Problem Set 6

October 20, 2015

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 pi/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 $1 \leq i \geq n$, having maximum density. It then continues by applying the greedy strategy to the remaining piece of length $n - i$.

Solution

Consider a rod of length 4 with its density according to the size of cuts.

length i	1	2	3	4
price p_i	1	22	39	40
p_i/i	1	11	13	10

According to greedy strategy, We first cut the rod of length 3 for price of 39 because it has the maximum density of 13. It leaves a rod of length 1 for price 1. The total price is 40. But, according to dynamic programming the optimal way is to cut into two rods of length 2, giving a total of 44. Hence, "greedy" strategy does not always determine an optimal way to cut rods.

15.1-5

The Fibonacci numbers are defined by recurrence $F_0 = 0$, $F_1 = 1$ and $F_i = F_{i-1} + F_{i-2}$. Give an $O(n)$ time dynamic-programming algorithm to compute the nth Fibonacci number. Draw the subproblem graph. How many vertices and edges are in the graph?

Solution

Algorithm to compute n^{th} Fibonacci number :

```
1  Declare an array of integers F of size n + 1
2  F[0] = 0;
3  F[1] = 1;
4  for( i = 2; i <= n; i++){
5        F[i] = F[i-1] + F[i-2];
6  }
7  return F[n];
```

This directly implements the recurrence relation of the Fibonacci numbers. Each number in the sequence is computed from the previous two numbers in the sequence.The running time is clearly O (n).
The sub- problem graph consists of n+1 vertices, v_0 ,v_1 ,...,v_n.
For i = 2 ,3,....,n , Vertex v_i has two leaving edges: to vertices v_{i-1} and v_{i-2} . No edge leaves vertices v and v_1. Thus, the subproblem graph has $2(n-1)$ edges.

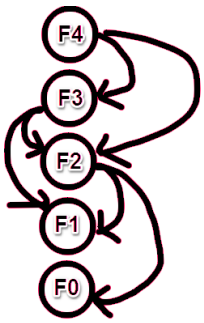

Figure 1: A subproblem graph for n = 4.

15.2-3

Use the substitution method to show that the solution to the recurrence

$$P(n) = \begin{cases} 1 & \text{for } n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{for } n \geq 2 \end{cases}$$

is $\Omega(2^n)$

Solution

Let us assume that $P(k) \geq c.2^k$ for all $k < n$. Let us prove that $P(n) \geq c.2^n$
For $n > 1$,

$$P(n) = \sum_{k=1}^{n-1} P(k)P(n-k)$$
$$\geq \sum_{k=1}^{n-1} c.2^k c.2^{n-k} = \sum_{k=1}^{n-1} c^2.2^n = (n-1)c^2.2^n$$
$$\geq c.2^n, \textit{provided that } (n-1)c > 1.$$

Because $n > 1$, there exists threshold n_0, and $c > n_0$ make $(n-1)c > 1$.
Hence, P(n) = $\Omega(2^n)$

15.3-4

As stated, in dynamic programming we first solve the subproblems and then choose which of them to use in an optimal solution to the problem. Professor Capulet claims that we do not always need to solve all the subproblems in order to find an optimal solution. She suggests that we can find an optimal solution to the matrixchain multiplication problem by always choosing the matrix A_k at which to split the subproduct $A_i, A_{i+1}...A_j$ (by selecting k to minimize the quantity $p_{i-1}p_kp_j$) before solving the subproblems. Find an instance of the matrix-chain multiplication problem for which this greedy approach yields a suboptimal solution

Solution

We can use a counter example to prove that the claim gives suboptimal solution.
Consider the case where A_1 is 5 * 6 , A_2 is 6*3, A_3 is 3 * 2. If we use her claim, then the result is $(A_1A_2)A_3 = 120$.
But we can group in another way, $(A_1(A_2A_3)) = 96$
So, her claim is not correct, it cant give us an optimal result.

15.4-1

Determine an LCS of <1,0,0,1,0,1,0,1> and <0,1,0,1,1,0,1,1,0>

Solution

	x_i	1	0	0	1	0	1	0	1
y_i	0	0	0	0	0	0	0	0	0
0	0	$\uparrow 0$	$\nwarrow 1$	$\nwarrow 1$	$\leftarrow 1$	$\nwarrow 1$	$\leftarrow 1$	$\nwarrow 1$	$\leftarrow 1$
1	0	$\nwarrow 1$	$\uparrow 1$	$\uparrow 1$	$\nwarrow 2$	$\leftarrow 2$	$\nwarrow 2$	$\leftarrow 2$	$\nwarrow 2$
0	0	$\uparrow 1$	$\nwarrow 2$	$\nwarrow 2$	$\uparrow 2$	$\nwarrow 3$	$\leftarrow 3$	$\nwarrow 3$	$\leftarrow 3$
1	0	$\nwarrow 1$	$\uparrow 2$	$\uparrow 2$	$\nwarrow 3$	$\leftarrow 3$	$\nwarrow 4$	$\leftarrow 4$	$\nwarrow 4$
1	0	$\nwarrow 1$	$\uparrow 2$	$\uparrow 2$	$\nwarrow 3$	$\uparrow 3$	$\nwarrow 4$	$\uparrow 4$	$\nwarrow 5$
0	0	$\uparrow 1$	$\nwarrow 2$	$\nwarrow 3$	$\uparrow 3$	$\nwarrow 4$	$\uparrow 4$	$\nwarrow 5$	$\uparrow 5$
1	0	$\nwarrow 1$	$\uparrow 2$	$\uparrow 3$	$\nwarrow 4$	$\uparrow 4$	$\nwarrow 5$	$\uparrow 5$	$\nwarrow 6$
1	0	$\nwarrow 1$	$\uparrow 2$	$\uparrow 3$	$\nwarrow 4$	$\uparrow 4$	$\nwarrow 5$	$\uparrow 5$	$\nwarrow 6$
0	0	$\uparrow 1$	$\nwarrow 2$	$\nwarrow 3$	$\uparrow 4$	$\nwarrow 5$	$\uparrow 5$	$\nwarrow 6$	$\uparrow 6$

By following the path of arrows from the bottom right cell, the Longest Common Subsequence is $< 0, 0, 1, 1, 0, 1 >$.