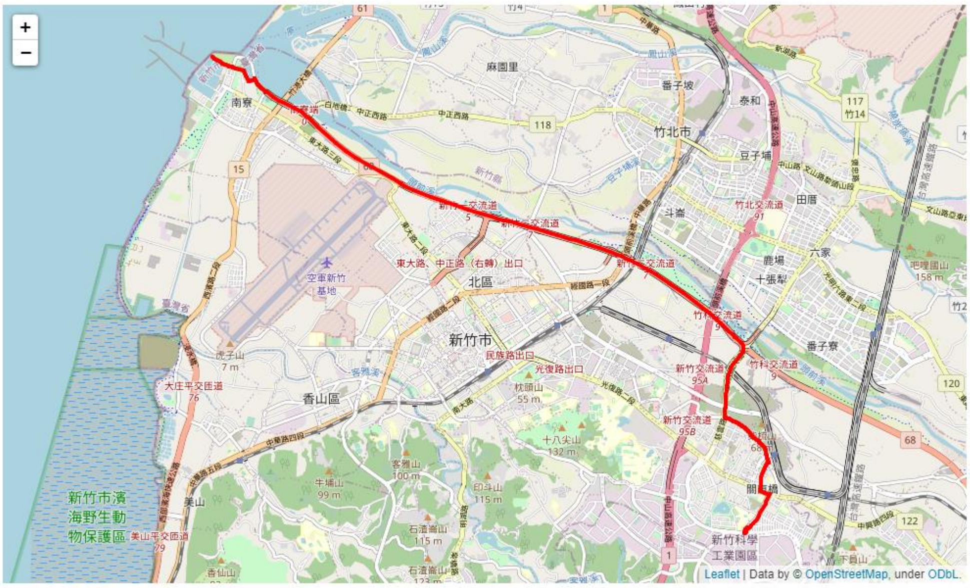
A*:

```
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: 7771
```



A*(time):

```
The number of nodes in the path found by A* search: 209
Total second of path found by A* search: 216.53553412134673 s
The number of visited nodes in A* search: 8879
```



Analysis:

From the results we got , DFS is apparently the one doing most unnecessary work among all algorithms we use this time . I think it's because DFS focuses on one vertex at a time to find a route. In addition , UCS gets the shortest distance while A* and A*(time) focuses on reducing time consumed for moving from starting point to destination .

Part III. Answer the questions :

1. How to transform the data in csv file into a form that is more convenient for me to do this homework had made me hesitate for a while . In the end , I changed it to the form of dictionary , which eventually satisfied me . Not only that , finishing the work requires quite many variables with different purpose , which confused me sometimes .

2. The level of traffic congestion on the road can be an attribute , since it also determines how long people may spend on the road . The former makes " number of traffic lanes of a road " also an attribute , because when there are more traffic lanes , cars can often go faster .

3. For mapping , we can use Google map api and camera drones to get the data we need . For localization , we can use GPS or Cartesian coordinate system on the map we already sketched to determine the absolute position.

4. I consider dynamic heuristic function to be a function which calculates the time required by updating current information of drivers' position and speed limit continuously . Not only that , it's also important to design a route for drivers to deliever food to several people at different places in a single route , without encountering barriers such like the block at the alley between NYCU and NTHU .