CS 430 Lecture 19 Activities

Fractional Knapsack Problem

- Prove optimal substructure
- Prove Greedy Choice

1. Prove that the Fractional Knapsack Problem has optimal substructure.
   - What is the problem?
   - Assume you have an optimal answer.

$$p : w_1, w_2, \ldots, w_n$$
$$v_1, v_2, \ldots, v_n$$
$$0 \leq f_i \leq \text{ is the fractional amount of each item}$$
$$\sum_{subset \ 1 \leq 1 \leq n} f_i v_i \text{ is max}$$
$$\sum_{subset} f_i w_i \leq W$$

2. Try various "common sense" greedy approaches that divide the problem into a sub-problem(s) and try to come up with counter-examples or prove the greedy choice is correct using the "cut and paste" proof.
   - Max Value First – if you fill the "bag" up immediately, you may miss out on a better value combination.

$\mathbf{v_i}$	$\mathbf{w_i}$
10	5
6	3
5	2

   - Max $\left(\frac{v_i}{w_i}\right)$ first. In Greedy Order: $O(n \lg n)$ sort

$$\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \cdots \geq \frac{v_n}{w_n}$$
Assume fractional optimal answer $0 \leq f \leq 1$

   What is the optimal answer does not have the greedy choice? The optimal answer does not contain as much of item #1 as possible. "Cut and paste" proof Somet___ greedy choice w of item 1: w_1 of other items removed and the *value out* $\leq v_1$. The runtime is $O(n \lg n)$, sorting in $O(n \lg n)$ time and then just taking items.

Huffman Codes Problem

Data Encoding Background

- Data is a sequence of characters
- Fixed Length – Each character is represented by a unique binary string. Easy to encode, concatenate the codes together. Easy to decode, break off 3-bit codewords and decode each one.
- Variable Length – Give frequent characters shorter codewords, infrequent characters get long codewords. However, how do we decode if the length of the codewords are variable?
- Prefix Codes – No codeword is a prefix of another codeword. Easy to encode, concatenate the code together. Easy to decode, since no codeword is a prefix of another, strip off one bit at a time and match to the unique prefix code

Huffman Codes

- Are a Data Compression technique using a greedy algorithm to construct an optimal variable length prefix code
- Use frequency of occurrence of characters to build an optimal way to represent each character as a binary string
- Use a Binary tree method – 0 means go to left child, 1 means go to right child (not a binary search tree).
- Cost of Tree in bits:

$$B(T) = \sum_{\text{for all } c \in C} \text{FREQ}(c) \times \text{DEPTH}(c)$$

Example

Table 19.1: A character-coding problem. A data file of 100,000 characters containes only the characters a–f, with the frequencies indicated. If each character is assigned a 3-bit codeword, the file can be encoded in 300,000 bits. Using the variable-length code shown, the file can be encoded in 224,000 bits.

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

The 224,000 bit came from multiplying the freqency (in thousands) of each character by the length of the variable codeword, which would represent its height in the binary tree, as described in Figure 19.1. $45(1) + 13(3) + 12(3) + 16(3) + 9(4) + 5(4) = 45 + 39 + 36 + 48 + 36 + 20 = 224$.

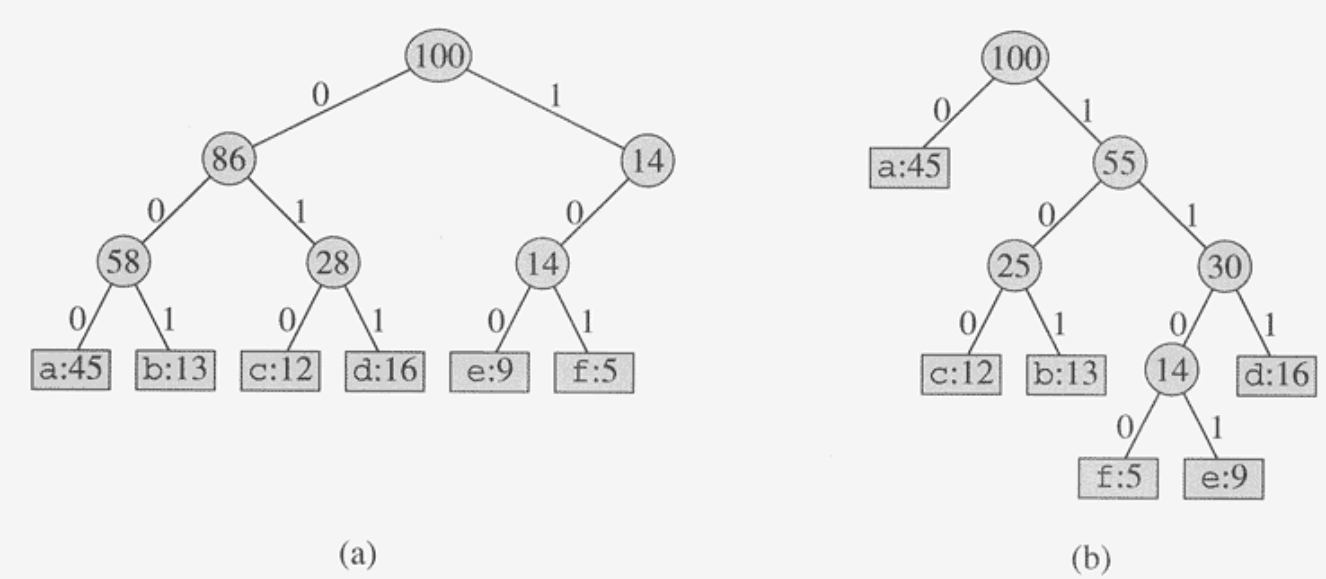


Figure 19.1: Trees corresponding to the coding schemes in Table 19.1. Each leaf is labeled with a character and its frequency of occurence. Each internal node is labeled with the sum of the frequencies of the leaves in its subtree. **(a)** The tree corresponding to the fixed-length code $a = 000, \ldots, f = 101$. **(b)** The tree corresponding to the optimal prefix code $a = 0, b = 101, \ldots, f = 1100$.

