CS 341 – Algorithms

Lecture 9 – Huffman Coding

16 June 2021

Today’s Plan

1. Compression, Optimal Prefix Code

2. Huffman’s Algorithm, Proof of Correctness

Compression

Suppose a text has 26 letters a,b,c,…,z. We can use 5 bits to represent each letter. $\lceil \log_2 26 \rceil = 5$

If each letter appears equally likely, then we cannot do much better than using $5n$ bits to store n letters.

Now suppose we have some statistics about the frequencies of each letter.

Let's say that we know that some letters appear with much higher frequencies.

$a = 10\%$ $b = 3\%$ $c = 2\%$ $d = 3\%$ $e = 9\%$ $z = 0.5\%$

Can we do better by using variable-length coding scheme?

The idea is to use fewer bits for more frequent letters, and more bits for less frequent letters,

so that the average number of bits used is fewer.

Decoding

If we use fixed-length encoding, then it is easy to decode.

$\underbrace{00001}_{b}\ \underbrace{00011}_{d}\ \underbrace{00010}_{c}\ \cdots$

But it may not be clear how to decode if we use variable-length encoding.

For example, suppose a=01, b=001, c=011, d=110, e=10.

Then how do we decode the compressed text such as 00101110?

$\underbrace{001}_{b}\ \underbrace{011}_{c}\ \underbrace{10}_{e}$ $\qquad$ $\underbrace{001}_{b}\ \underbrace{01}_{a}\ \underbrace{110}_{d}$ $\qquad$ ambiguity

Prefix Coding

To allow for easy decoding, we will construct **prefix code**,

so that no encoding string is a prefix of another encoded string.

The previous example was ambiguous because a=01 and c=011.

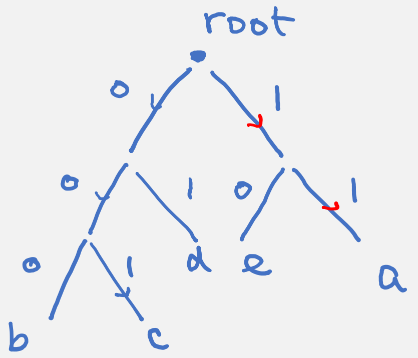
Suppose we use a prefix code for the five letters, a=11, b=000, c=001, d=01, e=10.

Then we encode the text “cabde” as $\underbrace{001}_{c}\underbrace{11}_{a}\underbrace{000}_{b}\underbrace{01}_{d}\underbrace{10}_{e}$.

And this can be decoded uniquely easily and efficiently.

Decoding Tree

It is useful to represent a prefix code as a binary tree.

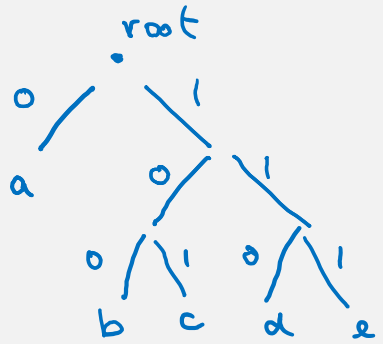


In the previous example, a=11, b=000, c=001, d=01, e=10

We can use the decoding tree to decode 001110000110.

b = 100 c = 101 d = 110 e = 111

Another example, a=0, b=~~101, c=110, d=101, e=110~~



It should be clear that there is a one-to-one correspondence

between binary decoding trees and binary prefix codes.

Optimal Prefix Code

Input: n symbols, with frequencies $f_1, f_2, \dots, f_n$ so that $\sum_{i=1}^{n} f_i = 1$.

objective function

Output: a binary decoding tree T with n leaves that minimizes $\left(\sum_{i=1}^{n} f_i \cdot depth_T(i)\ \cancel{= 1}.\right)$

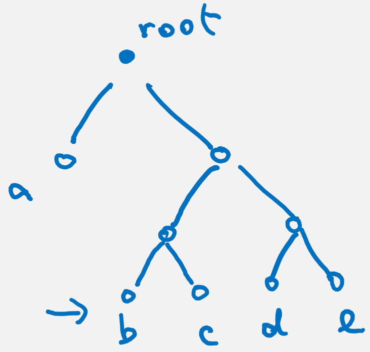
$\|$

depth of i in T

$\|$

length of encoded string of i

depth(a) = 1 →



depth(b) = 3 →

$f_a = 0.8$

$f_b = f_c = f_d = f_e = 0.05$

avg encoding length $= \sum_{i=1}^{n} f_i \cdot \text{length}(i) = \sum_{i=1}^{n} f_i \cdot depth_T(i)$

