For each of the following problems, design a dynamic programming algorithm. Follow the four steps:

(a) Describe (in words) the structure of an optimal solution (optimal substructure).

(b) Write a recurrence that defines the value of an optimal solution.

(c) Design an algorithm that computes the value of an optimal solution in a bottom-up fashion.

(d) Design an algorithm (and augment your existing bottom-up solution) that constructs an optimal solution from the computed information.

1. Problem 15-1 on page 404 (longest simple path in a DAG).

2. In CS 173 (Homework 12), you wrote a backtracking algorithm for the 0-1 knapsack problem. In this exercise, you will write a more efficient dynamic programming solution for the problem.

In the knapsack problem, you are given a list of items, each with a weight and a profit. The goal is to pack a knapsack with the most profitable set of items without exceeding the knapsack's weight capacity. For a dynamic programming algorithm to work, you will need to assume that the weights and capacity are positive integers.

The prototype for your function should look like:

```c
void Knapsack(const int w[], const int p[], int n, int capacity)
```

where `w` and `p` are the weights and profits of the items, respectively, `n` is the number of items, and `capacity` is the weight capacity of the knapsack. Use the same program skeleton that you used in 173 (still available on the web).