1. We say that the string $x = x_1 x_2 \ldots x_n$ is a subsequence of the string $y = y_1 y_2 \ldots y_m$ in the usual way: $m \geq n$ and there are indices $1 \leq i_1 < i_2 < \ldots < i_n \leq m$ such that, $(\forall k)x_k = y_{i_k}$. The goal is a greedy strategy to test in $O(n + m)$ time whether $x$ is a subsequence of $y$ (no code is required.)

Describe then justify your greedy choice.
(more space for the greedy justification)
2. The *longest common subsequence* problem is to find the longest (non-contiguous) sequence of characters shared by two strings \( X = x_1x_2\ldots x_n \) and \( Y = y_1y_2\ldots y_m \). For example, if \( X = CAB \) and \( Y = BACB \), the answer is 2 (they both share length 2 subsequences AB and CB). Here, we define a sub-problem \( C(i, j) \) to be the longest subsequence of \( x_1x_2\ldots x_i \) and \( y_1y_2\ldots y_j \). The recurrence we have seen for \( C \) is

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

Use the recurrence to fill in a table for \( C \) for the strings \( X = PEQNP \) and \( Y = PQPN \). You may use (or not) the sketch below.

<table>
<thead>
<tr>
<th></th>
<th>P</th>
<th>Q</th>
<th>P</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3. The following flow graph $G$ has some flow assignments already made. Use the Ford-Fulkerson method to continue the process of deriving the maximum flow from $s$ to $t$. Show the residual graph $G_f$, find an augmenting path in $G_f$, and then update the flow in $G$. One step should be enough to find the max-flow.
(more space for Ford-Fulkerson)
4. On the midterm we started the process of devising a dynamic programming solution to the ODOT Sign problem: The Oregon Department of Transportation (ODOT) has decided to place a series of warning signs along a section of a major road. This is a one-way road and the dangerous sections are at mileposts \((m_1, m_2, \ldots, m_n)\). They wish to cover each of these locations, meaning that for each milepost there should be a sign at that point or at most \(k\) miles before it. Formally, for each milepost \(m_i\), it is covered if there is a sign at some \(m_j\) \((j \leq i)\) with \(m_i - m_j \leq k\).

Signs can only be placed at a milepost and there is a cost \(c_i\) to place a sign at \(m_i\). The input consists of \([m_1, m_2, \ldots, m_n]\), \([c_1, c_2, \ldots, c_n]\), and \(k\). You can assume that the \(m_i\) are sorted \((m_{i-1} < m_i)\) and that \(c_i > 0\). The goal here is to determine the minimum total cost of a placement of signs at mileposts that covers all locations.

We define subproblem \(WS(i)\) to be the minimum placement cost of warning signs such that (i) there is a sign at location \(m_i\) and (ii) locations \(m_1, m_2, \ldots, m_{i-1}\) are also covered. The overall cost that we can report to ODOT is \(\min\{ WS(i) \mid 1 \leq i \leq n, \ m_n - m_i \leq k \}\).

**recurrence:**

\[
WS(i) = \begin{cases} 
   c_1 & \text{if } i = 1 \\
   c_i + \min \{ WS(j) \mid 1 \leq j < i, \ m_{i-1} - m_j \leq k \} & \text{otherwise.}
\end{cases}
\]

You are to write code that will initialize and fill out an array \(WS\) using the given recurrence. You may do so in either a bottom-up (iterative) or top-down (memoized) manner.

**PART I:** The pseudo-code is **Iterative** or **Memoized** (circle one)

**PART II:** The time bound of the pseudo-code is:

(continue on next page)
PART III: Write pseudo-code here.
5. Use Huffman’s algorithm to derive a prefix (-free) code for the given characters and frequencies.

<table>
<thead>
<tr>
<th>char</th>
<th>m</th>
<th>n</th>
<th>o</th>
<th>p</th>
<th>q</th>
<th>r</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>freq</td>
<td>2</td>
<td>3</td>
<td>8</td>
<td>21</td>
<td>55</td>
<td>144</td>
<td>377</td>
</tr>
</tbody>
</table>

Show your code tree and write out the derived code for each character.
6. (exercise 6.4, DPV) We want to devise a dynamic programming solution to the following problem: there is a string of characters which might have been a sequence of words with all the spaces removed, and we want to know if there is a way to insert spaces that separate valid English words. For example, *theyouthevent* could be from “the you the vent”, “the youth event” or “they out he vent”. If the input is *theeaglehaslande*, then there’s no such way. Assume that the original sequence of words had no other punctuation (such as periods), no capital letters, and no proper names - all valid words are in a fixed dictionary.

Let \( \text{dict}(w) \) be the function that will look up a provided word in the dictionary, and return \( \text{true} \) iff the word \( w \) is in it.

\[
\text{dict}(w) = \begin{cases} 
\text{true} & \text{if } w \text{ is a valid word} \\
\text{false} & \text{otherwise.}
\end{cases}
\]

Let the input string be \( x = x_1 x_2 ... x_n \). We define the subproblem \( \text{split}(i) \) as that of determining (answer \( \text{true} \) or \( \text{false} \)) whether it is possible to correctly add spaces to \( x_i x_{i+1} ... x_n \) so that only valid words are between the spaces.

Give a recurrence relation for \( \text{split}(i) \) (no code!).
7. Which of the following does NP stand for?

(a) Non-deterministically Possible
(b) Not Possible
(c) Non-Provable
(d) No Provolone
(e) Non-exponential Polynomial
(f) Now Polynomial
(g) Non-definable Polynomial
(h) Non-Polynomial
(i) Non-diptych Portraiture
(j) Never Polynomial
(k) Non-deterministic Polynomial
(l) Non-deterministic Provable

8. Answer the following questions about $P$ and $NP$:

(a) Claim: Every problem that can be solved in polynomial time non-deterministically can be solved by a polynomial time deterministic algorithm. **True** or **False** or **Open**?

(b) Claim: If a single NP-complete problem is shown to have a polynomial time (deterministic) algorithm, then it can be concluded that $P = NP$. **True** or **False** or **Open**?

points [11, 8, 10, 11, 8, 11, 2, 4] = 65 total