Date: 02/05/21

Black Board

Design and Analysis of Algorithms

Topics:

- Greedy Algorithms II
  - Huffman Encoding ✓

Fixed Length Encoding

ASCII

A — 65 → 01000001

B — 66 ...

8-bit

All codes have the same length.

File Compression using fixed length encoding

- Example: a file with 6 unique characters

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

2^8

100,000 characters ⟹ 800,000 bits.

aacdeb...

000000101...

2 × 2 × 2 = 8

Compression Ratio

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

Compression Ratio : $r = 1 - \frac{\text{Size of compressed file}}{\text{Size of original file}}$

A G C T

$= 1 - \frac{300{,}000 \text{ bits}}{800{,}000 \text{ bits}}$

$= 1 - \frac{3}{8} = 0.625$

62.5%

Variable Length Encoding

	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

Compression Ratio:

$$r = 1 - \frac{1011 \text{ bit}}{800{,}000 \text{ bit}} \times 100\% = 1 - \frac{(45.1 + 13.3 + 12.3 + 16.3 + 9.4 + 5.4)}{800{,}000}$$

$$= 1 - \frac{224{,}000}{800{,}000} = 72\%$$

a → 0, b → 001

abb...

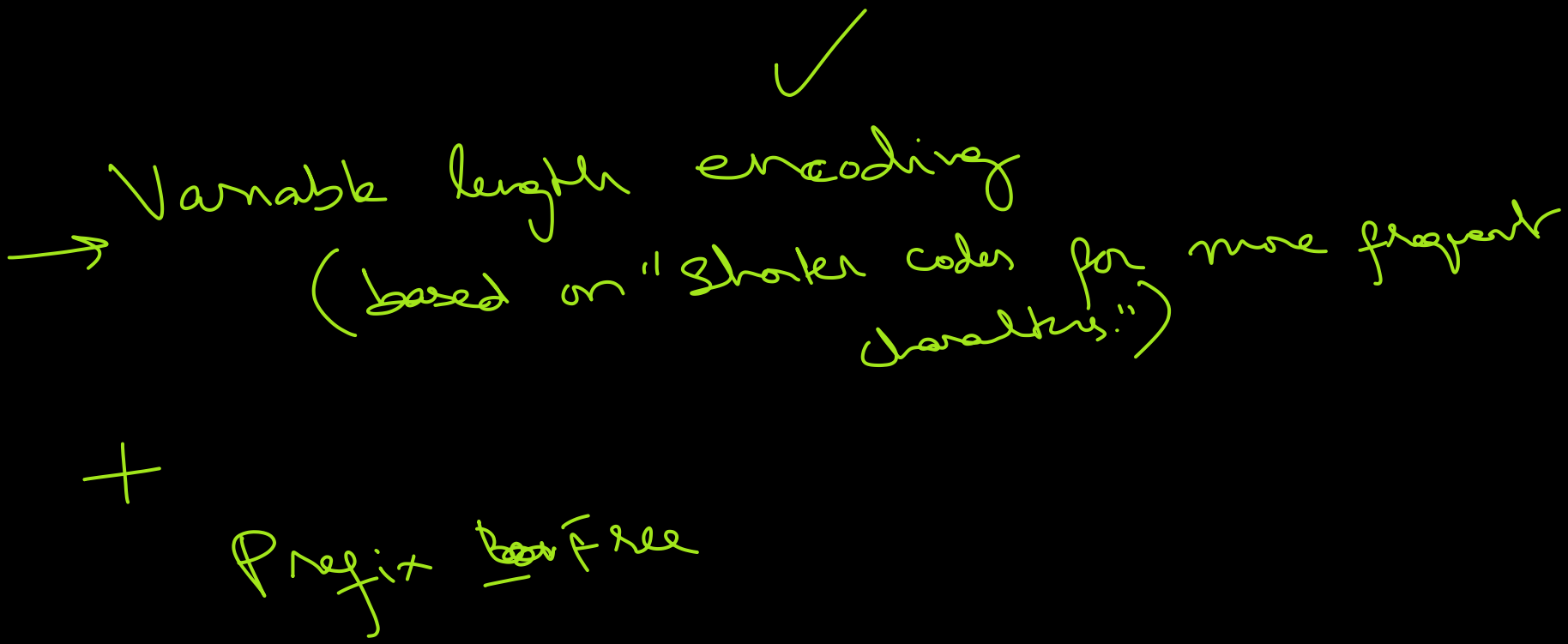
000|001|....

