Homework1_432001358

January 23, 2022

0.1 22 SPRING CSCE 629 600: ANALYSIS OF ALGORITHMS - Homework 1

0.1.1 Name: Rohan Chaudhury

0.1.2 UIN: 432001358

0.1.3 Question (1) Textbook page 370, Exercise 15.1-2.

Show, by means of a counterexample, that the following "greedy" strategy does not always determine an optimal way to cut rods. Define the density of a rod of length i to be p_i/i, that is, its value per inch. The greedy strategy for a rod of length n cuts off a first piece of length i, where 1 $\leq$ i $\leq$ n, having maximum density. It then continues by applying the greedy strategy to the remaining piece of length n - i.

0.1.4 Answer:

Let us consider the following example:

We have a rod of length n=5 and the following prices given for cuts of different lengths:

Length of cut	Price for the cut	Density
1	1	1
2	8	4
3	14	4.66
4	20	5

The densities (as defined in the question) for the different cuts is calculated and shown in the table above. Using the greedy strategy, it would first cut a length of i=4 first since it has the highest density of 5. Then we are left with a length of rod i=1 after cutting i=4 from n=5. So the total price we get after using greedy strategy is:

Total price= 5*4 + 1*1 = 21

However the optimal cut would be to cut the rod n=5 into lengths of 2 and 3 which would give the total price as:

Optimal total price = 14+8 = 22

Hence, this serves as a counterexample to show that the given "greedy" strategy does not always determine an optimal way to cut rods.

0.1.5 Question (2) Textbook page 370, Exercise 15.1-3.

Consider a modification of the rod-cutting problem in which, in addition to a price pi for each rod, each cut incurs a fixed cost of c. The revenue associated with a solution is now the sum of the prices of the pieces minus the costs of making the cuts. Give a dynamic-programming algorithm to solve this modified problem.

0.1.6 Answer

We are considering the bottom-up approach algorithm for optimal rod cutting from the book given on page 366 to design our new algorithm.

r[j] or r_j gives the optimal price to cut a rod of length j.

p[i] or p_i is the price given in the problem to cut a rod of length i

In the original algorithm we have to modify line 6 (which is q = max(q, p[i] + r[j - i])) to include the fixed cost 'c' since cutting the rod each time now incurs a fixed cost of c. So now the price for each cut becomes p_i - c and line 6 in the modified algorithm becomes q = max(q, p[i] -c + r[j - i]). That is, we are including the fixed cost '-c' when we are considering the first cut p[i] which is added to the optimal value for cutting the rod of length j-i (given by r[j-i]). In this case, when we vary i from 1 to j and take maximum of all the prices obtained, we get optimal value for cutting the rod of length j (given by r[j]). Initially we have assumed r[0]=0 as price obtained after cutting a rod of length 0 is 0. The modified algorithm is as follows:

Pseudocode for MODIFIED BOTTOM-UP-CUT-ROD ALGORITHM (p, n)

```
1 let r[0..n] be a new array
2 r[0] = 0
3 for j = 1 to n
4     q= -∞
5     for i = 1 to j
6         q = max(q, p[i] -c + r[j - i])
7     r[j] = q
8 return r[n]
```

The Recursive Function is as follows:

$r_n = max_{1<=i<=n} \{p_i - c + r_{n-i}\}$

Using this recursive function to calculate the optimal price after cutting rods from length 1 to n (in bottom-up manner):

$r_0 = 0$ (initially assumed as price obtained after cutting a rod of length 0 is 0)

$r_1 = p_1 - c$

$r_2 = \max\{p_1 - c + r_1, p_2 - c + r_0\}$

$r_3 = \max\{p_1 - c + r_2, p_2 - c + r_1, p_3 - c + r_0\}$

r_4 = max{p_1 - c + r_3, p_2 - c + r_2, p_3 - c + r_1, p_4 - c + r_0}

.

.

r_n = max{p_1 - c + r_{n-1}, p_2 - c + r_{n-2}, p_3 - c + r_{n-3}, . . . , p_n - c + r_0}

Proof of correctness:

We are varying j from 1 to n (in line 3) to calculate and store the optimal value obtained after cutting rods of length 1 to n respectively. To do that, in each iteration considering rod of length j, we are varying i from 1 to j (in line 5) to consider all the possible ways the rod can be cut first in length i (given by p[i] and with the added term '-c') and adding that price to the optimal value to cut the rest of the rod of length j-i which is computed in the previous iterations of j (given by r[j-i]). Maximum of all such values gives the optimal value for cutting the rod of length j (given by r[j]). At the end r[n] gives our required result of the optimal price to cut a rod of length n. Thus this algorithm is searching for all the possible ways to cut the rod from length 1 to n and storing the optimum values to be reused in later computation. Since all the possible ways are exploited, the algorithm returns the best possible way to cut the given rod to get the optimum price.

Time complexity of the algorithm:

The variable j in the for loop varies from 1 to n and variable i in the for loop varies from 1 to j. In the worst case, when j=n, the variable i varies from 1 to n. Thus the worst case time complexity would be: $O(n * n) = O(n^2)$