Goal: Make a Huffman code table for compressing the following string.

```
huffman fluffs many mums
```

Next step: Make a frequency table
**huffman fluffs many mums**

**Frequency table**

<table>
<thead>
<tr>
<th>char</th>
<th>frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>5</td>
</tr>
<tr>
<td>m</td>
<td>4</td>
</tr>
<tr>
<td>u</td>
<td>3</td>
</tr>
<tr>
<td>חלק</td>
<td>3</td>
</tr>
<tr>
<td>s</td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td>n</td>
<td>2</td>
</tr>
<tr>
<td>y</td>
<td>1</td>
</tr>
<tr>
<td>h</td>
<td>1</td>
</tr>
<tr>
<td>l</td>
<td>1</td>
</tr>
</tbody>
</table>

Next step: Start creating the Huffman tree.
We start by creating a priority queue where each list node refers to a tree node containing a single character.

**Process**
1. Take first two nodes from priority queue.
2. Combine them into a cluster. (Will require creating a new tree node.) The cluster will have the sum of the frequencies of its children.
3. Insert the cluster into priority queue.
4. Repeat (from step 1) until there is only one node in the priority queue.

**Priority queue compare function**
- Order by the frequency.
- If frequency is same, then nodes with just a single character come before clusters.
- If frequency is same and both are single-character nodes (i.e., not clusters order by ASCII value of character).

This summary is not a substitute for reading the homework description. In case of any discrepancy, it takes precedence.
Process
1. Take first two nodes from priority queue.
2. Combine them into a cluster. (Will require creating a new tree node.) The cluster will have the sum of the frequencies of its children.
3. Insert the cluster into priority queue.
4. Repeat (from step 1) until there is only one node in the priority queue.

Priority queue compare function
- Order by the frequency.
- If frequency is same, then nodes with just a single character come before clusters.
- If frequency is same and both are single-character nodes (i.e., not clusters order by ASCII value of character).

Next step: Join first two nodes

This summary is not a substitute for reading the homework description. In case of any discrepancy, it takes precedence.
Process
1. Take first two nodes from priority queue.
2. Combine them into a cluster. (Will require creating a new tree node.) The cluster will have the sum of the frequencies of its children.
3. Insert the cluster into priority queue.
4. Repeat (from step 1) until there is only one node in the priority queue.

Priority queue compare function
• Order by the frequency.
• If frequency is same, then nodes with just a single character come before clusters.
• If frequency is same and both are single-character nodes (i.e., not clusters order by ASCII value of character).

Next step: Join first two nodes

This summary is not a substitute for reading the homework description. In case of any discrepancy, it takes precedence.
Process
1. Take first two nodes from priority queue.
2. Combine them into a cluster. (Will require creating a new tree node.) The cluster will have the sum of the frequencies of its children.
3. Insert the cluster into priority queue.
4. Repeat (from step 1) until there is only one node in the priority queue.

Next step: Join first two nodes

Priority queue compare function
- Order by the frequency.
- If frequency is same, then nodes with just a single character come before clusters.
- If frequency is same and both are single-character nodes (i.e., not clusters order by ASCII value of character.)

This summary is not a substitute for reading the homework description. In case of any discrepancy, it takes precedence.
Process
1. Take first two nodes from priority queue.
2. Combine them into a cluster. (Will require creating a new tree node.) The cluster will have the sum of the frequencies of its children.
3. Insert the cluster into priority queue.
4. Repeat (from step 1) until there is only one node in the priority queue.

Priority queue compare function
• Order by the frequency.
• If frequency is same, then nodes with just a single character come before clusters.
• If frequency is same and both are single-character nodes (i.e., not clusters) order by ASCII value of character.

Next step: Join first two nodes

This summary is not a substitute for reading the homework description. In case of any discrepancy, it takes precedence.
Process
1. Take first two nodes from priority queue.
2. Combine them into a cluster. (Will require creating a new tree node.) The cluster will have the sum of the frequencies of its children.
3. Insert the cluster into priority queue.
4. Repeat (from step 1) until there is only one node in the priority queue.

