# 110550126 曾家祐

# Homework 2: Route Finding

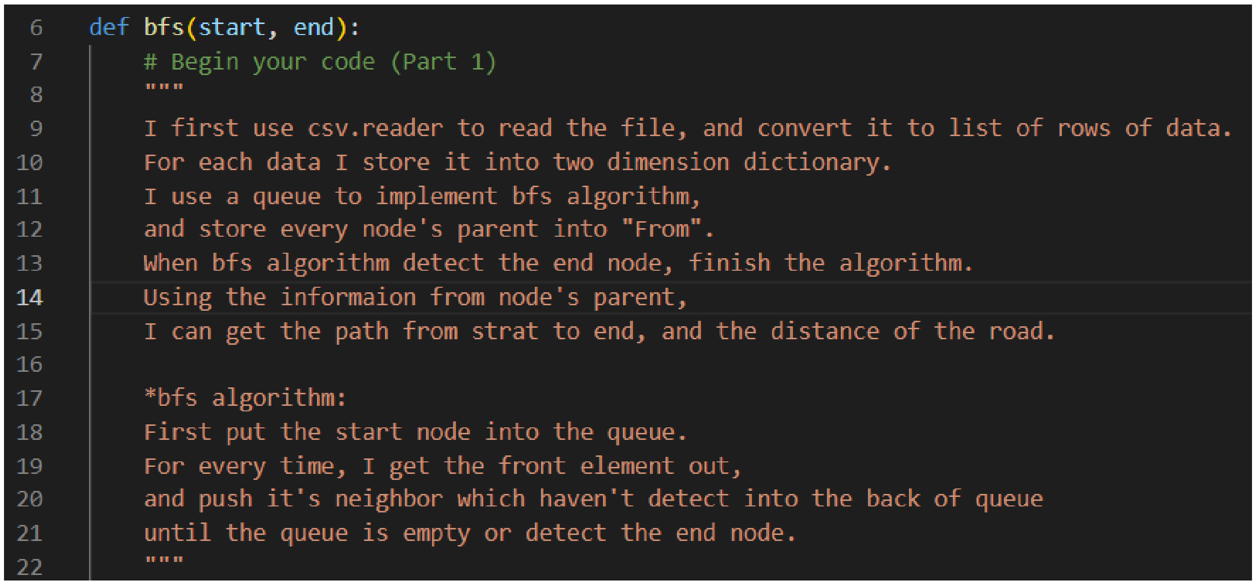
## Part I. Implementation :

- **Part 1: Breadth-first Search：bfs.py -> bfs()**

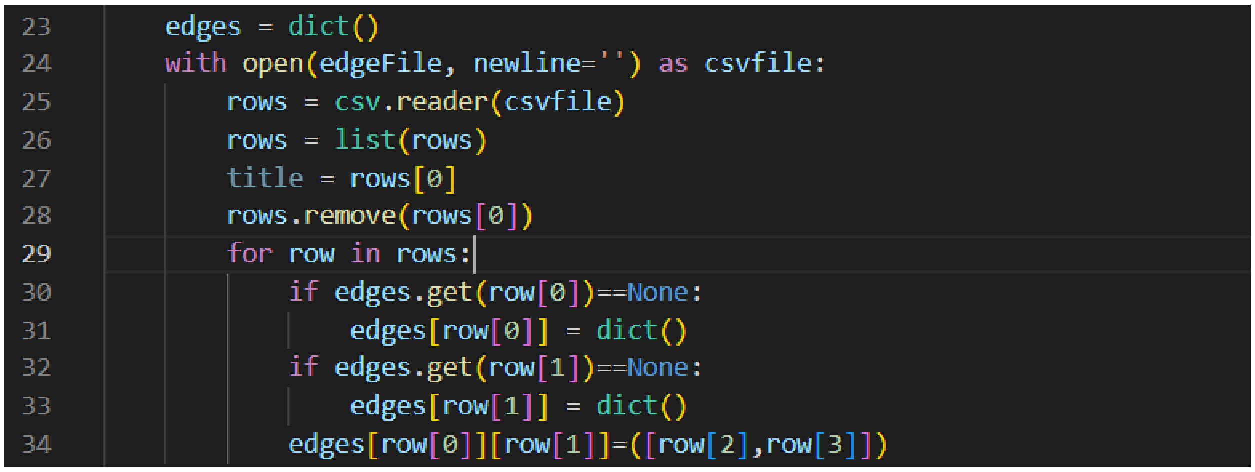
  **1. Read data: I use csv.reader to read file, and make each data (start, end, distance, speed limit)into dictionary (edges[start][end] = (distance, speed limit))**

  **2. bfs algorithm: First put the start node into the queue.**
  **For every time, I get the front element out, and push its neighbor which haven't detect into the back of queue.**
  **until the queue is empty or detect the end node.**

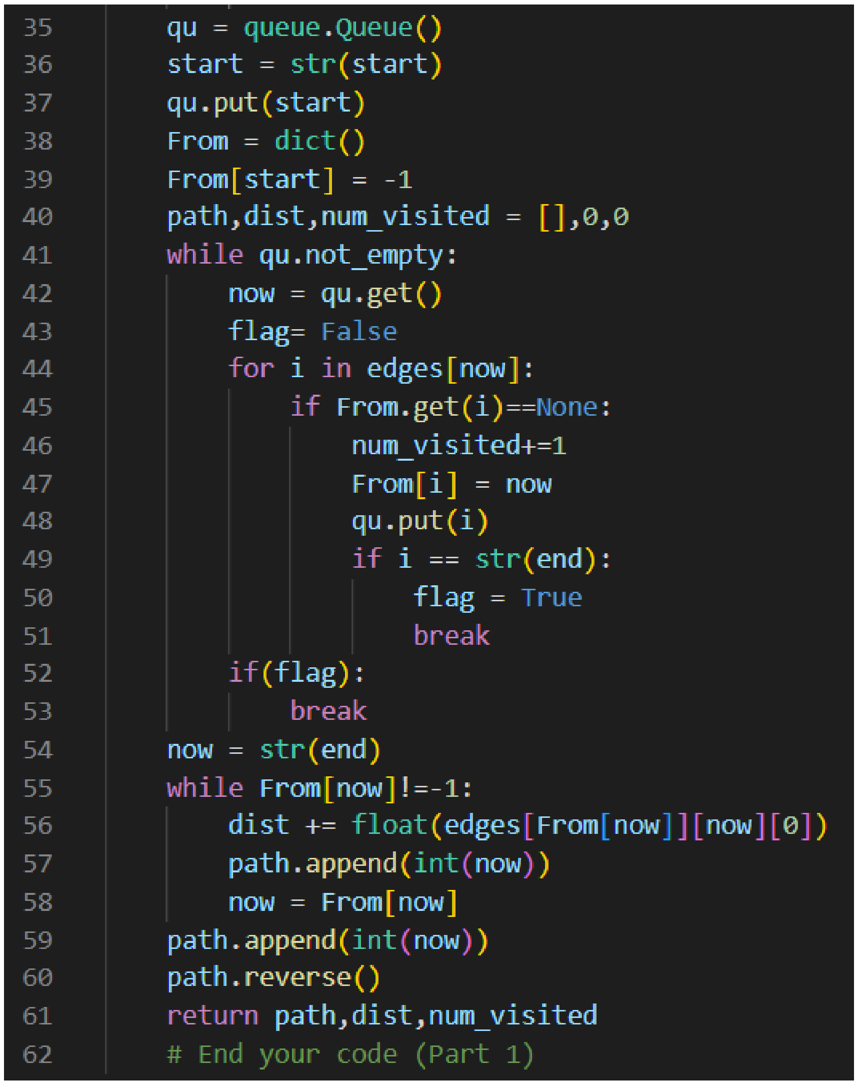
  **3. get result in the bfs: I store every node's parent, from the the information, I can get the path, and calculate the distance**



```python
def bfs(start, end):
    # Begin your code (Part 1)
    """
    I first use csv.reader to read the file, and convert it to list of rows of data.
    For each data I store it into two dimension dictionary.
    I use a queue to implement bfs algorithm,
    and store every node's parent into "From".
    When bfs algorithm detect the end node, finish the algorithm.
    Using the informaion from node's parent,
    I can get the path from strat to end, and the distance of the road.

    *bfs algorithm:
    First put the start node into the queue.
    For every time, I get the front element out,
    and push it's neighbor which haven't detect into the back of queue
    until the queue is empty or detect the end node.
    """
```



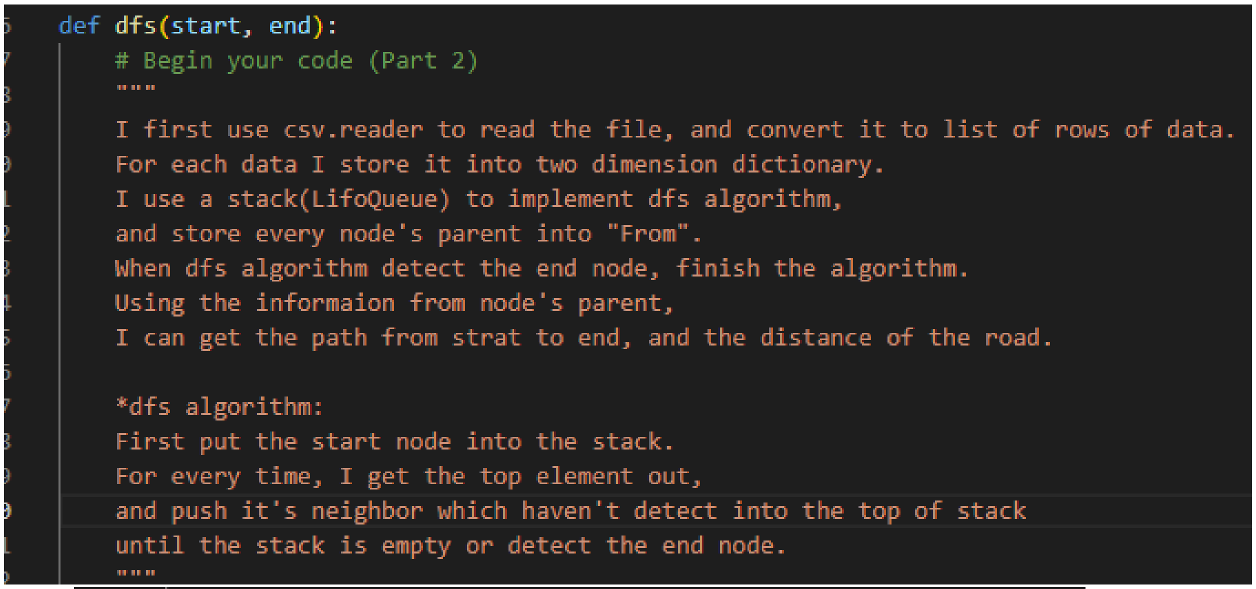
```python
    edges = dict()
    with open(edgeFile, newline='') as csvfile:
        rows = csv.reader(csvfile)
        rows = list(rows)
        title = rows[0]
        rows.remove(rows[0])
        for row in rows:
            if edges.get(row[0])==None:
                edges[row[0]] = dict()
            if edges.get(row[1])==None:
                edges[row[1]] = dict()
            edges[row[0]][row[1]]=([row[2],row[3]])
```



