Recitation Week 8 Extra Problem

April 1, 2024

Problem statement

You and a monster are in a labyrinth. When taking a step to some direction in the labyrinth, the monster may simultaneously take one as well. Your goal is to reach the exit square without ever sharing a square with a monster. Note that the only moves allowed are in 4 directions: Up, Right, Down, and Left.

Your task is to find out if your goal is possible. Your plan has to work in any situation; even if the monsters know your path beforehand.

Example 1

Your and the monster's initial locations are indicated by P (for player) and M (for monster) respectively. The exit is shown in red.

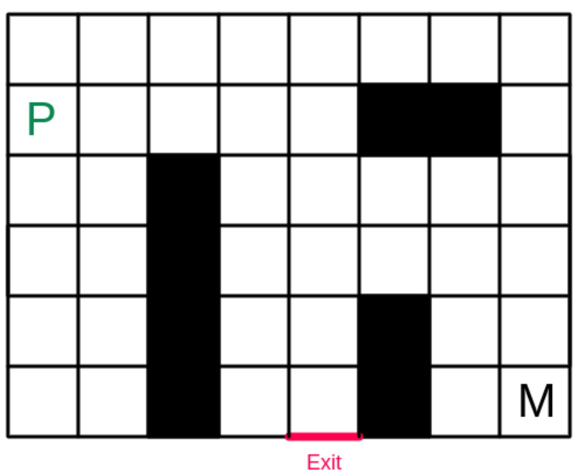

Figure 1: No exit plan, a.k.a Monster wins

No matter what path P takes here, Monster can get to one of the squares of that path faster than P and wait there to catch it.

Example 2

In the example below, the player can move towards the exit safely. The Monster can never catch up.

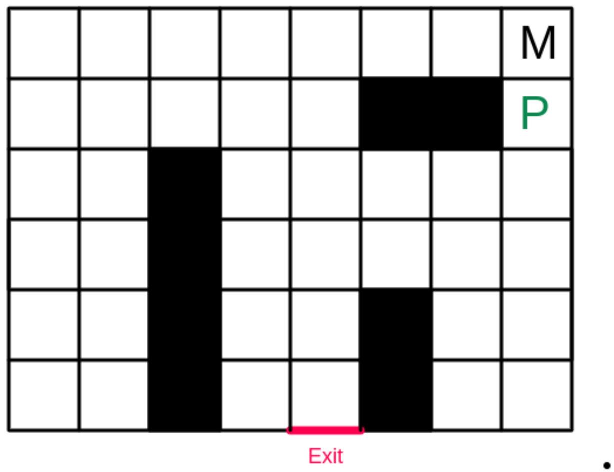

Figure 2: Player wins

Solution

Let the path that the player takes be $v_1 \to v_2 \to \ldots \to v_k$ where v_i represents a cell in the grid. It's obvious that the player must get to each v_i faster than the monster can, otherwise the monster would just wait for the player in that cell.

However, it is not necessary to check every single v_i, as the following observation suggests:

The player can get to $v_1, v_2, \ldots, v_k$ faster than the monster if and only if the player can get to v_k faster than the monster

Proof:

- $\Rightarrow$: This case is trivial. The player getting to the cell v_k is already included in the premise.
- $\Leftarrow$: Consider the opposite - that the monster gets to some v_i faster than the player. Then, the monster can follow the same path player would have taken, namely $v_i \to v_{i+1} \to \ldots \to v_k$ and get to v_k faster than the player. This is a contradiction since our assumption is that v_k is reached first by the player.

Therefore, we just need to see who reaches the exit cell v_k first! This can be done by initiating a BFS from the exit cell and checking the distances to P and to M. Or alternatively, you can run BFS from P, then from M and checking their respective distances to the exit. Below is an algorithm for the former approach:

```
1  EscapeTheMaze(Grid, P, M, E)
2      BFS(Grid, E)
3      if Grid[P].distance < Grid[M].distance then
4          return "player escapes"
5      else
6          return "no escape plan"
```

```
1  BFS(Grid, E)
2      for cell ∈ Grid do
3          cell.explored = false
4          cell.distance = ∞
5      Let Q be an empty queue
6      enqueue(Q, E)
7      Grid[E].explored = true
8      Grid[E].distance = 0
9      while Q ≠ ∅ do
10         u = dequeue(Q)
11         for dir ∈ {U, L, D, R} do
12             Let neighbor be the cell in direction dir of u
13             if neighbor is valid and Grid[neighbor].explored == false then
14                 Grid[neighbor].explored = true
15                 Grid[neighbor].distance = Grid[u].distance + 1
16                 enqueue(Q, neighbor)
```

Runtime:

On line 2 we only call BFS once, then on the lines following we do constant operation comparisons. If we assume the Grid is $n \times m$, the algorithm takes $O(nm)$ overall time due to BFS.