Huffman Codes (data compression)

- Data compression is an important technique for saving storage
- Given a file,
  - We can consider it as a string of characters
  - We want to find a compressed file
  - The compressed file should be as small as possible
  - The original file can be reconstructed from the compressed file
- This is useful when access to the file is infrequent (most files are this kind of files!)
  So we do not care about the work of compressing and decompressing.
- It is important in data communication where the cost of sending information is greater than the cost of processing them.
- We just show one special case of data compression: Huffman codes.

We assume that a file is a sequence of English letters.
Each letter is represented by a unique string of bits, 0 or 1, (this is called a code).

The set of all the codes is called an encoding.

Fixed length encoding
If the lengths of all codes are the same, then the number of bits representing the file only depends on the number of characters in the file.

Example: In ASCII

A is 1000001, B is 1000010 and so on
(every letter uses 7 bits)

Variable length encoding
However, it is possible to use variable length encoding.
We use fewer bits to represent letters that appear very often (for example A).
We use more bits to represent letters that appear less often (for example Z).
This way we may save space.

One example

- For simplicity, suppose that we only have four letters, A,B,C,D. We can code them with 00, 01, 10, and 11.
- String ABAACAADA is encoded as:
  00 01 00 00 10 00 00 11 00.
- Can still decode without the delimiters:
  000100001000001100
- Because A occurs much more frequently, encode A with one bit while the others also change. I.e., A,B,C,D $\rightarrow$ 1, 01, 001, 000. Then the string becomes
  1 01 1 1 001 1 1 000 1.
- We can still decode without the delimiters – as long as we know where to start.
  10111001110001.
- Not an arbitrary encoding can be decoded.

Prefix Constraint

In order to use variable-length encoding, we have to be able to uniquely recognize code from a string of bits.

Example: $A : 1001$, $B : 100101$

10010110
Is this a code for A or part of a code for B?

- In general, *the prefixes of a code must not equal the complete code of another code.*

  This is called **the prefix constraint**.
- With this constraint, we can scan the bits from left to right and determine the codes for letters.
- In general when we shorten the code for one letter, we may have to lengthen the code for another letter.
- Our goal is to find the best balance

The Problem: Given a text (a sequence of characters), find an encoding for all the characters that satisfies the prefix constraint and that minimizes the total number of bits needed to encode the text.

First we compute the number of times each character appears in the text. We call this value the frequency of the character. (In many cases we can use a standard frequency table.)

- Let $c_1, c_2, \ldots, c_n$ be the characters
- Let $f(c_1), f(c_2), \ldots, f(c_n)$ be their frequencies.
- Let T be an encoding method.
- Let $d_T(c_i)$ be the length of the code for c_i under T.
- The length, in bits, of the file compressed by using T is:

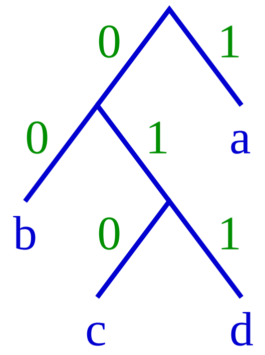
$$B(T) = \sum_{i=1}^{n} f(c_i) d_T(c_i)$$

- Our goal is to find an encoding T that satisfies prefix constraint and minimizes $B(T)$.

Tree representation of encoding

Given an encoding of n characters satisfying the prefix constraint, we can construct a binary tree to represent this encoding.

Example: 1 for a, 00 for b, 010 for c, 011 for d



Each code is a path from root to a leaf. 0 means go left, 1 means go right. Each leaf represents a character.

