Homework 2: Route Finding

Report Template

Please keep the title of each section and delete examples. Note that please keep the questions listed in Part III.

Part I. Implementation (6%):

- **Please screenshot your code snippets of Part 1 ~ Part 4, and explain your implementation. For example,**

BFS

```
5   def bfs(start, end):
6       # Begin your code (Part 1)
7       """
8           Open edges.csv first,then use defaultdict to construct lists named dict and visited.
9           'dict' is used to save the end node,distance and speed limit of certain start node.
10          'visited' is used to save whether the node is searched or not.
11          Use readlines and split to store every datas,then I construct list named explored,fronter
12          and distance.
13          'explored' is used to save the order of nodes that we should search.
14          'fronter' is used to save the parent of the node.
15          Pop the first data of explored to find out the start node(the node is named 'begin') we use
16          it to find out the end node from dict(the end node is named 'final').
17          If 'final' is not visited,append it to explored and plus one to num_visited.
18          'dist' is the sum of the distance in edges.csv of the number in fronter.
19          Finally,use path to save the nodes that are included in the path,and change it to
20          integer.
21
22      """
23      file=open('edges.csv')
24      dict=defaultdict(list)
25      visited=defaultdict(str)
26      lines=file.readlines()
27
28      for line in lines:
29          detail=line.split(',')
30          data=[]
31          data.append(detail[1])
32          data.append(detail[2])
33          data.append(detail[3].replace("\n",""))
34          dict[detail[0]].append(data)
35          visited[detail[0]]=False
36      num_visited=0
37      explored=[]
38      explored.append(start)
39      visited[str(start)]=True
40      fronter=defaultdict(str)
41      while len(explored)!=0:
42          begin=str(explored.pop(0))
43          for i in range(len(dict[begin])):
44              final=dict[begin][i][0]
45              if visited[final]!=True:
46                  fronter[final]=begin
47                  visited[str(final)]=True
48                  explored.append(final)
49                  num_visited+=1
50                  if final==str(end):
51                      explored.clear()
52                      break
```

```
53
54      path=[]
55      dist=0
56      path.append(str(end))
57      while path[0]!=str(start):
58          for i in range(len(dict[fronter[path[0]]])):
59              if dict[fronter[path[0]]][i][0]==path[0]:
60                  dist+=float(dict[fronter[path[0]]][i][1])
61          path.insert(0,fronter[path[0]])
62      for i in range(len(path)):
63          path[i]=int(path[i])
64
65      return path,dist,num_visited
66      raise NotImplementedError("To be implemented")
67      # End your code (Part 1)
68
```

DFS

```
4   def dfs(start, end):
5       # Begin your code (Part 2)
6       """
7           Open edges.csv first,then use defaultdict to construct lists named dict and visited.
8           'dict' is used to save the end node,distance and speed limit of certain start node.
9           'visited' is used to save whether the node is searched or not.
10          Use readlines and split to store every datas,then I construct list named explored,fronter
11          and distance.
12          'explored' is used to save the order of nodes that we should search.
13          'fronter' is used to save the parent of the node.
14          Pop the last data of explored to find out the start node(the node is named 'begin') we use
15          it to find out the end node from dict(the end node is named 'final').
16          If 'final' is not visited,append it to explored and plus one to num_visited.
17          'dist' is the sum of the distance in edges.csv of the number in fronter.
18          Finally,use path to save the nodes that are included in the path,and change it to
19          integer.
20
21      """
22      file=open('edges.csv')
23      dict=defaultdict(list)
24      visited=defaultdict(str)
25      lines=file.readlines()
26
27      for line in lines:
28          detail=line.split(',')
29          data=[]
30          data.append(detail[1])
31          data.append(detail[2])
32          data.append(detail[3].replace("\n",""))
33          dict[detail[0]].append(data)
34          visited[detail[0]]=False
35      num_visited=0
36      explored=[]
37      explored.append(start)
38      visited[str(start)]=True
39      fronter=defaultdict(str)
40      while len(explored)!=0:
41          begin=str(explored.pop())
42          for i in range(len(dict[begin])):
43              final=dict[begin][i][0]
44              if visited[final]!=True:
45                  fronter[final]=begin
46                  visited[str(final)]=1
47                  explored.append(final)
48                  num_visited+=1
49                  if final==str(end):
50                      explored.clear()
51                      break
```

```
    path=[]
    dist=0
    path.append(str(end))
    while path[0]!=str(start):
        for i in range(len(dict[fronter[path[0]]])):
            if dict[fronter[path[0]]][i][0]==path[0]:
                dist+=float(dict[fronter[path[0]]][i][1])
        path.insert(0,fronter[path[0]])
    for i in range(len(path)):
        path[i]=int(path[i])

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

UCS

