Homework 2: Route Finding Report

109550116 楊傑宇

Part I. Implementation (6%):

```
# Begin your code (Part 1)
"""
My main idea is to create a class that record every point's state
and which point they can go...

(line 44 to 56) is the construction of my class
d          represent the distance it have passed from start point to this point
num        represent the point address
color      represent whether it have been passed
pre        represent the previous point
list[]     represent all the points this point can go
distance[] represent the distance from this points to next point respectively

(line 59 to 76) is how I record point's information to class
a[] is a array to save all the class
b[] is just a array to save the point address
    in order to find the location of a point in a[]
j   is avoid getting the first line
tem is avoid creating the same point
count is to count how many point i visited

(line 79 to 82 ) is to get the start point and put it in quene

(line 84 to 103 ) is the main bfs function
step 1: I get the point(oldest) and put all points
        which it(oldest) can go to in quene and update their state
step 2: repeat step 1 until the end point be get from quene
* need to pay attention to the point not in
  a[](these points mean can't go to another points ) need add them to a[]

(line 105 to 122 ) is to find the result return
step 1: I find the end point and use it.pre to get its previous point
        until the start point gotten
step 2: record all the point in step 2 in path[]
step 3: dis is just the end point.d
step 4: count can be return directly

"""

class graph:
    def __init__(self,num,to,dis):
        self.d=0
        self.num=num
        self.color=0
        self.distance=[]
        self.pre=-1
        self.list=[]
        self.list.append(to)
        self.distance.append(dis)
    def add(self,to,dis):
```

```
        self.list.append(to)
        self.distance.append(dis)

def bfs(start, end):
    a=[]
    b=[]
    j=0
    tem=0
    count=0

    file=open(edgeFile)
    for i in file.readlines():
        if(j>0):
            ii=i.split(',')
            if(tem==int(ii[0])):
                a[len(a)-1].add(int(ii[1]),float(ii[2]))
            else:
                a.append(graph(int(ii[0]),int(ii[1]),float(ii[2])))
                b.append(int(ii[0]))
                tem=int(ii[0])
        j=j+1
    file.close()


    s=0
    for i in range(len(a)):
        if(a[i].num==start):
            s=i

    q = queue.Queue()
    q.put(a[s])
    while not(q.empty()):
        count=count+1
        u=q.get()
        if(u.num==end):
            break
        for i in range(len(u.list)):
            if u.list[i] in b :
                l=b.index(u.list[i])
                if(a[l].color==0):
                    q.put(a[l])
                    a[l].d=u.d+u.distance[i]
                    a[l].pre=u.num
                    a[l].color=1
            else :
                a.append(graph(u.list[i],0,0))
                b.append(u.list[i])
                a[len(a)-1].d=u.d+u.distance[i]
                a[len(a)-1].pre=u.num

```

```
    path=[]
    s=0
    for i in range(len(a)):
        if(a[i].num==end):
            s=i

    dis=a[s].d
    path.append(a[s].num)

    while(1):
        k=a[s].pre
        path.append(k)
        if(k==start):
            break
        else:
            s=b.index(k)

    return path , dis , count


#raise NotImplementedError("To be implemented")
# End your code (Part 1)
```

```python
# Begin your code (Part 2)
"""
My main idea is to create a class that record every point's state
and which point they can go...

(line 31 to 43) is the construction of my class
explanation the same as part 1

(line 46 to 61) is how I record point's information to class
explanation the same as part 1
*count become c[] because i want to bring it to recusive function

(line 79 to 87 ) is to get the start point and put it in function

(line 63 to 77 ) is the main dfs function
step 1: I get all points the start point can go to and do recursive
        so it will find point and update the point's state until the leaf
        (which means it will go the road until it can't get to any new point)
* I dont set terminal condition because too complicated

(line 89 to 106 ) is to find the result return
explanation the same as part 1
"""

class graph:
    def __init__(self,num,to,dis):
        self.d=0
        self.num=num
        self.color=0
        self.distance=[]
        self.pre=-1
        self.list=[]
        self.list.append(to)
        self.distance.append(dis)
    def add(self,to,dis):
        self.list.append(to)
        self.distance.append(dis)

def dfs(start, end):
    a=[]
    b=[]
    j=0
    tem=0
    file=open(edgeFile)
    for i in file.readlines():
        if(j>0):
            ii=i.split(',')
            if(tem==int(ii[0])):
                a[len(a)-1].add(int(ii[1]),float(ii[2]))
            else:
                a.append(graph(int(ii[0]),int(ii[1]),float(ii[2])))
                b.append(int(ii[0]))
                tem=int(ii[0])
        j=j+1
    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)
```

```
# Begin your code (Part 3)
"""
My main idea is to create a class that record every point's state
and which point they can go...

(line 25 to 37) is the construction of my class
explanation the same as part 1

(line 40 to 56) is how I record point's information to class
explanation the same as part 1

(line 58 to 61 ) is to get the start point and put it in q=[]

(line 64 to 96 ) is the main ucs function
ucs is like bfs so I explain different place
(line 67 to 74) I only choose the current shorest distance to pop
(line 84 to 87) if this point have shorter path update it's state

(line 98 to 115 ) is to find the result return
explanation the same as part 1
"""
class graph:
    def __init__(self,num,to,dis):
        self.d=0
        self.num=num
        self.color=0
        self.distance=[]
        self.pre=-1
        self.list=[]
        self.list.append(to)
        self.distance.append(dis)
    def add(self,to,dis):
        self.list.append(to)
        self.distance.append(dis)

def ucs(start, end):
    a=[]
    b=[]
    j=0
    tem=0
    count=0
    file=open(edgeFile)
    for i in file.readlines():
        if(j>0):
            ii=i.split(',')
            if(tem==int(ii[0])):
                a[len(a)-1].add(int(ii[1]),float(ii[2]))
            else:
                a.append(graph(int(ii[0]),int(ii[1]),float(ii[2])))
                b.append(int(ii[0]))
                tem=int(ii[0])
        j=j+1
    file.close()