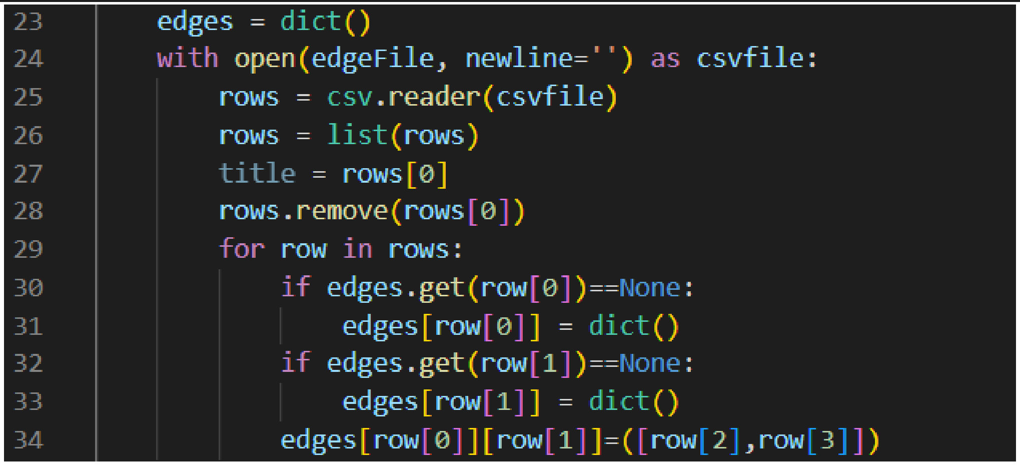
```python
    qu = queue.Queue()
    start = str(start)
    qu.put(start)
    From = dict()
    From[start] = -1
    path,dist,num_visited = [],0,0
    while qu.not_empty:
        now = qu.get()
        flag= False
        for i in edges[now]:
            if From.get(i)==None:
                num_visited+=1
                From[i] = now
                qu.put(i)
                if i == str(end):
                    flag = True
                    break
        if(flag):
            break
    now = str(end)
    while From[now]!=-1:
        dist += float(edges[From[now]][now][0])
        path.append(int(now))
        now = From[now]
    path.append(int(now))
    path.reverse()
    return path,dist,num_visited
    # End your code (Part 1)
```

- **Part 2: Depth-first Search: dfs_stack.py -> dfs()**

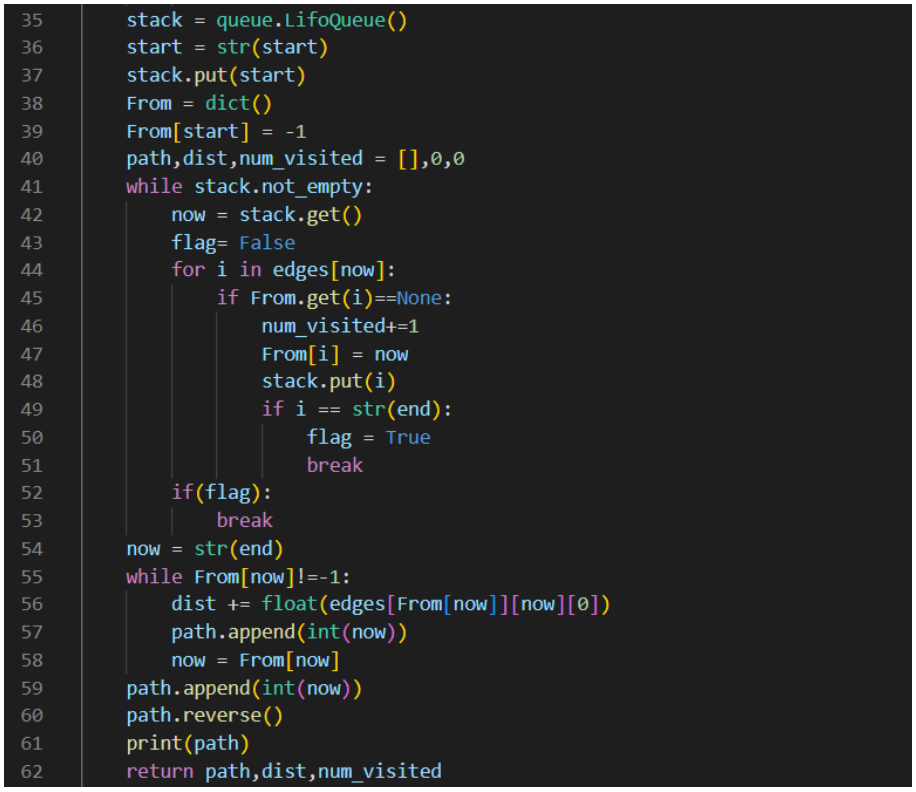
**1. Read data: I use csv.reader to read file, and make each data (start, end, distance, speed limit)into dictionary (edges[start][end] = (distance, speed limit))**
**2. dfs algorithm: First put the start node into the stack.**
**For every time, I get the top element out, and push its neighbor which haven't detect into the top of stack.**
**until the stack is empty or detect the end node.**
**3. get result in the dfs: I store every node's parent, from the information, I can get the path, and calculate the distance.**



```python
def dfs(start, end):
    # Begin your code (Part 2)
    """
    I first use csv.reader to read the file, and convert it to list of rows of data.
    For each data I store it into two dimension dictionary.
    I use a stack(LifoQueue) to implement dfs algorithm,
    and store every node's parent into "From".
    When dfs algorithm detect the end node, finish the algorithm.
    Using the informaion from node's parent,
    I can get the path from strat to end, and the distance of the road.

    *dfs algorithm:
    First put the start node into the stack.
    For every time, I get the top element out,
    and push it's neighbor which haven't detect into the top of stack
    until the stack is empty or detect the end node.
    """
```

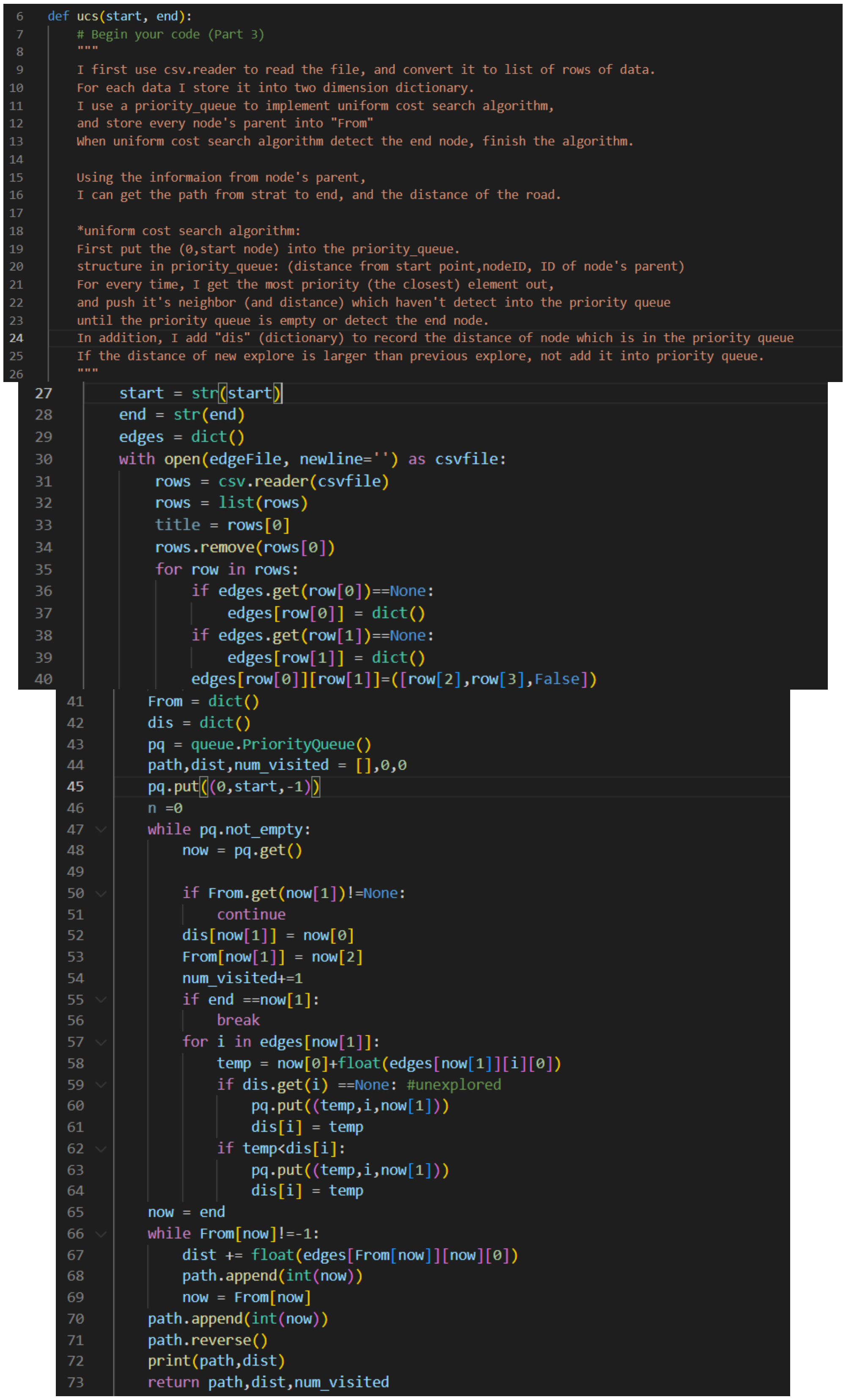


```python
    edges = dict()
    with open(edgeFile, newline='') as csvfile:
        rows = csv.reader(csvfile)
        rows = list(rows)
        title = rows[0]
        rows.remove(rows[0])
        for row in rows:
            if edges.get(row[0])==None:
                edges[row[0]] = dict()
            if edges.get(row[1])==None:
                edges[row[1]] = dict()
            edges[row[0]][row[1]]=([row[2],row[3]])
```



```
    stack = queue.LifoQueue()
    start = str(start)
    stack.put(start)
    From = dict()
    From[start] = -1
    path,dist,num_visited = [],0,0
    while stack.not_empty:
        now = stack.get()
        flag= False
        for i in edges[now]:
            if From.get(i)==None:
                num_visited+=1
                From[i] = now
                stack.put(i)
                if i == str(end):
                    flag = True
                    break
        if(flag):
            break
    now = str(end)
    while From[now]!=-1:
        dist += float(edges[From[now]][now][0])
        path.append(int(now))
        now = From[now]
    path.append(int(now))
    path.reverse()
    print(path)
    return path,dist,num_visited
```

