CSE 421

Greedy: Huffman Codes

Yin Tat Lee

Experiment Y: Iteration 1

- Iteration 1:

More quizzes or polls (Connor Aksama)

Returning to largely in-person classes and experiences Jan. 31 Inbox ×

President Ana Mari Cauce and Provost Mark... 12:07 PM (4 minutes ago)

to me

View the web version of this message

UNIVERSITY of WASHINGTON

OFFICE OF THE PRESIDENT

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

  Better?

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

  Even better:

  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

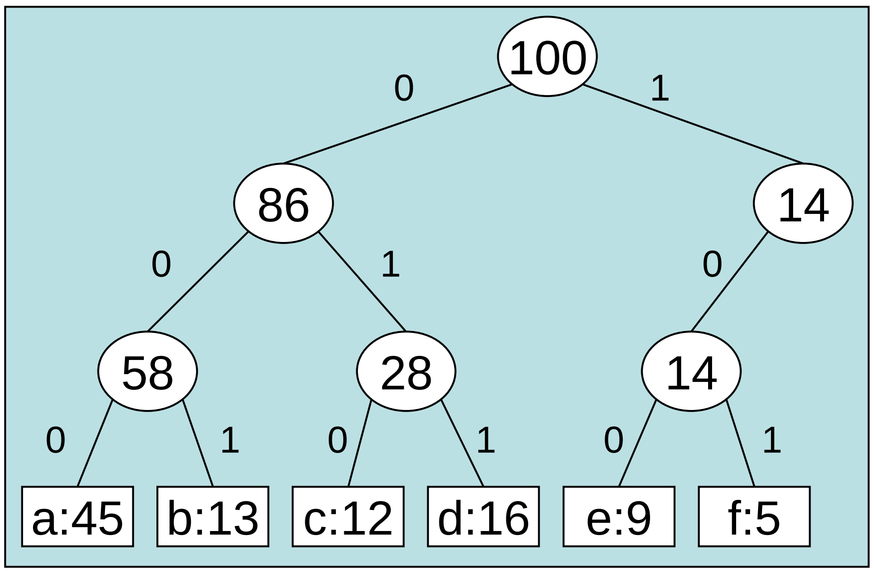
1101110 = cf or ec?

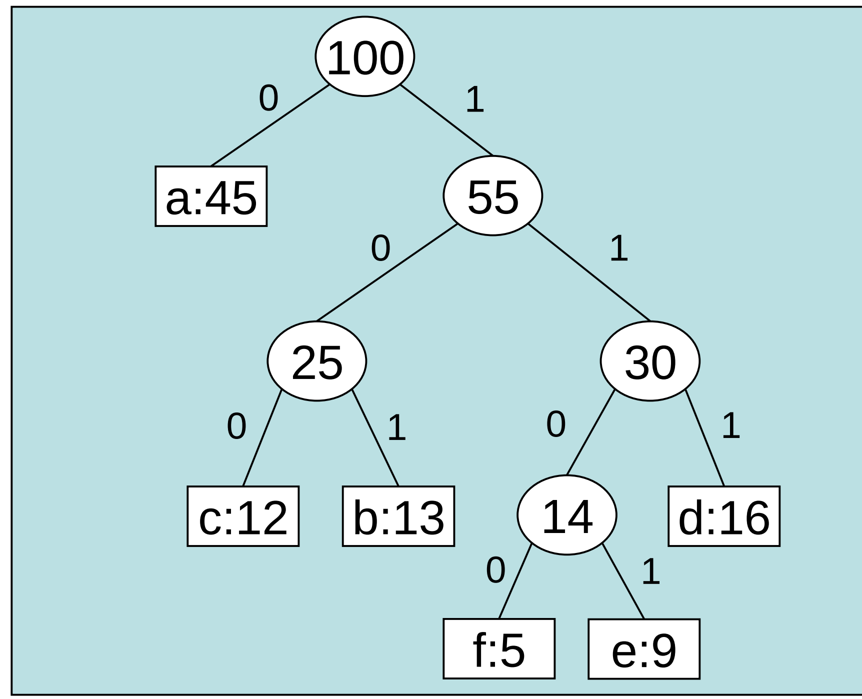
- Prefix codes

  no code word is prefix of another (unique decoding)

