Problem 1: PPM and BW practice (20pt)

The string bcabccabc is encoded using PPM. The (partial) dictionary constructed during encoding is given below. For the following questions, assume that escape count is given by the number of different characters for each context. Recall that we can optimize PPM by using exclusion, where we exclude the possibility of seeing certain characters when we switch down a context, thereby increasing the probabilities of the other characters and avoid sending additional escapes. However, for now, we’ll assume exclusion is not used, and we’ll come back to it later. Assume that the alphabet has 26 characters, and use $k = 2$.

1. Fill in the empty spaces in the dictionary below. (5pt)

<table>
<thead>
<tr>
<th>Context</th>
<th>Counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>empty</td>
<td>$a = 2$</td>
</tr>
<tr>
<td></td>
<td>$b = 3$</td>
</tr>
<tr>
<td></td>
<td>$c = 4$</td>
</tr>
<tr>
<td></td>
<td>$a$</td>
</tr>
<tr>
<td></td>
<td>$b$</td>
</tr>
<tr>
<td></td>
<td>$c$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Context</th>
<th>Counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>ab</td>
<td>$c = 2$</td>
</tr>
<tr>
<td></td>
<td>$b$</td>
</tr>
</tbody>
</table>

Figure 1: Dictionary

Solution: String: bcabccabc
2. Suppose the next letter in the string is \( b \). Compute the number of bits required to encode \( b \), and also list the changes made to the dictionary. (You do not have to compute the exact number of bits, simply write it as an expression containing logs). The answer need not be an integer. (5pt)

**Solution:**

Code = escape, escape, \( b \)

Space = \(-\log(2/4) - \log(2/5) - \log(3/12) = 4.32 \) bits

3. Now assume that exclusion is used. Recompute the number of bits required to encode the character \( b \). (5pt)

**Solution:**

Code = escape, \( b \)

Space = \(-\log(2/4) - \log(3/4) = 1.42 \) bits

4. Encode the above string using Burrows-Wheeler. Just show the sequence of characters after the BW transform (don’t bother compressing using move to front). (5pt)
**Problem 2: Substring matching using Burrows-Wheeler (20pt)**

In class, we saw the Burrows-Wheeler transform, and how it can be used to create a string that is more easily compressible. In this question, we will see another use for the Burrows-Wheeler transformation; in particular, for efficiently finding all occurrences of a substring $S$ in a string $X$.

Recall that the Burrows-Wheeler transform operates by first listing all cyclic shifts of the string $X$ (equivalently, contexts of each character in $X$). Instead of keeping track of the index of each character in the original string, we can append a special end-of-string character $\$ to $X$, which we define to be lexicographically smaller than all others. The Burrows-Wheeler transform then sorts these cyclic shifts of $X$ lexicographically. In this version, we sort these words from left to write (like English, and in the opposite direction from what we’ve seen in class). The result can be seen as a matrix of characters, in which each row is a shift of the string. The output is then the last column of this matrix (equivalently, the letters whose contexts we sorted). We denote the sorted matrix as $M_{BWT}$, and the output of the Burrows-Wheeler transform on string $X$ as $BWT(X)$. Additionally, for the purpose of the string matching application, we define the following:

- $C(a)$: the number of characters in $X\$ that are lexicographically smaller than $a$, for a character $a$.
- $F(a,i)$: the number of times character $a$ appears in the $i$ first characters of $BWT(X)$.
- $L(S)$: The index of the first row of $M_{BWT}$ that starts with the string $S$.
- $U(S)$: The index of the last row of $M_{BWT}$ that starts with the string $S$.

1. Show that $C(a) + 1$ is the index (1-based) of the first row in $M_{BWT}$ that starts with $a$. (6pt)

**Solution:** Note that since the rows of $M_{BWT}$ represent all the possible cyclic shifts of $X$, each character of $X$ appears in the first column of $M_{BWT}$ exactly once. Furthermore, recall that the rows of $M_{BWT}$ are sorted lexicographically from left to right, and therefore the first column is sorted lexicographically. Therefore the first time $a$ appears in the first column is exactly after all letters that are lexicographically smaller than it.  

2. Prove that $L(aS) = C(a) + F(a,L(S) - 1) + 1$, and $U(aS) = C(a) + F(a,U(S))$. (6pt)

**Solution:** We can think of $L(aS)$ as the index of the first row which starts with $a$ which is followed by $S$. From Part 1, we know that $C(a) + 1$ is the index of the first row that starts with $a$. So we just need to show that $F(a,L(S) - 1)$ represents the number of rows that start with $a$ that are lexicographically before $aS$. To see this, recall that
$BWT(X)$ is the last column of $M_{BWT}$, and that the rows of $M_{BWT}$ represent all cyclic shifts of $X$. Thus, for each row that ends in $a$, there is a row that begins in $a$, and the order of rows that end in $a$ represents the lexicographical order of the rows that begin with $a$, sorted starting from their second entry. Thus, $F(a, L(S) - 1)$ represents the number of rows that begin with $a$ which, sorted starting from their second entry, are lexicographically before $S$, as desired.

