








Brief Description of the problem:

We are given $m \times n$ matrix which can have a number between 0 and 7. Each number represents a pipe with a shape as follows:

0	No Pipe
1	
2	
3	
4	
5	
6	
7	

Two pipes are considered connected if their end points connect. For e.g.

If matrix is as follows:

```
0040
1360
5000
```

Pipe 1 and 3 {1 opens to right. 3 opens to left} are connected. Other connected pipes are 3 and 6 (3 opens to right. 6 opens to left).

4 and 6 are not connected as 6 does not open to top, and 4 does not open to bottom.

1 and 5 are also not connected as even though 1 opens to bottom, 5 is not open to top.

Given this matrix, start point (X, Y) and length of probe tool "L", find out how many pipes {matrix elements} can be reached.

Key Steps to solve:

1. Find which of the neighboring pipe elements can be accessed from one starting point {Maintaining exhaustive list of checks to ascertain neighboring criteria as defined above}.
2. Find a method to process through the elements to find the next set of elements which can be accessed. {Traversing through the graph}. It should be noted that we wish to reach all nodes which are accessible from start node

with distance less than “L”. The focus is not on reaching lot of nodes, but on reaching them with lesser moves from start point (X, Y).

Approaches to the problem of traversal:

1) **Recursion:**

We mark all the nodes as unvisited and visited node count as 0. If a node marked as unvisited is processed during recursion, increment visited node count by 1 and mark it as visited.

We start by visiting Start node (X, Y) and then recursively visiting all its connected neighbors (based on checks for connected criteria as defined above) till the recursion depth of "L".

Drawback:

In case of highly connected matrix (almost all pipes are connected together and multiple ways to reach a pipe exists), we reach a node multiple times and process it multiple times which may lead to high running time. We can optimize it by checking that we process a node again only if we have reached it with a lower recursion level.

This is not a DFS solution because we reprocess a node, even if already visited, as it is possible that the new visit may update the visiting depth of the node to lower level and greater depth be reached after this node.

2) **Breadth-First Search:**

We mark all the nodes as unvisited and visited node count as 0. If a node marked as unvisited is processed during processing, increment visited node count by 1.

We start by pushing the start node (X, Y) into a queue with depth 1 and then start processing the queue till it is empty.

In each iteration a member of queue is taken out and its connected neighbors are analyzed. If neighbors are unvisited and depth of current element is not greater than L, connected elements are put into queue, marked visited and visited node count increased by 1.

As BFS performs level order traversal of nodes, it is not possible that a visited node be reached in a smaller distance. So, drawback of previous method cannot exist. This is the optimal solution for this problem.

Simplification possible in the solutions for easy coding:

Since there are 7 types of pipes, we will have to put multiple if conditions leading to nested if statements which are difficult to debug. It can be simplified by converting the 7 values into direction based data. For e.g. each value can be transformed into a structure with 4 Boolean variables, one for each direction. Alternatively, each number can be mapped to a 4 bit number where each bit represents a direction. After this input reduction, checks will become simpler and with less complexity.

Prepared by : Ravi Surana (ravi.surana)