CS 630 – Fall 2024 – Lab 10
Nov 13, 2024

Problem 1 *Huffman encode/decode practice.*

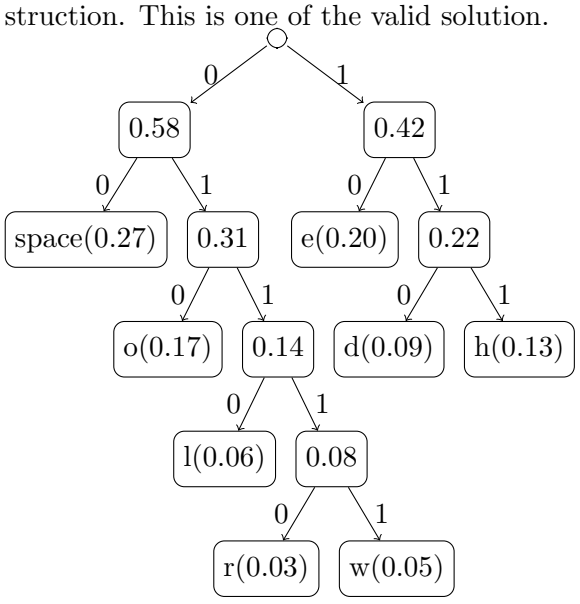
1. Given the following frequency table for the following characters.

Character	Frequency
`space`	0.27
`d`	0.09
`e`	0.20
`h`	0.13
`l`	0.06
`o`	0.17
`r`	0.03
`w`	0.05

Table 1: Character Frequencies

(a) Create a Huffman encoding tree, and write out the Huffman encoding for each characters.

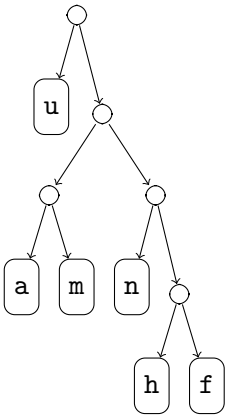
Solution: You answer might be different depending on the order of your tree construction. This is one of the valid solution.



(b) Encode phrase `hello world` using the Huffman tree you just created.

Solution: Based on the previous constructed tree, we have the following encoding:
`111 10 0110 0110 010 00 01111 01001110 0110 110`

2. Given the Huffman tree, decode the following binary `11100111111111101100110`.



Solution: The word is `huffman`.

Problem 2 *Min PQ encoding of Huffman Encoding*

So far we have seen how to design a Huffman encoding from the character frequencies in our data. For this we build a binary tree; the leaves correspond to the characters. Each node v has a frequency/priority associated, which implies the total frequency of the characters in the subtree rooted at v. Initially each character forms a tree of size 1. In each iteration the two subtrees with the lowest priorities are merged under a common root, and the root gets the sum of their priorities.

In this problem you have to construct a Huffman encoding algorithm that is using a min priority queue for its implementation. As input we are given the character frequencies.

Solution: Reminder: Priority queues are data structures that support the efficient lookup of the item with the current highest priority. (In this problem high priority = low frequency.) The operations that the PQ supports are

PEEK - show the highest priority in O(1) time
DELETE-MIN - return the highest priority item from Q
UPDATE-PIORITY - change the priority of an item

The priority queue is used to get fast access to the partial codes so far. It is initially populated with a singleton (root=leaf) tree for each symbol with priority copied from the symbol probability. Then the two lowest probability trees are repeatedly removed, merged, and added back to the priority queue with new priority that is the sum of the old priorities. When there is only one entry left, it is the tree of the Huffman encoding.

Problem 3 *Huffman Encoding base 3*

The Huffman Encoding algorithm covered in class used binary digits and binary trees. This problem considers ternary digits with values 0, 1 or 2 and ternary trees where the tree nodes can have up to 3 children.

1. In the optimality proof of the normal (binary) Huffman Encoding algorithm, there was a claim that there would always be an optimal tree where the two smallest probabilities were on the bottom level of the tree. Show that the three smallest probabilities are not necessarily on the bottom level of the tree using an example with just 4 symbols and their probabilities.

   Solution: For any optimal tree with 4 leaf nodes, the two highest probabilities are children of the root and the smallest 2 probabilities are children of the second internal node.

2. Generalize your previous example to state a precise condition for when the smallest three probability symbols are not necessarily at the bottom of the tree. This condition should only require checking the number of distinct symbols to be compressed.

   Solution: The claim is false for any number of distinct symbols where the ternary tree would not use all the leaf nodes. For a ternary tree, this is when the number of distinct symbols is even. Starting from a single root/leaf node, replacing a leaf node with a new internal node with three leaf nodes always results in a net increase in two leaf nodes. Since initial root/leaf node had one leaf, the number of leaf nodes will always be odd when they are all used.

3. Adapt the Huffman Encoding algorithm covered in class to ternary. Use dummy symbols with probability zero to pad the number of distinct symbols to address the issues raised in the previous parts.

   Solution: If the number of symbols is even, add one dummy symbol with probability zero. After that, run the normal Huffman Encoding algorithm like normal changing the greedy merging step to pick the three smallest probability nodes instead of the two smallest probability nodes.