BITS, PILANI – K. K. BIRLA GOA CAMPUS

Design & Analysis of Algorithms

(CS F364)

Lecture No.13

Coding

Suppose that we have a 100000 character data file that we wish to store. The file contains only 6 characters, appearing with the following frequencies:

	a	b	c	d	e	f
Frequency in thousands	**45**	**13**	**12**	**16**	**9**	**5**

A binary code encodes each character as a binary string or code word.

Goal: We would like to find a binary code that encodes the file using as few bits as possible

Fixed Length coding

In a fixed-length code each code word has the same length.

For our example, a fixed-length code must have at least 3 bits per code word. One possible coding is as follows:

	a	b	c	d	e	f
Frequency in thousands	**45**	**13**	**12**	**16**	**9**	**5**
Fixed length coding	**000**	**001**	**010**	**011**	**100**	**101**

The fixed length code requires 300000 bits to store the file

variable-length code

In a variable-length code, code words may have different lengths.

One possible coding for our example is as follows:

	a	b	c	d	e	f
Frequency in thousands	**45**	**13**	**12**	**16**	**9**	**5**
Variable length coding	**0**	**101**	**100**	**111**	**1101**	**1100**

The variable-length code uses only

(45x1+13x3+12x3+16x3+9x4+5x4)x1000= 224000 bits

Much better than fixed length code

Can we do better?

Key Property - Encoding

Key Property

Given an encoded message, decoding is the process of turning it back into the original message. A message is **uniquely decodable**, if it can only be decoded in one way.

Encoding should be done so that message is uniquely decodable.

Note:

1. **Fixed-length codes are always uniquely decodable**

2. May not be so for variable length encoding

Example : C = {a= 1; b= 110; c= 10; d= 111}

1101111 is not uniquely decipherable since it could have encoded either **bad** or **acad**.

Variable-length codes may not be uniquely decodable

Prefix Codes

Prefix Code: A code is called a prefix code if no code word is a prefix of another one.

Example:

{a= 0; b= 110; c= 10; d= 111} is a prefix code.

Important Fact:

Every message encoded by a prefix code is uniquely decipherable.

Example:

0110100 = **0110100** = **abca**

We are therefore interested in finding good prefix codes.

Representation – Binary Encoding

Representation

Prefix codes (in general any binary encoding) can be represented as a **binary tree.**

Left edge is labeled ***0*** and the right edge is labeled ***1***

Label of leaf is frequency of character.

Path from root to leaf is code word associated with character.

Cost of Tree

- For each character ***c*** in the set C let ***freq.c*** denote the frequency of ***c*** in the file
- Given a tree ***T*** corresponding to a prefix code
- ***d_T(c)*** denote the depth of ***c's*** leaf in the tree
- ***d_T(c)*** is also the length of the codeword for character c
- The number of bits required to encode a file is

$$B(T) = \sum_{c \in C} c.freq.d_T(c)$$

which is defined as **cost** of the tree ***T***

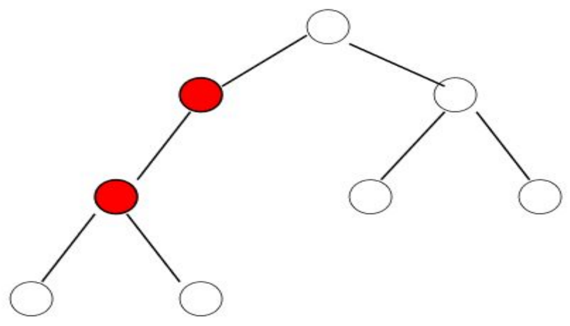
Full Binary Tree

Key Idea:

An **optimal code** for a file is always represented by a **full binary tree**.

Full binary tree is a binary tree in which every non-leaf node has two children

Proof: If some internal node had only one child then we could simply get rid of this node and replace it with its unique child. This would decrease the total cost of the encoding.



Greedy Choice Property

Lemma:

Consider the two letters, x and y with the smallest frequencies. Then there exists an optimal code tree in which these two letters are sibling leaves in the tree at the lowest level

Proof (Idea)

Take a tree T representing arbitrary optimal prefix code and modify it to make a tree representing another prefix code such that the resulting tree has the required greedy property.

Greedy Choice Property

Let T be an optimum prefix code tree, and let b and c be two siblings at the maximum depth of the tree (must exist because T is full).

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

Since x and y have the two smallest frequencies it follows that $f(x) \le f(b)$ and $f(y) \le f(c)$

Since b & c are at the deepest level of the tree $d(b) \ge d(x)$ and $d(c) \ge d(y)$

Now switch the positions of x and b in the tree resulting in new tree T'

Greedy Choice Property

Since T is optimum

$$\begin{aligned} B(T) &\le B(T') \\ &= B(T) - f(x)d(x) - f(b)d(b) + f(x)d(b) + f(b)d(x) \\ &= B(T) + (f(x) - f(b))(d(b) - d(x)) \\ &\le B(T). \end{aligned}$$

Therefore, $B(T') = B(T)$, that is, T' is an optimum tree. By switching y with c we get a new tree T'' which by a similar argument is optimum. The final tree T'' satisfies the statement of the claim.

