DS102 - Discussion 9

Wednesday, 6th November, 2019

1. Assume you have a rod that has length n inches. You also have a list of prices $p_1, p_2, \ldots, p_n$. Where p_i is the price of a rod of length i. Furthermore assume that you are able to cut your original rod every inch.

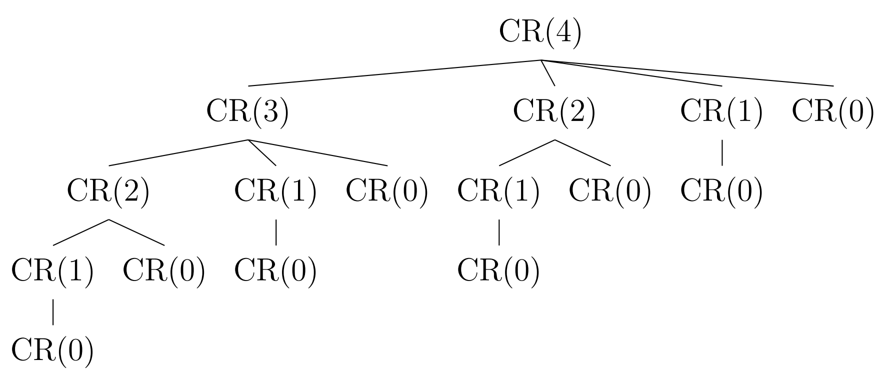
   (a) Write a naive recursive algorithm to find the maximum value you can get by cutting up your rod and selling the pieces.

   Solution:

```
def cut_rod(prices, n):
    if n == 0:
        return 0
    max_price = max([prices[i] +
                     cut_rod(prices, n-i-1)
                     for i in range(n)])
    return max_price
```

   (b) Assuming that $n = 4$, draw the recursion tree for the naive solution.

   Solution:



   (c) What is the time complexity of the naive solution?

   Solution: Assuming that calling the function with argument n results in $T(n)$ total subcalls, we see that $T(n) = \sum_{i=0}^{n-1} T(i)$. By induction we then see that the time complexity is $O(2^n)$.

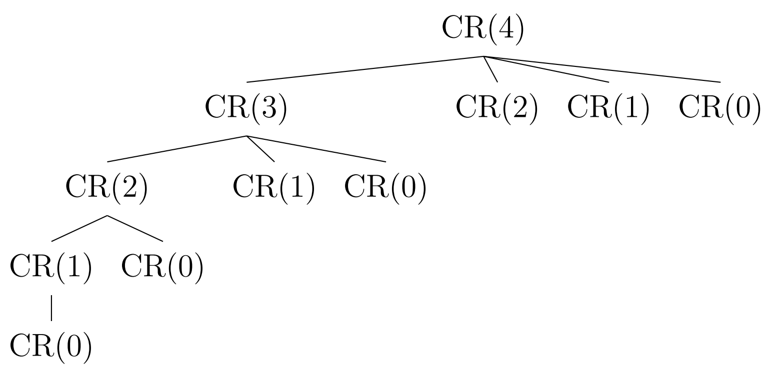
   (d) Augment the above algorithm to use dynamic programming to solve the problem faster.

   Solution: We see that we have optimal substructure since we can find the maximum price by first computing the maximum price for rods of length less than n. Furthermore we have overlapping subproblems since we can reuse our computation of the best price rods of length less than k for a rod of length $k+1$. Hence we can augment the above solution as follows

```
def cut_rod(prices, n, cache):
    val = cache.get(n)
    if val is not None:
        return val
    if n == 0:
        return 0
    max_price = max([prices[i] +
                     cut_rod(prices, n-i-1, cache)
                     for i in range(n)])
    cache[n] = max_price
    return max_price
```

(e) Assuming that $n = 4$, draw the recursion tree for the dynamic programming solution.

Solution:



(f) What is the time complexity of the dynamic programming solution?

Solution: The first call to $CR(k)$ will perform k calls, subsequent calls will just immediately return. Hence $CR(n)$ will perform n calls total, $CR(n-1)$ will perform $n-1$ calls etc. Hence we will perform $n + n - 1 + \ldots + 2 + 1$ calls in total. Which means our time complexity is $O(n^2)$.

2. Assume that we have the following gridworld

			1
	X	S	-100

where S represents our starting point, X is a square we can't access and the 1 and -100 cells represent terminal states with corresponding rewards.

(a) Recall that the optimal value function is defined as

$$V^*(s) = \max_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^*(s') \right],$$

where $T(s, a, s')$ is the probability of transitioning from state s to state s' given action a, $R(s, a, s')$ is the reward function, and γ is the discount factor.

Assuming state transitions are deterministic, meaning that an action in a specific direction always moves us in that direction (unless it's toward the X square in which case we remain stationary), write down the optimal value function at each cell when the discount factor γ is 0.9.

Solution:

0.9^2	0.9	1	N/A
0.9^3	X	0.9	N/A
0.9^4	0.9^3	0.9^2	0.9^3

(b) Recall that the optimal Q-function is given by

$$Q^*(s, a) = \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^*(s') \right].$$

Compute the Q-function at our starting point for the actions of going up, down, left, and right.

Solution: Going up gives us a Q-value of 0.9, going left gives us a Q-value of 0.9^2, going down gives us a Q-value of 0.9^3, and going right gives us a Q-value of -100.

(c) Based on the Q-function you just derived, what would be the optimal move for you to make at the given starting point?

Solution: You should go up since you want to maximize your Q-function.

(d) Now suppose the state transitions are stochastic such that you have a 0.8 probability of going in the direction you specified and a 0.1 probability of going in either direction that is perpendicular to the one you specified. For example if you decide to go up you will have a 0.8 probability of going up, a 0.1 probability of going left, and a 0.1 probability of going right. Given this stochastic state transition what is the best action to perform at the starting point?

Solution: You should go left. While that does make you stay on the same spot with probability 0.8 you will have a probability of 0 of landing in the bad final state. In contrast if you chose the faster option of trying to move up you might land in the bad final state.