Huffman Code Trees

Alexandra Stefan

10/13/2020

Huffman code

- Application: file compression
- Example from CLRS:
  - File with 100,000 characters.

  - Characters: a,b,c,d,e,f
  - Frequency in thousands (e.g. the frequency of b is 13000):
  - Goal: binary encoding that requires less memory.

	a	b	c	d	e	f	File size after encoding
Frequency (thousands)	45	13	12	16	9	5	-
Fix length ASCII code (8bits)							8*100000 = 800,000
Fix-length codeword (3 bits)	000	001	010	011	100	101	3*100000+information about the encoding 300,000
Variable-length codeword	0	101	100	111	1101	1100	[(45*1) + (13*3) +(12*3)+(16*3)+ 9*4 + 5 * 4] * 10^3 = (45+39+36+48+36 +20)*10^3 = 224,000

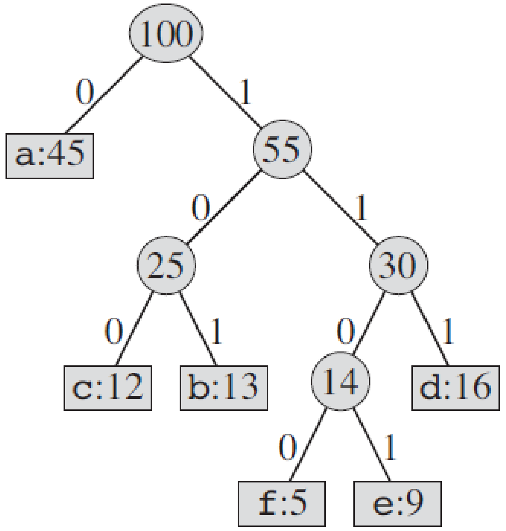
Number of bits and the number of different 0/1 patterns they produce :

1 bit (2) : 0,1
2 bits (4): 00,01,10,11
3 bits ($8=2^3$)
=> 3bits are sufficient to encode the given letters

Huffman codes

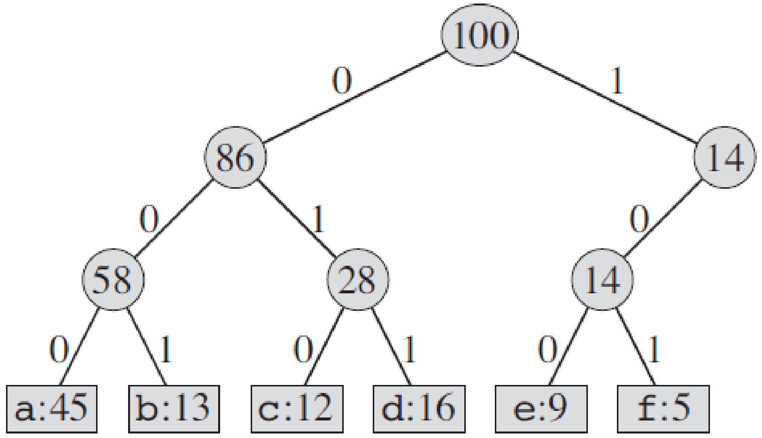
- Internal nodes contain the sum of probabilities of the leaves in that subtree.

Optimal prefix codeword tree (Huffman tree) – optimal encoding



a->0
b->101
c->100
d->111
e->1101
f->1100

Fixed-length codeword tree



Compute the number of bits needed for the whole file using each of these encodings.

Number of bits in code

45,000*1 + 13,000 * 3 + 12,000*3 + 16,000*3 + 9,000 * 4 + 5,000 * 4
= 224,000

100,000 * 3 = 300,000

(Images from CLRS.)

Building the Huffman Tree

1. Make 'leaves' with letters and their frequency and arrange them in **increasing** order of frequency.
2. Repeat until there is only one tree:
    1. Create a new tree from the two leftmost trees (with the smallest frequencies) and
    2. Put it in its place (in increasing order of frequency).
3. Label left/right branches with 0/1

Encoding of char = path from root to leaf that has that char

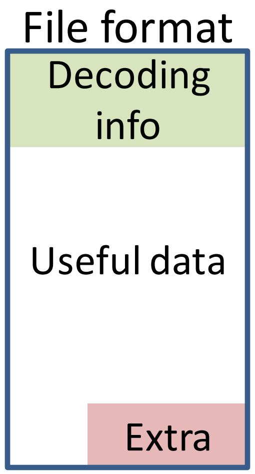
Tree property: Every internal node has two children

See book or lecture video for the step-by-step solution.
In exam or hw, you must use this method. Make sure that you always put the smallest child node to the left.

	a	b	c	d	e	f
Frequency (thousands)	45	13	12	16	9	5

Glancing at implementation issues and options

- Good reference: https://www2.cs.duke.edu/csed/poop/huff/info/
- File Compression with Huffman coding - Practical Issues
  - Given an encoded file, how will you know when it ended?
    - Typically files are stored by the OS in sizes that are multiples of a specific value, so even though your actual file takes 121 bits, on disk 128 bits are written and all that will be read when you open the file. You need to know that after 121 you are done.
    - a) encode length of useful data in the file or
    - b) need a new symbol to indicate end of file. Note that you must add this symbol (with count 1) to your set of distinct symbols used in building the Huffman tree, because you must be able to recognize it (decode it from bits into the symbol).
  - Given an encoded file, how will you know how to decode it?
    - The encoding tree is not standard, but it depends on the given file =>
    - Must record some information to know how to decode the file along with the file (at the beginning) =>
      - Store the Huffman tree (0- inner node, 1-leaf, after 1 read 8 bits and decode to symbol)
      - Store enough info to regenerate the tree (e.g. char and frequency pairs).



- Build the tree efficiently
  - Using a heap: Heaps are efficient (lgN) for finding min, removing min, inserting new item
    - With heap: O(NlgN)
    - With sort and reinsert (like on paper): $O(N^2)$
  - Using 2 sorted queues (https://en.wikipedia.org/wiki/Huffman_coding)