Finally, note that since rows are sorted lexicographically from left to right, all rows that begin with the same substring $S$ must be next to each other in $M_{BWT}$. The index of the last row that starts with $aS$ is (the index of the first row that begins with $a$) + (the number of rows that begin with $a$ and which are lexicographically smaller than or equal to $aS$) -1. From the previous results shown, we can rewrite this as $C(a) + 1 + F(a, U(S)) - 1$, as desired.

3. Assuming that $M_{BWT}$, $C(a)$, and $F(a, i)$ are precomputed for a string $X$, give an algorithm that can find the number of occurrences of a substring $S$ in $X$. (8pt)

**Solution:** Let $\text{next}(a)$ be the character that is lexicographically next in $X$ after $a$.

```python
def findMatches(S[1...k]):
    a = S[k]
    L = C(a) + 1
    U = C(\text{next}(a))
    i = k-1
    while (i > 0):
        a = S[i]
        L = C(a) + 1 + F(a, L-1)
        U = C(a) + F(a, U)
        i--
    return (U-L+1)
```

Problem 3: Huffman coding (20pt)

In class, we used Huffman’s algorithm to construct optimal prefix codes for a fixed message probability distribution. However, most practical compression algorithms will update the message probability distribution as they encode the input, which means that the Huffman code has to be updated on every message of the input sequence. A naive solution to this issue is to rerun Huffman’s algorithm from scratch on every step, which can be very costly. In this problem, we will study a more efficient way of updating the Huffman code.

We call the tree produced by Huffman’s algorithm a *Huffman tree*. Notice that instead of using message probability to determine the weight of nodes (as we did in class), we can use message frequency (number of occurrences), without changing any other aspect of the algorithm or its analysis. Let $w(u)$ denote the weight of node $u$. 

Consider the following property: a binary tree with \( n \) leaves has the sibling property if and only if:

(a) the \( n \) leaves \( a_1, a_2, \ldots, a_n \) have positive integer weights \( w(a_1), \ldots, w(a_n) \), and the weight of each internal node is the sum of the weights of its children; and

(b) the nodes can be numbered \( u_1, u_2, \ldots, u_{2n-1} \) in non-decreasing order by weight \( (i < j \implies w(u_i) \leq w(u_j)) \), so that nodes \( u_{2j-1} \) and \( u_{2j} \) are siblings, for \( 1 \leq j \leq n-1 \), and their common parent node is higher in the numbering.

1. Show that a binary tree is a Huffman tree if and only if it has the sibling property. (4pt)

**Solution:** (Only if) Consider the output tree from Huffman’s algorithm, and label nodes in the order they were merged (assigning the lower label to the lower weight node being merged). Point (a) is satisfied by the definition of the algorithm. Point (b) is satisfied because the algorithm always chooses the pair of available nodes with the lowest weights, and inserts a node with larger weight. Therefore it processes them in non-decreasing order. Furthermore, a node can only be processed by the algorithm if its descendants in the final tree have already been processed. Therefore, parents are always labeled with a higher number than their children.

(If) Consider a binary tree with the sibling property, and a corresponding order on the nodes. The algorithm can process the nodes in pairs in this order. Note first that by part (a), the weights of the tree nodes will correspond to the weights assigned by the algorithm. Since the nodes are ordered in non-decreasing order, the next pair of nodes to be processed is always available to the algorithm, since: (1) all nodes with strictly lower weight have already been processed and, (2) all children of these nodes have already been processed.

2. The level of a node is its distance from the root of the tree. Let \( u \) and \( v \) be two nodes from the tree, with weights \( w(u), w(v) \) and at levels \( l(u), l(v) \) respectively. Show that if \( w(u) < w(v) \) then \( l(u) \geq l(v) \). Hint: you can use the fact that Huffman codes are optimal. (4pt)

**Solution:** Suppose, for the sake of contradiction, that \( w(u) < w(v) \) and \( l(u) < l(v) \). Clearly, \( u \) is not an ancestor of \( v \), since \( w(v) > w(u) \), so we can exchange the subtrees rooted at \( u \) and \( v \). The resulting tree has lower average length, which is a contradiction since Huffman codes are optimal.

3. Show that no two nodes with the same weight can be two (or more) levels apart. (4pt)

**Solution:** Let \( u, v \) be two nodes such that \( l(u) > l(v) + 1 \). Let \( u' \) be \( u \)'s parent. Then the \( w(u') > w(v) \) and \( l(u') > l(v) \), which is a contradiction by the previous subproblem.

4. So far, we have assumed that we encode a sequence of messages all at once, which means that the frequency of each message is fixed. Suppose now, that we encode a
sequence of messages one by one, updating the Huffman tree on each step. Suppose that the next message in the input is $a$, and that $a$ already has a corresponding leaf in the Huffman tree. This message increases the frequency of $a$ by one, so suppose that we simply increment by one the weight of the leaf corresponding to $a$ and its ancestors (i.e., all the nodes in the path from this leaf to the root). Specify a condition under which the resulting tree is not a Huffman tree. (4pt)

**Solution:** If there are two nodes of the same weight on different levels, and the node farther from the root is incremented, then the resulting tree does not satisfy the sibling property (by the previous subproblem).