a=0, c=100

b=001

Consider 00100 → aac (bac)

Encoding must be **Prefix-free** for unambiguous decoding



The Huffman Encoding Algorithm

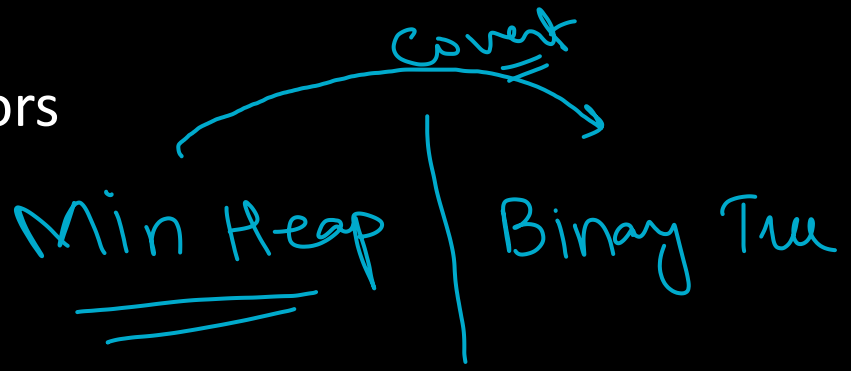
- Finds a prefix-free, variable length encoding for an input file.
- The encoding is optimal, i.e. achieves the highest compression ratio over all prefix-free, variable length encodings
  - (Proof of optimality uses induction on the file size)

Iterations of the Huffman Algorithm

Step 1: Compute a frequency table and create "PQT entries" for each unique original character.

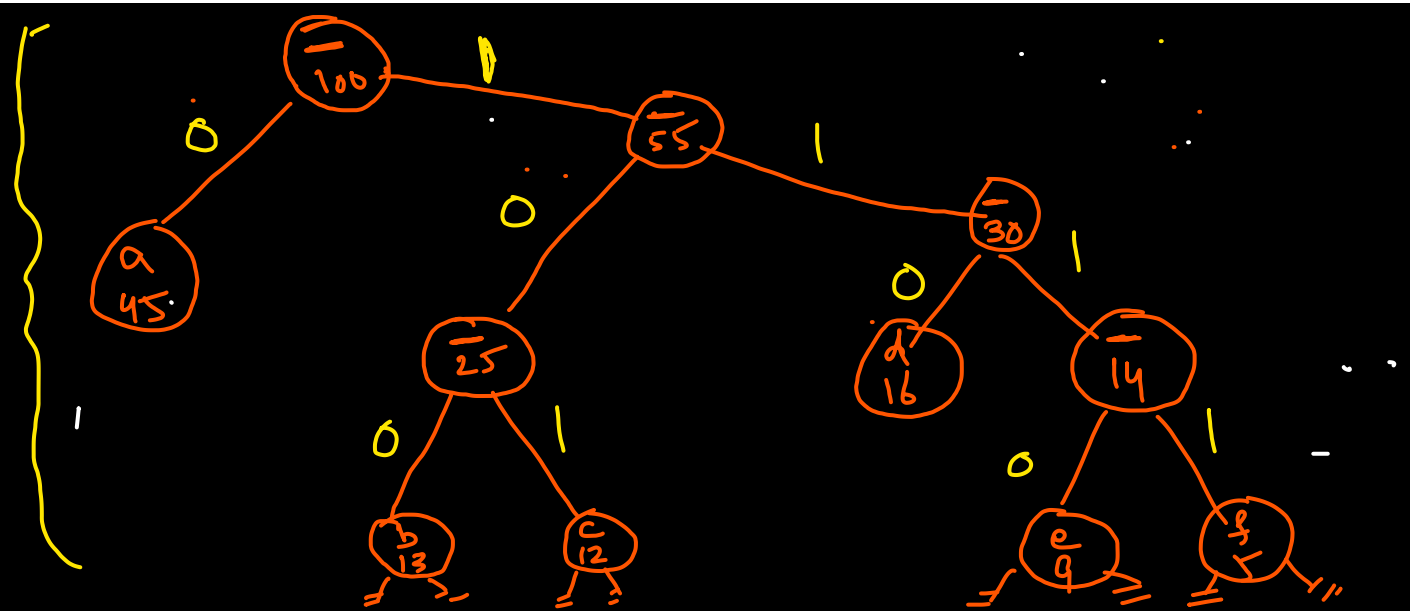
Each PQT or "Priority Queue" is a **struct type tree and priority queue node** containing:

- **key** (used to maintain the min-heap)
- **Sym**: (symbol of the character, either original or -)
- **lchild**, **rchild** pointers
- Necessary comparison operators



- Step 2: Build Priority Queue of the PQTs.

	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



- Step 3: Build Compression Tree from the Priority Queue in n iterations, where n is the number of unique symbols in the file.

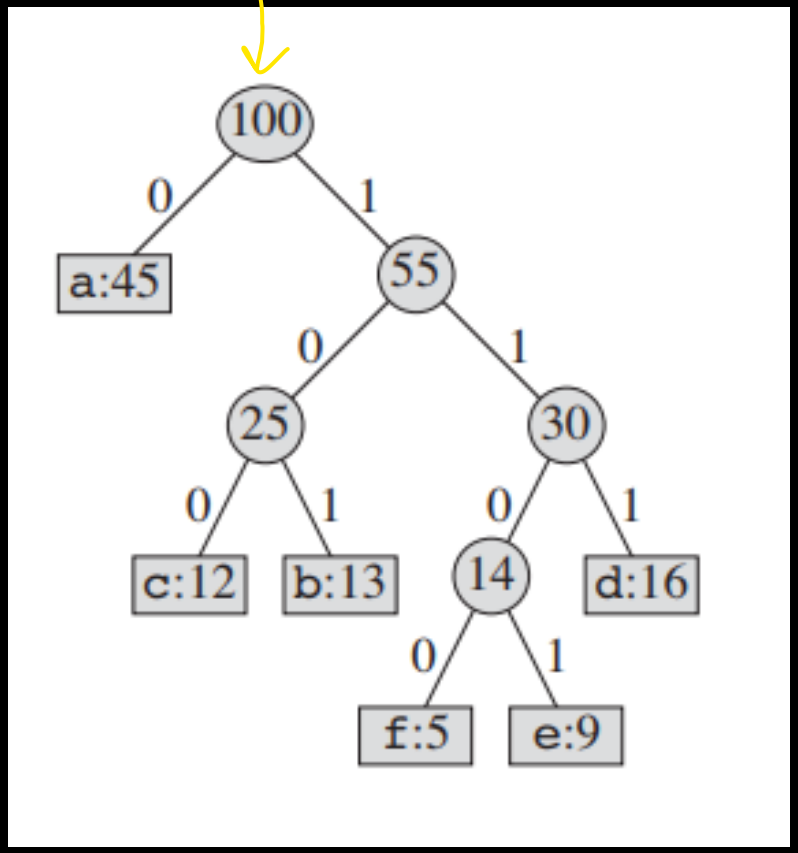
→ getMin() , getMin()
extractMin() , extractMin()

→ Make a parent node with accumulative frequency.

→ Insert parent node into the heap.

- Step 4: Compress the file using the compression tree: compression table, bit level packing, etc.

Decoding



0 0 1 0 0 1 1 0 0 ...

a a c f .....

n → size of file

N → # of unique symbols

T → tree

C(i) = length of the code of char i

What is the running time of Decompression Program.

Compression