Huffman Codes

Suppose we are trying to encode the set of five letters $S = \{a, b, c, d, e\}$. We can use fixed length encoding. 5 letters can be encoded using 3 bits. One possible encoding can be: $a = 000$, $b = 001$, $c = 010$, $d = 011$, $e = 100$. Using this encoding, the string badace will be encoded as 001 000 011 000 010 100. We just replace each letter with its encoding. Decoding a given binary string is also easy. We group the string in substrings of length 3 and decode individual substrings. For example: 100 010 000 011 000 001 is decoded as ecadab. We are using 3 bits per letter.

Another possibility of encoding is using variable length prefix codes. A prefix code for a set S of letters is a function r that maps each letter $x \in S$ to some sequence of 0's and 1's, in such a way that for distinct $x, y \in S$, the sequence $r(x)$ is not a prefix of the sequence $r(y)$. Any fixed length encoding is a prefix code. We can also use variable length prefix codes. One example of a prefix code is the encoding r_1 specified by $r_1(a) = 11$, $r_1(b) = 01$, $r_1(c) = 001$, $r_1(d) = 10$, $r_1(e) = 000$. Using this encoding, the string badace will be encoded as 01 11 10 11 001 000. For decoding a given binary string, we scan the string from left to right. As soon as we get a valid code, we decode it and repeat the process. For example: 000 001 11 10 11 01 is decoded as ecadab

Optimal Prefix codes: Suppose that for each letter $x \in S$, there is a frequency f_x, representing the fraction of letters in the text that are equal to x (for total n letters in a text, nf_x of these letters are equal to x).

$$\sum_{x \in S} f_x = 1$$

We can find encoding length of a text having n letters and using a prefix code r using the following formula:

$$\text{Encoding length} = \sum_{x \in S} n f_x |r(x)| = n \sum_{x \in S} f_x |r(x)|.$$

We define the average number of bits required per letter by encoding r as:

$$ABL(r) = \sum_{x \in S} f_x |r(x)|$$

Example: Let $S = \{a, b, c, d, e\}$ and $f_a = 0.32$, $f_b = 0.25$, $f_c = 0.20$, $f_d = 0.18$, $f_e = 0.05$.

ABL of constant bit encoding is:

$$ABL(CBE) = 0.32 \times 3 + 0.25 \times 3 + 0.20 \times 3 + 0.18 \times 3 + 0.05 \times 3$$
$$= (0.32 + 0.25 + 0.20 + 0.18 + 0.05) \times 3 = \boxed{3}$$

$$ABL(r_1) = 0.32 \times |11| + 0.25 \times |11| + 0.20 \times |001| + 0.18 \times |10|$$
$$+ 0.05 \times |000| = 0.32 \times 2 + 0.25 \times 2 + 0.20 \times 3 + 0.18 \times 2$$
$$+ 0.05 \times 3 = \boxed{2.25}$$

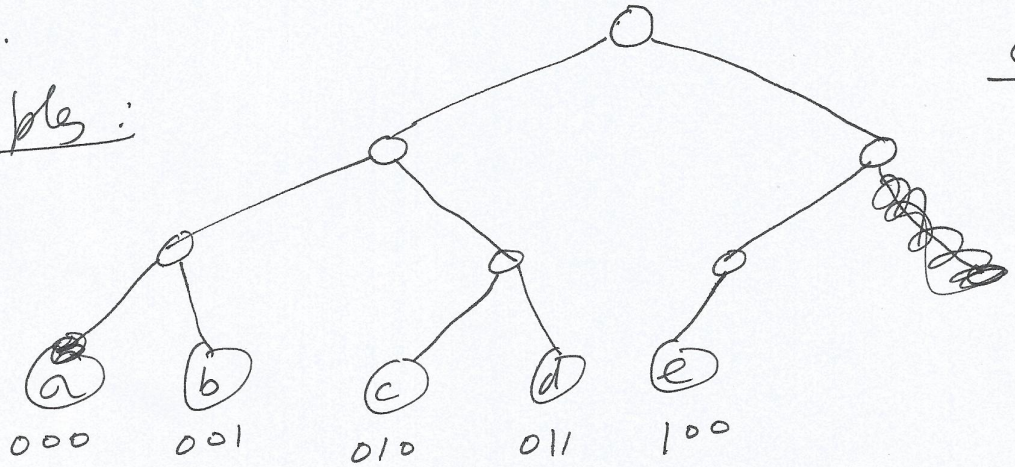
Using variable length prefix codes, we can reduce ABL.

