COP 3503 - Programming Assignment # 6

Dynamic Programming Hopscotch
Assigned: April 8, 2014 (Tuesday)
Due: April 22, 2014 (Tuesday) at 11:55 PM WebCourses time

Problem: This game, with a superficial similarity to the coin-row problem in Sec. 8.1 (and Exercise 2 on page 290), of your textbook, is a game of optimization played on a row of squares. A game of *order n*, has squares numbered from *0* to *n*. The player starts at square *n*, and her goal is to get to square *0* with the *lowest* score possible.

Moving obeys different rules depending on the number of the square the player is on.

1. On any square, the player has the option to move forward exactly one square, i.e., from square i to i-1 . This adds 1 to the score.
2. If the number of the square is a prime greater than 10, you may move forward by a number of squares equal to the units digit of the number. For example, a player can move from square 37 to square 37-7 = 30. This adds 3 to the score.
3. If the number of the square is a multiple of 11, the player may move forward by a number equal to the sum of the digits of the number. So from square 44, one can move to 44-(4+4) = 44-8 = 36. This adds 4 to the score.
4. If the number of the square is a multiple of 7, the player may move forward by exactly 4 squares. So from square 21, one can move to 21-4 = 17. This adds 2 to the score.

Note that any square could be in multiple cases at the same time – for example 77 can apply rules 1, 3 and 4. It is up to the player to decide which move they want to use.

You must write a dynamic programming solution that, given *n*, will determine the minimum score possible. Put this in a file called Hopscotch.java.

Note that it may be possible to do this using a greedy approach, but for the purposes of this assignment, you must come up with a DP approach.

Hint:
Define F(k) to be the minimum score if you start from square k. Now look at the (up to four) possible choices you have for your move. Express the value of F(k) in terms of the F-score for the squares you would land on if you made each possible valid move, keeping in mind that you need the smallest possible score.

Once you have this function figured out, you can simply write a recursive implementation and memoize it. Don't forget to define your base cases! Alternatively, you can simply write the iterative version using an array to hold the F-values. Either approach is acceptable.

Input File Format (hopscotch.in)
The first line of the input file will contain a single positive integer, T, representing the total number of cases in the file. Each subsequent line will contain an integer n, meaning that game is of order n.

Output Format (standard output)
For each game, you must print a line of the form Game #k: , where k is the 1-based index of the game in the input. This should be followed by a space and a single integer representing the lowest possible score when playing that game. See Sample I/O for examples.

Sample Input
5
1
16
7
39
98

Sample Output
Game #1: 1
Game #2: 12
Game #3: 5
Game #4: 21
Game #5: 48

Example explanation
The game of order 39 can be won with a score of 21, following the sequence 39-38-**37**-30-**29**-20-**19**-10-9-8-**7**-3-2-1. The numbers in bold are the squares from which we do a non-trivial move (something other than just moving one square forward). 37, 29 and 19 are prime, and we choose that move there. 7 is a multiple of 7 and we use it to jump 4 squares.

Similarly, for order 16, the best sequence is 16-15-**14**-10-9-8-**7**-3-2-1, and we use the multiple of 7 rule at 14 and 7.