

# Homework 2: Route Finding Report

109550116 楊傑宇

## Part I. Implementation (6%):

```
5  # Begin your code (Part 1)
6  """
7  My main idea is to create a class that record every point's state
8  and which point they can go...
9
10 (line 44 to 56) is the construction of my class
11 d      represent the distance it have passed from start point to this point
12 num    represent the point address
13 color   represent whether it have been passed
14 pre     represent the previous point
15 list[]  represent all the points this point can go
16 distance[] represent the distance from this points to next point respectively
17
18 (line 59 to 76) is how I record point's information to class
19 a[] is a array to save all the class
20 b[] is just a array to save the point address
21 |   in order to find the location of a point in a[]
22 j   is avoid getting the first line
23 tem is avoid creating the same point
24 count is to count how many point i visited
25
26 (line 79 to 82 ) is to get the start point and put it in quene
27
28 (line 84 to 103 ) is the main bfs function
29 step 1: I get the point(oldest) and put all points
30 |   which it(oldest) can go to in quene and update their state
31 step 2: repeat step 1 until the end point be get from quene
32 * need to pay attention to the point not in
33 | a[](these points mean can't go to another points ) need add them to a[]
34
35 (line 105 to 122 ) is to find the result return
36 step 1: I find the end point and use it.pre to get its previous point
37 |   until the start point gotten
38 step 2: record all the point in step 2 in path[]
39 step 3: dis is just the end point.d
40 step 4: count can be return directly
41
42 """
43
44 class graph:
45     def __init__(self,num,to,dis):
46         self.d=0
47         self.num=num
48         self.color=0
49         self.distance=[]
50         self.pre=-1
51         self.list=[]
52         self.list.append(to)
53         self.distance.append(dis)
54     def add(self,to,dis):
```



```
105     path=[]
106     s=0
107     for i in range(len(a)):
108         if(a[i].num==end):
109             s=i
110
111     dis=a[s].d
112     path.append(a[s].num)
113
114     while(1):
115         k=a[s].pre
116         path.append(k)
117         if(k==start):
118             break
119         else:
120             s=b.index(k)
121
122     return path , dis , count
123
124
125 #raise NotImplementedError("To be implemented")
126 # End your code (Part 1)
```

```

7  # Begin your code (Part 2)
8  """
9  My main idea is to create a class that record every point's state
10 and which point they can go...
11
12 (line 31 to 43) is the construction of my class
13 explanation the same as part 1
14
15 (line 46 to 61) is how I record point's information to class
16 explanation the same as part 1
17 *count become c[] because i want to bring it to recursive function
18
19 (line 79 to 87 ) is to get the start point and put it in function
20
21 (line 63 to 77 ) is the main dfs function
22 step 1: I get all points the start point can go to and do recursive
23 | so it will find point and update the point's state until the leaf
24 | (which means it will go the road until it can't get to any new point)
25 * I dont set terminal condition because too complicated
26
27 (line 89 to 106 ) is to find the result return
28 explanation the same as part 1|
29 """
30
31 class graph:
32     def __init__(self,num,to,dis):
33         self.d=0
34         self.num=num
35         self.color=0
36         self.distance=[]
37         self.pre=-1
38         self.list=[]
39         self.list.append(to)
40         self.distance.append(dis)
41     def add(self,to,dis):
42         self.list.append(to)
43         self.distance.append(dis)
44
45 def dfs(start, end):
46     a=[]
47     b=[]
48     j=0
49     tem=0
50     file=open(edgeFile)
51     for i in file.readlines():
52         if(j>0):
53             ii=i.split(',')
54             if(tem==int(ii[0])):
55                 a[len(a)-1].add(int(ii[1]),float(ii[2]))
56             else:
57                 a.append(graph(int(ii[0]),int(ii[1]),float(ii[2])))
58                 b.append(int(ii[0]))
59                 tem=int(ii[0])
60             j=j+1
61     file.close()

```

```

62
63     def dfss(p,a,b,c):
64
65         c[0]=c[0]+1
66         for i in range(len(p.list)):
67             if p.list[i] in b :
68                 l=b.index(p.list[i])
69                 if(a[l].color==0):
70                     a[l].d=p.d+p.distance[i]
71                     a[l].pre=p.num
72                     a[l].color=1
73                     dfss(a[l],a,b,c)
74             else:
75                 a.append(graph(p.list[i],0,0))
76                 a[len(a)-1].d=p.d+p.distance[i]
77                 a[len(a)-1].color=1
78
79         s=0
80         for i in range(len(a)):
81             if(a[i].num==start):
82                 s=i
83
84         c=[]
85         c.append(0)
86         a[s].color=1
87         dfss(a[s],a,b,c)
88
89         path=[]
90         s=0
91         for i in range(len(a)):
92             if(a[i].num==end):
93                 s=i
94
95             dis=a[s].d
96             path.append(a[s].num)
97
98             while(1):
99                 k=a[s].pre
100                 path.append(k)
101                 if(k==start):
102                     break
103                 else:
104                     s=b.index(k)
105
106         return path , dis , c[0]
107
108     #raise NotImplementedError("To be implemented")
109     # End your code (Part 2)