Let r_2 be defined as: $r_2(a) = 11$, $r_2(b) = 10$, $r_2(c) = 01$, $r_2(d) = 001$, $r_2(e) = 000$. $ABL(r_2) = 0.32 \times |11| + 0.25 \times |10| + 0.20 \times |01| + 0.18 \times |001| + 0.05 \times |000| = 0.32 \times 2 + 0.25 \times 2 + 0.20 \times 2 + 0.18 \times 3 + 0.05 \times 3 = \boxed{2.23}$

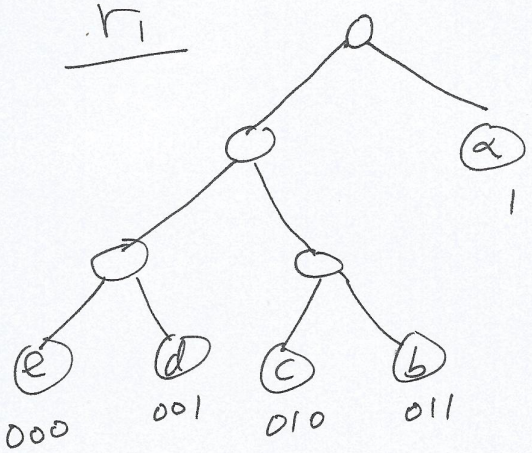
Representing Prefix Codes using Binary Trees : Suppose we take a binary tree T with number of leaves equal to the size of the alphabet S, and we label each leaf with a distinct letter in S. Such a labeled binary tree T naturally describes a prefix code, as follows: For each letter $x \in S$, we follow the path from the root to the leaf labeled x; each time the path goes from a node to its left child, we write down a 0, and each time the path goes from a node to its right child, we write down a 1. We take the resulting string of bits as the encoding of x.

CBE

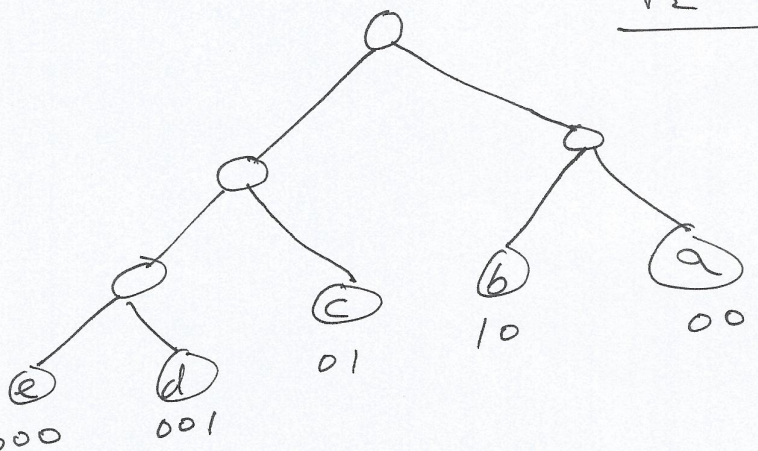
Example :



