CMPT 310

Assignment 1

Chuxuan Zhang

301267261

shellyz@sfu.ca

There are five functions in my assignment1.py. They are:

- changePoint(start_loc): it changes the start and goal into list first and then make the value[i][j] into value[i-1][j-1] for suit the grid. After all, change the list back to tuple again so that it can be express in dictionary.
- get_neighbor(n,row,column,visited): In the n rows and n columns grid, there is a given row number i and a given column number j. The goal of this function is looking for the value[i][j]'s unvisited neighbour. First, get all the (i,j) 's neighbor and then check them if it is visited or not. If it has not been visited, use append to add the item into list. Finally, return the list.
- min_distance(distance,unvisited): In the code, I use distance dictionary to record the point and its cost from the start point. For this function, I am looking for the smallest distance or cost in all the unvisited node.
- find_predecessor(predecessor, start, goal, path = []): I build a dictionary called predecessor. When a point got its smallest cost, put its parents into predecessor so that I can retrieval all the nodes' parents. The goal of this function is for getting the path from goal to start by the predecessor dictionary.

- path_find(n, start_loc, goal_loc, values):
    - Assign to every node a tentative distance value: set it to zero for start_loc and to infinity for all other nodes.
    - Set the initial node as current. Create a list of all nodes called unvisited and an empty list called visited.
    - For the current node, consider all of its neighbors in the unvisited list and calculate their distances. Compare the newly calculated distance to the current assigned value and get the smaller one. For example, if the current node *A* is marked with a distance of 6, and the edge connecting it with a neighbor *B* has length 2, then the distance to *B* (through *A*) will be 6 + 2 + 1 = 9. If B was previously marked with a distance greater than 9 then change it to 9. And put A into B's predecessor dictionary. Otherwise, keep the current value.
    - When we are done considering all of the neighbors of the current node, put the current node into visited and remove it from the unvisited list.
    - If the destination is equal to the current which means we already have gone its smallest cost, the loop has finished.
    - Otherwise, select the unvisited node with the smallest distance, set it as the new "current node", and keep doing.
    - After getting the smallest cost, we according the predecessor dictionary to get its path and plus one for each of the rows and columns, then return the path.

In the assignment, I use Dijkstra's algorithm to calculate the shortest cost path in the grid. And the properties of Dijkstra's algorithm are as follow:

- Complete: Yes
- Optimal: Yes