CSE 421
Algorithms

Huffman Codes:
An Optimal Data Compression Method

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

Compression Example

100k file, 6 letter alphabet:

File Size:

ASCII, 8 bits/char: 800kbits

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

Why?

Storage, transmission vs 5 Ghz cpu

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: →

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

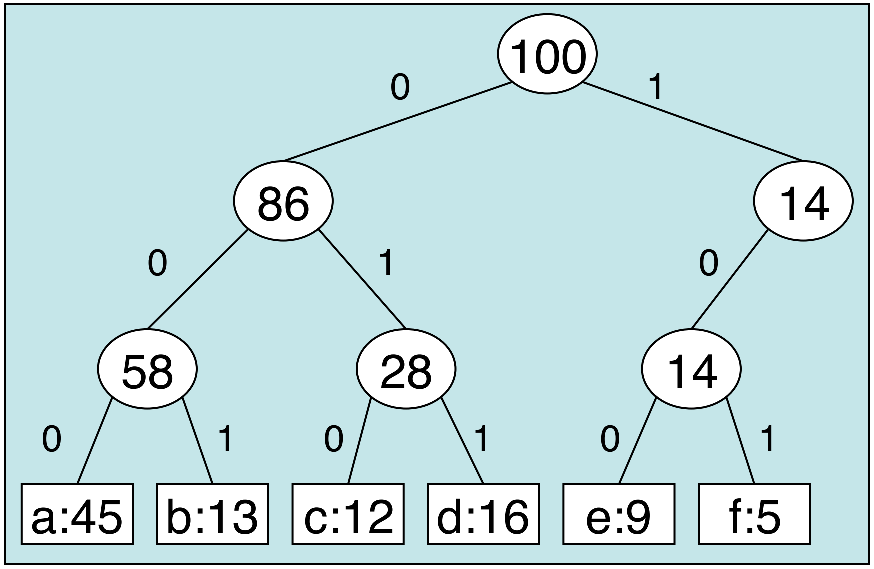
- all code words equal length?

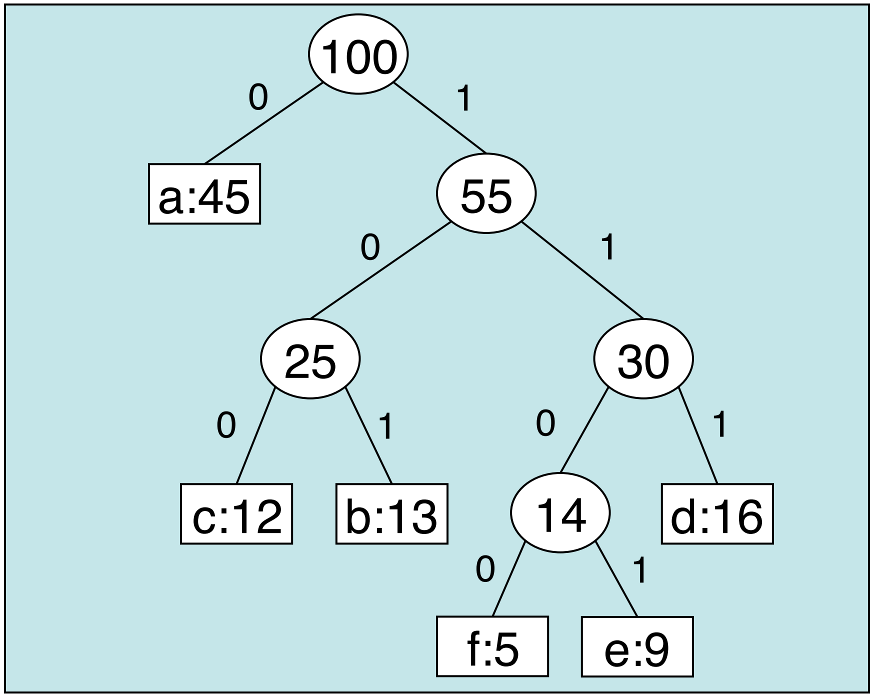
Prefix codes

- no code word is prefix of another (unique decoding)