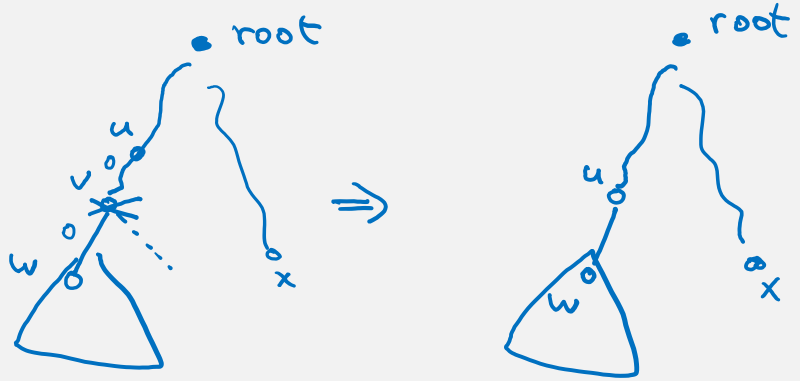
This problem doesn't look easy as the output space is more complicated.

It is also not clear how to make decisions greedily.

Full Binary Tree

A binary tree is **full** if every internal node has two children.

Observation. Any optimal binary decoding tree is full.



obj value

strictly smaller

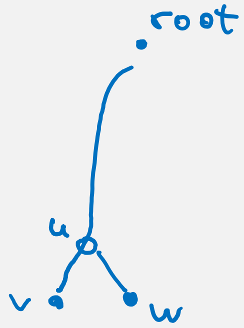
□

Corollary. There are at least two leaves of maximum depth that are siblings (i.e. having the same parent).

↑

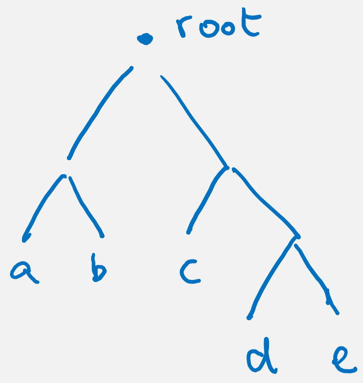
In an optimal tree

max depth →



Exchange Argument

Suppose we know the shape of an optimal binary tree (which we don't know yet).

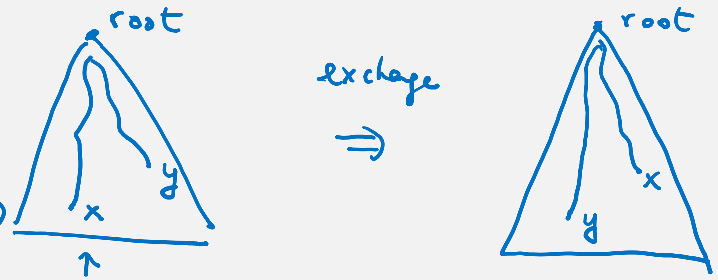


$f_a \geq f_b \geq f_c \geq \underline{f_d} \geq \underline{f_e}$

in general

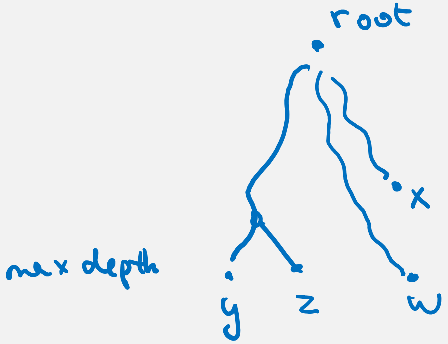
$f_x > f_y$

but $depth(x) > depth(y)$



$f_x \cdot depth(x) + f_y \cdot depth(y) > f_x \cdot depth'(x) + f_y\, depth'(y)$

Observation. There is an optimal solution in which the two symbols with lowest frequencies are assigned to leaves of maximum depth, and furthermore they are siblings.



f_y, f_z with lowest frequencies

$\square$

Today’s Plan

1. Compression, Optimal Prefix Code
2. Huffman’s Algorithm, Proof of Correctness

Huffman’s Idea

So far, we have deduced little information about how an optimal binary decoding tree should look like.

siblings

We know that there are 2 leaves of max depth, and we can assign symbols with lowest frequencies there.

We don’t know the shape of the tree, and don’t know how to use the frequencies to make decisions yet.

Perhaps surprisingly, Huffman realized that this is enough to design an efficient algorithm for the problem.

