Code Jam 2016 - World Finals

Analysis: Map Reduce

Map Reduce: Analysis

Some initial checks

First, calculate two values: the length L_i of any shortest path from the start to the finish (using BFS), and the Manhattan distance M from the start to the finish (ignoring walls). Based on those results, we can immediately detect some impossible cases:

- We can only remove walls, so we have no way to *increase* the length of the shortest path above L_i. So, if L_i is less than **D**, there is no solution.
- Similarly, if M is greater than **D**, removing walls does not help; even a blank maze consisting of only a border would still have a shortest path length > **D**.
- If the parity of L_i and **D** is different, there is no solution. The lengths of all paths between any two given cells have the same parity, because each step flips the parity of the sum of the indices of the row and column.

Surprisingly, in any other case, there is always a solution. The rest of the problem is to provide a constructive proof of that fact.

A crucial observation

We want to remove walls to change the current shortest path length L to match **D**. The key to solving this problem is to notice that we can always remove a wall such that the new shortest path has a length of either L or L-2. The proof of this is somewhat difficult, but we can discuss it intuitively (a formal proof follows at the end of this analysis).

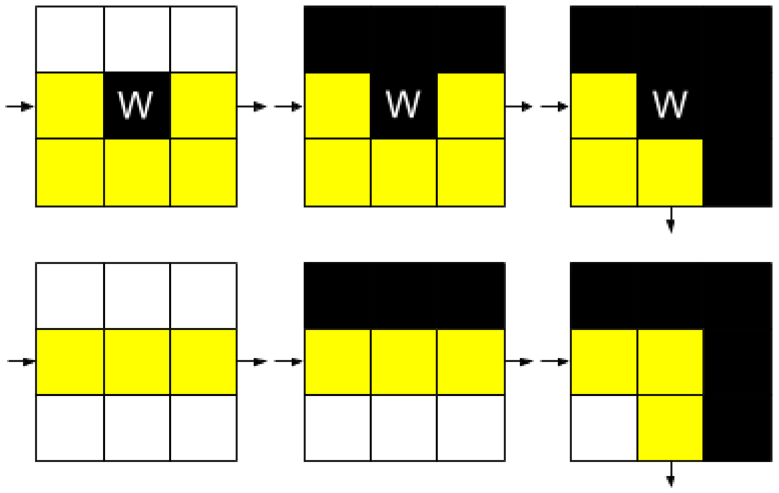
Consider a connected component of walls. It either includes the border or it doesn't. Now, pick some wall W within that connected component that is as far as possible away from either the border, or some arbitrary wall within the component if it doesn't include the border. Using the fact that all empty spaces are connected and that no two walls can connect at a corner, we can find that the 3×3 neighborhood of W looks like one of the following cases, up to symmetry (`#` is a wall, `.` is a space, `?` can be either):

```
...   ?#?   ?##
.W.   .W.   .W#
...   ...   ..?
```

Let's consider each case in turn. We will show that if the shortest path proceeded through the 3×3 neighborhood of W, removing W will decrease the length of the shortest path by at most 2:

- *W has zero neighbors that are walls*: The only path that would be shortened by removing W is a path that goes around W. So, removing W will shorten the path by 2, since the path can now go directly through W.
- *W has one neighbor that is a wall*: Say we have the case pictured above, and the shortest path proceeds from the top-left corner around the bottom to the top-right corner. Removing W again shortens the path by 2.
- *W has two neighbors that are walls*: Removing W doesn't shorten the path at all. (The reason we have this case is that removing this wall can open up other walls to be removed.)

Here are some illustrations of the above three cases. (For simplicity, we assume here that all the ?s are walls, but the argument holds regardless.)



To solve the problem, then, we just continually remove walls from the map (keeping in mind that removing a wall may make another wall removable) until the shortest path is equal to **D**. For the Small, we can repeatedly scan the map for removable walls and remove a wall; we continue this until the shortest path is the required length.

For the Large dataset, scanning the map repeatedly is too slow, so we need a different approach. We can figure out a list of walls to remove, in order, then binary search on the number of walls to remove to make a path of the required length. To find this list, we can scan the map once, then initialize a queue containing all the removable walls. Then, each time we choose a wall to remove next, we scan each of its neighbors to see if it's removable, and add any newly-removable walls to the back of the queue. We also need to scan the neighbors for walls that may have become unremovable, and remove any such walls from the queue. For example, if a connected component is just a 2x2 square, all of its walls are initially removable, but after removing one of them, only two of the three remaining walls can be removed. So we may sometimes need to remove a wall from the queue, and then perhaps even re-insert it later.

With this method, we're only scanning the entire map once, then doing constant additional work per wall we remove (O(N) total), so it's fast enough for the Large. (This method will eventually remove every wall, except that it might leave an extra-thick unremovable border. This border doesn't matter, since removing it wouldn't change the shortest path.)

A formal proof

As before, let L_i be the shortest path initially and M be the Manhattan distance. We claim that the problem is solvable for any **D** (of the same parity as L_i) between L_i and M. It suffices to show that you can always delete a wall while keeping the maze valid and without decreasing the current shortest path length L by more than 2.

Consider some connected component of walls. If it includes the outer boundary, let B be the set of walls on the outer boundary. Otherwise, let B be an arbitrary wall in the component. Let A be a wall in the component that is adjacent to an empty cell and maximally far apart from B (based on distance staying within the component). We'll delete A.

This adds an empty cell adjacent to another one, and so all empty cells stay connected. We next need to show that it can't make two walls touch only at a corner. By way of contradiction, suppose that X and Y are walls adjacent to both A and an empty cell Z. Z and A are connected by empty cells, so X and Y cannot be connected by walls after deleting A. Thus one of X or Y must be further from B than A is. But X and Y are not on the outer boundary, and are connected to B before A is deleted, and are adjacent to Z, so we have a contradiction.

Finally, we need to show that deleting A cannot make two empty cells greater than two steps closer to each other. The only way this could happen is if A were adjacent to empty opposite cells X and Y, and walls W and Z. As above, X and Y are connected, so Z and W cannot be connected by walls after deleting A. This leads to the same contradiction as before. W and Z may not be adjacent to an empty cell, but they are adjacent to something other than A that is. Either W or Z or this adjacent cell will contradict the choice of A. Note that the key claim here is false if walls are allowed to touch only at corners, but the problem setup disallows that.