Huffman Coding

Hashmat Rohian
CSE 6111

Motivation

To compress or not to compress, that is the question!

- ➢ reducing the space required to store files on disk or tape

- ➢ reducing the time to transmit large files.

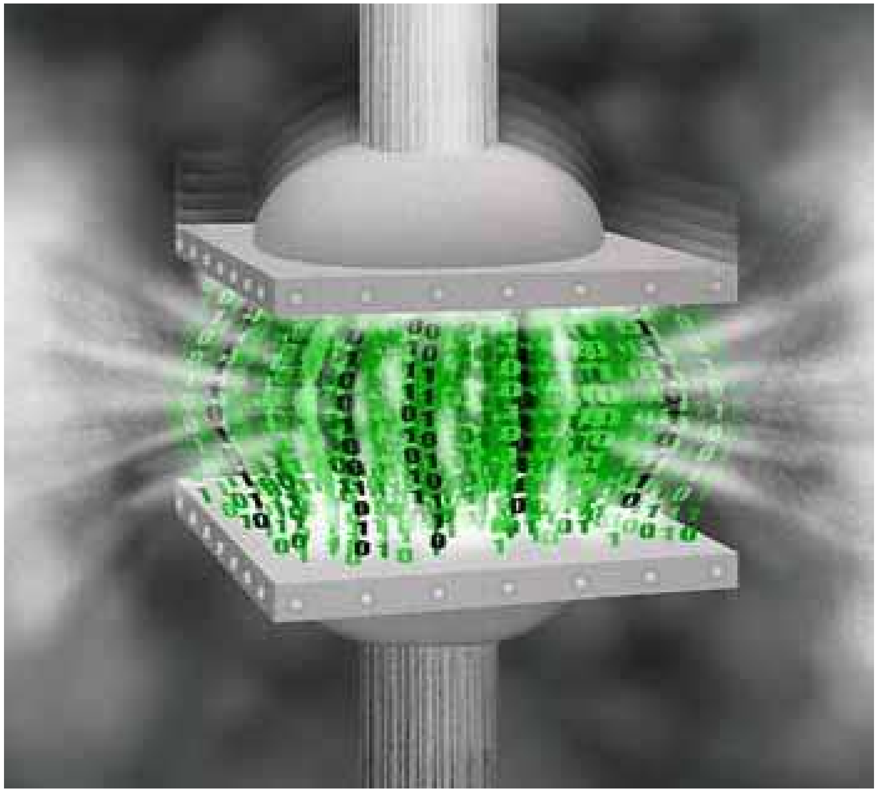


Image Source : plus.maths.org/issue23/ features/data/data.jpg

Example:

- A file with 100K characters

Character	a	b	c	d	e	f
Frequency (in 1000s)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101

Space = (45*3 + 13*3 + 12*3 + 16*3 + 9*3 + 5*3) * 1000

= **300K bits**

Can we do better ??

YES !!

- Use **variable-length** codes instead.
- Give frequent characters short codewords, and infrequent characters long codewords.

	a	b	c	d	e	f
Frequency (in 1000s)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

Space = (45*1 + 13*3 + 12*3 + 16*3 + 9*4 + 5*4) * 1000

= **224K bits** (Savings = 25%)

PREFIX-FREE CODE :

- No codeword is also prefix of some other codeword.

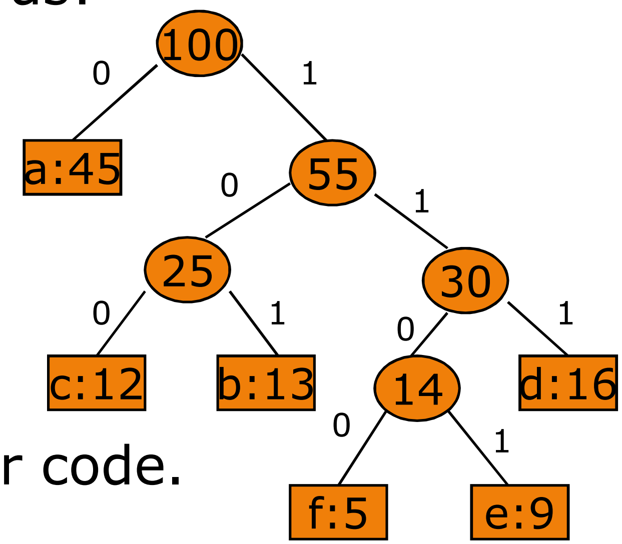
No Ambiguity !!

Variable-length codeword	0	101	100	111	1101	1100

Representation:

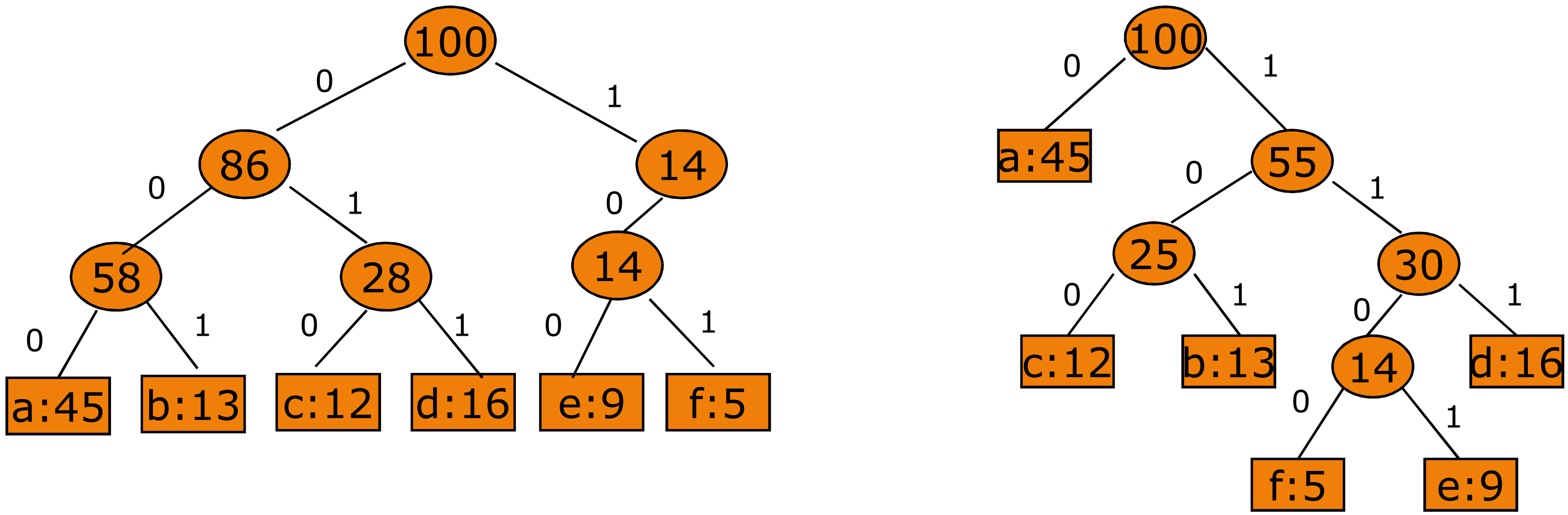
The Huffman algorithm is represented as:

- binary tree
- each edge represents either
  - 0, "go to the left child"
  - 1, "go to the right child"
- each leaf corresponds a particular code.



- Cost of the tree
  - B(T) = $\sum$**f(c) d$_T$(c) where c ε C**

Optimal Code



- Always a ***full*** **binary tree**
  - *One leaf for each letter of the alphabet*

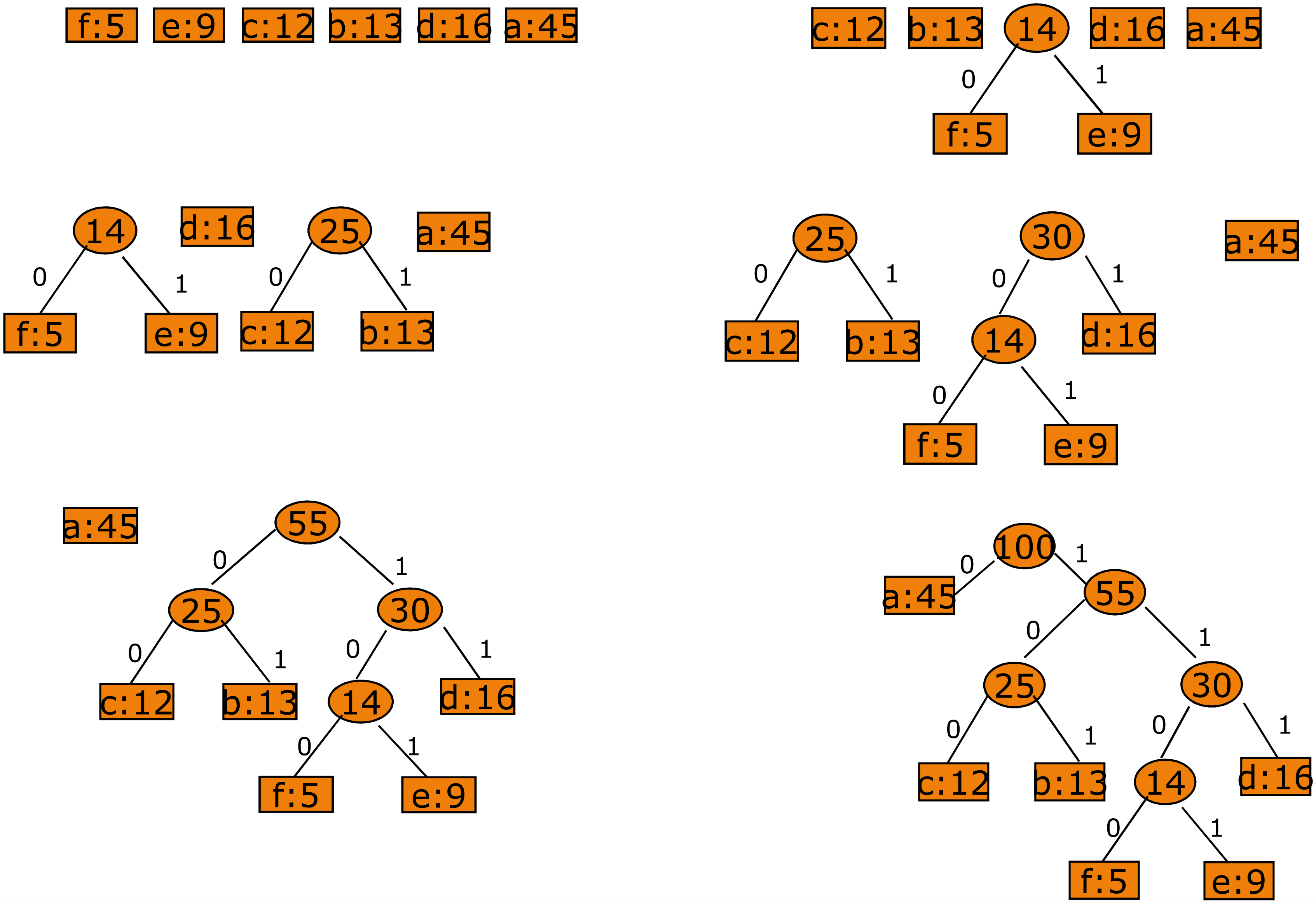
Constructing a Huffman code

- B*uild the tree T in a* *bottom-up* *manner.*
  - *Begins with a set of |C| leaves*
  - *Upward |C| - 1 "merging" operations*

Greedy Choice?

- *The two smallest nodes are chosen at each step.*

The steps of Huffman's algorithm



Running Time Analysis

Q is implemented as a binary min-heap.

