

# **CSE 421**

## **Greedy: Huffman Codes**

Yin Tat Lee

# Compression Example

100k file, 6 letter alphabet:

File Size:

ASCII, 8 bits/char: 800kbits

$2^3 > 6$ ; 3 bits/char: 300kbits

a	45%
b	13%
c	12%
d	16%
e	9%
f	5%

Why?

Storage, transmission

# Compression Example

a	45%
b	13%
c	12%
d	16%
e	9%
f	5%

- 100k file, 6 letter alphabet:

- File Size:

ASCII, 8 bits/char: 800kbits

$2^3 > 6$ ; 3 bits/char: 300kbits

better:  $\longrightarrow$

2.52 bits/char  $74\%*2 + 26\%*4$ : 252kbits

Optimal?

E.g.:	Why not:
a 00	00
b 01	01
d 10	10
c 1100	110
e 1101	1101
f 1110	1110

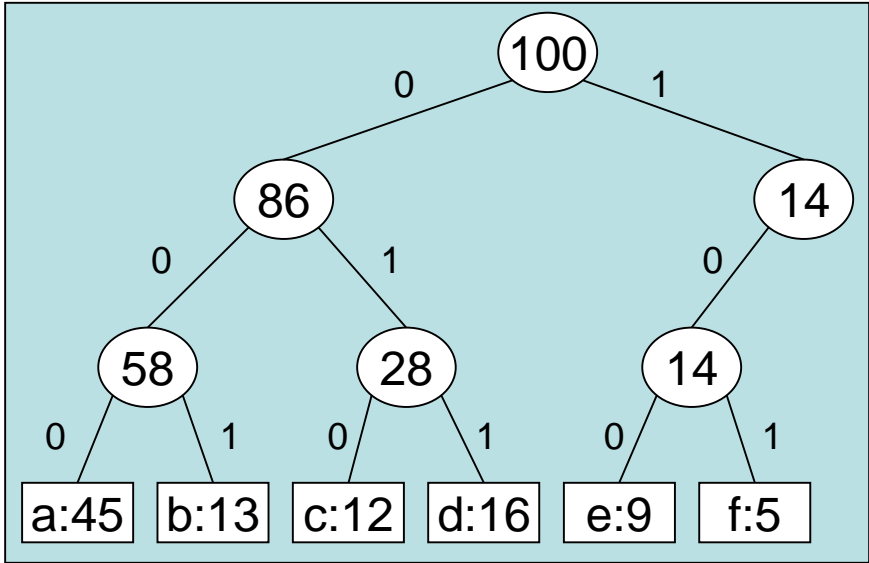
1101110 = cf or ec?

# Data Compression

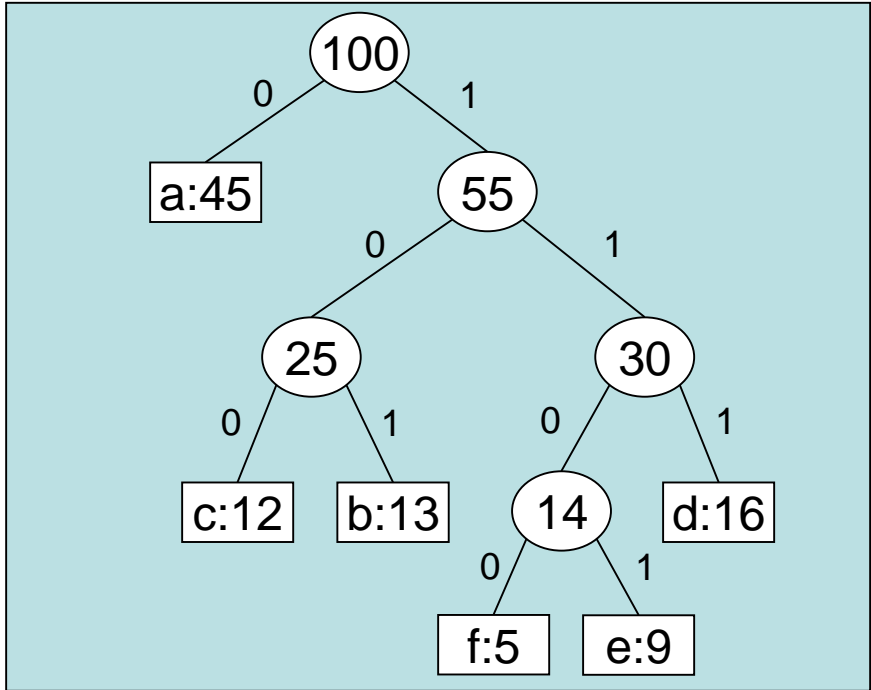
- Binary character code (“code”)
  - each  $k$ -bit *source string* maps to unique *code word*  
(e.g.  $k=8$ )
  - “compression” alg: concatenate code words for successive  $k$ -bit “characters” of source
- Fixed/variable length codes
  - all code words equal length?
- Prefix codes
  - no code word is prefix of another (unique decoding)

# Prefix Codes = Trees

a	45%
b	13%
c	12%
d	16%
e	9%
f	5%



1 0 1 0 0 0 0 0 1  
 └──┬──┬──┬──┬──┬──┬──┬──┬──  
 f a b

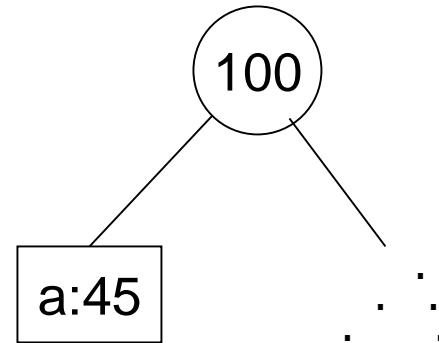


1 1 0 0 0 1 0 1  
 └──┬──┬──┬──┬──┬──┬──┬──┬──  
 f a b

# Greedy Idea #1

a	45%
b	13%
c	12%
d	16%
e	9%
f	5%

Put most frequent  
under root, then recurse ...