```

```

4   # Begin your code (Part 3)|
5   """
6   My main idea is to create a class that record every point's state
7   and which point they can go...
8
9   (line 25 to 37) is the construction of my class
10  explanation the same as part 1
11
12 (line 40 to 56) is how I record point's information to class
13 explanation the same as part 1
14
15 (line 58 to 61 ) is to get the start point and put it in q={}
16
17 (line 64 to 96 ) is the main ucs function
18 ucs is like bfs so I explain different place
19 (line 67 to 74) I only choose the current shorest distance to pop
20 (line 84 to 87) if this point have shorter path update it's state
21
22 (line 98 to 115 ) is to find the result return
23 explanation the same as part 1
24 """
25
26 class graph:
27     def __init__(self,num,to,dis):
28         self.d=0
29         self.num=num
30         self.color=0
31         self.distance=[]
32         self.pre=-1
33         self.list=[]
34         self.list.append(to)
35         self.distance.append(dis)
36     def add(self,to,dis):
37         self.list.append(to)
38         self.distance.append(dis)
39
40     def ucs(start, end):
41         a=[]
42         b=[]
43         j=0
44         tem=0
45         count=0
46         file=open(edgeFile)
47         for i in file.readlines():
48             if(j>0):
49                 ii=i.split(',')
50                 if(tem==int(ii[0])):
51                     a[len(a)-1].add(int(ii[1]),float(ii[2]))
52                 else:
53                     a.append(graph(int(ii[0]),int(ii[1]),float(ii[2])))
54                     b.append(int(ii[0]))
55             tem=int(ii[0])
56             j=j+1
57         file.close()

```

```

58     s=0
59     for i in range(len(a)):
60         if(a[i].num==start):
61             s=i
62
63
64     q=[]
65     q.append(a[s])
66     while not(len(q)==0):
67         m=0
68         min=1000000
69         for i in range(len(q)) :
70             if(q[i].d<min):
71                 min=q[i].d
72                 m=i
73
74         u=q[m]
75         q.pop(m)
76         u.color=1
77         count=count+1
78         if(u.num==end):
79             break
80         for i in range(len(u.list)):
81             if u.list[i] in b :
82                 l=b.index(u.list[i])
83                 if(a[l].color==0):
84                     if a[l] in q:
85                         if a[l].d>u.d+u.distance[i]:
86                             a[l].d=u.d+u.distance[i]
87                             a[l].pre=u.num
88                     else:
89                         q.append(a[l])
90                         a[l].d=u.d+u.distance[i]
91                     a[l].pre=u.num
92                 else :
93                     a.append(graph(u.list[i],0,0))
94                     b.append(u.list[i])
95                     a[len(a)-1].d=u.d+u.distance[i]
96                     a[len(a)-1].pre=u.num
97
98         path=[]
99         s=0
100        for i in range(len(a)):
101            if(a[i].num==end):
102                s=i
103
104        dis=a[s].d
105        path.append(a[s].num)
106
107        while(1):
108            k=a[s].pre
109            path.append(k)
110            if(k==start):
111                break
112            else:
113                s=b.index(k)
114
115        return path , dis , count
116    #raise NotImplementedError("To be implemented")
117    # End your code (Part 3)

```

```

4   # Begin your code (Part 4)
5   """
6   My main idea is to create a class that record every point's state
7   and which point they can go...
8
9   (line 27 to 39) is the construction of my class
10  explanation the same as part 1
11  * I add a h to represent point's heuristic
12
13  (line 42 to 73) is how I record point's information to class
14  explanation the same as part 1
15  * add I use h1,h2,h3 to save heuristic.csv
16
17  (line 74 to 77 ) is to get the start point and put it in q=[]
18
19  (line 78 to 113 ) is the main astar function
20  actually, it is similar to ucs, however i use .d to judge which edge go first
21  and now, i use .h to judge
22
23  (line 114 to 130 ) is to find the result return
24  explanation the same as part 1
25  """
26  class graph:
27      def __init__(self,num,to,dis):
28          self.d=0
29          self.h=0
30          self.num=num
31          self.color=0
32          self.distance=[]
33          self.pre=-1
34          self.list=[]
35          self.list.append(to)
36          self.distance.append(dis)
37      def add(self,to,dis):
38          self.list.append(to)
39          self.distance.append(dis)
40
41  def astar(start, end):
42      a=[]
43      b=[]
44      j=0
45      tem=0
46      count=0
47      file=open(edgeFile)
48      for i in file.readlines():
49          if(j>0):
50              ii=i.split(',')
51              if(tem==int(ii[0])):
52                  a[len(a)-1].add(int(ii[1]),float(ii[2]))
53              else:
54                  a.append(graph(int(ii[0]),int(ii[1]),float(ii[2])))
55                  b.append(int(ii[0]))
56                  tem=int(ii[0])
57          j=j+1
58      file.close()
59      j=0
60      h1={}
61      h2={}
62      h3={}
63      file=open(heuristicFile)
64      for i in file.readlines():
65          if(j>0):
66              ii=i.split('\r')
67              iii=ii[0].split(',')

