Problem Set 2

Problem 1 (20 points, Problem 15.2-1). Find an optimal parenthesization of a matrix-chain product whose sequence of dimensions is < 5, 10, 3, 12, 5, 50, 6 >.

Solution. We have \( p_0 = 5, p_1 = 10, p_2 = 3, p_3 = 12, p_4 = 5, p_5 = 50, p_6 = 6 \).

From the algorithm, we have

\[
m[i, j] = \begin{cases} 0 & \text{if } i == j \\ \min\{m[i, k] + m[k + 1, j] + p_{i-1}p_kp_j\} & \text{if } i \leq k < j \end{cases}
\]

Thus the optimal matrix chain product is 

\[
A_{1}(A_{2}((A_{3}A_{4})A_{5}A_{6})).
\]

Problem 2 (30 points, Problem 15.7). Viterbi algorithm

We can use dynamic programming on a directed graph \( G = (V, E) \) for speech recognition. Each edge \((u, v) \in E\) is labeled with a sound \( \sigma(u, v) \) from a finite set \( \Sigma \) of sounds. The labeled graph is a formal model of a person speaking a restricted language. Each path in the graph starting from a distinguished vertex \( v_0 \in V \) corresponds to a possible sequence of sounds produced by the model.

We define the label of a directed path to be the concatenation of the labels of the edges on that path.

a. Describe an efficient algorithm that, given an edge-labeled graph \( G \) with distinguished vertex 0 and a sequence \( s =< \sigma_1, \sigma_2, ..., \sigma_k > \) of sounds from \( \Sigma \), returns a path in \( G \) that begins at \( v_0 \) and has \( s \) as its label, if any such path exists. Otherwise, the algorithm should return NO-SUCH-PATH. Analyze the
Now, suppose that every edge \((u, v) \in E\) has an associated nonnegative probability \(p(u, v)\) of traversing the edge \((u, v)\) from vertex \(u\) and thus producing the corresponding sound. The sum of the probabilities of the edges leaving any vertex equals 1. The probability of a path is defined to be the product of the probabilities of its edges. We can view the probability of a path beginning at \(v_0\) as the probability that a "random walk" beginning at \(v_0\) will follow the specified path, where we randomly choose which edge to take leaving a vertex \(u\) according to the probabilities of the available edges leaving \(u\).

b. Extend your answer to part (a) so that if a path is returned, it is a most probable path starting at \(v_0\) and having label \(s\). Analyze the running time of your algorithm.

**Solution.** a. We define a \(k \times n\) matrix \(m\) as follows:

\[
m[i, j] = \begin{cases} 
1 & \text{if there is a path from } v_0 \text{ to } v_j \text{ with a sequence of sounds } <\sigma_1, ..., \sigma_i> \text{ as its label} \\
0 & \text{otherwise}
\end{cases}
\]

The matrix can be calculated as follows:

\[
m[i, j] = \begin{cases} 
1 & \text{if there is an edge } (v_h, v_j) \text{ s.t. } \sigma(v_h, v_j) = \sigma_i \text{ and } m[i-1, h] = 1 \\
0 & \text{otherwise}
\end{cases}
\]

Pseudocode:

The running time is \(O(kn^2)\).

b. Let \(f[i, j]\) denote the maximum probability of a path from \(v_0\) to \(v_j\) with <\(\sigma_1, ..., \sigma_i\)> as its label.

\[
f[i, j] = \max_{h} s.t. \sigma(v_h, v_j) = \sigma_i \{f[i-1, h]p(v_h, v_j)\}
\]

The pseudocode is similar to (a). The time complexity is \(O(kn^2)\).

**Problem 3** (30 points, Problem 16.1-3). Not just any greedy approach to the activity-selection problem produces a maximum-size set of mutually compatible activities. Give an example to show that the approach of selecting the activity of least duration from among those that are compatible with previously selected activities does not work. Do the same for the approaches of always selecting the compatible activity that overlaps the fewest other remaining activities and always selecting the compatible remaining activity with the earliest start time.

