

Homework 2: Route Finding

109550135 范恩宇

Part I. Implementation :

↓ ↓ Code and explanation of “ BFS ” ↓ ↓

```
1  import csv
2  edgeFile = 'edges.csv'
3
4  def bfs(start, end):
5      # Begin your code (Part 1)
6
7      """
8      First , build "graph" with data in the csv file . While queue is not empty ,
9      put nodes that are linked to vertices and not visited on queue's top . Also
10     in the while loop , record parents of vertex for tracing . When the route
11     encounters "end" , return the path , distance and number of visited nodes.
12     """
13
14     graph = {}
15     with open(edgeFile, newline='') as f:
16         rows = csv.DictReader(f)
17         for line in rows:
18             tmp = []
19             if int(line['start']) in graph:
20                 tmp.extend(graph[int(line['start'])])
21                 tmp.append((int(line['end']),float(line['distance'])))
22                 graph[int(line['start'])] = tmp
23             else:
24                 tmp.append((int(line['end']),float(line['distance'])))
25                 graph[int(line['start'])] = tmp
26
27     visited = [start]
28     queue = [start]
29     parent = {}
30     dist_dict = {}
31     num_visited = 0
```

```
33     while queue :
34         node = queue[0]
35         queue.pop(0)
36
37         if node in graph:
38             vertice = graph[node]
39             for i in vertice:
40                 if i[0] not in visited:
41                     queue.append(i[0])
42                     visited.append(i[0])
43                     parent[i[0]] = node
44                     dist_dict[i[0]] = i[1]
45
46                 if i[0] == end:
47
48                     dist = 0
49                     path = []
50                     tgt = end
51
52                     while tgt != start :
53                         dist += dist_dict[tgt]
54                         path.append(tgt)
55                         tgt = parent[tgt]
56
57                     path.append(start)
58                     num_visited = len(visited)
59                     return path,dist,num_visited
60     # End your code (Part 1)
```

↓ ↓ Code and explanation of “ DFS ” ↓ ↓

```
1 import csv
2 edgeFile = 'edges.csv'
3
4 def dfs(start, end):
5     # Begin your code (Part 2)
6     """
7     First , build "graph" with data in the csv file . While stack is not empty ,
8     put nodes that are near to vertices and not visited on stack's top ,
9     however , stack pops out the last node . Also in the while loop , record
10    parents of vertex for tracing . When the route encounters "end" , return
11    the path , distance and number of visited nodes.
12    """
13    graph = {}
14    with open(edgeFile, newline='') as f:
15        rows = csv.DictReader(f)
16        for line in rows:
17            tmp = []
18            if int(line['start']) in graph:
19                tmp.extend(graph[int(line['start'])])
20                tmp.append((int(line['end']),float(line['distance'])))
21                graph[int(line['start'])] = tmp
22            else:
23                tmp.append((int(line['end']),float(line['distance'])))
24                graph[int(line['start'])] = tmp
25
26    visited = [start]
27    stack = [start]
28    parent = {}
29    dist_dict = {}
30    num_visited = 0
31
```

```
32 while stack:
33     node = stack.pop()
34     visited.append(node)
35
36     if node in graph:
37         vertices = graph[node]
38         for i in vertices:
39             if i[0] not in visited:
40                 parent[i[0]] = node
41                 dist_dict[i[0]] = i[1]
42                 stack.append(i[0])
43             if i[0] == end:
44                 dist = 0
45                 path = []
46                 tgt = end
47
48                 while tgt != start :
49                     dist += dist_dict[tgt]
50                     path.append(parent[tgt])
51                     tgt = parent[tgt]
52
53                 path.append(start)
54                 num_visited = len(visited)
55
56                 return path,dist,num_visited
57
58 # End your code (Part 2)
```

↓ ↓ Code and explanation of “ UCS ” ↓ ↓

```
1 import csv
2 from queue import PriorityQueue
3 edgeFile = 'edges.csv'
4
5 def ucs(start, end):
6     # Begin your code (Part 3)
7     """
8     First , build "graph" with data in the csv file ,then put "start " into
9     priority queue "pq". While "pq" is not empty , put the node near the
10    vertex into "pq" and record distance between the node mentioned before
11    and "start" into "dist_dict" and replace the value if receiving smaller one.
12    When the route encounters "end" , return the path , distance and number
13    of visited nodes.
14    """
15    graph = {}
16    with open(edgeFile, newline='') as f:
17        rows = csv.DictReader(f)
18        for line in rows:
19            tmp = []
20            if int(line['start']) in graph:
21                tmp.extend(graph[int(line['start'])])
22                tmp.append((int(line['end']),float(line['distance'])))
23                graph[int(line['start'])] = tmp
24            else:
25                tmp.append((int(line['end']),float(line['distance'])))
26                graph[int(line['start'])] = tmp
27
28
29    visited = [start]
30    dis_dict = {start : 0}
31    parent = {}
32    pq = PriorityQueue()
33    pq.put((0,start))
34    # End your code (Part 3)
```

```
33    pq.put((0,start))
34    num_visited = 0
35
36    while not pq.empty():
37        tmp = []
38        node = pq.get()
39
40        if node[1] in graph:
41            tmp.extend(graph[node[1]])
42            for i in tmp:
43                if i[0] not in visited or dis_dict[i[0]] > node[0] + i[1]:
44                    if i[0] == end:
45                        dist = 0
46                        path = []
47                        tgt = end
48                        parent[tgt] = node[1]
49
50                        while tgt != start :
51                            path.append(parent[tgt])
52                            tgt = parent[tgt]
53
54                        dist = node[0] + i[1]
55                        path.append(start)
56                        num_visited = len(visited)
57
58                        return path,dist,num_visited
59
60        else:
61            visited.append(i[0])
62            dis_dict[i[0]] = node[0] + i[1]
63            parent[i[0]] = node[1]
64            pq.put((node[0]+i[1],i[0]))
65    # End your code (Part 3)
```