```

```

68
69         h1.update({int(iii[0]):float(iii[1])})
70         h2.update({int(iii[0]):float(iii[2])})
71         h3.update({int(iii[0]):float(iii[3])})
72         j=j+1
73     file.close()
74     s=0
75     for i in range(len(a)):
76         if(a[i].num==start):
77             s=i
78     q=[]
79     q.append(a[s])
80     while not(len(q)==0):
81
82         m=0
83         min=10000000
84         for i in range(len(q)) :
85             if(q[i].h<min):
86                 min=q[i].h
87                 m=i
88
89         u=q[m]
90         q.pop(m)
91         u.color=1
92         count=count+1
93         if(u.num==end):
94             break
95         for i in range(len(u.list)):
96             if u.list[i] in b :
97                 l=b.index(u.list[i])
98                 if(a[l].color==0):
99                     if a[l] in q:
100                         if a[l].d>u.d+u.distance[i]:
101                             a[l].d=u.d+u.distance[i]
102                             a[l].h=u.d+u.distance[i]+h1[a[l].num]
103                             a[l].pre=u.num
104                     else:
105                         q.append(a[l])
106                         a[l].d=u.d+u.distance[i]
107                         a[l].h=u.d+u.distance[i]+h1[a[l].num]
108                         a[l].pre=u.num
109                 else :
110                     a.append(graph(u.list[i],0,0))
111                     b.append(u.list[i])
112                     a[len(a)-1].d=u.d+u.distance[i]
113                     a[len(a)-1].pre=u.num
114     path=[]
115     s=0
116     for i in range(len(a)):
117         if(a[i].num==end):
118             s=i
119         dis=a[s].d
120         path.append(a[s].num)
121     while(1):
122         k=a[s].pre
123         path.append(k)
124         if(k==start):
125             break
126         else:
127             s=b.index(k)
128     return path , dis , count
129 #raise NotImplementedError("To be implemented")
130 # End your code (Part 4)

```

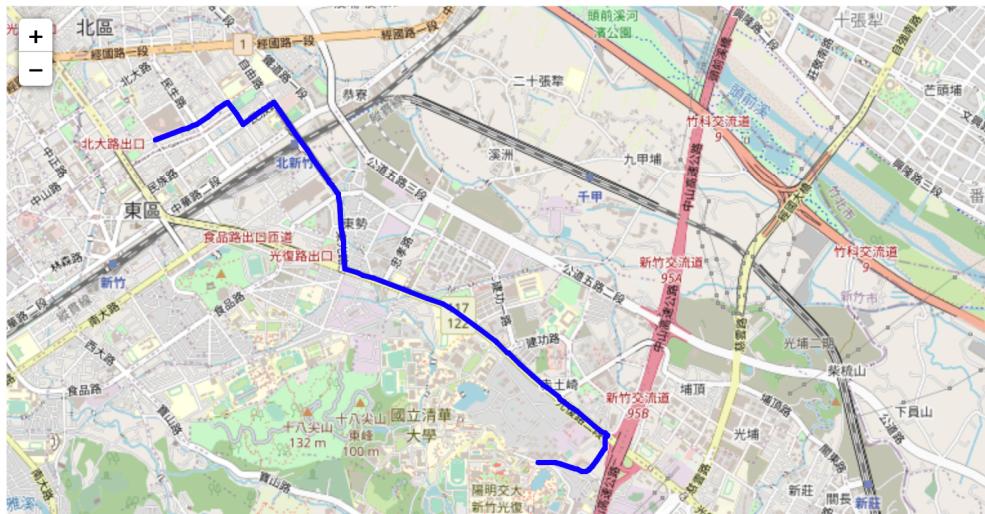
## Part II. Results & Analysis (12%):

Test1: 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: 4978.8820000000005 m  
The number of visited nodes in BFS: 4268

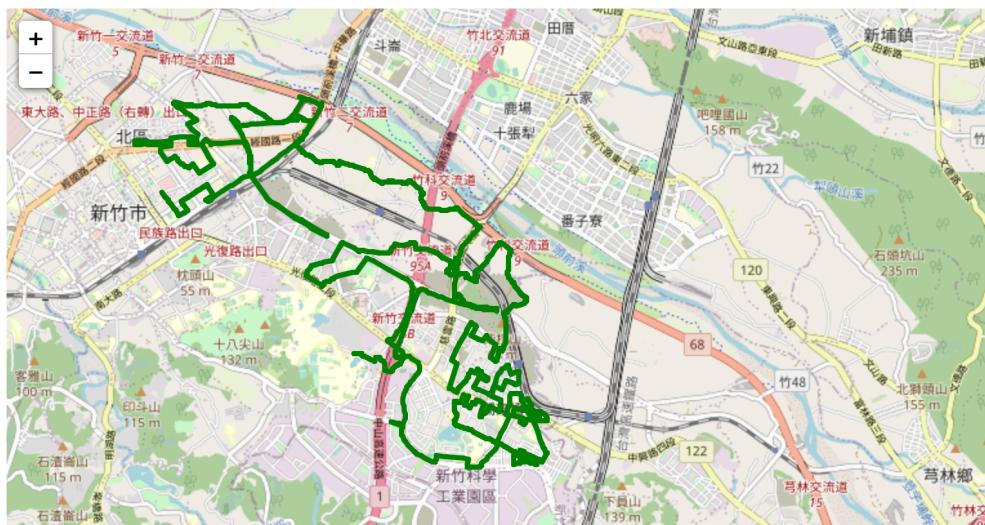
Out[6]:



DFS(recursive):

The number of nodes in the path found by DFS: 1311  
Total distance of path found by DFS: 48954.32100000007 m  
The number of visited nodes in DFS: 12037