T_1



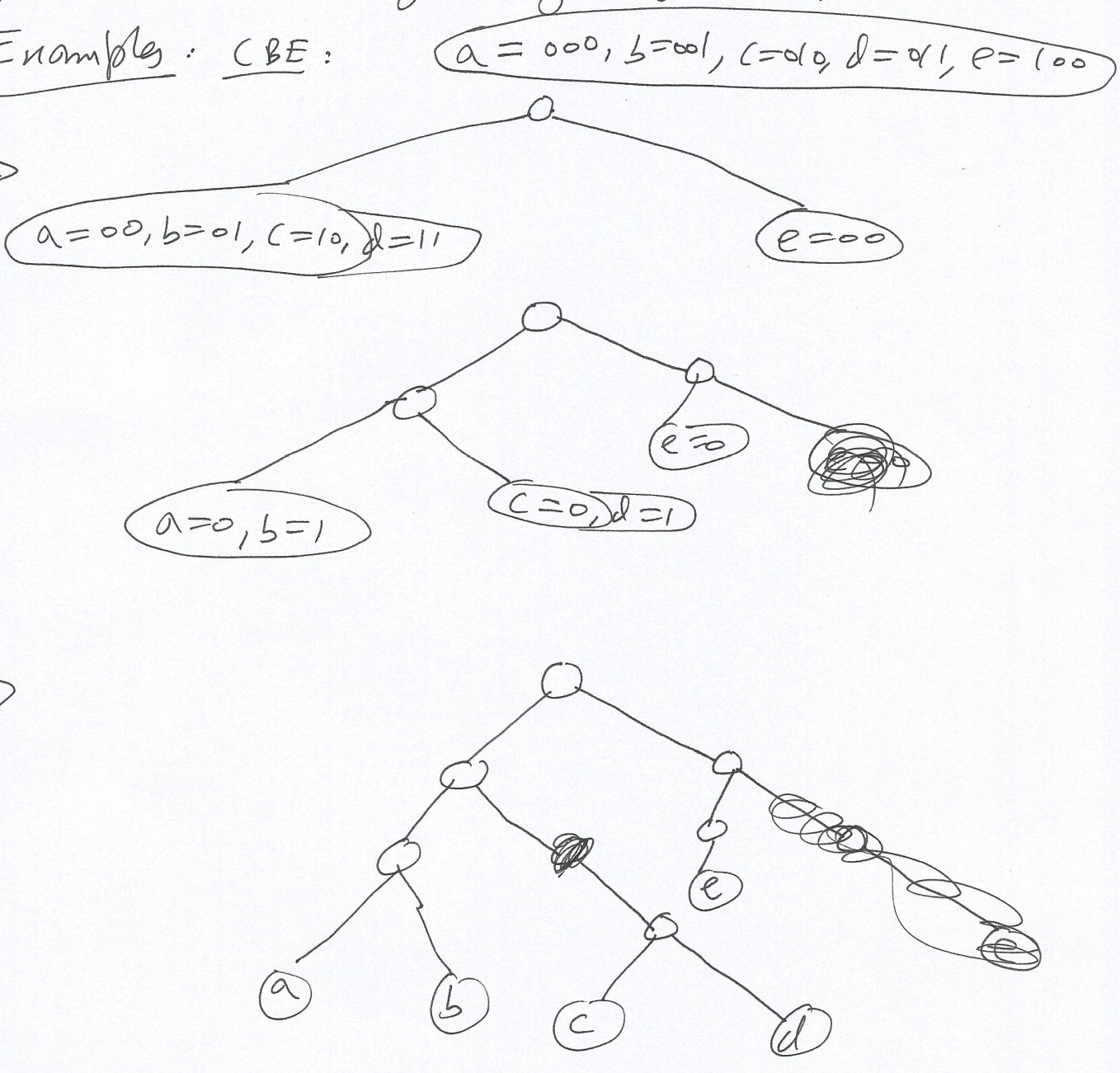
T_2



The encoding of S constructed from T is a prefix code because if the encoding of x is a prefix of the encoding of y then x will be on a path from root to $y \Rightarrow x$ cannot be a leaf node (a contradiction).

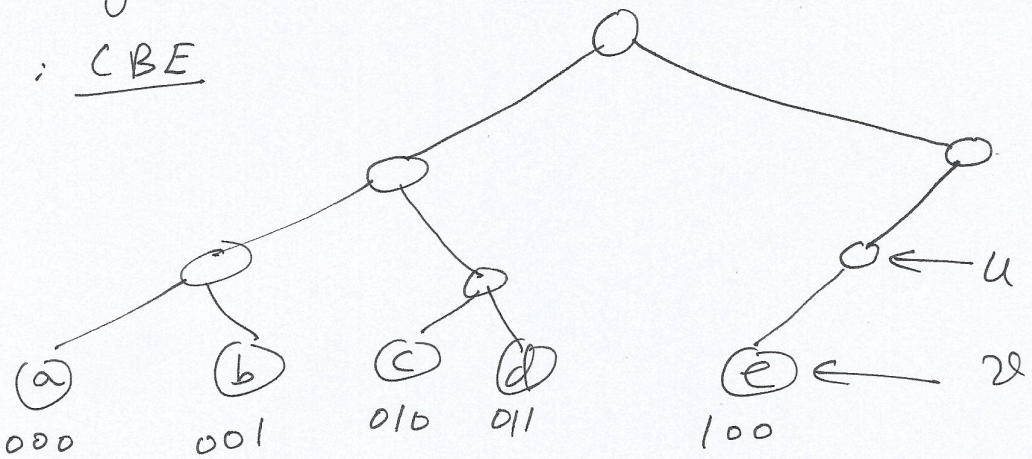
Given a prefix code γ, we can build a binary tree recursively as follows: We start with a root; all letters $x \in S$ whose encodings begin with a 0 will be leaves in the left subtree of the root, and all letters $y \in S$ whose encodings begin with a 1 will be leaves in the right subtree of the root. We now build these two subtrees recursively using this rule.

