Assignment 1

Siwei Tang (siweitang)
sw.tang@mail.utoronto.ca

Changyan Xu (xuchangy)
changyan.xu@mail.utoronto.ca

April 15, 2021

Reference

Not applicable.

Question 1

(a) Data Structure:

- **A matrix (direct access)** to represent the maze, which is composed of an array of array; each tuple represents a maze cell, including information of directions and colour of the arrow(s).
- **A tree** constructed by BFS with a default root node to represent the steps taken along the algorithm runs from the start, and to find the shortest path.
- **A queue** used to explore new nodes based on BFS, also used to check if all branches fall into dead ends.
- **A dictionary implemented by python built-in hash table (direct access)** traces all the visited nodes and to check if the program falls into dead ends when there is no solution for the maze.
  - Key: A tuple with information of (position, direction, parent position)
  - Value: A list of step sizes that taken placed on landing on this grid with the given key

Written description of the algorithm:

1. The algorithm is based on Breadth-First Search (BFS). For every steps it takes from the start position, the algorithm creates a node for each step and record information including
   - **position**: the row-column coordinate of this cell, while the top-left cell of the maze is represented as $(0, 0)$
   - **direction**: one direction in this cell that the path branch choose
   - **colour**: the colour of the direction in this cell
   - **depth**: the depth of the node in the BFS tree it creates
   - **parent**: the parent node of the current node
   - **step_size**: the current step size it takes before stepping onto this cell

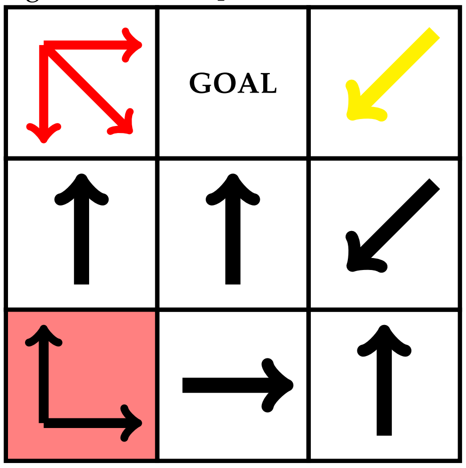
   , and constructing a BFS tree under a default root.
2. <u>If there exists a solution</u>, as the first time the BFS explores the position of the goal, the algorithm records the goal node and stops the BFS. Then it trace back from the goal node upto the root by accessing the `.parent` attribute for each node, so that we are able to obtain the path which is the shortest and its length.

3. If the maze has no solution, each of every branch of the BFS would at some point falls in to dead end by looping around the same nodes. We check if a ”stuck” situation has occurred for each branch with the dictionary and the queue. As we detect all branches (i.e. all nodes of the same depth in the BFS tree) have the ”stuck” situation, we terminate the program and declare no solution to the maze.

(b) In the `.txt` file, the inputs are represented in this way:

- Line 1: number of rows, i
- Line 2: number of columns, j
- Line 3 to Line $2 + i$: Each line consists of j elements separated by semicolons, where each element is represented by a list.
- In each list, before the colon is a tuple which indicates the direction of the arrows in the corresponding cell, after the colon is the colour of the arrow, for the goal cell, the tuple contains string goal, and just blank space after colon.
- Colours are represented by ”R” for red, ”B” for black, ”Y” for yellow, ” ” for blank
- Directions are represented by ”N”, ”E”, ”S”, ”W”, ”NE”, ”NW”, ”SE”, ”SW”, or a space for blank
- Line $3 + i$ (the last line): A tuple represents the (row, column) coordinates of the start cell, where we define the left-top corner of the maze as (0, 0).

e.g. For the `example_maze` on the handout,



The input `.txt` file would be represented as:

$$
\begin{array}{l}
3 \\
3 \\
[(E,\ \ S,\ \ SE) : R];\ \ [(goal) :\ ];\ \ [(SW) : Y] \\
[(N) : B];\ \ [(N) : B];\ \ [(SW) : B] \\
[(N,\ \ E) : B]\ [(E) : B];\ \ [(N) : B] \\
(2,\ \ 0)
\end{array}
$$

The information that be read for the algorithm use would be as following in Python:

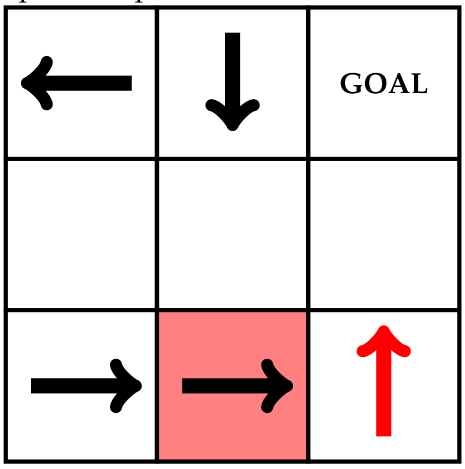
- row_size: 3
- col_size: 3

- maze representation: `[[(('E', 'S', 'SE'), 'R'), (('goal',), ''), (('SW',), 'Y')], [(('N',), 'B'), (('N',), 'B'), (('SW',), 'B')], [(('N', 'E'), 'B'), (('E',), 'B'), (('N',), 'B')]]`
- start position: `(2, 0)`
- goal position: `(0, 1)`

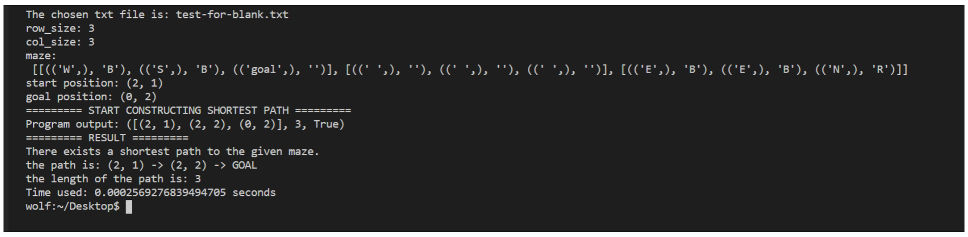
(c) Please see the algorithm implementation in `Alice.py` file.

(d) Testing with mazes:

- `Test-for-blank.txt` – with empty cell; exists a solution
  - Specific input:

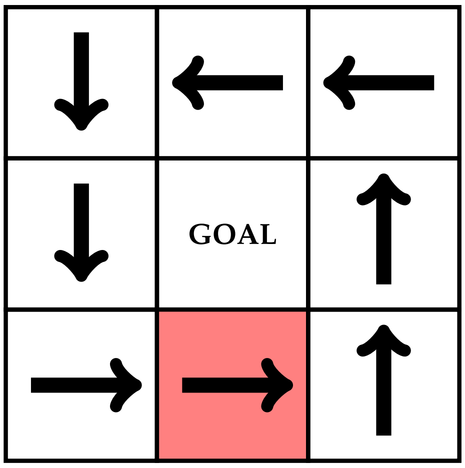


  - Expected output:
    - there exists a shortest path to the given maze.
    - the path is: $(2,1) \rightarrow (2,2) \rightarrow$ GOAL
    - the length of the path is: 3
  - Actual output:

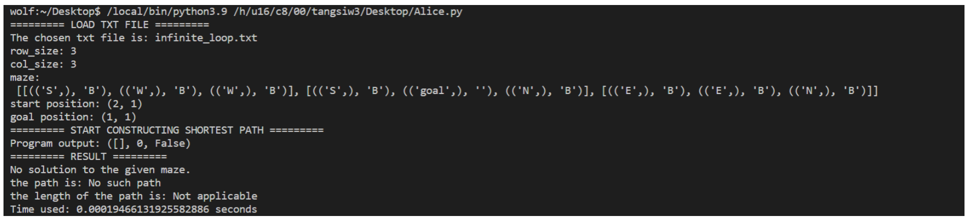


```
The chosen txt file is: test-for-blank.txt
row_size: 3
col_size: 3
maze:
 [[(('W',), 'B'), (('S',), 'B'), (('goal',), '')], [((' ',), ''), ((' ',), ''), ((' ',), '')], [(('E',), 'B'), (('E',), 'B'), (('N',), 'R')]]
start position: (2, 1)
goal position: (0, 2)
========= START CONSTRUCTING SHORTEST PATH =========
Program output: ([(2, 1), (2, 2), (0, 2)], 3, True)
========= RESULT =========
There exists a shortest path to the given maze.
the path is: (2, 1) -> (2, 2) -> GOAL
the length of the path is: 3
Time used: 0.0002569276839494705 seconds
wolf:~/Desktop$ █
```

- `infinite_loop.txt` – no empty cells; no solution
  - Specific input:

