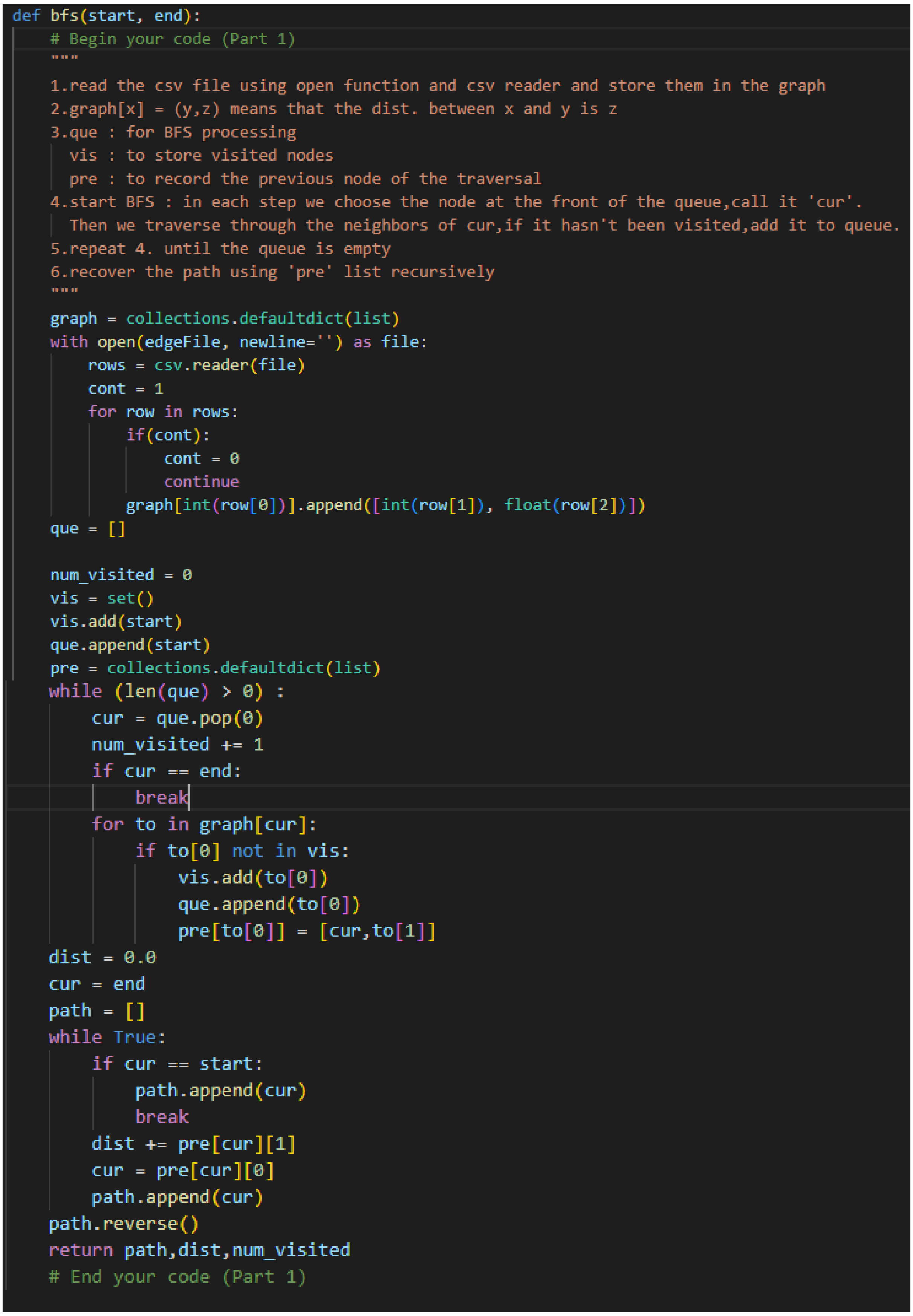
110550175鄭栩安

Part I. Implementation

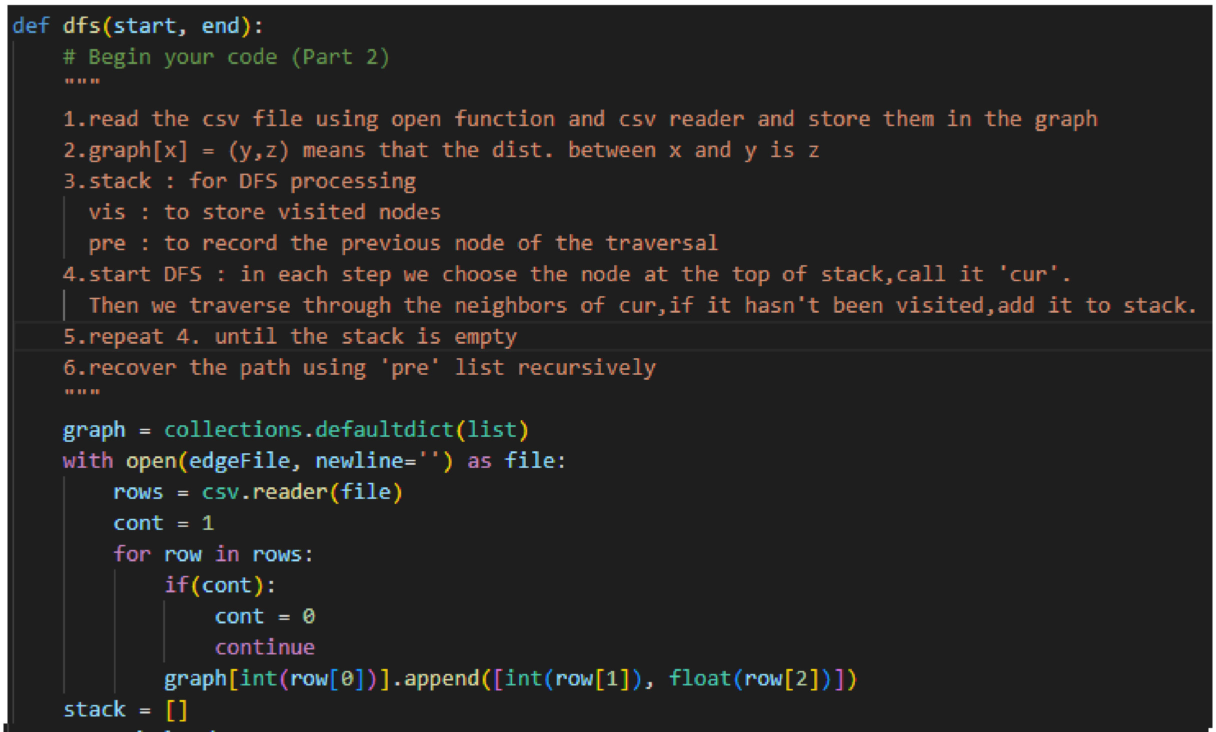
(1)BFS



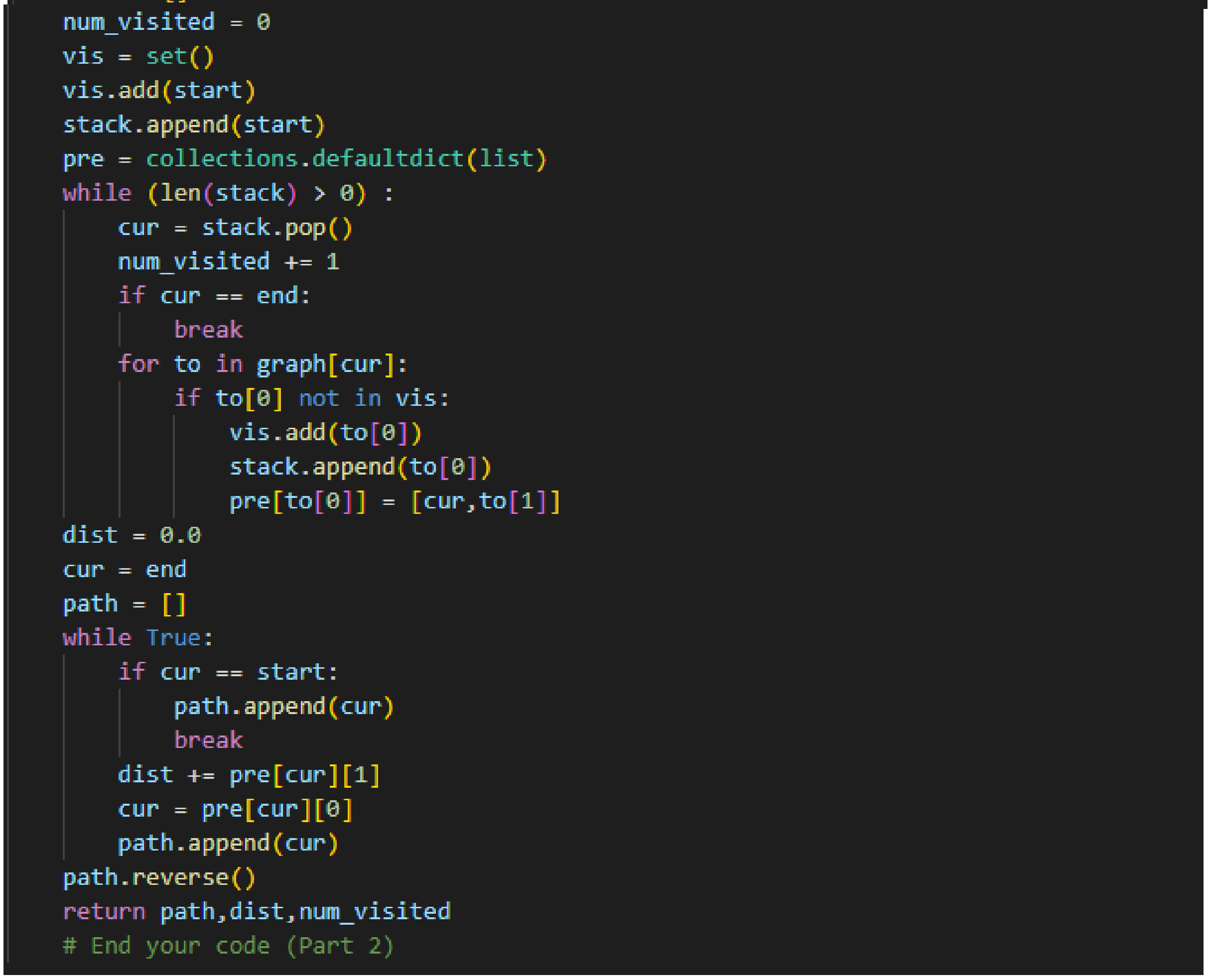
```python
def bfs(start, end):
    # Begin your code (Part 1)
    """
    1.read the csv file using open function and csv reader and store them in the graph
    2.graph[x] = (y,z) means that the dist. between x and y is z
    3.que : for BFS processing
      vis : to store visited nodes
      pre : to record the previous node of the traversal
    4.start BFS : in each step we choose the node at the front of the queue,call it 'cur'.
      Then we traverse through the neighbors of cur,if it hasn't been visited,add it to queue.
    5.repeat 4. until the queue is empty
    6.recover the path using 'pre' list recursively
    """
    graph = collections.defaultdict(list)
    with open(edgeFile, newline='') as file:
        rows = csv.reader(file)
        cont = 1
        for row in rows:
            if(cont):
                cont = 0
                continue
            graph[int(row[0])].append([int(row[1]), float(row[2])])
    que = []

    num_visited = 0
    vis = set()
    vis.add(start)
    que.append(start)
    pre = collections.defaultdict(list)
    while (len(que) > 0) :
        cur = que.pop(0)
        num_visited += 1
        if cur == end:
            break
        for to in graph[cur]:
            if to[0] not in vis:
                vis.add(to[0])
                que.append(to[0])
                pre[to[0]] = [cur,to[1]]
    dist = 0.0
    cur = end
    path = []
    while True:
        if cur == start:
            path.append(cur)
            break
        dist += pre[cur][1]
        cur = pre[cur][0]
        path.append(cur)
    path.reverse()
    return path,dist,num_visited
