Huffman Encoding

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

Encoding : "Character" ⟶ "Codeword"

"How to design a code minimizing length of messages

Decoding : "Codeword" ⟶ "Character"

We want: A message is uniquely decodable

"Prefix-free code"

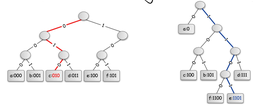
e.g. { a:0 , b:110 , c:10 , d:111 }. is prefix-free

since no codeword is a prefix (開頭) of another codeword

{ a:0 , b:01 , } ⟶ Not prefix-free (a is a prefix of b)

⇒ Problem: For a given input file , find the prefix-free code that result in the smallest encoded messages ("Compression")

Correspondence : Binary Tree and Prefix Codes



① Codeword (associated with the character on the leaf node)
= Binary string read of from the path : root → leaf node.

② Length of the codeword = Depth of leaf node

Problem Restated :

Given an alphabet A of n characters $a_1, a_2 \cdots a_n$

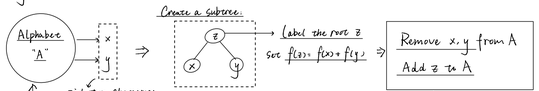
with weight $f(a_1), f(a_2) \cdots f(a_n)$, find a binary tree with n leaves

labeled $a_1, a_2, a_3 \cdots, a_n$ Such that :

$B(T) = \sum_{i=1}^{n} f(a_i) d(a_i)$ is minimized

depth of node a_i ⇒ Length of corresponding codeword

"Greedy Idea" :



Repeat this procedure ("merge")
until only One character left in A

The algorithm :

Huffman(A): "Min-heap"

create a min-priority queue Q on A , with weight as key

for i ← 1 to n-1 :

allocate a new node z

x ⟵ Extract-min(Q)

y ⟵ Extract-min(Q)

z.left ← x

z.right ← y

z.weight ⟵ x.weight + y.weight.

Insert(Q, z)

return Extract-Min(Q) Actually there is only 1 element in Q (i.e. the root of the tree)

Running Time : O(nlogn)

Proof : Correctness

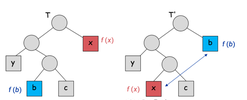
Lemma 1: Any optimal tree must to be "full". i.e.
every internal node must have two children

Pf: If some internal node had only one child,

then we could simply get rid of this node, replacing it with its child.
This would decrease the total cost of the encoding

Lemma 2: Let T be prefix code tree and T' be another obtained from T
by swapping two leaf node x and b

⇒ if $f(x) \le f(b)$, $d(x) \le d(b)$, then $B(T') \le B(T)$



x is more frequent than b
if $\ell m(\text{codeword}(x)) > \ell$. Then $B(T') >$

Proof: $B(T') = B(T) - f(b)d(b) - f(x)d(x) + f(b)d(x) + f(x)d(b)$

$B(T') - B(T) = f(b)(d(x) - d(b)) - f(x)(d(x) - d(b))$

$= (f(b) - f(x))(d(x) - d(b)) \le 0$

⇒ $B(T') \le B(T)$

Lemma 3 : Consider the two characters x, y with the smallest frequency
There is an optimal tree in which these two characters are siblings
leaves at the deepest level of the tree

Pf: Let T be any optimal prefix code tree, b and c be two siblings at the deepest level of the tree (must exist because T is full).

Assume without loss of generality that $f(x) \le f(b)$ and $f(y) \le f(c)$

- (If necessary) swap x with b and swap y with c.
- Proof follows from Lemma 2.
  (Lemma 2 ⇒ cost can't increase ⇒ since old tree optimal, new one is also)

Lemma 4 : Let T be a prefix code tree in which x, y are two sibling leaves
Let T' be obtained from T by removing x and y, naming their parent z,
and setting $f(z) = f(x) + f(y)$

Then : $B(T) = B(T') + f(x) + f(y)$

Proof : $B(T) = B(T') - f(z)d(z) + (f(x) + f(y))(d(z) + 1)$

$= B(T') - f(z)d(z) + f(z)(d(z) + 1)$

$= B(T') + f(z) = B(T') + f(x) + f(y)$

Theorem : The huffman tree is optimal

Proof by induction :

Base case : n = 2 . Tree with 2 leaves , obviously optimal

Inductive Hypothesis : Huffman's algorithm produce optimal
prefix tree for any inputs of n-1 characters

Inductive Step : Consider input of n characters

Let H be the tree produced by Huffman's Alg. Want to Show : H is optimal

Let A be the alphabet with n characters. (i.e. Alphabet for H).

- From operation of Huffman's algorithm:
  - There exist two characters x and y with two smallest frequencies that are sibling leaves in H.
- Let H' be obtained from H by
  (i) removing x and y,
  (ii) naming their parent z, and
  (iii) setting f(z) = f(x) + f(y)

⇒ Let A' be the alphabet for H' : $A' = A \cup \{z\} - \{x, y\}$

H is the tree produced by Huffman's algorithm on alphabet A

H' is the tree produced by Huffman's algorithm on alphabet A'

By Hypothesis. H' is optimal

By Lemma 3. There is an optimal tree T in which x, y are siblings
leaves at the deepest level of the tree

Let T' be obtained from T by
(i) removing x, y,
(ii) naming the parent z, and
(iii) setting f(z) = f(x) + f(y).
T' is a prefix code tree for alphabet A'.

⇒ $B(T) = B(T') + f(x) + f(y)$. (Lemma 4)

⇒ $B(H) = B(H') + f(x) + f(y)$

$\le B(T') + f(x) + f(y)$

$= B(T)$

⇒ $B(H) \le B(T)$, since T is optimal, H is optimal

Optimal 2-Way Merge

Given n Sorted Lists $L_1, L_2, L_3 \cdots L_n$ with Corresponding size

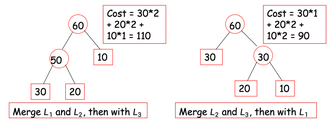
Want to combine them into 1 Sorted List

Find : an optimal merge pattern ⇒ minimize # comparisons

⇒ Equivalent Problem:

A set of leaf nodes $a_1, \cdots a_n$, with associated weights ("size of the list") $w(a_1) \cdots w(a_n)$

Create a binary tree with these leaf nodes , minimizing $B(T) = \sum_{i=1}^{n} w(a_i) d(a_i)$ (depth of a_i)



Algorithm (Similar to Huffman Encoding)

Create a min-heap T[1...n] base on n initial size

While (heap size > 1) do :

x ⟵ T.extract-min( )

y ⟵ T.extract-min( )

create node z whose children are a, b.

size(z) ⟵ size(a) + size(b)

T.insert(z)

return T.extract-min( )