# Greedy Idea #1

a	45%
b	13%
c	12%
d	16%
e	9%
f	5%

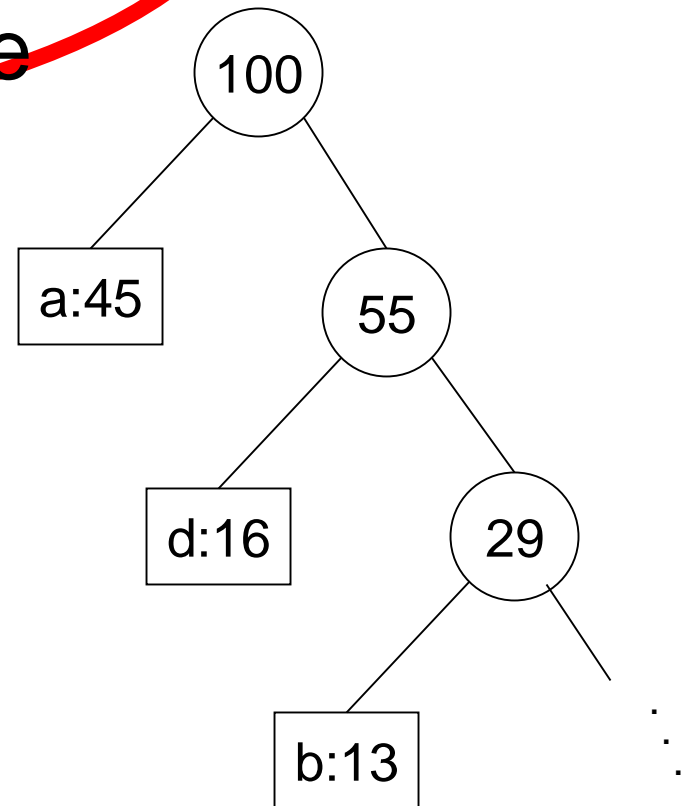
- Top down: Put *most* frequent under root, then recurse

- **Too greedy:  
unbalanced tree**

$$.45 \cdot 1 + .16 \cdot 2 + .13 \cdot 3 \dots = 2.34$$

not too bad, but imagine if all freqs were  $\sim 1/6$ :

$$(1+2+3+4+5+5)/6=3.33$$



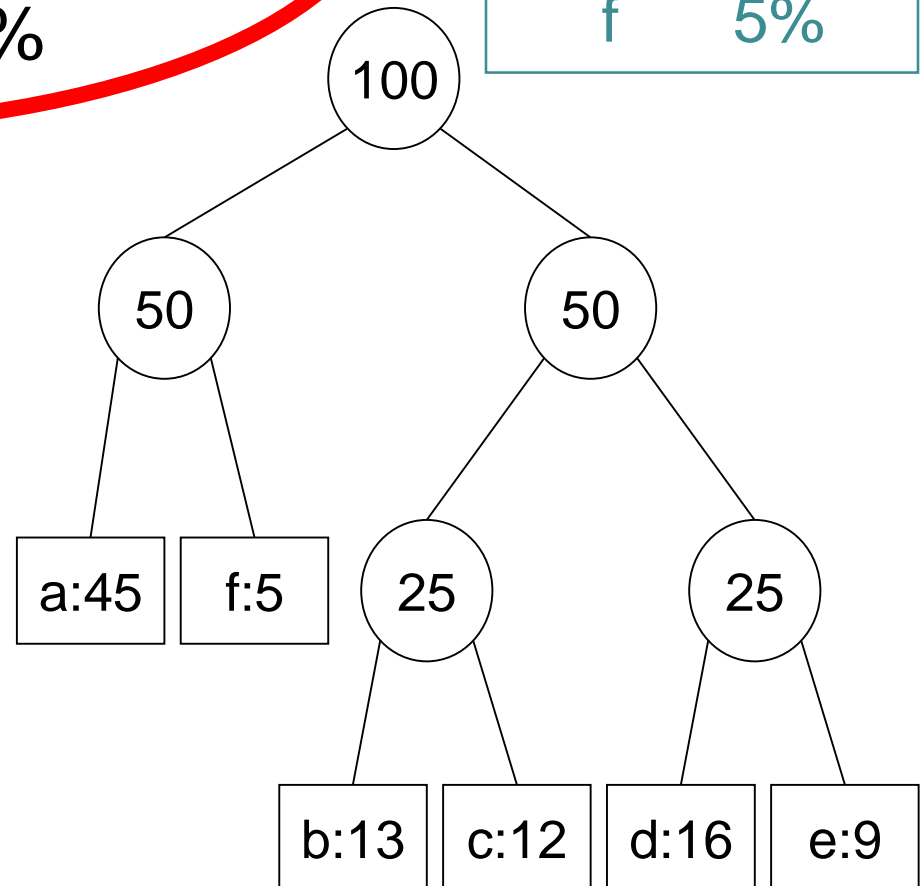
## Greedy Idea #2

a	45%
b	13%
c	12%
d	16%
e	9%
f	5%

- Top down: Divide letters into 2 groups, with ~50% weight in each; recurse (Shannon-Fano code)

- Again, not terrible  
 $2 \cdot .5 + 3 \cdot .5 = 2.5$

- But this tree can easily be improved! (How?)

