Q1 Instructions

0 Points

To receive full credit on this quiz, you must score at least 50%.

The Github repo for Lecture 8 is at:
https://github.com/ucsd-cse12-w21/ucsd-cse12-w21.github.io/tree/master/lectures/lecture-08

Q2 Maze

2 Points

The following uses a plain text format for mazes:

indicates a wall
_ indicates an empty space
F indicates the finish square
S indicates the start square

For example, the maze from class would be written:

```
#_#_
____
_##S
F___
```

When checking neighbors, you must add them to the data structure in the order East, South, West, North.

Note: this is a different order than was used in lecture.

For this example, the solution with a stack, and the add order specified above, is:

```
#_#_
****
*##S
F___
```

For this example, the solution with a queue, and the add order specified above, is:

```
#_#_
```

```
____
_##S
F***
```

Q2.1 Maze 1

1 Point

For the example maze above what are the row and col, respectively, of the first Square that gets added to the data structure?

- 1 and 3
- 2 and 2
- [✓ 2 and 3]
- 3 and 3
- 3 and 4

Q2.2 Maze 2

1 Point

For the example maze above what are the row and col, respectively, of the finish Square?

- 1 and 0
- 2 and 0
- 2 and 1
- [✓ 3 and 0]
- 3 and 1

Q3 Worklist (Stack/Queue)

2 Points

The pseudocode between the two asterisks is added to the search algorithm, printing the rows and cols of each Square in the data structure after an iteration of the while loop:

Note: worklist (wl) is either a stack or a queue.

```
initialize wl to be a new empty worklist (stack or queue)
add the start square to wl
while wl is not empty:
   let current = remove the first element from wl (pop or dequeue)
   mark current as visited
   if current is the finish square
      return current
   else
      for each neighbor of current that isn't a wall and isn't visited
         set the previous of the neighbor to current
         add the neighbor to the worklist (push or enqueue)
   *for each square in the worklist
      print its row + ", " + its col + "; "* // no newline

if the loop ended, return null (no path found)
```

For the example maze above, what will be printed after the second iteration of the while loop, before the third iteration, for a stack implementation? The printing starts from the BOTTOM of the stack.

- ○ 1, 3; 3, 2;
- ○ 3, 3; 1, 3;
- ○ 3, 3; 0, 3;
- ○ 2, 3; 3, 3; 1, 3;
- [✓ 3, 3; 1, 2; 0, 3;]