Part I. Implementation

a. BFS

```python
def findDist(path, distance):
    """
    3. to calculate the dist
    """
    dis = 0
    for i in range(1, len(path)):
        dis += distance[(path[i-1], path[i])]
    return dis
```

```python
def bfs(start, end):

    """
    1. load the file
        *graph: store start node and end node(adjacent node, which means that may not only one node)
        *distance: store distance two node's distance
    """
    graph = {}
    distance = {}

    with open(edgeFile) as row:
        for r in row:
            info = r.strip().split(",")
            if info[0] != "start":
                start_node, end_node, dista, lim_speed = info
                start_node = int(start_node)
                end_node = int(end_node)
                dista = float(dista)
                if start_node not in graph:
                    graph[start_node] = []
                graph[start_node].append(end_node)
                distance[(start_node, end_node)] = dista
```

```
57      ## deploy BFS
58      while q:
59          curr = q.popleft()
60
61          if curr == end:
62
63              """
64              line 66-68: get path
65              """
66              while curr is not None:
67                  path.append(curr)
68                  curr = visited_node.get(curr)
69
70              path.reverse()
71
72              dist = findDist(path, distance)
73
74
75          for new in graph.get(curr, []):
76              if new not in visit:
77                  visit.append(new)
78                  visited_node[new] = curr #for getting path
79                  num_visited += 1
80                  q.append(new)
81
```

b. DFS_stack

```python
def findDist(path, distance):
    """
    3. to calculate the dist
    """
    dis = 0
    for i in range(1, len(path)):
        dis += distance[(path[i-1], path[i])]
    return dis

def dfs(start, end):

    """
    1. load the file
        *graph: store start node and end node(adjacent node, which means that may not only one node)
        *distance: store distance two node's distance
    """
    graph = {}
    distance = {}

    with open(edgeFile) as row:
        for r in row:
            info = r.strip().split(",")
            if info[0] != "start":
                start_node, end_node, dista, lim_speed = info
                start_node = int(start_node)
                end_node = int(end_node)
                dista = float(dista)
                if start_node not in graph:
                    graph[start_node] = []
                graph[start_node].append(end_node)
                distance[(start_node, end_node)] = dista

    path = list()
    dist = float(0.0)
    num_visited = int(0)

    stack = deque([start])

    """
    visited_node: curr's last node
    visit: to avoid iterate a same node twice
    """
    visited_node = {}
    visit = set()
    visit.add(start)
```

almost same like BFS

```
52      ## deploy DFS
53      while stack:
54          curr = stack.pop()
55
56          if curr == end:
57
58              """
59              line 61-63: get path
60              """
61              while curr is not None:
62                  path.append(curr)
63                  curr = visited_node.get(curr)
64
65              path.reverse()
66
67              dist = findDist(path, distance)
68
69
70          for new in graph.get(curr, []):
71              if new not in visit:
72                  visit.add(new)
73                  visited_node[new] = curr
74                  num_visited += 1
75                  stack.append(new)
76
77      return path, dist, num_visited
```

just replace deque to stack to implement the dfs_stack

c. UCS

```python
def findDist(path, distance):
    """
    3. to calculate the dist
    """
    dis = 0
    for i in range(1, len(path)):
        dis += distance[(path[i-1], path[i])]
    return dis

def ucs(start, end):

    """
    1. load the file
        *graph: store start node and end node(adjacent node, which means that may not only one node)
        *distance: store distance two node's distance
    """
    graph = {}
    distance = {}

    with open(edgeFile) as row:
        for r in row:
            info = r.strip().split(",")
            if info[0] != "start":
                start_node, end_node, dista, lim_speed = info
                start_node = int(start_node)
                end_node = int(end_node)
                dista = float(dista)
                if start_node not in graph:
                    graph[start_node] = []
                graph[start_node].append((end_node, dista))  # (neighbor, distance)
                distance[(start_node, end_node)] = dista
```

```python
    #path = list()
    dist = float(0.0)
    num_visited = int(0)

    pq = PriorityQueue()
    pq.put((0, [start]))  # Initialize the priority queue with a cost of 0 for the initial node.

    """
    visited_node: curr's last node
    visit: to avoid iterate a same node twice
    """
    #visited_node = {}
    visit = set()
    visit.add(start)
```

```python
    ## deploy UCS
    while not p
        curr_co
        curr = path[-1] # Get the last node
        visit.add(curr)
        if curr == end:
            dist = findDist(path, distance)
            return path, dist, num_visited

        for neighbor, nei_cost in graph.get(curr, []):
            if neighbor not in visit:
                new_path = list(path)
                new_path.append(neighbor)
                num_visited += 1
                pq.put((curr_cost + nei_cost, new_path))  # Consider the total cost when placing it into the queue.

    return path, dist, num_visited
```