- **Part 3: Uniform Cost Search**
  **1. Read data: I use csv.reader to read file, and make each data (start, end, distance, speed limit)into dictionary (edges[start][end] = (distance, speed limit)),**
  **2. dfs algorithm: First put (0,start node, -1 )(distance, nodeID, parentID)into the priority_queue.**
  **For every time, I get the closest element out, and push its neighbor which haven't detect into priority_queue.**
  **until the priority_queue is empty or detect the end node.**
  **3. get result in the ucs: I store every node's parent, from the information, I can get the path, and calculate the distance.**



```python
def ucs(start, end):
    # Begin your code (Part 3)
    """
    I first use csv.reader to read the file, and convert it to list of rows of data.
    For each data I store it into two dimension dictionary.
    I use a priority_queue to implement uniform cost search algorithm,
    and store every node's parent into "From"
    When uniform cost search algorithm detect the end node, finish the algorithm.

    Using the informaion from node's parent,
    I can get the path from strat to end, and the distance of the road.

    *uniform cost search algorithm:
    First put the (0,start node) into the priority_queue.
    structure in priority_queue: (distance from start point,nodeID, ID of node's parent)
    For every time, I get the most priority (the closest) element out,
    and push it's neighbor (and distance) which haven't detect into the priority queue
    until the priority queue is empty or detect the end node.
    In addition, I add "dis" (dictionary) to record the distance of node which is in the priority queue
    If the distance of new explore is larger than previous explore, not add it into priority queue.
    """
    start = str(start)
    end = str(end)
    edges = dict()
    with open(edgeFile, newline='') as csvfile:
        rows = csv.reader(csvfile)
        rows = list(rows)
        title = rows[0]
        rows.remove(rows[0])
        for row in rows:
            if edges.get(row[0])==None:
                edges[row[0]] = dict()
            if edges.get(row[1])==None:
                edges[row[1]] = dict()
            edges[row[0]][row[1]]=([row[2],row[3],False])
    From = dict()
    dis = dict()
    pq = queue.PriorityQueue()
    path,dist,num_visited = [],0,0
    pq.put((0,start,-1))
    n =0
    while pq.not_empty:
        now = pq.get()

        if From.get(now[1])!=None:
            continue
        dis[now[1]] = now[0]
        From[now[1]] = now[2]
        num_visited+=1
        if end ==now[1]:
            break
        for i in edges[now[1]]:
            temp = now[0]+float(edges[now[1]][i][0])
            if dis.get(i) ==None: #unexplored
                pq.put((temp,i,now[1]))
                dis[i] = temp
            if temp<dis[i]:
                pq.put((temp,i,now[1]))
                dis[i] = temp
    now = end
    while From[now]!=-1:
        dist += float(edges[From[now]][now][0])
        path.append(int(now))
        now = From[now]
    path.append(int(now))
    path.reverse()
    print(path,dist)
    return path,dist,num_visited
```

- **Part 4: A* Search**

**1. Read data: I use csv.reader to read file, and make each data (start, end, distance, speed limit)into dictionary (edges[start][end] = (distance, speed limit)) and (startID, endID, distance) into (heur[startID][endID] = distance)**

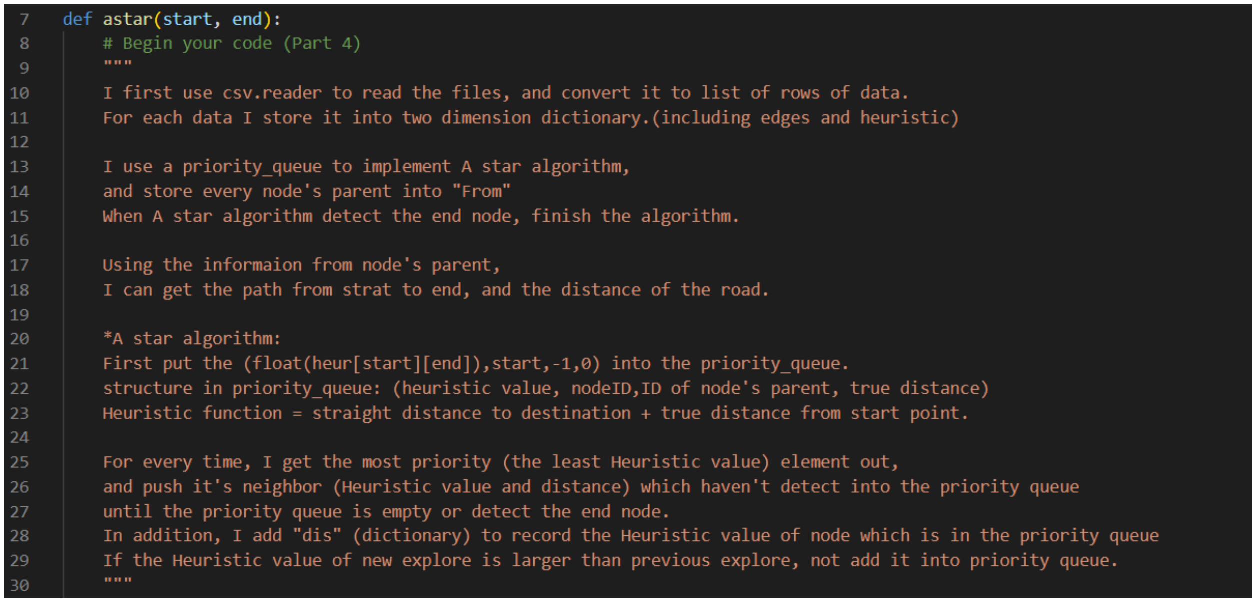
**2. A star algorithm: First put (heur[startID][endID],start node,-1,0)(heuristic value, nodeID, parentID, true distance)into the priority_queue.**

**For every time, I get the closest element out, and push its neighbor which haven't detect into priority_queue.**

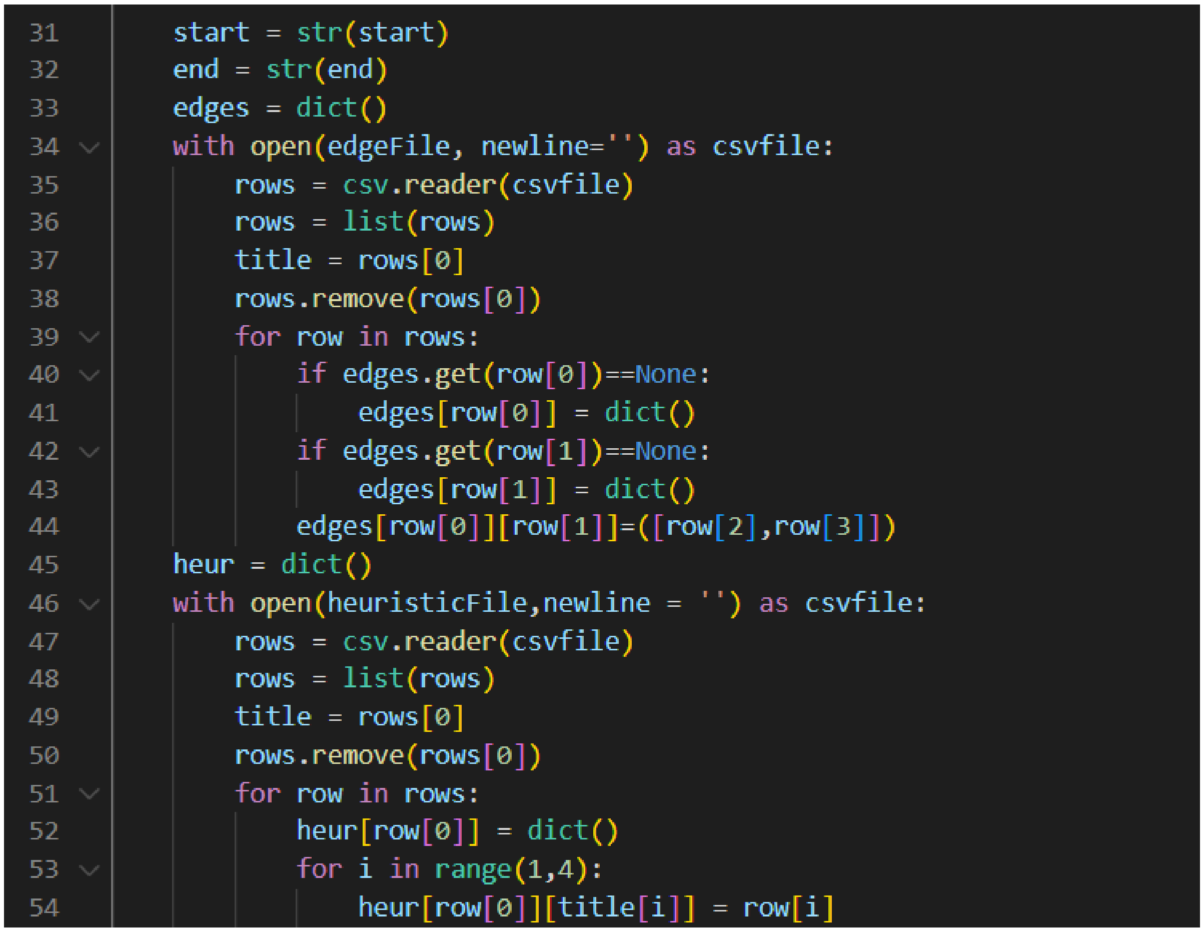
**until the priority_queue is empty or detect the end node.**

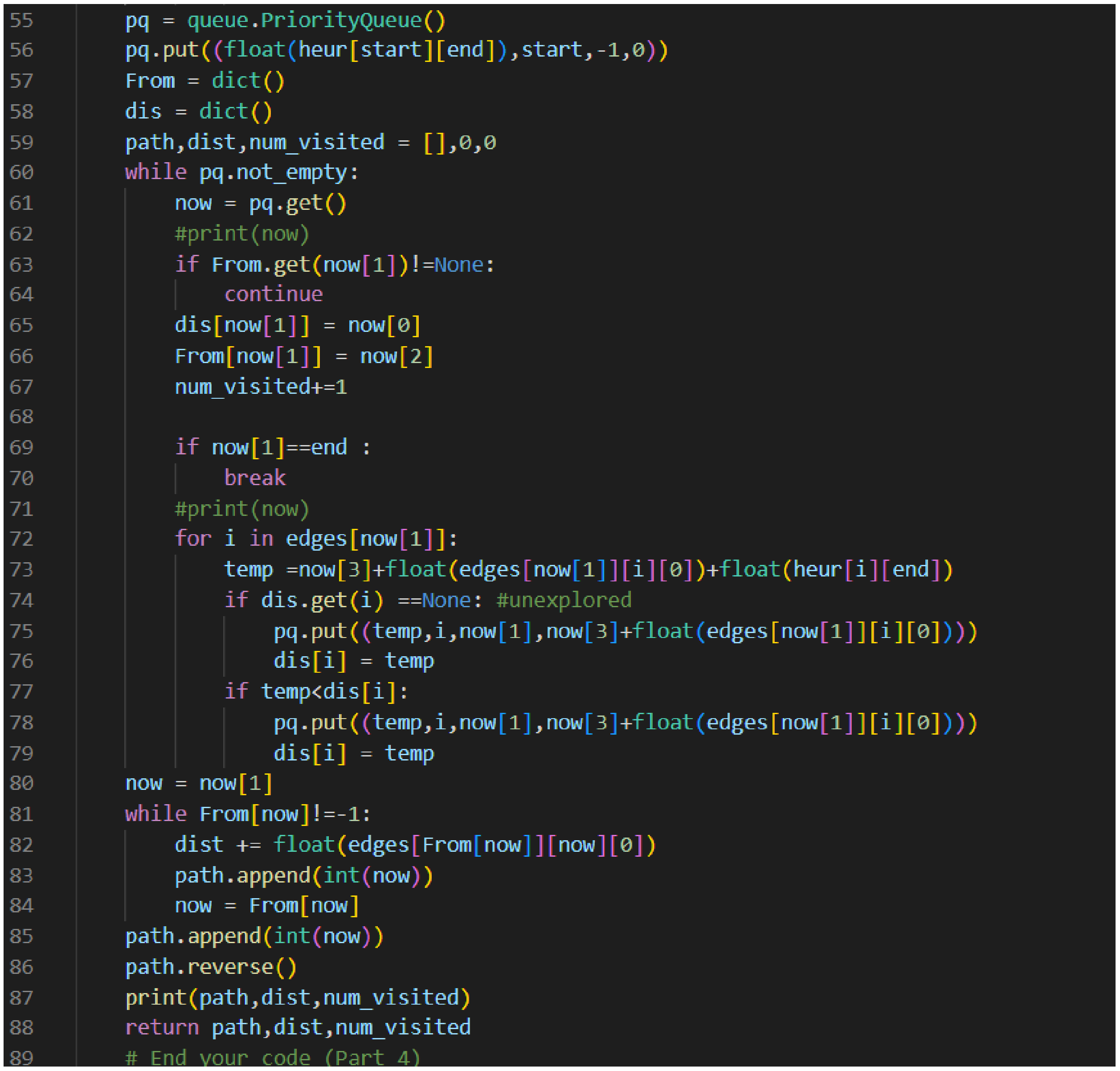
*** heuristic function: straight distance to end point + true distance from start point.**

