No Chutes, Just Ladders

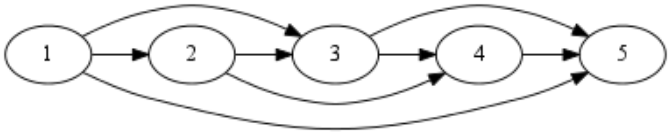
2016-10-20 Thu

A children's game has n spaces numbered $1, \ldots, n$. A game piece can progress from one space to the next or—in certain spaces marked with ladders—it can "jump" forward a fixed number of spaces.

You are given an array A of the spaces on the board, where each entry of A is the set of spaces from which a piece can arrive at that space. So, using 1-based indexing, $A[1] = \{\}$ because pieces start at space 1 but cannot move to there from anywhere. For every other index i of a space, $i - 1 \in A[i]$ because we can always arrive at i from space $i - 1$. To illustrate ladders, if space j had a ladder of length 3 and a ladder of length 7 leading to it, its set would be $A[j] = \{j - 1, j - 3, j - 7\}$.

Your goal is to count how many different ways there are to arrive at the final space.

For example, consider this gameboard first as a graph:



and then as an array: $[\{\}, \{1\}, \{1, 2\}, \{2, 3\}, \{1, 3, 4\}]$.

There is only 1 way to reach node 2 (from node 1). There are 2 ways to reach node 3 (via 2 or directly from 1). There are 3 ways to reach node 4 (via nodes 2 and 3, via just node 2, or via just node 3). There are 6 ways to reach the final space, node 5.

Design an efficient algorithm to count how many different ways there are to arrive at the final space. Your algorithm may use linear time and "linear" memory, where we assume that the count of ways to reach a node can be stored in constant space.

HINT: Start by designing a recurrence $C(n)$ that describes the number of ways to reach space n in terms of the number of ways to reach previous spaces on the board and in terms of $A[n]$ (the set of spaces from which space n is reachable). Then, either convert that into a recursive solution and memoize it or convert that into a dynamic programming solution.