Homework 2: Route Finding

110550080何曉嫻

Part I. Implementation (6%):

Part 1

```
    # Begin your code (Part 1)
    """
    1. Load the csv file into rows
    2. Store them into the list adj in the format:
        adj[from]= {[to, distance],[ ],...}
    3. Use queue to implement bfs, visit store the isited nodes
    4. parent{[,]} stores node's parent and their distance
    5. Run bfs
    6. Trace back parent to compute distance and path
    7. Return
    """

    adj=collections.defaultdict(list)
    with open(edgeFile,newline='') as file:
        content=csv.reader(file)
        headers = next(content)
        for row in content:
            adj[int(row[0])].append([int(row[1]), float(row[2])])

    num_visited=0
    parent={}
    visit =[]
    q =queue.Queue()
    q.put(start)
    visit.append(start)
    flag=1

    while (not q.empty()) & flag==1:
        node=q.get()
        for v in adj[node]:
            if(v[0]==end):
                parent[v[0]]=[node,0]
                flag=0
                break
            if v[0] not in visit:
                num_visited += 1
                visit.append(v[0])
                parent[v[0]] = [node, v[1]]
                q.put(v[0])

    tmp=end
    dist=0.0
    path=[]

    while (tmp!=start):
        path.insert(1,tmp)
        dist+=parent[tmp][1]
        tmp=parent[tmp][0]
    path.insert(1,start)

    return path, dist, num_visited
    #raise NotImplementedError("To be implemented")
    # End your code (Part 1)
```

Part 2

```python
def dfs(start, end):
    # Begin your code (Part 2)
    """
    1. Load the csv file into rows, and convert into adjacent list as above
    2. Use list to implement stack, .pop() return and delete the last element
       .append() push the element
    4. parent{[,]} stores node's parent and their distance
    5. Run dfs
    6. Trace back parent to compute distance and path
    7. Return
    """
    adj=collections.defaultdict(list)
    with open(edgeFile,newline='') as file:
        content=csv.reader(file)
        headers = next(content)
        for row in content:
            adj[int(row[0])].append([int(row[1]), float(row[2])])

    num_visited=0
    parent={}
    visit =[]
    stack =[start]
    visit.append(start)
    flag=1

    while (len(stack)!=0) & flag==1:
        node=stack.pop()
        visit.append(node)
        num_visited+=1
        for v in adj[node]:
            if(v[0]==end):
                parent[v[0]]=[node,0]
                flag=0
                break
            if v[0] not in visit:
                #num_visited += 1
                #visit.append(v[0])
                parent[v[0]] = [node, v[1]]
                stack.append(v[0])

    tmp=end
    dist=0.0
    path=[]

    while (tmp!=start):
        path.insert(1,tmp)
        dist+=parent[tmp][1]
        tmp=parent[tmp][0]
    path.insert(1,start)

    return path, dist, num_visited
    #raise NotImplementedError("To be implemented")
    # End your code (Part 2)
```

