

Homework 2: Route Finding

Part I. Implementation (6%):

● Please screenshot your code snippets of Part 1 ~ Part 4, and explain your implementation.

Part 1:

```
5  def bfs(start, end):
6      # Begin your code (Part 1)
7
8      #load edgeFile to adjacency list
9      adj_list = {}
10     with open(edgeFile) as f:
11         csvreader = csv.reader(f)
12         next(csvreader) #first row
13         for row in csvreader:
14             tmp = []
15             if int(row[0]) in adj_list: #node exist in adjacency list
16                 tmp.extend(adj_list[int(row[0])]) #row[0]:start_node
17             tmp.append([int(row[1]), float(row[2]), float(row[3])])
18             #row[1]:end_node #row[2]:distance #row[3]:speed limit
19             adj_list[int(row[0])] = tmp #adjacency list
20
21     #bfs
22     queue = Queue()
23     visited = []
24     parent = {}
25     queue.put((0, start)) #(distance from start to node, node)
26     visited.append(start)
27     parent[start] = None
28     path_found = False
29     while not queue.empty():
30         curr = queue.get() #curr[0]:distance curr[1]:current node
31         if curr[1] == end: #current node is end
32             path_found = True
33             break
34         if curr[1] in adj_list: #current node exist
35             for next_node in adj_list[curr[1]]: #next_node[0]:node #next_node[1]:distance
36                 if next_node[0] not in visited:
37                     queue.put((curr[0]+next_node[1], next_node[0]))
38                     #put distance from start to next_node, next_node to queue
39                     parent[next_node[0]] = curr[1] #parent of next_node
40                     visited.append(next_node[0])
41
42     #result
43     path = [] #start to end
44     dist = curr[0] #distance from start to end
45     num_visited = len(visited)
46     if path_found: #store path
47         path.append(end) #end - start
48         while parent[end] is not None:
49             path.append(parent[end])
50             end = parent[end]
51         path.reverse() #reverse path to get start to end
52     return path, dist, num_visited
53     # load edgeFile to get adjacency list (start_node: end_node, distance, speed limit)
54     # use bfs to find path, start to end distance, number visited
55     # get path from parent list
56     #raise NotImplementedError("To be implemented")
57     # End your code (Part 1)
58
```

Load egdeFile in every part is same.

Part 2:

```
3
4 def dfs(start, end):
5     # Begin your code (Part 2)
6     #load edgeFile to adj_list
7     adj_list = {}
8     with open(edgeFile) as f:
9         csvreader = csv.reader(f)
10        next(csvreader) #first row
11        for row in csvreader:
12            tmp = []
13            if int(row[0]) in adj_list:#node exist in adjacency list
14                tmp.extend(adj_list[int(row[0])]) #row[0]:start_node
15                tmp.append([int(row[1]), float(row[2]), float(row[3])])
16                #row[1]:end #row[2]:distance #row[3]:speed limit
17                adj_list[int(row[0])] = tmp #adjacency list
18
19    #dfs_stack
20    stack = []
21    visited = []
22    parent = {}
23    stack.append((0, start)) #(distance from start to node, node)
24    visited.append(start)
25    parent[start] = None
26    path_found = False
27    while stack:
28        curr = stack.pop() #curr[0]:distance curr[1]:current node
29        if curr[1] == end: #current node is end
30            path_found = True
31            break
32
33        if curr[1] in adj_list: #current node exist
34            for next_node in adj_list[curr[1]]: #next_node[0]:node #next_node[1]:distance
35                if next_node[0] not in visited:
36                    stack.append((curr[0]+next_node[1], next_node[0]))
37                    parent[next_node[0]] = curr[1] #parent of next_node
38                    visited.append(next_node[0])
39
40    #result
41    path = [] #start to end
42    dist = curr[0] #distance from start to end
43    num_visited = len(visited)
44    if path_found: #store path
45        path.append(end) #end - start
46        while parent[end] is not None:
47            path.append(parent[end])
48            end = parent[end]
49        path.reverse() #reverse path to get start to end
50    return path, dist, num_visited
51
52 # load edgeFile to get adjacency list (start_node: end_node, distance, speed limit)
53 # use dfs to find path, start to end distance, number visited
54 # get path from parent list
55 #raise NotImplementedError("To be implemented")
56 # End your code (Part 2)
```

