CS325 - Group Assignment 3
Group: 123gogogo
Group members:
Yu Chien, Lin
Xiaoru Chen
Ziwen Tong
2020-11-17

Description of our algorithm:

We solved this problem by using breadth first search algorithm. We consider all legal red token and blue token positions as the vertices of a graph. Based on the current positions of the two colored token, compute all possible locations of the two tokens for the next layer. When we find the red token switch from the upper left corner to the blue's bottom right corner, we stop the calculation and return the shortest path (the number of moves). If we are unable to continue exploring new (unmarked) or legal vertices based on the current positions of red and blue tokens, and we still haven't found the final red and blue position swap from the upper left corner and the bottom right corner, then the input matrix has no solution, returns -1.

STEP 1:

Read from the input file, get variable ***n*** and put the matrix into a Two-Dimensional array ***board***.

STEP 2:

Variables:
n: Matrix's side length.
state: starts at the starting vertex of our graph; Stores the x and y value of the red token and the blue token. The first pair of values represents red token's x and y values on the matrix, the second pair of values represents blue token's x and y values. For example, [[0,0], [n-1, n-1]] is the starting vertex.

start_phase: represents the starting state of both red and blue token including their coordinates.
dest_phase: represents the destination state of both red and blue token including their coordinates.

queue: A bag that function as a queue data structure. First In First Out.
visited: A list that store vertices and tells all the stored vertices are marked.
counter: Record the moves we did.

push state (starting point) into bag
while the bag is not empty:
in range of the current phase size (number of possible moves of the current phase)
pop the vertex that entered first (pop each vertex in order) and put into ***visited*** (mark it)

case 1:
Red token moves up, down, left, or right, based on blue token's current square's value

There are up to 4 possible moves for red token. So we can generate up to 4 possible ***states.*** If the coordinates of the red token in the generated state are beyond the chessboard or lie on the same square with the blue token or marked already, discard this possible state (no push into the ***queue***), and push all the legal ***state(s)*** of the next phase into the bag (***queue***), ***counter***++. If all the legal ***state(s)*** of the next phase contain(s) the final vertex that red token and blue token swap position from the upper left corner and the bottom right corner, then we find the shortest path and stop computing and return the number of moves, ***counter***.

case 2:
Blue token moves up, down, left, or right, based on red token's current square's value

There are up to 4 possible moves for blue token. So we can generate up to 4 possible ***states.*** If the coordinates of the blue token in the generated state are beyond the chessboard or lie on the same square with the red token or marked already, discard this possible state (no push into the ***queue***), and push all the legal ***state(s)*** of the next phase into the bag (***queue***), ***counter***++. If all the legal ***state(s)*** of the next phase contain(s) the final vertex that red token and blue token swap position from the upper left corner and the bottom right corner, then we find the shortest path and stop computing and return the number of moves, ***counter***.

If the bag(***queue)*** is empty and we still cannot find the vertex that red token and blue token swap position from the upper left corner and the bottom right corner, then the matrix has no solution, return -1.

STEP 3:

Write out ***counter*** into an output file.

Running time analysis:

Time complexity of adding a node into the queue or popping a node from the queue is O(1).
Time complexity of marking a node as marked is O(1).

Time complexity of searching marked node: O(n)
The worst case:

$n \times n = n^2$ points on the board and there are two colored tokens.

Therefore, $T(n) = O((n \times n)^2 \times O(n)) = O(n^5)$

Correctness:

Base case:
After the first pop a state (vertex), push the legal next phase state(s) (vertices), and check if we find the destination, our ***counter*** = 0 + 1 = 1. Our count tells the current number of valid moves correctly.

Induction step:
After the k^{th} pop a state (vertex), push its legal next phase state(s) (vertices), and check if we find the destination, our count = k. The k value tells the current number of valid moves correctly.

If the bag is empty, we still cannot find the destination, and we have nothing to pop. The k value becomes -1; otherwise, return the current k value, and that is the correct number of valid moves we find to the end point.