    # End your code (Part 1)
```

110550175鄭栩安

(2)DFS



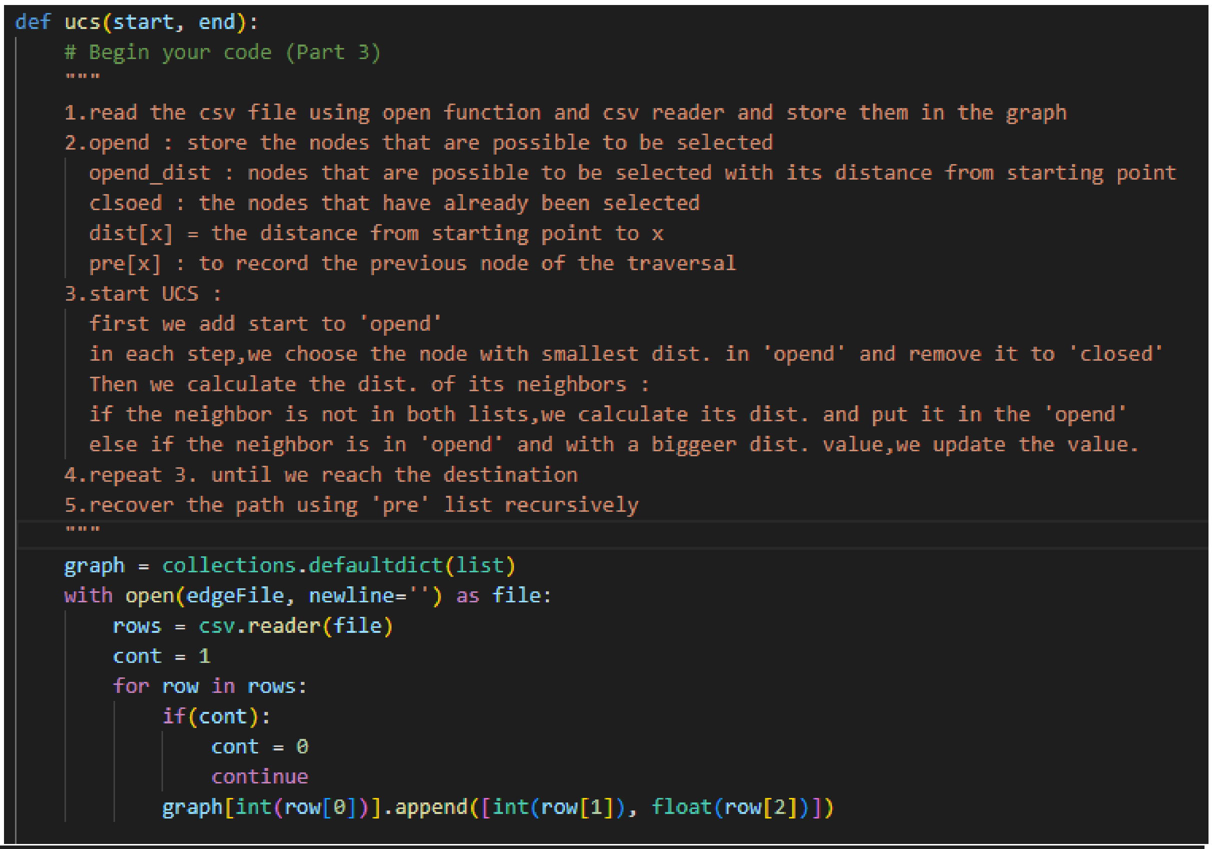
```python
def dfs(start, end):
    # Begin your code (Part 2)
    """
    1.read the csv file using open function and csv reader and store them in the graph
    2.graph[x] = (y,z) means that the dist. between x and y is z
    3.stack : for DFS processing
      vis : to store visited nodes
      pre : to record the previous node of the traversal
    4.start DFS : in each step we choose the node at the top of stack,call it 'cur'.
      Then we traverse through the neighbors of cur,if it hasn't been visited,add it to stack.
    5.repeat 4. until the stack is empty
    6.recover the path using 'pre' list recursively
    """
    graph = collections.defaultdict(list)
    with open(edgeFile, newline='') as file:
        rows = csv.reader(file)
        cont = 1
        for row in rows:
            if(cont):
                cont = 0
                continue
            graph[int(row[0])].append([int(row[1]), float(row[2])])
    stack = []
```



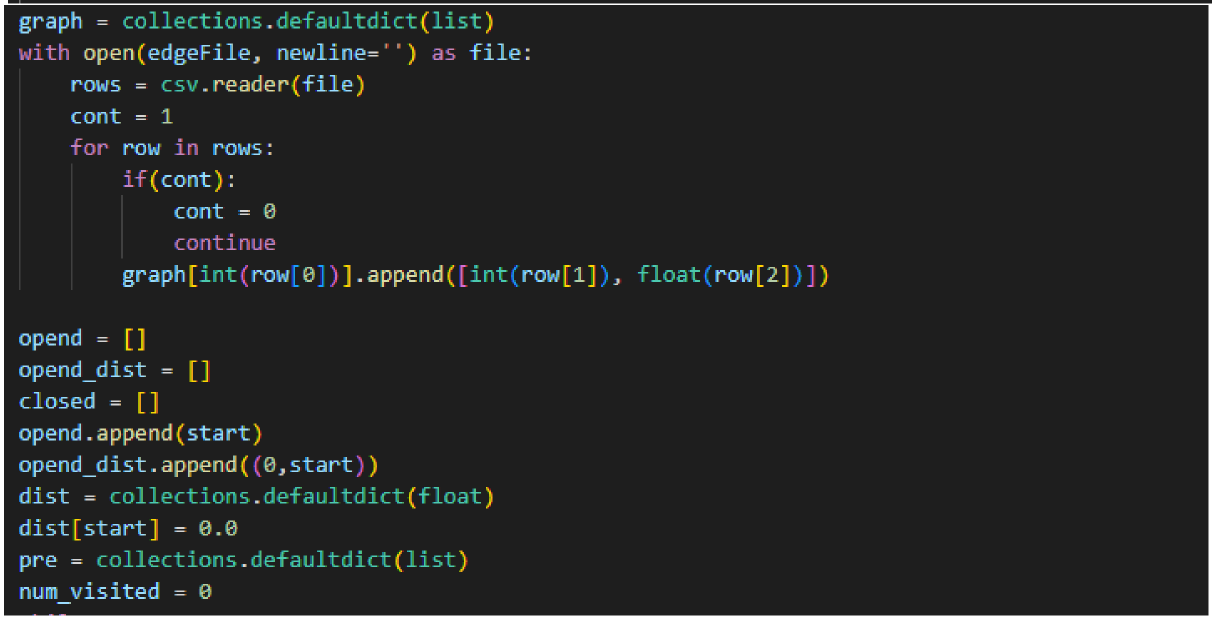
```python
    num_visited = 0
    vis = set()
    vis.add(start)
    stack.append(start)
    pre = collections.defaultdict(list)
    while (len(stack) > 0) :
        cur = stack.pop()
        num_visited += 1
        if cur == end:
            break
        for to in graph[cur]:
            if to[0] not in vis:
                vis.add(to[0])
                stack.append(to[0])
                pre[to[0]] = [cur,to[1]]
    dist = 0.0
    cur = end
    path = []
    while True:
        if cur == start:
            path.append(cur)
            break
        dist += pre[cur][1]
        cur = pre[cur][0]
        path.append(cur)
    path.reverse()
    return path,dist,num_visited
    # End your code (Part 2)
```