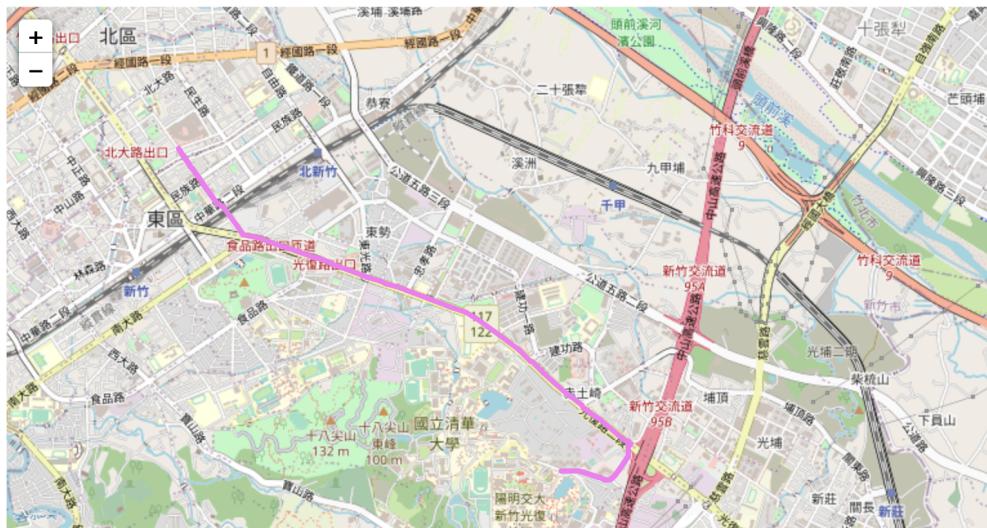
Out[7]:



## 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: 5077

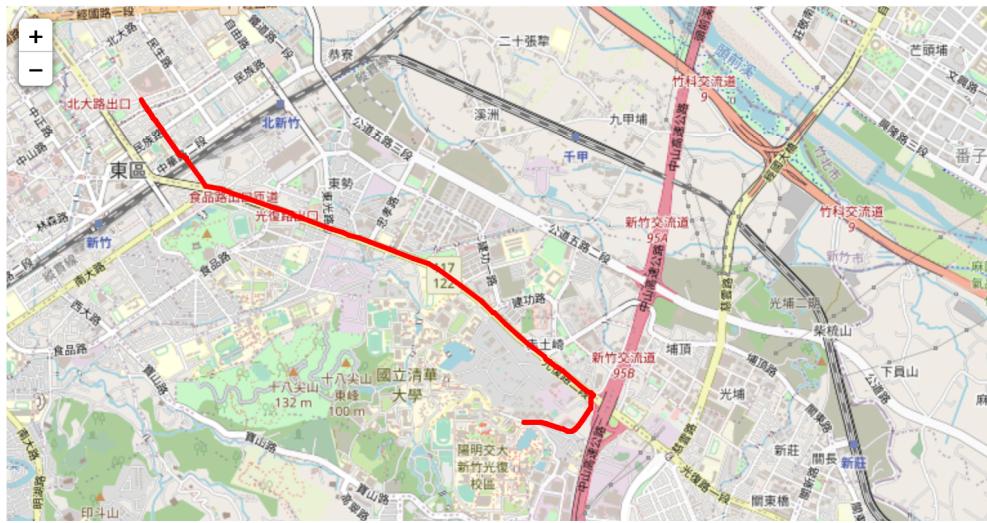
Out[8]:



## 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: 261

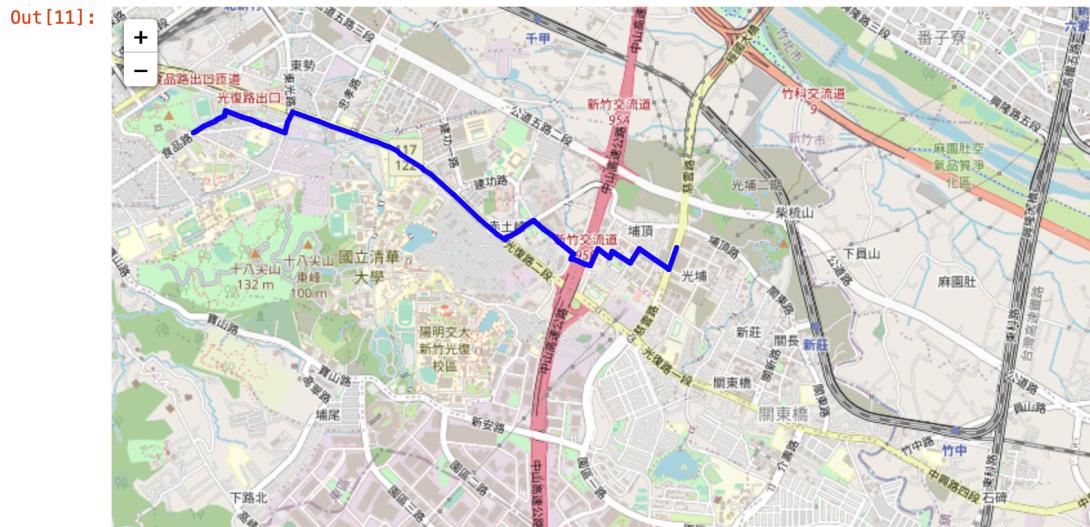
Out[9]:



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

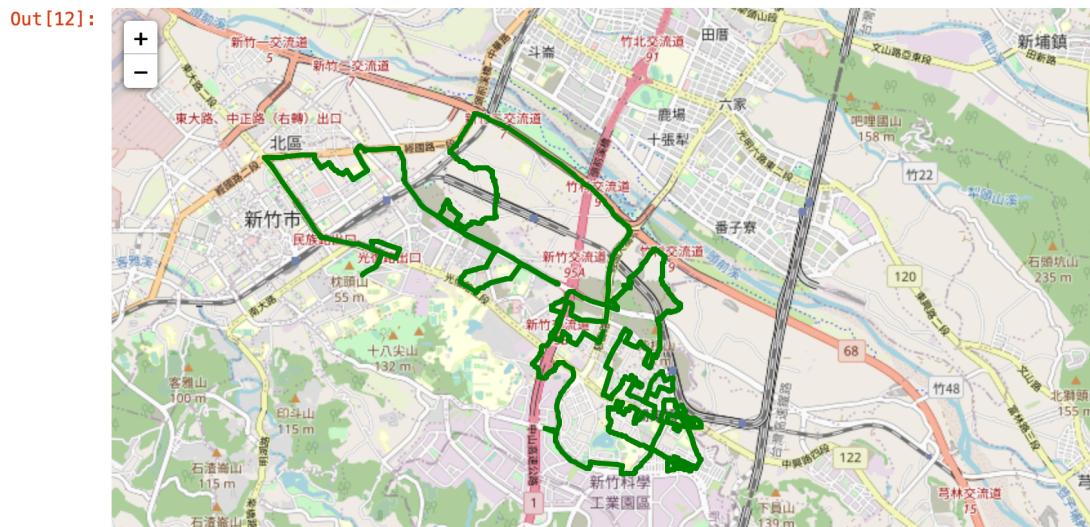
BFS:

The number of nodes in the path found by BFS: 60  
Total distance of path found by BFS: 4215.521 m  
The number of visited nodes in BFS: 4605



DFS(recursive):

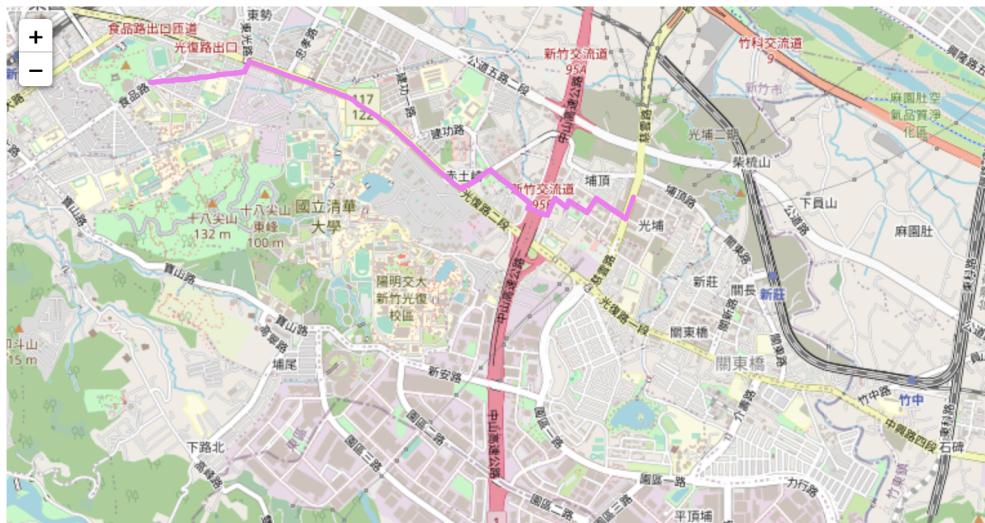
The number of nodes in the path found by DFS: 1016  
Total distance of path found by DFS: 43504.76899999994 m  
The number of visited nodes in DFS: 12037



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: 7207

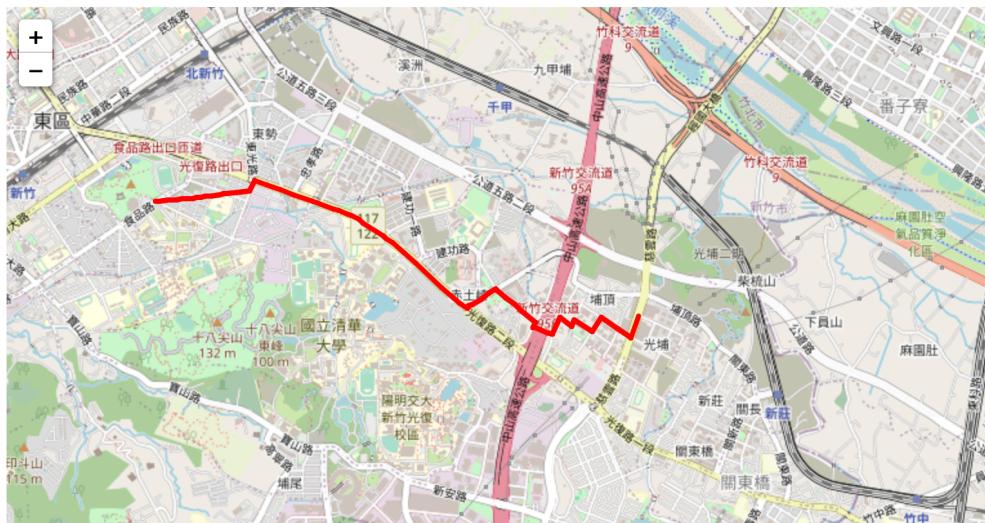
Out[13]:



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: 7858

Out[14]:



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: 15442.395000000002 m  
The number of visited nodes in BFS: 11229



DFS(recursive):

