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%):

- **Part1**

```python
def bfs(start, end):
    # Begin your code (Part 1)
    '''
    1. Construct adjlist with dictionary(start:[end, distance])
    2. Running bfs with queue
       While len(queue)>0
       Pop out the first node
       Check whether adjancent nodes are visited or not
       if not visit: mark it as visited node and add it to queue:list, visited:set
       if find end: break from the loop
    3. Backtracking for getting route and distance from start to end
    '''
    adjlist = {}
    dist = {}
    with open(edgeFile, newline='') as csvfile:
        rows = csv.reader(csvfile)
        headings = next(rows)
        for row in rows:
            temp_start = int(row[0])
            temp_end = int(row[1])
            temp_dist = float(row[2])
            if temp_start in adjlist:
                adjlist[temp_start].append([temp_end, temp_dist])
            else:
                adjlist[temp_start] = [[temp_end, temp_dist]]
    queue = []
    parent = {}
    visited = set()
    queue.append(start)
    visited.add(start)
    find = False
    while (len(queue)>0):
        temp_node = queue.pop(0)
        if adjlist.get(temp_node) == None:
            continue
        else:
            for neighbor in adjlist[temp_node]:
                if neighbor[0] not in visited :
                    queue.append(neighbor[0])
                    visited.add(neighbor[0])
                    parent[neighbor[0]] = [temp_node, neighbor[1]]
                    if neighbor[0] == end:
                        find = True
                        break
        if find: break
    path = [end]
    dist = 0.0
    while path[-1] != start:
        dist += parent[path[-1]][1]
        path.append(parent[path[-1]][0])
    path.reverse()

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

- **Part2**

```python
def dfs(start, end):
    # Begin your code (Part 2)
    '''
    1. Construct adjlist with dictionary(start:[end, distance])
    2. Running bfs with stack:list
       While len(queue)>0
       Pop out the last node
       Check whether adjancent nodes are visited or not
       if not visit: mark it as visited node and add it to stack:list, visited:set
       if find end: break from the loop
    3. Backtracking for getting route and distance from start to end
    '''
    adjlist = {}
    dist = {}
    with open(edgeFile, newline='') as csvfile:
        rows = csv.reader(csvfile)
        headings = next(rows)
        for row in rows:
            temp_start = int(row[0])
            temp_end = int(row[1])
            temp_dist = float(row[2])
            if temp_start in adjlist:
                adjlist[temp_start].append([temp_end, temp_dist])
            else:
                adjlist[temp_start] = [[temp_end, temp_dist]]
    stack = []
    parent = {}
    visited = set()
    stack.append(start)
    visited.add(start)
    find = False
    while (len(stack)>0):
        temp_node = stack.pop()
        if adjlist.get(temp_node) == None:
            continue
        else:
            for neighbor in adjlist[temp_node]:
                if neighbor[0] not in visited :
                    stack.append(neighbor[0])
                    visited.add(neighbor[0])
                    # if neighbor in parent:
                    #     parent[neighbor[0]].append ([temp_node, neighbor[1]])
                    parent[neighbor[0]] = [temp_node, neighbor[1]]
                    if neighbor[0] == end:
                        find = True
                        break
        if find: break

    path = [end]
    dist = 0.0
    while path[-1] != start:
        dist += parent[path[-1]][1]
        path.append(parent[path[-1]][0])
    path.reverse()

    return path, dist, len(visited)-1
    #raise NotImplementedError("To be implemented")
    # End your code (Part 2)
```

- **Part3**

```python
def ucs(start, end):
    # Begin your code (Part 3)
    '''
    1. Construct adjlist with dictionary(start:{end:distance})
    2. Running ucs with q = queue.priorityQueue()
       While len(q)>0
       Pop out the first node with smallest distance value
       if find end: break from the loop
       Calculate newdist with mindist:dict and adjlist
       if newdist < mindist: update mindist, record parent node, and but it into q again
    3. Backtracking for getting route from start to end
    '''

    adjlist = {}
    mindist = {}
    file = open(edgeFile)
    csvreader = csv.reader(file)
    header = next(csvreader)
    for row in csvreader:
        temp_start = int(row[0])
        temp_end = int(row[1])
        temp_dist = float(row[2])
        if temp_start not in adjlist:
            adjlist[temp_start] = {}
        adjlist[temp_start][temp_end] = temp_dist
        if temp_start not in mindist:
            mindist[temp_start] = float("inf")

    q = queue.PriorityQueue()
    visited = []
    parent = {}
```

```python
    mindist[start] = 0
    visited.append(start)
    q.put([mindist[start], start])
    while not q.empty():
        cur_dist, current = q.get()
        if current == end:
            break
        for neighbor in adjlist[current]:
            newdist = mindist[current] + adjlist[current][neighbor]
            if neighbor in mindist:
                if newdist < mindist[neighbor]:
                    mindist[neighbor] = newdist
                    parent[neighbor] = current
                    q.put([mindist[neighbor], neighbor])
                    # if neighbor not in visited:
                    visited.append(neighbor)

    path = [end]
    while path[-1] != start:
        current = path[-1]
        path.append(parent[current])

    return path, mindist[end], len(visited)
    #raise NotImplementedError("To be implemented")
    # End your code (Part 3)
```

- **Part4**

