M. SAVING MICKEY MOUSE

STACK, BACK TRACKING, DFS

(Data structure and Backtrack)

Time Limited: 3 seconds

Mickey mouse was in the maze[1] for very long time, with nothing to eat. So, we need to help Mickey (from now we just call mouse only) to find out the way to exit of the maze as soon as possible. *We just need to find out the first exiting road only, and don't need to find out all the exiting roads, and the best exiting road just only because of the mouse is very hungry*. Figure 1 shows an example of a maze, and Figure 2 shows the input data structure to represent of Figure 1.

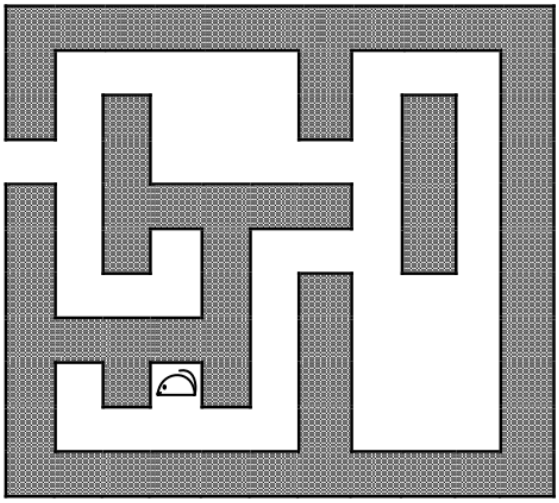

Figure 1. A mouse in a maze

```
1 1 1 1 1 1 1 1 1 1 1
1 0 0 0 0 0 1 0 0 0 1
1 0 1 0 0 0 1 0 1 0 1
e 0 1 0 0 0 0 0 1 0 1
1 0 1 1 1 1 1 0 1 0 1
1 0 1 0 1 0 0 0 1 0 1
1 0 0 0 1 0 1 0 0 0 1
1 1 1 1 1 0 1 0 0 0 1
1 0 1 m 1 0 1 0 0 0 1
1 0 0 0 0 0 1 0 0 0 1
1 1 1 1 1 1 1 1 1 1 1
```

Figure 2. Internal expression

e: exit , m: mickey mouse , 0: road , 1: wall

The mouse hopes to escape from the maze by systematically trying all the routes. If it reaches a dead end, it retracts its steps to the last position and begins to try at least one or more untried path. **For each position, the mouse must try to go in one of four directions: left (the highest priority), right (high priority), up (medium priority),**

[1] Mê cung

down (the lowest priority)[2]. Regardless of how close it is to the exit, it always tries the open paths in this order, which may lead to some unnecessary detours. By retaining information that allows for resuming the search after a dead end is reached, the mouse uses a method called backtracking.

The maze is implemented as a two-dimensional char array in which passages are marked with 0s, walls by 1s, exit position by the letter e, and the initial position of the mouse by the letter m (Figure 2). In this program, the maze problem is slightly generalized by allowing the exit to be in any position of the maze and allowing passages to be on the borderline. To protect itself from falling off the array by trying to continue its path when an open cell is reached on one of the borderlines, the mouse also has to constantly check whether it is in such borderline position or not. To avoid it, the program automatically puts a frame of 1s around the maze entered by the user.

The program uses two stacks: one to initialize the maze and the other to implement backtracking. The user enters maze data one line at a time. The maze entered by the user can have any number of rows and any number of columns. The only assumption the program makes is that all rows are of the same length and it uses only these characters: any number of 1s, any number of 0s, one e, and one m. The rows are pushed on the stack mazeRows in the order they are entered after attaching one 1 at the beginning and one 1 at the end. After all rows are entered, the size of the array store can be determined, and then the rows from the stack are transferred to the array.

The second stack, mazeStack, is used in the process of escaping the maze. To remember untried paths for subsequent tries, the positions of the untried neighbors of the current position (if any) are stored on a stack and always in the same order, first upper neighbor, then lower, then left, and finally right. After stacking the open avenues on the stack, the mouse takes the topmost position and tries to follow it by first storing untried neighbors and then trying the topmost position and so forth, until it reaches the exit or exhausts all possibilities and finds itself trapped. To avoid falling into an infinite loop of trying paths that have already been investigated, each visited position of the maze is marked with a period (using dot . to represent).