The *merge operation* is executed exactly $|n|$ - 1 times. Each heap operation requires time $O(\log n)$.

$$= O(n \log n)$$

- Huffman code

E	=	01
I	=	00
C	=	10
A	=	111
H	=	110

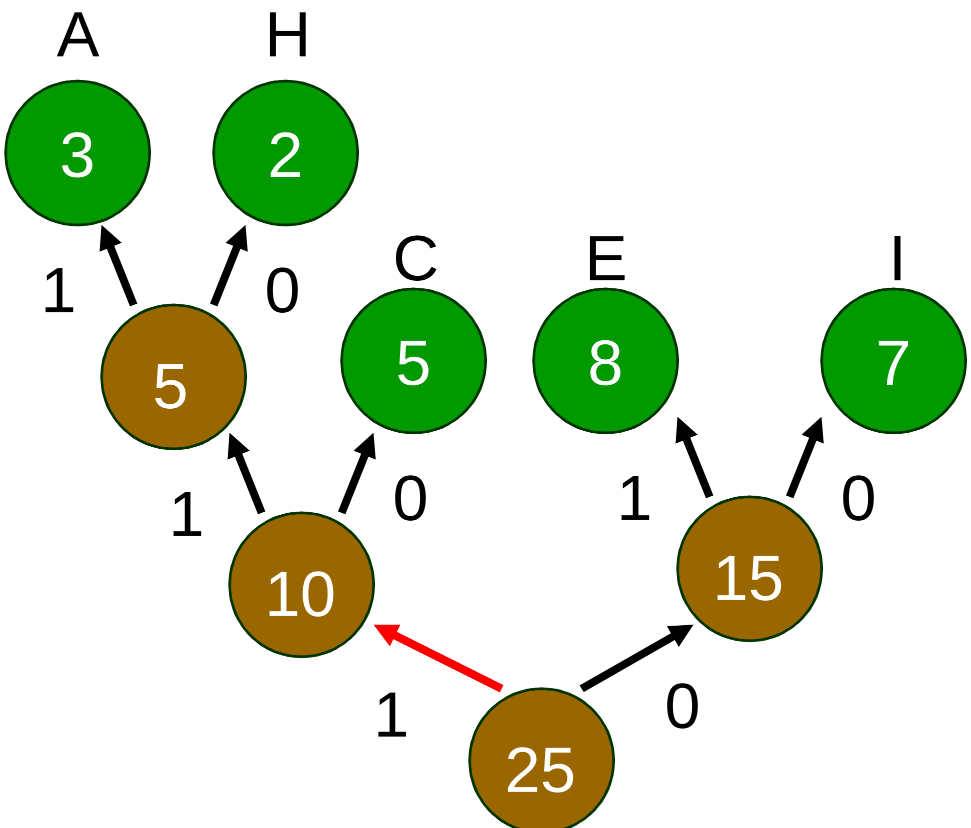
- Input
  - ACE
- Output
  - (111)(10)(01) = 1111001

Huffman Coding Example

- Decoding
  1. Read compressed file & binary tree
  2. Use binary tree to decode file
     Follow path from root to leaf