Prefix Codes = Trees

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





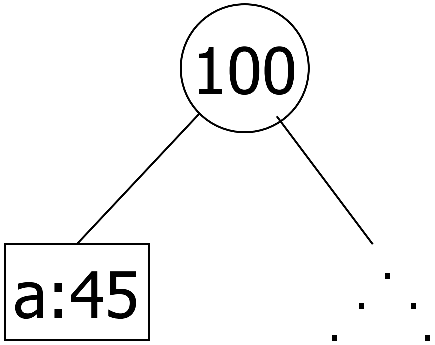
$$\underbrace{1\ 0\ 1}_{f}\ \underbrace{0\ 0\ 0}_{a}\ \underbrace{0\ 0\ 1}_{b}$$

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

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

Greedy Idea #1

Put most frequent under root, then recurse …



~~Greedy Idea #1~~

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

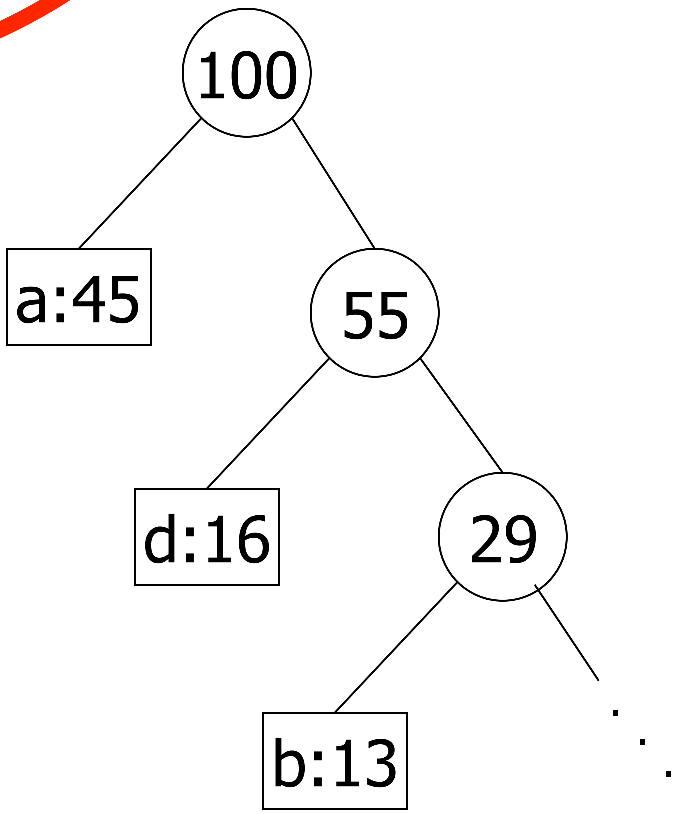
a	45%
b	13%
c	12%
d	16%
e	9%
f	5%

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



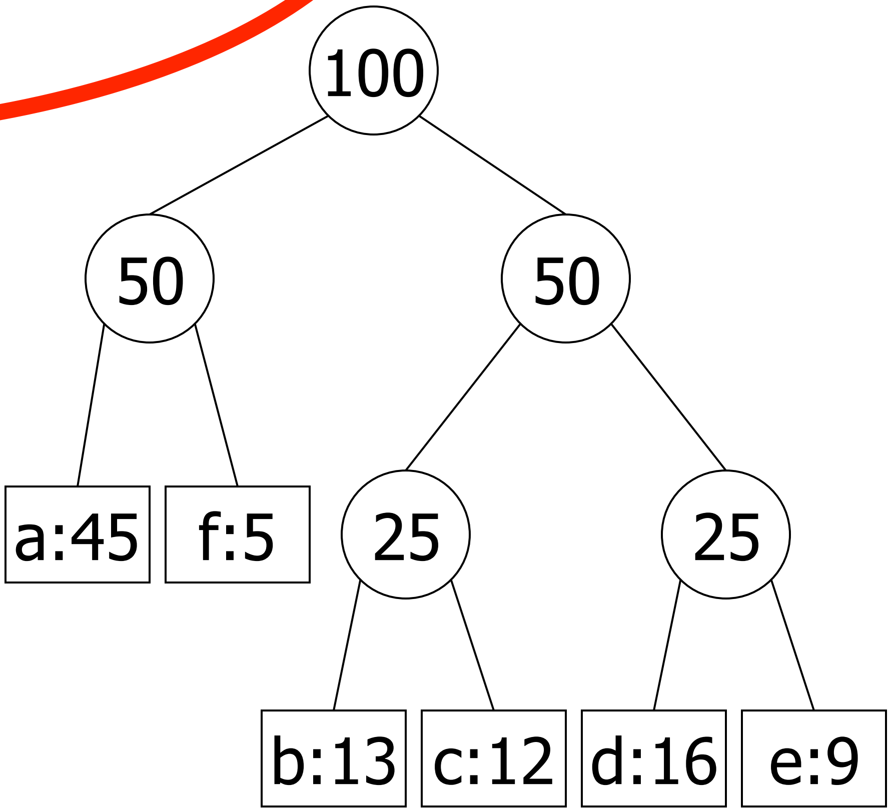
Greedy Idea #2

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

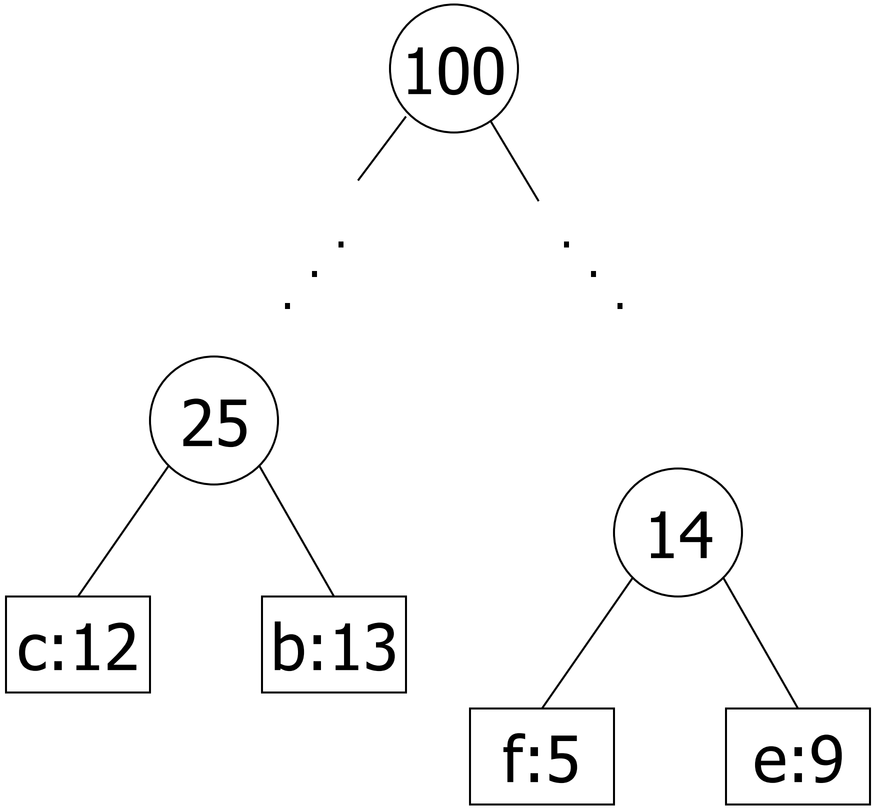
But this tree can easily be improved! (How?)

