Pseudocode

```
class Node{
      //false if no plank exists, true if there is a plank
      boolean up
      boolean down
      boolean left
      boolean right
      boolean visited
}
```

function SHORTESTPLANKPATH(*mazeArr*, *entrance*, *exit*) ▷ these inputs collectively tell about the size of the pillar grid and the plank layout
  check for invalid set size for *entrance* and *exit* ▷ entrance and exit are tuples containing the integer coordinate
  $minPath \leftarrow$ BFS on the current mazeArr ▷ path can be implemented as a list of integer tuples
  $toVisit \leftarrow$ a queue that stores the nodes to visit
  add the start node to the $toVisit$ queue
  while $toVisit$ is not empty **do**
    $node.visited \leftarrow true$
    for all direction not yet checked in node **do**
      $toVisitNode \leftarrow$ the node that the *direction* points to
      if the *direction* does not lead to node inside the *mazeArr* **then**
        continue
      if $toVisitNode.visited == true$ **then**
        continue
      if $direction == false$ **then**
        $mazeArrPlank \leftarrow$ copy the *mazeArr*
        $mazeArrPlank \leftarrow$ put plank down in the *direction*
        $plankPath \leftarrow$ BFS on that copy of *mazeArr*
        if *plankPath* != null **and** (*minPath* == null **or** length of *plankPath* < length of *minPath*) **then** ▷ BFS returns null if there is no path to the exit
          $minPath \leftarrow plankPath$
      else
        add $toVisitNode$ to the $toVisit$ queue
  return minPath

Proof of correctness
Loop Invariant
1. minPath contains the shortest path out of all the maze states using one plank formed so far (clarification: these are the copied maze with the plank put down).
2.all of the possible maze states using one plank are not examined yet.
conceptually, no path means that it is a path of infinite length, which is represented in this algorithm by having a null path.
Initialization
Basis
When the original maze (no plank put down) is traversed, it is the only path out of all the maze states using one plank formed so far. Thus it is trivially the shortest path.
Maintenance
Assume minPath contains the shortest path before the ith iteration.
At ith iteration, minPath is compared to plankPath (which is the path found in the new maze state using one plank) and the smaller one is put into minPath.
Termination
The loop terminates when there are no more possible maze states using one plank not yet examined.
Thus, minPath contains the smallest path out of all possible maze states using one plank.

Runtime analysis In the worst case, the entire maze grid has to be looked at, resulting in $O(n^2)$ for the loooking at the entire grid. The worst case running time for BFS is $O(E + V)$. There are n^2 amount of vertex in a n by n square grid. Also, there are $2n^2 - 2$ amount of edges in a n by n square grid. The addition os E+V is $2n^2 - 2 + n^2$, which is equal to $3n^2 - 2$. This means that the running time of BFS in a n by n square grid is $O(n^2)$.
Thus, the overall worst case running time for this loop is $O(n^4)$.