```
def ucs(start, end):
    # Begin your code (Part 3)
    """
        Open edges.csv first,then use defaultdict to construct lists named dict and visited.
        'dict' is used to save the end node,distance and speed limit of certain start node.
        'visited' is used to save whether the node is searched or not.
        Use readlines and split to store every datas,then I construct list named explored,fronter
        and distance.
        'explored' is used to save the order of nodes that we should search.
        'distance' is used to save the distance from the start node to the end node.
        'fronter' is used to save the parent of the node.
        Set 'minimum' as infinite,then compare minimum to each distace of the data of
        explored,if it is less than minimum,it is changed to become minimum.Use the index of
        this data to find out the start node(the node is named 'begin') we use it to find out the
        end node from dict(the end node is named 'final').
        num_visited pluses one.
        If 'final' is not visited,compare the distance from start to final and the distance
        from start to start node + distance in edges.csv,and change it to the smaller one.
        Finally,use path to save the nodes that are included in the path,andn change it to
        integer.

    """
    file=open('edges.csv')
    dict=defaultdict(list)
    visited=defaultdict(str)
    lines=file.readlines()

    for line in lines:
        detail=line.split(',')
        data=[]
        data.append(detail[1])
        data.append(detail[2])
        data.append(detail[3].replace("\n",""))
        dict[detail[0]].append(data)
        dict[detail[0]].sort(key=lambda s:s[1])
        visited[detail[0]]=False
    num_visited=0
    explored=[]
    explored.append(start)
    visited[str(start)]=True
    fronter=defaultdict(str)
    distance=defaultdict(float)
    distance[start]=0
```

```
    while len(explored)!=0:
        minimum=float("inf")
        index=0
        for i in range(len(explored)):
            if distance[explored[i]]<minimum:
                minimum=distance[explored[i]]
                index=i
        begin=str(explored.pop(index))
        visited[begin]=1
        num_visited+=1
        if begin==str(end):
            explored.clear()
            dist=distance[str(end)]
            break
        for i in range(len(dict[begin])):
            final=dict[begin][i][0]
            if visited[final]!=1:
                if final not in explored:
                    explored.append(final)
                if final in distance:
                    if distance[final]>distance[begin]+float(dict[begin][i][1]):
                        fronter[final]=begin
                        distance[final]=distance[begin]+float(dict[begin][i][1])
                else:
                    fronter[final]=begin
                    distance[final]=distance[begin]+float(dict[begin][i][1])
        distance[index]=float("inf")

    path=[]
    path.append(str(end))
    while path[0]!=str(start):
        path.insert(0,fronter[path[0]])
    for i in range(len(path)):
        path[i]=int(path[i])
    return path,dist,num_visited
    raise NotImplementedError("To be implemented")
    # End your code (Part 3)
```

A STAR

```
def astar(start, end):
    # Begin your code (Part 4)
    """
        Open edges.csv first,then use defaultdict to construct lists named dict and visited.
        'dict' is used to save the end node,distance and speed limit of certain start node.
        'visited' is used to save whether the node is searched or not.
        Open heuristic.csv,then use defaultdict to construct lists named value1,value2 and value3.
        'value1' is used to save the data of fist end node.
        'value2' is used to save the data of second end node.
        'value3' is used to save the data of third end node.
        Use readlines and split to store every datas,then I construct list named explored,fronter,addition
        and distance.
        'explored' is used to save the order of nodes that we should search.
        'distance' is used to save the distance from the start node to the end node.
        'fronter' is used to save the parent of the node.
        'addition' is used to save the sum of distance from start to the current end node.
        Set 'minimum' as infinite,then compare minimum to each distace of the data of
        explored,if it is less than minimum,it is changed to become minimum.Use the index of
        this data to find out the start node(the node is named 'begin') we use it to find out the
        end node from dict(the end node is named 'final').
        num_visited pluses one.
        If 'final' is not visited,compare the distance from start to final and the distance
        from start to start node + distance in edges.csv,and change it to the smaller one.
        The sum of this distance and the data of  'final' in the list named value will be
        stored in addition.
        Finally,use path to save the nodes that are included in the path,andn change it to
        integer.