Prefix Codes = Trees

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





$\underbrace{1\ 0\ 1}_{f}\ \underbrace{0\ 0\ 0}_{a}\ \underbrace{0\ 0\ 1}_{b}$

$\underbrace{1\ 1\ 0\ 0}_{f}\ \underbrace{0}_{a}\ \underbrace{1\ 0\ 1}_{b}$

Quiz

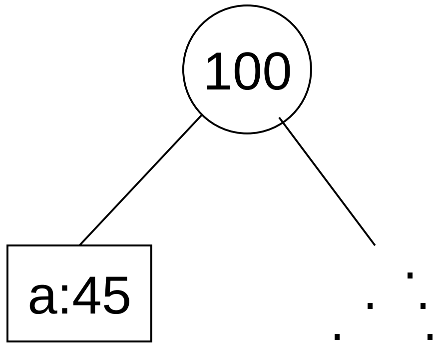
Given k symbols. Show that there is a prefix code with length l_i for symbol i if

$$\sum_i 2^{-l_i} = 1.$$

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%

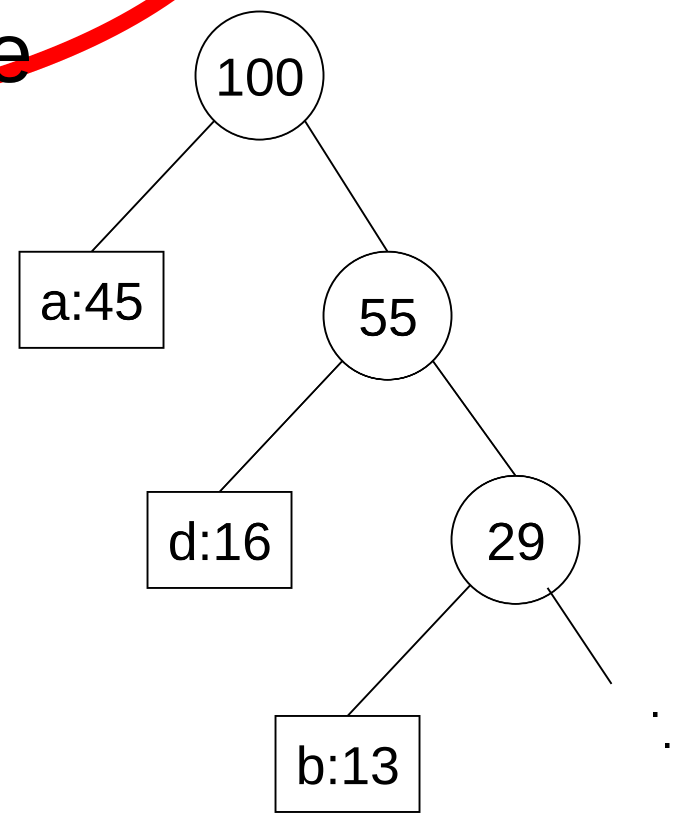
- Top down: Put *most* frequent under root, then recurse

- **Too greedy: unbalanced tree**

  .45*1 + .16*2 + .13*3 … = 2.34

  not too bad, but imagine if all freqs were ~1/6:

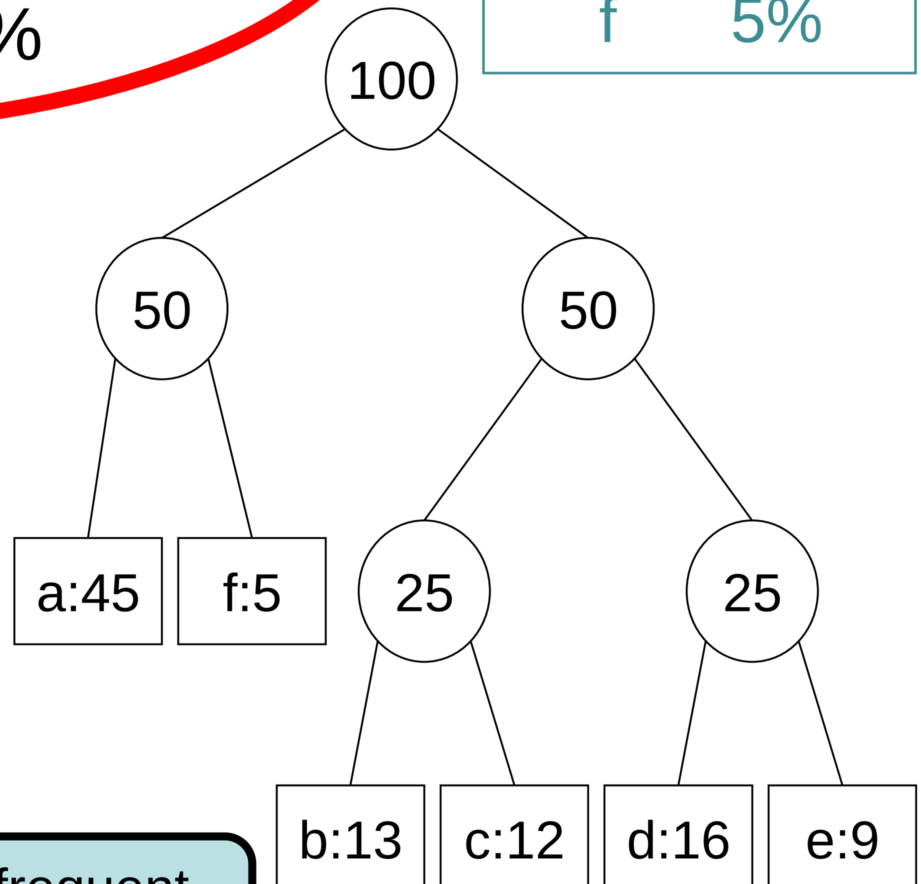
  (1+2+3+4+5+5)/6=3.33



~~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*.5+3*.5 = 2.5
- But this tree can easily be improved! (How?)