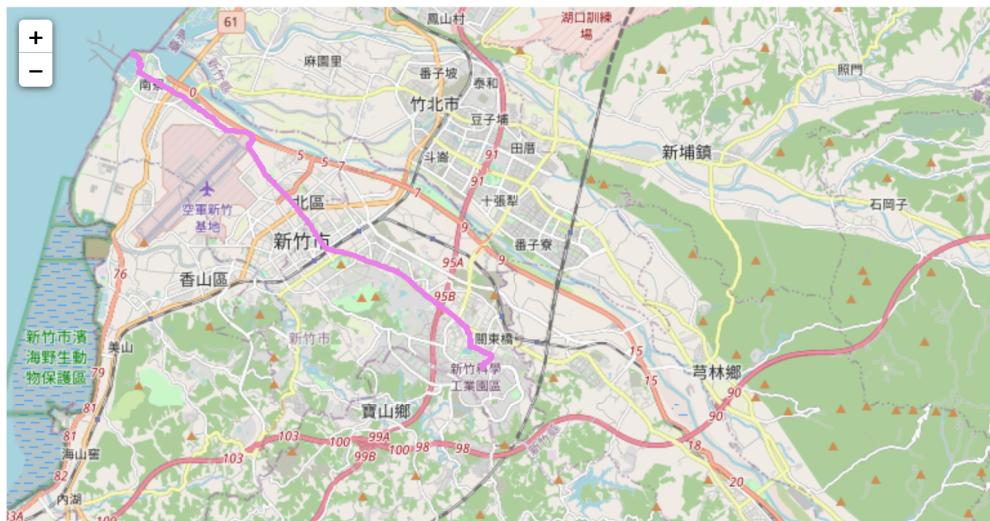
The number of nodes in the path found by DFS: 2635  
Total distance of path found by DFS: 120440.44299999984 m  
The number of visited nodes in DFS: 12037



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: 11910
```

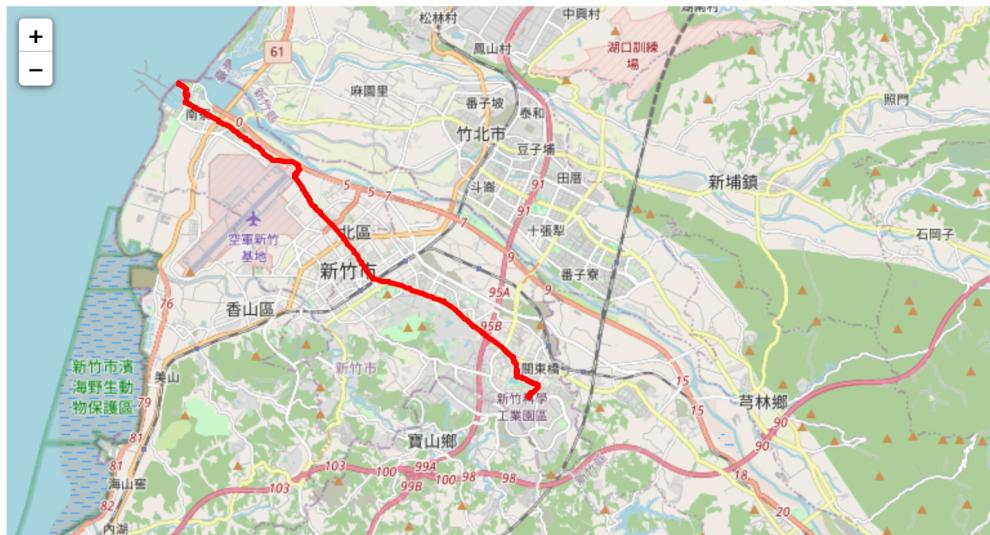
Out[20]:



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: 12025
```

Out[21]:



Discussion:

According to above analysis, we know that A\* and UCS can really give us the shortest route. For A\* may estimate faster UCS while we need appropriate heuristic function. For BFS, because our point is high density, the result is close to UCS. For DFS, it is the worst.

**Part III. Answer the questions (12%):**

1. Please describe a problem you encountered and how you solved it.

A: When I was writing dfs(recursive) it shows recursion error all the time. After I modify the recursion limit, it can run correctly. Another problem is that when I was running the main on jupyter, it can't find the function from .py. After looking the discussion on e3, some students say that if you modify the .py, you have to restart the jupyter. Therefore, I solved those problems.

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.

A: My idea is that you have to consider the vehicle. For example, motorcycle is not allow to ride on freeway, so you have to find another workable road for rider, walker.....

3. As mentioned in the introduction, a navigation system involves mapping, localization, and route finding. Please suggest possible solutions for mapping and localization components?

A: For localization, I come up with satellite. Using satellite to catch the target and catch the nearest point become start point and doing route finding again. For mapping, I come up with satellite first. However, satellite may have error and inaccurate. Then I consider that google use their car to go everywhere on the earth. Maybe, it costs more, but it can increase the safety of driver.

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 based on other attributes. Please define a dynamic heuristic function for ETA. Please explain the rationale of your design.

A: My idea is that at the beginning we can still use the distance to route, and record the driver spent how much time to pass the road section and making them to big data. So we can use pass time to become heuristic function and this function will still change. For more accuracy, we can build heuristic function for different period of day. For example, at rush hour, Baoshan road will have traffic jam. Therefore, the arrival time will become later or it can find another route for driver.