```python
def astar(start, end):
    # Begin your code (Part 4)
    '''
    1. Construct adjlist with dictionary(start:{end:distance})
    2. Running ucs with q = queue.priorityQueue()
       While len(q)>0
       Pop out the first node with smallest distance value
       if find end: break from the loop
       Calculate newdist with mindist:dict and herusticlist:dict(dict)
       if newdist < mindist: update mindist, record parent node, and but it into q again
    3. Backtracking for getting route from start to end
    '''
    adjlist = {}
    mindist = {}
    file = open(edgeFile)
    csvreader = csv.reader(file)
    header = next(csvreader)
    for row in csvreader:
        temp_start = int(row[0])
        temp_end = int(row[1])
        temp_dist = float(row[2])
        if temp_start not in adjlist:
            adjlist[temp_start] = {}
        adjlist[temp_start][temp_end] = temp_dist
        if temp_start not in mindist:
            mindist[temp_start] = float("inf")
    heuristiclist = {}
    file = open(heuristicFile)
    csvreader = csv.reader(file)
    header = next(csvreader)
    for row in csvreader:
        temp_node = int(row[0])
        heuristiclist[temp_node] = {}
        heuristiclist[temp_node][int(header[1])] = float(row[1])
        heuristiclist[temp_node][int(header[2])] = float(row[2])
        heuristiclist[temp_node][int(header[3])] = float(row[3])
```

```python
    q = queue.PriorityQueue()
    visited = []
    parent = {}
    mindist[start] = 0
    visited.append(start)
    q.put([mindist[start], start])
    while not q.empty():
        cur_dist, current = q.get()
        if current == end:
            break
        for neighbor in adjlist[current]:
            newdist = mindist[current] + adjlist[current][neighbor]
            if neighbor in mindist:
                if newdist < mindist[neighbor]:
                    mindist[neighbor] = newdist
                    priority = mindist[neighbor] + heuristiclist[neighbor][end]
                    parent[neighbor] = current
                    q.put([priority, neighbor])
                    visited.append(neighbor)

    path = [end]
    while path[-1] != start:
        current = path[-1]
        path.append(parent[current])

    return path, mindist[end], len(visited)
    #raise NotImplementedError("To be implemented")
    # End your code (Part 4)
```

- **Part6(Bouns)**

  For this part, I change the value I stored in adjlist from distance to time(distance/speed_limit). Heuristiclist is also changed to distance/max_speed_limit. Then I can route the path by time in this part.

```python
def astar_time(start, end):
    # Begin your code (Part 6)
    '''
    1. Construct adjlist with dictionary(start:{end:time})
    2. Running ucs with q = queue.priorityQueue()
       While len(q)>0
       Pop out the first node with smallest travel time
       if find end: break from the loop
       Calculate newtime with mintime:dict and herusticlist:dict(dict)
       if newtime < mintime: update mintime, record parent node, and but it into q again
    3. Backtracking for getting route from start to end
    '''
    adjlist = {}
    mintime = {}
    speed = []
    file = open(edgeFile)
    csvreader = csv.reader(file)
    header = next(csvreader)
    for row in csvreader:
        temp_start = int(row[0])
        temp_end = int(row[1])
        temp_speed = (float(row[3])/3.6)
        speed.append(temp_speed)
        temp_time = float(row[2])/temp_speed
        if temp_start not in adjlist:
            adjlist[temp_start] = {}
        adjlist[temp_start][temp_end] = temp_time
        if temp_start not in mintime:
            mintime[temp_start] = float("inf")
    heuristiclist = {}
    file = open(heuristicFile)
    csvreader = csv.reader(file)
    header = next(csvreader)
    max_speed = max(speed)
    for row in csvreader:
        temp_node = int(row[0])
        heuristiclist[temp_node] = {}
        heuristiclist[temp_node][int(header[1])] = float(row[1])/max_speed
        heuristiclist[temp_node][int(header[2])] = float(row[2])/max_speed
        heuristiclist[temp_node][int(header[3])] = float(row[3])/max_speed