**3. get result in the A star: I store every node's parent, from the information, I can get the path, and calculate the distance.**



```python
def astar(start, end):
    # Begin your code (Part 4)
    """
    I first use csv.reader to read the files, and convert it to list of rows of data.
    For each data I store it into two dimension dictionary.(including edges and heuristic)

    I use a priority_queue to implement A star algorithm,
    and store every node's parent into "From"
    When A star algorithm detect the end node, finish the algorithm.

    Using the informaion from node's parent,
    I can get the path from strat to end, and the distance of the road.

    *A star algorithm:
    First put the (float(heur[start][end]),start,-1,0) into the priority_queue.
    structure in priority_queue: (heuristic value, nodeID,ID of node's parent, true distance)
    Heuristic function = straight distance to destination + true distance from start point.

    For every time, I get the most priority (the least Heuristic value) element out,
    and push it's neighbor (Heuristic value and distance) which haven't detect into the priority queue
    until the priority queue is empty or detect the end node.
    In addition, I add "dis" (dictionary) to record the Heuristic value of node which is in the priority queue
    If the Heuristic value of new explore is larger than previous explore, not add it into priority queue.
    """
```

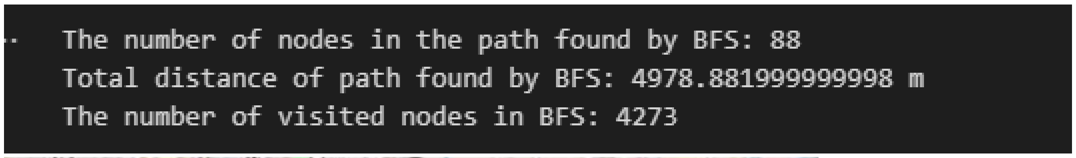

```
    start = str(start)
    end = str(end)
    edges = dict()
    with open(edgeFile, newline='') as csvfile:
        rows = csv.reader(csvfile)
        rows = list(rows)
        title = rows[0]
        rows.remove(rows[0])
        for row in rows:
            if edges.get(row[0])==None:
                edges[row[0]] = dict()
            if edges.get(row[1])==None:
                edges[row[1]] = dict()
            edges[row[0]][row[1]]=([row[2],row[3]])
    heur = dict()
    with open(heuristicFile,newline = '') as csvfile:
        rows = csv.reader(csvfile)
        rows = list(rows)
        title = rows[0]
        rows.remove(rows[0])
        for row in rows:
            heur[row[0]] = dict()
            for i in range(1,4):
                heur[row[0]][title[i]] = row[i]
```



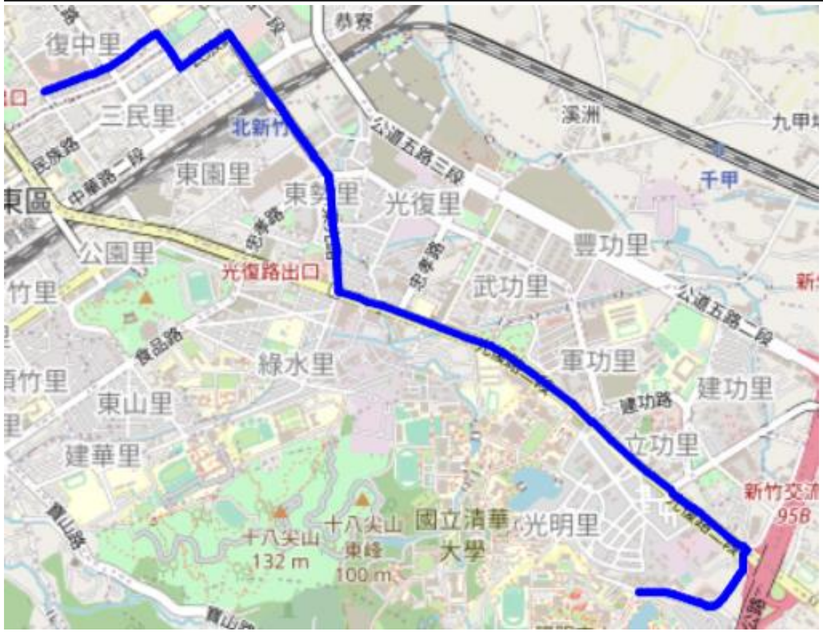
```
    pq = queue.PriorityQueue()
    pq.put((float(heur[start][end]),start,-1,0))
    From = dict()
    dis = dict()
    path,dist,num_visited = [],0,0
    while pq.not_empty:
        now = pq.get()
        #print(now)
        if From.get(now[1])!=None:
            continue
        dis[now[1]] = now[0]
        From[now[1]] = now[2]
        num_visited+=1

        if now[1]==end :
            break
        #print(now)
        for i in edges[now[1]]:
            temp =now[3]+float(edges[now[1]][i][0])+float(heur[i][end])
            if dis.get(i) ==None: #unexplored
                pq.put((temp,i,now[1],now[3]+float(edges[now[1]][i][0])))
                dis[i] = temp
            if temp<dis[i]:
                pq.put((temp,i,now[1],now[3]+float(edges[now[1]][i][0])))
                dis[i] = temp
    now = now[1]
    while From[now]!=-1:
        dist += float(edges[From[now]][now][0])
        path.append(int(now))
        now = From[now]
    path.append(int(now))
    path.reverse()
    print(path,dist,num_visited)
    return path,dist,num_visited
    # End your code (Part 4)
```

## Part II. Results & Analysis

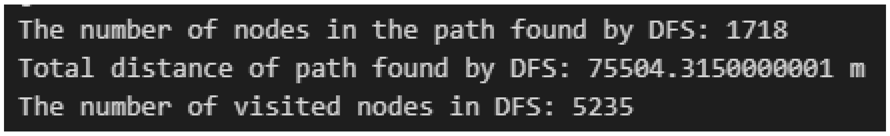
- **Test1: from National Yang Ming Chiao Tung University (ID: 2270143902) to Big City Shopping Mall (ID: 1079387396)**



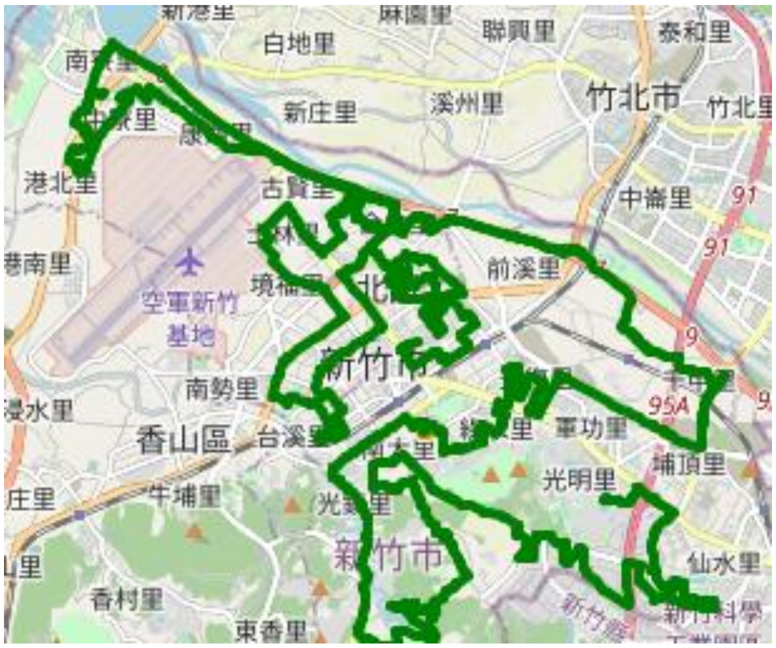
```
The number of nodes in the path found by BFS: 88
Total distance of path found by BFS: 4978.881999999998 m
The number of visited nodes in BFS: 4273
```



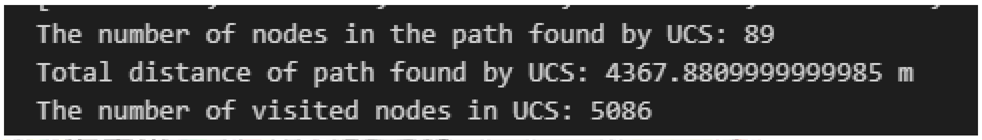
**DFS(stack):**



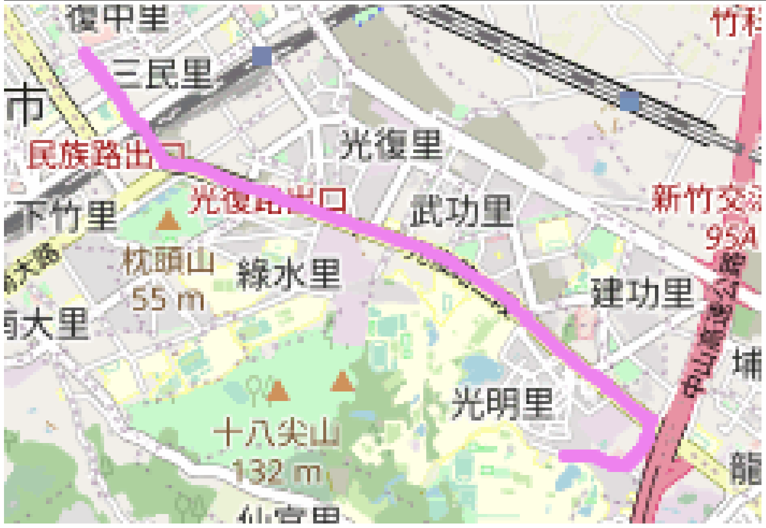
```
The number of nodes in the path found by DFS: 1718
Total distance of path found by DFS: 75504.3150000001 m
The number of visited nodes in DFS: 5235
```



