HW-2 report(0616076 朱彥勳)

Part 1:

1. Read “edges.csv”

```
sample_text = open("edges.csv")
txt = sample_text.readlines()
start_node=[]
end_node=[]
cost = []
for i in range(1,len(txt)):
  a, b, c, d = txt[i].split(',')
  start_node.append(a)
  end_node.append(b)
  cost.append(c)
```

2. Put in “total states”, including all start nodes and end nodes.

```
temp1 = set(start_node)
temp2 = set(end_node)
total_states = temp1.union(temp2)
print("Total states= ",total_states)
total_states = list(total_states)
```

3. Set up a dictionary, if an index called ‘A’: [‘B’, ‘C’], it means a path from A to B and a path from A to C are exited.

```
values = dict()
for i in total_states:
  values[i] = []
for i in total_states:
  for j in range(0, len(start_node)):
    if i == start_node[j]:
      values[i].append(end_node[j])
print(values)
```

4. Define bfs(start, end).

   Visited are the neighbors of all the pop elements from Queue.

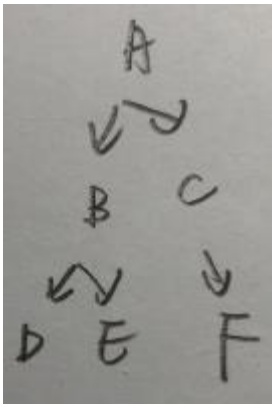
   Queue includes “start” and neighbors not in Visited.

   Bfs_visited are all the pop elements from Queue.

```
def bfs(start, end):
  visited.append(start)
  Bvisited.append(-1)
  queue.append(start)
  bfs_path = []
  bfs_dist = 0
  bfs_visited = []

  while queue:
    s = queue.pop(0)
    bfs_visited.append(s)
    #print(bfs_path)
    if s == end:
      break
    for neighbour in values[s]:
      #print(neighbour)
      if neighbour not in visited:
        visited.append(neighbour)
        Bvisited.append(s)
        queue.append(neighbour)
```

5. Define Bfs_path



   Bvisited should be used to record the previous elements of Visited.

   Assume that a Bfs_path is A->B->E and the Bfs_visited is A->B->C->D->E,

   I record -1->A->A->B->B in Bvisited. From the last element to the front, when I see B in Bvisited, I ignore all the element in Bfs_visited until seeing a B in it. Then, Bfs_path should be like B->A->-1. After that, I reverse Bfs_path, add the last element E in it, and pop the first element -1. Finally, Bfs_path should be like A->B->E.

```
t = 1
for i in range(len(bfs_visited)-1, -1, -1):
  #print(bfs_visited[i])
  for j in range(len(visited)-1, -1, -1):
    if t == 1 and bfs_visited[i] == visited[j]:
      bfs_path.append(int(Bvisited[i]))
      t = Bvisited[i]
    elif bfs_visited[i] == visited[j] and bfs_visited[i] == t :
      bfs_path.append(int(Bvisited[i]))
      t = Bvisited[i]
bfs_path.reverse()
bfs_path.append(int(end))
bfs_path.pop(0)
```

6. Find Bfs_dist.

   If start_node[i] = bfs_path[j] and end_node[i] = bfs_path[j+1], add the cost into the Bfs_dist. Then, return all the value.

```
for j in range(0, len(bfs_path)-1):
  for i in range(1,len(txt)-1):
    if int(start_node[i]) == int(bfs_path[j]) and int(end_node[i]) == int(bfs_path[j+1]):
      bfs_dist += float(cost[i])

      #print(i, j, bfs_dist)
return bfs_path, bfs_dist, len(bfs_visited)
```

Result:

```
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: 4274
```

Part 2:

What I need to do is same as bfs but change queue to stack.