110550175鄭栩安

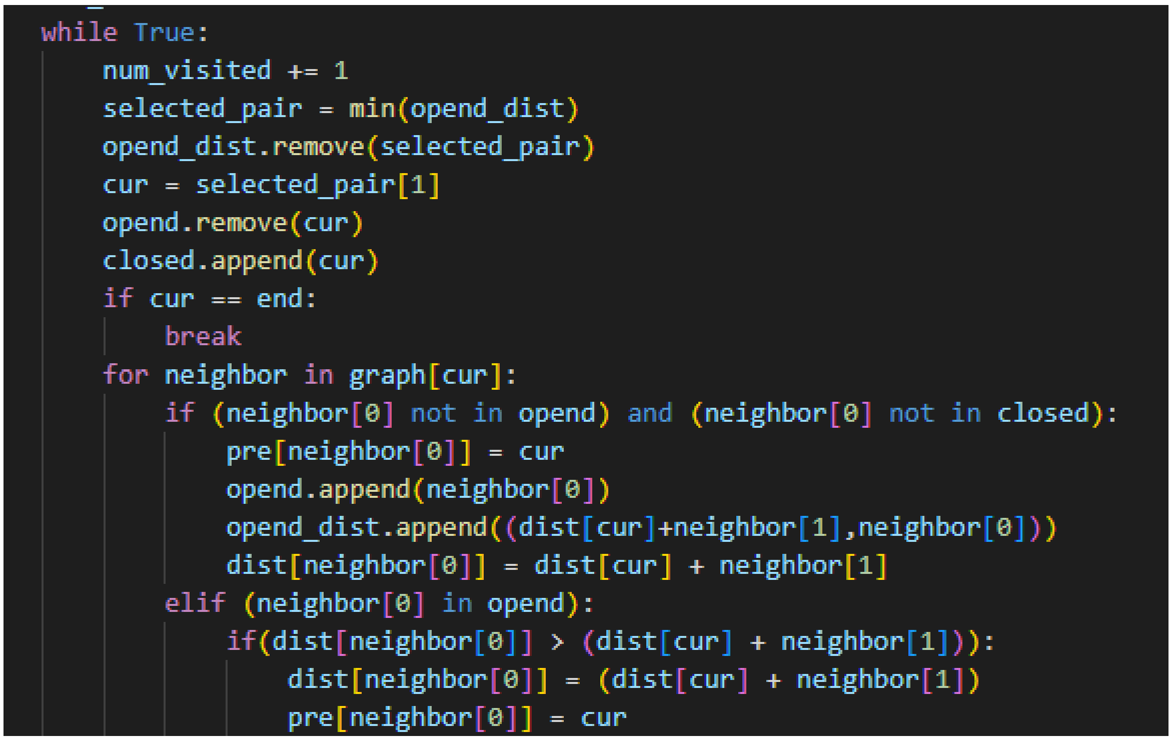
(3)USC



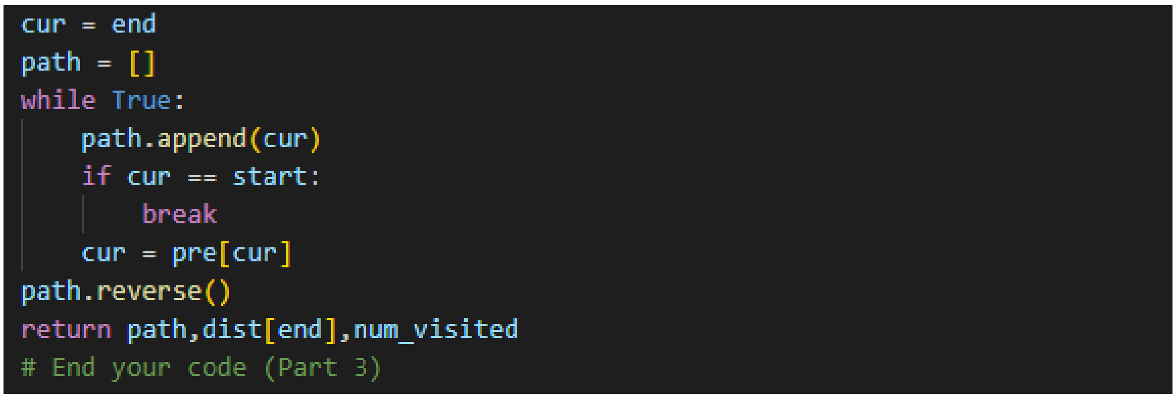
```python
def ucs(start, end):
    # Begin your code (Part 3)
    """
    1.read the csv file using open function and csv reader and store them in the graph
    2.opend : store the nodes that are possible to be selected
      opend_dist : nodes that are possible to be selected with its distance from starting point
      clsoed : the nodes that have already been selected
      dist[x] = the distance from starting point to x
      pre[x] : to record the previous node of the traversal
    3.start UCS :
      first we add start to 'opend'
      in each step,we choose the node with smallest dist. in 'opend' and remove it to 'closed'
      Then we calculate the dist. of its neighbors :
      if the neighbor is not in both lists,we calculate its dist. and put it in the 'opend'
      else if the neighbor is in 'opend' and with a biggeer dist. value,we update the value.
    4.repeat 3. until we reach the destination
    5.recover the path using 'pre' list recursively
    """
    graph = collections.defaultdict(list)
    with open(edgeFile, newline='') as file:
        rows = csv.reader(file)
        cont = 1
        for row in rows:
            if(cont):
                cont = 0
                continue
            graph[int(row[0])].append([int(row[1]), float(row[2])])
```



```python
graph = collections.defaultdict(list)
with open(edgeFile, newline='') as file:
    rows = csv.reader(file)
    cont = 1
    for row in rows:
        if(cont):
            cont = 0
            continue
        graph[int(row[0])].append([int(row[1]), float(row[2])])

opend = []
opend_dist = []
closed = []
opend.append(start)
opend_dist.append((0,start))
dist = collections.defaultdict(float)
dist[start] = 0.0
pre = collections.defaultdict(list)
num_visited = 0
```