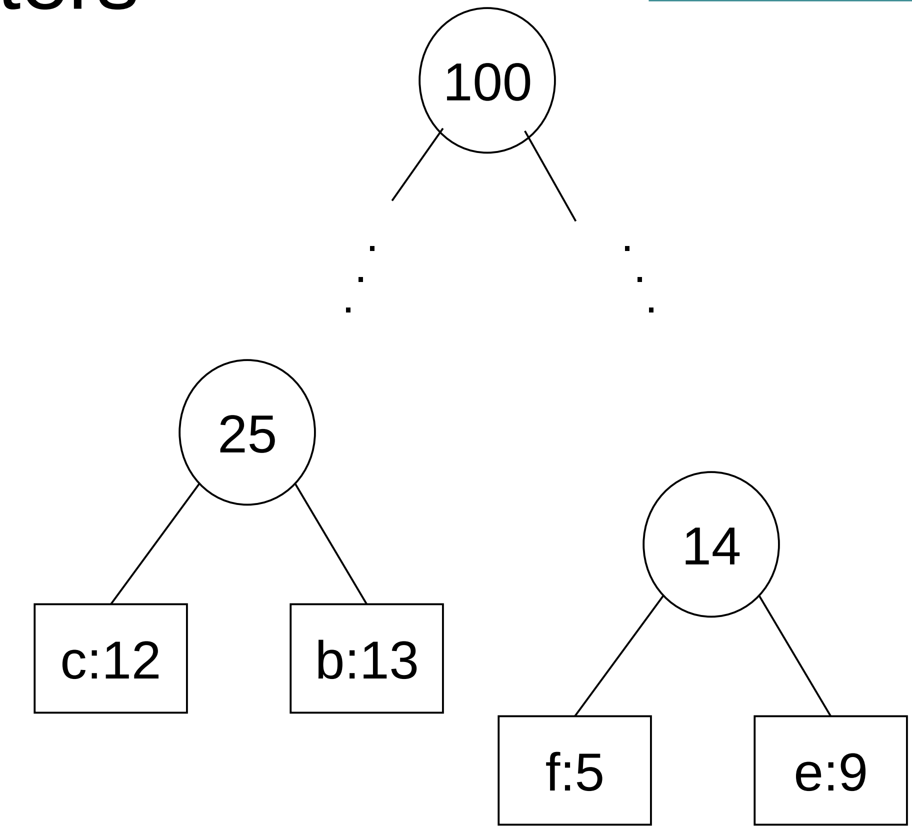


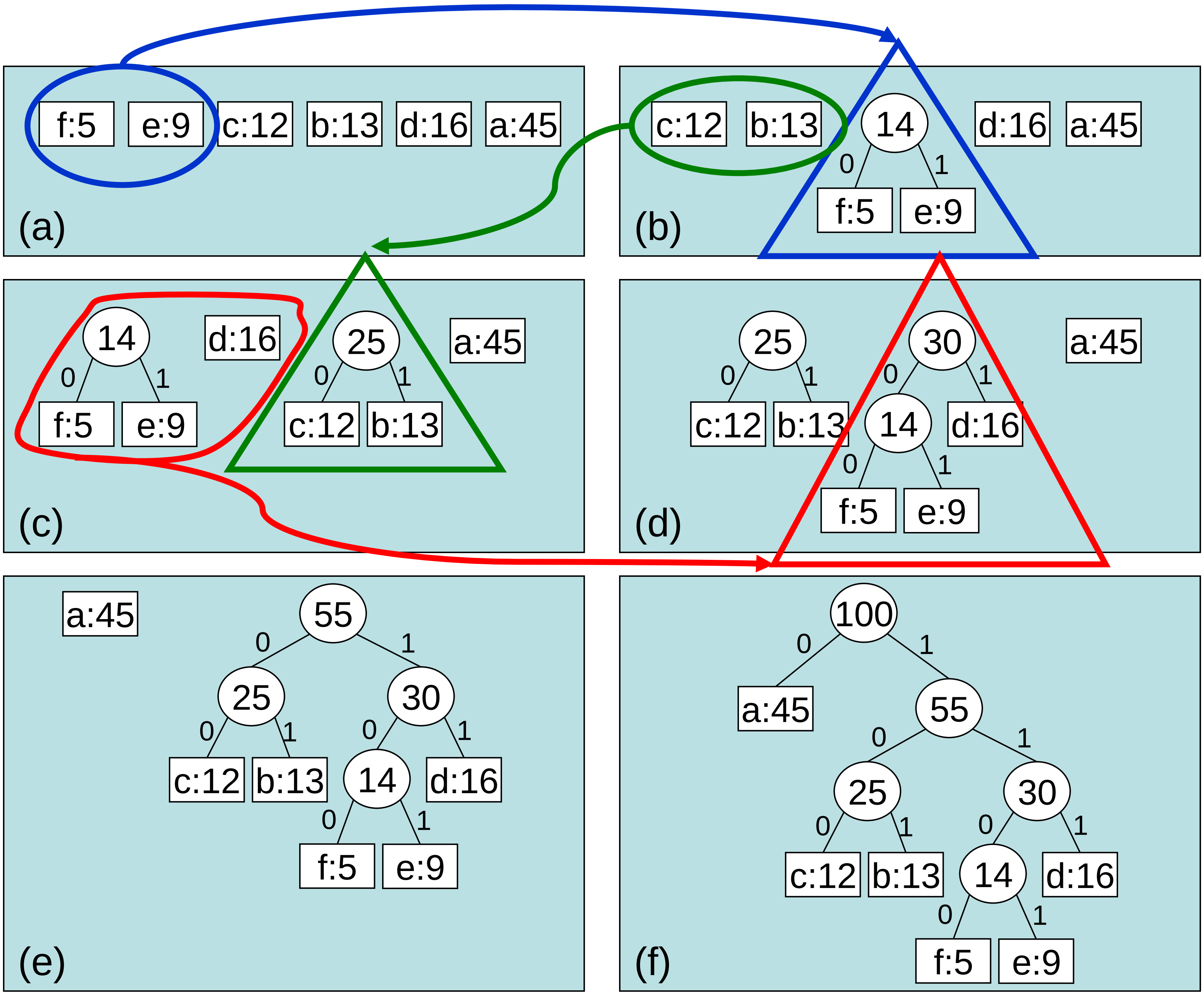
Idea: To avoid swapping, the lowest frequent letters must be at the bottom.