(variable) path: Any

d. A*

```
def findDist(path, distance):
    """
    Function to calculate the total distance of a given path.
    """
    total_distance = 0
    for i in range(1, len(path)):
        total_distance += distance[(path[i-1], path[i])]
    return total_distance

def astar(start, end):


    # Initialize graph and distance dictionaries
    graph = {}
    distance = {}

    # Load edge info
    with open(edgeFile) as row:
        for r in row:
            info = r.strip().split(",")
            if info[0] != "start":
                start_node, end_node, dista, lim_speed = info
                start_node = int(start_node)
                end_node = int(end_node)
                dista = float(dista)
                if start_node not in graph:
                    graph[start_node] = {}
                graph[start_node][end_node] = dista
                distance[(start_node, end_node)] = dista
```

```
    # Load heuristic info
    heru_ = {}
    idx = 0
    with open(heuristicFile) as row:
        for r in row:
            info = r.strip().split(",")
            if info[0] == 'node':
                for i in range(1, 4):
                    if info[i] == str(end):
                        idx = i
                        break
                continue
            heru_[info[0]] = float(info[idx])
```

```
    """
    Initialize priority queue, visited set, and parent dictionary
    """
    pq = PriorityQueue()
    pq.put((0, start))  # Put the start node in the priority queue with priority 0
    visit = set()  # Initialize set to store visited nodes
    parent = {}  # Initialize dictionary to store parent-child relationships
    g_val = {start: 0}  # Initialize dictionary to store g values (distance from start node)
```

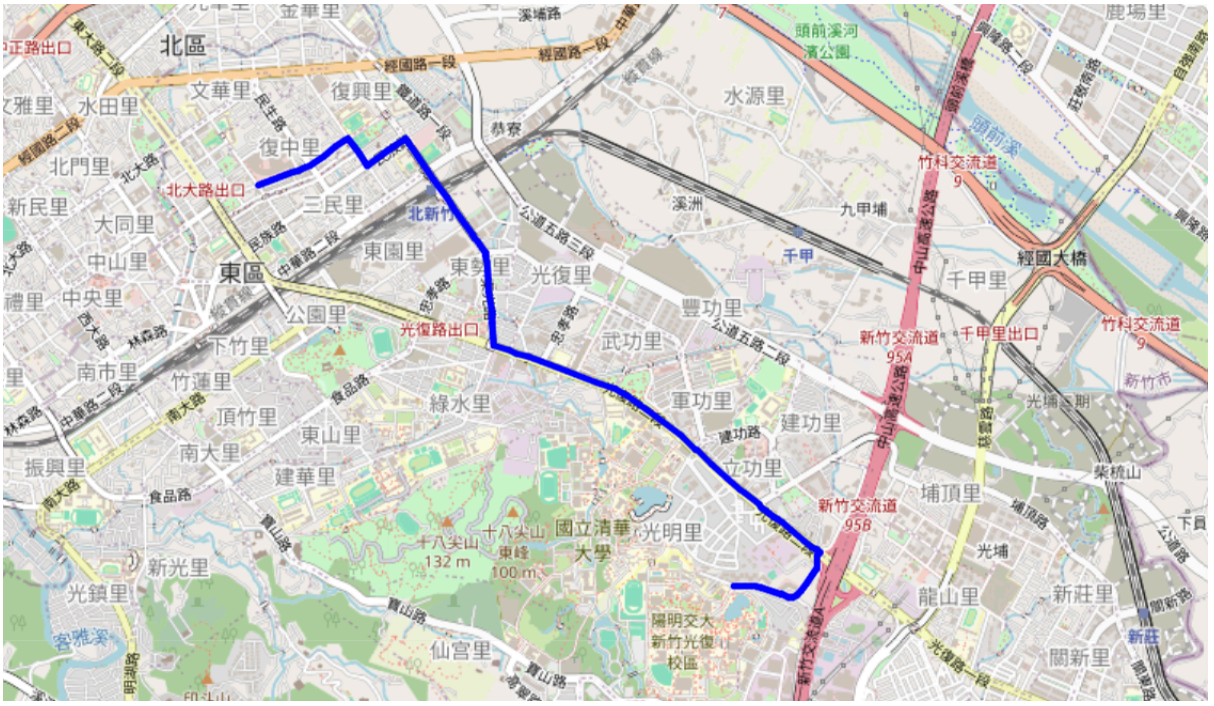
```
    while not pq.empty():
        dontUse, curr = pq.get()

        if curr == end:
            path = [curr]
            while curr != start:
                curr = parent[curr]
                path.append(curr)
            path.reverse()
            total_distance = findDist(path, distance)
            return path, total_distance, len(visit)

        visit.add(curr) # Add current node to visited set
        for neighbor, dist in graph.get(curr, {}).items():
            if neighbor in visit:
                continue
            tg = g_val[curr] + dist # Calculate tentative g value (distance from start to neighbor)
            if neighbor not in g_val or tg < g_val[neighbor]: # Check if new path to neighbor is better
                g_val[neighbor] = tg # Update g value for neighbor
                f_val = tg + heru_.get(str(neighbor), float('inf'))
                pq.put((f_val, neighbor))
                parent[neighbor] = curr # Update parent of neighbor

    return path, total_distance, len(visit)
```