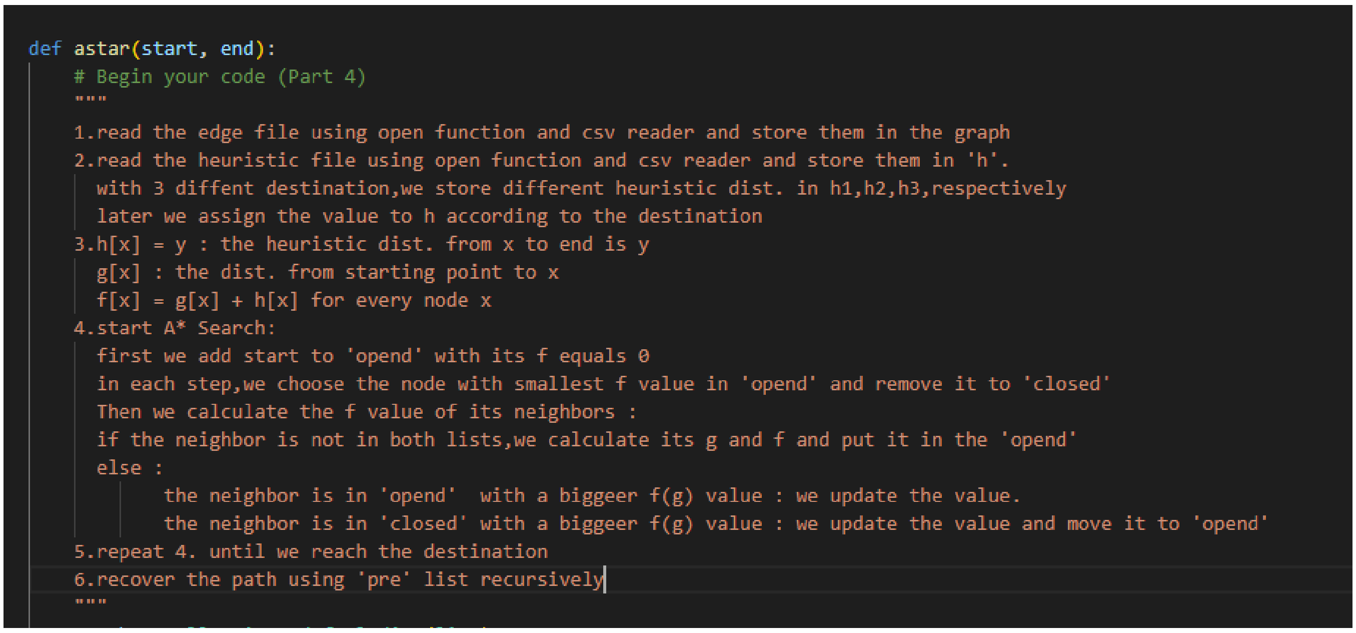


```
while True:
    num_visited += 1
    selected_pair = min(opend_dist)
    opend_dist.remove(selected_pair)
    cur = selected_pair[1]
    opend.remove(cur)
    closed.append(cur)
    if cur == end:
        break
    for neighbor in graph[cur]:
        if (neighbor[0] not in opend) and (neighbor[0] not in closed):
            pre[neighbor[0]] = cur
            opend.append(neighbor[0])
            opend_dist.append((dist[cur]+neighbor[1],neighbor[0]))
            dist[neighbor[0]] = dist[cur] + neighbor[1]
        elif (neighbor[0] in opend):
            if(dist[neighbor[0]] > (dist[cur] + neighbor[1])):
                dist[neighbor[0]] = (dist[cur] + neighbor[1])
                pre[neighbor[0]] = cur
```

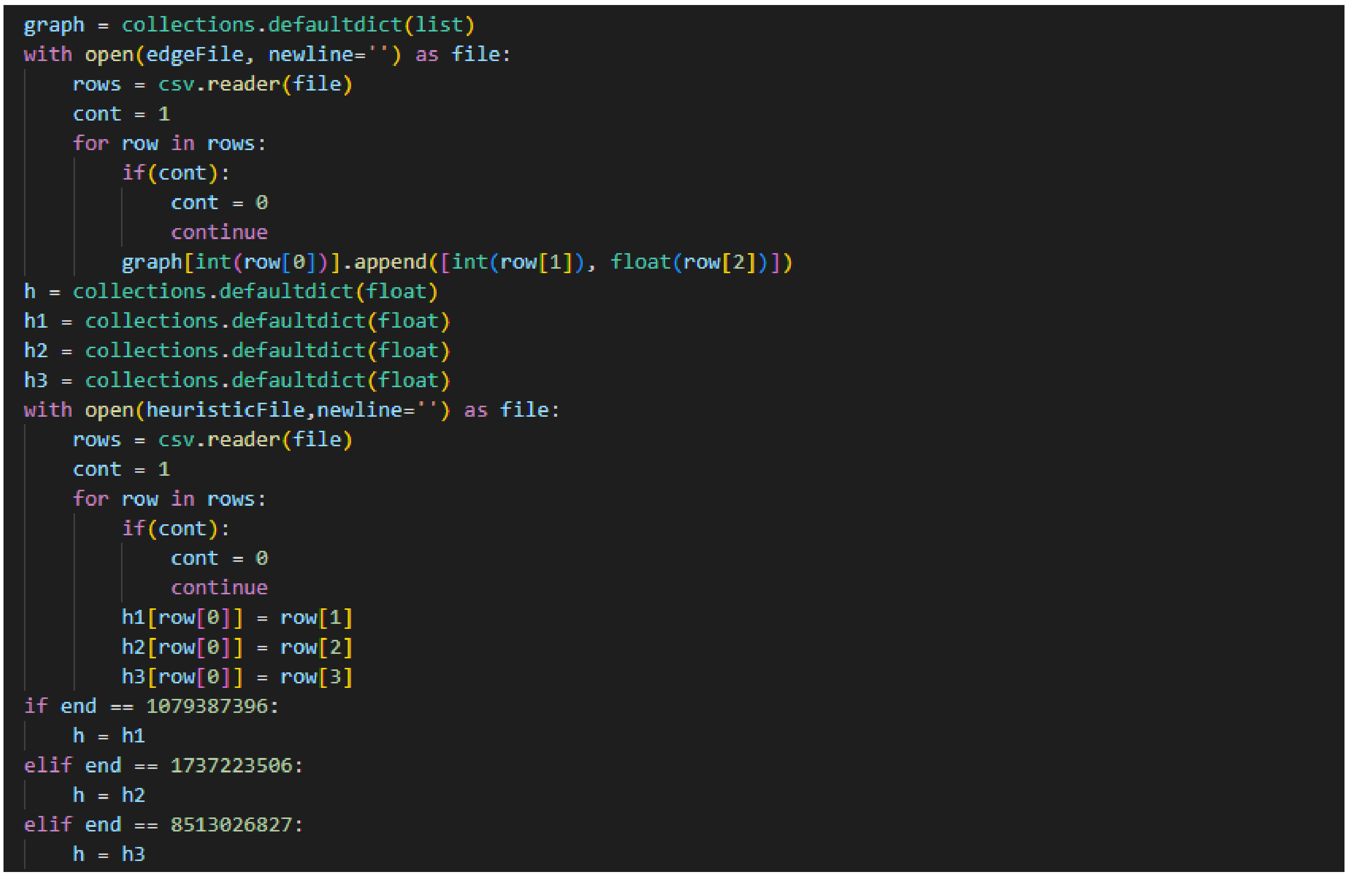


```
cur = end
path = []
while True:
    path.append(cur)
    if cur == start:
        break
    cur = pre[cur]
path.reverse()
return path,dist[end],num_visited
# End your code (Part 3)
```