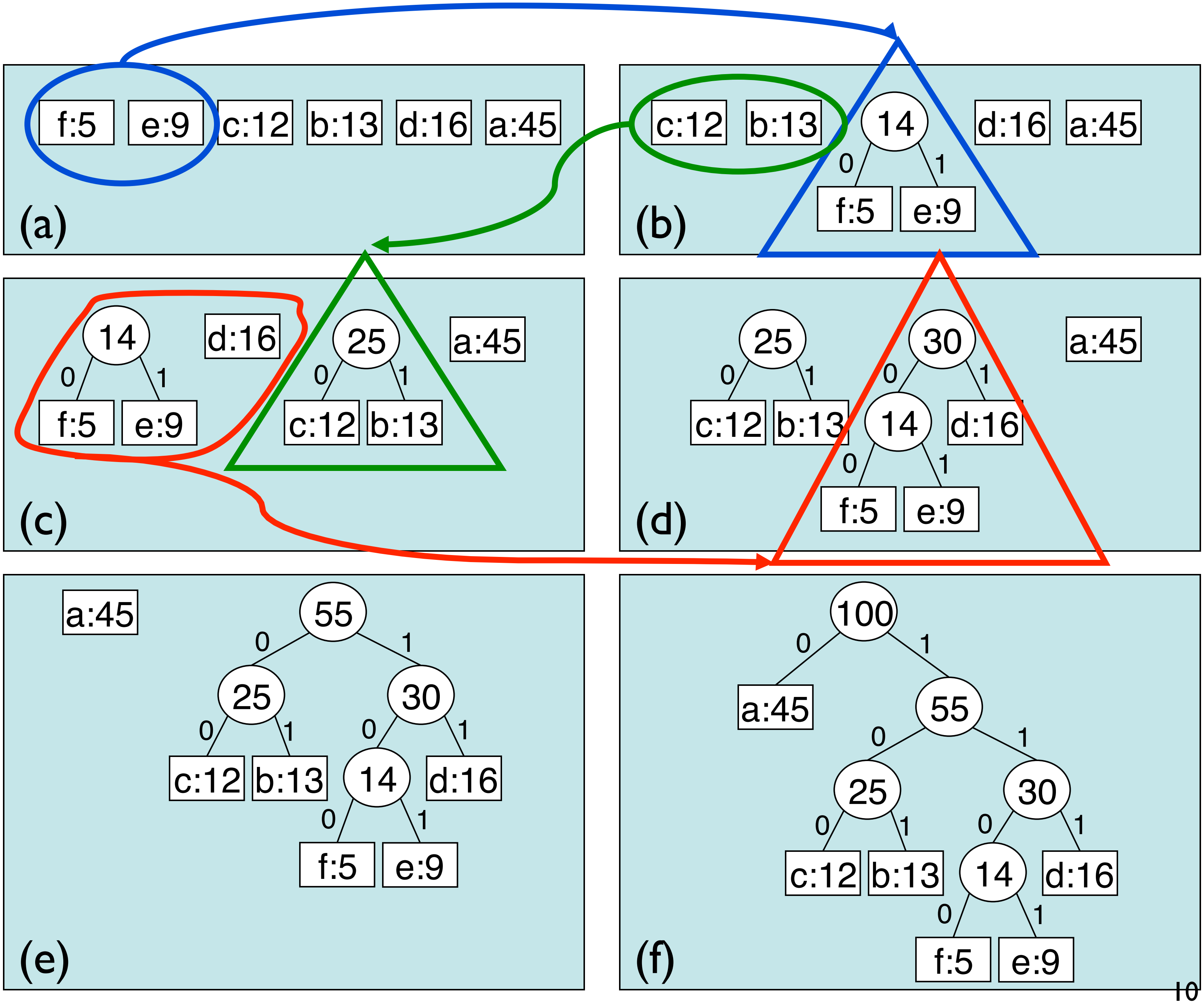


a	45%
b	13%
c	12%
d	16%
e	9%
f	5%

Greedy idea #3

Bottom up: Group *least* frequent letters near bottom





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

Huffman’s Algorithm (1952)