Part II. Results & Analysis

Test 1:
from National Yang Ming Chiao Tung University (ID: 2270143902) to
Big City Shopping Mall (ID: 1079387396)

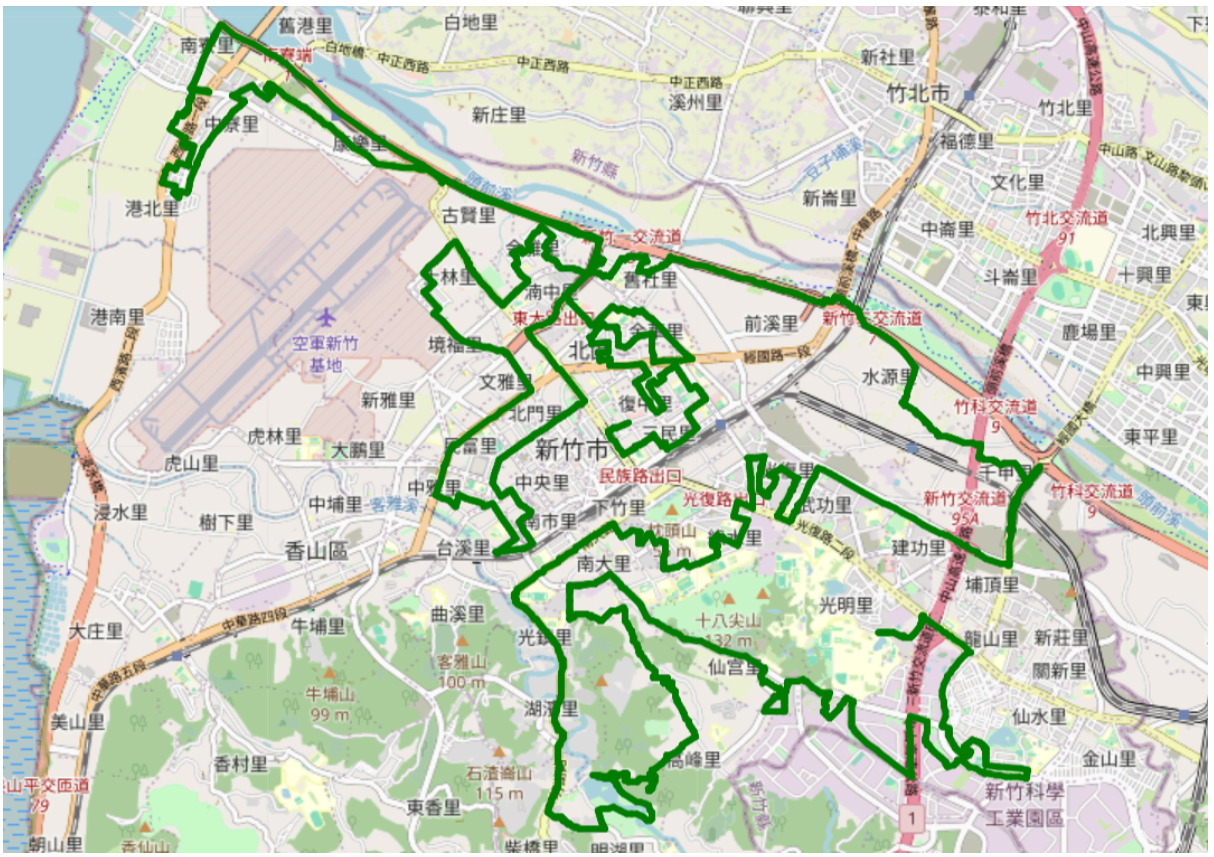
BFS

```
The number of nodes in the path found by BFS: 88
Total distance of path found by BFS: 4978.8820000000005 m
The number of visited nodes in BFS: 12053
```



