CS 4040/5040
Advanced Algorithms
Review Lecture : Review of Dynamic Programming

David Juedes
Ohio University Russ College of Engineering and Technology

December 3rd, 2017
As many of you may know, the term *programming* in dynamic programming does not refer to the modern use of the term (e.g., C++ programming), it refers to a *table based approach*. Nevertheless, it is often hard to see when dynamic programming can be used, and when it is not applicable. Here are some examples.
Introduction

Example #1

Problem (1)

Assume we have an array $A$ with $n$ numbers. Each number could be positive or negative. Find the indices $i$ and $j$ such that

$$j \sum_{l=i}^{j} A[l]$$

is maximized.

Example #1, cont’d

Example

Solve problem # 1 on the array

\[
A = \{1, -10, 5, -3, 7, -11, 15, -2, -3, -4, 15, -20, 1\}.
\]
The maximum sum is $21 = 15 + -2 + -3 + -4 + 15$. 
Solution #1: Brute Force

We can solve this problem with a brute force algorithm, given below.

```
(1) int max=0; int maxi=0; int maxj=0;
(2) for (int i=0;i<t.size();i++) {
(3)    for (int j=i+1;j<t.size();j++) {
(4)        int sum=0;
(5)        for (int k=i;k<=j;k++) {
(6)            sum+=t[k];
(7)        }
(8)        if (sum > max) {
(9)            max=sum; maxi=i; maxj=j;
(10)        }
(11)    }
(12) }
```
This algorithm is not dynamic programming. Also, it is slow. What’s the run time of this code? Off the cuff, the answer seems like it should be $O(n^3)$, where $n$ is the number of elements in the array. Now, to be precise, we really have to get into the nitty-gritty of what this code is actually doing.
Introduction

Analysis, cont’d

1. How long does the code on line 1 take?
2. How long does the code on line 4 take for a single iteration of the loop?
3. How long does the code on line 6 take for a single iteration?
4. How long does the code on lines 8-10 take for a single iteration?
5. Give a formula for the running time of the code on lines 5-7.
6. Give a formula for the running time of the code on lines 3–11.
7. Give a formula for the running time of the code on lines 2–12.
8. Simply the formulas give above in terms of $n$. 
Here are the answers:

1. $O(1)$.
2. $O(1)$.
3. $O(1)$.
4. $O(1)$.
5. $O(1)$.

\[ \sum_{k=i}^{j} O(1) = O(j - i + 1). \]

6. $\sum_{j=i+1}^{n-1} O(j - i + 1) + O(1) = f(n, i) = O\left(\frac{n^2 - n}{2} + \frac{i^2 + i}{2} - i \cdot n + 2n - 2i - 2 \right)$.

7. $\sum_{i=0}^{n-1} f(n, i) = O(n^3)$. 
\[ \sum_{i=0}^{n-1} f(n, i) = O\left( \sum_{i=0}^{n-1} \frac{n^2 - n}{2} + \frac{i^2 + i}{2} - i \cdot n + 2n - 2i - 2 \right) \]

\[ = O\left( \frac{n^3 - n^2}{2} + \frac{1}{2} \left( \sum_{i=0}^{n-1} i^2 + \sum_{i=0}^{n-1} i \right) + \sum_{i=0}^{n-1} (in + 2n - 2i - 2) \right) \]

\[ = O\left( \frac{n^3 - n^2}{2} + \frac{n(n-1)(2n-1)}{12} + \frac{n(n-1)}{4} \right. \]
\[ \left. + \frac{n^2(n-1)}{2} + 2n(n-1) - \frac{n(n-1)}{2} - 2(n-1) \right) \]

\[ = O(n^3). \]
Improving the code

We can improve the code given on page 6 by using the following simple observation.

**Observation**

$$\sum_{l=i}^{j} A[l] = \sum_{l=1}^{j} A[l] - \sum_{l=1}^{i-1} A[l]$$

**Exercise (1)**

*Rewrite the code on page 6 by using Observation #1. Perform the same analysis done on page 6-8. Your code should run in $O(n^2)$ steps.*

So, we can improve our code from $O(n^3)$ to $O(n^2)$ with a simple rewrite. But, we can do better. Dynamic programming will allow us to get down to $O(n)$ steps.
Dynamic Programming Approach

Here’s the “magic.” Consider the sub-arrays $A_i = A[1..i]$. We will find the maximum “length” subsequences in $A_i$. We will look at two cases, when the subsequences end at $A[i]$ and when they don’t.

Define

- $C_0[i] = \text{maximum over all } 1 \leq u \leq v < i \text{ of }$
  
  $$\sum_{l=u}^{v} A[l]$$

- and

- $C_1[i] = \text{maximum over all } 1 \leq u \leq i \text{ of }$
  
  $$\sum_{l=u}^{i} A[l]$$

.
Notice that the value of the maximum length subsequence is the maximum of $C_0[n]$ and $C_1[n]$.
Likewise, $C_0[1] = 0$, $C_1[1] = A[1],
\[
C_0[i] = \max\{C_0[i - 1], C_1[i - 1]\}, \quad \text{and} \\
C_1[i] = \max\{C_1[i - 1] + A[i], A[i]\}.
\]
**Problem**

*Use the optimal substructure of this problem to write a program that finds the maximum value of the maximum valued subsequence in $O(n)$ steps.*

**Problem**

*Use backtracking the optimal substructure of this problem to write a program that finds the indices of the maximum valued subsequence in $O(n)$ steps.*
Practice Problems

1. Project Euler, Problem #67
2. Project Euler, Problem #81
3. Project Euler, Problem #82