Algorithm:

```
insert node for 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) heap ops: O(n log n)

Goal: Minimize $Cost(T) = \sum_{c \in C} \text{freq}(c)*\text{depth}(c)$

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

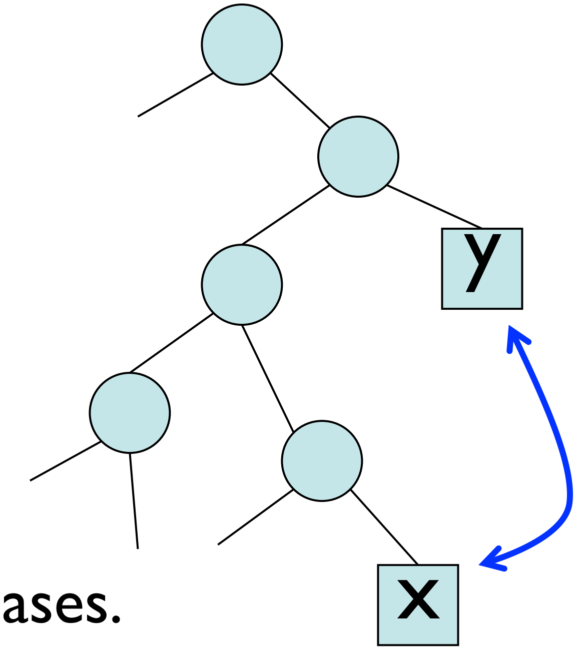
- Identify *inversions:* node-pairs whose swap improves tree
- To compare trees T (arbitrary) to H (Huffman): run Huff alg, tracking subtrees in common to T & H; discrepancies flag inversions; swapping them incrementally xforms T to H

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.

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

$$(d(x) - d(y)) * (f(x) - f(y)) \ge 0$$

I.e., non-negative cost savings.

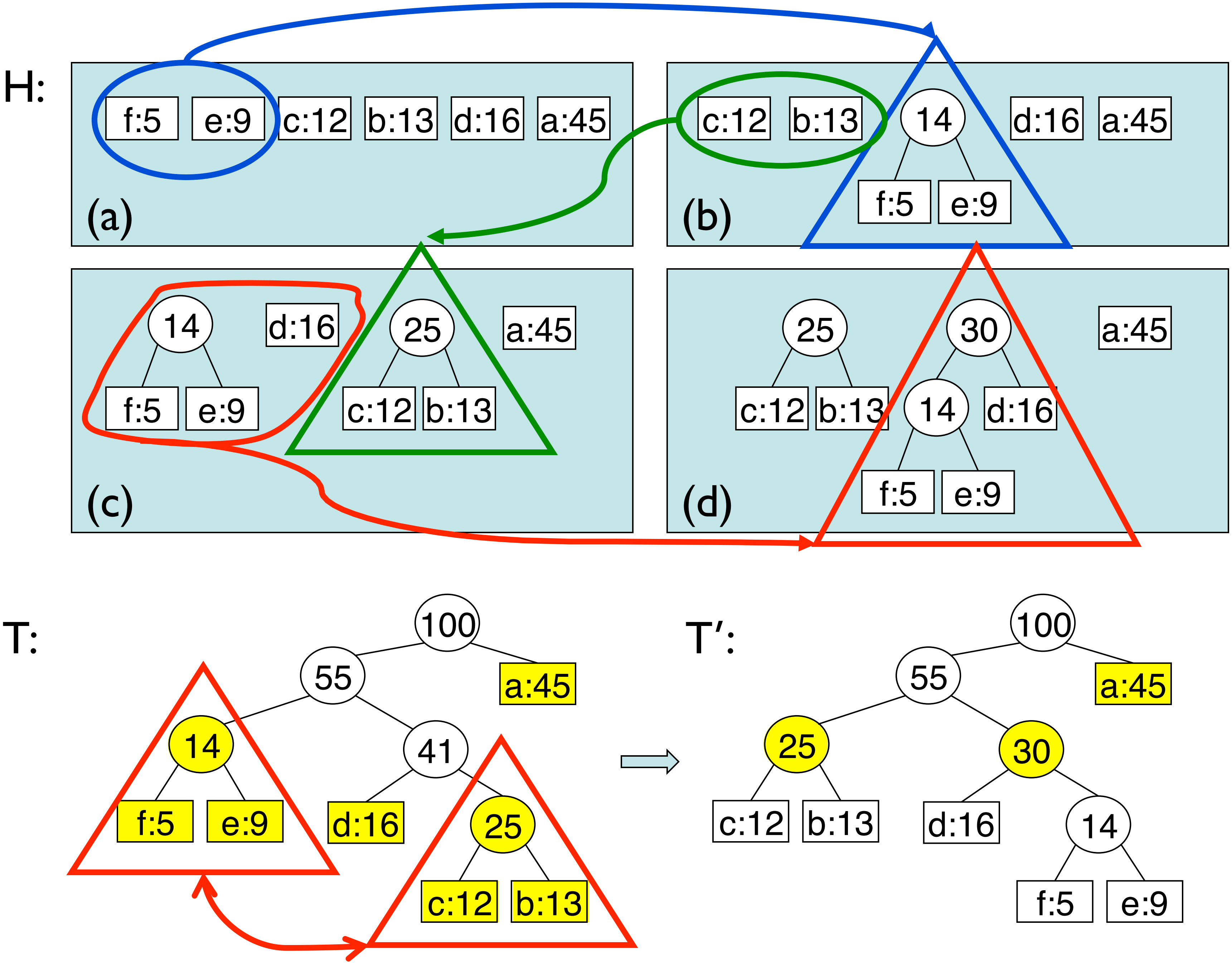
General Inversions

Define the frequency of an *internal* node to be the sum of the frequencies of the leaves in that subtree (as shown in the example trees above).

Given that, the definition of inversion on slide 13 easily generalizes to an arbitrary pair of nodes, and the associated claim still holds: exchanging an inverted pair of nodes (& associated subtrees) cannot raise the cost of a tree.