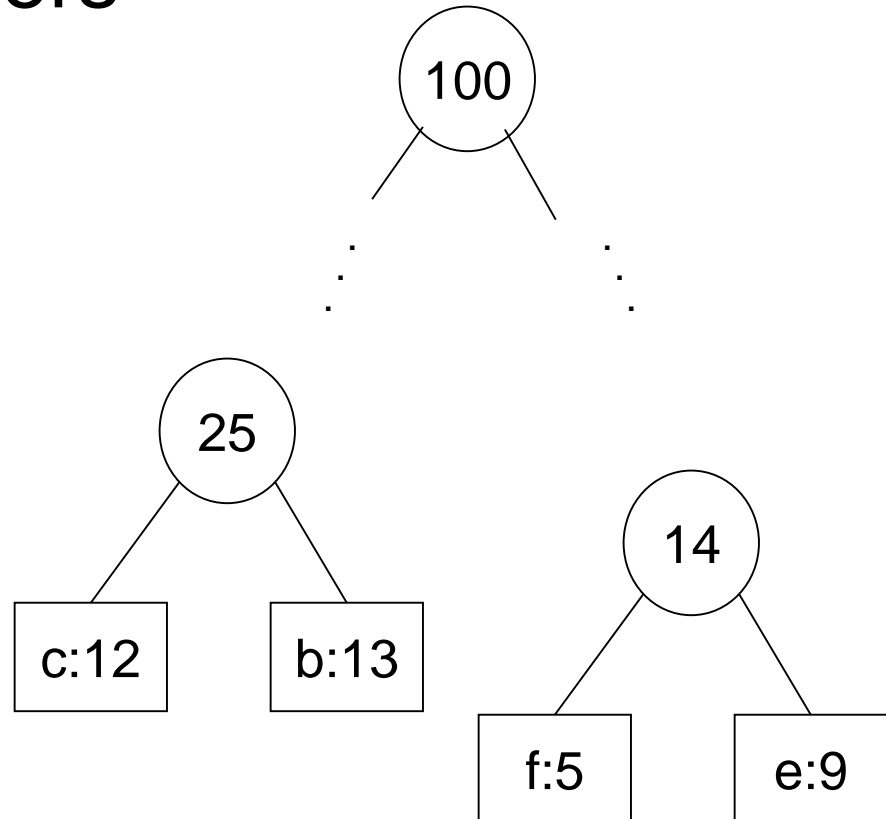


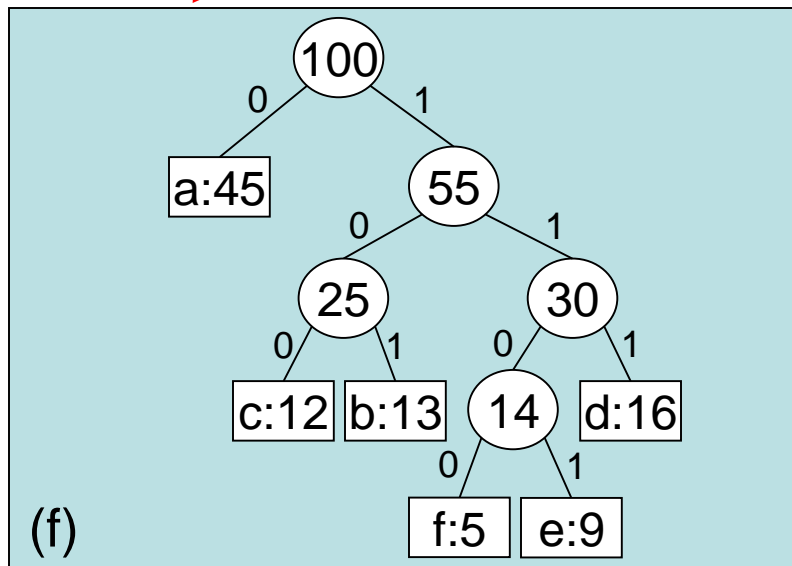
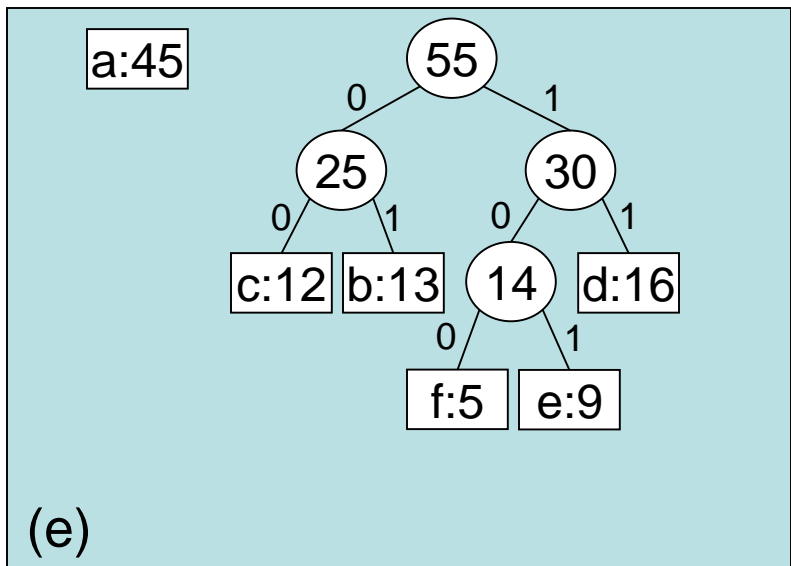
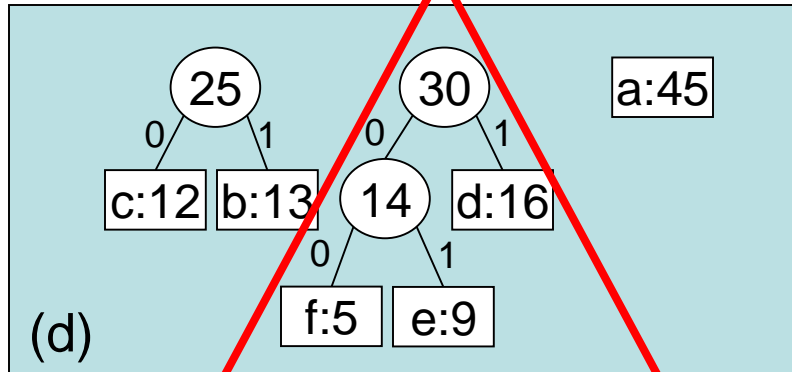
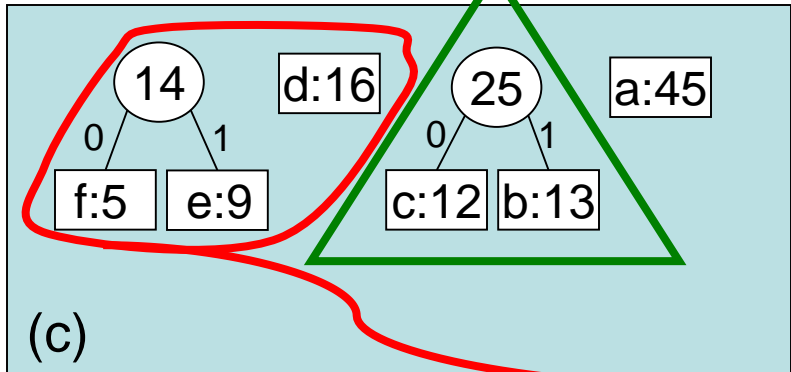
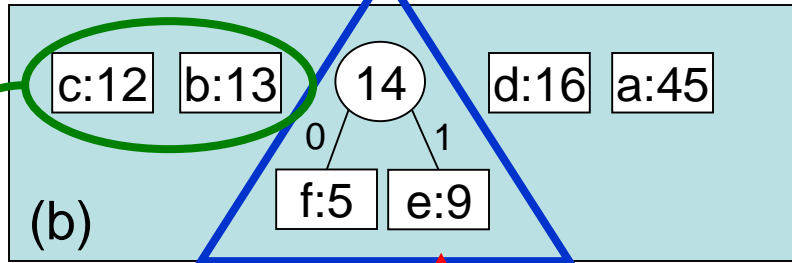
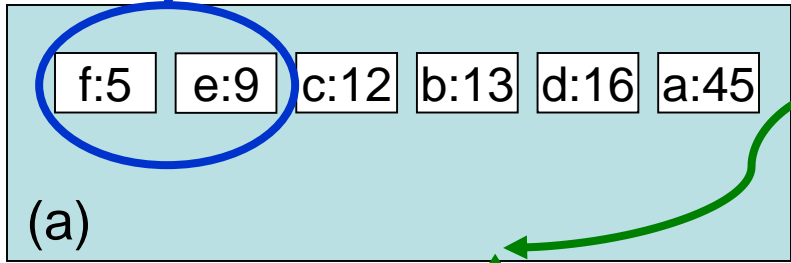


# Greedy idea #3

a	45%
b	13%
c	12%
d	16%
e	9%
f	5%

- Bottom up: Group *least* frequent letters near bottom





$.45 \cdot 1 + .41 \cdot 3 + .14 \cdot 4 = 2.24$  bits per char

# Huffman's Algorithm (1952)

- Algorithm:

insert each letter into priority queue by freq

while queue length > 1 do

    remove smallest 2; call them x, y

    make new node z from them, with  $f(z) = f(x) + f(y)$

    insert z into queue

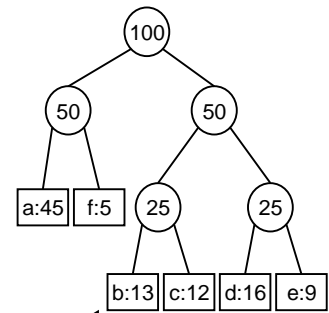
- Analysis:  $O(n \log n)$

- Goal: Minimize  $cost(T) = \sum_c freq(c) \cdot depth(c)$

T = Tree  
C = alphabet  
(leaves)

- Correctness: ???