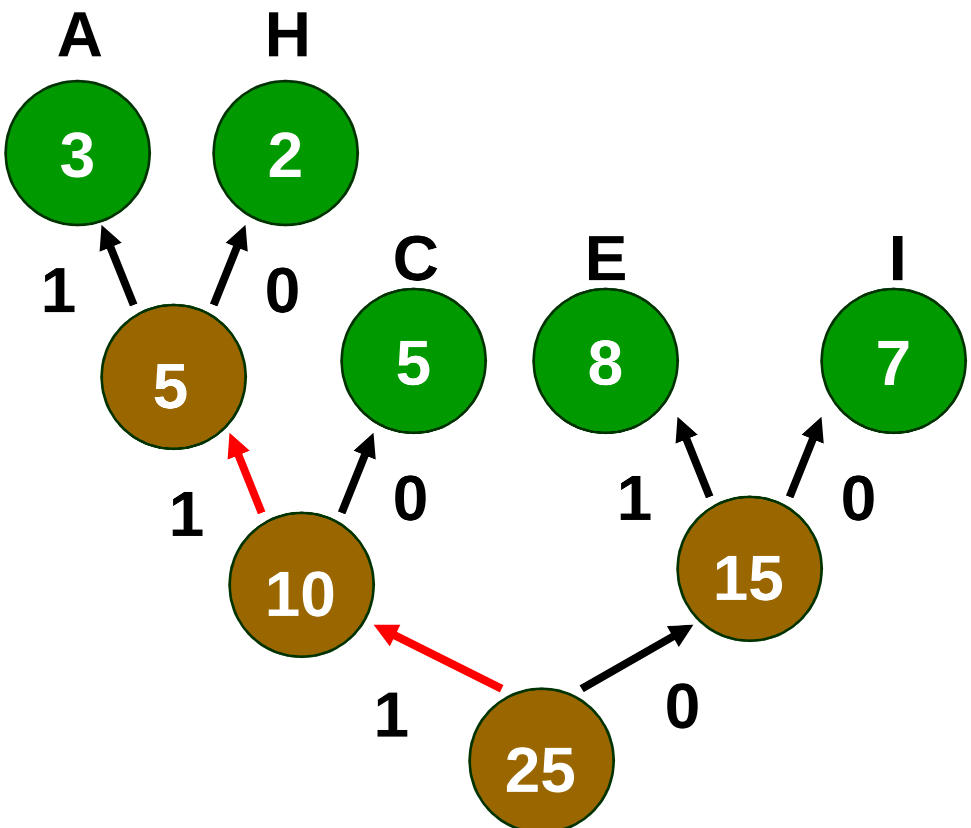
Huffman Code Algorithm Overview

Huffman Decoding 1



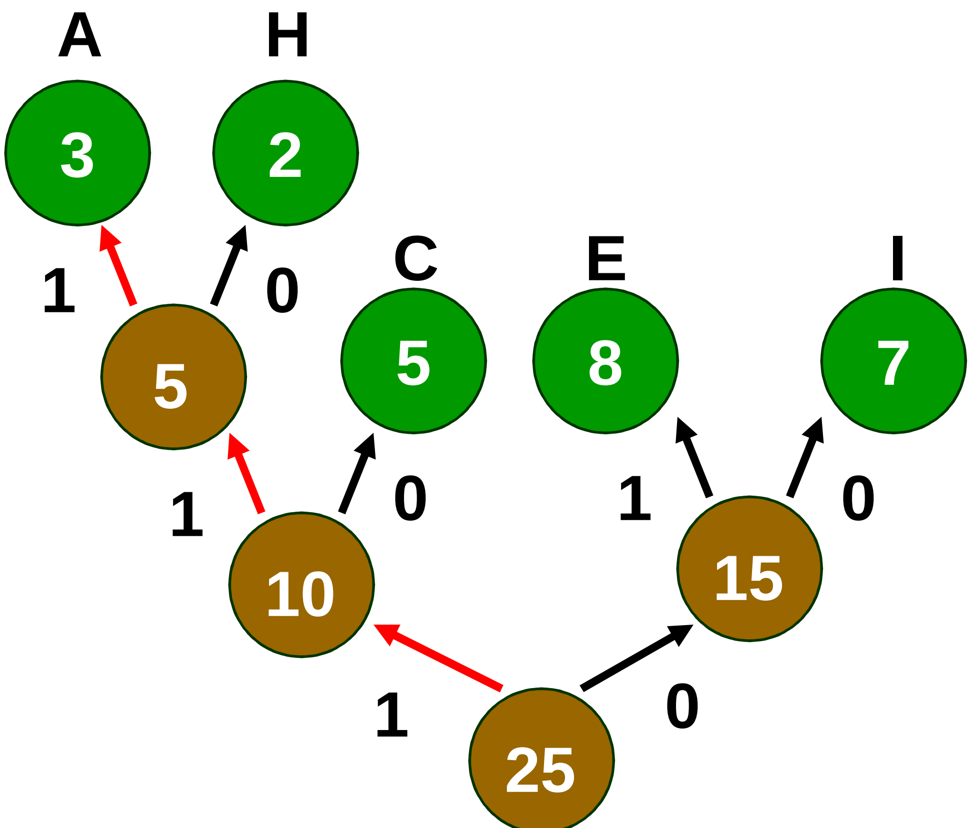
1111001

Huffman Decoding 2



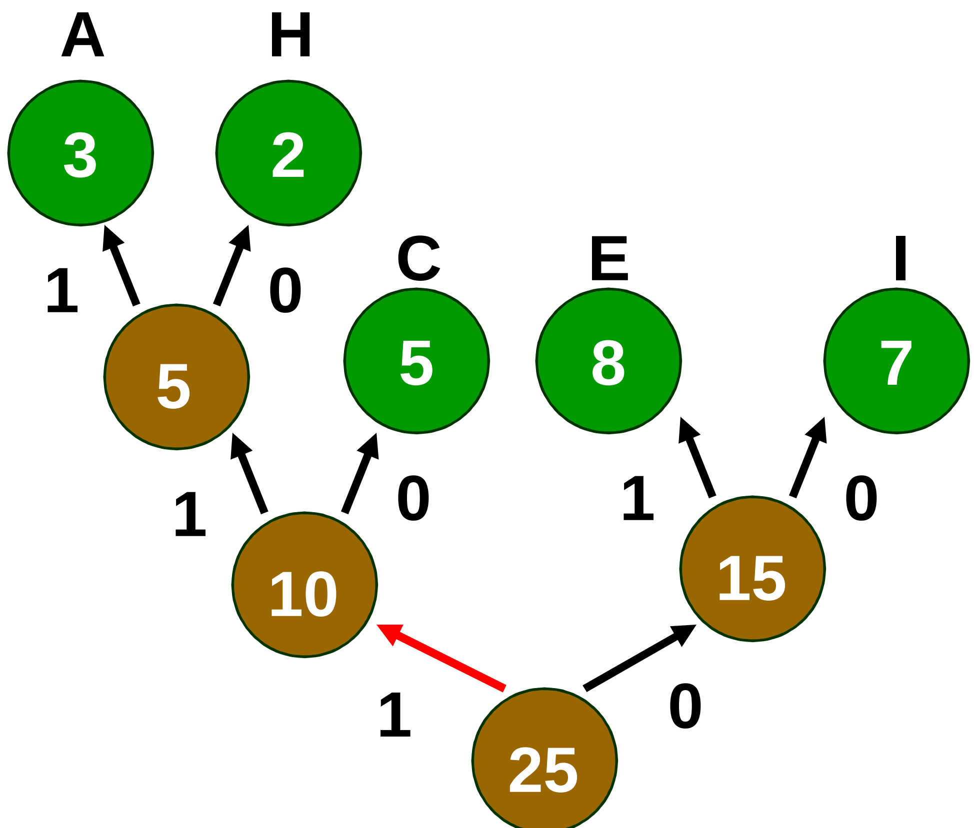
1111001

Huffman Decoding 3



1111001

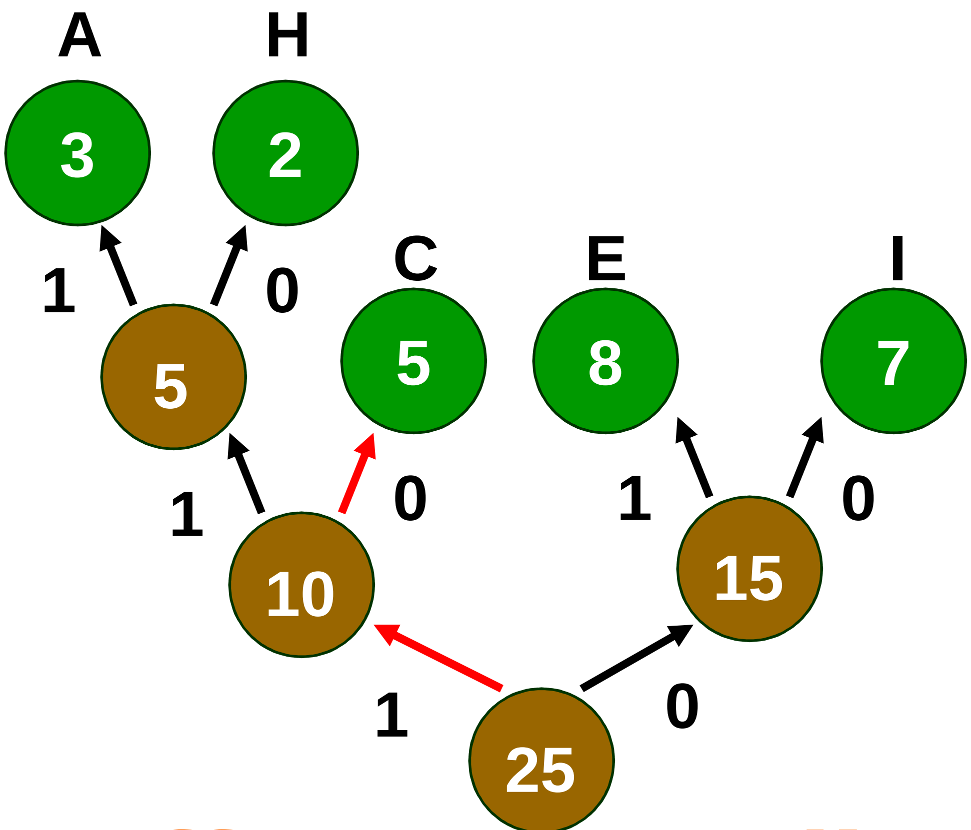
A



1111001

A

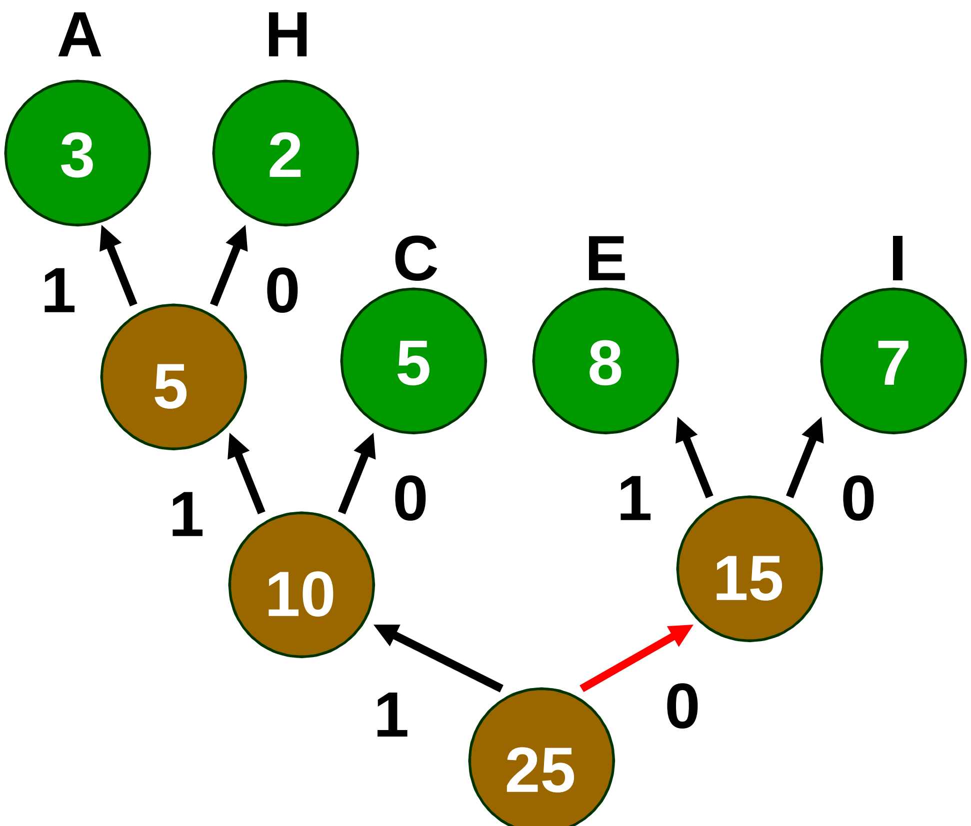
Huffman Decoding 4



1111001

AC

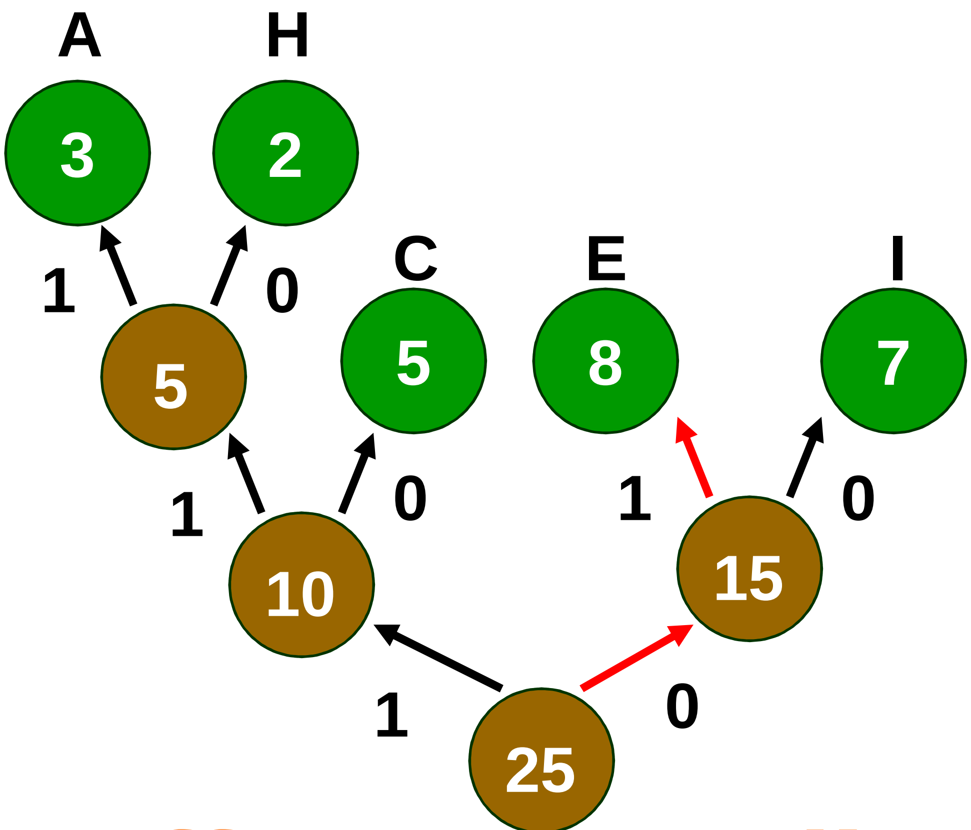
Huffman Decoding 5



1111001

AC

Huffman Decoding 6



1111001

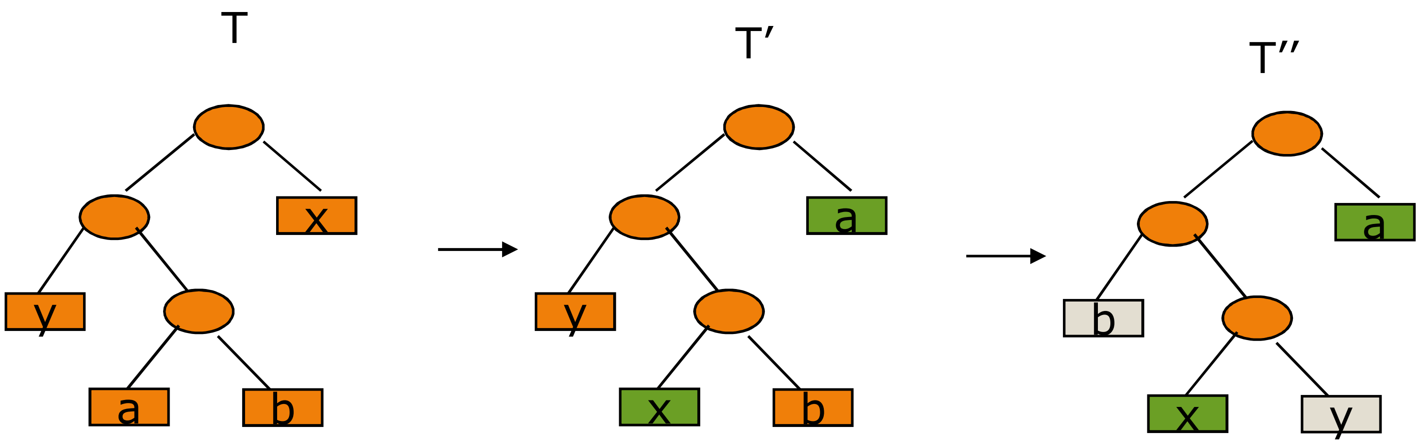
ACE

Huffman Decoding 7

- Induction on the number of code words
- The Huffman algorithm finds an optimal code for $n = 1$
- Suppose that the Huffman algorithm finds an optimal code for codes size n, now consider a code of size $n + 1$ . . .