5. Consider now the following two-phase procedure. Initially, we have a Huffman tree that contains the message $a$, among other messages, and an order over the nodes of the tree (given by the sibling property). In the first phase, we set the leaf corresponding to $a$ as the current node. Then, we repeatedly exchange the subtree rooted at the current node with the subtree rooted at the highest numbered node of the same weight, and make the parent of the latter node the current node. In the second phase, we increment the weights of the leaf corresponding to $a$ and its ancestors (as in the previous subproblem). Both phases can be performed at the same time, as shown by the following procedure.

```plaintext
procedure update(Huffman tree $T$, message $a$)
    $u$ ← leaf in $T$ corresponding to $a$
    while $u$ ≠ root($T$) do
        $v$ ← highest numbered node such that $w(u) = w(v)$
        Exchange the subtree rooted at $u$ with the subtree rooted at $v$
        (the positions of $u$ and $v$ in the order are swapped)
        $w(u)$ ← $w(u) + 1$
        $u$ ← parent of $u$
    end while
    $w(u)$ ← $w(u) + 1$ (update weight of the root)
end procedure
```

Show that the resulting binary tree is a Huffman tree. (4pt)

**Solution:** Since all the exchanges are of nodes of the same weight, the tree still has the sibling property after phase 1. Each of the incremented nodes is the largest numbered node of its weight before phase 2. Therefore, the ordered nodes are still in non-decreasing order.

---

**Problem 4: Entropy (20pt)**

In this problem, we will prove some basic properties of entropy. Let $X$ and $Y$ be discrete random variables. Define the joint entropy of $X$ and $Y$ as $H(X, Y) = \sum_{x \in X, y \in Y} p(x, y) \log \frac{1}{p(x, y)}$.

1. Prove the chain rule: $H(Y \mid X) = H(X, Y) - H(X)$. (5pt)
Solution:

\[ H(Y \mid X) = \sum_{x \in X} \left( p(x) \sum_{y \in Y} p(y \mid x) \log \frac{1}{p(y \mid x)} \right) \]

\[ = \sum_{x \in X} \left( p(x) \sum_{y \in Y} \frac{p(x,y)}{p(x)} \log \frac{p(x)}{p(x,y)} \right) \]

\[ = \sum_{x \in X, y \in Y} \left( p(x,y) \log \frac{1}{p(x,y)} - p(x,y) \log \frac{1}{p(x)} \right) \]

\[ = H(X,Y) - H(X) \]

2. Prove Bayes’ rule: \( H(Y \mid X) = H(X \mid Y) - H(X) + H(Y) \). (5pt)

Solution: By the previous sub-problem, we have:

\[ H(X,Y) = H(Y \mid X) + H(X) = H(X \mid Y) + H(Y) + H(X) \]

The statement follows clearly from this.

3. Assume i.i.d. \( X \) and \( Y \). Show that \( H(X,Y) = H(X) + H(Y) \), and \( H(Y \mid X) = H(Y) \). (5pt)

Solution:

\[ H(X,Y) = \sum_{x \in X, y \in Y} p(x,y) \log \frac{1}{p(x,y)} \]

\[ = \sum_{x \in X, y \in Y} p(x,y) \left( \log \frac{1}{p(x)} + \log \frac{1}{p(y)} \right) \]

\[ = \sum_{x \in X} p(x) \log \frac{1}{p(x)} + \sum_{y \in Y} p(y) \log \frac{1}{p(y)} \]

\[ = H(X) + H(Y) \]

The second equation follows from this and the chain rule.

4. Show that \( H(Y \mid X) \leq H(Y) \). (5pt)

Hint: use the following inequality: for all \( x_1, \ldots, x_n > 0 \) and \( \alpha_1, \ldots, \alpha_n > 0 \) such that \( \sum_{1 \leq i \leq n} \alpha_i = 1 \) it holds that: \( \sum_i \alpha_i \log x_i \leq \log (\sum_i \alpha_i x_i) \) (this is a consequence of Jensen’s inequality).
Solution: We have that:

\[
H(Y \mid X) - H(Y) = \sum_{x \in X} \left( p(x) \sum_{y \in Y} p(y \mid x) \log \frac{1}{p(y \mid x)} \right) - \sum_{y \in Y} p(y) \log \frac{1}{p(y)}
\]

\[
= \sum_{x \in X, y \in Y} \left( p(x, y) \log \frac{p(x)}{p(x, y)} - p(x, y) \log \frac{1}{p(y)} \right)
\]

\[
= \sum_{x \in X, y \in Y} p(x, y) \log \frac{p(x)p(y)}{p(x, y)}
\]

\[
\leq \log \left( \sum_{x \in X, y \in Y} p(x)p(y) \right) \leq \log 1 = 0
\]