Proof: Homework

The following slide is heavily animated, which doesn’t show too well in print. The point is to illustrate the Lemma on slide 17. Idea is to run Huffman alg on the example above and compare successive subtrees it builds to subtrees in an arbitrary tree T. While they agree (marked by yellow), repeat; when they first differ (in this case, when Huffman builds node 30), identify an inversion in T whose removal would allow them to agree for at least one more step, i.e., T’ is more like H than T, but costs no more. Slide 16 is an example; slide 17 sketches the proof in general.



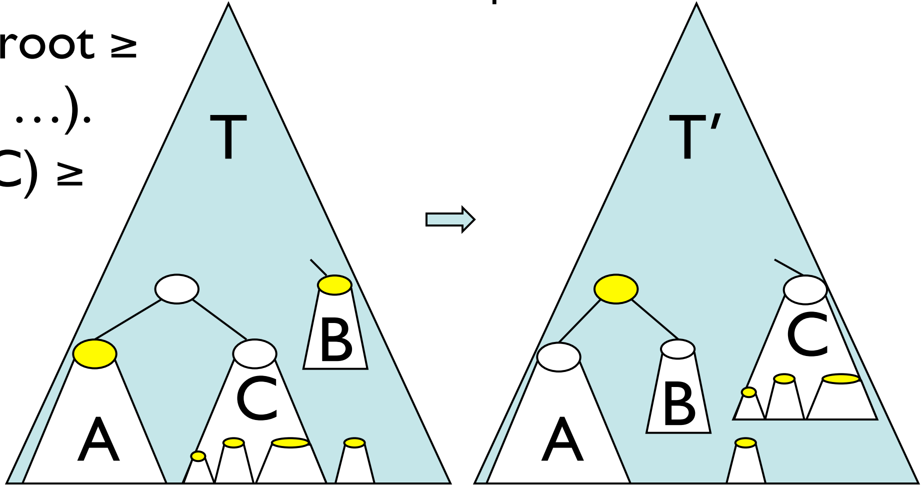
In short, where T first differs from H flags an inversion in T

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

Pf Idea: Run Huffman alg; "color" T's nodes to track matching subtrees between T, H. Inductively: yellow nodes in T match subtrees of H in Huffman's heap at that stage in the alg. & yellow nodes partition leaves.
Initially: leaves yellow, rest white.
At each step, Huffman extracts A, B, the 2 min heap items; both yellow in T.