Optimal Substructure Property

Optimal Substructure Property

Lemma:

Let T be a full binary tree representing an optimal prefix code over an alphabet C, where frequency f[c] is define for each character c belongs to set C. Consider any two characters x and y that appear as sibling leaves in the tree T and let z be their parent. Then, considering character z with frequency f[z] = f[x] + f[y], tree T` = T - {x, y} represents an optimal code for the alphabet C` = C - {x, y}U{z}.

Proof : See CLRS

Huffman Coding

Step1:

Pick two letters ***x; y*** from alphabet C with the smallest frequencies and create a subtree that has these two characters as leaves.

Label the root of this subtree as ***z***

Step2:

Set frequency **f(z) = f(x) +f(y).**

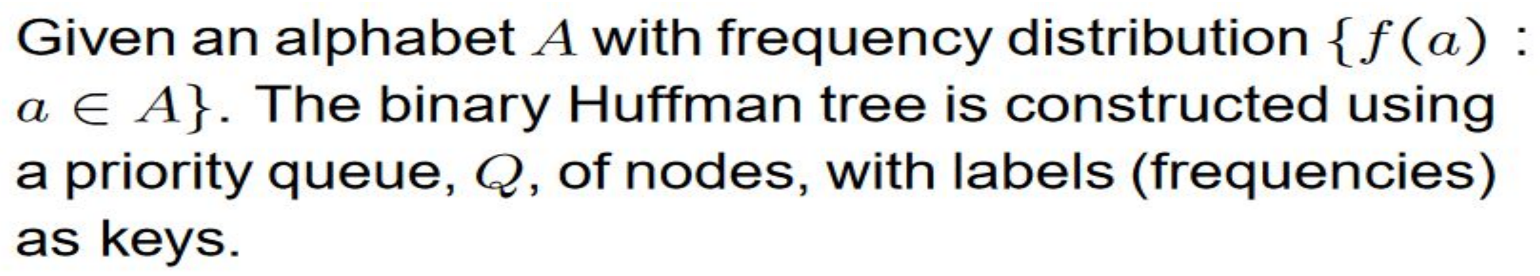
Remove ***x; y*** and add ***z*** creating new alphabet

$A^* = A \cup \{z\} - \{x, y\}$, Note that $|A^*| = |A| - 1$

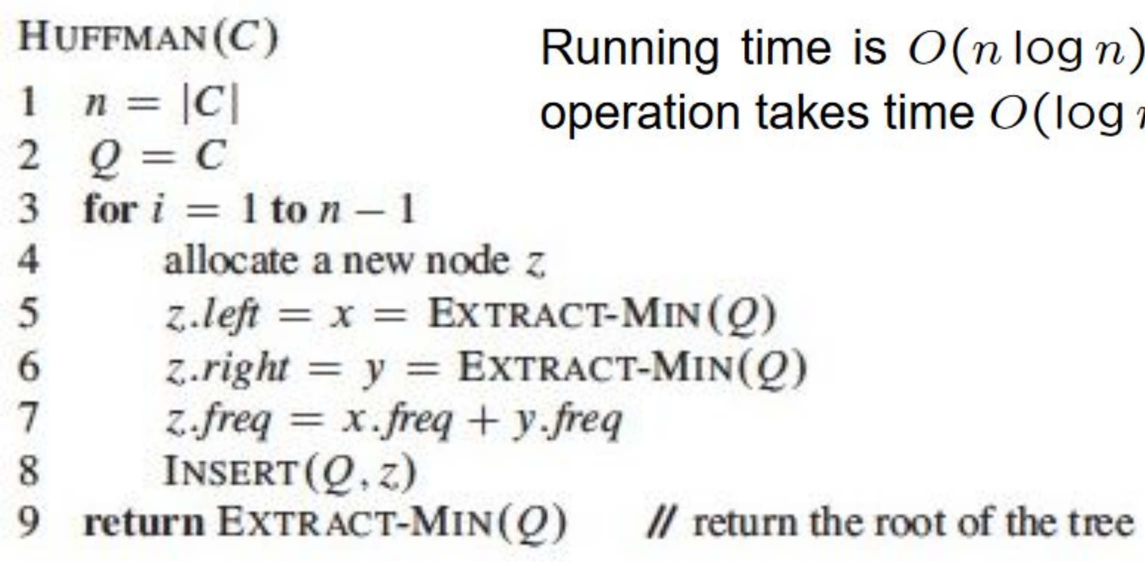
Repeat this procedure, called **merge,** with new alpha-bet ***A**** until an alphabet with only one symbol is left.

Huffman Code Algorithm

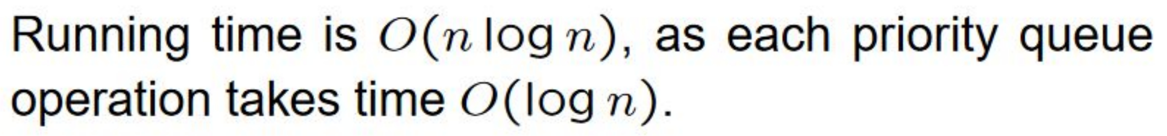
Huffman Code Algorithm



Given an alphabet A with frequency distribution $\{f(a) : a \in A\}$. The binary Huffman tree is constructed using a priority queue, Q, of nodes, with labels (frequencies) as keys.



