Class Tutorial 1

Short review on DP - WikipediaDP

1. Rod Cutting (Knapsack variant)
(From Introduction to Algorithms)

A company buys long steel rods and cuts them into shorter rods. The price table for the shorter rods is as follows:

<table>
<thead>
<tr>
<th>Length n</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Price p_n</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>17</td>
<td>17</td>
<td>20</td>
<td>24</td>
<td>30</td>
</tr>
</tbody>
</table>

The cost of making a cut is zero. Given a (long) rod of length \( n \), the problem is how to cut it in order to maximize the revenue \( r_n \).

Example: \( n = 4 \)

(a) \[ \begin{array}{c}
9 \\
\end{array} \]
(b) \[ \begin{array}{c}
1 \\
8 \\
\end{array} \]
(c) \[ \begin{array}{c}
5 \\
5 \\
\end{array} \]
(d) \[ \begin{array}{c}
8 \\
1 \\
\end{array} \]

(e) \[ \begin{array}{c}
1 \\
1 \\
5 \\
\end{array} \]
(f) \[ \begin{array}{c}
1 \\
5 \\
1 \\
\end{array} \]
(g) \[ \begin{array}{c}
5 \\
1 \\
1 \\
\end{array} \]
(h) \[ \begin{array}{c}
1 \\
1 \\
1 \\
1 \\
\end{array} \]

a. The trivial solution: enumerate all possibilities. How many different cuts exist for a rod of length \( n \) ?

b. A recursive solution: Given the maximal revenues \( r_1, \ldots, r_{n-1} \) compute the revenue \( r_n \).

Write down a recursive algorithm for the problem, and compute its time and space complexity.

c. Now making a cut costs \( c \). Modify the algorithm for this case.

Solution:

a. \( 2^{n-1} \), since each segment boundary can be cut or not.

b. \( r_n = \max_{1 \leq i < n} \left( p_i + r_{n-i} \right) \), since each cut can be viewed as a composition of a piece of length \( i \), and all the other pieces.
Proof: Assume exists a revenue \( \bar{r} \) such that \( \bar{r} > r_n \). This would mean that for any \( i \in \{1, \ldots, n - 1\} \) it holds that \( \bar{r} - p_i > r_{n-i} \), which is a contradiction. We assumed that for any \( i \in \{1, \ldots, n - 1\} \), \( r_{n-i} \) is the maximal revenue.

The complexity time of the algorithm is \( O(n^2) \) and \( O(n) \) space algorithm.

2. Longest Common Subsequence
(From Introduction to Algorithms)

Given a sequence \( X = \langle x_1, \ldots, x_m \rangle \), we say that the sequence \( Z = \langle z_1, \ldots, z_k \rangle \) is a subsequence of \( X \) if there exists a strictly increasing sequence \( \langle i_1, \ldots, i_k \rangle \) such that for all \( j = 1, \ldots, k \) we have \( X_{i_j} = Z_j \). For example, \( Z = \langle B, C, D, B \rangle \) is a subsequence of \( X = \langle A, B, C, B, D, A, B \rangle \).

Given two sequences \( X, Y \) we say that \( Z \) is a common subsequence of \( X \) and \( Y \) if \( Z \) is a subsequence of both \( X \) and \( Y \). In the longest-common-subsequence (LCS) problem we are given two sequences \( X = \langle x_1, \ldots, x_m \rangle \) and \( Y = \langle y_1, \ldots, y_n \rangle \), and we need to find the maximum length common subsequence of \( X \) and \( Y \).

a. Warm-up: find the LCS of \( X = \langle A, B, C, B, D, A, B \rangle \) and \( Y = \langle B, D, C, A, B, A \rangle \).

b. Brute-force algorithm: enumeration of all subsequences. How many subsequences does \( X \) have? What is the complexity of such an algorithm?

c. Let \( X_i \) denote the \( i \)'th prefix of \( X \) : \( X_i = \langle x_1, \ldots, x_i \rangle \). Prove the following theorem:
Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$ be sequences, and let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of $X$ and $Y$.

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and $Z_{k-1}$ is an LCS of $X_{m-1}$ and $Y_{n-1}$.
2. If $x_m \neq y_n$, then $z_k \neq x_m$ implies that $Z$ is an LCS of $X_{m-1}$ and $Y$.
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that $Z$ is an LCS of $X$ and $Y_{n-1}$.

**d. Dynamic programming algorithm:** let $c[i, j]$ denote the length of the LCS of $X_i$ and $Y_j$. Write a recursive formula for $c[i, j]$. Derive an algorithm for the length of the LCS of $X$ and $Y$. What is its complexity?

**e. (Homework) derive the actual LCS from $c[i, j]$.**

**Solution:**

a. For example, $\langle B, C, B, A \rangle$ or $\langle B, D, A, B \rangle$.

b. $2^m$

c.

**Proof**

1. If $z_k \neq x_m$, then we could append $x_m = y_n$ to $Z$ to obtain a common subsequence of $X$ and $Y$ of length $k + 1$, contradicting the supposition that $Z$ is a longest common subsequence of $X$ and $Y$. Thus, we must have $z_k = x_m = y_n$. Now, the prefix $Z_{k-1}$ is a length-$(k-1)$ common subsequence of $X_{m-1}$ and $Y_{n-1}$.

We wish to show that it is an LCS. Suppose for the purpose of contradiction that there exists a common subsequence $W$ of $X_{m-1}$ and $Y_{n-1}$ with length greater than $k - 1$. Then, appending $x_m = y_n$ to $W$ produces a common subsequence of $X$ and $Y$ whose length is greater than $k$, which is a contradiction.

2. If $z_k \neq x_m$, then $Z$ is a common subsequence of $X_{m-1}$ and $Y$. If there were a common subsequence $W$ of $X_{m-1}$ and $Y$ with length greater than $k$, then $W$ would also be a common subsequence of $X_m$ and $Y$, contradicting the assumption that $Z$ is an LCS of $X$ and $Y$.

3. The proof is symmetric to (2).

**d. Based on the previous theorem, we have**

$$c[i, j] = \begin{cases} 
0 & \text{if } i = 0 \text{ or } j = 0, \\
[c[i - 1, j - 1] + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j, \\
\max(c[i, j - 1], c[i - 1, j]) & \text{if } i, j > 0 \text{ and } x_i \neq y_j.
\end{cases}$$

An $O(mn)$ algorithm:
LCS-L (X, Y)
1 m = X.length
2 n = Y.length
3 let b[1..m, 1..n] and c[0..m, 0..n] be new tables
4 for i = 1 to m
5 c[i, 0] = 0
6 for j = 0 to n
7 c[0, j] = 0
8 for i = 1 to m
9 for j = 1 to n
10 if x_i == y_j
11 c[i, j] = c[i - 1, j - 1] + 1
12 b[i, j] = "\n"
13 elseif c[i - 1, j] ≥ c[i, j - 1]
14 c[i, j] = c[i - 1, j]
15 b[i, j] = "↑"
16 else c[i, j] = c[i, j - 1]
17 b[i, j] = "←"
18 return c and b