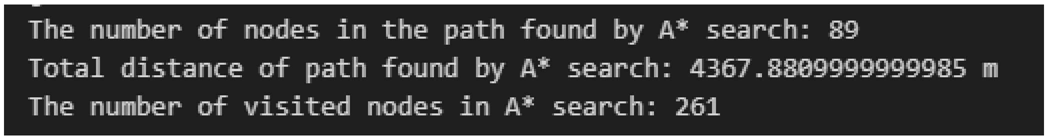
**UCS:**



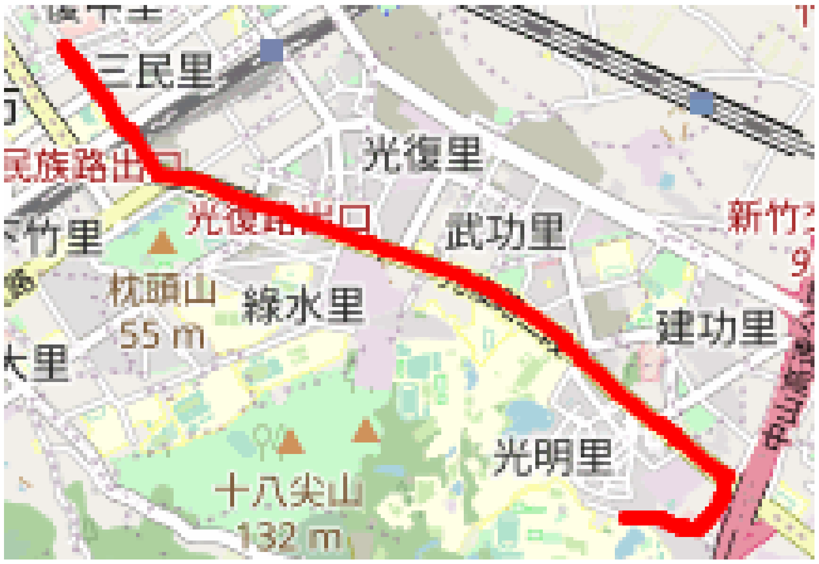
```
The number of nodes in the path found by UCS: 89
Total distance of path found by UCS: 4367.8809999999985 m
The number of visited nodes in UCS: 5086
```



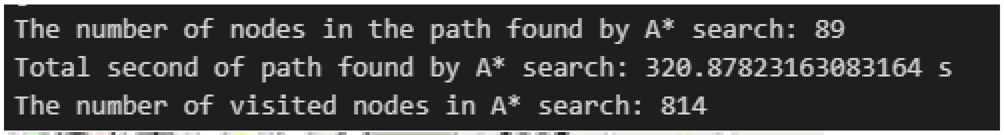
**A star:**



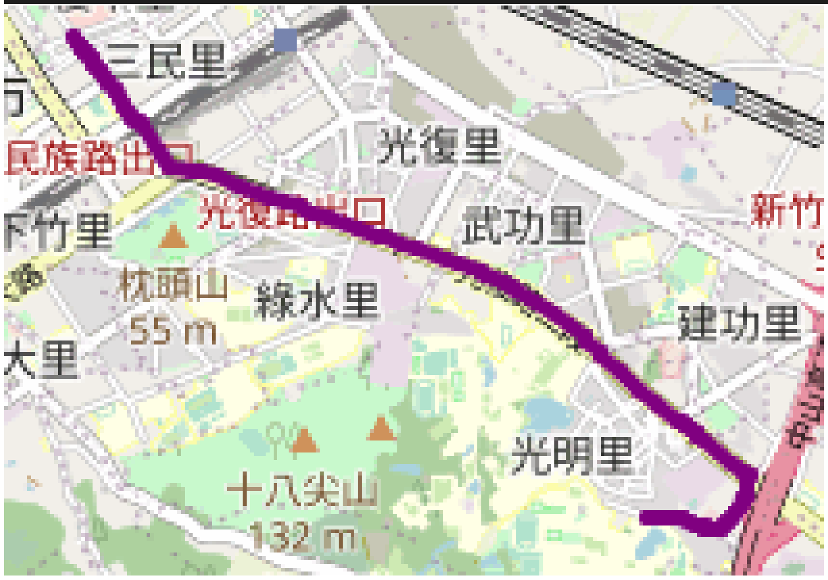
```
The number of nodes in the path found by A* search: 89
Total distance of path found by A* search: 4367.8809999999985 m
The number of visited nodes in A* search: 261
```



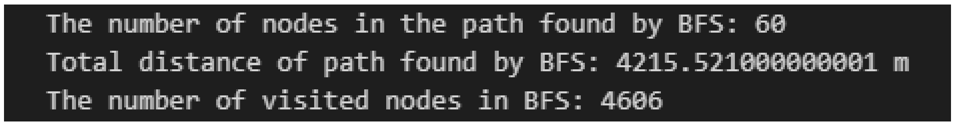
**A star time:**



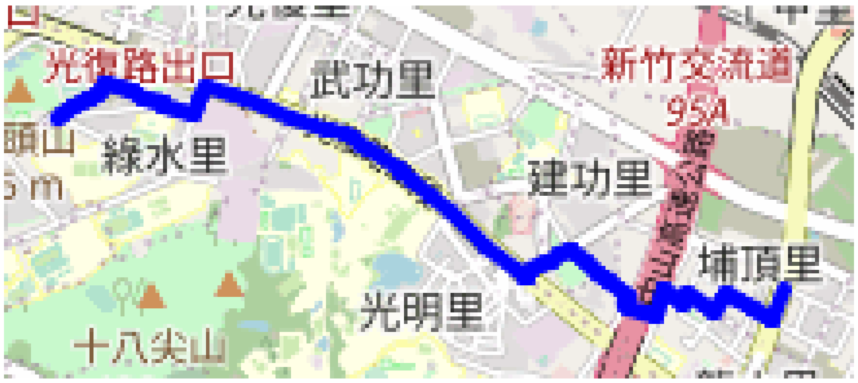
```
The number of nodes in the path found by A* search: 89
Total second of path found by A* search: 320.87823163083164 s
The number of visited nodes in A* search: 814
```



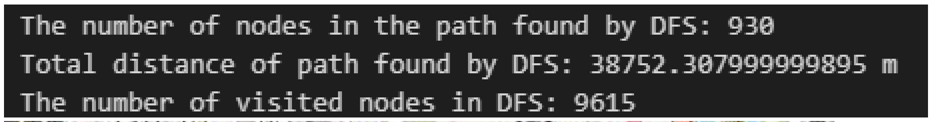
- **Test2: from Hsinchu Zoo (ID: 426882161) to COSTCO Hsinchu Store (ID: 1737223506)**
  **BFS:**



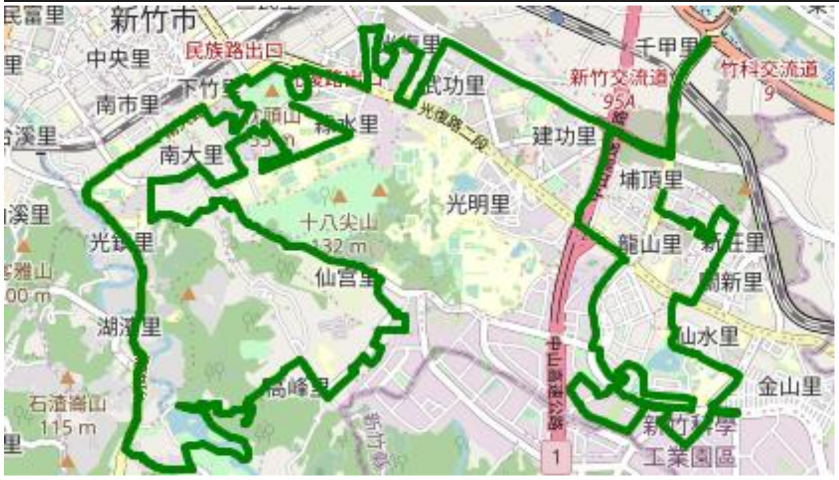
```
The number of nodes in the path found by BFS: 60
Total distance of path found by BFS: 4215.521000000001 m
The number of visited nodes in BFS: 4606
```



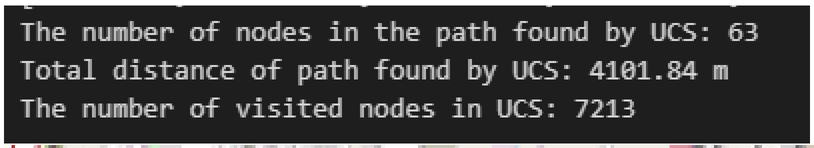
**DFS(stack):**



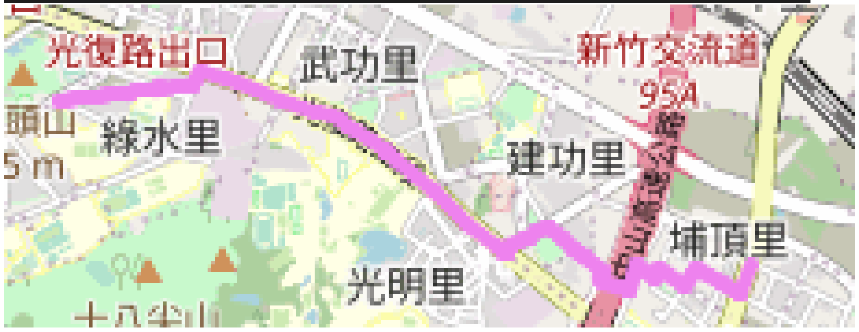
```
The number of nodes in the path found by DFS: 930
Total distance of path found by DFS: 38752.307999999895 m
The number of visited nodes in DFS: 9615
```



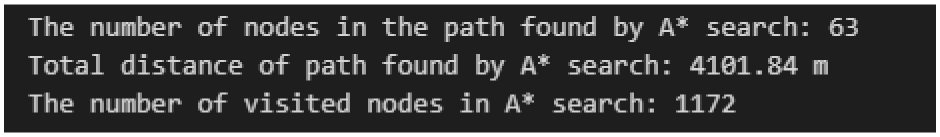
**UCS:**



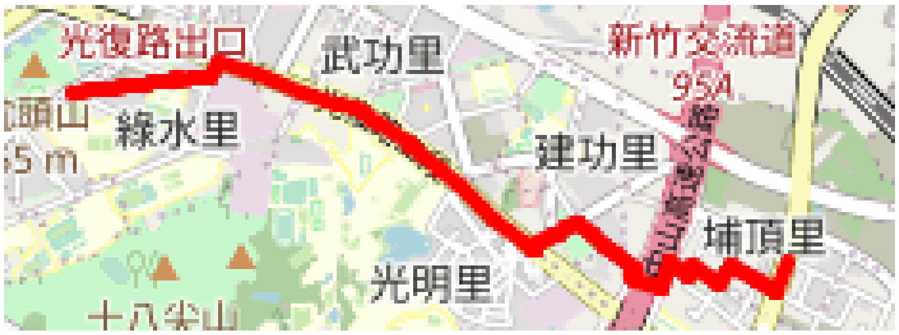
```
The number of nodes in the path found by UCS: 63
Total distance of path found by UCS: 4101.84 m
The number of visited nodes in UCS: 7213
```