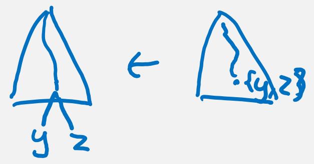
His idea is to **reduce** the problem size by one, by combining two symbols with lowest frequencies into one, knowing that they can be assumed to be siblings of maximum depth.

$f_y, f_z \qquad f_y+f_z$

$y, z \rightarrow \{y, z\}$

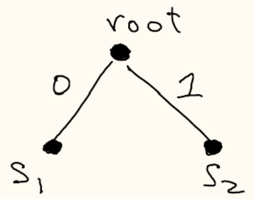
How the tree should look like will become apparent when the problem size becomes small enough.

And then we can construct back a bigger tree from a smaller tree one step at a time.



Huffman's Algorithm

Base case: If $|S| = 2$, encode one symbol using 0 and another symbol using 1, and return the tree.

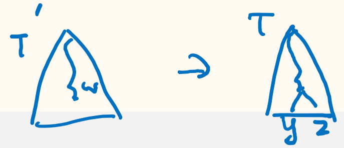


Induction step: Let y and z be two symbols with lowest frequencies, denoted by f_y and f_z.

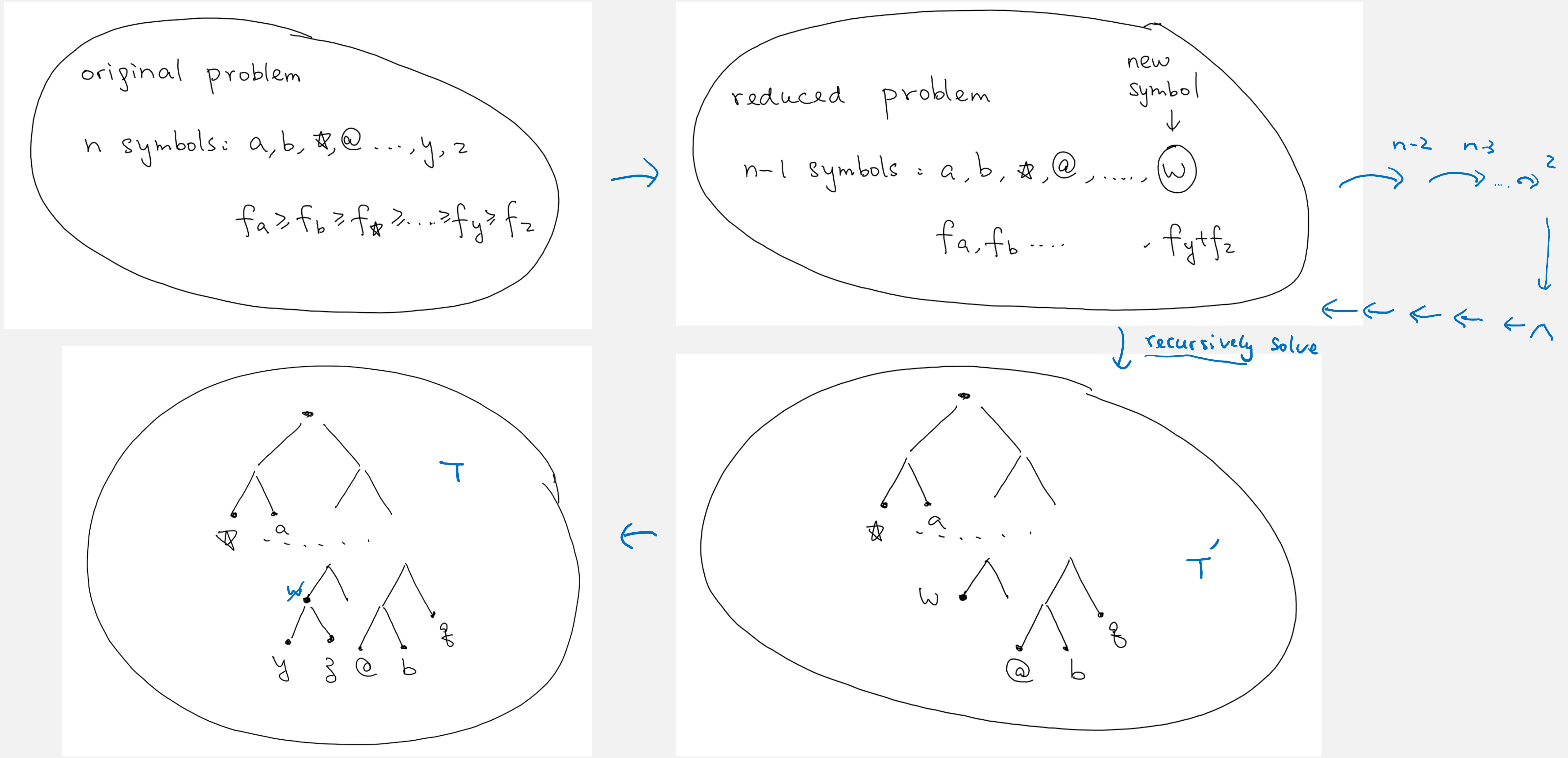
1. Delete symbols y and z from S. Add a new symbol w with frequency $f_y + f_z$.