Correctness proof

Greedy Choice Proof



Assume that f[a] < f[b] and f[x] < f[y]

Since f[x] and f[y] are the two lowest frequencies,

f[x] < f[a] and f[y] < f[b].

- T – Tree constructed by Huffman
- X – Any code tree

- Show C(T) <= C(X)

- T′ and X′ – Trees from the greedy choice

- C(T′) = C(T)
- C(X′) <= C(X)

- T′′ and X′′ – Trees with minimum cost leaves x and y removed

Finish the induction proof

- $C(X'') = C(X') - x - y$
- $C(T'') = C(T') - x - y$
- $C(T'') <= C(X'')$

- $C(T) = C(T')$
- $= C(T'') + x + y$
- $<= C(X'') + x + y$
- $= C(X')$
- $<= C(X)$

X : Any tree, X': – modified,
X'' : Two smallest leaves removed


What is
our
next
step?

Challenges and how to tackle them?

Two passes over the data:

- One pass to collect frequency counts of the letters
- A second pass to encode and transmit the letters, based on the static tree structure.

Problems:

Delay (network communication, file compression applications)

Extra disk accesses slow down the algorithm.

We need one-pass methods, in which letters are encoded "***on the fly*".**

Dynamic Huffman codes

Algorithm FGK

- The next letter of the message is encoded on the basis of a Huffman tree for the previous letters.
- Encoding length = ***(2S + t),*** where S is the encoding length by a static Huffman code, and t is the number of letters in the original message.

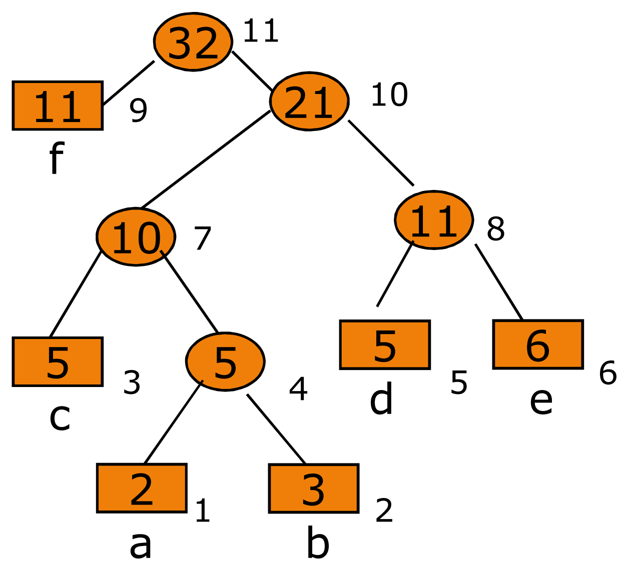
Sender and receiver

- start with the same initial tree
- use the same algorithm to modify the tree after each letter is processed and thus always have equivalent copies of it.