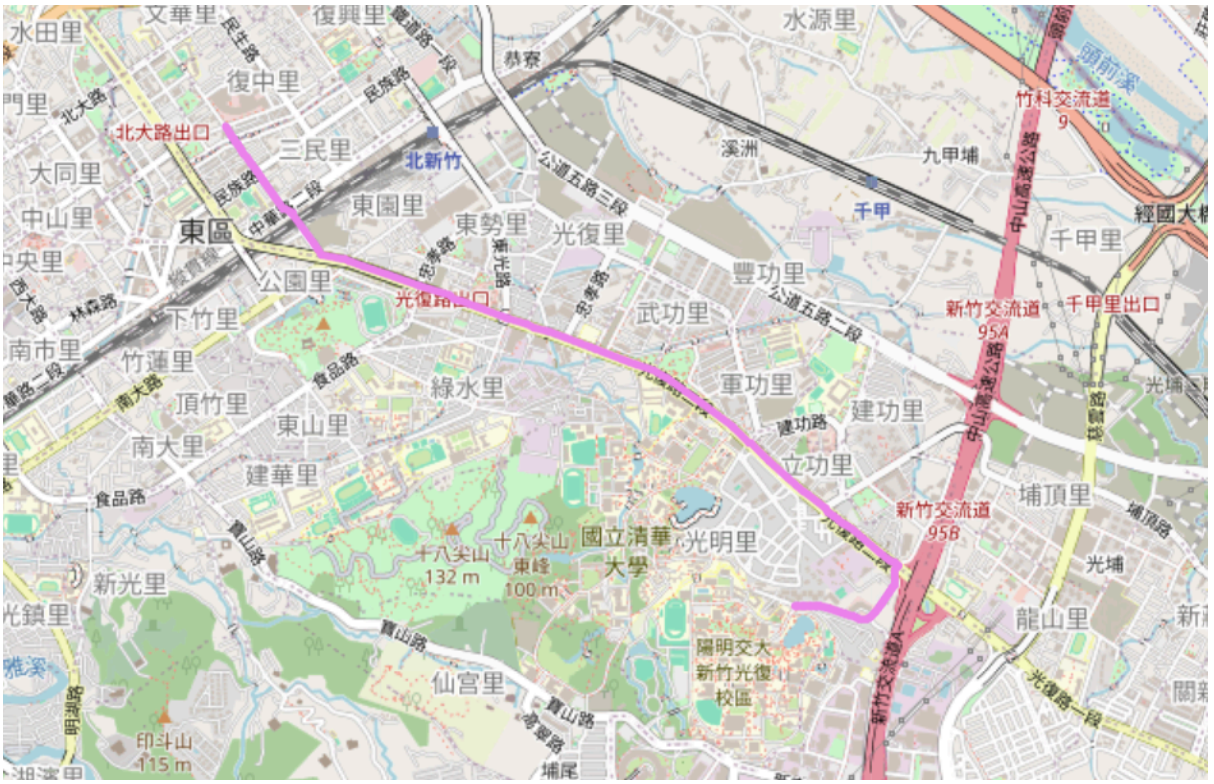
DFS(stack)

```
The number of nodes in the path found by DFS: 1718
Total distance of path found by DFS: 75504.31499999983 m
The number of visited nodes in DFS: 12053
```



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: 11036



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: 260
```

A*(time)