Examples: CBE:

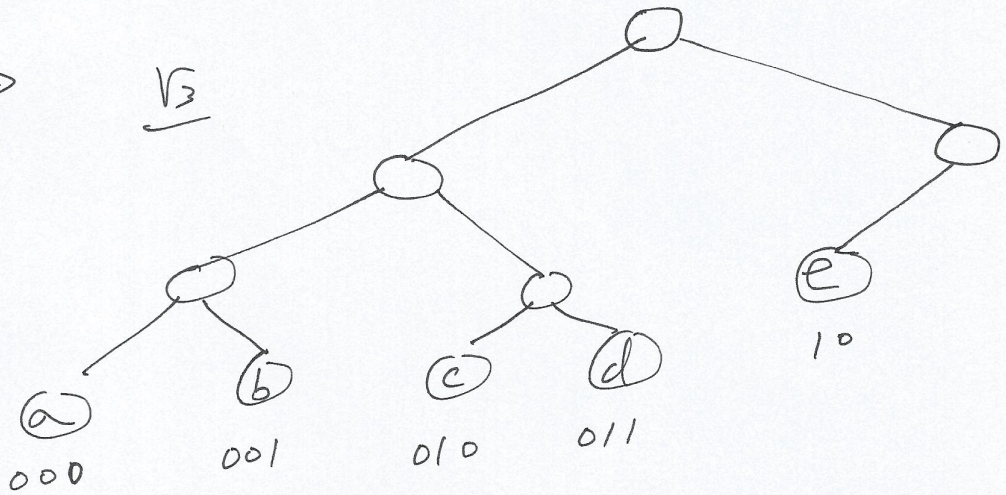


The binary tree corresponding to the optimal prefix code is full (each node that is not a leaf has two children). We prove this using <u>exchange argument</u>. Let T denote the binary tree corresponding to the optimal prefix code, and suppose it contains a node u with exactly one child v. Now convert T into a tree T' by replacing node u with v.

<u>Example</u> : <u>CBE</u>



$\Rightarrow$ <u>T_3</u>



This change decreases the number of bits needed to encode any leaf in the subtree rooted at node u, and it does not affect other leaves. So the prefix code corresponding to T' has a smaller ABL than the prefix code for T, contradicting the optimality of T.

$ABL(T_3) = \underbrace{0.32\times|000| + 0.25\times|001| + 0.20\times|010| + 0.18\times|011|}_{\hookrightarrow \text{Same as } ABL(CBE)} + \underbrace{0.05\times|110|}_{\hookrightarrow < ABL(CBE)}$

$= ABL(CBE) - 0.05\times 1 = 3 - 0.05 = \boxed{2.95}$

Suppose that u and v are leaves of T^*, such that depth(u) < depth(v) (depth of a leaf node is the path length from root to the leaf node). Further, suppose that in a labeling of T^* corresponding to an optimal prefix code, leaf u is labeled with $y \in S$ and leaf v is labeled with $z \in S$. Then $f_y \geq f_z$.

We use exchange argument. If $f_y < f_z$, then exchange y and z in T^* to get a labeling r'. Assuming that r was the original encoding!