Sibling Property:

A binary tree with p leaves of nonnegative weight is a Huffman tree iff

- the p leaves have nonnegative weights $w_1, \ldots, w_p$, and the weight of each internal node is the sum of the weights of its children; and
- the nodes can be numbered in non-decreasing order by weight, so that nodes $(2j - 1)$and $2j$ are siblings, for $1 \leq j \leq p - 1$, and their common parent node is higher in the numbering.



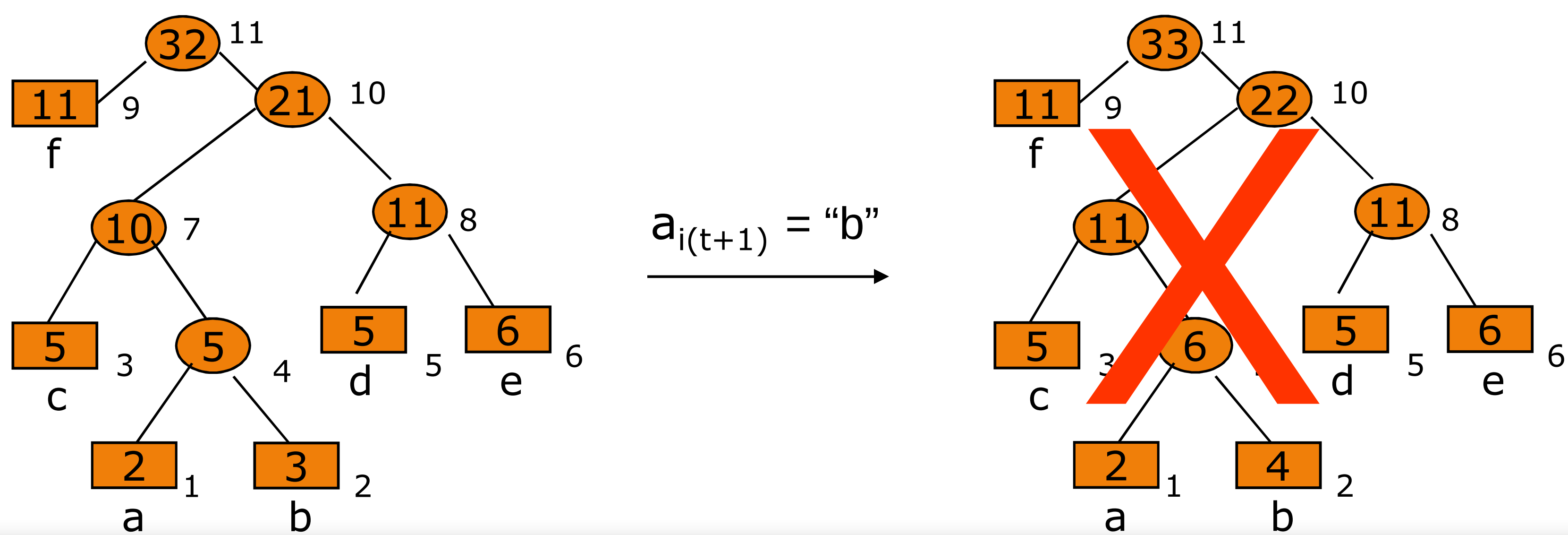
Difficulty

Suppose that $\mathcal{M}_T = a_{i1}, a_{i2}, \ldots, a_{it}$, has already been processed.

$a_{i(t+1)}$ is encoded and decoded using Huffman tree for $\mathcal{M}_T$.

How to modify this tree quickly in order to get a Huffman tree for $\mathcal{M}_{T+1}$?

