Jordan Allard
CSCI 264-01
Homework 5

HW5 Problem 4 Writeup

a) Verbal Description

This algorithm uses breadth-first search to test various configurations, which consist of stored positions of Thing 1 and Thing 2, to determine the shortest amount of moves required for both Things to leave the map at the same time, if such a task is possible.

b) Pseudocode

```
Given a map of size a x b
Given Thing 1 start position (a1, b1) and Thing 2 start position (a2, b2)
beg = 1
end = 2
queue[beg] = (a1, b1, a2, b2, 0)
visited[a1, b1, a2, b2] = true
solutionDepth = -1
While solutionDepth == -1 and beg < end:
      head = queue[beg]
      north = (a1 - 1, b1, a2 - 1, b2, 0)
      south = (a1 + 1, b1, a2 + 1, b2, 0)
      east = (a1, b1 + 1, a2, b2 + 1, 0)
      west = (a1, b1 - 1, a2, b2 - 1, 0)
      For i = north, south, east, west:
            If map[i.a1, i.b1] is not an open spot: Reset a1 and b1
            If map[i.a2, i.b2] is not an open spot: Reset a2 and b2
            If Thing 1 and Thing 2 are both outside or both inside the map:
                  If both Thing 1 and Thing 2 are outside the map:
                        solutionDepth = head.depth
                  Else if visited[i] == true:
                        queue[end] = i
                        i.depth = head.depth + 1
                        visited[i] = true
                        end++
      beg++
Return solutionDepth
```

c) Proof of Correctness

This algorithm will check every possible combination of positions that Thing 1 and Thing 2 can move into, in the order that they can move into them. By using BFS, the first time a solution is encountered is guaranteed to be the shortest path to get to that solution. By keeping track of which configurations have already been visited, no duplicates will counted. Thus, as long as the algorithm checks that each new configuration follows the rules of movement before adding it to the BFS queue, it is guaranteed to find the shortest path to a solution, if a solution is possible.

d) Running Time Estimate

$O((ab)^2)$

e) Running Time Estimate Reasoning

The running time for this algorithm can be broken down as follows:

- Initializing the visited array --> $O(a * b * a * b)$
- Initializing the map matrix --> $O(a * b)$
- Other initiallization --> $O(1)$
- BFS --> $O((ab)^2)$

Everything else --> O(1)

Regarding the running time for BFS: In the worst case scenario, the while loop will run once for each possible configuration, and there are (ab)^2 possible configurations (a*b possible spots for Thing 1, and a*b possible spots for Thing 2). The for loop within the while loop will always run at most 4 times, so it can be counted as constant time.

Thus, the final running time for this algorithm is O((ab)^2) + O(ab) + O(1) = O((ab)^2).