(Following is Bonus)

## Bonus:

```
5 # Begin your code (Part 6)
6 ...
7 Because my astar_time is similar to astar, I just mention the diffence
8 First: I add a state .t represent the time spent from start to the point
9     and time[] it save the speed limit
10 Second: I use .t to judge which point pop first instead of .d, which means
11     I use time to become my heuristic function
12
13
14
15
16 |
17
18 ...
19 class graph:
20     def __init__(self,num,to,dis,time):
21         self.t=0
22         self.d=0
23         self.h=0
24         self.num=num
25         self.color=0
26         self.distance=[]
27         self.time=[]
28         self.pre=-1
29         self.list=[]
30         self.list.append(to)
```

```
31         self.time.append(time)
32     def add(self,to,dis,time):
33         self.list.append(to)
34         self.distance.append(dis)
35         self.time.append(time)
36
37     def astar_time(start, end):
38         a=[]
39         b=[]
40         j=0
41         tem=0
42         file=open(edgeFile)
43         for i in file.readlines():
44             if(j>0):
45                 ii=i.split(',')
46                 if(tem==int(ii[0])):
47                     a[len(a)-1].add(int(ii[1]),float(ii[2]),float(ii[3]))
48                 else:
49                     a.append(graph(int(ii[0]),int(ii[1]),float(ii[2]),float(ii[3])))
50                     b.append(int(ii[0]))
51                     tem=int(ii[0])
52             j=j+1
53         file.close()
54
55         j=0
56         h1={}
```

```

83     for i in range(len(q)) :
84         if(q[i].t<min):
85             min=q[i].t
86             m=i
87
88         u=q[m]
89         q.pop(m)
90         u.color=1
91         count=count+1
92         if(u.num==end):
93             break
94         for i in range(len(u.list)):
95             if u.list[i] in b :
96                 l=b.index(u.list[i])
97                 if(a[l].color==0):
98                     if a[l] in q:
99                         if a[l].t>u.t+(u.distance[i]/(u.time[i]*1000/3600)):
100                             a[l].d=u.d+u.distance[i]
101                             a[l].h=u.d+u.distance[i]+h1[a[l].num]
102                             a[l].t=u.t+(u.distance[i]/(u.time[i]*1000/3600))
103                             a[l].pre=u.num
104                     else:
105                         q.append(a[l])
106                         a[l].d=u.d+u.distance[i]
107                         a[l].h=u.d+u.distance[i]+h1[a[l].num]
108                         a[l].t=u.t+(u.distance[i]/(u.time[i]*1000/3600))

```

```

57     h2={}
58     h3={}
59
60     file=open(heuristicFile)
61     for i in file.readlines():
62         if(j>0):
63             ii=i.split('\r')
64             iii=ii[0].split(',')
65             h1.update({int(iii[0]):float(iii[1])})
66             h2.update({int(iii[0]):float(iii[2])})
67             h3.update({int(iii[0]):float(iii[3])})
68         j=j+1
69     file.close()
70
71
72     s=0
73     for i in range(len(a)):
74         if(a[i].num==start):
75             s=i
76
77         count=0
78         q=[]
79         q.append(a[s])
80         while not(len(q)==0):
81             m=0
82             min=10000000

```

```

83     for i in range(len(q)) :
84         if(q[i].t<min):
85             min=q[i].t
86             m=i
87
88         u=q[m]
89         q.pop(m)
90         u.color=1
91         count=count+1
92         if(u.num==end):
93             break
94         for i in range(len(u.list)):
95             if u.list[i] in b :
96                 l=b.index(u.list[i])
97                 if(a[l].color==0):
98                     if a[l] in q:
99                         if a[l].t>u.t+(u.distance[i]/(u.time[i]*1000/3600)):
100                             a[l].d=u.d+u.distance[i]
101                             a[l].h=u.d+u.distance[i]+h1[a[l].num]
102                             a[l].t=u.t+(u.distance[i]/(u.time[i]*1000/3600))
103                             a[l].pre=u.num
104                     else:
105                         q.append(a[l])
106                         a[l].d=u.d+u.distance[i]
107                         a[l].h=u.d+u.distance[i]+h1[a[l].num]
108                         a[l].t=u.t+(u.distance[i]/(u.time[i]*1000/3600))

```

```

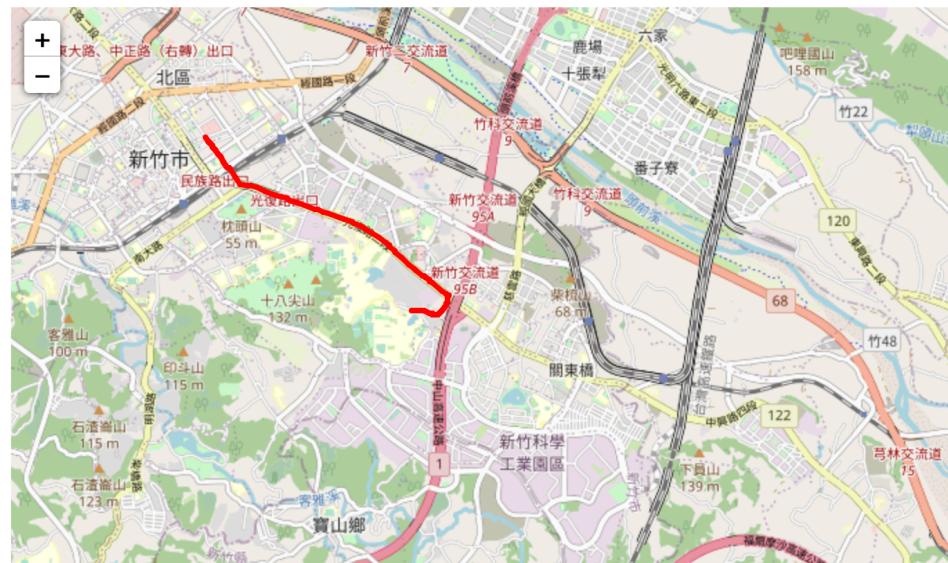
109                         a[l].pre=u.num
110
111             path=[]
112             s=0
113             for i in range(len(a)):
114                 if(a[i].num==end):
115                     s=i
116
117             dis=a[s].t
118             path.append(a[s].num)
119
120             while(1):
121                 k=a[s].pre
122                 path.append(k)
123                 if(k==start):
124                     break
125                 else:
126                     s=b.index(k)
127
128             return path , dis , count
129
130             #raise NotImplementedError("To be implemented")
131             # End your code (Part 6)