    q = queue.PriorityQueue()
    visited = []
    parent = {}
    mintime[start] = 0
    visited.append(start)
    q.put([mintime[start], start])
    while not q.empty():
        cur_dist, current = q.get()
        if current == end:
            break
        for neighbor in adjlist[current]:
            newtime = mintime[current] + adjlist[current][neighbor]
            if neighbor in mintime:
                if newtime < mintime[neighbor]:
                    mintime[neighbor] = newtime
                    priority = mintime[neighbor] + heuristiclist[neighbor][end]
                    parent[neighbor] = current
                    q.put([priority, neighbor])
                    visited.append(neighbor)
    path = [end]
    while path[-1] != start:
        current = path[-1]
        path.append(parent[current])
    return path, mintime[end], len(visited)
    # End your code (Part 6)
```

Part II. Results & Analysis (12%):

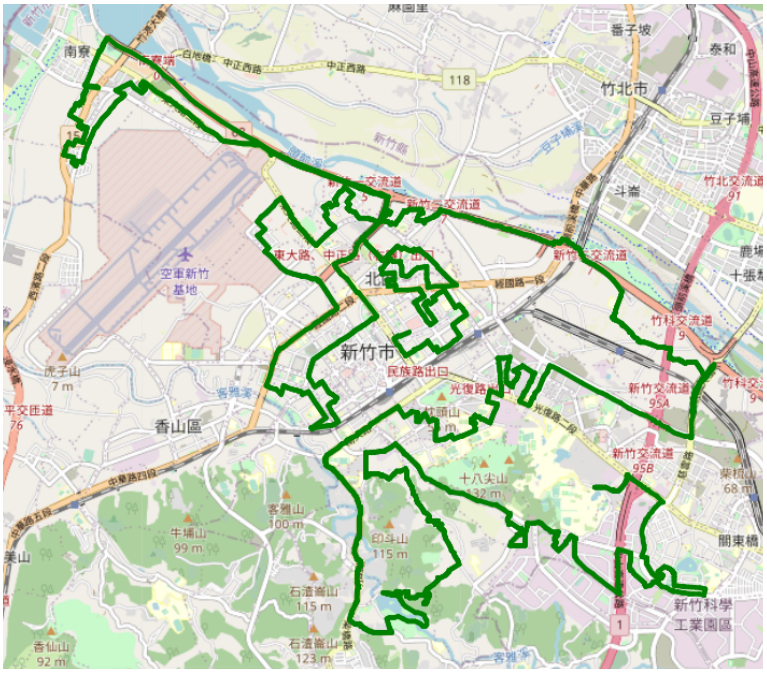
- **Test 1:**
  from National Yang Ming Chiao Tung University (ID: 2270143902)
  to Big City Shopping Mall (ID: 1079387396)

  - **BFS**

```
The number of nodes in the path found by BFS: 88
Total distance of path found by BFS: 4978.881999999998 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: 5298
```

- **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: 318
```

- **A*(time)**

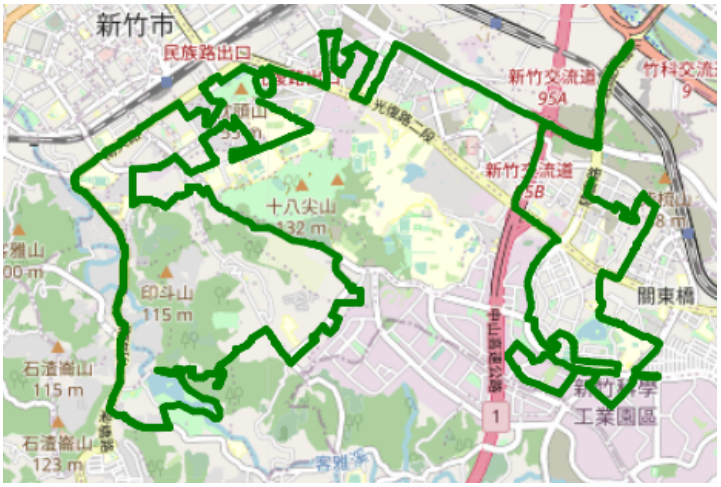
```
The number of nodes in the path found by A* search: 89
Total second of path found by A* search: 320.87823163083164 s
The number of visited nodes in A* search: 2089
```

- **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: 4215.521000000001 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: 7560
```

- **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: 1308
```

