Computer Science Department,
College of Computer and Information Sciences,
King Saud University.

CSC 311: Design and Analysis of Algorithms[1]
Dr. Waleed Alsalih

5.2 Optimal substructure

Some computational problems maintain the **optimal substructure** which can be defined as follows: a solution to a problem contains (or depends on) solutions to subproblems. Such a property may be used as an indicator for the applicability of dynamic programming. This makes sense because dynamic programming is a bottom-up design technique where we start with a very small subproblem and based on solutions to smaller problems we find solutions to bigger problems.
Example:
Consider the following two problems and tell whether they maintain the optimal substructure property or not. Given an unweighted, directed graph, tell if the optimal substructure property applies to the following problems or not:

1. Finding the shortest path between a pair of nodes u and v.
2. Finding the longest simple path between a pair of nodes u and v.

Answer: It applies to the shortest path problem but does not apply to the longest path problem, why?

A general plan for dynamic programming algorithms:

1. Identify and understand the optimal substructure.
2. Find a way to solve subproblems based on solutions of smaller subproblems.

[1]This is a summary of the material we cover from the textbook: *Introduction to the Design & Analysis of Algorithms*, A. Levitin, Second Edition, Pearson Addison-Wesley, 2006.

Examples on dynamic programming algorithms

Maximum Sum Contiguous Subsequence: Design an algorithm that finds the maximum sum contiguous subsequence in an array of integers $A[0..n-1]$ (the array may have negative integers; otherwise, the answer is trivial).
Let $M[i]$ be the maximum sum contiguous subsequence ending at $A[i]$. To solve the maximum sum contiguous subsequence problem, we will need to find i such that $M[i]$ is maximized. Can we build the array $M[0..n-1]$ in a bottom-up fashion? How?
The algorithm below uses dynamic programming to find the maximum sum contiguous subsequence.

```
Algorithm MaximumSumContiguousSubsequence(A[0..n − 1])
M[0]:=A[0];
MAX:=0;
for i = 1 to  n − 1 do
    if M[i − 1] > 0 then
        M[i]:=M[i − 1] + A[i];
    else
        M[i]:=A[i];
    end
    if M[i] > M[MAX] then
        MAX:=i;
    end
end
return M[MAX];
```

It is obvious that this algorithm runs in $O(n)$ time.