Problem 1. (10 points)

You are given a directed graph $G$ with non-negative integer edge weights, a pair of vertices $s$ and $t$, and integers $K$ and $W$. Describe a dynamic-programming algorithm for deciding whether there exists a path from $s$ to $t$ that has total weight $W$ and uses exactly $K$ edges. Your algorithm should run in time $O((n + m)WK)$, where $n$ is the number of vertices and $m$ is the number of edges in $G$.

Answer:

Let’s define $P[v, w, k]$ to be true if there is a path from $s$ to $v$ that has total weight $w$ and uses exactly $k$ edges. (For any vertex $v$ and any integers $w$ and $k$ with $0 \leq w \leq W$ and $0 \leq k \leq K$.)

The following recurrence holds:
- $P[v, 0, 0]$ is true for each vertex $v$.
- $P[v, w, 0]$ is false for $w > 0$ and each vertex $v$.
- For $k > 0$, $P[v, w, k]$ is true if and only if $P[u, w - w(u, v), k - 1]$ is true for some edge $(u, v)$.

Here is the corresponding algorithm:

Algorithm FindPath $(G, s, t, K, W)$

1. set $P[v, 0, 0] \leftarrow$ true for each vertex $v$
2. set $P[v, w, 0] \leftarrow$ false for each vertex $v$ and $w = 1, 2, \ldots, W$
3. for $k \leftarrow 1, 2, \ldots, K$ do
   for $w \leftarrow 0, 1, 2, \ldots, W$ do
      set $P[v, w, k] \leftarrow$ true if there is an edge $(u, v)$
      such that $P[u, w - w(u, v), k - 1] = $ true (for each vertex $v$)
   return $P[t, W, K]$

The outer loop executes $K$ times, the inner loop executes $W$ times (for each iteration of the outer loop), and implementing the innermost set takes linear time. The complexity is $O((n + m)WK)$

Problem 2. (10 points)

Consider the following data compression technique. We have a table of $m$ text strings, each of length at most $k$. We want to encode a data string $D$ of length $n$ using as few text strings as possible. For example, if our table contains $(a, ba, abab, b)$ and the data string is bababbaababa, the best way to encode it is $(b, abab, ba, abab, a)$ - a total of five code words. Give and $O(nmk)$ dynamic-programming algorithm to find the length of the best encoding. You may assume that the string has an encoding in terms of the table.

Answer:

The subproblems involve the suffixes (could also do prefixes) of the data string $D$. When considering the suffix $D[i \ldots n]$, the shortest encoding can be determined by finding which
substrings starting at $D[i]$ match any of the encoding strings, and if that encoding string was used what the smallest encoding of the rest of the data would be. $L[i]$ will store the minimal number of encoding strings needed to encode the data string suffix $D[i \ldots n]$. Assume that the array $S$ holds the $m$ encoding strings.

$$L[i] = \min_{1 \leq j \leq m} \begin{cases} 1 + L[i + \text{Len}(S[j])] & \text{if } D[i \ldots (i + \text{Len}(S[j]) - 1)] = S[j] \text{ and } (i + \text{Len}(S[j])) - 1 \leq n \\ \infty & \text{otherwise} \end{cases}$$

The base case is $L[n + 1] = 0$. In order to output the encoding strings used, the index of the encoding string that results in a minimal $L[i]$ with be stored in $ES[i]$.

**Inputs:** array of encoding strings $S$, data string $D$, number of encoding strings $m$, largest size of encoding string $k$, length of data string $n$

**Output:** the length of the best encoding

**Algorithm** \textsc{FindEncodingLength} $(S, D, m, k, n)$

for $i \leftarrow 1$ to $n$ do

$L[i] \leftarrow \infty$

$ES[i] \leftarrow \text{nil}$

// use results from previous suffixes to compute length

for $i \leftarrow n$ downto 1 do

for $str \leftarrow 1$ to $m$ do

if $D[i \ldots (i + \text{Len}(S[str]) - 1)] = S[str]$ then

if $(1 + L[i + \text{Len}(S[str])]) < L[i]$ then

$L[i] \leftarrow 1 + L[i + \text{Len}(S[str])]$

$ES[i] \leftarrow str$

return $L[1]$

The worst-case running time is $O(nmk)$. There are $n$ subproblems and to calculate each one it is necessary to compare $m$ strings of length at most $k$. To produce the list of encoding strings used we can use the following algorithm.

**Algorithm** \textsc{OutputStrings}$(S, m, ES)$

for $i \leftarrow 1$ to $n$ do

if $ES[i] \neq \text{nil}$ then

output $S[ES[i]]$

**Problem 3.** (10 points)

Let $A$ be a $n \times m$ array of 0’s and 1’s. Design a $O(nm)$ time algorithm for finding the largest square block of 1’s only.

**Answer:** Let $l(i, j)$ be the length of the side of the largest square block of 1’s whose bottom right corner is $A[i, j]$.

$$l(i, j) = \begin{cases} 0 & \text{if } A[i, j] = 0 \\ \min\{l(i - 1, j - 1), l(i - 1, j), l(i, j - 1)\} + 1 & \text{otherwise} \end{cases}$$
Once the matrix $l$ is computed, simply scan for the largest number. Time complexity is $O(nm)$, space complexity is $O(n + m)$. 