Huffman's Coding

Tao Hou

Huffman Coding: Setting

- Suppose you need to send a large text to another party
- Let's say, the text consists of six characters, each of which has a frequency of appearing in the text:

sym.	*freq.*
a	45%
b	13%
c	12%
d	16%
e	9%
f	5%

- We would like to encode the text as a sequence of bits while ensuring that:
  1. we should be able to decode text *without ambiguity*, with an encoding table given
  2. the length of the encoded binary string is *minimal*, or in another word, the average no. of bits for encoding each character is the minimal

Fixed Length Encoding

- A first attempt is to assign each character the same code length
- We have 6 characters, so we need a code of length 3 (bits)
- Such an encoding is referred to as a *fixed-length* encoding

sym.	*freq.*	*code*
a	45%	000
b	13%	001
c	12%	010
d	16%	011
e	9%	100
f	5%	101

- Average number of bits to encode a character: 3

Variable-length Encoding

- Can we do better?
- If we can, then we need to forsake the use of fixed-length encoding, and allow character codes to have different lengths
- Such a code is called a *variable-length* code
- Intuitively, we would like to assign characters with high frequencies short codes
- Let’s try that

Variable-length Encoding

sym.	*freq.*	*code*
a	45%	0
b	13%	1
c	12%	00
d	16%	01
e	9%	10
f	5%	11

- Average no. of bits:

$$.45 \times 1 + .13 \times 1 + .12 \times 2 + .16 \times 2 + .9 \times 2 + .5 \times 2 = 1.42$$

Variable-length Encoding

sym.	*freq.*	*code*
a	45%	0
b	13%	1
c	12%	00
d	16%	01
e	9%	10
f	5%	11

- Average no. of bits:

$$.45 \times 1 + .13 \times 1 + .12 \times 2 + .16 \times 2 + .9 \times 2 + .5 \times 2 = 1.42$$

- There is a problem! The above encoding is ambiguous: The text can be decoded in multiple ways

- One way for ensuring decodability in a variable-length code is to require the code to be a *prefix code*: No code is a prefix of another

sym.	*freq.*	*code*
a	45%	000
b	13%	001
c	12%	010
d	16%	011
e	9%	10
f	5%	11

Why Prefix Codes Work?

Consider an algorithm for decoding the bit sequence $b[1, \ldots, n]$:

1. Maintain a variable i which is the index of the 'next' bit you are scanning at a certain step, i.e., we have scanned all bits $b[1, \ldots, i-1]$ and decoded the texts in them
2. Initially, $i = 0$.
3. Do the following until $i > n$:
   3.1 Find an $\ell \geq i$ such that $b[i, \ldots, \ell]$ matches a code in the table, and output the corresponding character
   3.2 Let $i = \ell + 1$

Why Prefix Codes Work?

Consider an algorithm for decoding the bit sequence $b[1, \ldots, n]$:

1. Maintain a variable i which is the index of the 'next' bit you are scanning at a certain step, i.e., we have scanned all bits $b[1, \ldots, i-1]$ and decoded the texts in them
2. Initially, $i = 0$.
3. Do the following until $i > n$:
   - 3.1 Find an $\ell \geq i$ such that $b[i, \ldots, \ell]$ matches a code in the table, and output the corresponding character
   - 3.2 Let $i = \ell + 1$

We can use proof by contradiction to show that the above process has no ambiguity (i.e., each step finds a unique ℓ) if the bits are encoded with a prefix code

Problem Definition

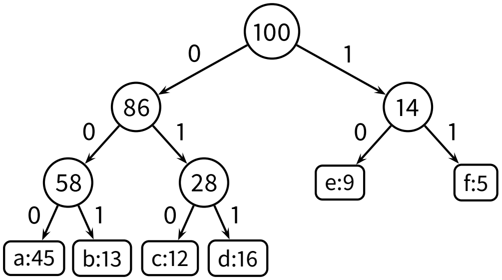
Given a set of characters C and a frequency function $f : C \longrightarrow \mathbb{R}$, compute a prefix-code that minimizes

$$\sum_{c \in C} f(c) \times \text{code-length}(c)$$

Prefix Codes & Binary Trees

- Any prefix code can be represented as a binary tree *T* whose leaves represent the characters
- An edge between a tree node and its left (resp. right) child is labeled 0 (resp. 1)
- the code of a character is the concatenation of the labels on the path from the root to the leaf corresponding to the character

sym.	*freq.*	*code*
a	45%	000
b	13%	001
c	12%	010
d	16%	011
e	9%	10
f	5%	11



More on Prefix Codes & Binary Trees

- The algorithm to build such a binary tree: For each character c, we can build a path starting with the root, ending with c, with the edges going to left or right based on the 0/1 in the code of c
- We then combine all such paths to get the binary tree
- Since no code of a character is the prefix of another, we have that no node of a character in the tree is an ancestor of another $\Rightarrow$ all characters correspond to leaves
- So, there is a ***one-to-one*** correspondence from all such prefix codes to all the binary trees