# Correctness Strategy



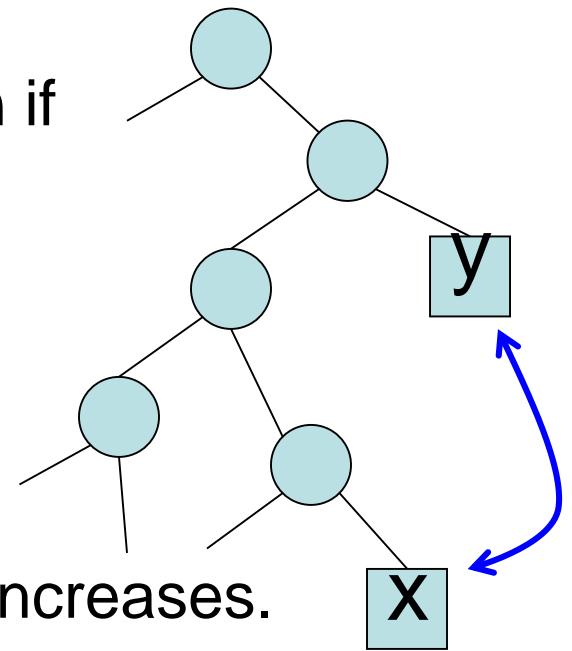
- Optimal solution may not be **unique**, so cannot prove that greedy gives the *only* possible answer.
- Instead, show greedy's solution is **as good as any**.
- How: an exchange argument
- Identify inversions: node-pairs whose swap improves tree

**Defn:** A pair of leaves  $x, y$  is an **inversion** if

$$\text{depth}(x) \geq \text{depth}(y)$$

and

$$\text{freq}(x) \geq \text{freq}(y)$$



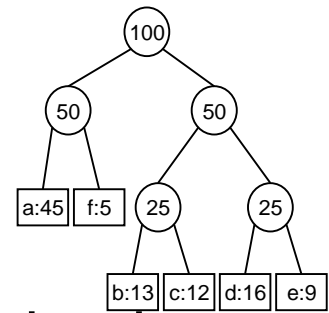
**Claim:** If we **flip** an inversion, cost never increases.

Why? All other things being equal, better to give more frequent letter the shorter code.

$$\begin{aligned} & \underbrace{(d(x)f(x) + d(y)f(y))}_{\text{before}} - \underbrace{(d(x)f(y) + d(y)f(x))}_{\text{after}} \\ &= (d(x) - d(y))(f(x) - f(y)) \geq 0 \end{aligned}$$

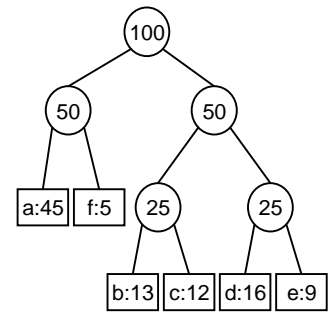
I.e., non-negative cost savings.

# General Inversions



- Define the frequency of an **internal** node to be the sum of the frequencies of the leaves in that subtree.
- We can generalize
  - the defn of inversion for any pair of nodes.
  - the associated claim still holds:
    - exchanging an inverted pair of nodes (& associated subtrees) cannot raise the cost of a tree.
- Proof: Same.

# Correctness Strategy



## Lemma:

Any prefix code tree  $T$  can be converted to a Huffman tree  $H$  via inversion-exchanges

## Corollary:

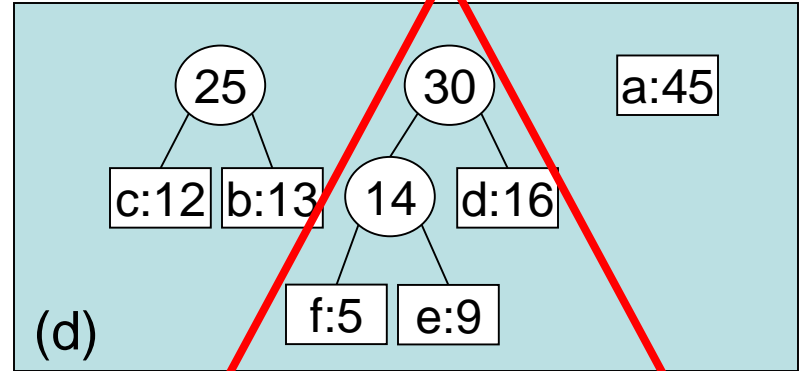
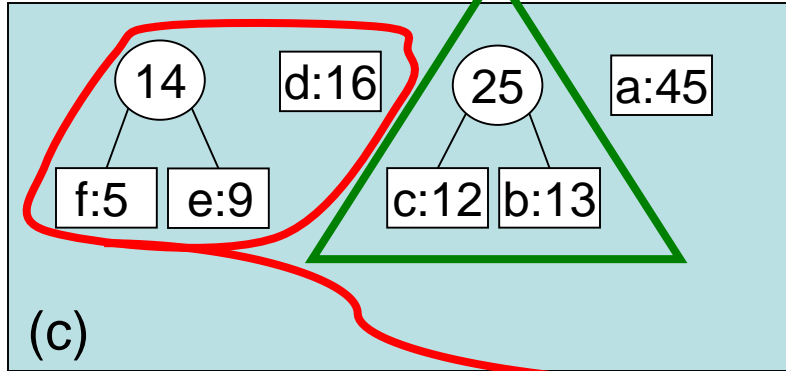
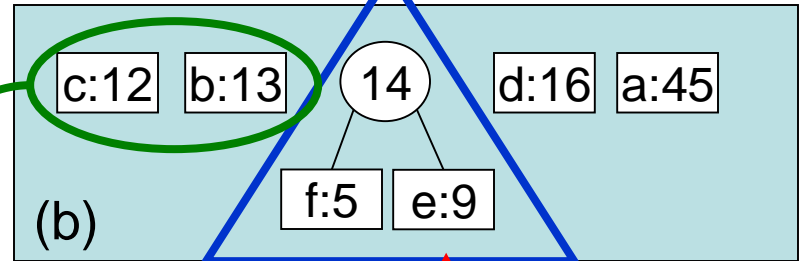
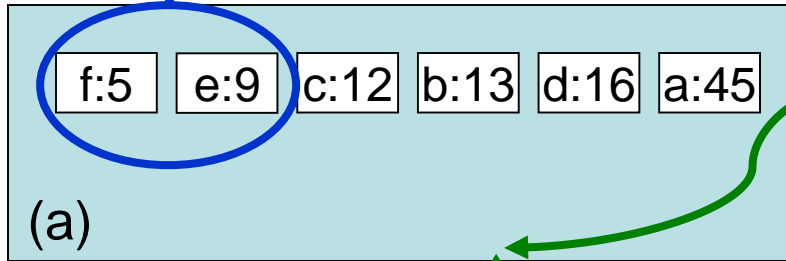
Huffman tree is optimal.

