(i) PG Problems 395, 396, and 397. Here are some possible answers.

395: \( s_1 = 3, s_2 = 2, s_3 = 2 \), and \( S = 4 \).

396: \( s_1 = 1, s_2 = 2, s_3 = 2 \), and \( S = 4 \).

397: \( s_1 = 3, s_2 = 2, s_3 = 2 \), and \( S = 4 \).

(ii) DPV Problem 6.11.
As the hint suggested, we want to compute \( L[i, j] = \) the length of the longest common subsequence of \( x_{1..i} = x_1 x_2 \ldots x_i \) and \( y_{1..j} = y_1 y_2 \ldots y_j \). \( L[0, j] = L[i, 0] = 0 \) since in those cases one of the strings is empty. Suppose \( i > 0 \) and \( j > 0 \) and we know \( L[i - 1, j], L[i - 1, j] \), and \( L[i, j - 1] \). The formula for \( L[i, j] \) is:

\[
\begin{cases}
\max(L[i, j - 1], L[i - 1, j]), & \text{if } x_i \neq y_j; \\
\max(L[i, j - 1], L[i - 1, j], 1 + L[i - 1, j - 1]), & \text{if } x_i = y_j
\end{cases}
\]

The idea is that if \( x_i \neq y_j \), then we cannot use both \( x_i \) and \( y_j \), so the best we can do is the max \( L[i, j - 1] \) (the length of the l.c.s. of \( x_{1..i} \) and \( y_{1..j} \) that doesn’t use \( y_j \)) and \( L[i - 1, j] \) (the length of the l.c.s. of \( x_{1..i} \) and \( y_{1..j} \) that doesn’t use \( x_i \)). If \( x_i = y_j \), then we also have the possibility of using \( x_i \) and \( y_j \) at the end of the l.c.s. of \( x_{1..(i-1)} \) and \( y_{1..(j-1)} \).

So here is the algorithm, which clearly runs in \( O(m \cdot n) \) time.

(iii) DPV Problem 6.18.
To make change for \( u \) with denominations 1 through \( i \) you have the choice of using a denomination \( i \) coin or not. If you use the coin, then you have to make change for \( u - x_i \) using denominations 1 through \( i - 1 \). If you don’t use the coin, then you have to make change for \( u \) using denominations 1 through \( i - 1 \). So the equation for \( C[i, u] \) is:

\[
C[i, u] = \begin{cases}
C[i - 1, u] \text{ or } C[i - 1, u - x_i], & \text{if } x_i \leq u; \\
C[i - 1, u], & \text{if } x_i > u.
\end{cases}
\]

So we can compute \( C[n, v] \) by as above by changing the if-statement to:

\[
\begin{aligned}
\text{if } x_i & \leq u \\
\text{then } C[i, u] & \leftarrow C[i - 1, u] \text{ or } C[i - 1, u - x_i] \\
\text{else } C[i, u] & \leftarrow C[i - 1, u]
\end{aligned}
\]

(iv) DPV Problem 6.20.
See page 90 in PG.

(v) DPV Problem 6.21.
Here is a high-level version of a non-DP algorithm:

1. If the tree has fewer than 2 nodes, then \( \emptyset \) works as the cover.
2. If the tree has 2 nodes (i.e., 0–2), then pick one of the endpoints for the cover.
3. Suppose the tree has at least 3 vertices. Every tree has to have at least two leaves. So for each edge joining a leaf to an interior node:

   (a) put the interior node in the cover,

   (b) delete all covered edges from the tree (and all isolated vertices), and

   (c) solve the cover problem for the resulting tree.

To implement this, first represent the tree with an adjacency list. We do an initial depth-first search to find all the leaves. For each leaf, 3a is \( \Theta(1) \) time. We do 3b once per edge. Moreover, when we 3b, we make a list of the leaves of the new tree. Since each vertex eventually becomes a leaf in this process, it follows that this all costs \( \Theta(|V| + |E|) \) time.