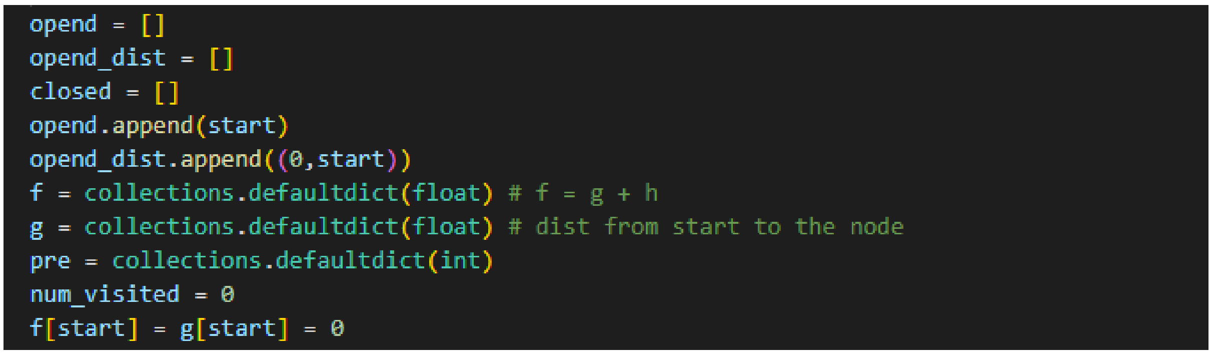
(4) A* Search



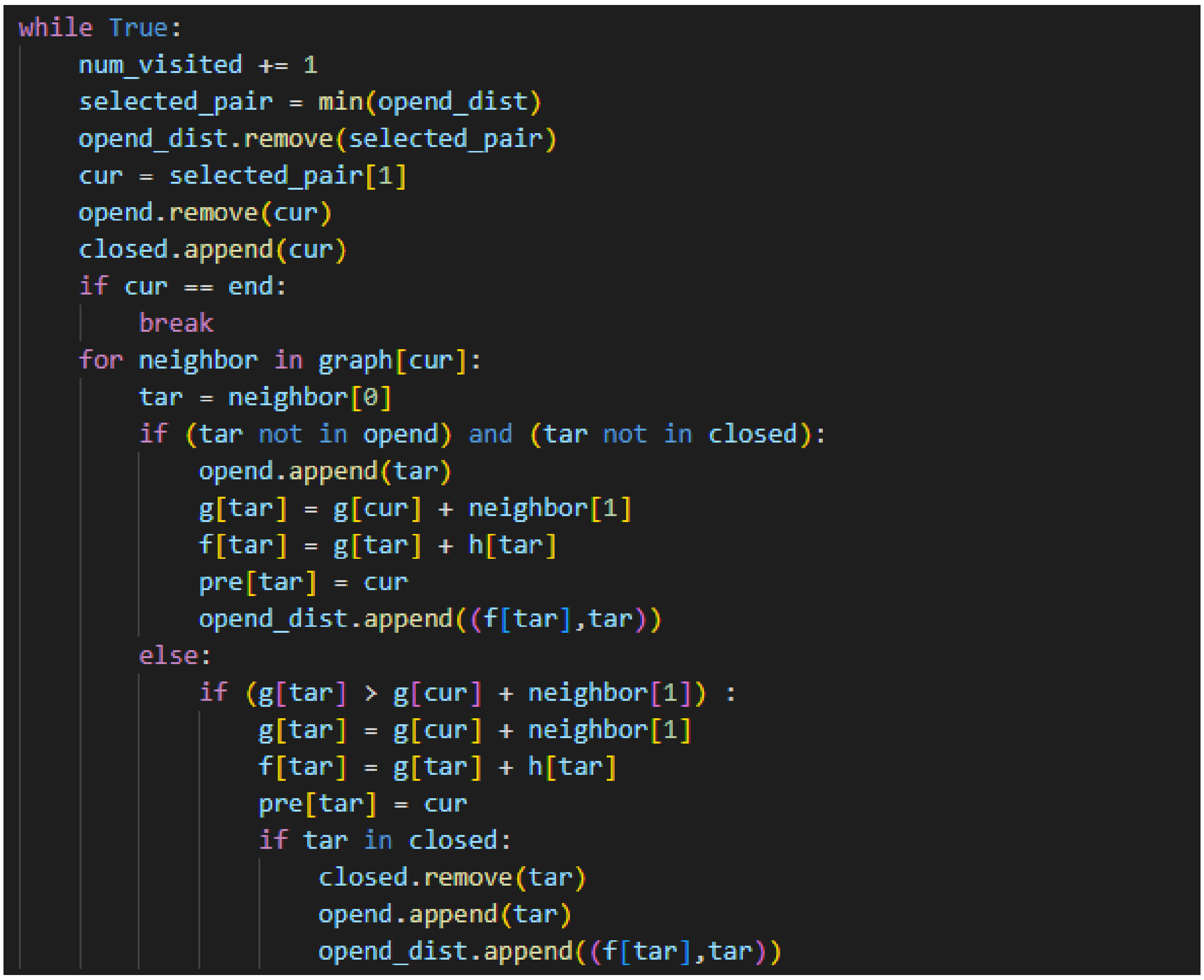
```
def astar(start, end):
    # Begin your code (Part 4)
    """
    1.read the edge file using open function and csv reader and store them in the graph
    2.read the heuristic file using open function and csv reader and store them in 'h'.
      with 3 diffent destination,we store different heuristic dist. in h1,h2,h3,respectively
      later we assign the value to h according to the destination
    3.h[x] = y : the heuristic dist. from x to end is y
      g[x] : the dist. from starting point to x
      f[x] = g[x] + h[x] for every node x
    4.start A* Search:
      first we add start to 'opend' with its f equals 0
      in each step,we choose the node with smallest f value in 'opend' and remove it to 'closed'
      Then we calculate the f value of its neighbors :
      if the neighbor is not in both lists,we calculate its g and f and put it in the 'opend'
      else :
            the neighbor is in 'opend'  with a biggeer f(g) value : we update the value.
            the neighbor is in 'closed' with a biggeer f(g) value : we update the value and move it to 'opend'
    5.repeat 4. until we reach the destination
    6.recover the path using 'pre' list recursively
    """
```



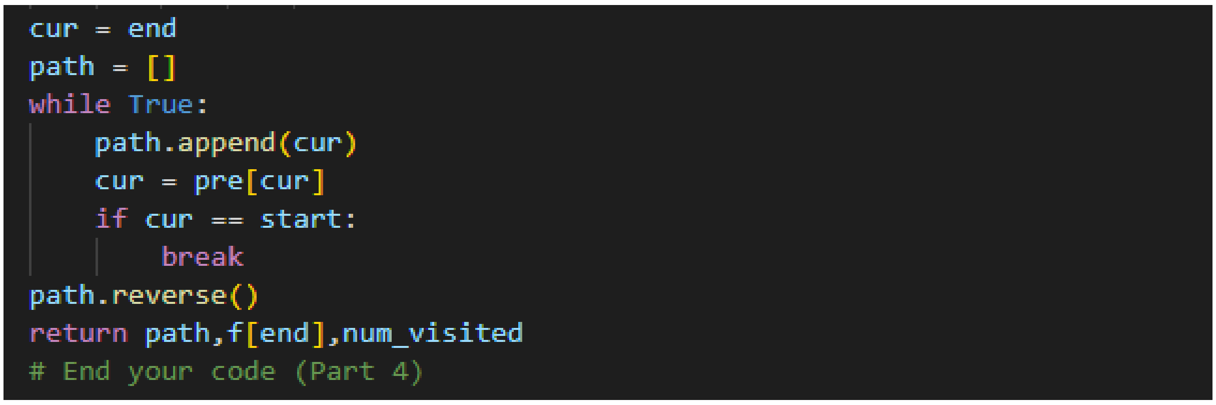
```
graph = collections.defaultdict(list)
with open(edgeFile, newline='') as file:
    rows = csv.reader(file)
    cont = 1
    for row in rows:
        if(cont):
            cont = 0
            continue
        graph[int(row[0])].append([int(row[1]), float(row[2])])
h = collections.defaultdict(float)
h1 = collections.defaultdict(float)
h2 = collections.defaultdict(float)
h3 = collections.defaultdict(float)
with open(heuristicFile,newline='') as file:
    rows = csv.reader(file)
    cont = 1
    for row in rows:
        if(cont):
            cont = 0
            continue
        h1[row[0]] = row[1]
        h2[row[0]] = row[2]
        h3[row[0]] = row[3]
if end == 1079387396:
    h = h1
elif end == 1737223506:
    h = h2
elif end == 8513026827:
    h = h3
```



```python
opend = []
opend_dist = []
closed = []
opend.append(start)
opend_dist.append((0,start))
f = collections.defaultdict(float) # f = g + h
g = collections.defaultdict(float) # dist from start to the node
pre = collections.defaultdict(int)
num_visited = 0
f[start] = g[start] = 0
```



```python
while True:
    num_visited += 1
    selected_pair = min(opend_dist)
    opend_dist.remove(selected_pair)
    cur = selected_pair[1]
    opend.remove(cur)
    closed.append(cur)
    if cur == end:
        break
    for neighbor in graph[cur]:
        tar = neighbor[0]
        if (tar not in opend) and (tar not in closed):
            opend.append(tar)
            g[tar] = g[cur] + neighbor[1]
            f[tar] = g[tar] + h[tar]
            pre[tar] = cur
            opend_dist.append((f[tar],tar))
        else:
            if (g[tar] > g[cur] + neighbor[1]) :
                g[tar] = g[cur] + neighbor[1]
                f[tar] = g[tar] + h[tar]
                pre[tar] = cur
                if tar in closed:
                    closed.remove(tar)
                    opend.append(tar)
                    opend_dist.append((f[tar],tar))
```



```python
cur = end
path = []
while True:
    path.append(cur)
    cur = pre[cur]
    if cur == start:
        break
path.reverse()
return path,f[end],num_visited
# End your code (Part 4)
```

