CMPT310
Assignment 1
Sen Lin – sla248@sfu.ca – 301250505

Program Description

The program reads in a data file named "Asst1.data.txt" and retrieves the size, start/end points information and map data from the file. Sample data file is provided in the zip file. The program prints the path found with A* and Best First Search (BFS) respectively. Each path contains the information of nodes on the path, including FScores, GScores and HScores information for A* and HScores information for BFS. The program also prints and number of nodes expanded during the search.

Heuristic Function

The chosen heuristics is Manhattan Distance. The formula of Manhattan Distance between node n and endpoint e is:

$$H(n) = Abs(n.x - e.x) + Abs(n.y - e.y)$$

The reason of using Manhattan Distance is that Manhattan Distance is the minimum distance between a node to end node. Considering a map with no elevation difference: from start node to end node, the distance will be H(n)

Work Flow

A*

1. Retrieve start node and end node, with F(n) = 0.
2. Initialize open list and closed list. Put start node in the open list.
3. If open node is empty, search fails, and end node is not reachable. Otherwise get the node with smallest f(n), denoted as "current".
4. Find neighbours for f(n). For each neighbour:
   a) If neighbour is in closed list. Ignore.
   b) If neighbour is not in open list, calculate g(n) for neighbour by g(parent) + cost(neighbour, parent). Calculate h(n) using Manhattan Distance. Calculate and store f(n) = g(n) + h(n). Set parent of neighbour to current. Put neighbour into open list.
   c) If neighbour is in open list, calculate new g(n) -> g'(n) for neighbour using g(parent), calculate new f(n) => f'(n) with f'(n) = g'(n) + h(n). If f'(n) < f(n), update neighbour's f(n) with f'(n), update neighbour's parent with current. Otherwise, ignore neighbour.
5. Put current into closed list, remove current from open list.
   a) If current == end node, then A* reaches the goal.
   b) Get the parent of the end node. Then get the parent of the parent (repeat this process until no parent node found). All the parents construct a path.
   c) Otherwise, go back to step 3.

Best First Search

1. Retrieve start node and end node. Put start node into open list
2. If open list is empty, search fails, and end node is not reachable. If open list is not empty, retrieve the node with smallest heuristic value, denoted as current.
3. Get neighbours of current. For each neighbour, if neighbour is in open list, or closed list, ignore this neighbour. Otherwise set neighbour's parent as current, put neighbour into open list.
4. If current is not end node, return to step 2.
5. If current == end node, get the parent of the end node. Then get the parent of the parent (repeat this process until no parent node found). All the parents construct a path.

Run Instruction

Please put "Asst1.data.txt" with "main.py" file and execute "main.py". For other test case files, please rename them to "Asst1.data.txt" before execution.

Test Cases

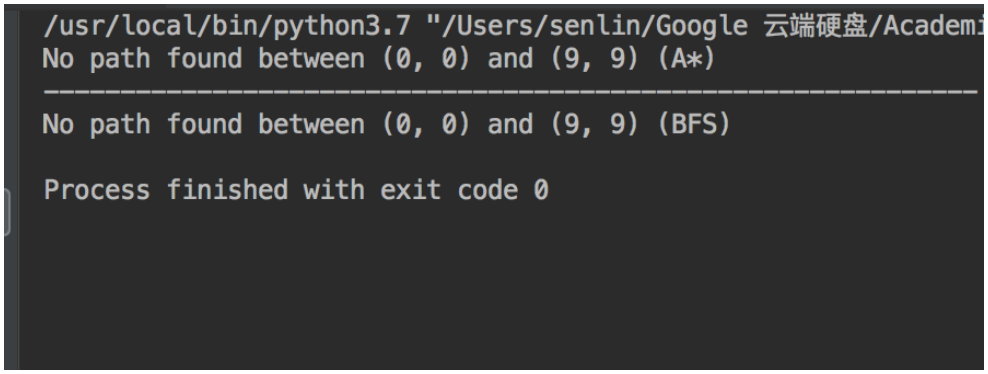
The zip file contains 3 test cases: "Asst.data.txt", "Asst.data(2)".txt and "Asst.data(3)".txt Please rename them to "Asst.data.txt" before running the program.
"Asst.data.txt": Regular test case.
"Asst.data(2).txt": Test case with designed path.
"Asst.data(3).txt": Test case with no possible path.

Output

The output prints the final path, number of nodes expanded during search, number of iterations used, and the information of each node in the path for the result found by A* and BFS.
Sample result is shown below:

```
A* Search has found the path
There are 80 nodes expended
The number of iterations is:  64
The path found from (0, 0) to (9, 9) is
Node(x, y):  (0, 0) FScore is: 0, GScore is: 0, HScore is: 18
Node(x, y):  (0, 1) FScore is: 18, GScore is: 1, HScore is: 17
Node(x, y):  (0, 2) FScore is: 18, GScore is: 2, HScore is: 16
Node(x, y):  (1, 2) FScore is: 19, GScore is: 4, HScore is: 15
Node(x, y):  (1, 3) FScore is: 19, GScore is: 5, HScore is: 14
Node(x, y):  (2, 3) FScore is: 19, GScore is: 6, HScore is: 13
Node(x, y):  (3, 3) FScore is: 19, GScore is: 7, HScore is: 12
Node(x, y):  (4, 3) FScore is: 19, GScore is: 8, HScore is: 11
Node(x, y):  (4, 4) FScore is: 20, GScore is: 10, HScore is: 10
Node(x, y):  (4, 5) FScore is: 20, GScore is: 11, HScore is: 9
Node(x, y):  (4, 6) FScore is: 20, GScore is: 12, HScore is: 8
Node(x, y):  (5, 6) FScore is: 20, GScore is: 13, HScore is: 7
Node(x, y):  (5, 7) FScore is: 20, GScore is: 14, HScore is: 6
Node(x, y):  (6, 7) FScore is: 20, GScore is: 15, HScore is: 5
Node(x, y):  (7, 7) FScore is: 20, GScore is: 16, HScore is: 4
Node(x, y):  (7, 8) FScore is: 20, GScore is: 17, HScore is: 3
Node(x, y):  (7, 9) FScore is: 20, GScore is: 18, HScore is: 2
Node(x, y):  (8, 9) FScore is: 20, GScore is: 19, HScore is: 1
Node(x, y):  (9, 9) FScore is: 20, GScore is: 20, HScore is: 0
------------------------------------------------------------
Best First Search has found the path
```

If A*/ BFS cannot find the path, program will print no path found message.



Components

This part explains some components in the python script. Other functions are helpers used by these functions.

Class Node:

Node class represent a point in the map. It has elevation, row, column and parent attributes.

createNode(row, column):

When the position is not out of bound, create Node. Otherwise return None.

getCurrentNode(nodes, fScores, end):

Get current node for A* or BFS. When fScore is None, get current node for BFS.

The design here is that the nodes is not a priority queue. It simply returns the node with smallest FScore to functionally replace the sorting.

getPath(node):

From a given node, get its predecessors and construct a path.

aStar(start, end)/BFS(start, end):

Returns a path found by A*/BFS algorithm between start node and end node. If end node is not reachable, function will print not found message.

writePath(nodes, start, end, mode, fScores, gScores)

Writes the path information, together with nodes information for each node in the path.