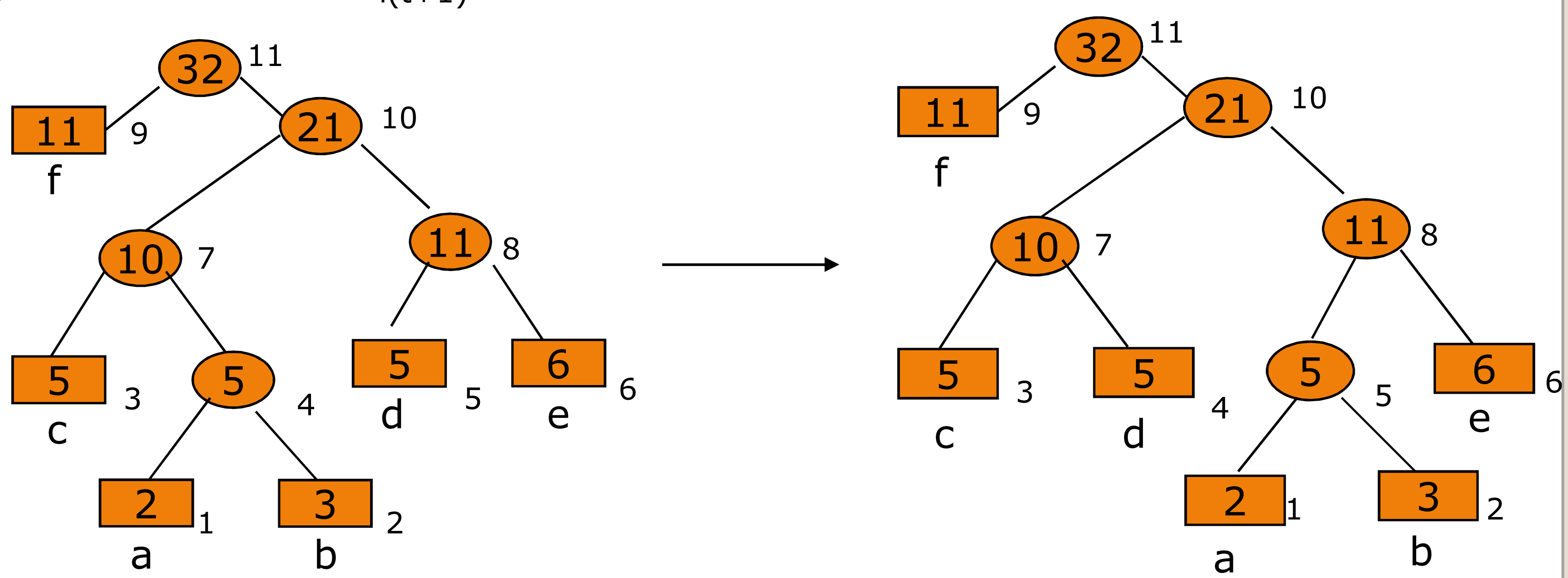
Eg. Assume t = 32,

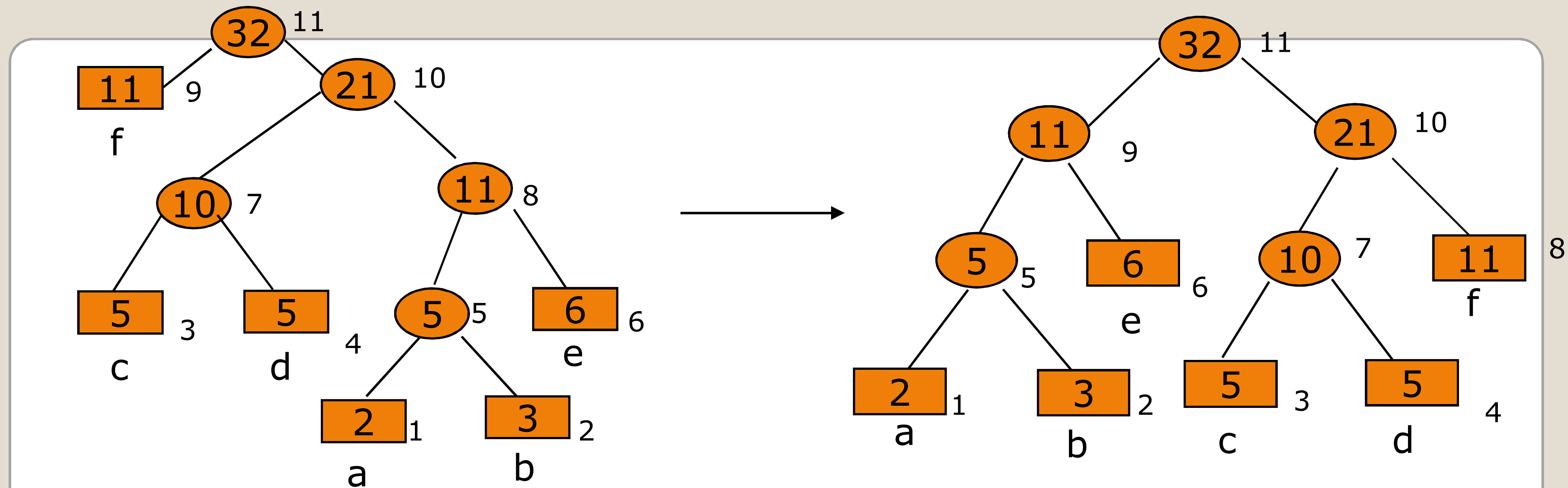


First phase

- Begin with the leaf of $a_{i(t+1)}$, as the current node.
- Repeatedly interchange the contents of the current node, including the subtree rooted there, with that of the highest numbered node of the same weight
- Make the parent of the latter node the new current node.
- Halt when the root is reached.

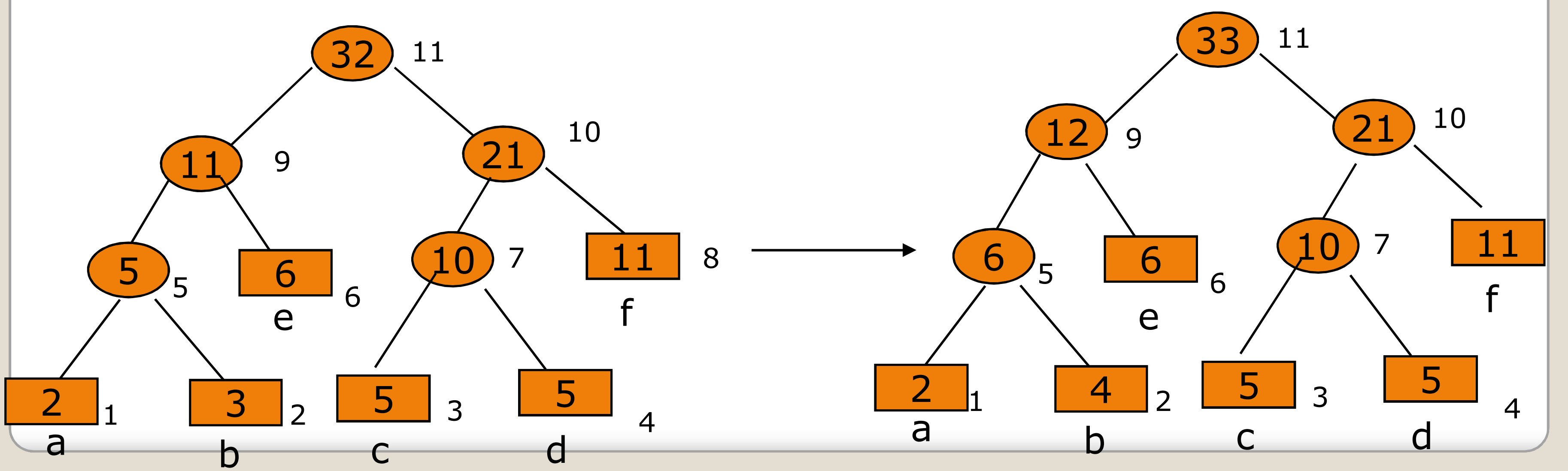
Eg. Assume t = 32, $a_{i(t+1)}$ = “b”





Second phase

• We turn this tree into the desired Huffman tree for $\mathcal{M}_{T+1}$ by incrementing the weights of $a_{i(t+1)}$'s leaf and its ancestors by 1



- Huffman savings are between 20% - 90%
- Dynamic Huffman Coding optimal and efficient
- Optimal data compression achievable by a character code can <u>always be achieved</u> with a prefix code.
- Better compression possible (depends on data)
  - Using other approaches (e.g., pattern dictionary)

Conclusions

Thank you for your attention!