CSCI 3005 – SPRING 2019 – PROGRAMMING ASSIGNMENT – Dynamic Programming

Due to heavy rains and widespread flooding, you are stuck indoors. To help pass the time, your friend convinces you to play a game he/she just discovered. The game is as follows: starting with a pile of n tokens, a player removes tokens from the pile until only one token is left, subject to the following rules:

- It is always possible to remove exactly one token per move
- If the number of tokens in the pile is divisible by 2, you can remove half of the tokens
- If the number of tokens in the pile is divisible by 3, you can remove two-thirds of the tokens

The objective of the game is to reduce the number of tokens to 1 in as few moves as possible. In spite of your best efforts, it seems as if your opponent is consistently achieving the goal in fewer moves than you. Since he/she is constantly looking at his/her phone, you suspect that your opponent is getting some help. So you decide to take matters into your own hands by writing a program that determines the optimal sequence of moves. After some thought, you realize that the number of moves required can be defined recursively:

Let steps(n) be the minimum number of steps required to reduce a pile of n tokens, then:

steps(1) = 0

steps(n) = 1 + min(steps(n – 1), steps(n / 2), steps(n / 3)), for n > 1

Clearly, the steps(n / 2) and steps(n / 3) cases need to be considered only if n is divisible by 2 and 3, respectively. After coding the recursive routine, you realize that it takes too long, mainly because of the many overlapping recursive calls required. Therefore, you decide to try a dynamic programming approach. When using this approach, your solution should maintain an n-element array, so that the array element at location k contains the value of steps(k). The program can then compute and store the values of steps(k) from k = 1 to n without the need to recalculate partial results.

Your assignment is to write a Java class named **MinSteps** which provides public methods with the following signatures and functionality:

```
MinSteps(int n)     // initializes a game with n initial tokens
int recSolution()   // computes minimum number of steps required recursively
int dpSolution()    // computes minimum number of steps using dynamic programming
String getMoves()   // returns sequence of moves required (see format below)
```

You should initially test your methods using the `MinStepsTest` class provided. Here is a sample sequence of method calls and return values:

```
MinSteps game = new MinSteps(7);
game.recSolution();          // returns 3
game.dpSolution();           // returns 3
game.getMoves();             // returns "7 --> 6 --> 3 --> 1"
```

Once you are satisfied that your solution works, the source code for **MinSteps.java** should be submitted to Mimir for testing.