```
The number of nodes in the path found by A* search: 93
Total second of path found by A* search: 94.70316160063828 s
The number of visited nodes in A* search: 141
```

三民里
北新竹
東園里
東勢里
光復里
東區
民族路出口
食品路出口
光復路出口
公園里
下竹里
南大路
枕頭山
55 m
工業技術
研究院光
復院區
綠水里
陽明交大
博愛校區
頂竹里
東山里
國立新竹
高級商業
職業學校
國立新竹
高級中學
建華里
十八尖山
132 m
國立清華
大學
光明里
光復中學
武功里
軍功里
立功里
赤土崎
建功里
豐功里
溪洲
千甲
公道五路三段
公道五路二段
中華路二段
寶山路
大學路

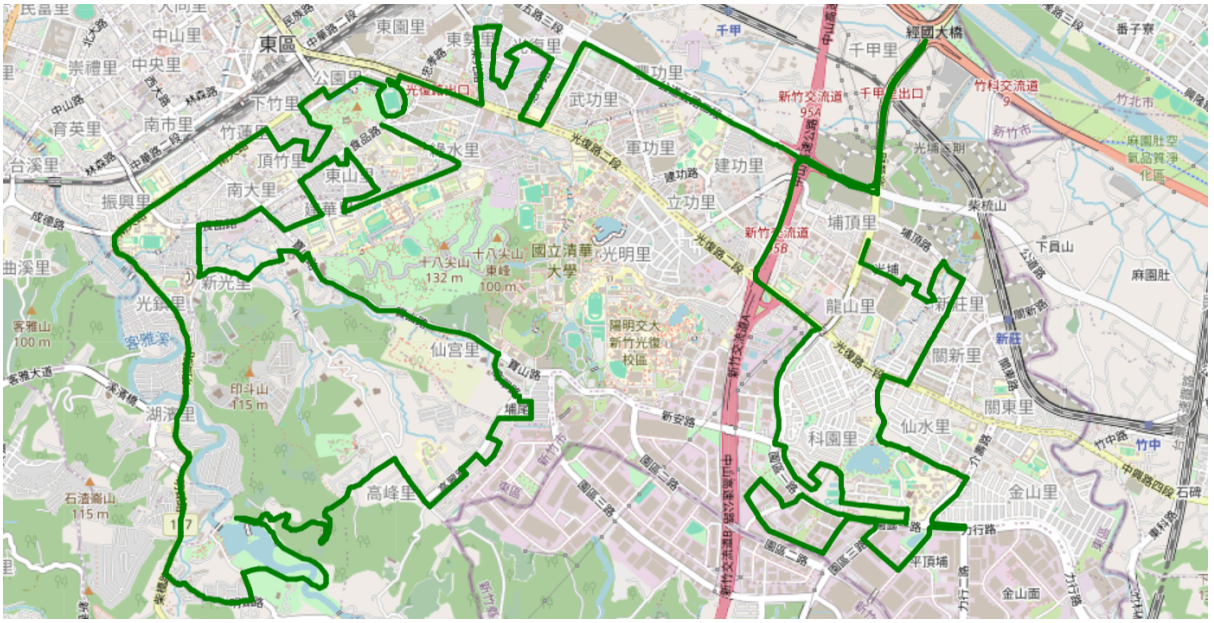
Test 2:
from Hsinchu Zoo (ID: 426882161)
to COSTCO Hsinchu Store (ID: 1737223506)

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: 12049
```

DFS(stack)

```
The number of nodes in the path found by DFS: 930
Total distance of path found by DFS: 38752.30799999996 m
The number of visited nodes in DFS: 12049
```