```

```
    s=0
    for i in range(len(a)):
        if(a[i].num==start):
            s=i


    q=[]
    q.append(a[s])
    while not(len(q)==0):
        m=0
        min=1000000
        for i in range(len(q)) :
            if(q[i].d<min):
                min=q[i].d
                m=i

        u=q[m]
        q.pop(m)
        u.color=1
        count=count+1
        if(u.num==end):
            break
        for i in range(len(u.list)):
            if u.list[i] in b :
                l=b.index(u.list[i])
                if(a[l].color==0):
                    if a[l] in q:
                        if a[l].d>u.d+u.distance[i]:
                            a[l].d=u.d+u.distance[i]
                            a[l].pre=u.num
                    else:
                        q.append(a[l])
                        a[l].d=u.d+u.distance[i]
                        a[l].pre=u.num
            else :
                a.append(graph(u.list[i],0,0))
                b.append(u.list[i])
                a[len(a)-1].d=u.d+u.distance[i]
                a[len(a)-1].pre=u.num

    path=[]
    s=0
    for i in range(len(a)):
        if(a[i].num==end):
            s=i

    dis=a[s].d
    path.append(a[s].num)

    while(1):
        k=a[s].pre
        path.append(k)
        if(k==start):
            break
        else:
            s=b.index(k)

    return path , dis , count
#raise NotImplementedError("To be implemented")
# End your code (Part 3)
```

```
# Begin your code (Part 4)
"""
My main idea is to create a class that record every point's state
and which point they can go...

(line 27 to 39) is the construction of my class
explanation the same as part 1
* I add a h to represent point's heuristic

(line 42 to 73) is how I record point's information to class
explanation the same as part 1
* add I use h1,h2,h3 to save heuristic.csv

(line 74 to 77 ) is to get the start point and put it in q=[]

(line 78 to 113 ) is the main astar function
actually, it is similar to ucs, however i use .d to judge which edge go first
and now, i use .h to judge

(line 114 to 130 ) is to find the result return
explanation the same as part 1
"""
class graph:
    def __init__(self,num,to,dis):
        self.d=0
        self.h=0
        self.num=num
        self.color=0
        self.distance=[]
        self.pre=-1
        self.list=[]
        self.list.append(to)
        self.distance.append(dis)
    def add(self,to,dis):
        self.list.append(to)
        self.distance.append(dis)

def astar(start, end):
    a=[]
    b=[]
    j=0
    tem=0
    count=0
    file=open(edgeFile)
    for i in file.readlines():
        if(j>0):
            ii=i.split(',')
            if(tem==int(ii[0])):
                a[len(a)-1].add(int(ii[1]),float(ii[2]))
            else:
                a.append(graph(int(ii[0]),int(ii[1]),float(ii[2])))
                b.append(int(ii[0]))
                tem=int(ii[0])
        j=j+1
    file.close()
    j=0
    h1={}
    h2={}
    h3={}
    file=open(heuristicFile)
    for i in file.readlines():
        if(j>0):
            ii=i.split('\r')
            iii=ii[0].split(',')
```

```
            h1.update({int(iii[0]):float(iii[1])})
            h2.update({int(iii[0]):float(iii[2])})
            h3.update({int(iii[0]):float(iii[3])})
        j=j+1
    file.close()
    s=0
    for i in range(len(a)):
        if(a[i].num==start):
            s=i
    q=[]
    q.append(a[s])
    while not(len(q)==0):

        m=0
        min=10000000
        for i in range(len(q)) :
            if(q[i].h<min):
                min=q[i].h
                m=i

        u=q[m]
        q.pop(m)
        u.color=1
        count=count+1
        if(u.num==end):
            break
        for i in range(len(u.list)):
            if u.list[i] in b :
                l=b.index(u.list[i])
                if(a[l].color==0):
                    if a[l] in q:
                        if a[l].d>u.d+u.distance[i]:
                            a[l].d=u.d+u.distance[i]
                            a[l].h=u.d+u.distance[i]+h1[a[l].num]
                            a[l].pre=u.num
                    else:
                        q.append(a[l])
                        a[l].d=u.d+u.distance[i]
                        a[l].h=u.d+u.distance[i]+h1[a[l].num]
                        a[l].pre=u.num
            else :
                a.append(graph(u.list[i],0,0))
                b.append(u.list[i])
                a[len(a)-1].d=u.d+u.distance[i]
                a[len(a)-1].pre=u.num
    path=[]
    s=0
    for i in range(len(a)):
        if(a[i].num==end):
            s=i
    dis=a[s].d
    path.append(a[s].num)
    while(1):
        k=a[s].pre
        path.append(k)
        if(k==start):
            break
        else:
            s=b.index(k)
    return path , dis , count
#raise NotImplementedError("To be implemented")
# 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:

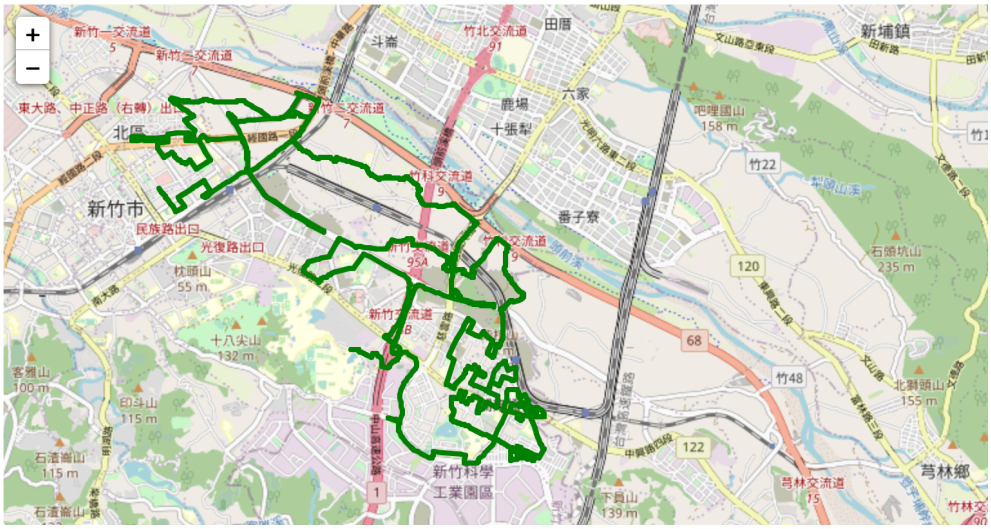
```
The number of nodes in the path found by BFS: 88
Total distance of path found by BFS: 4978.8820000000005 m
The number of visited nodes in BFS: 4268
```

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:

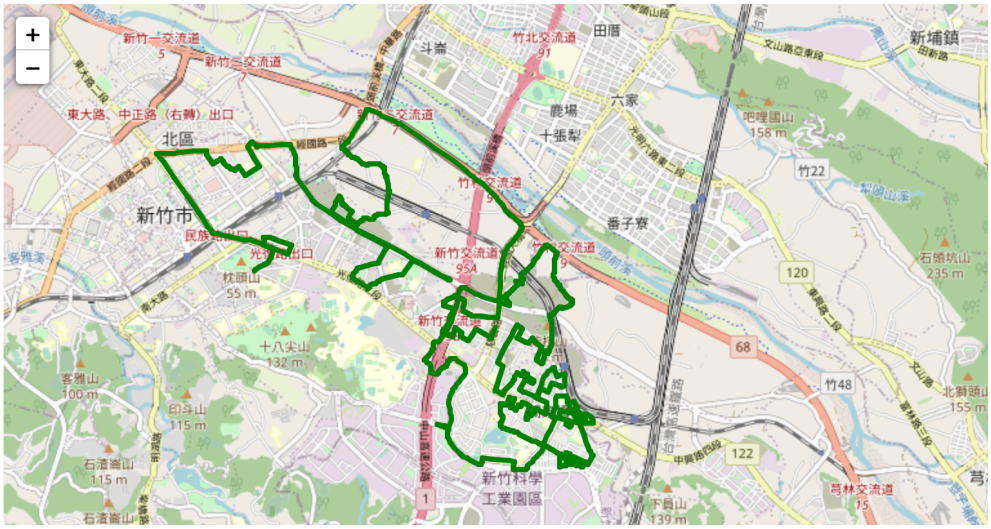
```
The number of nodes in the path found by BFS: 60
Total distance of path found by BFS: 4215.521 m
The number of visited nodes in BFS: 4605
```

Out[11]:

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
```