**Solution.** Counter example for always selecting the activity with the least duration:

Let \((s_1, f_1) = (3, 5), (s_2, f_2) = (1, 4), (s_3, f_3) = (4, 7)\). The greedy solution is \{activity 1\}. The optimal solution is \{activity 2, activity 3\}.

Counter example for always selecting the task with fewest overlaps: Let \((s_1, f_1) = (3, 5), (s_2, f_2) = (0, 2), (s_3, f_3) = (6, 8), (s_4, f_4) = (2, 4), (s_5, f_5) = (4, 6), (s_6, f_6) = (1, 3), (s_7, f_7) = (1, 3), (s_8, f_8) = (5, 7), (s_9, f_9) = (5, 7)\). The Greedy solution is \{activity 1, activity 2, activity 3\}, optimal solution is \{activity 2, activity 3, activity 4, activity 5\}.  

2
Algorithm 1 Psedo-code for Problem 2(a)

Input: $G = (V, E)$, $s$

Output: a path with label $s$ or "NO-SUCH-PATH"

1: let $m[i, j] = 0$ $(i = 1, ..., k; j = 0, ..., n - 1)$
2: for $j = 0$ to $n - 1$ do
3:   if $\sigma(v_0, v_j) == s_1$ then
4:     let $m[1, j] = 1$
5:   end if
6: end for
7: for $i = 2$ to $k$ do
8:   for $j = 0$ to $n - 1$ do
9:     if there is an edge $(v_h, v_j)$ s.t. $\sigma(v_h, v_j) = s_i$ and $m[i - 1, h] == 1$ then
10:        let $m[i, j] = 1$
11:     end if
12: end for
13: end for
14: let $path = ""$
15: for $j = 0$ to $n - 1$ do
16:   if $m[k, j] == 1$ then
17:     $path = path + v_j$
18:     let $v_b = v_j$
19:     for $i = k$ downto $2$ do
20:       if there is an edge $(v_a, v_b)$ s.t. $\sigma(v_a, v_b) = \sigma_i$ and $m[i - 1, a] == 1$ then
21:         let $v_b = v_a$
22:         let $path = path + v_a$
23:       end if
24:     end for
25:     let $path = path + v_0$
26:     return $path$
27: end if
28: end for
29: return "NO-SUCH-PATH"

Counter example for always selecting task with earliest start time: Let $(s_1, f_1) = (1, 5), (s_2, f_2) = (2, 3), (s_3, f_3) = (4, 5)$. The Greedy solution is \{activity 1\}. Optimal solution is \{activity 2, activity 3\}. 


Algorithm 2 Pseudo-code for Problem 2(b)

Input: $G = (V, E), s, p$
Output: a path with label $s$ or "$NO$-SUCH-PATH$"

1: let $f[i, j] = 0$ (i = 1, ..., k; j = 0, ..., n - 1)
2: for $j = 0$ to $n - 1$ do
3:   if $\sigma(v_0, v_j) == s$ then
4:     let $f[1, j] = p(v_0, v_j)$
5:   end if
6: end for
7: for $i = 2$ to $k$ do
8:   for $j = 0$ to $n - 1$ do
9:     $f[i, j] = \max_h$ s.t.$\sigma(v_h, v_j) = \sigma$, \{ $f[i - 1, h]p[v_h, v_j]$ \}
10: end for
11: end for
12: let $path = "\"
13: let $i = -1, pr = 0$
14: for $j = 0$ to $n - 1$ do
15:   if $f[k, j] > pr$ then
16:     let $pr = f[k, j]$
17:     let $i = j$
18: end if
19: end for
20: if $i == -1$ then
21:   return "$NO$-SUCH-PATH$"
22: end if
23: let $path = path + v_i$
24: for $j = k$ down to 2 do
25:   find h s.t. $f[j - 1, h]p(v_h, v_i) == f[j, i]$ and $\sigma(v_h, v_i) == \sigma_j$
26:   let $v_i = v_h$
27:   let $path = path + v_h$
28: end for
29: let $path = path + v_0$
30: return path