↓ ↓ Code and explanation of “A*” ↓ ↓

```

1  import csv
2  from queue import PriorityQueue
3  edgeFile = 'edges.csv'
4  heuristicFile = 'heuristic.csv'
5
6  def astar(start, end):
7      # Begin your code (Part 4)
8      """
9      First , build "graph" and "heuri " with data in 2 csv files ,then put
10     "start " into priority queue "pq" . While "pq" is not empty , put
11     the node near the vertex into "pq" and record the distance between the
12     node mentioned before and "start" into "dist_dict" and replace the
13     value if receiving smaller one . In addition , sum the distance and
14     the "heuri" to compare . When the route encounters "end" , return the
15     path , distance and number of visited nodes.
16
17     """
18     graph = {}
19     heuri = {}
20
21     with open(edgeFile, newline='') as f:
22         rows = csv.DictReader(f)
23         for line in rows:
24             tmp = []
25             if int(line['start']) in graph:
26                 tmp.extend(graph[int(line['start'])])
27                 tmp.append((int(line['end']),float(line['distance'])))
28                 graph[int(line['start'])] = tmp
29             else:
30                 tmp.append((int(line['end']),float(line['distance'])))
31                 graph[int(line['start'])] = tmp
32
33     with open(heuristicFile, newline='') as f2:
34         rows = csv.DictReader(f2)
35         for line in rows:
36             heuri[int(line['node'])] = float(line[str(end)])
37
38     visited = [start]
39     dist_dict = {start : 0}
40     parent = {}
41     pq = PriorityQueue()
42     pq.put((0,start))
43     num_visited = 0
44
45     while not pq.empty():
46         tmp = []
47         node = pq.get()
48
49         if node[1] in graph:
50             tmp.extend(graph[node[1]])
51
52         for i in tmp:
53             if i[0] not in visited or dist_dict[i[0]] > dist_dict[node[1]] + i[1]:
54                 if i[0] == end:
55                     dist = 0
56                     path = []
57                     tgt = end
58                     parent[tgt] = node[1]
59
60                     while tgt != start :
61                         path.append(parent[tgt])
62                         tgt = parent[tgt]
63
64                     dist = dist_dict[node[1]] + i[1]
65                     path.append(start)
66                     num_visited = len(visited)
67
68                     return path,dist,num_visited
69
70             else:
71                 visited.append(i[0])
72                 dist_dict[i[0]] = dist_dict[node[1]] + i[1]
73                 parent[i[0]] = node[1]
74                 pq.put((heuri[i[0]] + dist_dict[node[1]]+i[1],i[0]))
75
76     # End your code (Part 4)

```

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

```

1 import csv
2 from queue import PriorityQueue
3 edgeFile = 'edges.csv'
4 heuristicFile = 'heuristic.csv'
5
6 def astar_time(start, end):
7
8     # Begin your code (Part 6)
9     """
10     Basically the same thing like "astar.py", but I used the biggest
11     speed limit to divide "heuri". In addition, I divide each distance
12     by speed limit to get the time, not distance.
13     """
14
15     max_lmt = 0
16     graph = {}
17     heuri = {}
18     with open(edgeFile, newline='') as file:
19         rows = csv.DictReader(file)
20         for line in rows:
21             tmp = []
22             if int(line['start']) in graph:
23                 tmp.extend(graph[int(line['start'])])
24                 tmp.append((int(line['end']), float(line['distance']), float(line['speed limit'])))
25                 max_lmt = max(max_lmt, float(line['speed limit']))
26                 graph[int(line['start'])] = tmp
27             else:
28                 tmp.append((int(line['end']), float(line['distance']), float(line['speed limit'])))
29                 max_lmt = max(max_lmt, float(line['speed limit']))
30                 graph[int(line['start'])] = tmp
31

