2 Optimal Moves in Snake and Ladder Game

Snake and Ladders is a traditional board game that is played by two or more players on a square board with a grid of $n \times n$ cells. The objective of the game is to reach the final cell, which is numbered n^2, before the other players. Each player takes turns rolling a dice to move their game piece along the board. If a player lands on a ladder, they need to advance to a higher-numbered cell, while landing on a snake sends them back to a lower-numbered cell.

In this question, we consider a slightly different problem. Here, you are given a board which consists of 'N' cells and 'M' snakes and ladders. Your objective is to find out the optimal number of moves required to complete the game by a single player.

You should start at cell 1 of the board. In each move, starting from cell '$curr$', choose a destination vertex '$next$' with a label in the range $[curr + 1,\ \min(curr + 6,\ N)]$. If your '$next$' destination has a snake or ladder, you must move to the destination of that snake or ladder. If that destination also has a snake or ladder, you must move to it's destination (you must follow the concatenated sequence of snakes and ladders until you reach a cell where no snake or ladder begins). The game ends when you reach the cell labeled N.

You are given a class `SnakesLadder` which has the following class variables:

- `int N`: Total number of cells on the board.
- `int M`: Total number of snakes and ladders on the board.
- `int snake[]`: An integer array of the length 'N' which will be used to store the endpoints of snakes on the board. For a snake starting at index 'i' and ending at index 'j', $snakes[i] = j$. If no snake starts at cell 'i', $snake[i] = -1$.
- `int ladder[]`: An integer array of the length 'N' which will be used to store the endpoints of ladders on the board. For a ladder starting at index 'i' and ending at index 'j', $ladders[i] = j$. If no ladder starts at cell 'i', $ladder[i] = -1$.

Your task is implement the following four class functions:

- `public SnakeLadder()`: Initializes a board game with 'N' cells and 'M' snakes and ladders.
- `public int OptimalMoves()`: Returns the optimal number of moves required to win the snake and ladder game.
- `public int Query(x, y)`: Returns $+1$ if adding the snake or ladder from x to y will improve the optimal solution, else returns -1.
- `public int[] FindBestNewSnake()`: Finds the best snake, if it exists, whose addition to the board will improve the optimal number of moves by highest possible value.

We next explain these functions in detail.