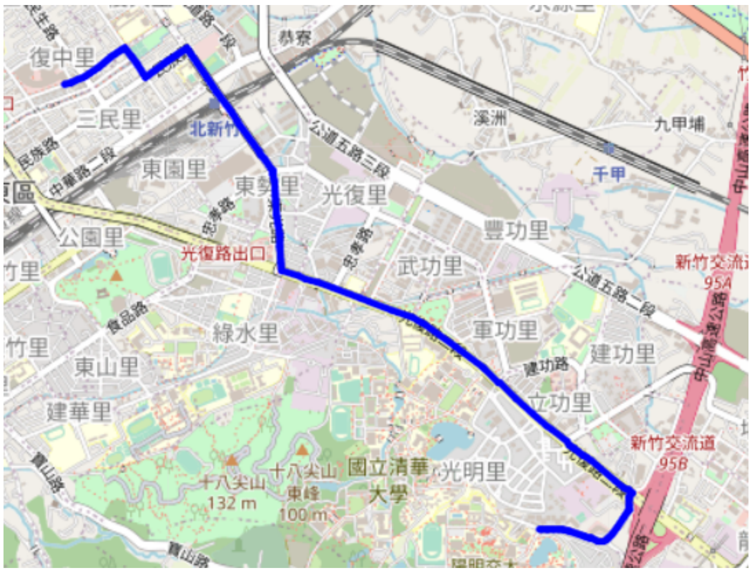
110550175鄭栩安

Part II. Results ：

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

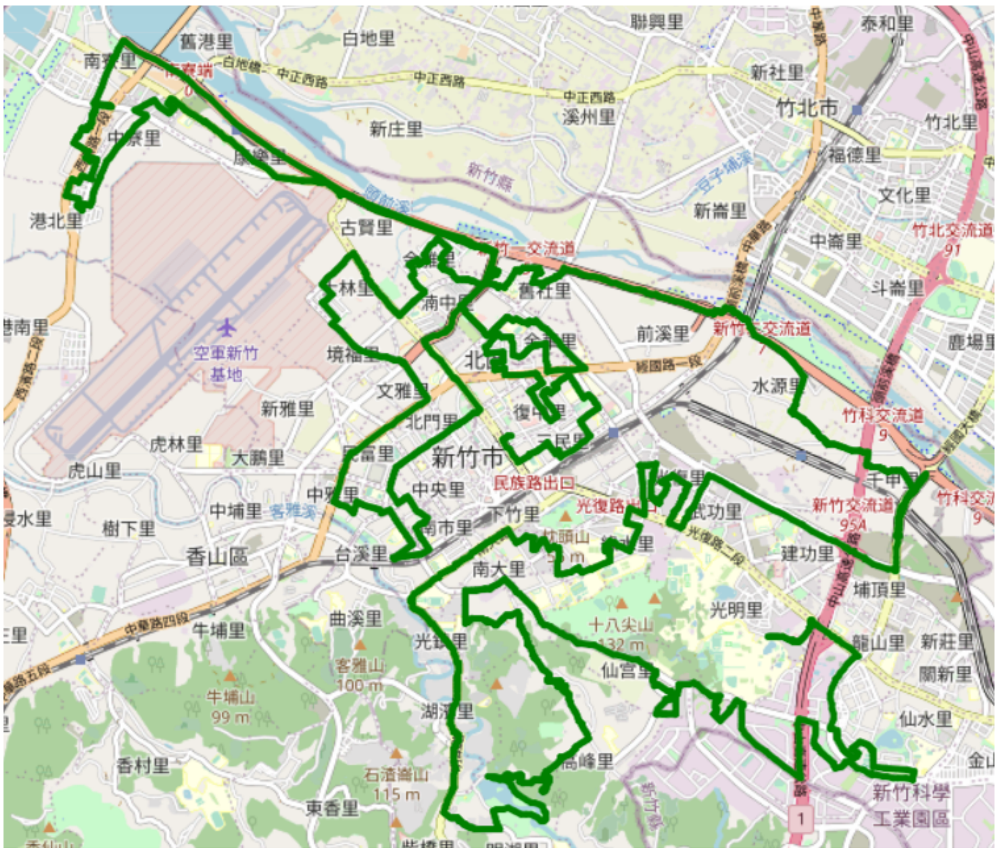
BFS

```
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: 4273
```



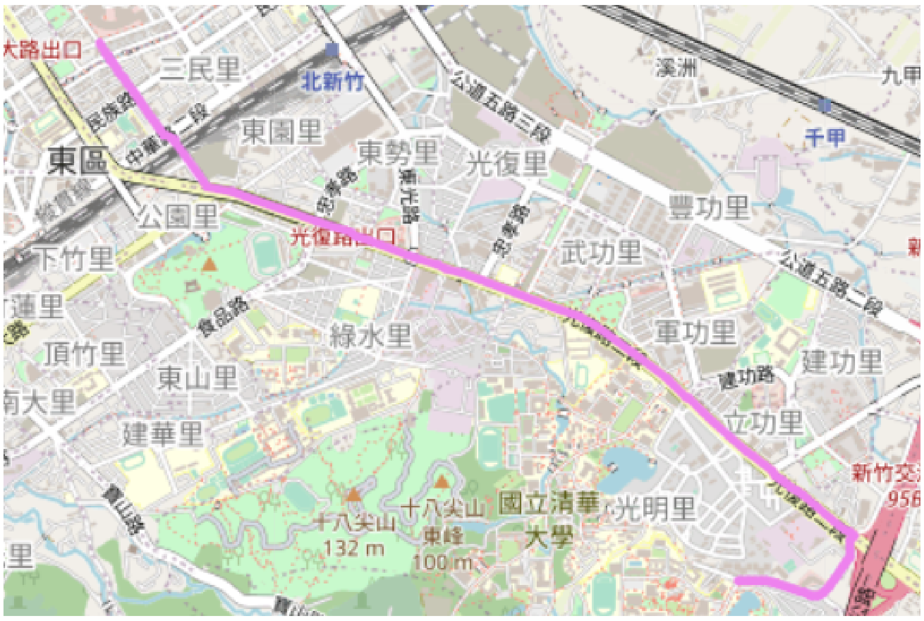
DFS

```
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: 4712
```



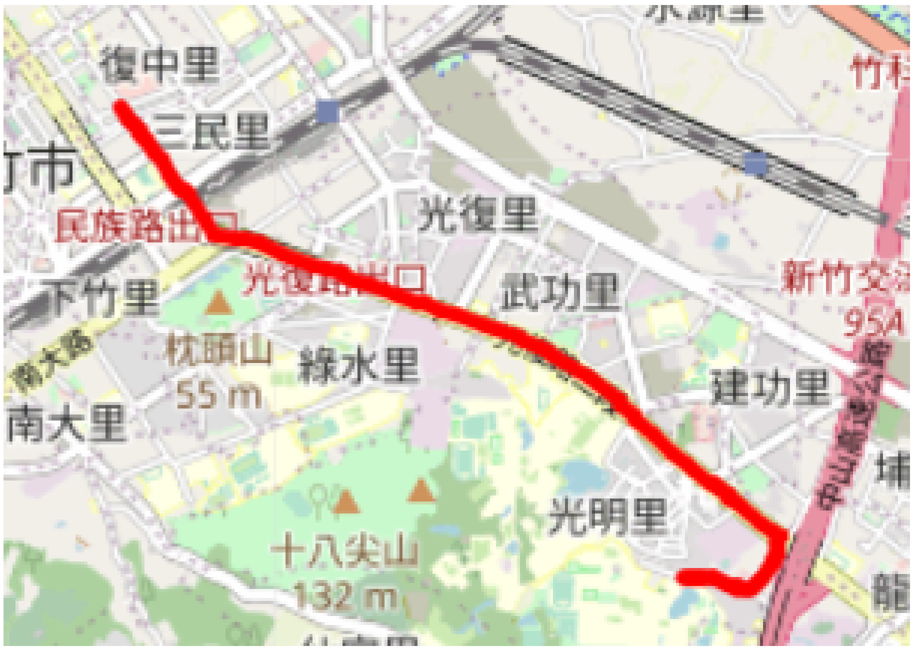
UCS

```
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: 5084
```



A* Search

```
The number of nodes in the path found by A* search: 88
Total distance of path found by A* search: 4367.881 m
The number of visited nodes in A* search: 5085
```

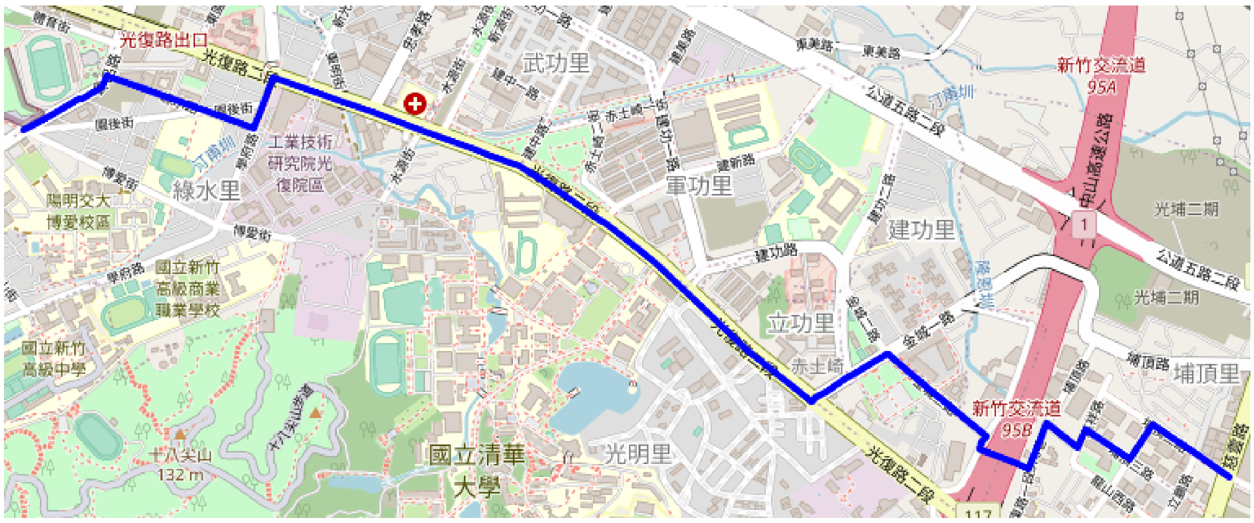


110550175鄭栩安

Test2 : from Hsinchu Zoo (ID: 426882161) to COSTCO Hsinchu Store (ID: 1737223506)

