15.3-2 Because there is no overlapping subproblems, and the value of each node of the recursion tree is only determined by one calculation.
15.3-3 Yes, similar to the minimization case.
15.3.4 Consider the case of $n = 3$, $p_0 = 4, p_1 = 3, p_2 = 2, p_3 = 1$. The greedy algorithm will select $k = 2$, which yields total number scalar multiplication of 32, but if $k = 1$, the total number scalar multiplication is 24.
15-1
Let $p[v_i] = \{s, v_0, v_1, \cdots, v_i\}$ be the set of vertex in the longest weighed path from s to v_i, and $w[v_i]$ be the total weight of that path, then we have the recurrence

$$w[v_i] = \max_{(v,v_i)\in E}(w[v] + u(v, v_i)), w[s] = 0$$

Suppose $v^* = \underset{(v^*,v_i)\in E}{argmax}(w[v^*] + u(v^*, v_i))$, then we update the path

$$p[v_i] = p[v^*] + \{v_i\}$$

15-3
Firstly, we identify the unique leftmost and rightmost point v_1 and v_n. Let $p_R[v_i] = \{v_0, v_1, \cdots, v_i\}(1 \leq i \leq n)$ be the set of vertex in the longest weighed path from v_0 to v_i, and $w_R[v]$ be the total weight of that path. Similarly, we define $p_L(v_i) = \{v_n, v_1, \cdots, v_i\}(1 \leq i \leq n)$ be the set of vertex in the longest weighed path from v_n to v_i, and $w_L[v]$ be the total weight of that path. For the left-to-right procedure, the recurrence can be written as follows:

$$w_R[v_i] = \max_{v.x<v_i.x}(w_R[v] + d(v, v_i)), w_R[v_0] = 0$$

Suppose $v^* = \underset{v^*.x<v_i.x}{argmax}(w_R[v^*] + d(v^*, v_i))$, then we update the path

$$p_R[v_i] = p_R[v*] + \{v_i\}, p_R[v_0] = v_0$$

For the right-to-left part, it's similar, but we also need to check whether a point has been visited in the left-to-right part. Formally,

$$w_L[v_i] = \max_{v.x>v_i.x, v\notin p_R[v_n]}(w_L[v] + d(v, v_i)), w_L[v_n] = 0$$

Suppose $v^* = \underset{v^*.x>v_i.x, v^*\notin p_R[v_n]}{argmax}(w_L[v^*] + d(v^*, v_i))$, then we update the path

$$p_L[v_i] = p_L[v*] + \{v_i\}, p_L[v_n] = \{\}$$

The final result is $p_R[v_n] + p_L[v_0]$
15-5

a. Let $c[i][j]$ be the minimal cost when we are at $x[i]$ and $y[j]$, then we have

$$c[i][j] = min \begin{cases} c[i-1][j-1] + cost(copy) \\ c[i-1][j-1] + cost(replace) \\ c[i-1][j] + cost(delete) \\ c[i][j-1] + cost(insert) \\ c[i-2][j-2] + cost(twiddle) \\ c[i-1][j] + cost(kill) \end{cases}$$

$$c[0][0] = 0,$$

$$c[m+1][n+1] = \min_i(c[m+1][n+1], c[i][n+1] + cost(kill))$$

b. Simply use copy, twiddle and insert (only space), and find the maximal score. So $c[m+1][n+1]$ is the edit distance, and the operational sequence can be recorded accordingly. Both the space and time complexity are $O(n^2)$.

15-7

a. simply do a BFS from v_0, move from v_{i-1} to $v_i (1 \leq i \leq k)$ if and only if $\sigma_i(v_{i-1}, v_i) \in E$. If no v_k found, then return NO-SUCH-PATH; otherwise, return $\{v_0, v_1, \cdots, v_k\}$

b. Let prefix $s_i = \langle \sigma_1, \sigma_2, \cdots, \sigma_i \rangle$, and define subproblem $w[i][v]$ be the *probability* of the most probable path from v_0 to v that has the label s_i. We have the following recurrence

$$w[i][v] = \max_{\sigma_i(u,v) \in E} w[i-1][u] \cdot p(u, v),$$

$$w[0][v_0] = 1,$$

$$w[0][v] = 0 \quad (v \neq v_0),$$

(initialize the whole w array to 0 except for $w[0][v_0]$.)

Let $v^* = \mathbf{argmax}_v w[k][v]$. You can use a backpointer to reconstruct the optimal path corresponding to $w[k][v^*]$. Complexity: $O(k(|V| + |E|))$.

Liang's note: This "Viterbi algorithm" is a specific instance of the general "Viterbi algorithm" we taught in class. Here the graph G' of subproblems $w[i][v]$ can be viewed as "k copies of the original graph G", where each edge in G' is of the form $((i-1, u), (i, v))$, and thus G' must be acyclic (even if G itself is cyclic). As discussed in class, the general Viterbi algorithm simply follow a topological sort on G' and update the solutions to subproblems.