Daily Coding Problem #245

Problem

This problem was asked by Yelp.

You are given an array of integers, where each element represents the maximum number of steps that can be jumped going forward from that element. Write a function to return the minimum number of jumps you must take in order to get from the start to the end of the array.

For example, given `[6, 2, 4, 0, 5, 1, 1, 4, 2, 9]`, you should return 2, as the optimal solution involves jumping from 6 to 5, and then from 5 to 9.

Solution

Let's look at the example array above. Note that for our first jump, we can land anywhere from 1 to 6 steps forward. For each of these jumps, there will be several options for the next jump, depending on the value of the element we land on. If we enumerate every one of these options, and keep track of the number of jumps each one takes, we could then return the minimum number of jumps over all paths.

This leads to a recursive solution. Our base case will be if we are at the end of the array, or if we have jumped past it. If either of these is true, we return the number of jumps taken so far. Otherwise, we look at every possible jump we can take from the current location, and call our function again with this new starting point and an incremented number of jumps

call our function again with this new starting point and an incremented number of jumps. Once all the subproblems have been solved, we return the path that uses the smallest number of jumps.

One thing to be careful about is if an element is zero: in this case, we cannot move forward at all, so we will say it takes an infinite number of jumps.

```
def min_jumps(arr, jumps=0):
    if not arr or len(arr) == 1:
        return jumps

    if arr[0] == 0:
        return float("inf")

    moves = [min_jumps(arr[i:], jumps + 1) for i in range(1, min(arr[0] + 1, len(arr)))]

    return min(moves)
```

Unfortunately, at each index `i` we can branch off into possibly `N - i - 1` subproblems, so the time complexity here is `O(N!)`.

Since we are revisiting these subproblems again and again, we can improve this with dynamic programming.

First, we will create an array, `moves`, that stores the optimal number of moves to reach the end of the array, starting from any index. Initially this will be infinite for all indices except for the last, which will be zero. Then, starting from the penultimate element and going backwards, if we find that a jump can provide a shorter path from that element to the end, we update the value of `moves[element]`.

After traversing through the array, the value of the first element will give us our answer.

```
def min_jumps(arr):
    moves = [float("inf") for _ in range(len(arr) - 1)] + [0]

    for i in range(len(arr) - 1, -1, -1):
        if arr[i] != 0:
            for jump in range(i + 1, min(i + 1 + arr[i], len(arr))):
                moves[i] = min(moves[i], 1 + moves[jump])

    return moves[0]
```

```
return moves[0]
```

Since we must check up to `N` possibilities for each element, the time complexity will be $O(N^2)$.