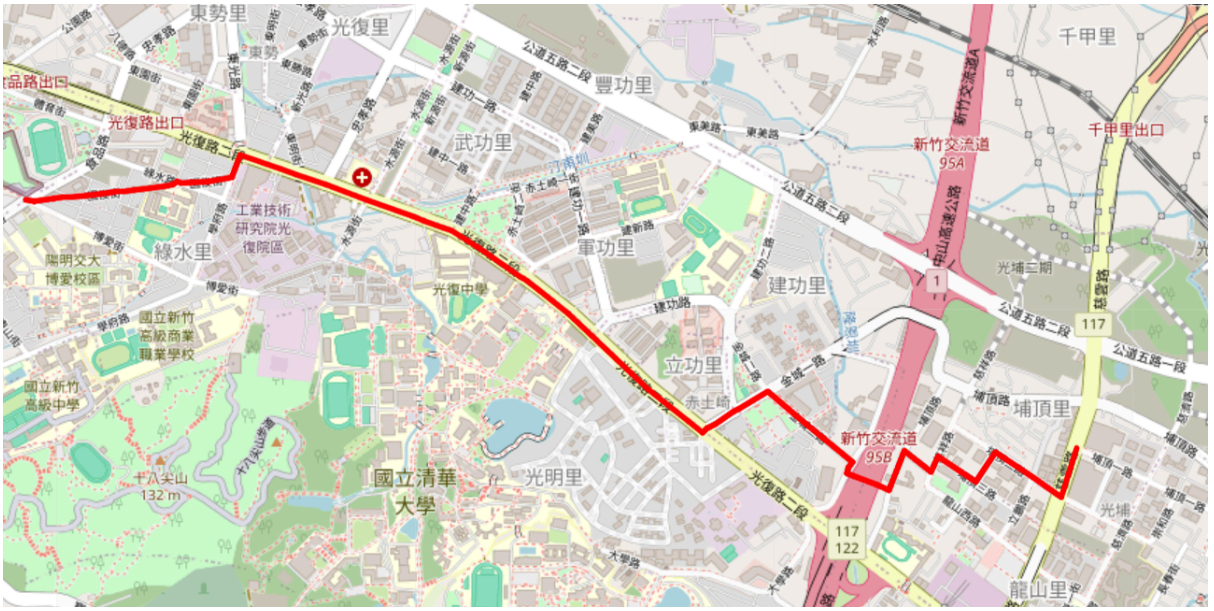
UCS

```
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: 18182

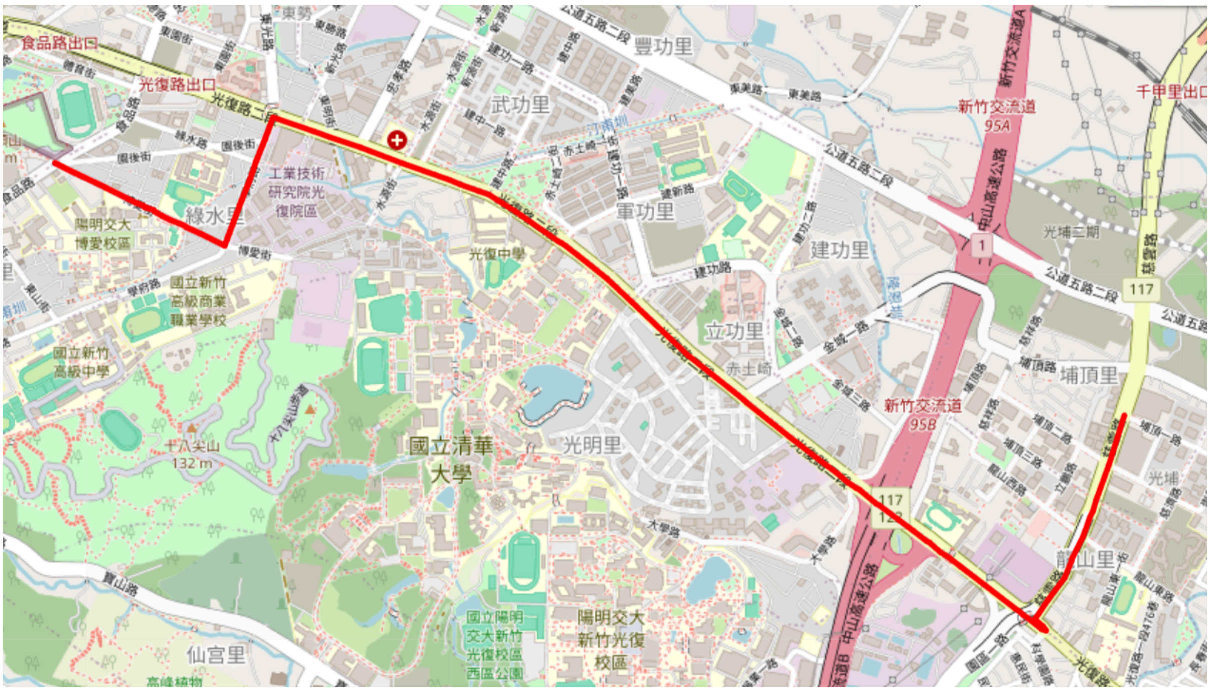
A*

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: 1171



A*(time):

The number of nodes in the path found by A* search: 76
Total second of path found by A* search: 92.93327896551723 s
The number of visited nodes in A* search: 83



食品路出口
光復路出口
光復路二段
豐功里
武功里
軍功里
建功里
立功里
綠水里
工業技術
研究院光
復院區
陽明交大
博愛校區
國立新竹
高級商業
職業學校
國立新竹
高級中學
光復中學
十八尖山
132 m
國立清華
大學
光明里
國立陽明
交大新竹
光復校區
西區公園
陽明交大
新竹光復
校區
仙宮里
新竹交流道
95A
95B
中山高速公路
公道五路二段
光埔三期
埔頂里
光埔
龍山里
千甲里出口
117
赤土崎
寶山路
大學路

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: 12051
```