    """
    file=open('edges.csv')
    dict=defaultdict(list)
    visited=defaultdict(str)
    lines=file.readlines()

    for line in lines:
        detail=line.split(',')
        data=[]
        data.append(detail[1])
        data.append(detail[2])
        data.append(detail[3].replace("\n",""))
        dict[detail[0]].append(data)
        dict[detail[0]].sort(key=lambda s:s[1])
        visited[detail[0]]=False
```

```
    file1=open('heuristic.csv')
    value1=defaultdict(float)
    value2=defaultdict(float)
    value3=defaultdict(float)
    lines1=file.readlines()
    i=0
    for line in lines1:
        if i==0:
            i=1
            continue
        detail=line.split(',')
        value1[detail[0]]=detail[1]
        value2[detail[0]]=detail[2]
        value3[detail[0]]=detail[3].replace("\n","")
    num_visited=0
    explored=[]
    explored.append(start)
    visited[str(start)]=True
    fronter=defaultdict(str)
    distance=defaultdict(float)
    distance[start]=0
    addition=defaultdict(float)
    if str(end)=='1079387396':
        value=value1
    elif str(end)=='1737223506':
        value=value2
    elif str(end)=='8513026827':
        value=value3
    addition[start]=value[start]
    while len(explored)!=0:
        minimum=float("inf")
        index=0
        for i in range(len(explored)):
            if addition[explored[i]]<minimum:
                minimum=addition[explored[i]]
                index=i
        begin=str(explored.pop(index))
        visited[begin]=1
        num_visited+=1
        if begin==str(end):
            explored.clear()
            dist=addition[str(end)]
            break
        for i in range(len(dict[begin])):
            final=dict[begin][i][0]
            if visited[final]!=1:
                if final not in explored:
                    explored.append(final)
                if final in distance:
                    if distance[final]>distance[begin]+float(dict[begin][i][1]):
                        fronter[final]=begin
                        distance[final]=distance[begin]+float(dict[begin][i][1])
                else:
                    fronter[final]=begin
                    distance[final]=distance[begin]+float(dict[begin][i][1])
                addition[final]=distance[final]+float(value[int(final)])
        distance[index]=float("inf")
        addition[index]=float("inf")

    path=[]
    path.append(str(end))
    while path[0]!=str(start):
        path.insert(0,fronter[path[0]])
    for i in range(len(path)):
        path[i]=int(path[i])
    return path,dist,num_visited


    raise NotImplementedError("To be implemented")
    # End your code (Part 4)
```

Part II. Results & Analysis (12%):

- **Please screenshot the results. For instance,**

  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: 4978.881999999998 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: 4367.881 m
The number of visited nodes in UCS: 5086
```

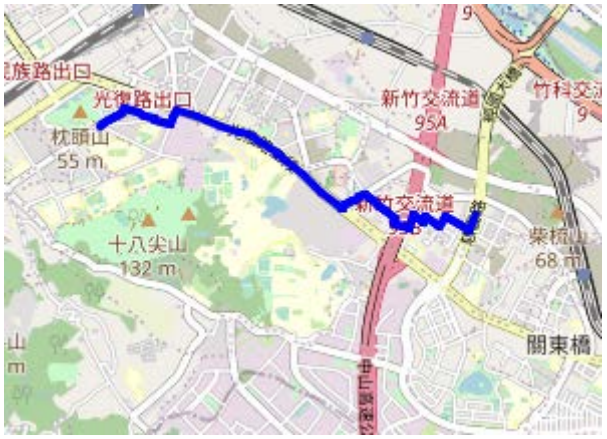
A STAR:

```
The number of nodes in the path found by A* search: 89
Total distance of path found by A* search: 4367.881 m
The number of visited nodes in A* search: 5086
```

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:

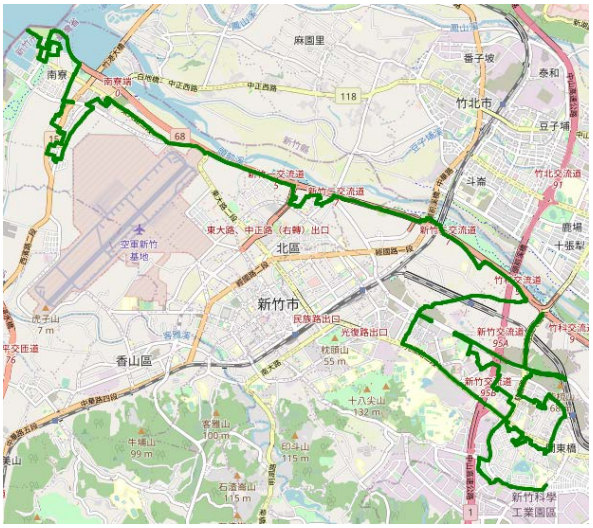
```
The number of nodes in the path found by A* search: 63
Total distance of path found by A* search: 4101.84 m
The number of visited nodes in A* search: 7213
```

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
```

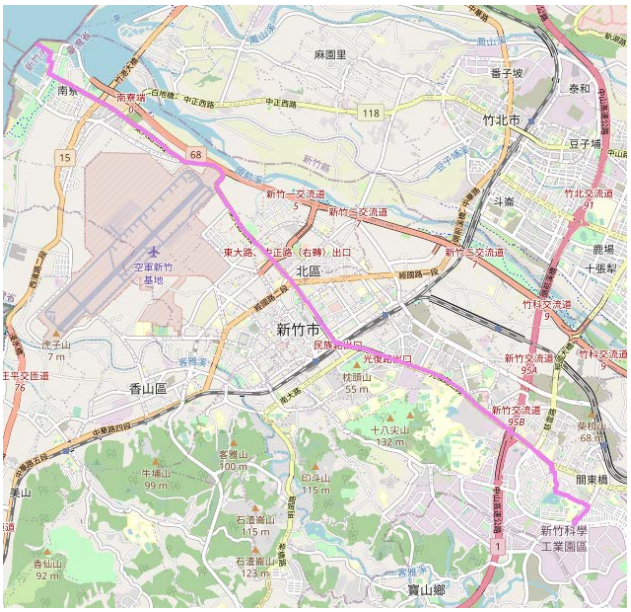
DFS (stack):

```
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: 14212.412999999997 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: 14212.412999999997 m
The number of visited nodes in A* search: 11926
```

The number of nodes in the path found by DFS is always the largest , and the number of nodes in the path found by UCS and A* are the same.
Total distance of path found by DFS is always the largest , and total distance of path found by UCS and A* are the same.
The number of visited nodes in DFS is always the largest , and The number of visited nodes in UCS and A* are the same.

Part III. Question Answering (12%):

1. Please describe a problem you encountered and how you solved it.
   A : First , it is so hard to figure out how to read edge.csv , I even don't know what "defaultdict" is. Also , the concept of USC and A* is confusing. Thanks to my classmate and GOOGLE , I did spent much time on confirming those algorithms' meaning to make the process of going this homework smoothly. At the beginning , the graphs did not show lines , that problem cost me lot of time to solve. The reason is "path" was saved as a string list , it should be an integer list.
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 : Some roads are limited by the type of vehicle , so we should decide the road not only depend on speed limit and distance , but also the vehicle users choose.

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 : the process of concurrently building up a map of the environment and using this map to obtain improved estimates of the location of the vehicle. We can define the interfaces for the components, provide implementations, and connect them in a data flow or dependency graph. The final implementation presented supports, in theory, particle filter localization and can benefit from automated parallelization.

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 : We propose a restricted dynamic programming heuristic for constructing the vehicle routes, and an efficient heuristic for optimizing the vehicle's departure times for each (partial) vehicle route, such that the complete solution algorithm runs in polynomial time. we apply it to the extended node set concerned with the giant-tour representation of vehicle routing solutions. When a state is expanded with a vehicle route-end node, then the associated vehicle route is completed. In the next stage, we consider the route-start node of the successive vehicle as the only feasible expansion, such that a new vehicle route is started. In order to obtain feasible vehicle routes, we add state dimensions that indicate. When we expand a state, we perform feasibility checks to ensure that vehicle capacities are not exceeded.