Next step: Join first two nodes

Priority queue compare function
• Order by the frequency.
• If frequency is same, then nodes with just a single character come before clusters.
• If frequency is same and both are single-character nodes (i.e., not clusters order by ASCII value of character.

This summary is not a substitute for reading the homework description. In case of any discrepancy, it takes precedence.
Process
1. Take first two nodes from priority queue.
2. Combine them into a cluster. (Will require creating a new tree node.) The cluster will have the sum of the frequencies of its children.
3. Insert the cluster into priority queue.
4. Repeat (from step 1) until there is only one node in the priority queue.

Priority queue compare function
- Order by the frequency.
- If frequency is same, then nodes with just a single character come before clusters.
- If frequency is same and both are single-character nodes (i.e., not clusters) order by ASCII value of character.

This summary is not a substitute for reading the homework description. In case of any discrepancy, it takes precedence.

Next step: Join first two nodes
Process
1. Take first two nodes from priority queue.
2. Combine them into a cluster. (Will require creating a new tree node.) The cluster will have the sum of the frequencies of its children.
3. Insert the cluster into priority queue.
4. Repeat (from step 1) until there is only one node in the priority queue.

Priority queue compare function
- Order by the frequency.
- If frequency is same, then nodes with just a single character come before clusters.
- If frequency is same and both are single-character nodes (i.e., not clusters) order by ASCII value of character.

Next step: Join first two nodes

This summary is not a substitute for reading the homework description. In case of any discrepancy, it takes precedence.
Process
1. Take first two nodes from priority queue.
2. Combine them into a cluster. (Will require creating a new tree node.) The cluster will have the sum of the frequencies of its children.
3. Insert the cluster into priority queue.
4. Repeat (from step 1) until there is only one node in the priority queue.

Priority queue compare function
- Order by the frequency.
- If frequency is same, then nodes with just a single character come before clusters.
- If frequency is same and both are single-character nodes (i.e., not clusters order by ASCII value of character.

Next step: Join first two nodes
1. Take first two nodes from priority queue.
2. Combine them into a cluster. (Will require creating a new tree node.) The cluster will have the sum of the frequencies of its children.
3. Insert the cluster into priority queue.
4. Repeat (from step 1) until there is only one node in the priority queue.

Priority queue compare function
• Order by the frequency.
• If frequency is same, then nodes with just a single character come before clusters.
• If frequency is same and both are single-character nodes (i.e., not clusters order by ASCII value of character.

Next step: Remove head of priority queue, leaving only the tree.

This summary is not a substitute for reading the homework description. In case of any discrepancy, it takes precedence.
Next step: Create the code table
Notice that no code is a prefix of another.
More frequently occurring characters get shorter codes.
huffman fluffs many mums

<table>
<thead>
<tr>
<th>char</th>
<th>code</th>
<th># of bits</th>
<th>frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>00</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>m</td>
<td>110</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>‿</td>
<td>011</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>u</td>
<td>100</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>s</td>
<td>1111</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td>1011</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>n</td>
<td>1110</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>y</td>
<td>1010</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>h</td>
<td>0100</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>l</td>
<td>0101</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

Encoded string:
0100 100 00 00 110
h u f f f m
1011 1110 011 00 0101
a n ‿ f l
100 00 00 1111 011 110
u f f s ‿ m
1011 1110 1010 011 110
a n y ‿ m
100 110 1111
u m s
huffman fluffs many mums

<table>
<thead>
<tr>
<th>char</th>
<th>code</th>
<th># of bits</th>
<th>frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>00</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>m</td>
<td>110</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>‿</td>
<td>011</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>u</td>
<td>100</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>s</td>
<td>1111</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td>1011</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>n</td>
<td>1110</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>y</td>
<td>1010</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>h</td>
<td>0100</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>l</td>
<td>0101</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

Encoded string 10 bytes
01001000 00011010
h u f f m a

11111001 10001011
n ‿ f l u

00000001 11011110
ff s ‿ m

10111110 10100111
a n y ‿ m

10100110 11110000
u m s