Spring 2024 Computer Science 2 Program #2: Hexagram

I may take some of our assignments from past programming contests, such as I am doing for this program. Since it is easiest, I will use the .pdf of the problem from the contest (separately posted) and then supplement it with a document like this, which explains what I would like students to submit and the format for students to use. This will usually be fairly consistent from assignment to assignment, but please do read the accompanying document for each assignment just in case there are some changes.

Reading Input/Producing Output

Please use standard input and standard output!!!
Your grade **for this assignment** will be based on the number of test cases that are correctly solved within the time limit. Since there's a good chance that some programs may take too long on individual cases, it's best to process each case one at a time.

For most assignments, a setup like this would work:

```
public static void main(String[] args) {
    Scanner stdin = new Scanner(System.in);
    int numCases = stdin.nextInt();
    for (int loop=1; loop<=numCases; loop++) {

        // Declare other variables here.
        // Read input for one case.
        // Process information.
        // Output answer for one case.
    }
}
```

For this assignment, you have 12 sentinel values that end the input. So something like this is more appropriate:

```
public static void main(String[] args) {
    Scanner stdin = new Scanner(System.in);
    int[] puzzle = new int[12];
    for (int i=0; i<12; i++) puzzle[i] = stdin.nextInt();
    while (!zero(puzzle)) {

        // Declare other variables here.
        // Read input for one case.
        // Process information.
        // Output answer for one case.
        for (int i=0; i<12; i++) puzzle[i] = stdin.nextInt();
    }

}
```

In general, just read the prompt and follow the directions. Some assignments (like P1) will have one input case per file, some will have multiple cases sentinel controlled (like P2), some will list the number of cases (like the Sudoku and Tentaizu examples from class), and some will be controlled by EOF (Recitation Program 1). It's essential that students learn to navigate various input formats on their own. But everything in this class will be **standard input/standard output.**

How to Test Your Code

1. Type up a sample input file. (Strongly suggested to type your own beyond what is provided.)
2. Test on the command line as follows:

```
>>java hexagram < myinput.in > myoutput.out
```

3. Either visually inspect the contents of myoutput.out to see if it matches the results you expect, or, if you know what the answers should be, store those in a separate file, say, `correct.out` and then use any file comparison tool to see if the files are the same or not. In windows, you can type fc at the command line:

```
fc myoutput.out correct.out
```

In a Unix system like Eustis, you can type:

```
diff -w myoutput.out correct.out
```

Note: The names of the files can be anything you want. Both fc and diff just compare the contents of the two files that are given to them as inputs.

4. If there are differences, inspect them more carefully using a tool such as WinMerge.

Implementation Requirements

This assignment is testing backtracking. Your code should step through, location by location in the star, through the 12 locations and try all of the possible numbers that are not yet filled in.

Clearly, trying all possibilities leads to 12! possible options, which would take too long to evaluate. Thus, in some instances, you may be able to decide that even though a number hasn't been placed yet, it's impossible for it to be in the current slot and for the puzzle to be solvable.

Hint: There is a way to determine the magic sum before placing any numbers!

If you code the problem appropriately, then any individual puzzle should be solved in under a second. Our input has 80 test cases. The total time limit will be set to three times my (Arup's) solution.

What To Submit

For this assignment, please submit a single Java program named **`hexagram.java`** which solves the posted problem. This program is sizeable enough that it should have multiple methods. The structure of it should be similar to other backtracking programs shown in class.