DFS(stack)

```
The number of nodes in the path found by DFS: 900
Total distance of path found by DFS: 39219.99299999996 m
The number of visited nodes in DFS: 12051
```

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: 48620

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: 7072
```

A*(time)

```
The number of nodes in the path found by A* search: 343
Total second of path found by A* search: 308.0797483899724 s
The number of visited nodes in A* search: 354
```


新港里
舊港里
白地里
麻園里
聯興里
泰和里
南寮里
南寮端
1
白地橋
中正西路
中正西路
中正西路
溪州里
新社里
竹北市
中寮里
新庄里
康樂里
新竹縣
頭前溪
福德里
文化里
港北里
古賢里
新崙里
中崙里
金雅里
新竹一交流道
5
上林里
舊社里
斗崙里
湳中里
東大路出口
前溪里
新竹二交流道
7
空軍新竹基地
境福里
金華里
北區
水源里
文雅里
新雅里
復中里
北門里
竹科交流道
9
虎林里
三民里
大鵬里
民富里
新竹市
千甲里
虎山里
民族路出口
光復里
中雅里
中央里
光復路出口
武功里
新竹交流道
95A
中埔里
客雅溪
下竹里
樹下里
南市里
枕頭山
55 m
綠水里
建功里
香山區
台溪里
南大路
光復路二段
埔頂里
曲溪里
十八尖山
132 m
光明里
牛埔里
光鎮里
龍山里
客雅山
100 m
仙宮里
牛埔山
99 m
湖濱里
仙水里
香村里
高峰里
石滴嶺山

Part III. Question Answering

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

A:

BFS:

some coding gammers, like how to correctly initialize the deque and set the element.-> try and google.

UCS:

```
56          curr = path[-1] # Get the last node
```

Also the typing error.

```
    curr = path[-1]
           ~~~~^^^^
TypeError: 'int' object is not subscriptable
```

-> And then I change some element's type.But unfortunately, I have another problem.

```
    pq.put(curr_cost + distance[(curr, new)], new_path)
                       ~~~~~~~~^^^^^^^^^^^^^
KeyError: (426882161, (6497517989, 18.726))
```

-> So I change the iteration(there is only one iterator in the original, and then up to two iterators.

A*:

I think the most hard part is to understand the algorithm. I spent maybe at least 4-5 hours to realize how to work and how to translate the ideas to the code.

-> google search and try.

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:

another attribute:

Traffic congestion. Like google maps, we can choose the best road, at the same time, we also can predict the arrival time when we schedule our itinerary.

rationale:

Incorporating real-time traffic information into route planning can help optimize the chosen path by avoiding congested areas or selecting alternative routes with less traffic. This attribute is crucial because it directly impacts the travel time and overall efficiency of the navigation system.

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:

mapping:

Using satellite imagery to create detailed maps with accurate geographical information.

localization:

Implementing GPS to determine the device's current location accurately.

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.

A:

dynamic heuristic equation for ETA:

ETA = Base_Time + Meal_Prep_Time + Delivery_Priority_Adjustment + Multiple_Orders_Adjustment

Base_Time:

Estimated travel time under normal conditions.

Meal_Prep_Time:

Time for meal preparation at the restaurant, varying based on order complexity or workload.

Delivery_Priority_Adjustment:

ETA adjustment based on delivery priority; higher priority orders get shorter delivery times.

Multiple_Orders_Adjustment:

Accounts for handling multiple orders concurrently, optimizing routes to reduce overall delivery time.

rationale:

Offer users precise and current estimates that consider diverse factors influencing delivery durations.

By integrating several elements, such as meal prep time, delivery priority, and efficiency from handling multiple orders, to enhance ETA accuracy and satisfy our customers.