## Proof:

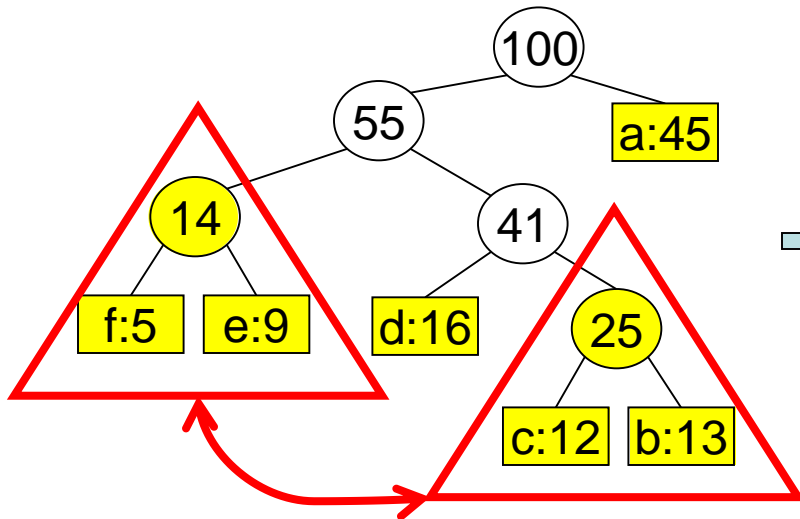
Apply the above lemma to any optimal tree  $T = T_1$ . The lemma only exchanges inversions, which never increase cost.

So,  $cost(T_1) \geq cost(T_2) \geq cost(T_4) \geq \dots \geq cost(H)$ .

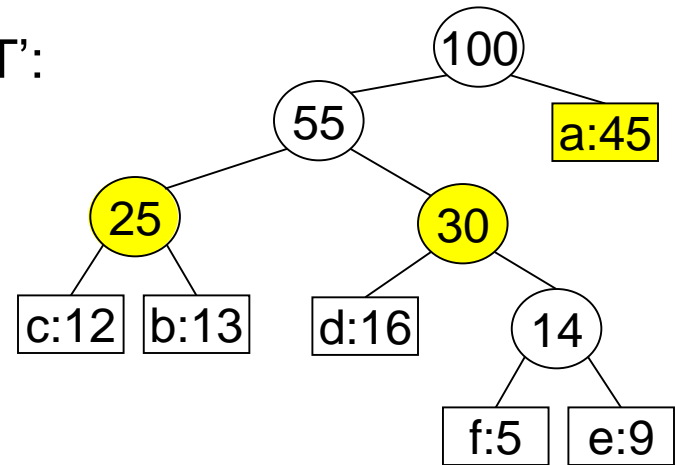
H:



T:



T':





# Lemma: prefix $T \rightarrow$ Huffman $H$ via inversion

**Induction Hypothesis:** At  $k^{th}$  iteration of Huffman, all nodes in the queue is a subtree of  $T$  (after inversions).

**Base case:** all nodes are leaves of  $T$ .

**Inductive step:** Huffman extracts  $A, B$  from the  $Q$ .

**Case 1:**  $A, B$  is a siblings in  $T$ .

Their newly created parent node in  $H$  corresponds to their parent in  $T$ .

(used induction hypothesis here.)

# Lemma: prefix $T \rightarrow$ Huffman $H$ via inversion

**Induction Hypothesis:** At  $k^{\text{th}}$  iteration of Huffman, all nodes in the queue is a subtree of  $T$  (after inversions).

**Case 2:**  $A, B$  is not a siblings in  $T$ .

WLOG, in  $T$ ,  $\text{depth}(A) \geq \text{depth}(B)$  &  $A$  is  $C$ 's sib.