```

```

31
32     with open(heuristicFile, newline='') as f2:
33         rows = csv.DictReader(f2)
34         for line in rows:
35             heuri[int(line['node'])] = float(line['end'])
36
37     visited = [start]
38     sec_dict = {start : 0}
39     parent = {}
40     pq = PriorityQueue()
41     pq.put((0, start))
42     num_visited = 0
43

```

```

44     while not pq.empty():
45         tmp = []
46         node = pq.get()
47
48         if node[1] in graph:
49             tmp.extend(graph[node[1]])
50
51         for i in tmp:
52             if i[0] not in visited or sec_dict[i[0]] > sec_dict[node[1]] + i[1]/i[2]:
53                 if i[0] == end:
54                     sec = 0
55                     path = []
56                     tgt = end
57                     parent[tgt] = node[1]
58
59                     while tgt != start :
60                         path.append(parent[tgt])
61                         tgt = parent[tgt]
62
63                     sec = sec_dict[node[1]] + i[1]/i[2]
64                     path.append(start)
65                     num_visited = len(visited)
66
67                     return path, sec, num_visited
68                 else:
69                     visited.append(i[0])
70                     sec_dict[i[0]] = sec_dict[node[1]] + i[1]/i[2]
71                     parent[i[0]] = node[1]
72                     pq.put((heuri[i[0]]/max_lmt + sec_dict[node[1]] + i[1]/i[2], i[0]))
73
74     # End your code (Part 6)
75

```

Part II. Results & Analysis :

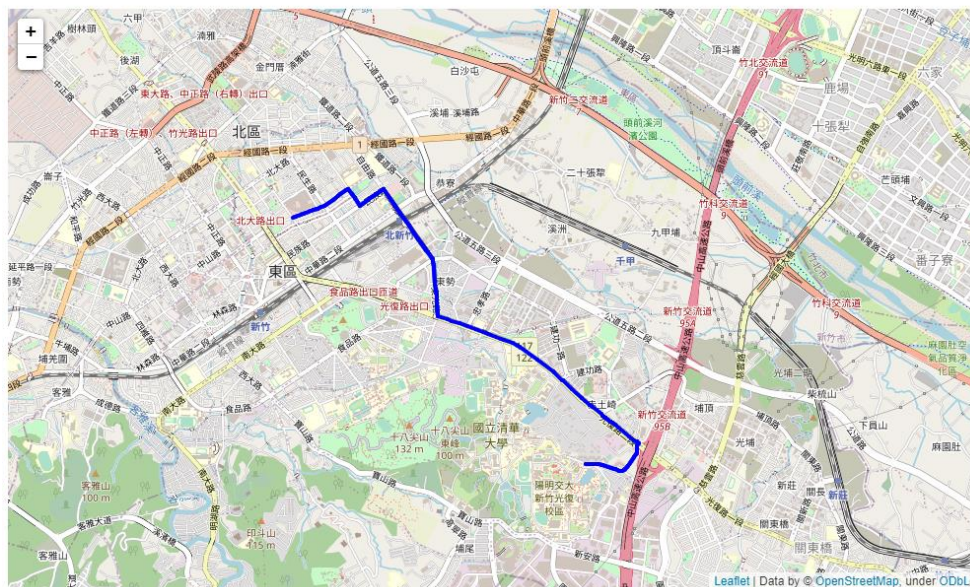
Results:

Test 1 :

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

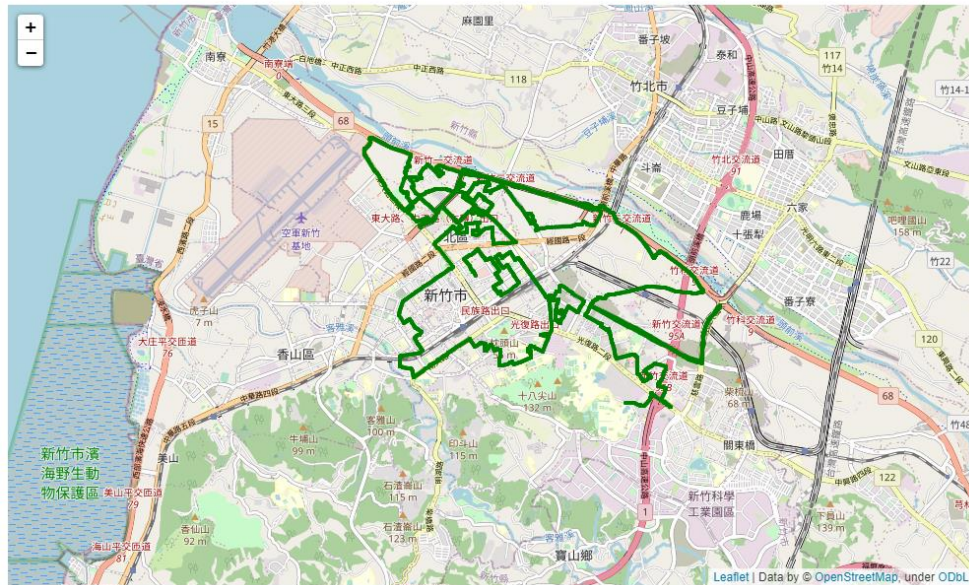
BFS:

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: 4274



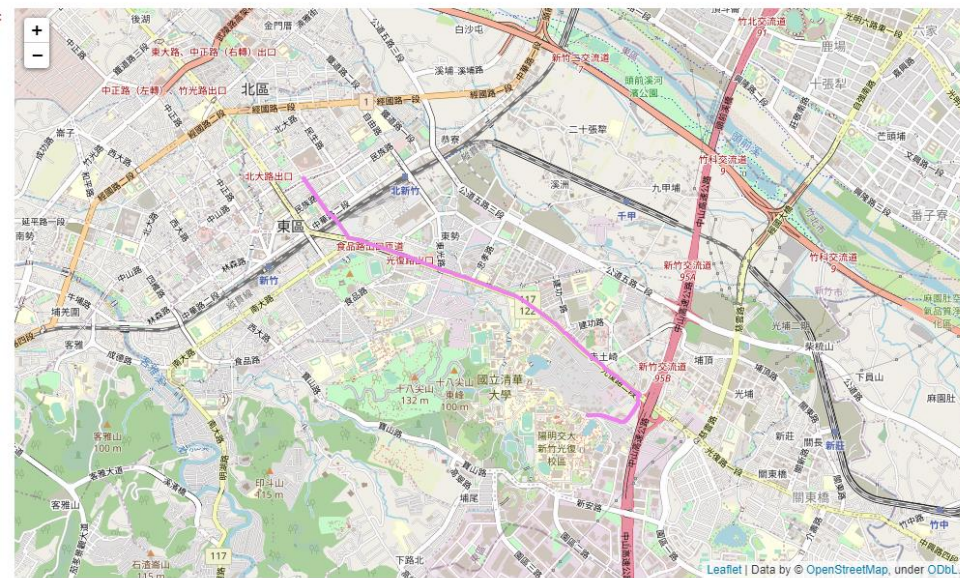
DFS(stack):

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



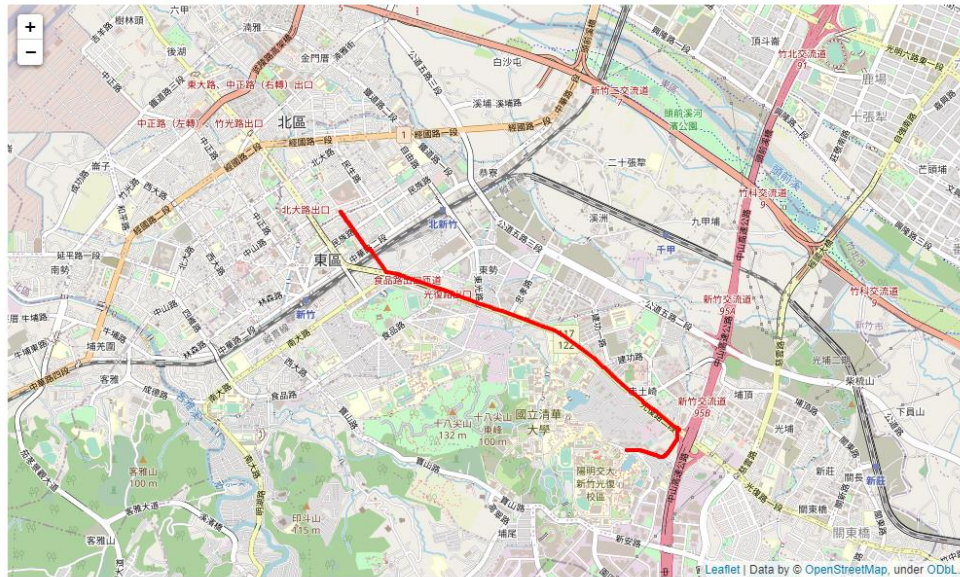
UCS:

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



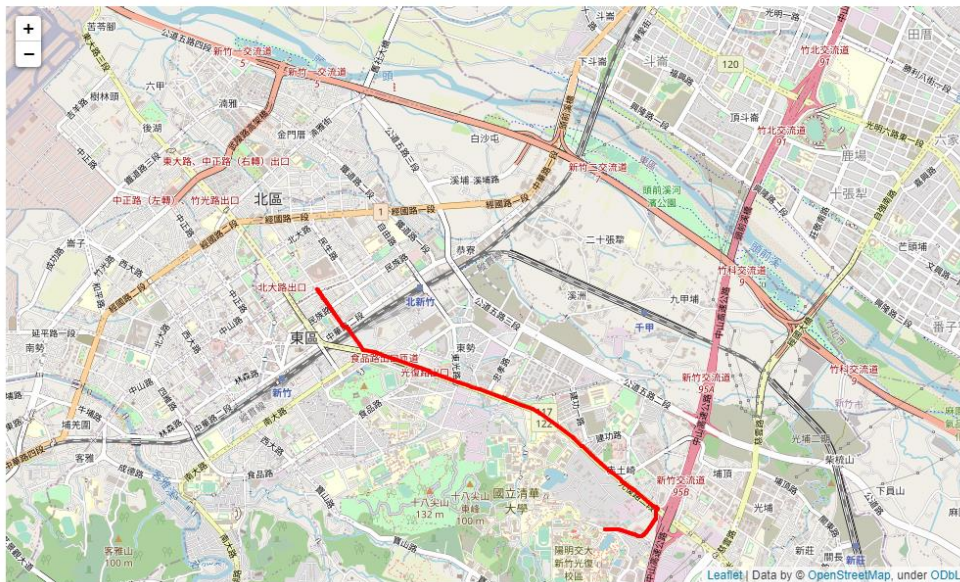
A*:

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

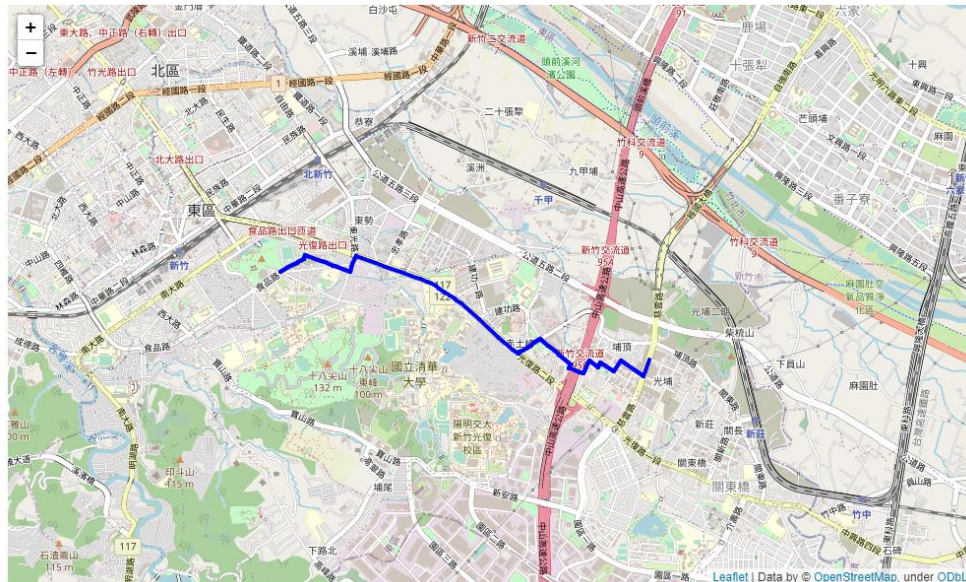


Test 2 :

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

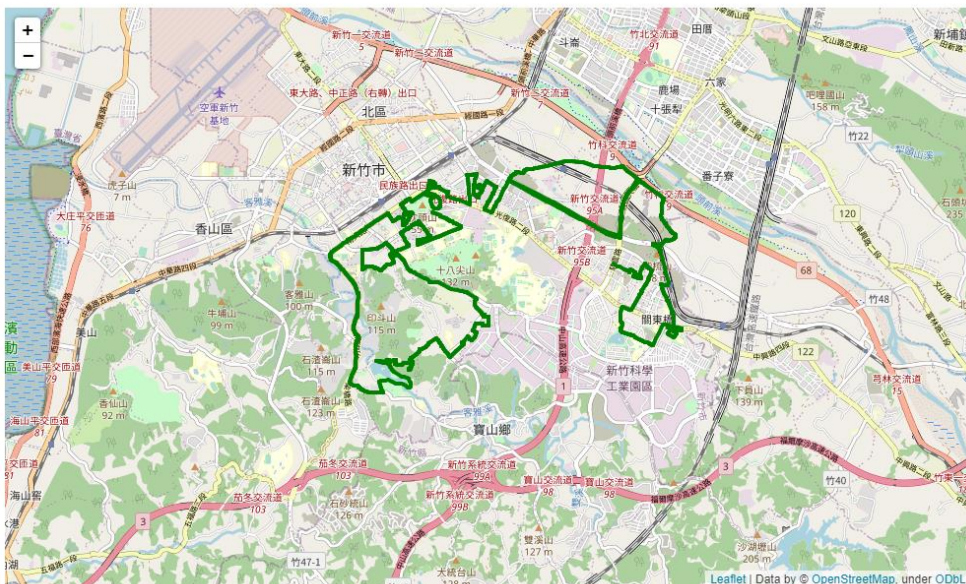
BFS:

The number of nodes in the path found by BFS: 60
Total distance of path found by BFS: 4215.52100000001 m
The number of visited nodes in BFS: 4607