Part 3:

```
5 def ucs(start, end):
6     # Begin your code (Part 3)
7     #load edgeFile to adj_list
8     adj_list = {}
9     with open(edgeFile) as f:
10         csvreader = csv.reader(f)
11         next(csvreader) #first row
12         for row in csvreader:
13             tmp = []
14             if int(row[0]) in adj_list: #node exist in adjacency list
15                 tmp.extend(adj_list[int(row[0])]) #row[0]:start_node
16             tmp.append([int(row[1]), float(row[2]), float(row[3])])
17             #row[1]:end_node #row[2]:distance #row[3]:speed limit
18             adj_list[int(row[0])] = tmp #adjacency list
19
20     #ucs
21     queue = PriorityQueue() #use priority queue to priority smallest distance
22     visited = []
23     parent = {}
24     distance = {}
25     queue.put((0, start)) #(distance from start to node, node)
26     distance[start] = 0
27     visited.append(start)
28     parent[start] = None
29     path_found = False
30     while not queue.empty():
31         curr = queue.get() #curr[0]:distance curr[1]:current node
32
33         if curr[1] == end: #current node is end
34             path_found = True
35             break
36         if curr[1] in adj_list:
37             for next_node in adj_list[curr[1]]:
38                 #none_visited or has smaller than distance
39                 if next_node[0] not in visited or curr[0]+next_node[1] < distance[next_node[0]]:
40                     queue.put((curr[0]+next_node[1], next_node[0]))
41                     #calculator start to next_node distance
42                     distance[next_node[0]] = distance[curr[1]] + next_node[1]
43                     parent[next_node[0]] = curr[1] #parent of next_node
44                     if next_node[0] not in visited:
45                         visited.append(next_node[0])
46
47     #result
48     path = [] #start to end
49     dist = curr[0] #distance from start to end
50     num_visited = len(visited)
51     if path_found: #store path
52         path.append(end) #end - start
53         while parent[end] is not None:
54             path.append(parent[end])
55             end = parent[end]
56         path.reverse() #reverse path to get start to end
57     return path, dist, num_visited
58
59 # load edgeFile to get adjacency list (start_node: end_node, distance, speed limit)
60 # use ucs to find path, start to end smallest distance, number visited
61 # queue will be priority smallest distance from start to node
62 # get path from parent list
63 #raise NotImplementedError("To be implemented")
64 # End your code (Part 3)
```

Part 4:

```
5
6 def astar(start, end):
7     # Begin your code (Part 4)
8     #load edgeFile to adj_list
9     adj_list = {}
10    with open(edgeFile) as f:
11        csvreader = csv.reader(f)
12        next(csvreader) #first row
13        for row in csvreader:
14            tmp = []
15            if int(row[0]) in adj_list: #node exist in adjacency list
16                tmp.extend(adj_list[int(row[0])]) #row[0]:start_node
17            tmp.append([int(row[1]), float(row[2]), float(row[3])])
18            #row[1]:end_node #row[2]:distance #row[3]:speed limit
19            adj_list[int(row[0])] = tmp #adjacency list
20
21    #load heuristicFile
22    with open(heuristicFile) as f:
23        csvreader = csv.reader(f)
24        ids = next(csvreader) #end node id
25        for index in range(1, len(ids)): #get end column index
26            if (int(ids[index]) == end): #column is end node
27                break
28        heu = {}
29        for row in csvreader:
30            heu[int(row[0])] = float(row[index]) #heuristic list
31
32    #Astar
33    queue = PriorityQueue() #use priority queue to priority smallest distance
34    visited = []
35    parent = {}
```

```
31
32    #Astar
33    queue = PriorityQueue() #use priority queue to priority smallest distance
34    visited = []
35    parent = {}
36    distance = {} #distance from start to node
37    f = {}
38    queue.put((0, start)) #(start to node + node to end distance, node)
39    distance[start] = 0
40    f[start] = 0
41    parent[start] = None
42    visited.append(start)
43    path_found = None
44    while not queue.empty():
45        curr = queue.get()
46        if curr[1] == end:
47            path_found = True
48            break
49        if curr[1] in adj_list:
50            for next_node in adj_list[curr[1]]:
51                #none_visited or has smaller than f
52                if next_node[0] not in visited or heu[next_node[0]]+next_node[1] + distance[curr[1]] < f[next_node[0]]:
53                    queue.put((heu[next_node[0]]+next_node[1] + distance[curr[1]], next_node[0]))
54                    f[next_node[0]] = heu[next_node[0]]+next_node[1] + distance[curr[1]]
55                    # calculator total start to node and node to end distance
56                    distance[next_node[0]] = distance[curr[1]] + next_node[1] #distance start to next_node
57                    parent[next_node[0]] = curr[1] #next_node of parent
58                    if next_node[0] not in visited:
59                        visited.append(next_node[0])
60
61    #result
62    path = []
63    dist = distance[end]
64    num_visited = len(visited)
65
66    if path_found: #store path
67        path.append(end) #end - start
68        while parent[end] is not None:
69            path.append(parent[end])
70            end = parent[end]
71        path.reverse() #reverse path to get start to end
72    return path, dist, num_visited
73
74    # load edgeFile to get adjacency list (start_node: end_node, distance, speed limit)
75    # load heuristicFile to get heuristic list of node to end distance (node: distance)
76    # use a* to find path, start to end smallest distance, number visited
77    # queue will be priority smallest f
78    # f = distance + heuristic
79    # get path from parent list
80    #raise NotImplementedError("To be implemented")
81    # End your code (Part 4)
```