Case 1: A, B match siblings in T. Then their newly created parent node in H corresponds to their parent in T; paint it yellow, A & B revert to white.

Case 2: A, B not sibs in T. WLOG, in T, depth(A) ≥ depth(B) & A is C's sib. Note B can't overlap C (B = C ⇒ case 1; B subtree of C contradicts depth; B contains C contradicts partition). In T, the freq of C's root ≥ freqs of all yellow nodes init (≠ ∅ since …). Huff's picks (A & B) were min, so freq(C) ≥ freq(B). ∴ B:C is an inversion–B is no deeper/no more frequent than C. Swapping gives T′ more like H; repeating ≤ n times converts T to H.



Theorem: Huffman is optimal

Pf: Apply the above lemma to any optimal tree $T=T_1$. The lemma only exchanges inversions, which never increase cost, so, cost of successive trees is monotonically non-increasing, and the last tree is H: $\text{cost}(T_1) \geq \text{cost}(T_2) \geq \text{cost}(T_3) \geq \ldots \geq \text{cost}(H)$.

Corr: can convert any tree to H by inversion-exchanges (general exchanges, not just leaf exchanges)

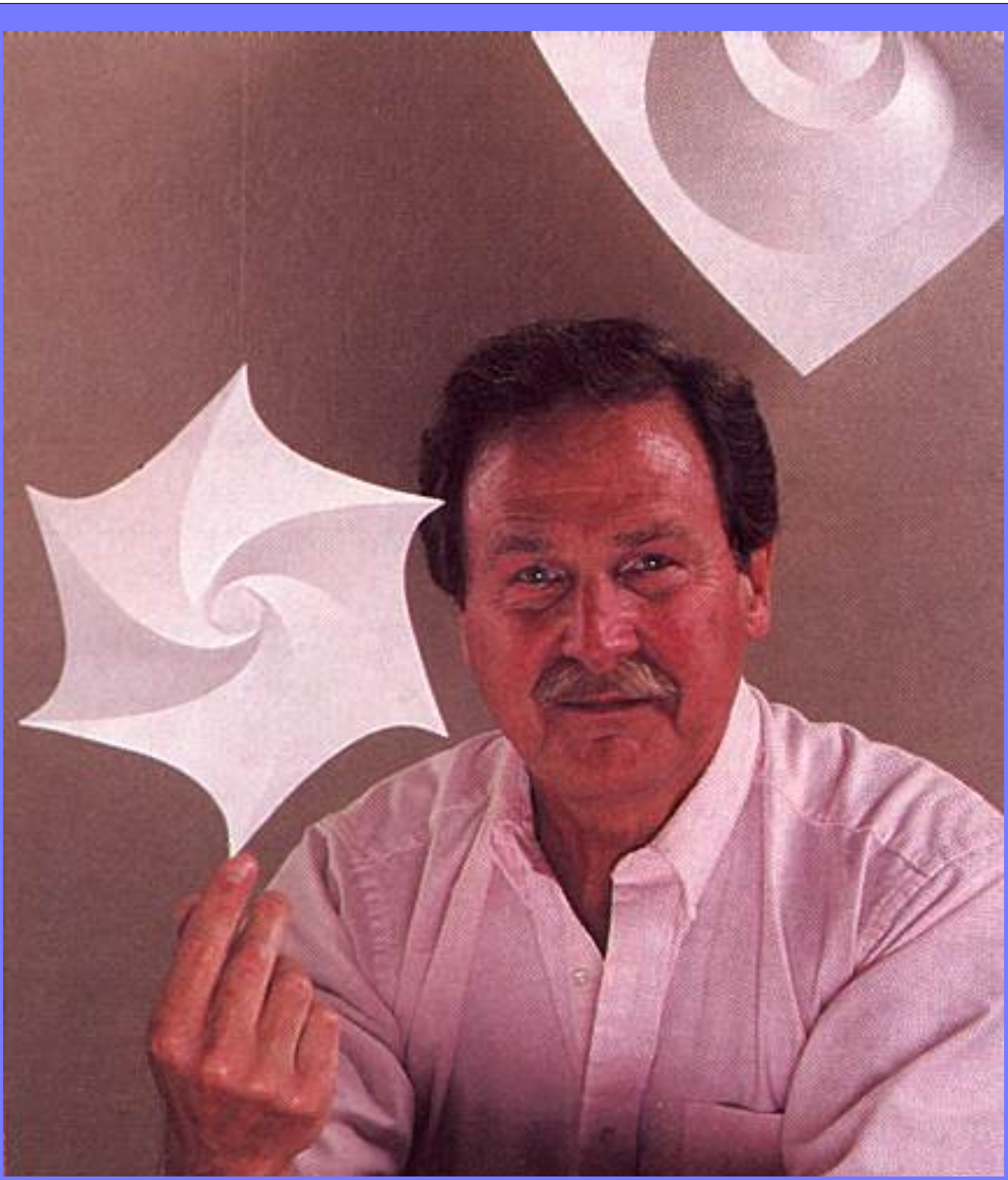
Data Compression

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?

LZW, MPEG, …



David A. Huffman, 1925-1999