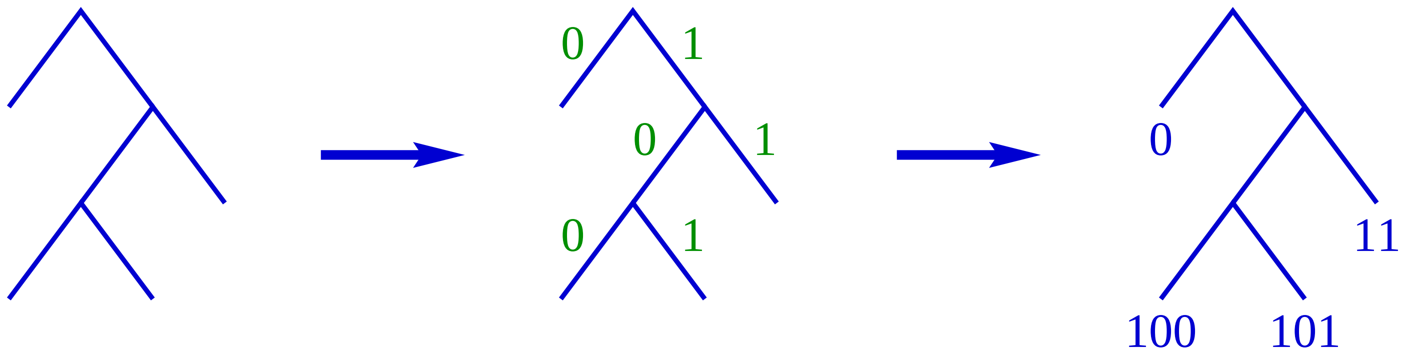
The number of leaves is n.

Given an n-leave binary tree, it represents an encoding of n characters satisfying prefix constraint.

We label edge to the left with 0.
We label edge to the right with 1.

Each leaf corresponds to one code for a character.

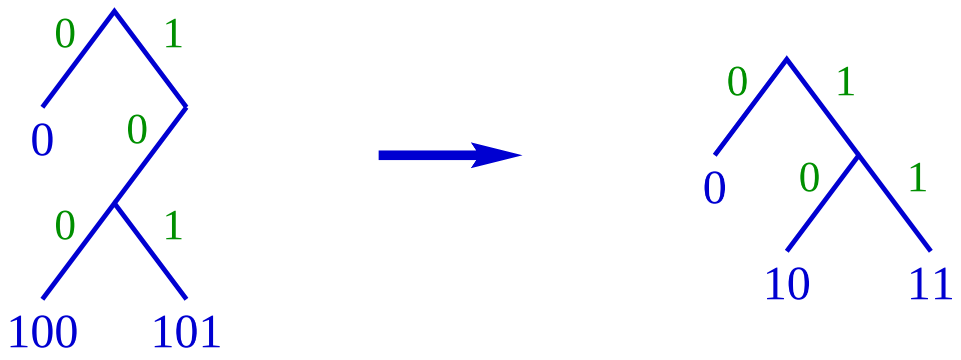
The code is the string when we travel from root to this leaf.



- An encoding of n characters satisfying prefix constraint is equivalent to an n-leave binary tree.

An optimal encoding is always represented by a 2-tree.

Why?
If one internal node has only one child, we can delete this internal node, therefore shortening the encoding for some characters.



In an optimal encoding T, there are x and y such that $d_T(x) = d_T(y)$ is maximal and their codes only differ in the last bit.

E.G. Consider the binary 2-tree of this optimal encoding. Let its height be h. Let N be an internal node of level $h-1$. Then the children of N have the above property.

Let x and y be two characters with lowest frequencies. Then there is an optimal encoding T' such that $d_{T'}(x) = d_{T'}(y)$ is maximal and their codes differ only in the last bit.

Consider an optimal encoding T.
There are a and b such that $d_T(a) = d_T(b)$ is maximal and their codes only differ in the last bit.
Now, assuming that $f(x) \le f(y)$ and $f(a) \le f(b)$, we construct a new encoding T' by exchanging the codes in T for x and a, and for y and b.

$$B(T) - B(T') = \sum_{t=1}^{n} f(c_t)d_T(c_t) - \sum_{t=1}^{n} f(c_t)d_{T'}(c_t)$$
$$= \quad f(x)d_T(x) + f(y)d_T(y) + f(a)d_T(a) + f(b)d_T(b)$$
$$- (f(x)d_{T'}(x) + f(y)d_{T'}(y) + f(a)d_{T'}(a) + f(b)d_{T'}(b))$$
$$= \quad f(x)d_T(x) + f(y)d_T(y) + f(a)d_T(a) + f(b)d_T(b)$$
$$- f(x)d_T(a) - f(y)d_T(b) - f(a)d_T(x) - f(b)d_T(y)$$
$$= \quad f(x)(d_T(x) - d_T(a)) + f(a)(d_T(a) - d_T(x)) + f(y)(d_T(y) - d_T(b)) + f(b)(d_T(b) - d_T(y))$$
$$= (f(a) - f(x))(d_T(a) - d_T(x)) + (f(b) - f(y))(d_T(b) - d_T(y)) \ge 0$$
which means T' is an optimal encoding.