2. Solve this new problem (with $n-1$ symbols) recursively and get an optimal tree T'.

3. In T', look at the leaf associated with w, add two leaves to it (so that w becomes an internal node), and associate y and z with the two new leaves.



Reduction Scheme



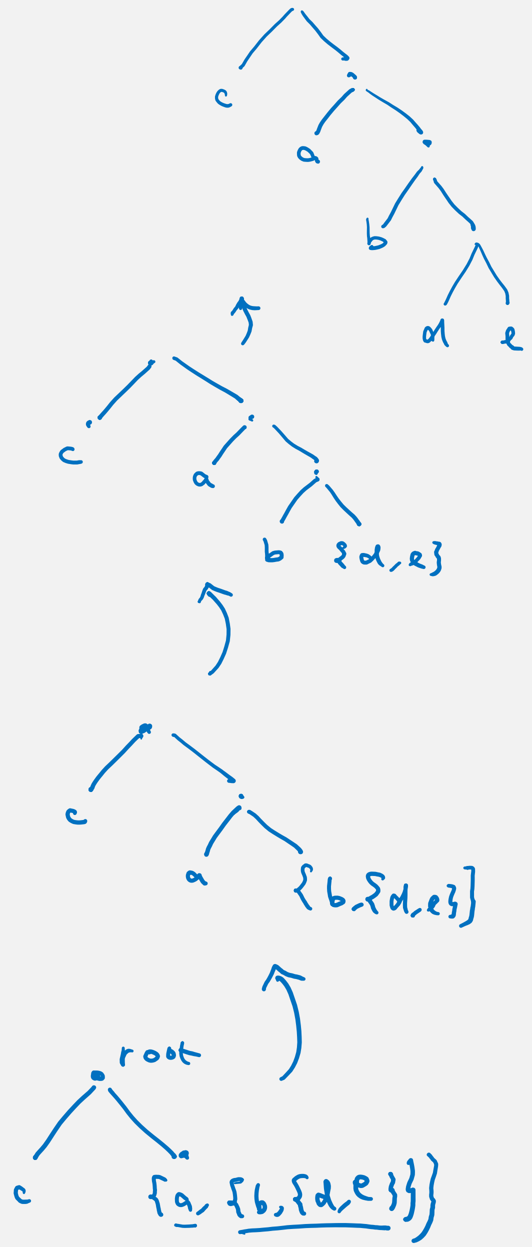
Example 1

Five symbols a, b, c, d, e, with $f_a = 0.3,\ f_b = 0.2,\ f_c = 0.4,\ \underline{f_d = 0.05},\ \underline{f_e = 0.05}$.

$a, b, c, \{d, e\}$ $\quad f_a = 0.3 \quad \underline{f_b = 0.2} \quad f_c = 0.4 \quad \underline{f_{\{d,e\}} = 0.1}$

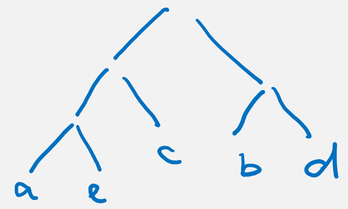
$a, c, \{b, \{d, e\}\}$ $\quad \underline{f_a = 0.3} \quad f_c = 0.4 \quad \underline{f_{\{b,\{d,e\}\}} = 0.3}$

$c, \{a, \{b, \{d, e\}\}\}$ $\quad f_x = 0.6 \quad f_c = 0.4$

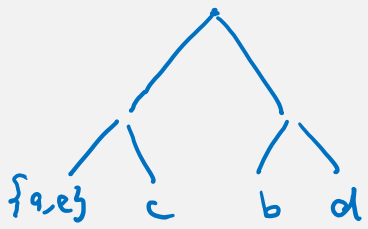


Example 2

Five symbols a, b, c, d, e, with $f_a = 0.18$, $f_b = 0.24$, $f_c = 0.26$, $f_d = 0.2$, $f_e = 0.12$.