Load heuristicFile in Part 4, Part 5 is same.

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: 4978.8820000000005 m

The number of visited nodes in BFS: 4403



DFS (STACK):

The number of nodes in the path found by DFS: 1718

Total distance of path found by DFS: 75504.314999999983 m

The number of visited nodes in DFS: 5236



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: 5158

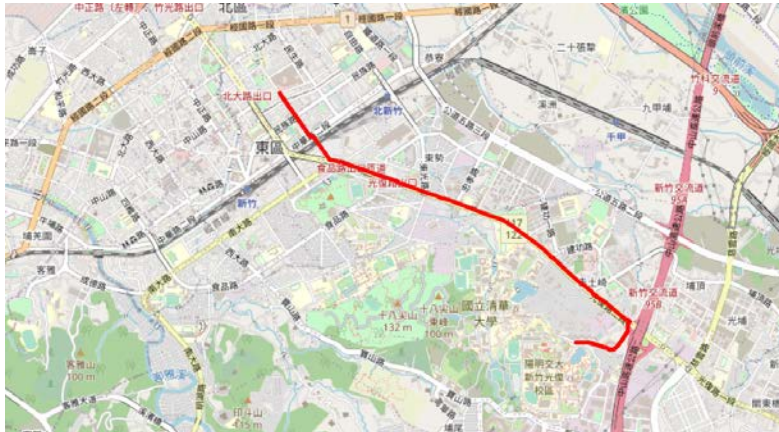


Astar:

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: 312



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: 4215.521 m

The number of visited nodes in BFS: 4752



DFS (STACK):

The number of nodes in the path found by DFS: 930

Total distance of path found by DFS: 38752.307999999996 m

The number of visited nodes in DFS: 9616



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: 7318



Astar:

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: 1228



Test3: from National Experimental High School At Hsinchu Science Park (ID: **1718165260**) to Nanliao Fighting 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: 11266



DFS (STACK):

The number of nodes in the path found by DFS: 900

Total distance of path found by DFS: 39219.992999999996 m

The number of visited nodes in DFS: 2494



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: 11934



Astar:

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: 7247



● Analysis

BFS has smallest number of nodes in the path but total distance of path and number of visited nodes are larger.

DFS has largest number of nodes in the path, total distance of path and number of visited nodes.

UCS and A* search have same smallest number of nodes in the path and total distance of path. But A* number of visited nodes smaller than UCS.

Number of nodes in the path: $BFS > A^* = UCS > DFS$

Total distance of path: $A^* = UCS > BFS > DFS$

Number of visited nodes: $A^* > UCS > BFS > DFS$

So A* search is best choice to find the shortest path.

Part III. Question Answering (12%):

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

I had error result in test 2 and test 3, I found out I didn't check from the current node to visited nodes that had shorter distance. In addition to unvisited nodes, I have added a distance test condition.

I don't know how to use JUPYTER NOTEBOOK, I use google search to find out about it.

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.

Traffic lights, traffic jam, pedestrians, blocked or unworkable roads...
Traffic lights, traffic jam, passersby will affect speed and increase waiting time at each stop, or blocked or unworkable roads will affect the route.

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

Address

Coordinates

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.

Heuristic function will be time from shipper to seller and time from seller to buyer. Will set to 2 start nodes and 2 end nodes. First is shipper to seller location, second is seller location to buyer location. This will ensure the delivery and delivery process of the shipper.