Greedy idea #3

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

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





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

Huffman's Algorithm (1952)

According to wiki, this is Huffman's class project.

- Algorithm:

  Insert each letter as a leaf into priority queue by freq

  While queue length > 1

  Remove smallest 2 nodes; call them x, y

  Create a new node z with x, y as children and

  $$\text{freq}(z) = \text{freq}(x) + \text{freq}(y)$$

  insert z into queue

- Runtime: $O(n \log n)$
- Goal: Minimize $cost(T) = \sum_c \text{freq}(c) \cdot \text{depth}(c)$

T = Tree
c = alphabet
(leaves)

Correctness Strategy

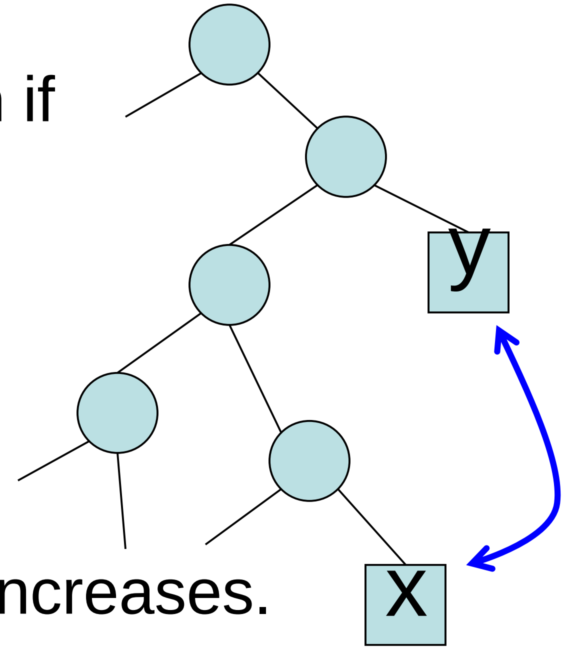
- Optimal solution may not be unique, so cannot prove that greedy gives the *only* possible answer.
- Instead, show greedy's solution is as good as any.
- How: an exchange argument
- Identify inversions: node-pairs whose swap improves tree

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

depth(x) ≥ depth(y)

and

freq(x) ≥ freq(y)



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

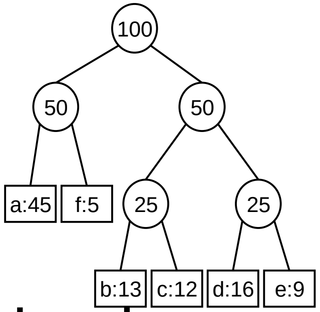
Why? All other things being equal, better to give more frequent letter the shorter code.

$$\overbrace{\big(d(x)f(x) + d(y)f(y)\big)}^{\text{before}} - \overbrace{\big(d(x)f(y) + d(y)f(x)\big)}^{\text{after}}$$

$$= \big(d(x) - d(y)\big)\big(f(x) - f(y)\big) \geq 0$$

I.e., non-negative cost savings.

General Inversions

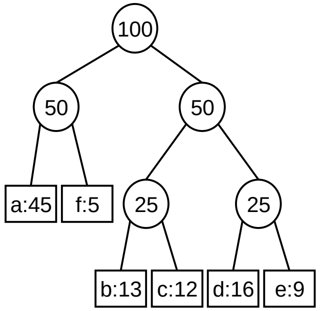


- Define the frequency of an **internal** node to be the sum of the frequencies of the leaves in that subtree.

