50.004 Introduction to Algorithms, Fall 2018
Singapore University of Technology and Design — 20 Nov, 2018
Gemma Roig, Ioannis Panageas, Subhajit Datta — Home Exercises - Week 11

Student Information

Name: Wang Tianduo Student ID: 1002963
Due Date: 27 Nov 11:59pm.
Submit answers on eDimension in pdf format. Submission without student information will **NOT** be marked! Any questions regarding the homework can be directed to the TA through email (contact information on eDimension).

Week 11

1. In comparison to other methods that do not take advantage of overlapping subproblems, dynamic programming takes far less time in finding a solution. [True/False]. Explain why.
   Only half of the full marks will be awarded if answer is correct without explanation.
   DP is fast because it remember the overlapping subproblems and reuse them to avoid unnecessary work.
2. What are the changes in time and space complexities when a top-down approach of dynamic programming is applied to a problem?

   A. Time and space complexities decrease.
   B. Time and space complexities increase.
   C. Time complexity increases and space complexity decreases.
   D. Time complexity decreases and space complexity increases.

   D

3. Modify the rod-cutting problem in Figure 1 to include for each cut, a fixed cost c in addition to the price p_i for each rod. The revenue is now the sum of the prices of the pieces minus the costs of making cuts. Show the modified dynamic programming algorithm.
   ***Hint**: Modify line 4, 5 and 6.*

```
BOTTOM-UP-CUT-ROD(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] + r[j - i])
7      r[j] = q
8  return r[n]
```

Figure 1: Rod-cutting algorithm

Modified-cut-Rod(p,n,c)
let r[0...n] be new array
r[0] = 0
for j=1 to n
  q = p[j]
  for i=1 to j-1
    q = max(q, p[i] + r[j-i] - c)
  r[j] = q
return r[n]

The original DP equation
$r_n = \max\{p_n, r_1+r_{n-1}, \ldots, r_{n-1}+r_1\}$

new:
$r_n = \max\{p_n, r_1+r_{n-1}-c, \ldots, r_{n-1}+r_1-c\}$

4. recursion + memoization can be called dynamic programming. If we simply use recursion, the same subproblems will be calculated many times, which is inefficient. So we use memorization to reuse the results that we have calculated before and save time

4. Dynamic programming algorithms are usually based on recursive equations. Why don't dynamic programming algorithms simply use recursion to implement those equations directly? Briefly explain.

5. Let P(n) be the number of n-length binary strings that do not have any three consecutive ones (i.e. they do not have "111" as a substring). For example:

   P(1) = 2: 0, 1,
   P(2) = 4: 00, 01, 10, 11
   P(3) = 7: 000, 001, 010, 011, 100, 101, 110
   P(4) = 13: 0000, 0001, 0010, 0011, 0100, 0101, 0110, 1000, 1001, 1010, 1011, 1100, 1101
   Hint: *Each binary string has either one of the following 3 properties: a) Last bit is 0, b) Last two bits are 01, c) Last three bits are 011.*

   (i) Write down the recursive formula to compute P(n).

   (ii) Suppose we implement a dynamic programming algorithm using the recursive formula in (i), what is the running time of the dynamic programming algorithm?

(i) $P(n) = P(n-1) + P(n-2) + P(n-3)$ when $n \geq 3$

Since when $n < 3$, '111' cannot exist

So let $P(0) = 1$, $P(1) = 2^1$, $P(2) = 2^2 = 4$

(ii) Assume that we use memoization method

Initiate a hash table with {0:1, 1:2, 2:4}.

```
def P(n):
    if n in table:
        x = table[n]
    else:
        x = P(n-1) + P(n-2) + P(n-3)
        table(n) = x
    return x
```

Since we use a hashtable to store the value of subproblem, we can solve each subproblem in O(1)

Therefore, total time complexity is $O(1) \cdot n = O(n)$

Here we assume that the time complexity of each addition is O(1)

Here, if we consider that the result increases exponentially as the input n increases linearly, the final answer will have O(n) digits. Thus, each addition requires O(n) time. With O(N) additions, this makes our algorithm $O(N^2)$