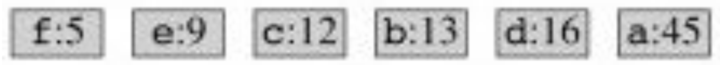
```
HUFFMAN(C)
1  $n = |C|$
2  $Q = C$
3  for $i = 1$ to $n - 1$
4      allocate a new node $z$
5      $z.left = x =$ EXTRACT-MIN($Q$)
6      $z.right = y =$ EXTRACT-MIN($Q$)
7      $z.freq = x.freq + y.freq$
8      INSERT($Q, z$)
9  return EXTRACT-MIN($Q$)      // return the root of the tree
```



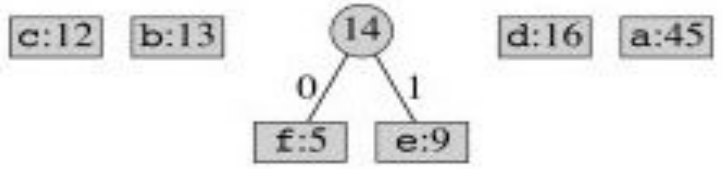
Running time is $O(n \log n)$, as each priority queue operation takes time $O(\log n)$.

Huffman Code - Example

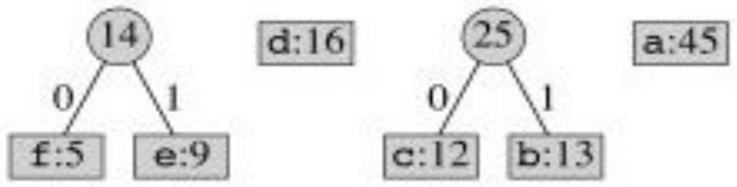
Example:



Take out the two lowest frequency items and make a subtree that is put back on the queue as if it is a combined character:

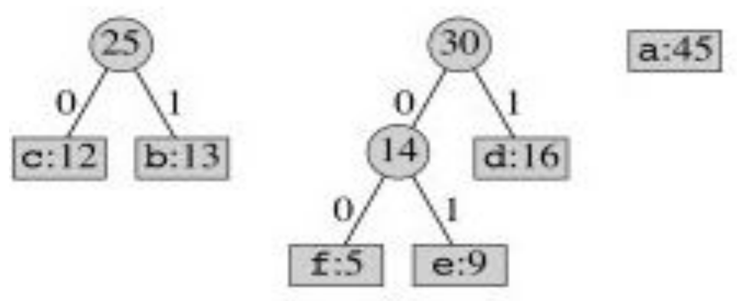


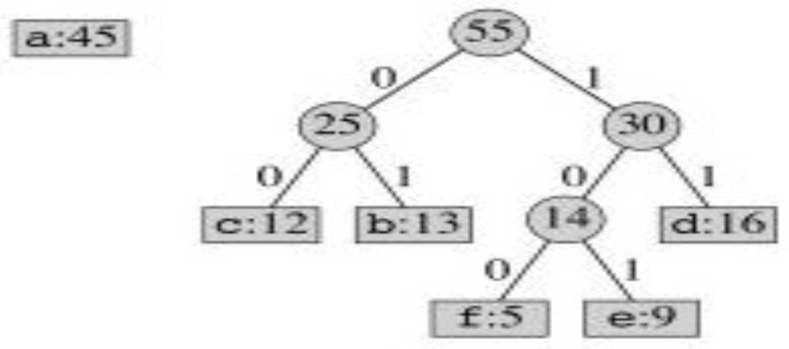
Combine the next lowest frequency characters:



Huffman Code - Example

Example Continued

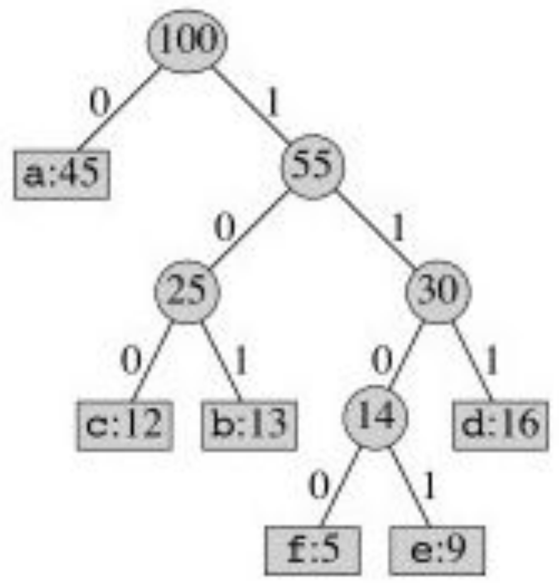




Huffman Code - Example

Example Continued

The highest frequency character gets added to the tree last, so it will have a code of length 1:


100
0
1
a:45
55
0
1
25
30
0
1
0
1
c:12
b:13
14
d:16
0
1
f:5
e:9