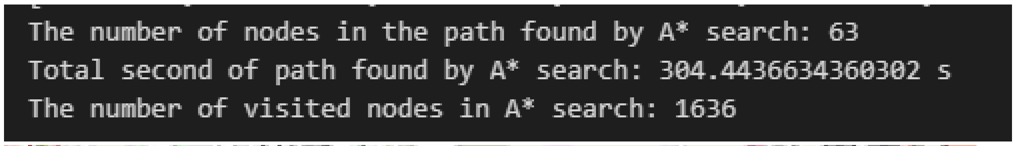
**A star:**



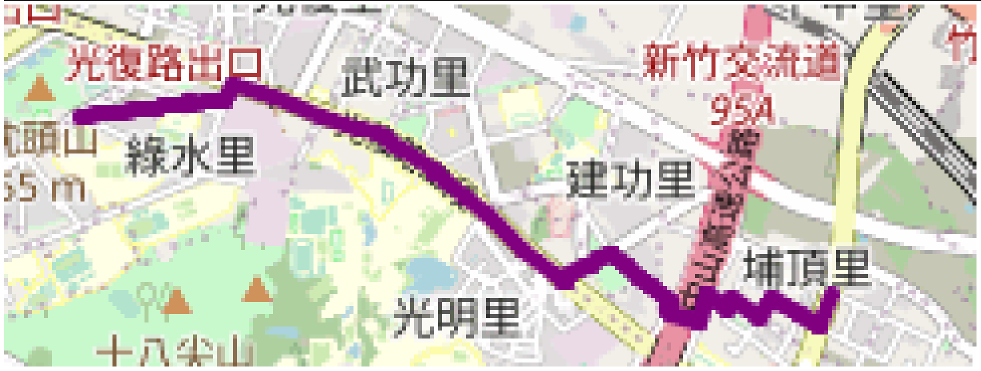
```
The number of nodes in the path found by A* search: 63
Total distance of path found by A* search: 4101.84 m
The number of visited nodes in A* search: 1172
```



**A star time:**

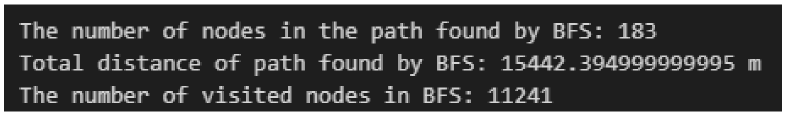


```
The number of nodes in the path found by A* search: 63
Total second of path found by A* search: 304.4436634360302 s
The number of visited nodes in A* search: 1636
```

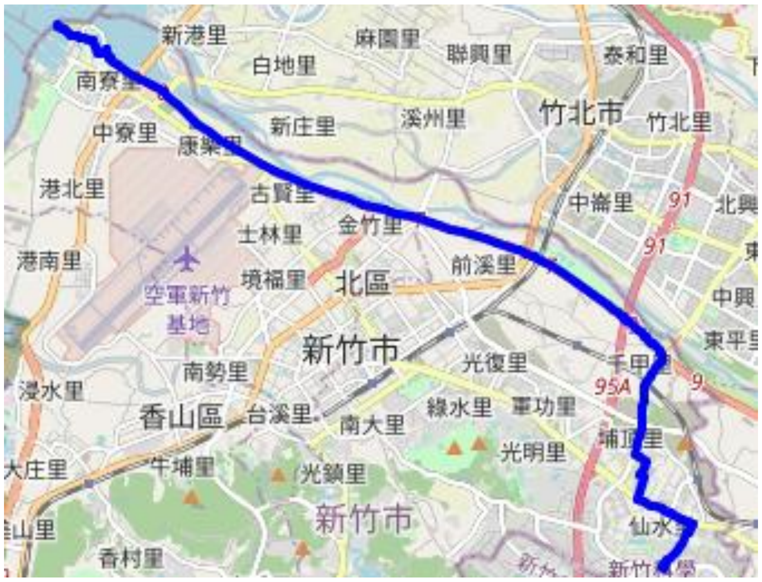


**Test3: from National Experimental High School At Hsinchu Science Park (ID: 1718165260) to Nanliao Fighing Port (ID: 8513026827)**

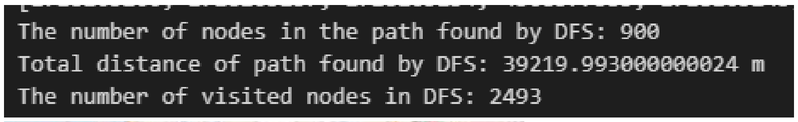
**BFS:**



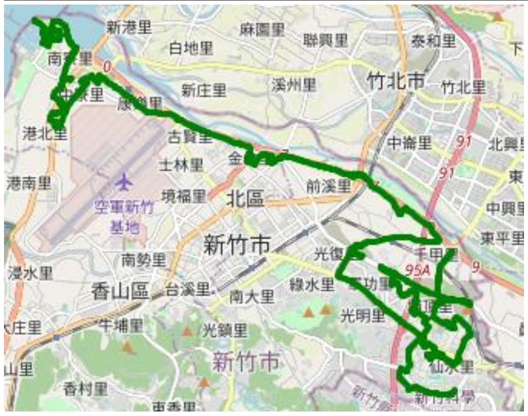
```
The number of nodes in the path found by BFS: 183
Total distance of path found by BFS: 15442.394999999995 m
The number of visited nodes in BFS: 11241
```



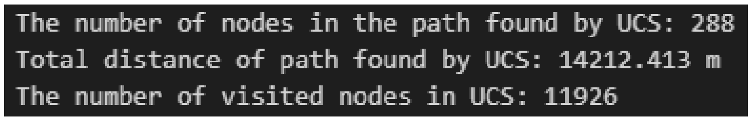
**DFS(stack):**



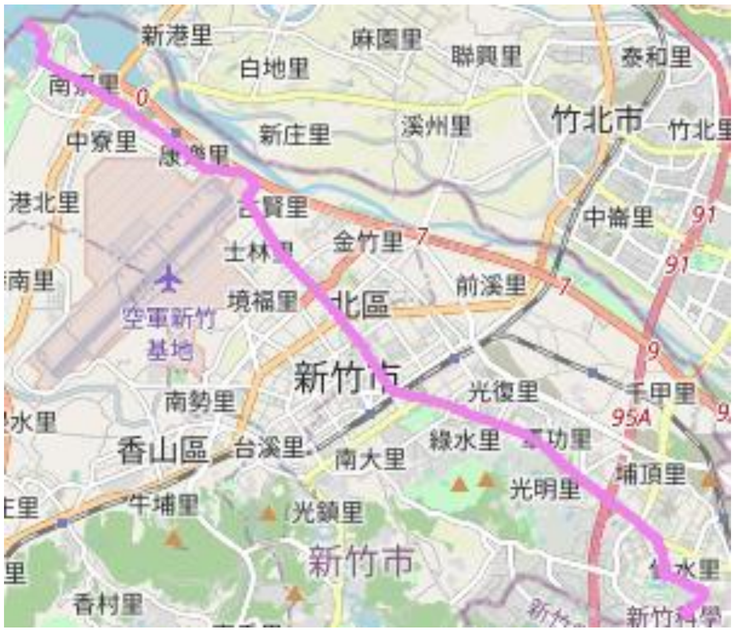
```
The number of nodes in the path found by DFS: 900
Total distance of path found by DFS: 39219.993000000024 m
The number of visited nodes in DFS: 2493
```



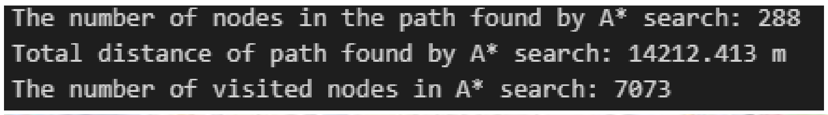
**UCS:**



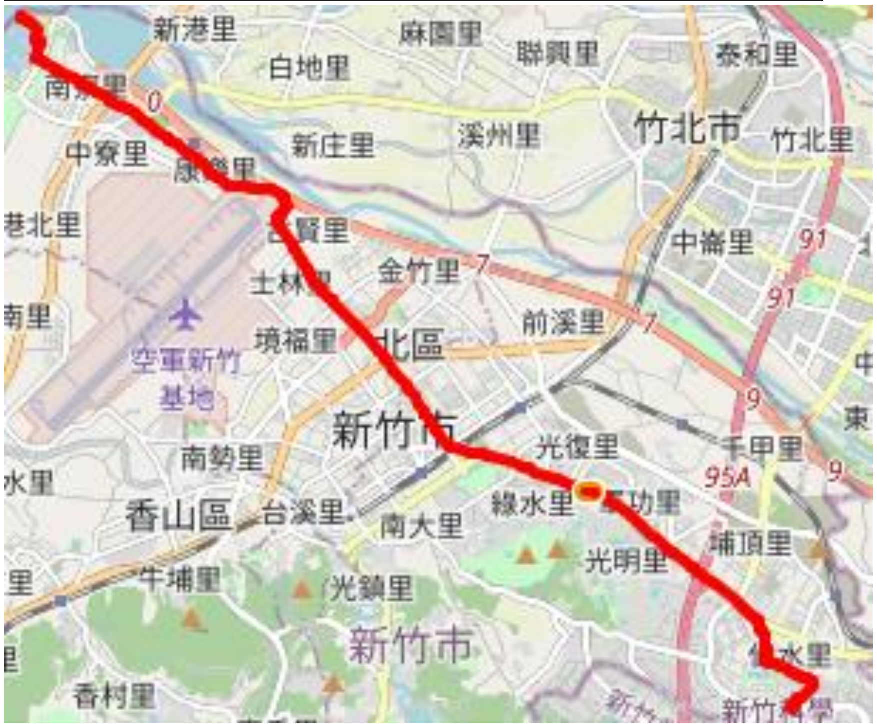
```
The number of nodes in the path found by UCS: 288
Total distance of path found by UCS: 14212.413 m
The number of visited nodes in UCS: 11926
```



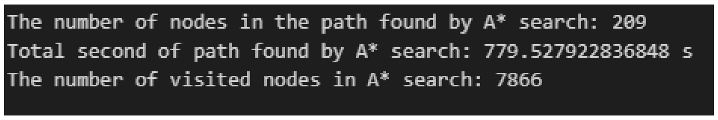
## A star:



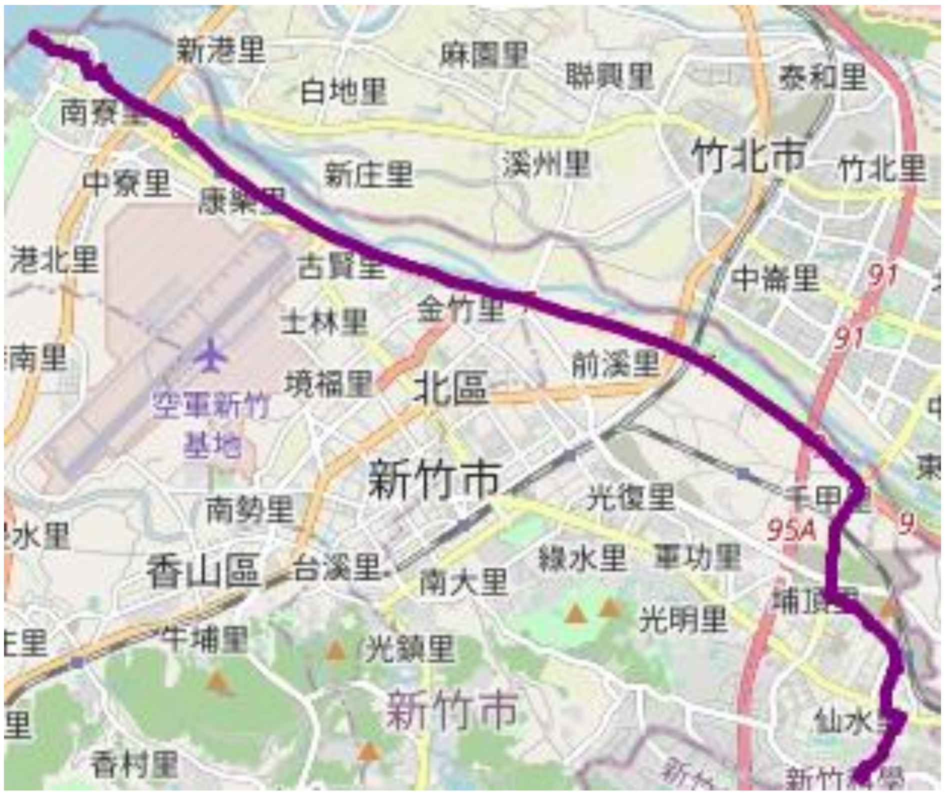
```
The number of nodes in the path found by A* search: 288
Total distance of path found by A* search: 14212.413 m
The number of visited nodes in A* search: 7073
```



## A star time:



```
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: 7866
```



## Part III. Question Answering

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

   The problem I encounter is ucs and A star algorithm, these two algorithm is new for me. So I did a lot of search on the internet, and try to understand how it works. After know how why the heuristic function work and come up with the heuristic function  that works.

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.**

   When the time is commute time, the traffic is heavy on some road. Although the distance is shorter, but the time I travel is longer, so I think the time and the traffic is busy or not is also important to consider. Furthermore, traffic light also need to be considered because some small road have shortest distance, but need to wait longer red light.

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: To let computer know the map, we can use the help from satellite which can overview the road and the data from google map etc. to construct a graph.

   Localization: To locate where the start(user) and destination are. This need the help of GPS to know where the start point is. With the information, we can locate the start and end point and use in path finding system.

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.**

   The element we need to consider is everything the deliver may encounter. Ex: reach the restaurant before meal prepared and deliver need to wait for the meal. Whether the deliver have multiple order and the priority in these order, we assume each delivery need 15 mins0

   ETA = max(to restaurant time, meal prep time) + priority rank *15

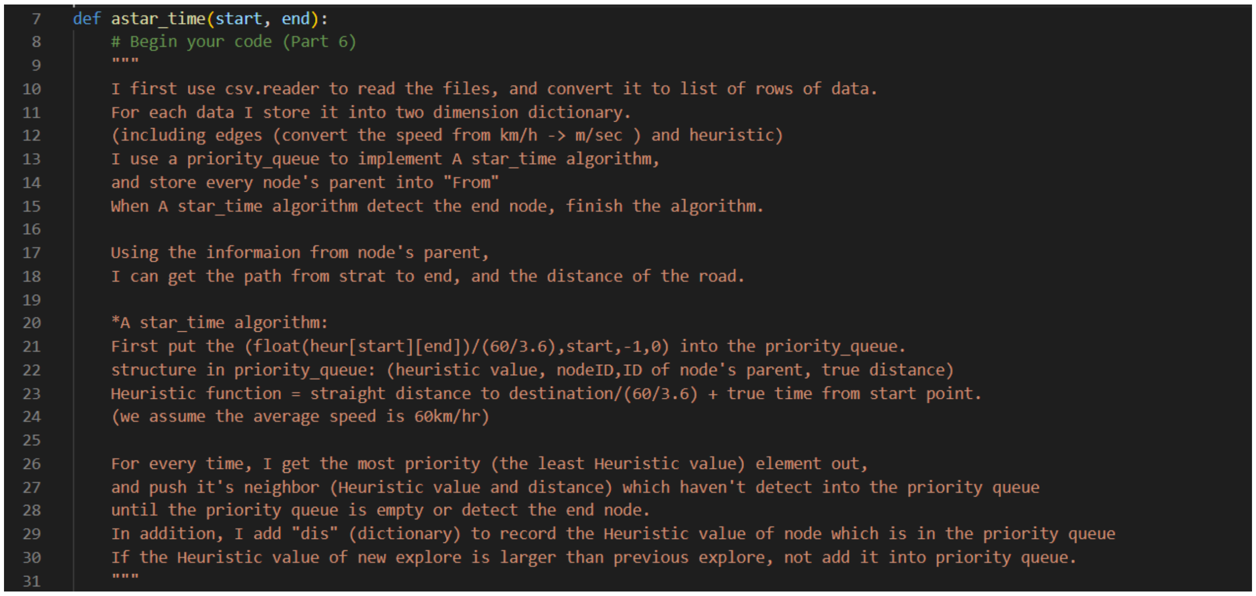
## Bonus: Part 6: Search with a different heuristic

1. **Read data: I use csv.reader to read file, and make each data (start, end, distance, speed limit)into dictionary**

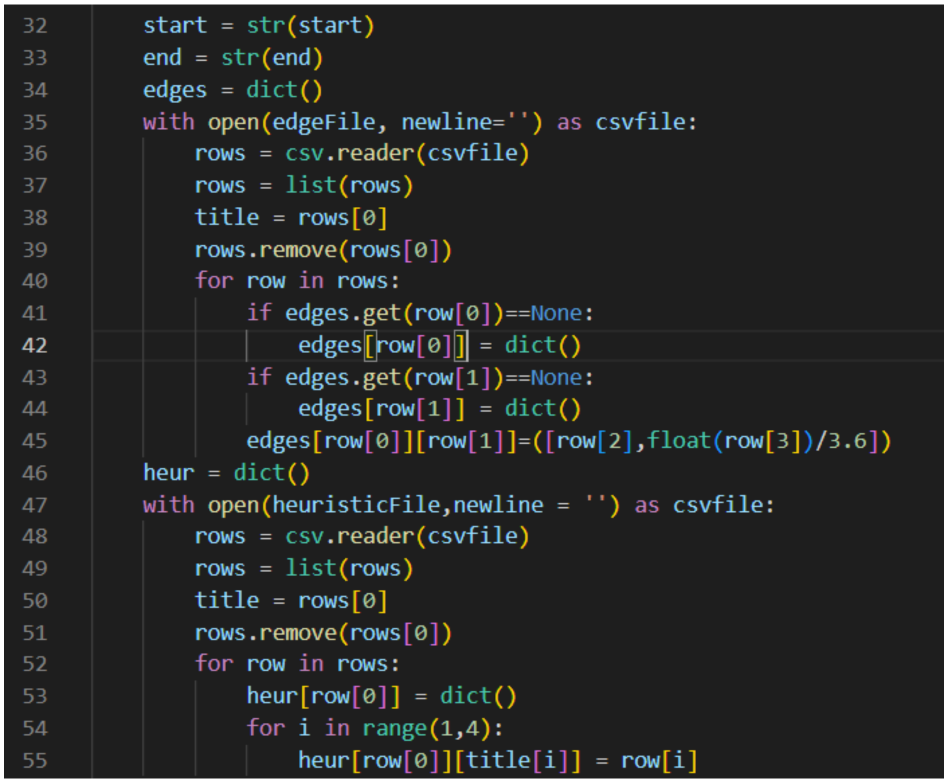
(edges[start][end] = (distance, speed limit)) and (startID, endID, distance) into (heur[startID][endID] = distance)
*convert speed from km/h into m/sec

2. A star time algorithm: First put (heur[startID][endID]/(60/3.6),start node,-1,0)(heuristic value, nodeID, parentID, true time)into the priority_queue.
For every time, I get the closest element out, and push its neighbor which haven't detect into priority_queue.
until the priority_queue is empty or detect the end node.
* heuristic function: straight distance to end point/(60/3.6) + true time from start point. We assume the average is 60km/h

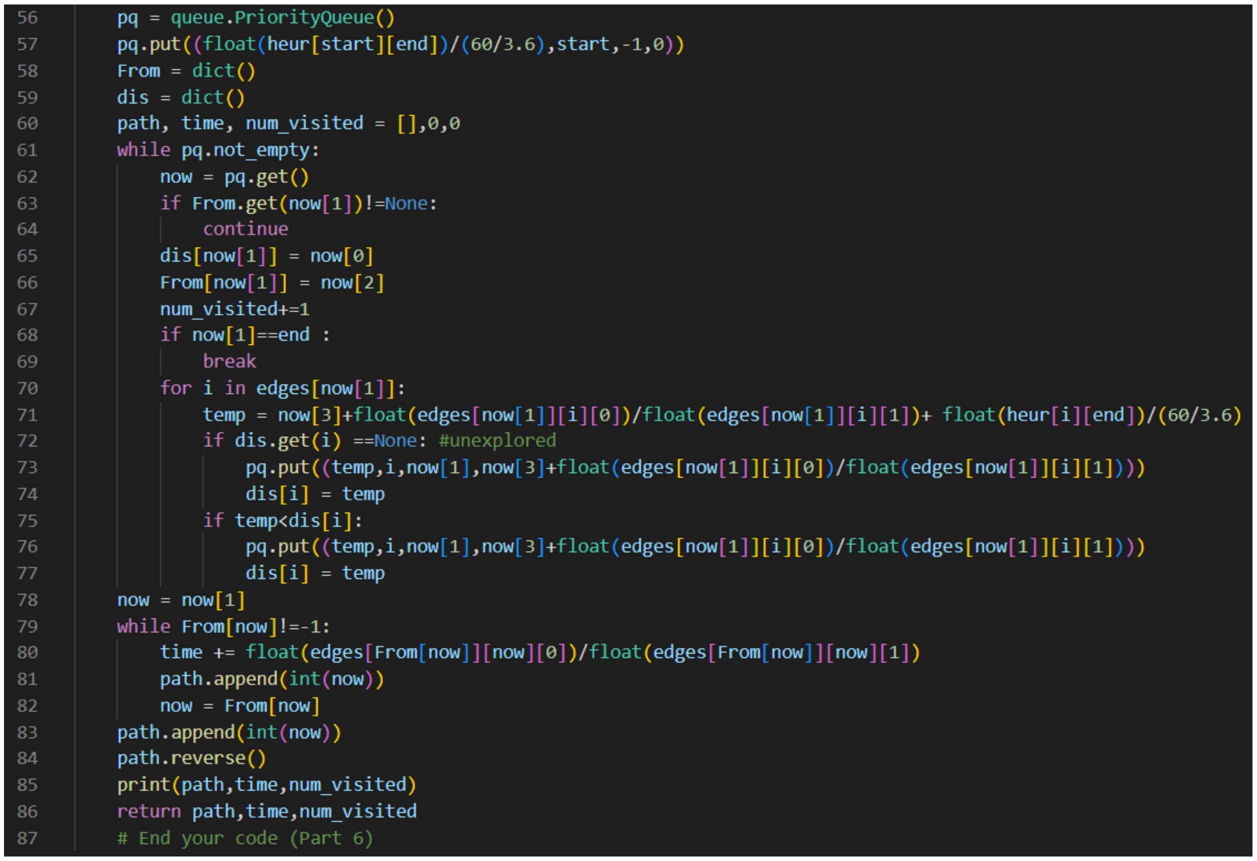
3. get result in the A star time : I store every node's parent, from the information, I can get the path, and calculate the time.



```python
def astar_time(start, end):
    # Begin your code (Part 6)
    """
    I first use csv.reader to read the files, and convert it to list of rows of data.
    For each data I store it into two dimension dictionary.
    (including edges (convert the speed from km/h -> m/sec ) and heuristic)
    I use a priority_queue to implement A star_time algorithm,
    and store every node's parent into "From"
    When A star_time algorithm detect the end node, finish the algorithm.

    Using the informaion from node's parent,
    I can get the path from strat to end, and the distance of the road.

    *A star_time algorithm:
    First put the (float(heur[start][end])/(60/3.6),start,-1,0) into the priority_queue.
    structure in priority_queue: (heuristic value, nodeID,ID of node's parent, true distance)
    Heuristic function = straight distance to destination/(60/3.6) + true time from start point.
    (we assume the average speed is 60km/hr)

    For every time, I get the most priority (the least Heuristic value) element out,
    and push it's neighbor (Heuristic value and distance) which haven't detect into the priority queue
    until the priority queue is empty or detect the end node.
    In addition, I add "dis" (dictionary) to record the Heuristic value of node which is in the priority queue
    If the Heuristic value of new explore is larger than previous explore, not add it into priority queue.
    """
```