Problem Reformulation

Given a set of characters C and a frequency function $f : C \longrightarrow \mathbb{N}$, construct a binary tree T whose leaves are the characters that minimizes

$$Cost(T) = \sum_{c \in leaves(T)} f(c) \times depth(c)$$

Huffman Algorithm: Ideas

- We work on a bunch of binary trees, whose leaves correspond to the characters
- Each tree has a ***frequency***, which is the sum of the frequency of all characters its leaves correspond to

Huffman Algorithm: Ideas

- We work on a bunch of binary trees, whose leaves correspond to the characters
- Each tree has a ***frequency***, which is the sum of the frequency of all characters its leaves correspond to
- Initially, each character corresponds to a tree with a single node (frequency is obvious)

Huffman Algorithm: Ideas

- We work on a bunch of binary trees, whose leaves correspond to the characters
- Each tree has a ***frequency***, which is the sum of the frequency of all characters its leaves correspond to
- Initially, each character corresponds to a tree with a single node (frequency is obvious)
- Then, in each iteration, we extract two trees with the minimum frequency, and merge the two tree into one by letting them sharing a new root (the frequency will be summed): This is the ***greedy choice***
- We do this until we are left with only a single tree

Huffman Algorithm

```
HUFFMAN(C, f)
    Q = empty heap
    for each c ∈ C
        create an isolated tree node z
        z.char = c
        z.freq = f(c)
        HEAP-INSERT(Q, z, z.freq)
    for i = 1 to n − 1
        create a new tree node z
        z.left = EXTRACT-MIN(Q)
        z.right = EXTRACT-MIN(Q)
        z.freq = z.left.freq + z.right.freq
        HEAP-INSERT(Q, z, z.freq)
    return EXTRACT-MIN(Q)
```

Huffman Algorithm

```
HUFFMAN(C, f)
    Q = empty heap
    for each c ∈ C
        create an isolated tree node z
        z.char = c
        z.freq = f(c)
        HEAP-INSERT(Q, z, z.freq)
    for i = 1 to n − 1
        create a new tree node z
        z.left = EXTRACT-MIN(Q)
        z.right = EXTRACT-MIN(Q)
        z.freq = z.left.freq + z.right.freq
        HEAP-INSERT(Q, z, z.freq)
    return EXTRACT-MIN(Q)
```

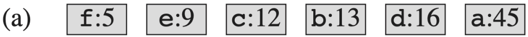
- Time complexity:

Huffman Algorithm

```
HUFFMAN(C, f)
Q = empty heap
for each c ∈ C
        create an isolated tree node z
        z.char = c
        z.freq = f(c)
        HEAP-INSERT(Q, z, z.freq)
for i = 1 to n − 1
        create a new tree node z
        z.left = EXTRACT-MIN(Q)
        z.right = EXTRACT-MIN(Q)
        z.freq = z.left.freq + z.right.freq
        HEAP-INSERT(Q, z, z.freq)
return EXTRACT-MIN(Q)
```

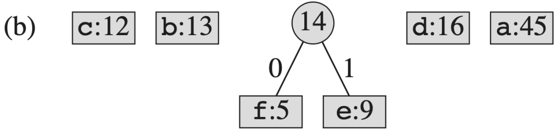
- Time complexity: $O(n \log n)$

Example



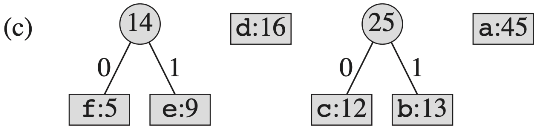
(Example from CLRS)

Example



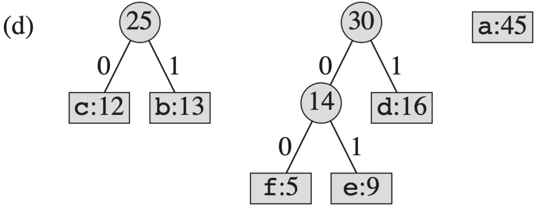
(Example from CLRS)