Before: $ABL(r) = \sum_{x \in S} f_x |r(x)| = \sum_{\substack{x \in S \\ x \neq y,z}} f_x |r(x)| + f_y |r(y)| + f_z |r(z)|$

After: $ABL(r') = \sum_{x \in S} f_x |r'(x)| =$

$$= \sum_{\substack{x \in S \\ x \neq y,z}} f_x |r'(x)| + f_y |r'(y)| + f_z |r'(z)|$$

$$\Rightarrow ABL(r') - ABL(r) = f_y(|r'(y)| - |r(y)|) + f_z(|r'(z)| - |r(z)|)$$

Before: u — y $|r(y)|$; v — z $|r(z)|$; distance d

After: u — z $|r'(z)|$; v — y $|r'(y)|$; distance d

Assuming $|r(z)| - |r(y)| = |r'(y)| - |r'(z)| = d$

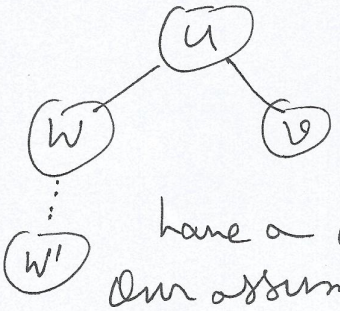
we get $|r'(y)| - |r(y)| = d$ and

$$|r'(z)| - |r^*(z)| = -d$$

$$\Rightarrow ABL(r') - ABL(r) = d(f_y - f_z) < 0$$

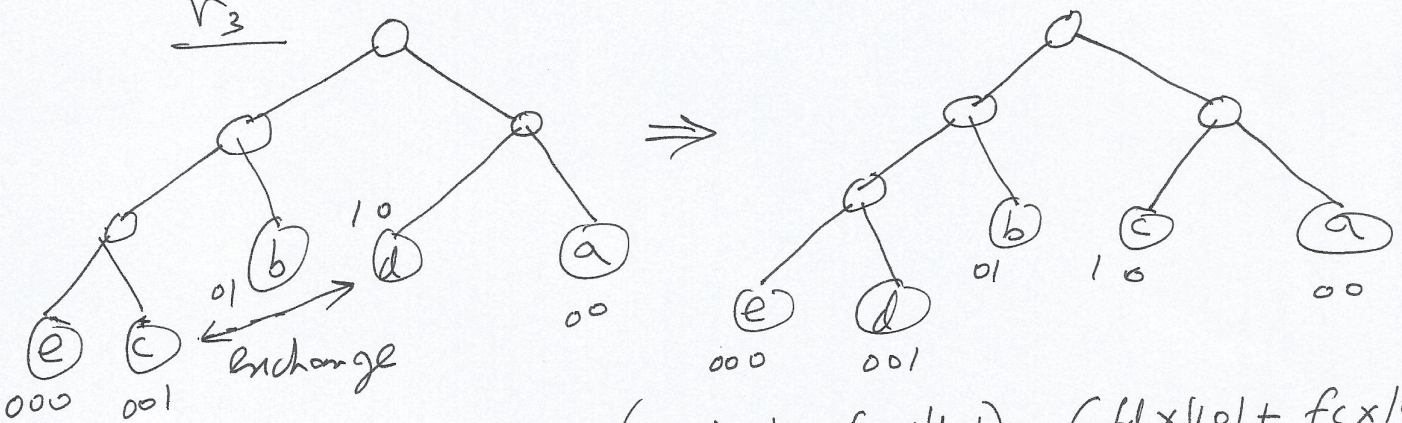
contradicting our assumption that r is optimal encoding for T^*.

Consider a leaf v in T^* (the binary tree corresponding to an optimal prefix code) whose depth is as large as possible. Leaf v has a parent u, and T^* is a full binary tree, so u has another child w which is a leaf of T^*.



This is because if w were not a leaf, there would be some leaf w' in the subtree below it. But then w' would have a depth greater than that of v, contradicting our assumption that v is a leaf of maximum depth in T^*.