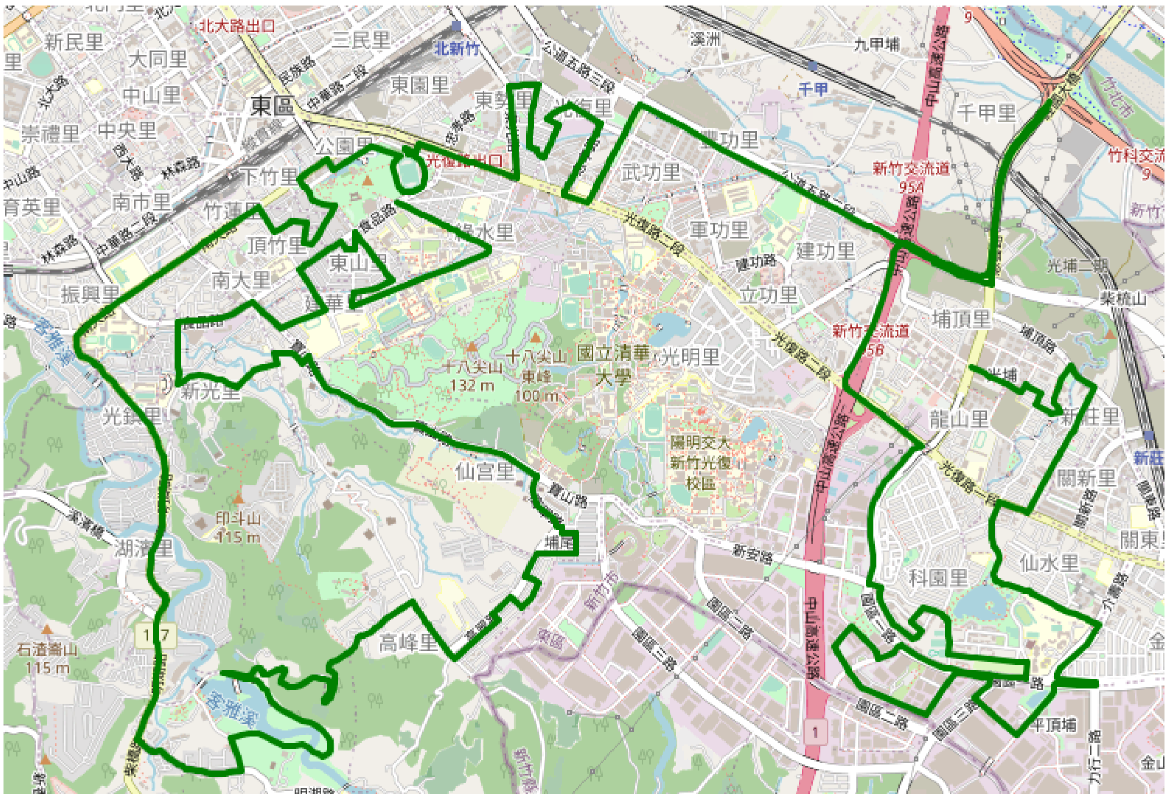
BFS

```
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

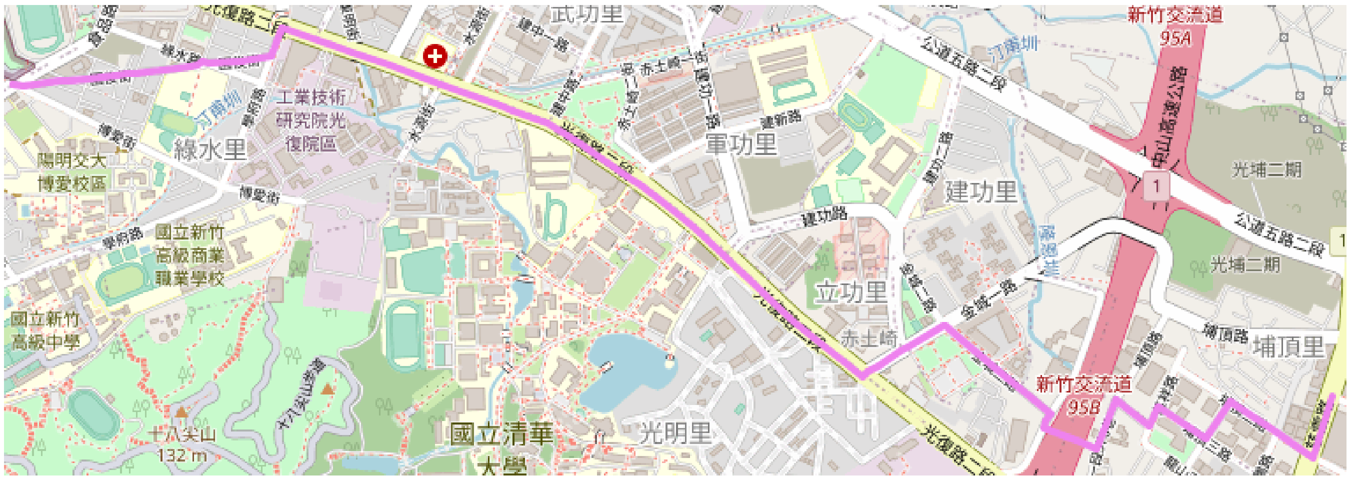
```
The number of nodes in the path found by DFS: 930
Total distance of path found by DFS: 38752.307999999895 m
The number of visited nodes in DFS: 9366
```



110550175鄭栩安

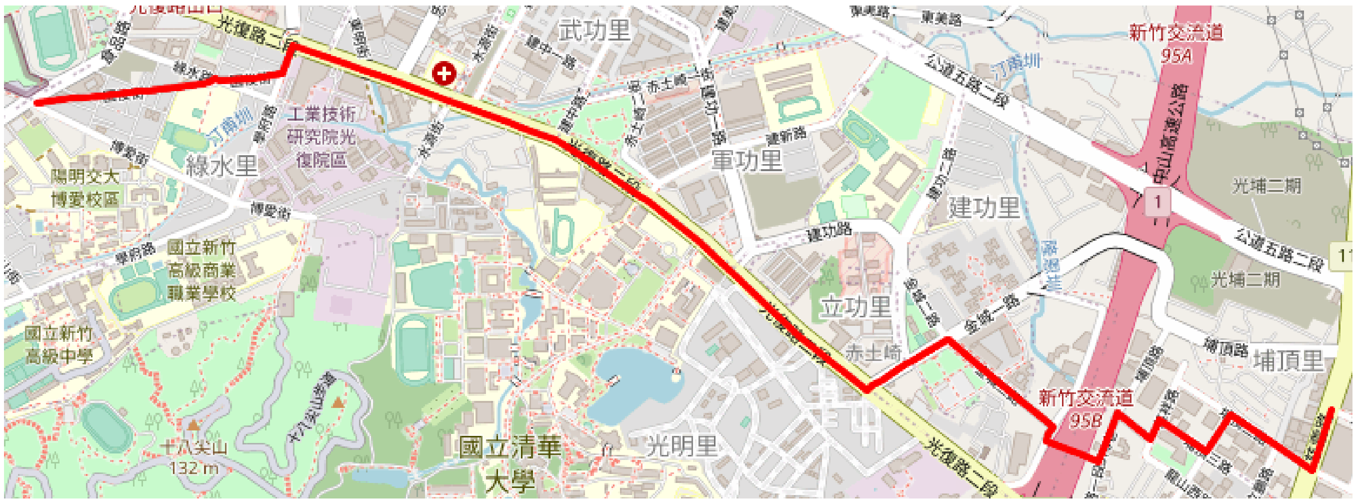
UCS