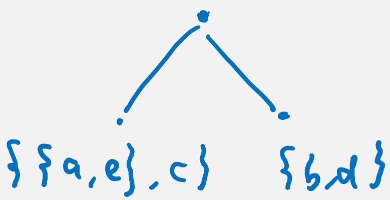


$\{a,e\}, b, c, d$ $\quad f_{\{a,e\}} = 0.3 \quad f_b = 0.24 \quad f_c = 0.26 \quad f_d = 0.2$



2 steps

$\{a,e\}, \{b,d\}, c$ $\quad f_{\{a,e\}} = 0.3 \quad f_{\{b,d\}} = 0.44 \quad f_c = 0.26$



$\{\{a,e\},c\}, \{b,d\}$ $\quad f_{\{\{a,e\},c\}} = 0.56 \quad f_{\{b,d\}} = 0.44$

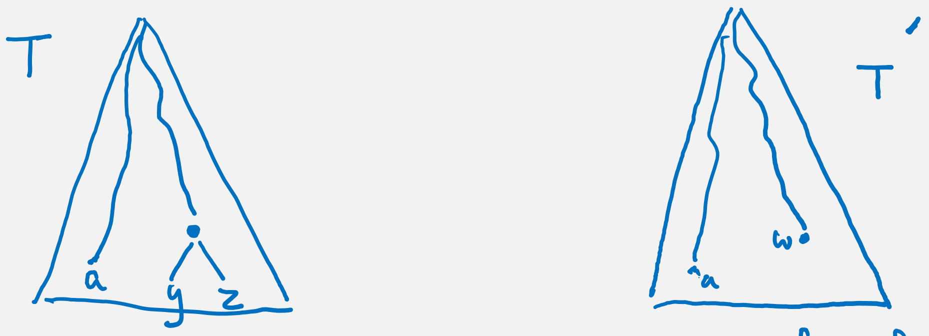
Correctness Proof

First we compute the objective value of our greedy solution.

prove by induction.

base case ✓

I.H: T' is an optimal decoding tree for the reduced problem



$f_w = f_y + f_z$

$$obj(T) = \underline{obj(T')} - f_w \cdot depth(w) + f_y \cdot (depth(w)+1) + f_z \cdot (depth(w)+1)$$

$$= \underbrace{obj(T') + f_y + f_z}.$$

next, want to argue any opt tree has obj value at least this RHS.

Correctness Proof

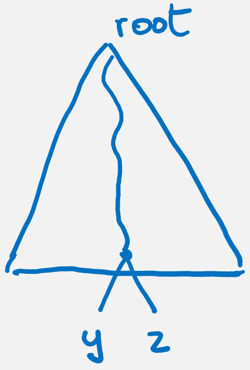
Next we compute the objective value of any solution.

↑ optimal

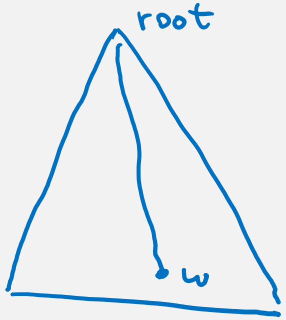
T^*

opt solution

exchange argument



$\Longrightarrow$



$f_w = f_y + f_z$

$T^{*\prime}$

reduced problem

① $T^{*\prime}$ is a feasible solution to the reduced problem

$$\mathrm{obj}(T') \le \mathrm{obj}(T^{*\prime})$$

② $$\mathrm{obj}(T^*) = \mathrm{obj}(T^{*\prime}) + f_y + f_z$$

① + ② $$\mathrm{obj}(T^*) \overset{②}{=} \mathrm{obj}(T^{*\prime}) + f_y + f_z \overset{①}{\ge} \mathrm{obj}(T') + f_y + f_z \overset{\text{previous slide}}{=} \mathrm{obj}(T) \quad \square$$

Implementation

In each iteration, we need to find two symbols of lowest frequencies,

delete them and add a new symbol with the frequency as their sum.

straightforward: $O(n)$ time to find the two symbols of lowest frequencies.

heap: insert, extract-min / delete $O(\log n)$ time.

initially. insert $f_1 \dots f_n$ into heap: $O(n \log n)$ time.

in each iteration, extract-min twice $O(\log n)$ f_y, f_z

add them, insert $f_z + f_y$ $O(\log n)$

total time complexity $O(n \log n)$