FIND A PATH A RAT CAN TRAVEL THROUGH A MAZE

FIND A PATH A RAT CAN TRAVEL THROUGH A MAZE

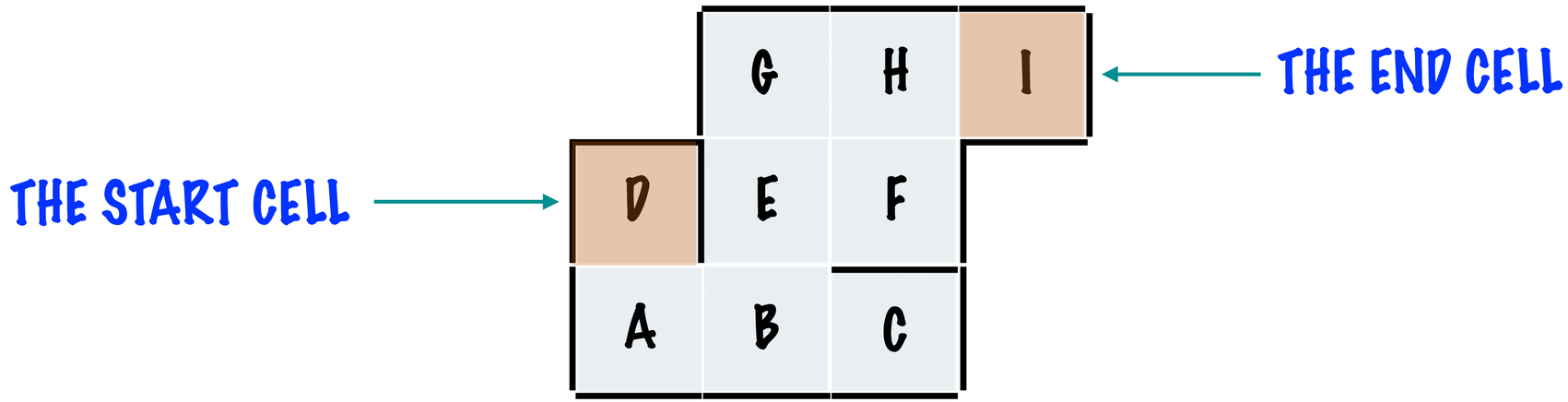
ASSUME THE MAZE IS MADE UP OF CELLS WITH 4 WALLS. EACH WALL CAN HAVE A DOOR

ALL WALLS DO NOT HAVE A DOOR, ONLY SOME DO

THE RAT HAS BEEN PLACED IN ONE OF THE CELLS, IT HAS TO MAKE IT'S WAY TO A CELL WHICH IS THE "END" CELL

THE RAT SHOULD NOT GO THROUGH THE SAME CELL TWICE WHILE FINDING A PATH

RAT AND MAZE



THERE ARE 2 POSSIBLE WAYS
TO NAVIGATE THE MAZE

D->A->B->E->G->H->I OR D->A->B->E->F->H->I

START THE RAT EXPLORING EACH DOOR IN TURN

KEEP TRACK OF THE CELLS IT HAS VISITED, DO NOT GO THROUGH TO A CELL IT HAS ALREADY VISITED IN ONE TRY

RECURSIVELY DO THIS TILL THE RAT GETS TO THE LAST CELL WHICH IS THE DESTINATION

WHAT IS THE BASE CASE?

1. THE RAT HAS REACHED THE END CELL

WHAT IS THE RECURSIVE CASE?

KEEP VISITING CELLS BY PASSING THROUGH DOORS WHEN THEY EXIST

DO NOT VISIT CELLS IF THEY ARE PRESENT IN THE CURRENT PATH, WE DON'T WANT THE RAT TO MOVE IN CIRCLES

A SINGLE CELL IN A MAZE

```java
public static class Cell {
    private String id;
    private boolean isEnd = false;

    private List<Cell> neighborList = new ArrayList<>();

    public Cell(String id) {
        this.id = id;
    }

    public String getId() {
        return id;
    }

    public List<Cell> getNeighborList() {
        return neighborList;
    }

    public void addNeighbor(Cell... neighbors) {
        for (Cell neighbor : neighbors) {
            neighborList.add(neighbor);
        }
    }

    public boolean isEnd() {
        return isEnd;
    }

    public void setIsEnd(boolean isEnd) {
        this.isEnd = isEnd;
    }

    public String toString() {
        return id;
    }
}
```

EACH CELL HAS AN ID AND A MARKER FOR WHETHER IT IS THE DESTINATION CELL FOR THE RAT

HOLD THE NEIGHBORS OF THE CELL

HELPER METHODS TO SET UP NEIGHBORS FOR TESTING

HELPER METHOD TO MARK A CELL AS THE DESTINATION CELL

FIND PATH THROUGH THE MAZE

```
public static boolean findPath(Cell current, List<Cell> currentPath) {
    currentPath.add(current);
    if (current.isEnd()) {
        return true;
    }

    for (Cell neighbor : current.getNeighborList()) {
        if (!currentPath.contains(neighbor)) {
            List<Cell> neighborPath = new ArrayList<>();
            neighborPath.addAll(currentPath);

            if (findPath(neighbor, neighborPath)) {
                currentPath.clear();
                currentPath.addAll(neighborPath);
                return true;
            }
        }
    }

    return false;
}
```

THE CURRENT CELL THE RAT IS AT, THIS IS THE START CELL IN THE FIRST CALL

KEEP TRACK OF THE CURRENT PATH THE RAT HAS TRAVERSED

VISIT EACH NEIGHBOR ONE BY ONE TO SEE IF THEY CAN LEAD TO THE DESTINATION CELL, ENSURE THE CURRENT PATH DOES NOT ALREADY INCLUDE THE NEIGHBOR

RECURSIVELY CALL FINDPATH TO GET TO THE DESTINATION

RETURN TRUE IF THE PATH IS FOUND

THE COMPLEXITY DEPENDS ON THE NUMBER OF NEIGHBORS EACH CELL HAS, IT'S NOT A SIMPLE NUMBER

IT'S DIRECTLY IMPACTED BY THE CONNECTIVITY OF THE MAZE