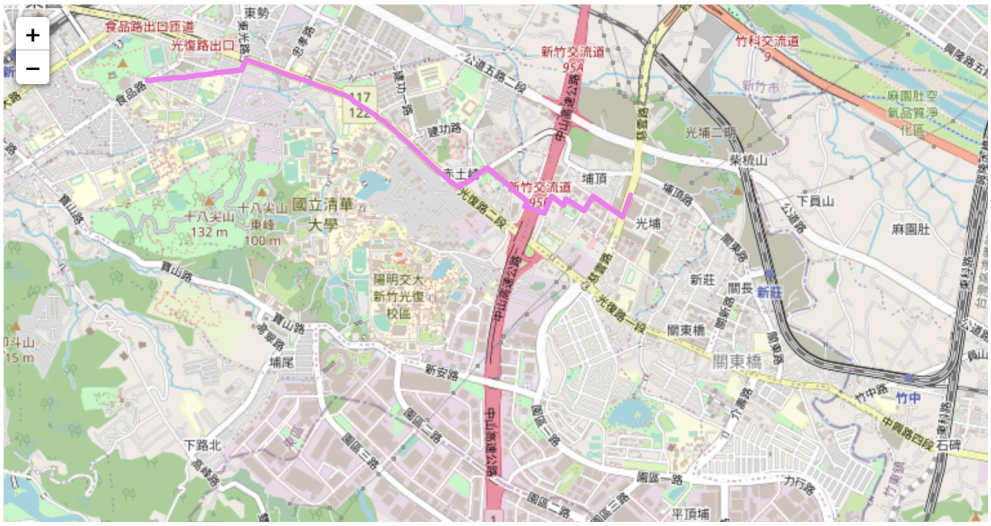
Out[12]:



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:

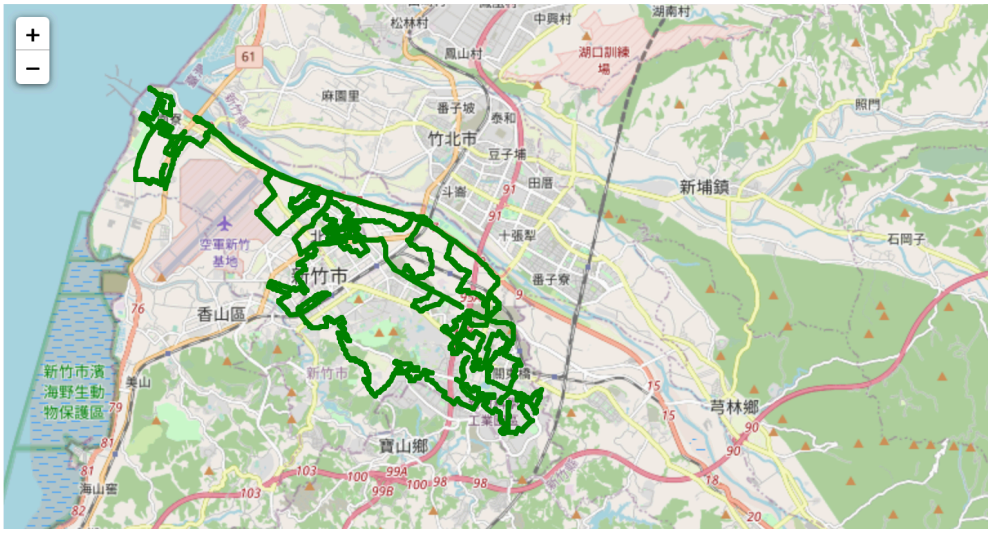
```
The number of nodes in the path found by BFS: 183
Total distance of path found by BFS: 15442.395000000002 m
The number of visited nodes in BFS: 11229
```