```

Test1: from National Yang Ming Chiao Tung University (ID: 2270143902) to Big City Shopping Mall (ID: 1079387396)

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: 5108

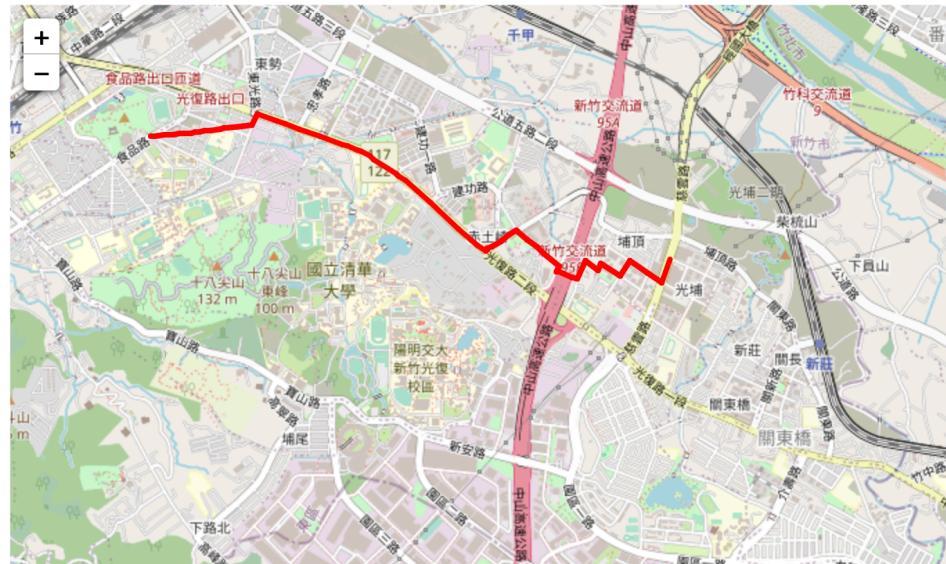
Out [25]:



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

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: 7487

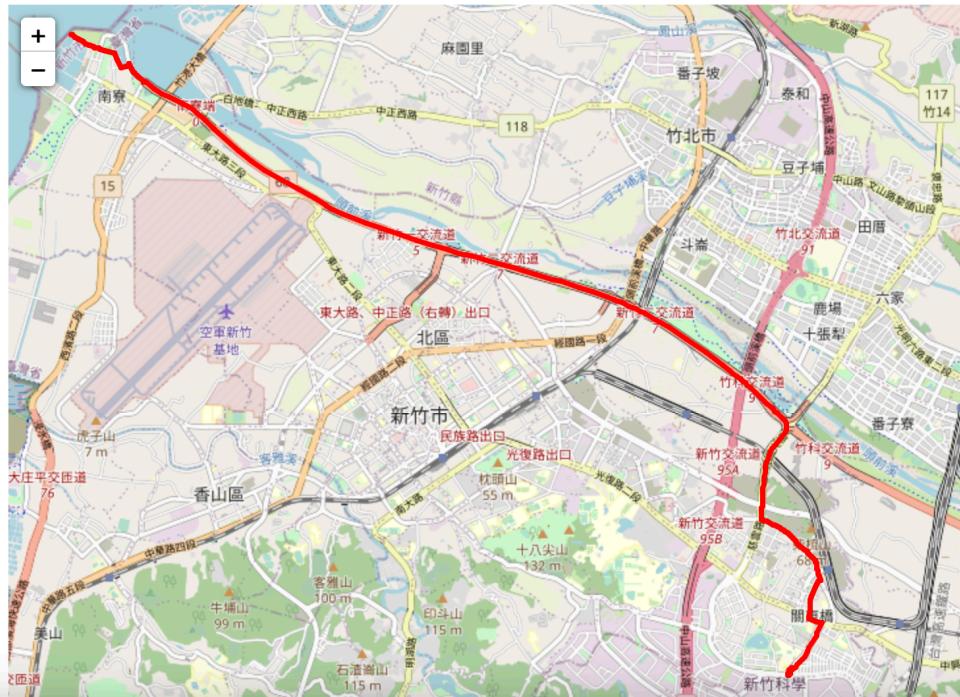
Out [27]:



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

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: 11536

Out [29]:



### Discussion

My idea is because I choose the shortest time to go every section and update the state if their has better route so i can go B from A in the shortest time.