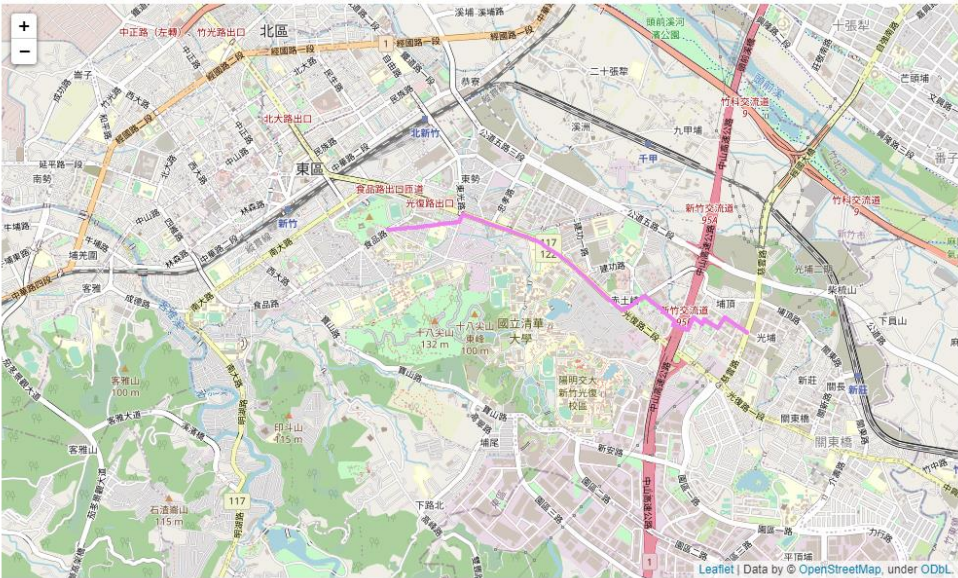
DFS(stack):

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



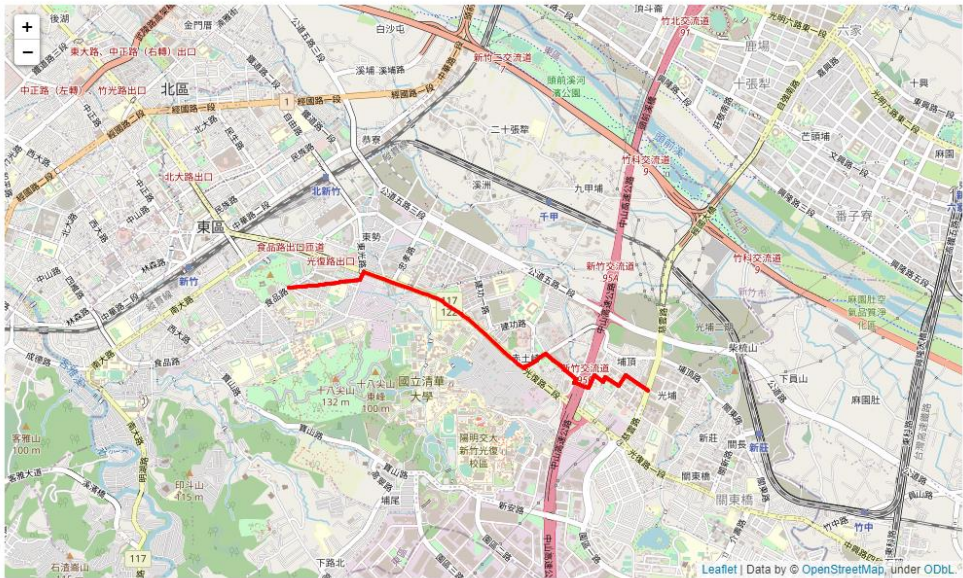
UCS:

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



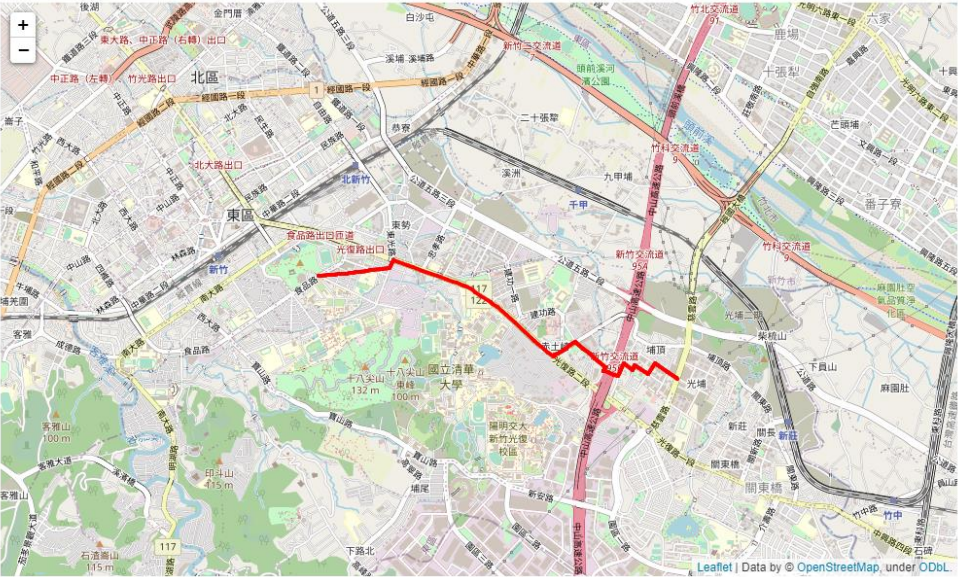
A*:

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

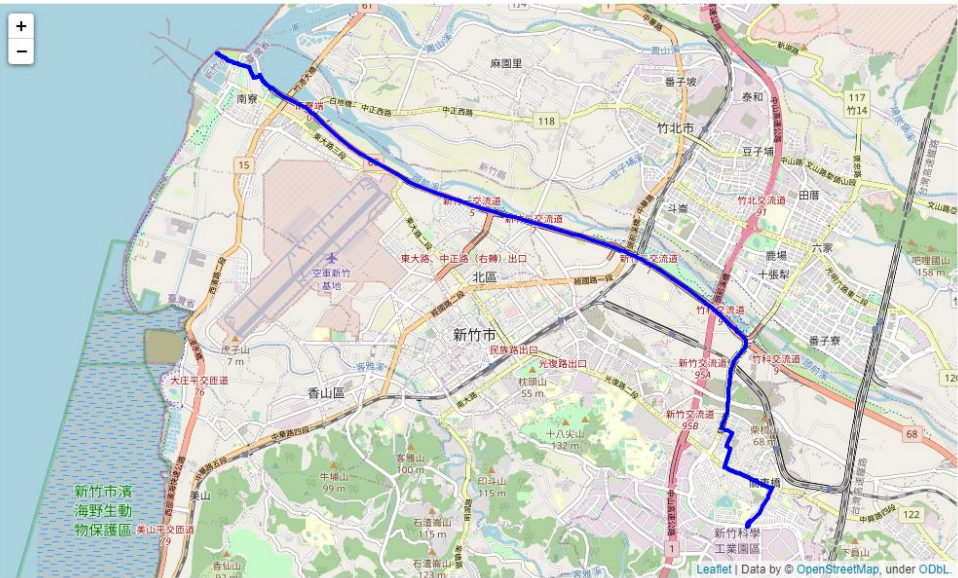


Test 3 :

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

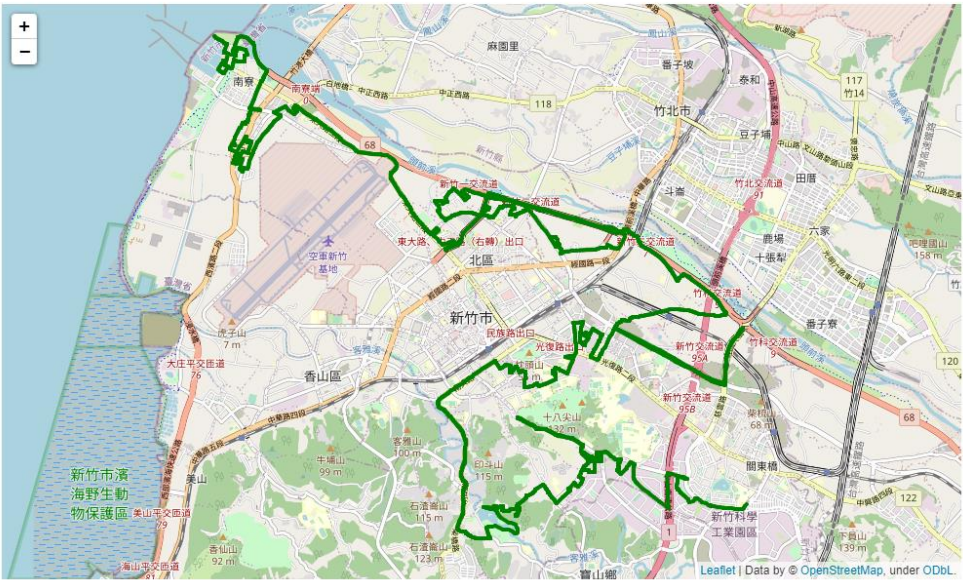
BFS:

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



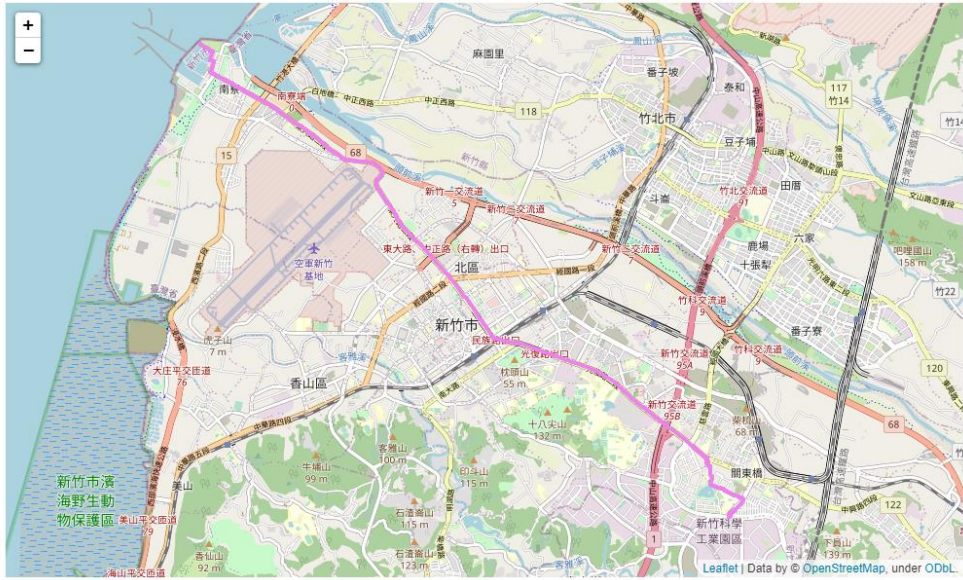
DFS(stack):

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



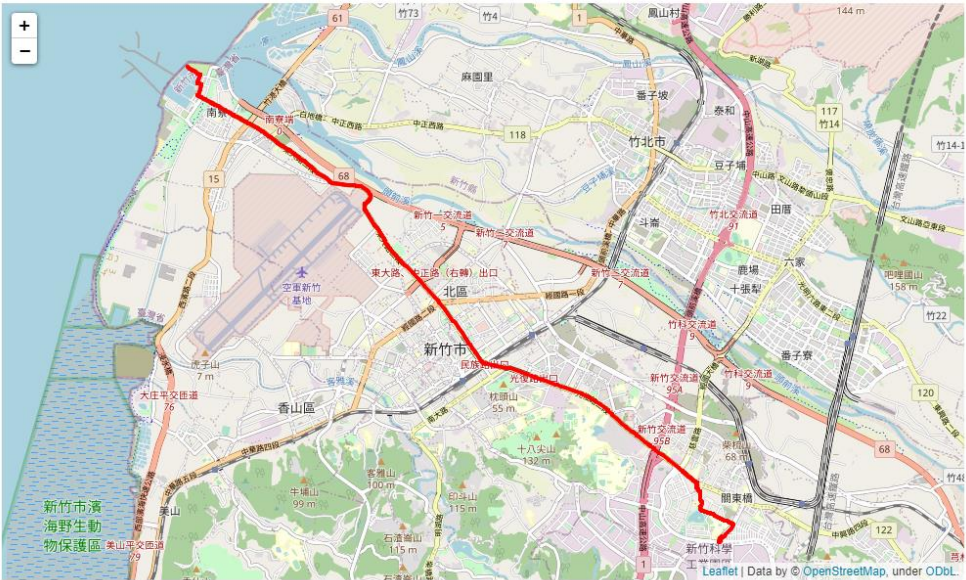
UCS:

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



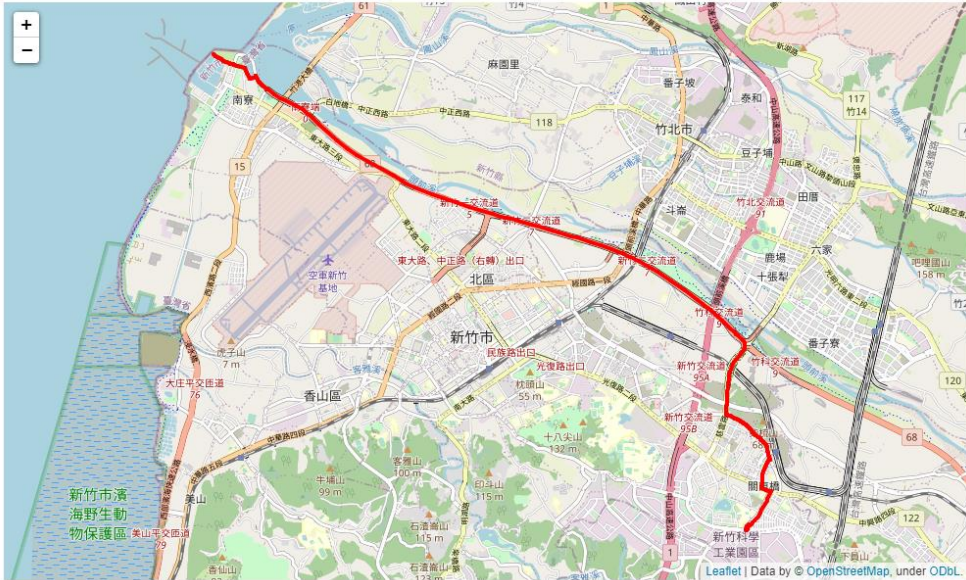
A*:

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 deliver 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 .