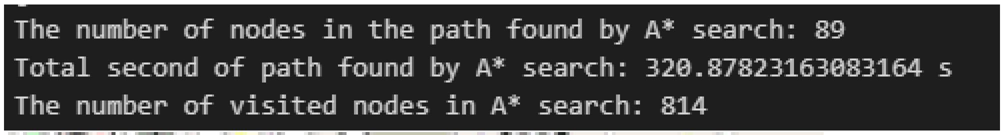


```python
    start = str(start)
    end = str(end)
    edges = dict()
    with open(edgeFile, newline='') as csvfile:
        rows = csv.reader(csvfile)
        rows = list(rows)
        title = rows[0]
        rows.remove(rows[0])
        for row in rows:
            if edges.get(row[0])==None:
                edges[row[0]] = dict()
            if edges.get(row[1])==None:
                edges[row[1]] = dict()
            edges[row[0]][row[1]]=([row[2],float(row[3])/3.6])
    heur = dict()
    with open(heuristicFile,newline = '') as csvfile:
        rows = csv.reader(csvfile)
        rows = list(rows)
        title = rows[0]
        rows.remove(rows[0])
        for row in rows:
            heur[row[0]] = dict()
            for i in range(1,4):
                heur[row[0]][title[i]] = row[i]
```

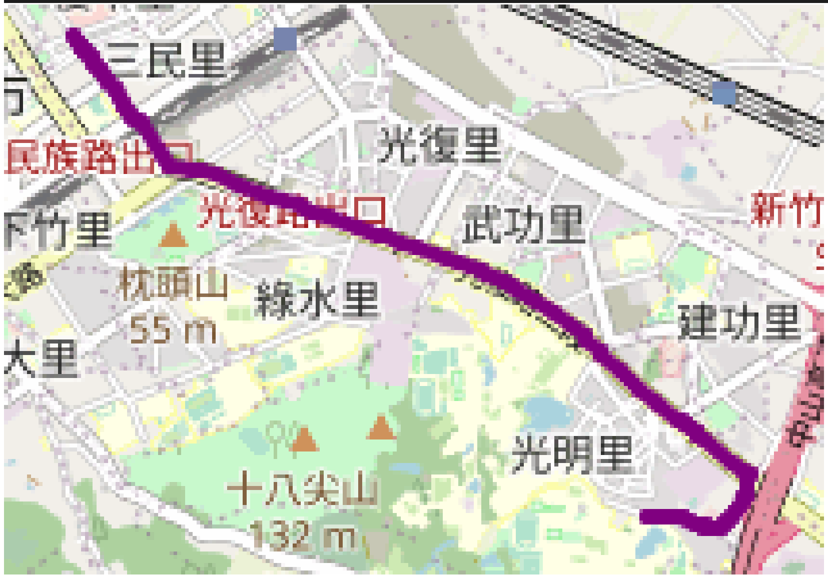


```python
    pq = queue.PriorityQueue()
    pq.put((float(heur[start][end])/(60/3.6),start,-1,0))
    From = dict()
    dis = dict()
    path, time, num_visited = [],0,0
    while pq.not_empty:
        now = pq.get()
        if From.get(now[1])!=None:
            continue
        dis[now[1]] = now[0]
        From[now[1]] = now[2]
        num_visited+=1
        if now[1]==end :
            break
        for i in edges[now[1]]:
            temp = now[3]+float(edges[now[1]][i][0])/float(edges[now[1]][i][1])+ float(heur[i][end])/(60/3.6)
            if dis.get(i) ==None: #unexplored
                pq.put((temp,i,now[1],now[3]+float(edges[now[1]][i][0])/float(edges[now[1]][i][1])))
                dis[i] = temp
            if temp<dis[i]:
                pq.put((temp,i,now[1],now[3]+float(edges[now[1]][i][0])/float(edges[now[1]][i][1])))
                dis[i] = temp
    now = now[1]
    while From[now]!=-1:
        time += float(edges[From[now]][now][0])/float(edges[From[now]][now][1])
        path.append(int(now))
        now = From[now]
    path.append(int(now))
    path.reverse()
    print(path,time,num_visited)
    return path,time,num_visited
    # End your code (Part 6)
```

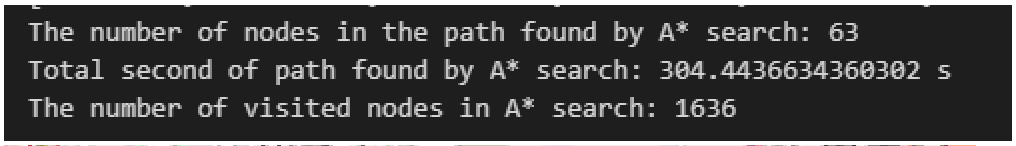
**Test1:**



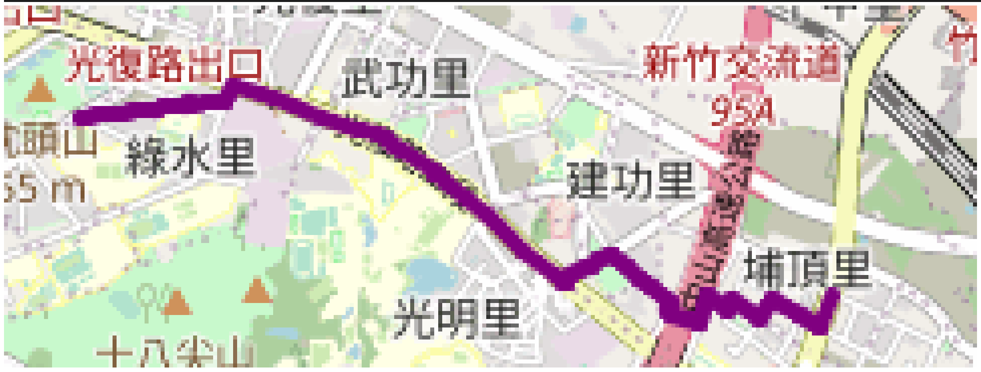
```
The number of nodes in the path found by A* search: 89
Total second of path found by A* search: 320.87823163083164 s
The number of visited nodes in A* search: 814
```



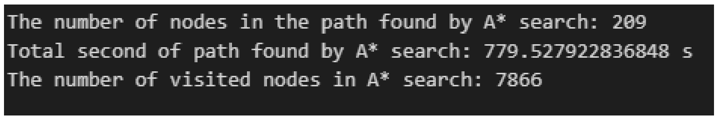
**Test2:**



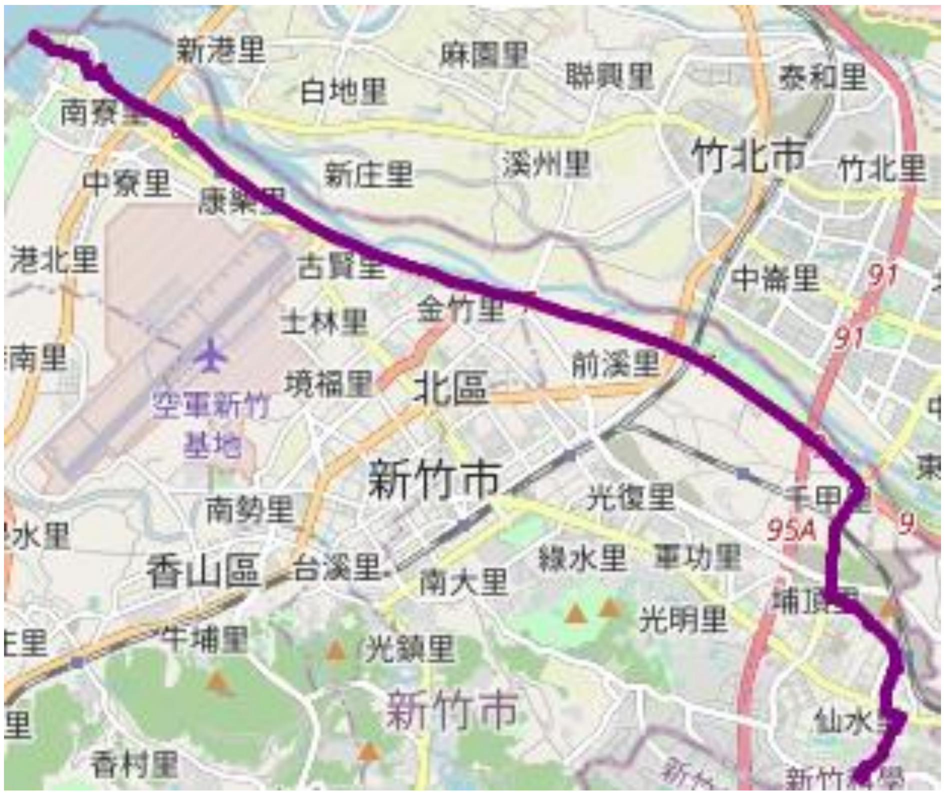
```
The number of nodes in the path found by A* search: 63
Total second of path found by A* search: 304.4436634360302 s
The number of visited nodes in A* search: 1636
```



**Test 3:**



```
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: 7866
```


新港里
麻園里
聯興里
泰和里
白地里
南寮里
竹北市
竹北里
溪州里
中寮里
新庄里
康樂里
港北里
古賢里
中崙里
金竹里
士林里
南里
前溪里
北區
境福里
空軍新竹基地
新竹市
光復里
千甲里
南勢里
水里
綠水里
軍功里
香山區
台溪里
南大里
埔頂里
光明里
牛埔里
光鎮里
新竹市
仙水里
香村里