Example:

1) After the mouse enters the maze.

```
1 1 0 0
0 0 0 e
```

[2] Please note for this important requirement to find out the first exiting road for the mickey mouse.

```
0 0 m 1
```

The maze is immediately surrounded with a wall/frame of 1.

```
1 1 1 1 1 1
1 1 1 0 0 1
1 0 0 0 e 1
1 0 0 m 1 1
1 1 1 1 1 1
```

entryCell and currentCell are initialized to (3, 3) and exitCell to (2, 4) (Figure 3a).

Note: index is started from zero (0)

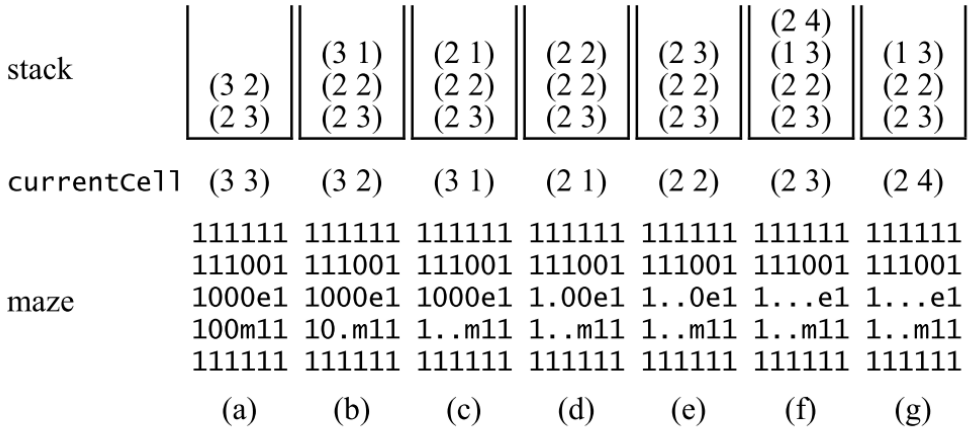

Figure 3. An example of processing a maze

Explain: e: exit , m: mickey mouse , 0: road , 1: wall , "." : visited

2) Because currentCell is not equal to exitCell, all four neighbors of the current cell (3, 3) are tested, and only two of them are candidates for processing, namely, (3, 2) and (2, 3); therefore they are pushed onto the stack. The stack is checked to see whether it contains any position, and because it is not empty, the topmost position (3, 2) becomes current (Figure 3b).

3) currentCell is still not equal to exitCell; therefore the two viable options accessible from (3, 2) are pushed onto the stack, namely, positions (2, 2) and (3, 1).

Note that the position holding the mouse is not included in the stack. After the current position is marked as visited, the situation in the maze is as in Figure 3c.

Now, the topmost position, (3, 1), is popped off the stack, and it becomes the value of currentCell. The process continues until the exit is reached, as shown step by step in Figure 3d through 3f.

Note that in step four (Figure 3d), the position (2, 2) is pushed onto the stack, although it is already there. However, this poses no danger, because when the second instance of this position is popped from the stack, all the paths leading from this position have already been investigated using the first instance of this position on the stack. Note also that the mouse makes a detour, although there is a shorter path from its initial position to the exit.

***You be asked to help mickey mouse to find out the first exiting road only, and must follow the condition:* "For each position, the mouse must try to go in one of four directions: left (the highest priority), right (high priority), up (medium priority), down (the lowest priority)"**

Example 1:

Input	Output
1110 101e 10m1	1110 101e 10m1 111111 111101 1101e1 110m11 111111 111111 111101 1101e1 11.m11 111111 111111 111101 11.1e1 11.m11 111111

This is the input data

The result of immediately surrounded maze with a wall/frame of 1.

The mouse start to move, with . (dot) character represent the direction of mouse movement.

In this case, mouse move to cell (3, 2)

After move to cell (2, 2), mouse cannot move to the others cell, so Failure.

	Failure

Example 2:

Input	Output
1000 101e 10m1	1000 101e 10m1 111111 110001 1101e1 110m11 111111 111111 110001 1101e1 11.m11 111111 111111 110001 11.1e1 11.m11 111111 111111 11.001 11.1e1 11.m11 111111 111111 11..01 11.1e1 11.m11 111111 111111 11...1

	11.1e1 11.m11 111111 Success. This is just only the first successful exiting road.

Example 3:

Input	Output
1100 000e 00m1	1100 000e 00m1 111111 111001 1000e1 100m11 111111 111111 111001 1000e1 10.m11 111111 111111 111001 1000e1 1..m11 111111 111111 111001 1.00e1 1..m11 111111 111111 111001 1..0e1 1..m11

	111111 111111 111001 1...e1 1..m11 111111 Success. This is just only the first successful exiting road.