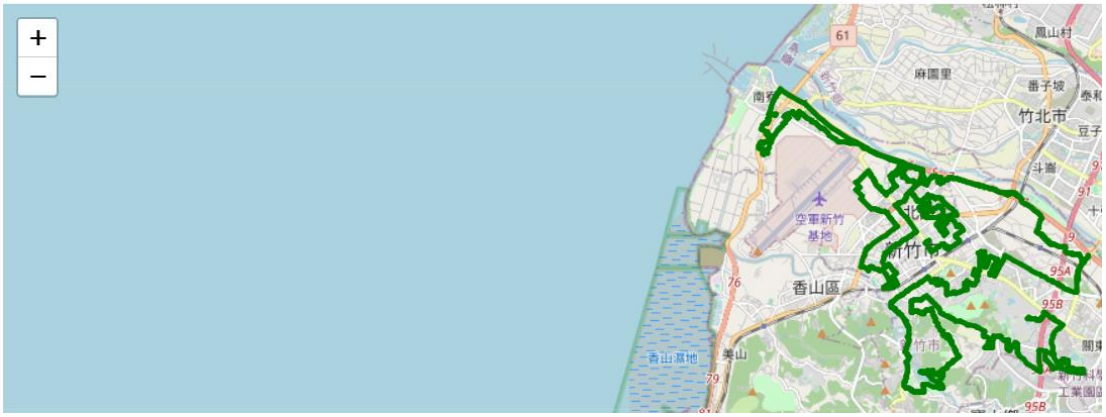
```
dvisited = [] # List to keep track of visited nodes.
dBvisited = []
dstack = []
def dfs(start, end):
 start = str(start)
 end = str(end)
 dvisited.append(start)
 dBvisited.append(-1)
 dstack.append(start)
 dfs_path = []
 dfs_dist = 0
 dfs_visited = []
 while dstack:
  s = dstack.pop()
  dfs_visited.append(s)
  #print(bfs_path)
  if s == end:
   break
  for neighbour in values[s]:
   #print(neighbour)
   if neighbour not in dvisited:
    dvisited.append(neighbour)
    dBvisited.append(s)
    dstack.append(neighbour)
 #print("visited:", visited)
 #print("Bvisited:", Bvisited)
 #print("bfs_visited:", bfs_visited)
 t = 1
 for i in range(len(dfs_visited)-1, -1, -1):
  #print(bfs_visited[i])
  for j in range(len(dvisited)-1, -1, -1):
   if t == 1 and dfs_visited[i] == dvisited[j]:
    dfs_path.append(int(dBvisited[j]))
    t = dBvisited[j]
   elif dfs_visited[i] == dvisited[j] and dfs_visited[i] == t :
    dfs_path.append(int(dBvisited[j]))
    t = dBvisited[j]

 #print("bfs_path:", bfs_path)
 dfs_path.reverse()
 dfs_path.append(end)
 dfs_path.pop(0)
 for j in range(0, len(dfs_path)-1):
  for i in range(1,len(txt)-1):
   if int(start_node[i]) == int(dfs_path[j]) and int(end_node[i]) == int(dfs_path[j+1]):
    dfs_dist += float(cost[i])


    #print(i, j, bfs_dist)
 return dfs_path, dfs_dist, len(dfs_visited)
```

Result:

The number of nodes in the path found by DFS: 1718
Total distance of path found by DFS: 75504.31499999983 m
The number of visited nodes in DFS: 4712



Part3:

Usc just like bfs, we need to use queue. However, the difference between usc and bfs is the concept of “cost”. Using bfs, we do not need to calculate the total distance when we draw the route from one location to another; using usc, we need to consider “distance”. In other words, distance will influence what we draw.

In usc, I use a priority queue instead of queue because of the different distance. In addition, I print the total distance and all the location I use in every iteration. For example, 23 (64.31, ['2270143902', '4421728047', '3226037491', '2903276023', '3226037491']). 23 is the number of iterations; 64.31 is the total distance; and the last five numbers is the five locations I use in this iteration.

```
def ucs(start, end):
  start = str(start)
  end = str(end)
  usc_path = []
  usc_dist = 0
  usc_visited = []
  ttt = 0
  queue = Q.PriorityQueue()
  queue.put((0, [start]))
  t = 0
  while not queue.empty():
    node = queue.get()
    t += 1
    print(t, node)
    current = node[1][len(node[1]) - 1]

    if end in str(node[1]) or t == 25000:
      for i in range(0, len(node[1][:])):
        if node[1][:][i] not in usc_visited:
          usc_visited.append(node[1][:][i])
      break

    costv = node[0]
    for i in range(0, len(node[1][:])):
      if node[1][:][i] not in usc_visited:
        usc_visited.append(node[1][:][i])
    for neighbor in values[current]:
      temp = node[1][:]
      for i in range(1,len(txt)-1):  #len
        if int(start_node[i])==int(current) and int(end_node[i])==int(neighbor):
          ttt = cost[i]
      temp.append(neighbor)
      queue.put((float(costv) + float(ttt), temp))
  #print(node[1])
  for i in range(0, len(node[1])):
    usc_path.append(int(node[1][i]))   #int

  usc_dist = node[0]
  return usc_path, usc_dist, len(usc_visited)
```