Part 3

```python
    # Begin your code (Part 3)
    """
    1. Load the csv file into rows, and convert into adjacent list as above
    2. pre store the[diatance, start, end] of each roads
    3. parent stores node's parent
    4. Run ucs: add all road can reached from start to pre
                sort pre and choose the shortest one as next road
                add that road's destination(node[2]) to visit
                add all roads adjacent to node[2] to pre
                run the loop till find end
    6. Trace back parent to compute distance and path
    7. Return
    """
    adj=collections.defaultdict(list)
    with open(edgeFile,newline='') as file:
        content=csv.reader(file)
        headers = next(content)
        for row in content:
    num_visited=1
    pre=[] # dis ,from, to
    parent={}
    visit =[start]

    for v in adj[start]:
        if v[0] not in visit:
            pre.append([v[1], start, v[0]])

    while pre:
        pre=sorted(pre) #get top element(smallest dist)
        node=pre[0]
        del pre[0]

        if(node[2]==end):
            parent[node[2]]=node[1]

        if node[2] not in visit:
            visit.append(node[2])
            num_visited+=1
            parent[node[2]]=node[1]
            for v in adj[node[2]]:
                pre.append([node[0]+v[1], node[2], v[0]])

    tmp=end
    path=[]

    while (tmp!=start):
        path.insert(1,tmp)
        tmp=parent[tmp]
    path.insert(1,start)

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

Part 4

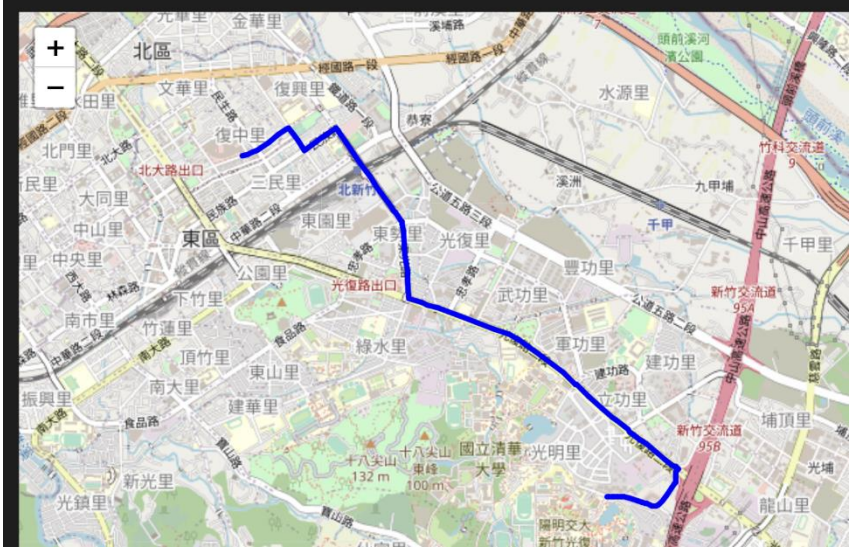
```python
    # Begin your code (Part 4)
    """
    1. Load the csv file, and convert into adjacent list and heuristic function
    2. parent store parent, dis store distance from start to the node
    3. open_list is a list of nodes which have been visited, but their neighbors
       haven't all been inspected, starts off with the start node
       closed_list is a list of nodes which have been visited
       and who's neighbors have been inspected
    4. Run astar:
    5. Find a node with the lowest value of f()
    6. If the current node isn't in both open_list and closed_list
       add it to open_list and note n as it's parent
       Otherwise, check if it's closer to first visit n, then m
    7. Remove n from the open_list, and add it to closed_list
    8. Return
    """
    adj=collections.defaultdict(list)
    with open(edgeFile,newline='') as file:
        content=csv.reader(file)
        headers = next(content)
        for row in content:
            adj[int(row[0])].append([int(row[1]), float(row[2])])

    h={}
    with open(heuristicFile,newline='') as file:
        content=csv.reader(file)
        headers = next(content)
        for row in content:
            h[int(row[0])]=(float(row[1]))

    opened= set([start])
    closed= set([])
    dis = {}
    dis[start] = 0
    parent={}
    parent[start]= 0
    num_visited=0


    while opened:
        n=None
        num_visited+=1
        for v in opened:
            if n==None or dis[v]+h[v] < dis[n]+h[n]:
                n=v

        if n==end:
            dist=dis[n]
            path=[]
            tmp=end
            while tmp!=start:
                path.append(tmp)
                tmp=parent[tmp]
            path.append(start)

        for m in adj[n]:
            if m[0] not in opened and m[0] not in closed:
                opened.add(m[0])
                parent[m[0]]=n
                dis[m[0]]=dis[n]+m[1]
            else:
                if dis[m[0]]>dis[n]+m[1]:
                    dis[m[0]]=dis[n]+m[1]
                    parent[m[0]]=n
                    if m[0] in closed:
                        closed.remove(m[0])
                        opened.add(m[0])

        opened.remove(n)
        closed.add(n)
    return path, dist, num_visited

    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: 4780.899999999999 m
The number of visited nodes in BFS: 4272
```



DFS (stack):

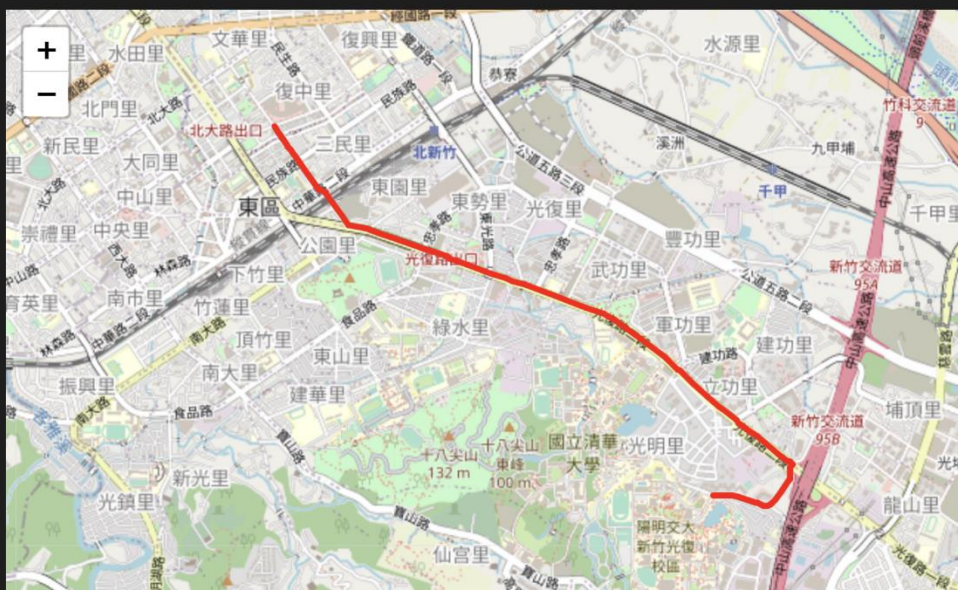
```
The number of nodes in the path found by DFS: 1232
Total distance of path found by DFS: 57116.384000000005 m
The number of visited nodes in DFS: 4380
```

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: 5085
```