- **A*(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: 3054
```

- **Test 3:**
  from National Experimental High School At Hsinchu Science Park (ID: 1718165260)
  to Nanliao Fishing 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)**

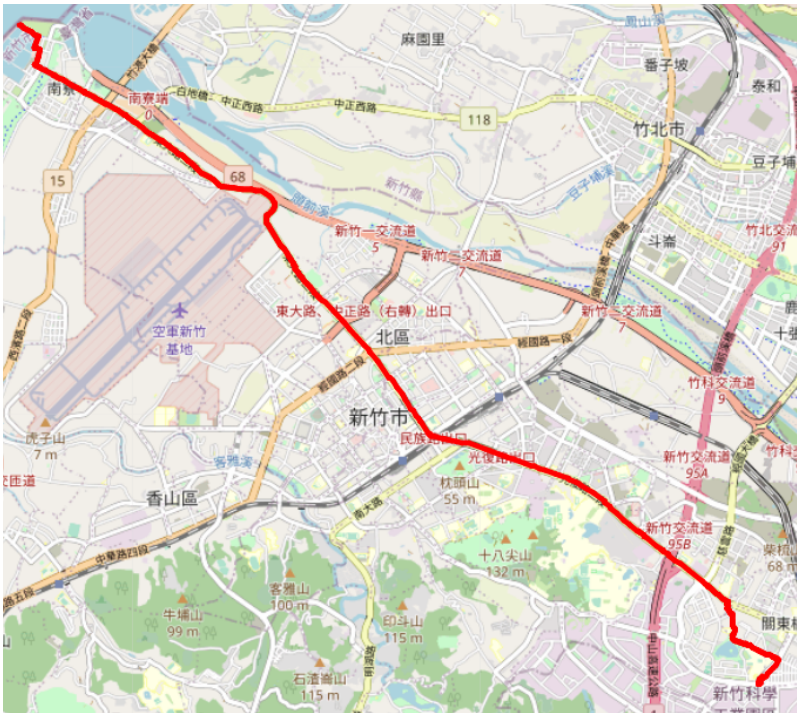
```
The number of nodes in the path found by DFS: 900
Total distance of path found by DFS: 39219.993000000024 m
The number of visited nodes in DFS: 2493
```

- **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: 12303
```

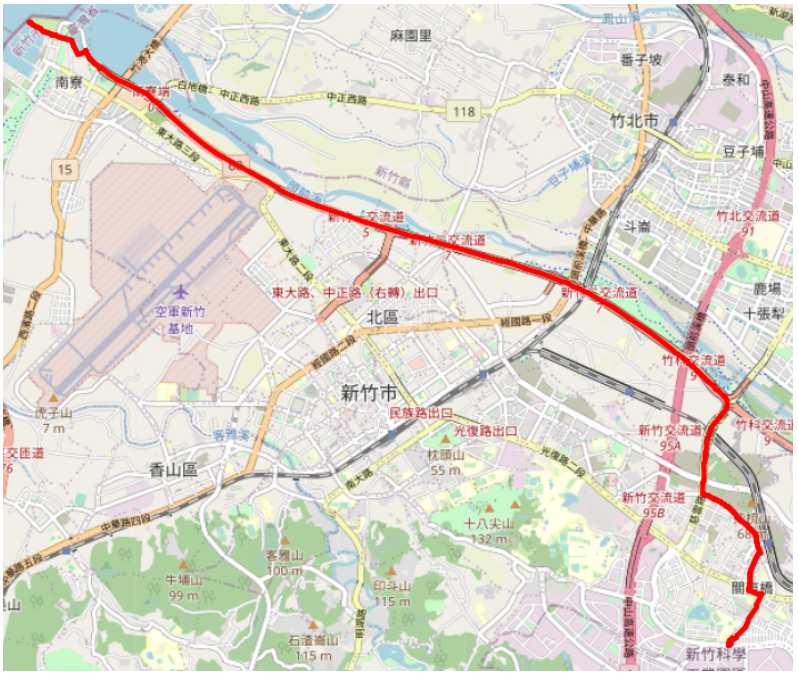
- **A***

```
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: 7766
```



- **A*(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: 8898
```



Part III. Question Answering (12%):

1. **<u>Please describe a problem you encountered and how you solved it.</u>**
   The biggest problem i have encountered is that I am not familiar with UCS algorithm and A* algorithm. Therefore, I spent a lot of time studying it by online resources and course slides and finally I figured it out.

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.**
   Real-time traffic status is also essential for route finding. If the navigation system calculates the shortest path between two spots, but a heavy traffic jam occurs on the path, It is definitely not the best route for drivers.

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

   Global Positioning System(GPS) might be a possible solution. It provides real time and high accuracy information for mapping and localization.

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.**

   We apply real-time traffic status into the heuristic function. Since estimated time of arrival changes over time, traffic status is an important factor that affects it. So I will define the dynamic heuristic function as follow:
   mintime = min(min[a][b]+real_time_traffic[a][b])