However, this function will cost a lot of time. At first, I put a new “end” called 7224393854 and the result will be:

```
The number of nodes in the path found by UCS: 6
Total distance of path found by UCS: 76.293 m
The number of visited nodes in UCS: 10
```

Then, in order to present the result of end=1079387396, I wait for more than an hour and the final result be like:

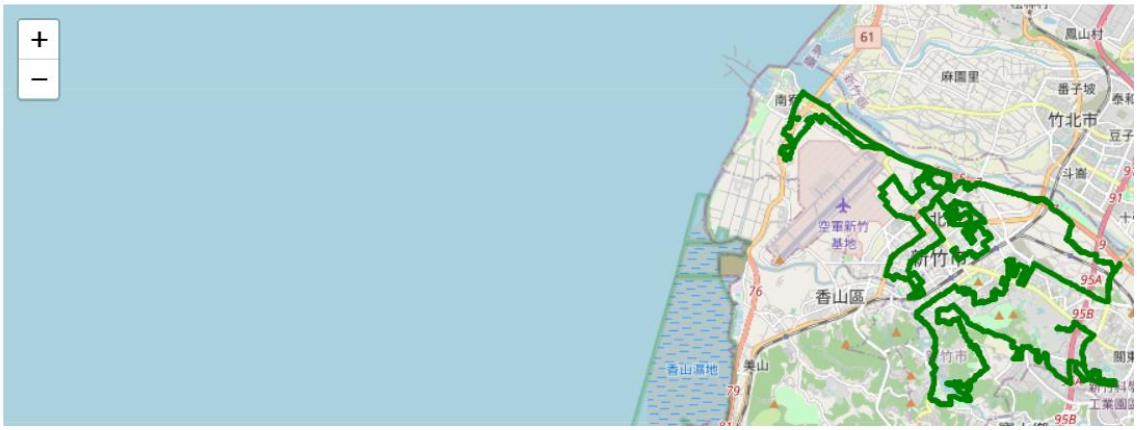
Part 5:

1. 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: 4274

DFS:

The number of nodes in the path found by DFS: 1718
Total distance of path found by DFS: 75504.31499999983 m
The number of visited nodes in DFS: 4712



2. 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: 4607

DFS:

The number of nodes in the path found by DFS: 930
Total distance of path found by DFS: 38752.30799999996 m
The number of visited nodes in DFS: 9366

3. 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: 15442.395000000002 m
The number of visited nodes in BFS: 11242

DFS:

The number of nodes in the path found by DFS: 900
Total distance of path found by DFS: 39219.99299999996 m
The number of visited nodes in DFS: 2248

Compare: BFS uses a queue to build the structure which follows the rule of first in first out. Also, one vertex is selected at a time when it is visited; then, all the adjacent vertex will be stored in the queue. DFS uses a stack to build the structure which follows the rule of first in last out. In addition, one vertex is pushed into the stack, and the next-visited vertex will also be pushed into the stack until no vertex can be visited. USC uses a queue as well but it should also consider the amount of distance. The least distance near the first vertex will be chosen until all distance be considered. Thus, using USC can find a unique total distance of path; on the contrary, BFS and DFS cannot.

Problems I met:

Although I had known the use of bfs and dfs before the class, I still need more patience to do it. Problems I met in part one is how to write down all the value of bfs_path. After using the bfs function, first I had is all the location in bfs_vistited instead of bfs_path. In order to find bfs_path, I should write down all the previous location of bfs_visited. To do this, I could use link list or build another list to save all these previous locations. Although it took more time to run the code, I used another list to save them because it was easier for me to write the code. After saving all these value, I could trace all the location in bfs_visited from the last to the front. Finally, I reversed the list and bfs_path was built. In part two, less problems did I meet because the step to provide dfs path was almost same as bfs. What I need to do was change the queue in bfs into the stack. In part three, ucs was a little bit like bfs but I needed to change queue into priority queue. This did not make much problems. However, the code I wrote needed very much time to run (some needed more than an hour). When the end point was close to the start point, It would not cost much time; in the contrary, it took a hard time for me to wait.