Can we merge x and y and then use induction?

Let x and y be two characters with lowest frequencies. We now consider an encoding T such that x and y are two leaves sharing the same parent.
Let z be a new character with frequency $f(z) = f(x) + f(y)$.
If we delete two leaves corresponding to x and y from T, the resulting tree is an encoding, T', for

$$\{c_1, c_2, \ldots, c_n\} \setminus \{x, y\} \cup \{z\}.$$

And $B(T) = B(T') + f(x) + f(y)$ since

$$B(T) = \sum_{k=1}^{n} f(c_k) d_T(c_k) =$$

$$\sum_{k=1, c_k \neq x, c_k \neq y}^{n} f(c_k) d_T(c_k) + f(x) d_T(x) + f(y) d_T(y)$$

$$= B(T') - f(z) d_{T'}(z) + f(x) d_T(x) + f(y) d_T(y)$$

$$= B(T') - f(z) d_{T'}(z) + f(x)(d_{T'}(z) + 1) + f(y)(d_{T'}(z) + 1)$$

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

Conclusion: to minimize $B(T)$, we only need to minimize $B(T')$.

Let T' be an optimal encoding for

$$\{c_1, c_2, \ldots, c_n\} \setminus \{x, y\} \cup \{z\}$$

Then we can get an encoding T for

$$\{c_1, c_2, \ldots, c_n\}$$

by adding 0 to code of z for x and adding 1 to code of z for y.

Keep all other codes unchanged.

It is clear that T is an encoding such that x and y are sharing the same parent and $B(T) = B(T') + f(x) + f(y)$.

Theorem. T is optimal.
Proof.
Since $B(T')$ is minimal, we know that $B(T)$ is minimal among all encodings such that x and y have the same parent.

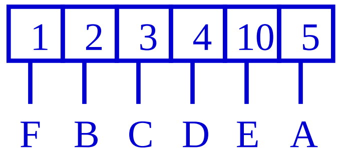
Since we know optimal encodings for $\{c_1, c_2, \ldots, c_n\}$ are among this kind of encodings, we know T is optimal.

$\boxed{\text{Algorithm by induction}}$

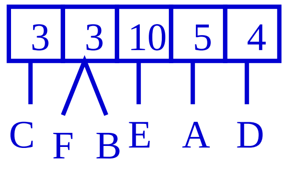
- We want to solve problem for $\{c_1, c_2, \ldots, c_n\}$
- Find x, and y with lowest frequencies.
- Solve problem for
$$\{c_1, c_2, \ldots c_n\} \setminus \{x, y\} \cup \{z\}$$
where $f(z) = f(x) + f(y)$ (by induction)
- Replace leaf corresponding to z by an internal node with two leaves, one for x and one for y.
- In order to reduce problem from size n to size $n - 1$, we introduce an "artificially made" new character!
- complexity: $O(n \log n)$ time.

Example:	{A,	B,	C,	D,	E,	F}	Characters
	5	2	3	4	10	1	Frequencies

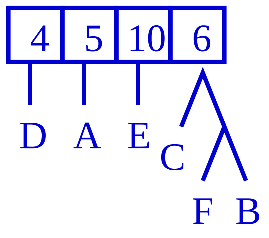
1 heap



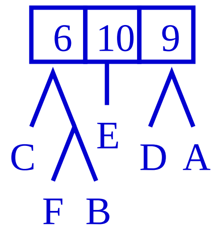
2 delete F, delete B, insert FB(3)



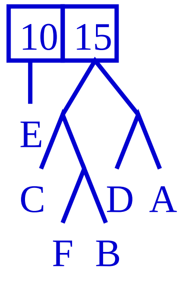
3 delete C, delete FB, insert CFB(6)



4 delete D, delete A, insert DA(9)



5 delete CFB, delete DA, insert CFBDA(15)



6 delete E, delete CFBDA, insert ECFBDA(25)