Note  $B$  can't overlap  $C$  because

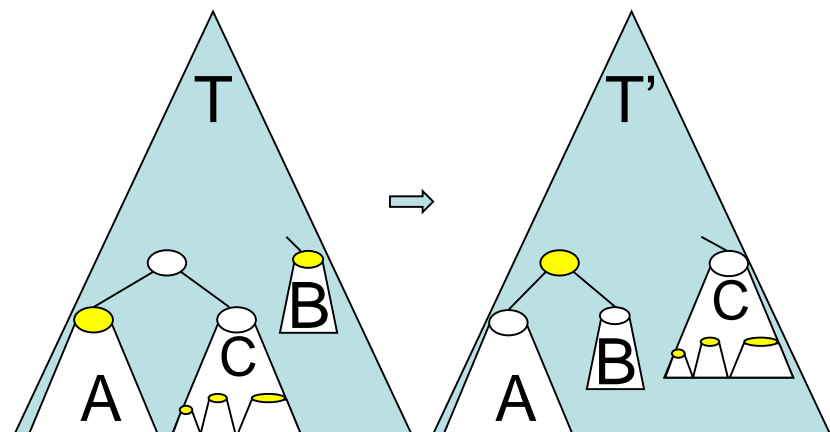
- If  $B = C$ , we have case 1.
- If  $B$  is a subtree of  $C$ ,  $\text{depth}(B) > \text{depth}(A)$ .
- If  $C$  is a subtree of  $B$ ,  $A$  and  $B$  overlaps.

Now, note that

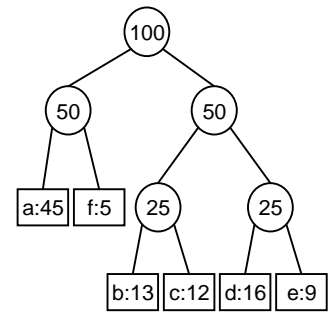
- $\text{depth}(A) = \text{depth}(C) \geq \text{depth}(B)$
- $\text{freq}(C) \geq \text{freq}(B)$  because Huff picks the min 2.

So,  $B - C$  is an inversion.

Swapping gives  $T'$  that satisfies the induction.



# Correctness Strategy



## Lemma:

Any prefix code tree  $T$  can be converted to a Huffman tree  $H$  via inversion-exchanges

## Corollary:

Huffman tree is optimal.

## Proof:

Apply the above lemma to any optimal tree  $T = T_1$ . The lemma only exchanges inversions, which never increase cost.

So,  $cost(T_1) \geq cost(T_2) \geq cost(T_4) \geq \dots \geq cost(H)$ .

# Data Compression

- Huffman is **optimal**.
- **BUT** still might do better!
  - Huffman encodes fixed length blocks. What if we vary them?
  - Huffman uses one encoding throughout a file. What if characteristics change?
  - What if data has structure? E.g. raster images, video,...
  - Huffman is lossless. Necessary?
- LZ77, JPG, MPEG, ...

The rest are just for fun.  
(I learnt compression this morning.)

# Adaptive Huffman coding

Often, data comes from a stream.

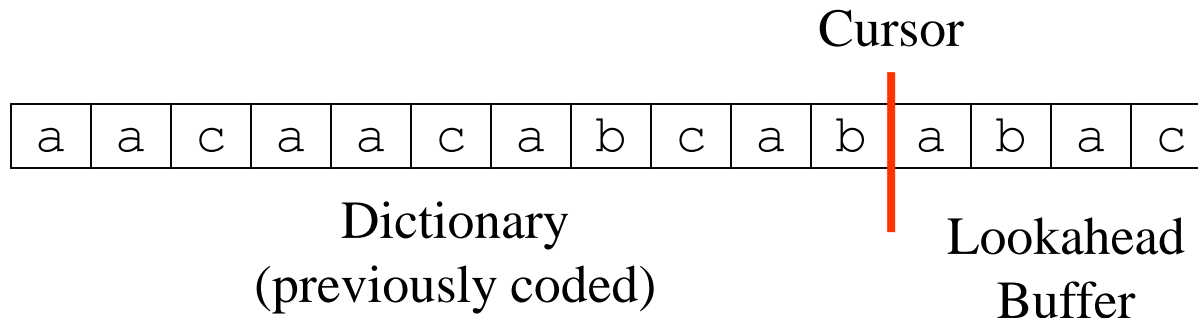
Difficult to know the frequencies in the beginning.

There are multiple ways to update Huffman tree.

FGK (Faller-Gallager-Knuth)

- There is a special external node, called 0-node, is used to identify a newly-coming character.
- Maintain the tree is sorted.
- When the freq is increased by 1, it can create inverted pairs. In that case, we swap nodes, subtrees, or both.
- (Exercise to figure out how to do exactly!)