Out[18]:

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
```

Out[19]:



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:

```
# Begin your code (Part 6)
'''
Because my astar_time is similar to astar, I just mention the diffence
First:  I add a state .t represent the time spent from start to the point
        and time[] it save the speed limit
Second: I use .t to judge which point pop first instead of .d, which means
        I use time to become my heuristic function




'''
class graph:
    def __init__(self,num,to,dis,time):
        self.t=0
        self.d=0
        self.h=0
        self.num=num
        self.color=0
        self.distance=[]
        self.time=[]
        self.pre=-1
        self.list=[]
        self.list.append(to)
        self.time.append(time)
    def add(self,to,dis,time):
        self.list.append(to)
        self.distance.append(dis)
        self.time.append(time)

def astar_time(start, end):
    a=[]
    b=[]
    j=0
    tem=0
    file=open(edgeFile)
    for i in file.readlines():
        if(j>0):
            ii=i.split(',')
            if(tem==int(ii[0])):
                a[len(a)-1].add(int(ii[1]),float(ii[2]),float(ii[3]))
            else:
                a.append(graph(int(ii[0]),int(ii[1]),float(ii[2]),float(ii[3])))
                b.append(int(ii[0]))
                tem=int(ii[0])
        j=j+1
    file.close()

    j=0
    h1={}
```

```
        for i in range(len(q)) :
            if(q[i].t<min):
                min=q[i].t
                m=i

        u=q[m]
        q.pop(m)
        u.color=1
        count=count+1
        if(u.num==end):
            break
        for i in range(len(u.list)):
            if u.list[i] in b :
                l=b.index(u.list[i])
                if(a[l].color==0):
                    if a[l] in q:
                        if a[l].t>u.t+(u.distance[i]/(u.time[i]*1000/3600)):
                            a[l].d=u.d+u.distance[i]
                            a[l].h=u.d+u.distance[i]+h1[a[l].num]
                            a[l].t=u.t+(u.distance[i]/(u.time[i]*1000/3600))
                            a[l].pre=u.num
                    else:
                        q.append(a[l])
                        a[l].d=u.d+u.distance[i]
                        a[l].h=u.d+u.distance[i]+h1[a[l].num]
                        a[l].t=u.t+(u.distance[i]/(u.time[i]*1000/3600))
```

```
    h2={}
    h3={}

    file=open(heuristicFile)
    for i in file.readlines():
        if(j>0):
            ii=i.split('\r')
            iii=ii[0].split(',')
            h1.update({int(iii[0]):float(iii[1])})
            h2.update({int(iii[0]):float(iii[2])})
            h3.update({int(iii[0]):float(iii[3])})
        j=j+1
    file.close()


    s=0
    for i in range(len(a)):
        if(a[i].num==start):
            s=i

    count=0
    q=[]
    q.append(a[s])
    while not(len(q)==0):
        m=0
        min=10000000
```

```
        for i in range(len(q)) :
            if(q[i].t<min):
                min=q[i].t
                m=i

        u=q[m]
        q.pop(m)
        u.color=1
        count=count+1
        if(u.num==end):
            break
        for i in range(len(u.list)):
            if u.list[i] in b :
                l=b.index(u.list[i])
                if(a[l].color==0):
                    if a[l] in q:
                        if a[l].t>u.t+(u.distance[i]/(u.time[i]*1000/3600)):
                            a[l].d=u.d+u.distance[i]
                            a[l].h=u.d+u.distance[i]+h1[a[l].num]
                            a[l].t=u.t+(u.distance[i]/(u.time[i]*1000/3600))
                            a[l].pre=u.num
                    else:
                        q.append(a[l])
                        a[l].d=u.d+u.distance[i]
                        a[l].h=u.d+u.distance[i]+h1[a[l].num]
                        a[l].t=u.t+(u.distance[i]/(u.time[i]*1000/3600))
```