- We can generalize
  - the defn of inversion for any pair of disjoint 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

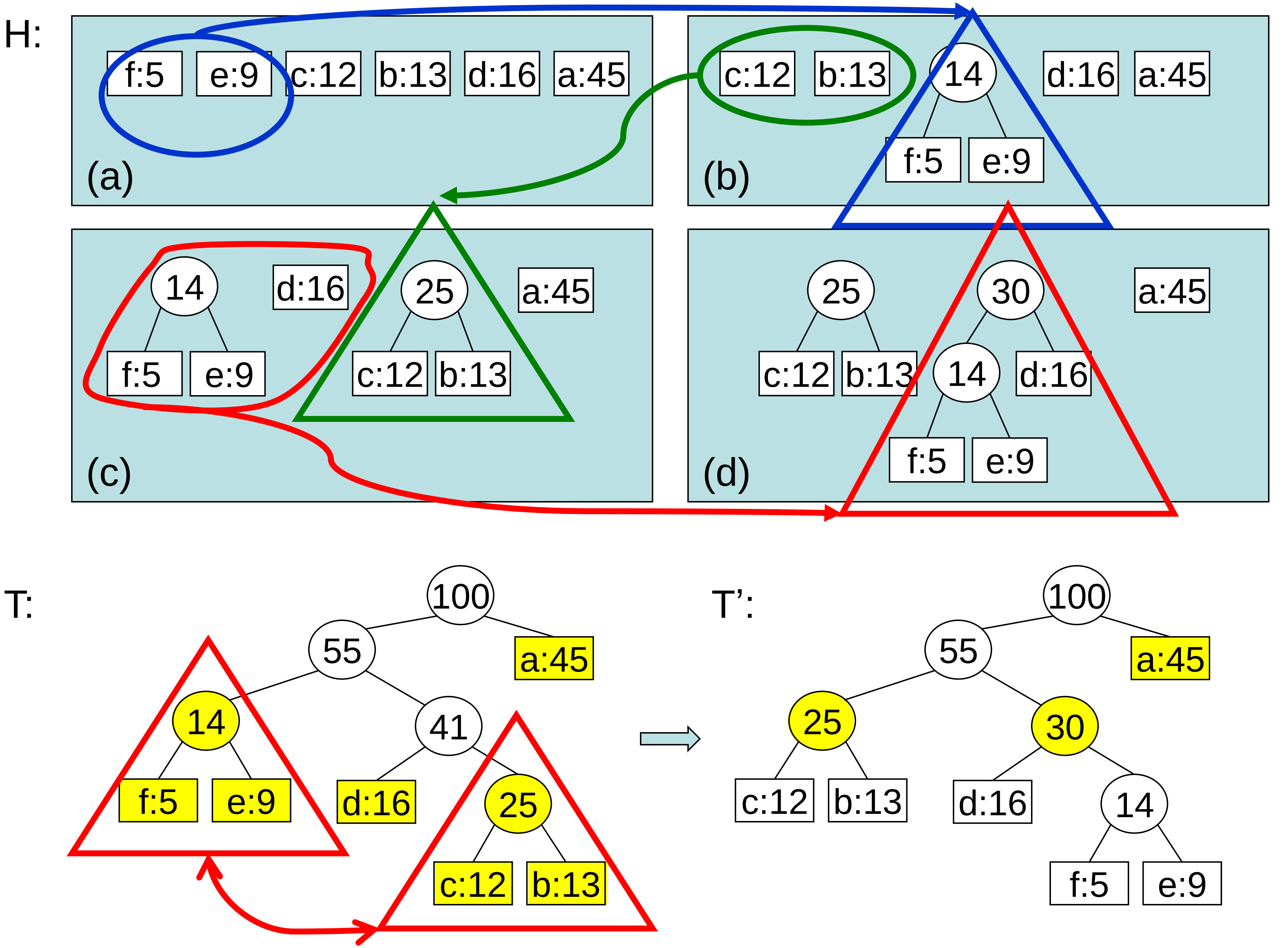


Lemma:
Any prefix code tree T can be converted to a huffman tree H via inversion-exchanges

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

Induction: All nodes in the queue is a subtree of *T* (after inversions)



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^{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, $depth(A) \geq depth(B)$ & A is C's sib.