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: 12054
```



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

BFS:

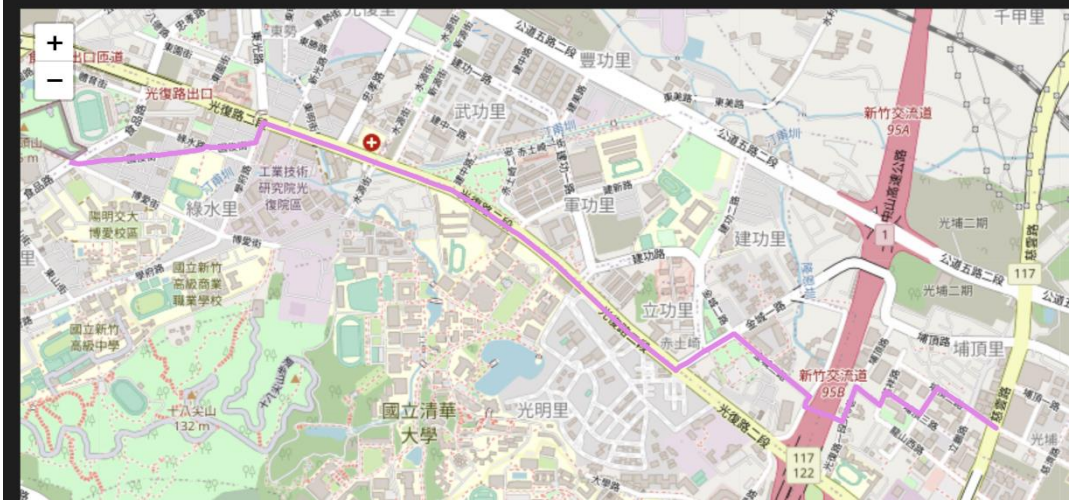
```
The number of nodes in the path found by BFS: 60
Total distance of path found by BFS: 4058.854999999999 m
The number of visited nodes in BFS: 4605
```

DFS (stack):

```
The number of nodes in the path found by DFS: 998
Total distance of path found by DFS: 40937.99199999992 m
The number of visited nodes in DFS: 8627
```

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: 7212
```



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: 12054
```

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: 15426.745999999996 m
The number of visited nodes in BFS: 11240
```

DFS (stack):

```
The number of nodes in the path found by DFS: 1521
Total distance of path found by DFS: 64805.95499999999 m
The number of visited nodes in DFS: 3370
```

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: 11925
```

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: 12054
```

Analysis: From results above, we can know dfs find the worst path. The path found by ucs and A* are almost same.

Part III. Question Answering (12%):

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

   My dfs finds a different path with the answer ta provides. I tried to change the place where I put "num_visited +=1, visit.append()" . I found that I marked the node(A) as visit whe n I put it into stack , this will make some node adjacent to A not discovered and put into st ack. So, I marked the node as visit when it is gotten from the stack.

2. Besides speed limit and distance, could you please come up with another attribute that is es sential for route finding in the real world? Please explain the rationale.

   Road conditions, like traffic jam, car accident and traffic light. For example, if there is hea vy traffic on a particular route due to a traffic jam or accident, it may be faster to take a dif ferent route even if it is longer in distance. Similarly, if there are several traffic lights on on e route, and another route has fewer traffic lights, the latter may be a better option, even if i t is slightly longer.

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: Satellite imagery, it can be used to create maps of large areas.

   localization: GPS, It can be used to determine the location of a device with high accuracy.

4. The estimated time of arrival (ETA) is one of the features of Uber Eats. To provide accurat e estimates for users, Uber Eats needs to dynamically update ETA based on their mechanis m. Please define a **dynamic heuristic equation** for ETA and explain the rationale of your de sign. Hint: You can consider meal prep time, delivery priority, multiple orders, etc.

   heuristic equation for ETA may be : $h = D/S + T + C$
   D is the distance to the destination.
   S is the average speed on the route.
   T is the estimated delay due to traffic, accident, or other factors.
   C is estimated due to the probable time that delivery man deliver food to other customer b efore that user.

The rationale behind this design is to provide an estimate of the travel time that takes into account real-time information about the current traffic conditions. By using a heuristic app roach, the equation can adapt to changes in traffic conditions and provide a more accurate estimate of the travel time.