COP 3503 Homework #4: Drones Deliver to Senior Design Groups

Filename: drones.java
Time Limit: 3 seconds (per input case)
Standard Input, Standard Output

Computer Science is so hard these days that students in a senior design group have decided to work on a project to have drones fly them dinner while they are hard at work on their coursework. Naturally, the senior design group would like to offer this feature for multiple groups of people, each of who may be located in different places and might want food from different places.

The group did a great job with their drone design. It flawlessly picked up food and could fly that food to the appropriate location. Unfortunately, they only built one remote control for the (upto) four drones to deliver food. That means that even if best path for two drones is different, the drones are limited to moving in the same direction each time. Of course, if this were always true, then the drones would rarely be able to complete all of their deliveries. Luckily, however, some areas of the sky are marked as "no-fly zones." Thus, if a drone is directed to move into one of these no-fly zones, instead of moving, it will stay put. This means that with just one controller, different drones might end up moving different paths. Also, once a drone successfully delivers food to its senior design group, it will ignore all future commands and stay put.

We can model the map the drones will be navigating as an 8 by 8 grid, where some squares are marked as no-fly zones, some squares are the starting spots for the drones (this is the restaurant from which they pick up food), and some squares are the locations of the senior design groups:

```
__ __ __ __ __ __ __ G3
__ __ D2 __ D1 __ __ __
__ __ __ __ __ __ XX XX
__ __ __ __ G1 __ __ __
__ __ G2 __ __ __ __ __
__ __ __ __ __ __ D3 __
__ __ __ __ __ __ __ __
__ __ __ __ __ __ XX __
```

In the grid above, there are three drones delivering meals to three senior design groups. The drones are numbered D1 through D3 and the senior design groups are numbered G1 through G3. The drone Dx is delivering food to the group Gx.

The remove control allows the drones to move north (up), south (down), east (right) and west (left), a single grid square. For example, if we hit the down button twice, the new state of our grid is

```
__ __ __ __ __ __ __ G3
__ __ __ __ __ __ __ __
__ __ __ __ __ __ XX XX
__ __ D2 __ A1 __ __ __
__ __ G2 __ __ __ __ __
__ __ __ __ __ __ __ __
__ __ __ __ __ __ D3 __
__ __ __ __ __ __ XX __
```

After two moves, the first drone has successfully delivered its food. (This is denoted by A1 for arriving above.) The second drone has also moved down two squares and is one square away from its destination. The third drone has moved down only one square because when the remote told it to go down the second time, the third drone saw that the square below was a no-fly zone.

After a third move down, the second drove delivers its food:

```
__ __ __ __ __ __ __ G3
__ __ __ __ __ __ __ __
__ __ __ __ __ __ XX XX
__ __ __ __ A1 __ __ __
__ __ A2 __ __ __ __ __
__ __ __ __ __ __ __ __
__ __ __ __ __ __ D3 __
__ __ __ __ __ __ XX __
```

From here, the shortest route for the third drove to deliver its food takes 9 moves. (One such sequence is up, up, up, left, up, up, up, right, right.) In total using this set of button presses, it takes 12 button presses to get all three drones to successfully deliver its food.

Some other notes:

1) The entire 8 by 8 grid is surrounded by no-fly zone squares (not drawn in but just assumed to be there.)

2) Two drones may occupy the same grid square at the same time so long as it's not a no-fly zone square.

3) A drone treats each group's location except for the one it's delivering to as a no-fly zone. It would be unfair to tempt students in the wrong group with food and then just fly through.

The Problem:

Given the initial state of the 8 by 8 grid, showing the initial locations of the drones, the locations each of the drones must deliver their food and the no-fly zone squares, determine the fewest number of remote control button presses necessary to have all of the drones successfully deliver their food. If this is impossible, determine that the task is not possible.

The Input:

The first line of input contains a single integer, n ($1 \leq n \leq 4$), representing the number of drones for the input case. The corresponding drones are labeled D1 through Dn and the corresponding groups to which they are delivering food are labeled G1 through Dn.

The following eight lines each contain 8 space separated strings of two characters each, representing that row of the input grid. If the two characters are "__", the grid square is an open square. If the two characters are "Dk" where k is a digit in between 1 and n, then that represents drone k's starting position. If the two characters are "Gk" where k is a digit in between 1 and n,

then that represents the destination drone k is delivering food. Finally, if the two characters are "XX", that means the corresponding grid square is a no-fly zone. (These are capital letter X's.)

The Output:

Output a single integer: -1 if it's impossible to get the drones to all make their delivering or a positive integer representing the fewest number of remote control button pushes necessary to get the drones to all make their deliveries.

Sample Input	**Sample Output**
3 __ __ __ __ __ __ __ G3 __ __ D2 __ D1 __ __ __ __ __ __ __ __ __ XX XX __ __ __ __ G1 __ __ __ __ __ G2 __ __ __ __ __ __ __ __ __ __ __ D3 __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ XX __	12
4 G2 __ __ __ __ __ __ G1 __ __ __ __ __ __ __ __ __ __ __ __ __ __ XX XX __ D3 __ __ __ D4 G3 G4 __ __ __ D1 __ __ __ __ __ XX D2	26
2 __ __ __ __ __ __ __ D2 __ XX G2 XX __ __ __ __ __ __ __ XX __ __ __ __ __ __ __ XX __ __ __ __ __ D1 __ XX __ __ __ G1 __	-1

Implementation Requirements

The appropriate algorithm to solve the problem is a breadth first search. The difficulty is storing each possible state of the board. Note that since the location of the groups doesn't ever change, we can keep those as "fixed" variables (class variables in Java is fine) that are not part of the state.

The key information we need to know about the state is the location of each drone. Each drone has a row number (0 to 7) and a column number (0 to 7). We can store a row number using three bits and a column number using three bits. Thus, the position of one drone can be stored as a single integer in between 0 and 63, which takes up 6 bits. If we have n drones, we can simply use a single integer that has 6n bits. Thus, if we have four drones, an integer which uses the 24 least significant bits (in between 0 and 2^{24}-1) suffices to completely store the state of the board.

For example, for the first sample, the first drone is at row 1, column 4, which is equal to position 1 x 8 + 4 = 12, the second drone is at row 1, column 2 (position = 1 x 8 + 2 = 10), and the third drone is at row 5, column 6 (position = 5 x 8 + 6 = 46). This means the integer storing the initial position of the board is:

(46 << 12) + (10 << 6) + 12 = 189068

In binary, this number is:

101110001010001100

Drone 1 is highlighted in purple, drone 2 is highlighted in blue and drone 3 is highlighted in yellow. (It's more natural to think about them as drone 0, drone 1 and drone 2.) Notice that the bits in purple, when split into groups of three convert to (1, 4), the bits in blue when split into 3 convert to (1, 2) and the bits in yellow when split into groups of 3 convert to (5, 6). So, the BFS then starts at "board position" 189068 for the first sample.

In the BFS, for each possible board state (integer), we'll store the "distance", number of remote control moves, from the initial position to that board state. Since we can make an array of size 2^{24}, we'll store our distances in an array of size 2^{6n}, where n is the number of drones for the input case. Thus, distance[189068] = 0 in the beginning and distance[x] = -1 for all other values of x in the beginning of the BFS for the first sample input. Eventually, the distance array will fill with values 1, 2, 3, etc. When the final board position of (7 << 12) + (34 << 6) + 28 = 30876 is reached, the BFS can return the corresponding shortest distance.

What To Submit

For this assignment, please submit a single Java program named **`drones.java`** which solves the posted problem.