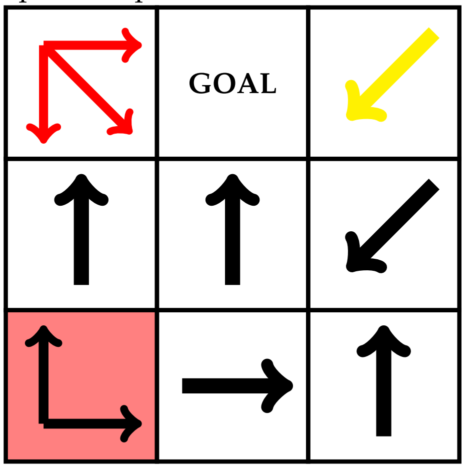


  * Expected output:
    - No solution to the given maze.
    - the path is: No such path
    - the length of the path is: Not applicable
  * Actual output:

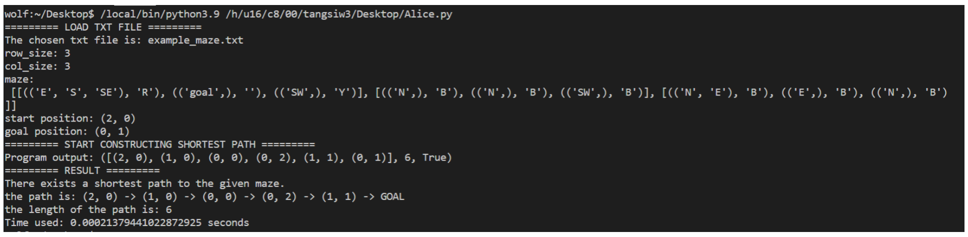


```
wolf:~/Desktop$ /local/bin/python3.9 /h/u16/c8/00/tangsiw3/Desktop/Alice.py
========= LOAD TXT FILE =========
The chosen txt file is: infinite_loop.txt
row_size: 3
col_size: 3
maze:
 [[(('S',), 'B'), (('W',), 'B'), (('W',), 'B')], [(('S',), 'B'), (('goal',), ''), (('N',), 'B')], [(('E',), 'B'), (('E',), 'B'), (('N',), 'B')]]
start position: (2, 1)
goal position: (1, 1)
========= START CONSTRUCTING SHORTEST PATH =========
Program output: ([], 0, False)
========= RESULT =========
No solution to the given maze.
the path is: No such path
the length of the path is: Not applicable
Time used: 0.00019466131925582886 seconds
```

- `example_maze.txt` – no empty cells; exists a solution
  * Specific input:

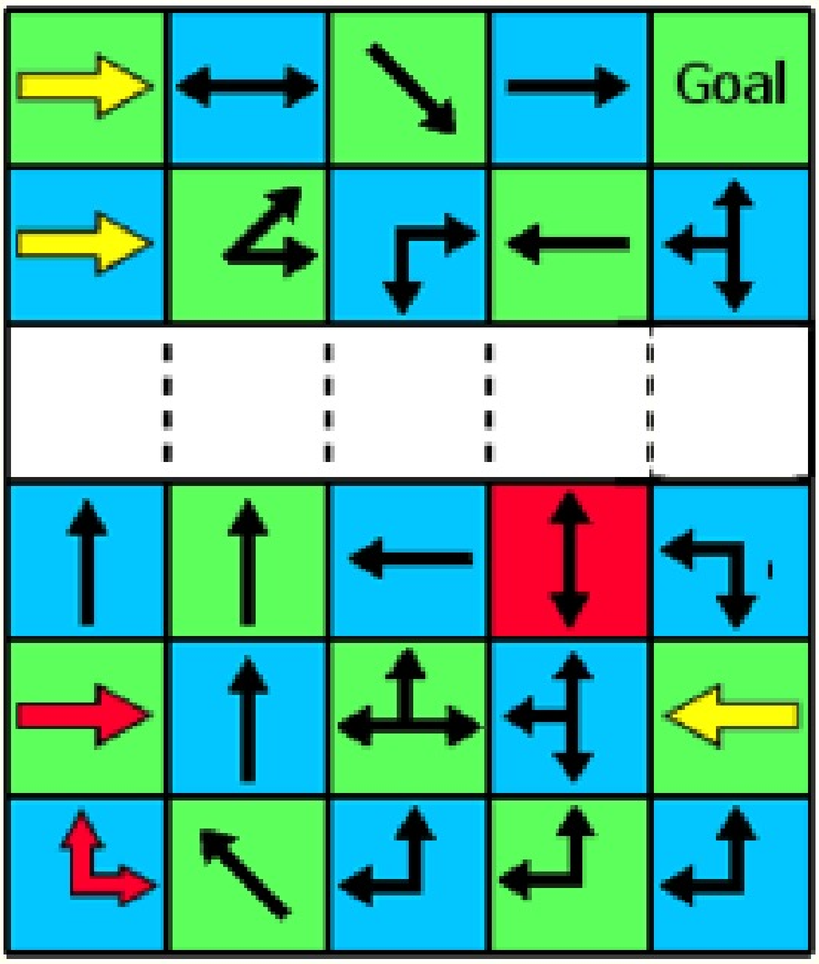


  * Expected output:
    - There exists a shortest path to the given maze.
    - the path is: (2, 0) $\rightarrow$ (1, 0) $\rightarrow$ (0, 0) $\rightarrow$ (0, 2) $\rightarrow$ (1, 1) $\rightarrow$ GOAL
    - the length of the path is: 6
  * Actual output:

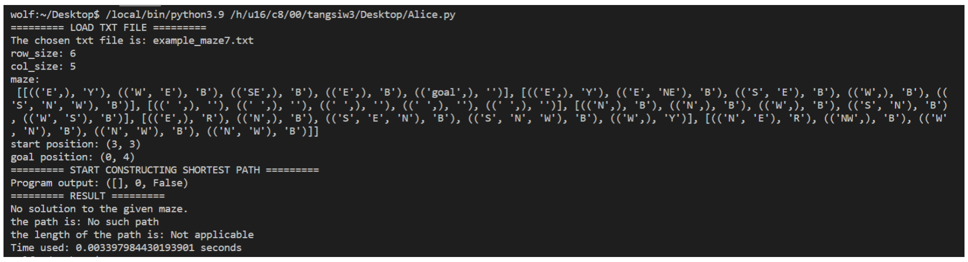


```
wolf:~/Desktop$ /local/bin/python3.9 /h/u16/c8/00/tangsiw3/Desktop/Alice.py
========= LOAD TXT FILE =========
The chosen txt file is: example_maze.txt
row_size: 3
col_size: 3
maze:
 [[(('E', 'S', 'SE'), 'R'), (('goal',), ''), (('SW',), 'Y')], [(('N',), 'B'), (('N',), 'B'), (('SW',), 'B')], [(('N', 'E'), 'B'), (('E',), 'B'), (('N',), 'B')
]]
start position: (2, 0)
goal position: (0, 1)
========= START CONSTRUCTING SHORTEST PATH =========
Program output: ([(2, 0), (1, 0), (0, 0), (0, 2), (1, 1), (0, 1)], 6, True)
========= RESULT =========
There exists a shortest path to the given maze.
the path is: (2, 0) -> (1, 0) -> (0, 0) -> (0, 2) -> (1, 1) -> GOAL
the length of the path is: 6
Time used: 0.00021379441022872925 seconds
```

- `example_maze7.txt` – empty cells; no solution, rectangle
  * Specific input:



  * Expected output:
    · No solution to the given maze.
    · the path is: No such path
    · the length of the path is: Not applicable
  * Actual output:



```
wolf:~/Desktop$ /local/bin/python3.9 /h/u16/c8/00/tangsiw3/Desktop/Alice.py
========= LOAD TXT FILE =========
The chosen txt file is: example_maze7.txt
row_size: 6
col_size: 5
maze:
 [[(('E',), 'Y'), (('W', 'E'), 'B'), (('SE',), 'B'), (('E',), 'B'), (('goal',), '')], [(('E',), 'Y'), (('E', 'NE'), 'B'), (('S', 'E'), 'B'), (('W',), 'B'), ((
'S', 'N', 'W'), 'B')], [((' ',), ''), ((' ',), ''), ((' ',), ''), ((' ',), ''), ((' ',), '')], [(('N',), 'B'), (('N',), 'B'), (('W',), 'B'), (('S', 'N'), 'B')
, (('W', 'S'), 'B')], [(('E',), 'R'), (('N',), 'B'), (('S', 'E', 'N'), 'B'), (('S', 'N', 'W'), 'B'), (('W',), 'Y')], [(('N', 'E'), 'R'), (('NW',), 'B'), (('W'
, 'N'), 'B'), (('N', 'W'), 'B'), (('N', 'W'), 'B')]]
start position: (3, 3)
goal position: (0, 4)
========= START CONSTRUCTING SHORTEST PATH =========
Program output: ([], 0, False)
========= RESULT =========
No solution to the given maze.
the path is: No such path
the length of the path is: Not applicable
Time used: 0.003397984430193901 seconds
```