CS566 – Extra Credit Assignment

Hanxiong Shi

[hanxiong@bu.edu](mailto:hanxiong@bu.edu)

**Compile Instructions**

The program is compiled in python, version 3.6.1. All the input files are put into a separate input folder. The program will read the input files during compilation.

Steps:

1. Unzip [Shi\_Hanxiong\_ec.zip] to folder [Shi\_Hanxiong\_ec]
2. $cd Shi\_Hanxiong\_ec/EC-ProgrammingAssignment

**If only python3 is installed on the computer**

1. $python Algorithm\_1.py // Run algorithm 1
2. $python Algorithm\_2.py // Run algorithm 2

**If both python2 and python3 are installed on the computer, try**

3. $python3 Algorithm\_1.py // Run algorithm 1

1. $python3 Algorithm\_2.py // Run algorithm 2

**Project Structure**

EC-ProgrammingAssignment

|- Input

|- direct\_distance\_1.txt

|- graph\_input\_1.txt

|- Algorithm1.py

|- Algorithm2.py

|- Common.py

The core code component consists of 3 files:

* common.py – contains all common functions that are shared between Algorithm1.py and Algorithm2.py
* Algorithm1.py – functions for demo-ing Algorithm1 only
* Algorithm2.py – functions for demo-ing Algorithm2 only

The input files for the graph and direct distance files are stored under input folder. So for further testing, just simply replace both files with the same names.

**Pseudo Code**

Common.py

// Read the direct distance file into a 2D array

// e.g.

// Read the graph input file into a 2D matrix

// read file by lines and store into an array

// Read user input, re-prompt when the input is invalid

// Prompt user to type in an input

// If input is single digit and is a letter

// Convert the node to uppercase, when necessary

// Recursive call read user input function again

// Check if two nodes are connected

// Look through the map graph and return a list of all the adjacent nodes that have not been visited

Algorithm1.py

// is to store the final shortest path taken to ‘Z’

// is to store all nodes that were once visited

// shortest node default to the first node in the list

// If the current node is the destination node, terminates

// Find adjacent and unvisited nodes

// If the node is already in the path, remove it from the node candidates array

// If there is no adjacent node that are also unvisited, it is dead end

// Find out the node to back track to, it should be the previously visited node

// Since the current node is no longer legit, remove it from the path map

// Recursive call algorithm\_1

// Loop through all adjacent nodes and find the one with smallest

// Check if the shortest node candidate is the destination node

// Add the destination node to the shortest path map

Algorithm2.py

// is to store the final shortest path taken to ‘Z’

// is to store all nodes that were once visited

// shortest node default to the first node in the list

// Check if the node has the shortest total distance

// If the current node is the destination node, terminates

// Find adjacent and unvisited nodes

// If the node is already in the path, remove it from the node candidates array

// If there is no adjacent node that are also unvisited, it is dead end

// Find out the node to back track to, it should be the previously visited node

// Since the current node is no longer legit, remove it from the path map

// Recursive call algorithm\_2

// Loop through all adjacent nodes and find the one with smallest

// Check if the shortest node candidate is the destination node

// Add the destination node to the shortest path map