There is an optimal prefix code, with corresponding tree T^*, in which the two lowest frequency letters are assigned to leaves that are siblings in T^*. Assign the two lowest frequency letters to the siblings at the largest depth. If this is not the case, then we can use exchange argument and bring the lowest frequency letters at the largest depth and reduce the ABL contradicting our assumption that we have an optimal prefix code. Example:



$ABL(T_2) - ABL(T_3) = (fd \times 3 + fc \times 3) - (fd \times 2 + fc \times 3)$

$= fd \times 1 - fc \times 1 = fd - fc < 0$ because $fd < fc$

Huffman's Algorithm for Constructing an Optimal Prefix Code

To construct a prefix code for an alphabet S with given frequencies:

If S has two letters then

Encode one letter using 0 and the other letter using 1

Else

Let y^* and z^* be the two lowest-frequency letters

Form a new alphabet S' by deleting y^* and z^* and replacing them with a new letter w of frequency $f_{y^*} + f_{z^*}$

Recursively construct a prefix code γ' for S', with tree T'

Define a prefix code for S as follows:

Start with T'

Take the leaf labeled w and add two children below it labeled y^* and z^*

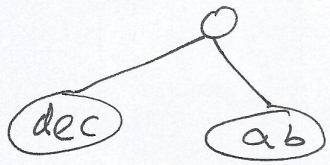
Endif

Example: $(f_a = 0.32,\ f_b = 0.25,\ f_c = 0.20,\ f_d = 0.18,\ f_e = 0.05)$

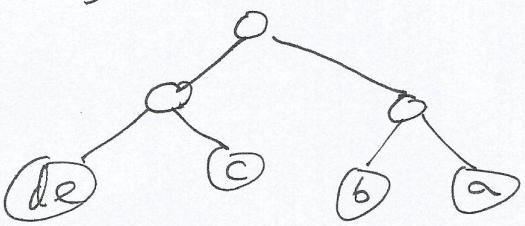
$\Rightarrow (f_a = 0.32,\ f_b = 0.25,\ f_{de} = 0.23,\ f_c = 0.20)$

$\Rightarrow (f_{dec} = 0.43,\ f_a = 0.32,\ f_b = 0.25)$

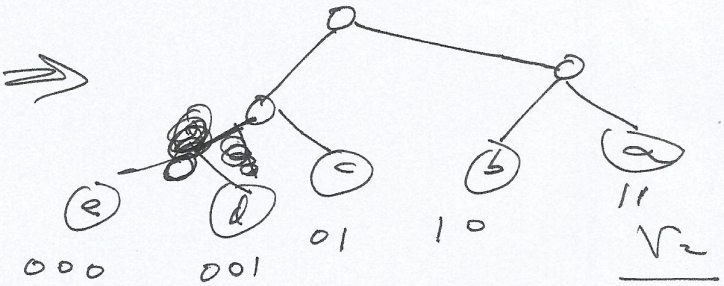
$\Rightarrow (f_{ab} = 0.57,\ f_{dec} = 0.43) \Rightarrow$



$\Rightarrow$



$\Rightarrow$



γ_2

$ABL(T') = ABL(T) - f_w$.

The depth of each letter x other than y^*, z^* is the same in both T and T'. Also, the depth of y^* and z^* in T are each one greater than the depth of w in T', and we also have $f_w = f_{y^*} + f_{z^*}$. We get:

$$ABL(T) = \sum_{x \in S} f_x \cdot depth_T(x)$$

$$= f_{y^*} \cdot depth_T(y^*) + f_{z^*} \cdot depth_T(z^*) + \sum_{x \neq y^*, z^*} f_x \cdot depth_T(x)$$

$$= (f_{y^*} + f_{z^*}) \cdot (1 + depth_{T'}(w)) + \sum_{x \neq y^*, z^*} f_x \cdot depth_T(x)$$

$$= f_w \cdot (1 + depth_{T'}(w)) + \sum_{x \neq y^*, z^*} f_x \cdot depth_{T'}(x)$$

$$= f_w + f_w \cdot depth_{T'}(w) + \sum_{x \neq y^*, z^*} f_x \cdot depth_{T'}(x)$$

$$= f_w + \sum_{x \in S'} f_x \cdot depth_{T'}(x)$$

$$= f_w + ABL(T')$$

Optimality of T: If T is not optimal, assume Z to be optimal with leaves labeled y^* and z^*. ($ABL(Z) < ABL(T)$) From Z we create Z' by deleting y^* and z^* from Z, and label their parent with w having frequency $f_{y^*} + f_{z^*} = f_w$, defining a prefix code S'. $\Rightarrow ABL(Z') = ABL(Z) - f_w$

$ABL(Z) < ABL(T) \Rightarrow ABL(Z') < ABL(T')$ A contradict.

We are using Induction

complexity is $O(n \log n)$ using priority queues as heaps.