In the last few lectures we discussed a few problems that need dynamic programming to solve them efficiently. In this lecture, you will practice solving 2 examples on the problems we discussed in class. Moreover, you will get to practice solving a new problem from scratch using dynamic programming.

**The Knapsack problem - 15 minutes**

Last week, we discussed the Knapsack problem. We decided that computing an optimal solution involves solving two sub-problems, as shown in the recurrence equation below.

\[
OPT(j, w) = \max(v_j + OPT(j - 1, w - w_j), OPT(j - 1, w))
\]

Solve the problem given the following input, with a total weight \( W = 8 \).

<table>
<thead>
<tr>
<th>Item</th>
<th>Value (v)</th>
<th>Weight (w)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

Draw your table here, and identify the optimal value of the items packed into the knapsack and the corresponding set of items.
The Longest Common Subsequence (LCS) problem - 25 minutes

[Description from CMU course 15-451/651] The Longest Common Subsequence (LCS) problem is as follows. We are given two strings: string S of length n, and string T of length m. Our goal is to produce their longest common subsequence: the longest sequence of characters that appear left-to-right (but not necessarily in a contiguous block) in both strings.

For example, consider:
S = ABAZDC
T = BACBAD
In this case, the LCS has length 4 and is the string ABAD. Another way to look at it is we are finding a 1-1 matching between some of the letters in S and some of the letters in T such that none of the edges in the matching cross each other.

(1) Read the definition of the problem carefully, and discuss it with your group if you are not sure what it means.
(2) Find the recurrence that represents the relationship between subproblems in this problem.
(3) Test the recurrence on the example you have above, to check if it works. You can also try it on other examples that you might have created.
The Shortest Path problem - 20 minutes

Last week, we also discussed the Shortest Path problem. We decided that computing an optimal solution involves solving two sub-problems, as shown in the recurrence equation below.

\[ OPT(i, v) = \min(OPT(i - 1, v), \min_{w \in V}(OPT(i - 1, w) + c_{wv})) \]

Solve the problem given the following graph, with target node t.