Example



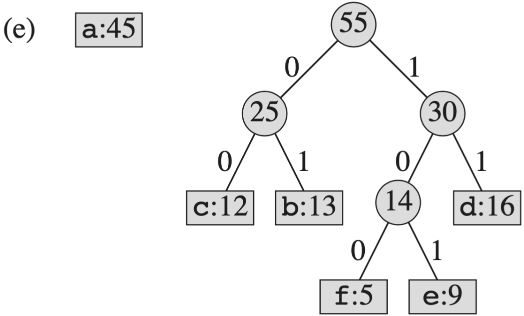
(Example from CLRS)

Example



(Example from CLRS)

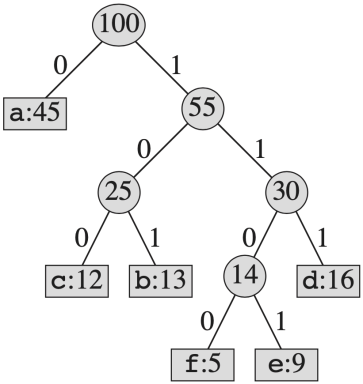
Example



(Example from CLRS)

Example

(f)



(Example from CLRS)

Lemma 1

Assumption:

- Let C be an alphabet where each character $c \in C$ has frequency $f(c)$.
- Let x and y be two characters in C with minimum frequency.
- Construct another alphabet C' from C by removing x and y and adding a new character z.
  - Define f for C' as for C, except that $f(z) = f(x) + f(y)$.
- Let T' be any binary tree representing an optimal prefix code for the alphabet C'.

Conclusion:

- Then the tree T, obtained from T' by replacing the leaf node for z with an internal node having x and y as children, represents an optimal prefix code for the alphabet C.

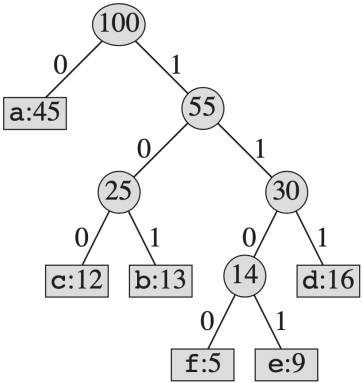
Implication of the Lemma

- For the tree T built by Huffman's algorithm, starting from the simplest form containing a single node, we can iteratively expand the tree by replacing a leaf with an internal node plus two leaves, and eventually arrive at T s.t. each tree we have along the expansion is an optimal tree for codes we have.

Implication of the Lemma

- For the tree T built by Huffman's algorithm, starting from the simplest form containing a single node, we can iteratively expand the tree by replacing a leaf with an internal node plus two leaves, and eventually arrive at T s.t. each tree we have along the expansion is an optimal tree for codes we have.
- Let's work on the example using board:

(f)



Proof of Lemma 1

Proving Lemma 1 needs two propositions.

Proposition 1

- Let C be an alphabet where each character $c \in C$ has frequency $f(c)$.
- Let x and y be two characters in C with minimum frequency.
- Then there exists an optimal binary tree for C where x and y are siblings (children of the same parent).

Proposition 1

- Let C be an alphabet where each character $c \in C$ has frequency $f(c)$.
- Let x and y be two characters in C with minimum frequency.
- Then there exists an optimal binary tree for C where x and y are siblings (children of the same parent).

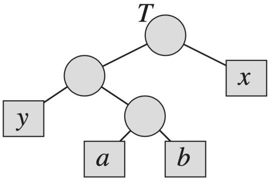
Idea of the proof:

- Take an optimal tree T for C and modify it to make another optimal tree T'' where x and y are sibling leaves of maximum depth.

Proof of Proposition 1

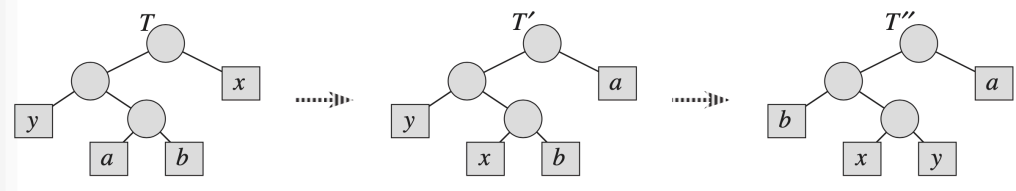
- Let a and b be two characters that are sibling leaves of maximum depth in T.
- WLOG, we can assume $f(a) \leq f(b)$ and $f(x) \leq f(y)$.
- Since $f(x)$ and $f(y)$ are the two min frequencies, we have $f(x) \leq f(a)$ and $f(y) \leq f(b)$.

