Code Jam to I/O for Women 2019 - Code Jam to I/O for Women

Analysis: Grid Escape

Test set 1

In the first test set, there can be at most 8 rooms in the grid. Each room can have its working exit in any one of the four directions, so we can use brute force to check all 4^8 possible ways to choose the 8 working exits, and see how many players escape from each possible layout. Either we will find a solution, or we will show that the case is `IMPOSSIBLE`.

Test set 2

In the second test set, we can no longer get away with an exhaustive approach! However, since we have some freedom in designing the grid, it's worth looking for a constructive solution.

If **R** = **C** = 1, then **K** = 0 is impossible, and any solution works for **K** = 1. Otherwise, imagine drawing a "snaking" path through all of the rooms of the grid, choosing our working doors to allow us to follow this path. We start in the northwesternmost cell of the grid, and move eastward through rooms until we reach the northeasternmost cell. (If **C** = 1, we will already be there and so we will not need to move.) Then we make one move south, and if we have not exited the grid, we move westward through rooms until we reach a cell on the western border. Then we make one more move south (if possible), and then move eastward, and so on, until one of our moves to the south exits the grid.

(By the way, a more obscure term for this type of winding path is *boustrophedon*, which derives from a way to direct oxen when plowing a field. Algorithms have been useful for a very long time!)

Notice that all of the players can escape along our path, no matter where they start. Now consider choosing some room along the path and "reversing" its working door — that is, choosing the door that opens onto the previous room on the path. Then every player who starts in that room or a room earlier than it along the path cannot escape — they will be trapped in an infinite loop! Every other player will still be able to escape just as before.

Since we can choose any room along the path, we can allow any desired number of players to escape... well, *almost* any number. If we try to use the above strategy to allow all but one of the players to escape, it breaks down because the very first room (where that player is) has no "previous room on the path".

In fact, we can never allow all but one of the players to escape. Consider the room with the one player who we want to prevent from escaping. One of their doors has to work, and it cannot open onto the outside world, so it must open onto another room. But that means that our player will share the fate of the player in that other room — either they will both be able to escape, or neither will.

So, if **K** = **R** × **C** - 1, the case is `IMPOSSIBLE`, and for any other case, the above construction gets us our answer. If **K** = **R** × **C**, we do not modify any of the pathâ€™s doors; otherwise, to allow exactly **K** of the players to escape, we "reverse" the door in the (**K**+1)-th-from-last room of the path.

Many other approaches are possible. For example, we can design around the target number of players who cannot escape. As long as that number is at least two, we can create a "nucleus" of the trap in the upper left corner of the grid: either an `E` room to the west of a `W` room, or an `S`

room to the north of an `N` room, depending on our given dimensions. Then we can turn cells sharing rows with the nucleus into `W`s, cells sharing columns with the nucleus into `N`s, and other cells (scanning top-to-bottom, then left-to-right) into `W`s until our set of trapped rooms is large enough. The remaining rooms can safely be filled with `S`. Alternatively, we can use flood fill to extend the sink.