3. Prove that the Huffman Codes Problem has optimal substructure.
   Problem

   $$c_1, c_2, \ldots, c_n \quad \text{characters}$$
   $$f_1, f_2, \ldots, f_n \quad \text{frequency}$$

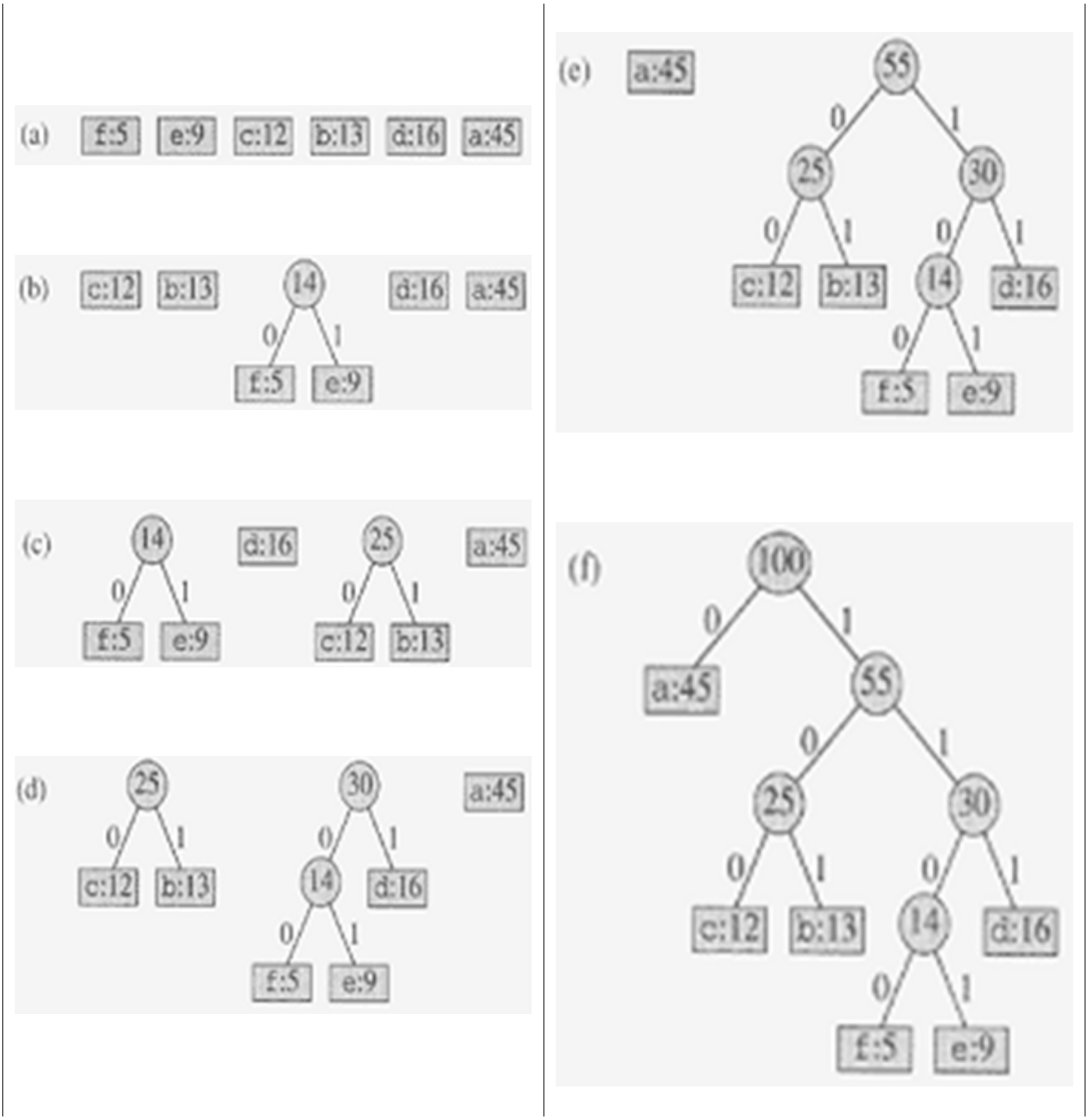
   Optimal Answer:

   $l_1, l_2, \ldots, l_n$ bit length of variable length prefix code

   $\sum_{i=1}^{n} f_i l_i$ is minimized

   There is an encoder/decoder decision tree for this optimal answer. There must be a character in that optimal answer with the shortest (or one of the shortest if there are multiple) l_k. If you remove char l_k from the problem,

The greedy approach is: Build the tree bottom up by using a minimum priority queue to merge 2 least frequent objects (objects are leaf nodes or other subtrees) together into a new subtree.



Huffman http://www.cs.auckland.ac.nz/software/AlgAnim/huffman.html

4. Proof this greedy algorithm leads to an optimal Huffman Code tree: Build the tree bottom up by using a minimum priority queue to merge 2 least frequent objects (objects are leaf nodes or other subtrees) together into a new subtree.