- Let a and b be two characters that are sibling leaves of maximum depth in T.
- WLOG, we can assume $f(a) \leq f(b)$ and $f(x) \leq f(y)$.
- Since $f(x)$ and $f(y)$ are the two min frequencies, we have $f(x) \leq f(a)$ and $f(y) \leq f(b)$.
- To make the arguments easier, assume $x, y \neq a, b$, e.g.:

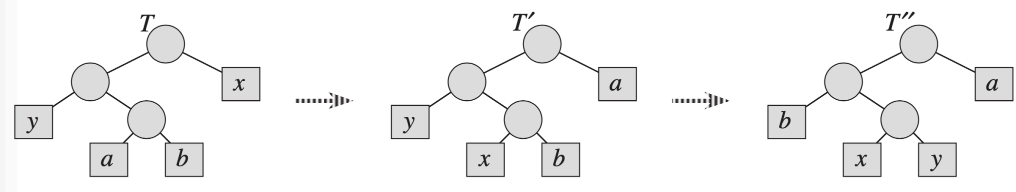


  - The case of (one of) x, y could be equal to (one of) a, b is trickier and we omit (focus on the big picture)

- We switch x with a, y with b, to produce the tree T'' we want, where the avg code length cannot increase, so that T'' is also an optimal binary tree:

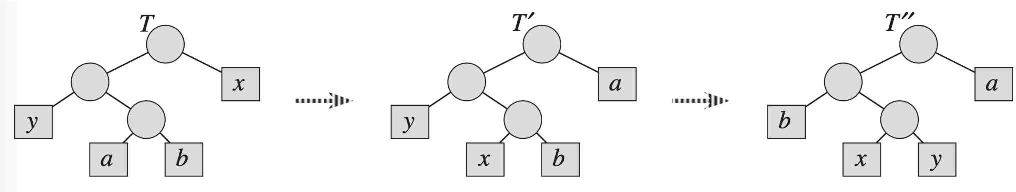


- We switch x with a, y with b, to produce the tree T'' we want, where the avg code length cannot increase, so that T'' is also an optimal binary tree:



- $$\begin{aligned} Cost(T) - Cost(T') &= \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c) && (1) \\ &= f(x)d_T(x) + f(a)d_T(a) - f(x)d_{T'}(x) - f(a)d_{T'}(a) && (2) \\ &= f(x)d_T(x) + f(a)d_T(a) - f(x)d_T(a) - f(a)d_T(x) && (3) \\ &= (f(a) - f(x))(d_T(a) - d_T(x)) \geq 0 && (4) \end{aligned}$$

- We switch x with a, y with b, to produce the tree T'' we want, where the avg code length cannot increase, so that T'' is also an optimal binary tree:



-

$$\begin{aligned} Cost(T) - Cost(T') &= \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c) && (1) \\ &= f(x)d_T(x) + f(a)d_T(a) - f(x)d_{T'}(x) - f(a)d_{T'}(a) && (2) \\ &= f(x)d_T(x) + f(a)d_T(a) - f(x)d_T(a) - f(a)d_T(x) && (3) \\ &= (f(a) - f(x))(d_T(a) - d_T(x)) \geq 0 && (4) \end{aligned}$$

- $Cost(T) \geq Cost(T') \geq Cost(T'')$

Proposition 2

- Let T' be a binary tree (corresponding to a prefix code) with a leaf z
- Let tree T be obtained from T' by replacing z with an internal node having x and y as children, s.t. $f(z) = f(x) + f(y)$.
- Then, $Cost(T') = Cost(T) - f(x) - f(y)$

Proposition 2

- Let T' be a binary tree (corresponding to a prefix code) with a leaf z
- Let tree T be obtained from T' by replacing z with an internal node having x and y as children, s.t. $f(z) = f(x) + f(y)$.
- Then, $Cost(T') = Cost(T) - f(x) - f(y)$

Proof:

-

$$\begin{aligned} Cost(T) - Cost(T') &= \sum_{c \in C} f(c)d_T(c) - \sum_{c \in C'} f(c)d_{T'}(c) && (5) \\ &= f(x)d_T(x) + f(y)d_T(y) - f(z)d_{T'}(z) && (6) \end{aligned}$$

Proposition 2

- Let T' be a binary tree (corresponding to a prefix code) with a leaf z
- Let tree T be obtained from T' by replacing z with an internal node having x and y as children, s.t. $f(z) = f(x) + f(y)$.
- Then, $Cost(T') = Cost(T) - f(x) - f(y)$

Proof:

-

$$\begin{aligned} Cost(T) - Cost(T') &= \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C'} f(c) d_{T'}(c) && (5) \\ &= f(x) d_T(x) + f(y) d_T(y) - f(z) d_{T'}(z) && (6) \end{aligned}$$

- Since $d_T(x) = d_T(y) = d_{T'}(z) + 1$ and $f(z) = f(x) + f(y)$, We have:

$$\begin{aligned} Cost(T) - Cost(T') &= (f(x) + f(y))(d_{T'}(z) + 1) - (f(x) + f(y)) d_{T'}(z) && (7) \\ &= f(x) + f(y) && (8) \end{aligned}$$

Proof of Lemma 1

- Use proof by contradiction. Suppose that T is not an optimal binary tree for C.
- Then there exists an optimal tree T'' for C such that $Cost(T'') < Cost(T)$.

Proof of Lemma 1

- Use proof by contradiction. Suppose that T is not an optimal binary tree for C.
- Then there exists an optimal tree T'' for C such that $Cost(T'') < Cost(T)$.
- Since x and y have minimum frequency in C, we can also let T'' be an optimal tree where x and y are siblings (by Proposition 1).

- Use proof by contradiction. Suppose that T is not an optimal binary tree for C.
- Then there exists an optimal tree T'' for C such that $Cost(T'') < Cost(T)$.
- Since x and y have minimum frequency in C, we can also let T'' be an optimal tree where x and y are siblings (by Proposition 1).
- Let T''' be derived from T'' with the common parent of x and y replaced by a leaf z with $f(z) = f(x) + f(y)$.
    - T''' is a binary tree for C'

- Use proof by contradiction. Suppose that T is not an optimal binary tree for C.
- Then there exists an optimal tree T'' for C such that $Cost(T'') < Cost(T)$.
- Since x and y have minimum frequency in C, we can also let T'' be an optimal tree where x and y are siblings (by Proposition 1).
- Let T''' be derived from T'' with the common parent of x and y replaced by a leaf z with $f(z) = f(x) + f(y)$.
  - T''' is a binary tree for C'
-

$$Cost(T''') \tag{9}$$

$$= Cost(T'') - f(x) - f(y) \tag{10}$$

Proof of Lemma 1

- Use proof by contradiction. Suppose that T is not an optimal binary tree for C.
- Then there exists an optimal tree T'' for C such that $Cost(T'') < Cost(T)$.
- Since x and y have minimum frequency in C, we can also let T'' be an optimal tree where x and y are siblings (by Proposition 1).
- Let T''' be derived from T'' with the common parent of x and y replaced by a leaf z with $f(z) = f(x) + f(y)$.
  - T''' is a binary tree for C'
-

$$Cost(T''') \tag{9}$$

$$= Cost(T'') - f(x) - f(y) \tag{10}$$

$$< Cost(T) - f(x) - f(y) \tag{11}$$

- Use proof by contradiction. Suppose that T is not an optimal binary tree for C.
- Then there exists an optimal tree T'' for C such that $Cost(T'') < Cost(T)$.
- Since x and y have minimum frequency in C, we can also let T'' be an optimal tree where x and y are siblings (by Proposition 1).
- Let T''' be derived from T'' with the common parent of x and y replaced by a leaf z with $f(z) = f(x) + f(y)$.
  - T''' is a binary tree for C'
- $$\begin{aligned} & Cost(T''') && (9) \\ & = Cost(T'') - f(x) - f(y) && (10) \\ & < Cost(T) - f(x) - f(y) && (11) \\ & = Cost(T') && (12) \end{aligned}$$

  where (9)-(10), (11)-(12) are by Proposition 2.
- This contradicts the fact that T' is an optimal binary tree for C'

Acknowledgement

- Certain figures in the slides are taken from [CLRS]