```
                        a[l].pre=u.num

    path=[]
    s=0
    for i in range(len(a)):
        if(a[i].num==end):
            s=i

    dis=a[s].t
    path.append(a[s].num)

    while(1):
        k=a[s].pre
        path.append(k)
        if(k==start):
            break
        else:
            s=b.index(k)

    return path , dis , count

    #raise NotImplementedError("To be implemented")
    # End your code (Part 6)
```

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

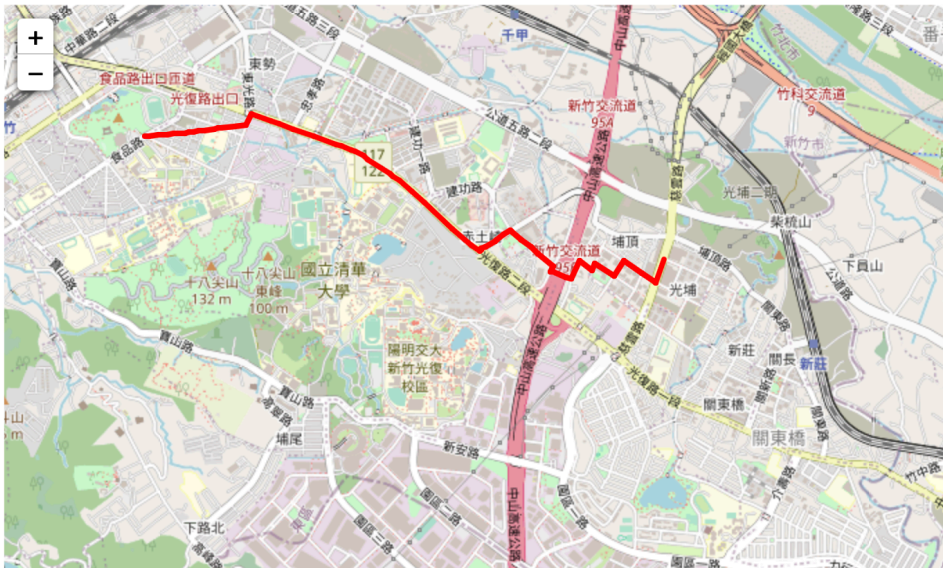
```
The number of nodes in the path found by A* search: 89
Total second of path found by A* search: 320.87823163083164 s
The number of visited nodes in A* search: 5108
```

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
```

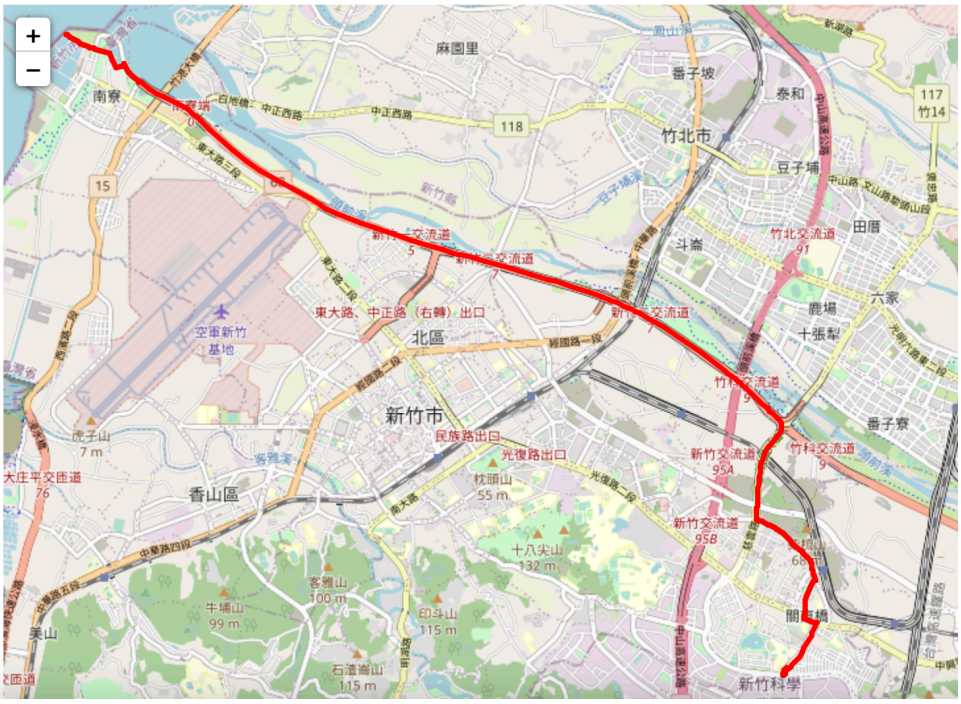
Out[27]:



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.