Note B can't overlap C because

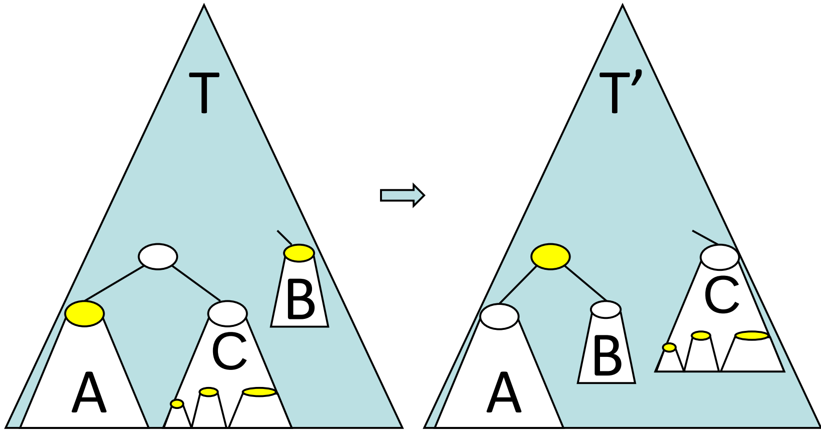
- If $B = C$, we have case 1.
- If B is a subtree of C, $depth(B) > depth(A)$.
- If C is a subtree of B, A and B overlaps.

Now, note that

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

So, $B - C$ is an inversion.

Swapping gives T' that satisfies the induction.



Quiz

This is the last lecture of greedy method.
So, I give some example with different favors.

YinTat wants to throw a zoom party where
- every person knows at least 4 people
- every person doesn't know at least 4 people.

Given the undirected graph representing the friendship status of his n friends.

Question: Find an efficient algorithm that finds the largest number of people he can invite subject to those constraints.

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?
- GZIP, JPG, MPEG, …

Adaptive Huffman coding

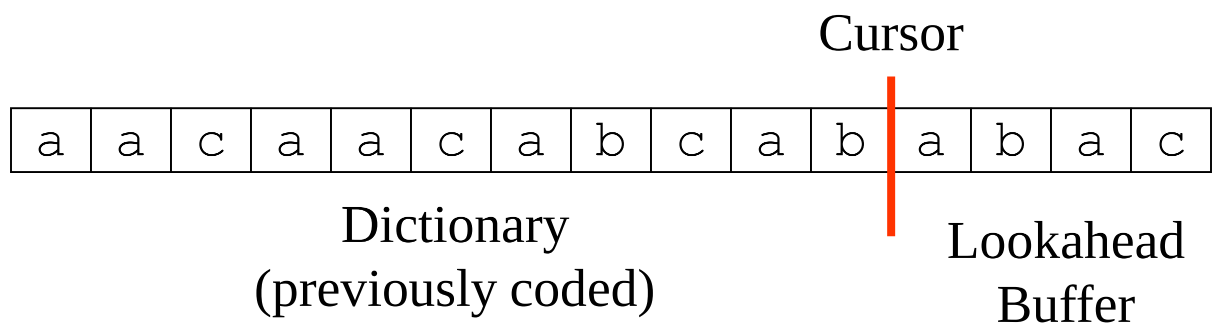
Often, data comes from a stream.
Difficult to know the frequencies in the beginning.
There are multiple ways to update Huffman tree.

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.

Only algorithm in the IEEE Milestones

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)

