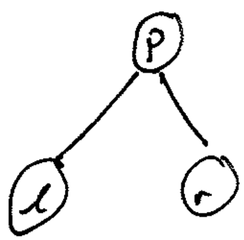
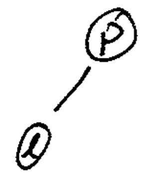
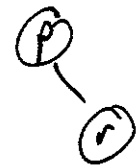
Binary Trees

- A tree is a graph $T = (V, E)$ that is acyclic (contains no cycles)
  - between any 2 vertices there is exactly $\underline{1}$ paths
- binary tree: oriented, parent, left, right child

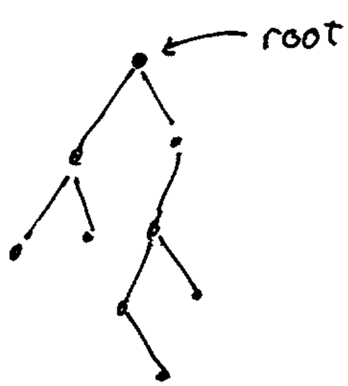


no children: leaf



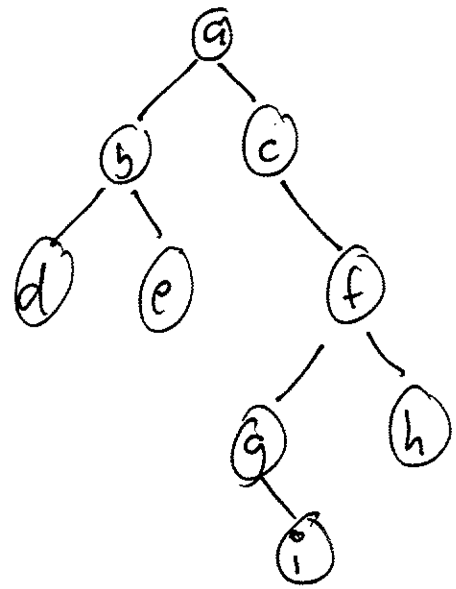


- a vertex with no parent



root to any node: 1 path

~~len~~ length of that path $r \rightsquigarrow u$ is the <u>depth</u> of u



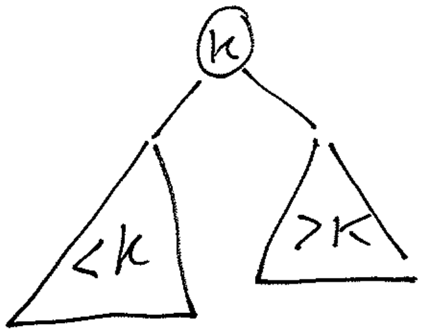
node	depth	height
f	2	2
c	1	3
a	0	4
i	4	0

Depth of T : max. depth of any node

Depth : 4

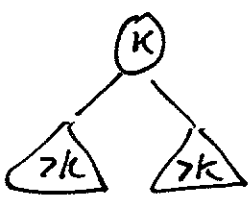
Height of a node u is the longest path to
any of its descendent leaves

Binary Search Tree:

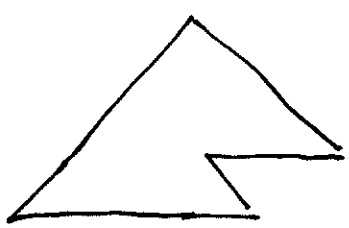


Heap: (min heap)

- Binary tree
- k in a node u is less than <u>both</u> its children



- Full: every node is present except the last level, full to the left



Coding Theory:

- encoding symbols (in binary)
- ASCII: fixed length encoding all codewords have length 8

$$A = 65 = \underbrace{01000001}_{8\text{bits}}$$

- Variable length encoding:
  - common letters: shorter code words
  - rare letters: longer code words

  reduces the average code word length

  - omit some characters

$$\begin{array}{cccc} A & B & C & D \\ \{0, & 01, & 101, & 010\} \end{array}$$ variable length code

message: 010 $\longrightarrow$ D ambiguous

$\longrightarrow$ BA

Goal: a prefix free code: no code word is the prefix of another

A is a prefix of B 0 vs 01

B is a prefix D

Idea:

- Want to encode a file using a variable length, prefix free code that minimizes the average code word length (compression) lossless

- Huffman Coding

  - high frequency (common) letters: small code words
  - low frequency (rare) letters: longer code words

- Given ~~a~~ frequencies, build a tree:

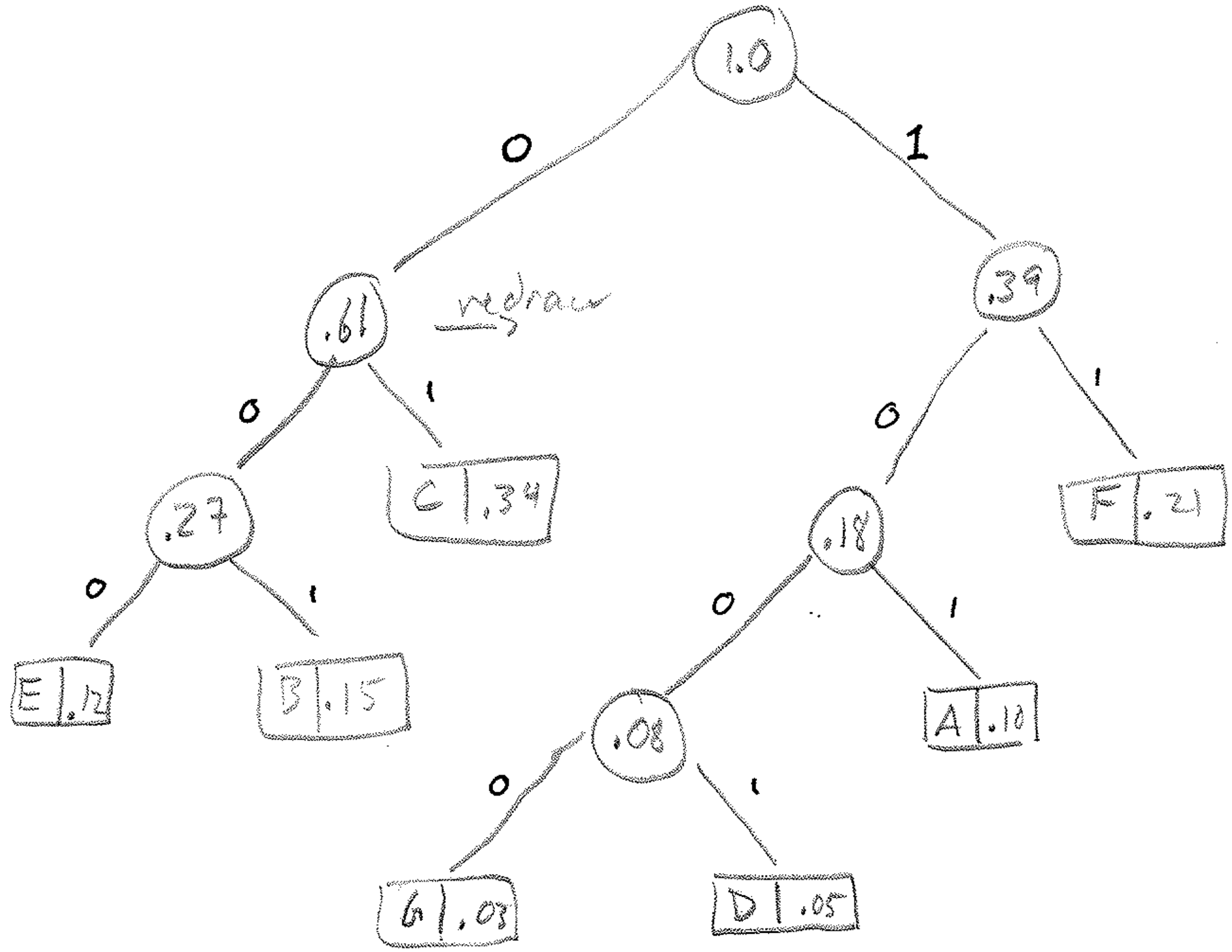
  combine "small" frequencies as children to a new root node → new tree

- continue until 1 tree remains

- path root → leaf defines a code word.

freq

~~A~~	~~.10~~
~~B~~	~~.15~~
C.	.34
~~D~~	~~.05~~
~~E~~	~~.12~~
~~F~~	~~.21~~
~~G~~	~~.03~~



Fixed length encoding: 3

000
001
010
011
100
101
110
~~111~~

Symbol	codeword	length + freq	
A	101	3	.10
B	001	3	.15
C	01	2	.34
D	1001	4	.05
E	000	3	.12
F	11	2	.21
G	1000	4	.03

$$\frac{3 - 2.53}{3} = 15.67\%$$

(compression ratio)

Average code word length

pack

$.3 + .45 + .68 + .2 + .36 + .42 + .12$

$= 2.53$

Zip ≠ Huffman

Input: Alphabet of symbols, Σ

frequency of symbols $freq : \Sigma \to \mathbb{R}^+$ $\quad freq(x) =$ frequency of the symbol x in the file

Output: A Huffman Tree

$H \leftarrow$ min Heap

//initialization

for each $x \in \Sigma$

- $T_x \leftarrow$ single node tree
- $wt(T_x) \leftarrow freq(x)$
- $H.insert(T_x)$

while $H.size > 1$ $\quad (n-1)$

- $T_p \leftarrow$ new root
- $T_\ell \leftarrow H.getMin$
- $T_r \leftarrow H.getMin$
- $T_p.leftChild \leftarrow T_\ell$
- $T_p.rightChild \leftarrow T_r$
- $wt(T_p) \leftarrow wt(T_\ell) + wt(T_r)$
- $H.insert(T_p)$

output $H.getMin$

getMin, insert:

$O(d) = O(\log(n))$

$n =$ number of things in the heap

$$\log(a) + \log(b)$$
$$= \log(a \cdot b)$$

total cost:

$$\sum_{k=2}^{n} \log(k) = \log(2) + \log(3) + \log(4) + \cdots \log(n)$$
$$= \log(2 \cdot 3 \cdot 4 \cdot \cdots n)$$
$$= \log(n!)$$
$$\leq \log(n^n)$$
$$= n\log(n)$$

$$\Theta(n\log(n))$$

iteration	size of H	getMin/insert cost
1	n	$\log(n)$
2	$n-1$	$\log(n-1)$
3	$n-2$	$\log(n-2)$
$\vdots$		
i	$n-i(+1)$	$\log(n-i)$
$\vdots$		
~~n-1~~ $n-1$	2	$\log(2)$