```
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: 7317
```



A* Search

```
The number of nodes in the path found by A* search: 62
Total distance of path found by A* search: 4101.84 m
The number of visited nodes in A* search: 7318
```

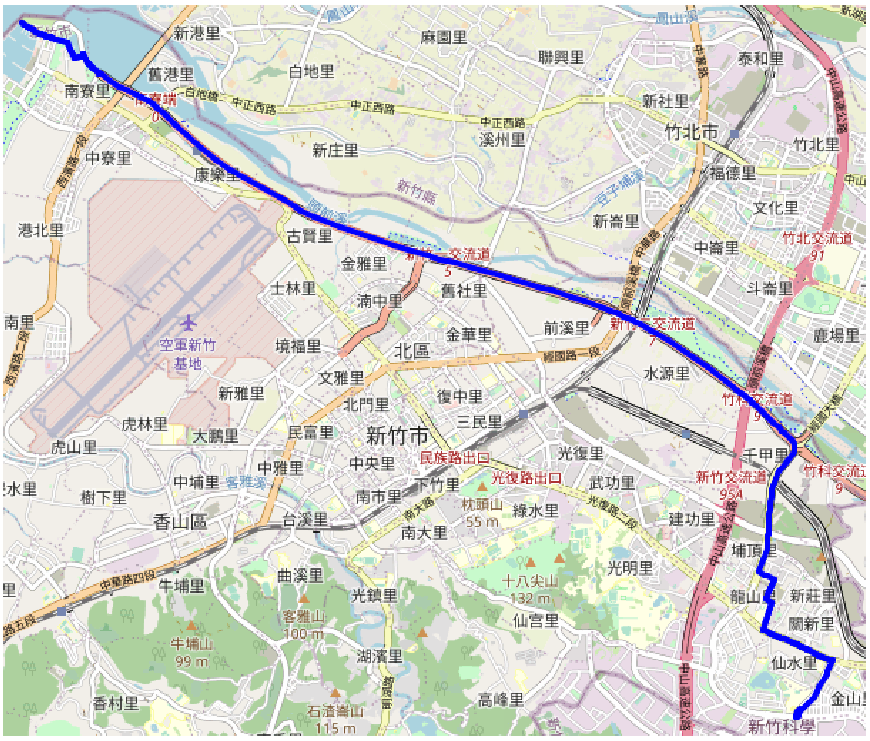


110550175鄭栩安

Test3 : from National Experimental High School At Hsinchu Science Park (ID: 1718165260) to Nanliao Fighing Port (ID: 8513026827)

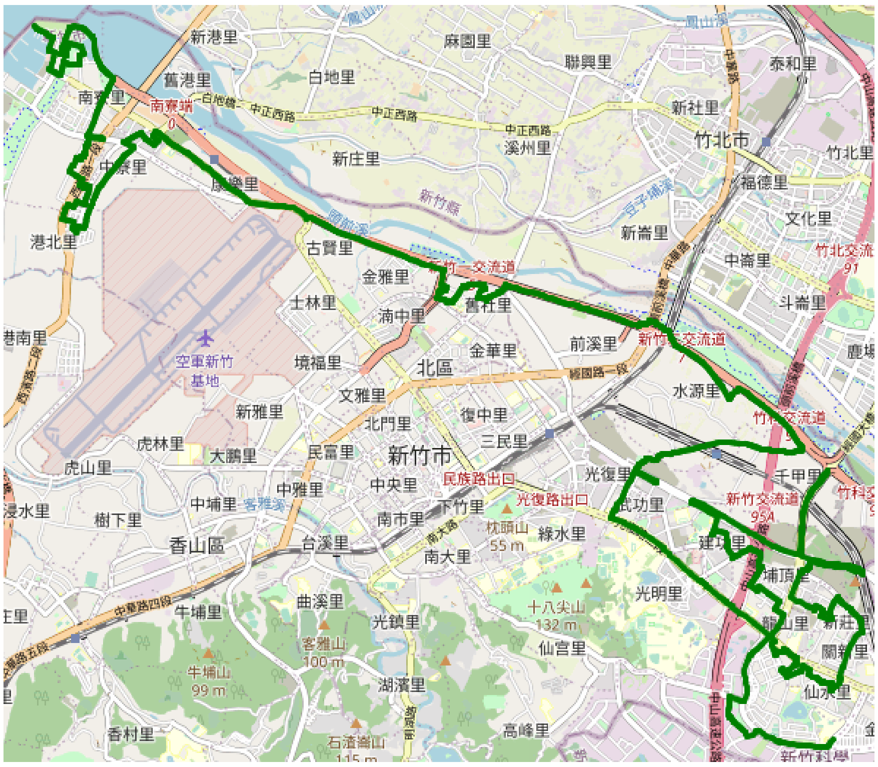
BFS

```
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

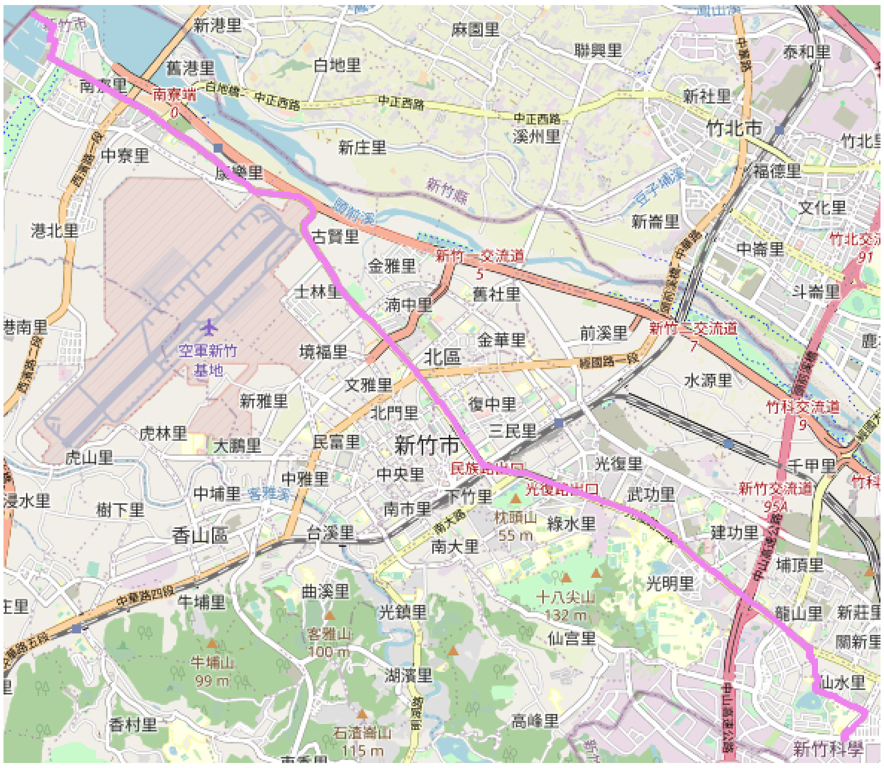
```
The number of nodes in the path found by DFS: 900
Total distance of path found by DFS: 39219.993000000024 m
The number of visited nodes in DFS: 2248
```



110550175鄭栩安

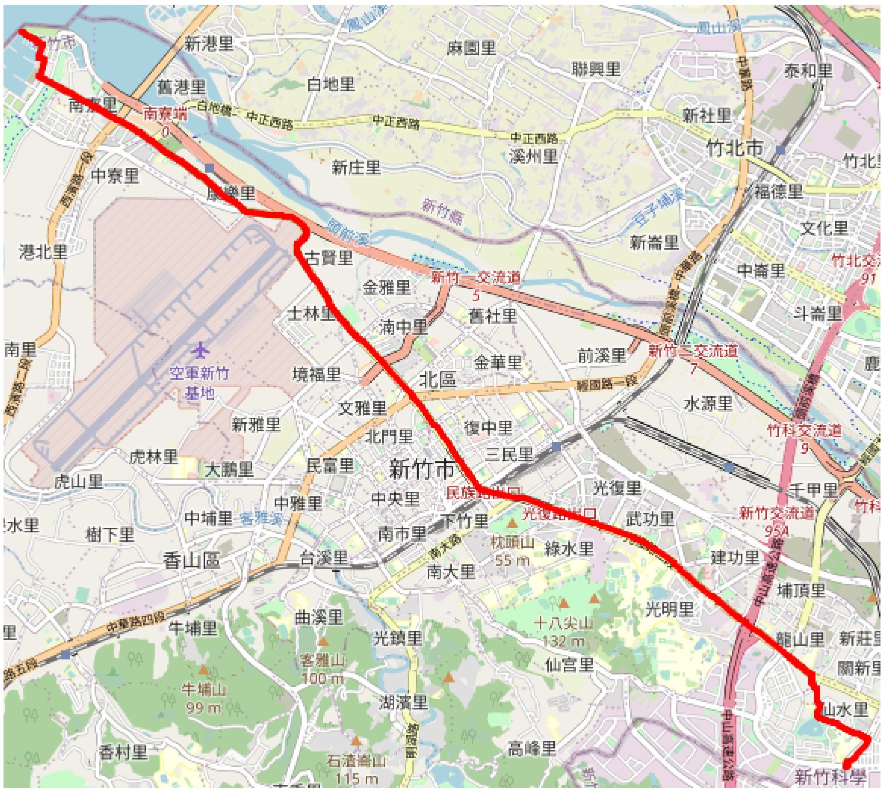
UCS

```
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* Search

```
The number of nodes in the path found by A* search: 287
Total distance of path found by A* search: 14212.412999999997 m
The number of visited nodes in A* search: 11932
```



Part II. Analysis :

1.In each test case,BFS takes the least number of nodes in the path.

2.The distance,path,and # of nodes in the path found by UCS and A* are the same.

3.The result of DFS is even worse when the roads on the map are complicated(Test1 v.s.Test3)

4.Compared with other 3 searches,DFS passes through the most # of nodes,and therefore takes the longest distance.

Part III. Question Answering (12%):

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

   It’s the first time I implement algorithm problems using python,so the biggest problem I encountered is that I don't know how to store adjacent list in python.I solve it by using ‘collections’ library.It provides the ‘defaultdict’ function,which provides a default value for the key that does not exists.

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

   Besides speed limit and distance,I think the scale of traffic flow is also important.

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

   mapping : using satellite image is a good way for mapping because it can capture the whole scene of a specific area. If the surrounding area is complicated and can’t be mapped precisely,using user-generated maps is a good alternative way.

   localization : using GPS is a common and accurate way for localization.

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.

110550175鄭栩安

ETA = Meal Prepare Time + Delivery Priority + Multiple Orders + Traffic Delay

(1)Meal Prepare Time : the preparation time of the ordered meal.It varies according to the type and amount of the order.
(2)Delivery Priority : Delivery Priority is based on the distance of delivery. Orders that are closer to the user and have a shorter distance should have a higher priority.
(3)Multiple Orders : The delivery driver might not have only one order in a delivery process,because the orders in a nearby area might be assigned to the same driver.
(4)Traffic Delay : This is the delay caused by congestion on the road.