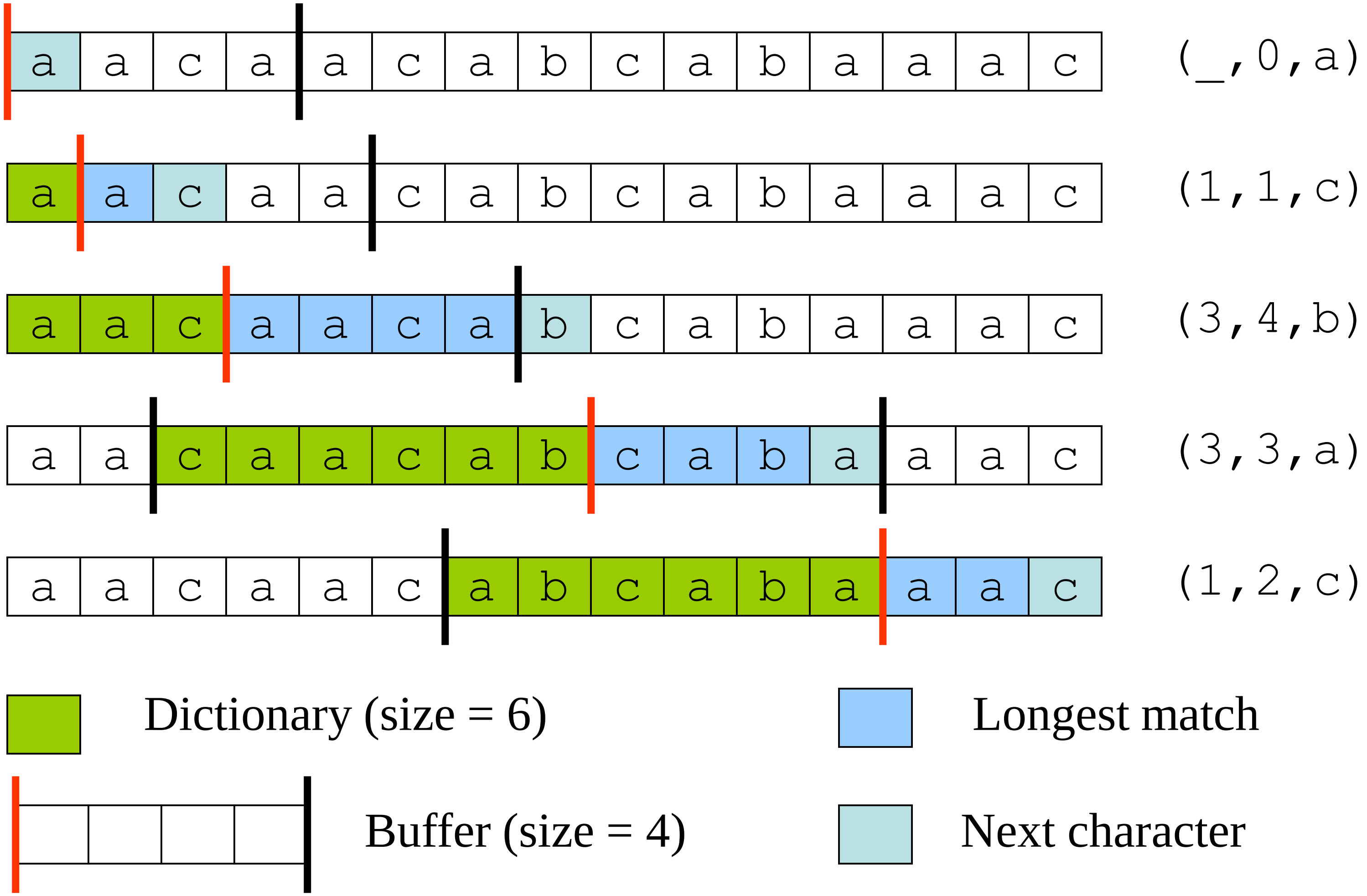
  l = length of longest match

  c = next char in buffer beyond longest match

  Advance 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



How to do it even better?

gzip

1. Based on LZ77.
2. Adaptive Huffman code the positions, lengths and chars
3. ….

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 neural network to predict and compression 1 GB of wiki to 108MB.

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

See http://www.mattmahoney.net/dc/text.html

500'000€ Prize for Compressing Human Knowledge
(widely known as the Hutter Prize)

Compress the 1GB file `enwik9` to less than the current record of about 115MB

News: *Artemiy Margaritov is the* ***fifth*** *Winner! Congratulations!*

... the contest continues ...

- The Task
- Motivation
- **Detailed Rules for Participation**
- Previous Records
- More Information
- **Discussion forum on the contest and prize**
- History
- Committee
- **Frequently Asked Questions**
- Contestants
- Links
- Disclaimer

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 1GB 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?

Kolmogorov complexity $K(T) = \min_{\text{Program } P \text{ outputs } T} \text{length}(P)$.