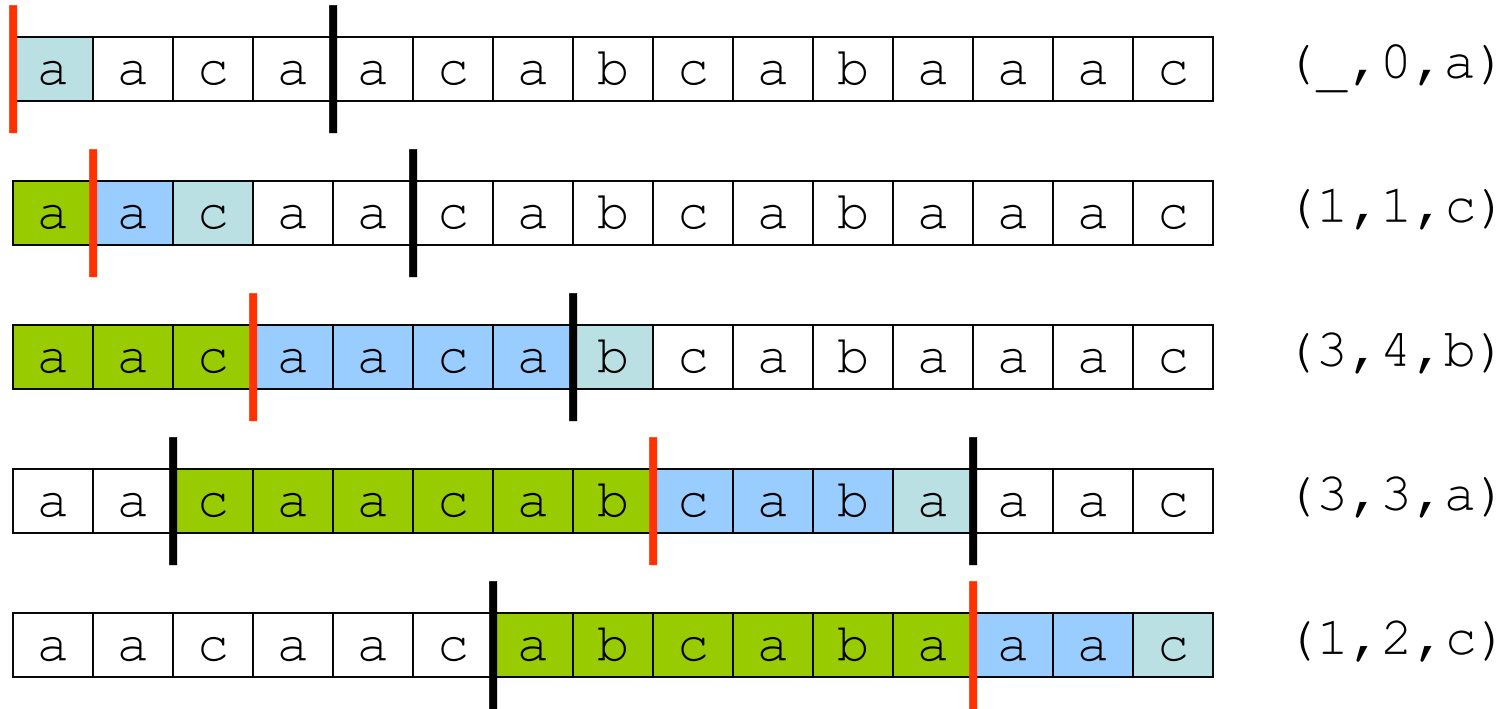
# LZ77




- **Dictionary** and **buffer** “windows” are fixed length and slide with the **cursor**
- **Repeat:**  
Output  $(p, l, c)$  where
  - $p$  = position of the longest match that starts in the dictionary (relative to the cursor)
  - $l$  = length of longest match
  - $c$  = next char in buffer beyond longest matchAdvance window by  $l + 1$

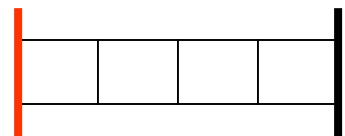
Theory: it is optimal if the windows size tends to  $+\infty$  and string is generated by Markov chain. [WZ94]

# LZ77: Example



 Dictionary (size = 6)

 Longest match

 Buffer (size = 4)

 Next character



# How to do it even better?

gzip

1. Based on LZ77.
2. Adaptive Huffman code the positions, lengths and chars
3. Non greedy: possibly use shorter match so that next match is better
4. ....

In general, compression is like prediction.

1. The entropy of English letter is 4.219 bits per letter
2. 3-letter model yields 2.77 bits per letter
3. Human experiments suggested 0.6 to 1.3 bits per letter.

For example, you can use deep net to predict and compression 1 GB of wiki to 160MB. (search deepzip)

(to compare, gzip 330MB, Huffman 500-600MB)

We are reaching the limit of human compression!

(8bit \* 0.16 ~ 1.2 bit per symbol)

# 50'000€ Prize for Compressing Human Knowledge

(widely known as the Hutter Prize)

Compress the 100MB file enwik8 to less than the current record of about 16MB

- [The Task](#)
- [Motivation](#)
- [Detailed Rules for Participation](#)
- [Previous Records](#)
- [More Information](#)
- [Newsgroup on the contest and prize](#)
- [History](#)
- [Committee](#)
- [Donations](#)
- [Frequently Asked Questions](#)
- [Contestants](#)
- [Links](#)
- [Disclaimer](#)

*News: Alexander Rhatushnyak is also the **fourth Winner!** Congratulations!*



*... the contest continues ...*



Being able to compress well is closely related to intelligence as explained below. While intelligence is a slippery concept, file sizes are hard numbers. Wikipedia is an extensive snapshot of Human Knowledge. If you can compress the first 100MB of Wikipedia better than your predecessors, your (de)compressor likely has to be smart(er). The intention of this prize is to encourage development of intelligent compressors/programs as a path to AGI.

Ultimate Answer? See Kolmogorov complexity.

Disclaimer: I don't recommend playing such competitions unless you REALLY know what you are doing.