Unit IV **11L**

Text Processing: Sting Operations, Brute-Force Pattern Matching, The Boyer- Moore Algorithm, The Knuth-Morris-Pratt Algorithm, Standard Tries, Compressed Tries, Suffix Tries, The Huffman Coding Algorithm, The Longest Common Subsequence Problem (LCS), Applying Dynamic Programming to the LCS Problem.

Unit IV

Text Processing: Sting Operations, Brute-Force Pattern Matching, The Boyer- Moore Algorithm, The Knuth-Morris-Pratt Algorithm, Standard Tries, Compressed Tries, Suffix Tries, The Huffman Coding Algorithm, The Longest Common Subsequence Problem (LCS), Applying Dynamic Programming to the LCS Problem.

ADVISED TO WATCH ALL VIDEOS IN ATLEAST 1.5x SPEED

String Intro

https://www.youtube.com/watch?v=JvKMLSa0Rq0

String Operations

Operation	Description
Concatenation	The Addition operator, “+”, can be used to concatenate strings together.
Formatting Data	The STRING function is used to format data into a string. The READS procedure can be used to read values from a string into IDL variables.
Case Folding	The ToLower() method returns a copy of the string converted to lowercase. Similarly, the ToUpper() method returns a copy converted to uppercase. The CapWords() method returns a copy where the first letter of each word is capitalized. See also the STRLOWCASE and STRUPCASE functions.
White Space Removal	The Compress() and Trim() methods can be used to eliminate unwanted white space. See also the STRCOMPRESS and STRTRIM functions.
Length	The Strlen() method (or STRLEN function) returns the length of the string.
Substrings	See the CharAt(), Extract(), IndexOf(), Insert(), LastIndexOf(), Remove(), Replace(), Reverse() , and Substring() methods. See also the STRPOS, STRPUT, and STRMID routines.
Splitting and Joining	The Split() method is used to break strings apart, and the Join() method can be used to join them. See also the STRSPLIT and STRJOIN functions.
Comparing Strings	The "EQ" operator can be used to directly compare two strings. The Contains(), EndsWith(), Matches(), and StartsWith() methods can be used to compare portions of strings. See also the STRCMP, STRMATCH, and STREGEX functions.

https://www.youtube.com/watch?v=-pSyzCWsBA8

String/Text Processing

Insert- https://www.youtube.com/watch?v=Rj1Tcb5RFzU

Delete- https://www.youtube.com/watch?v=SYeMm5HiJQ0

Replace- https://www.youtube.com/watch?v=uLZUfgqh-DQ

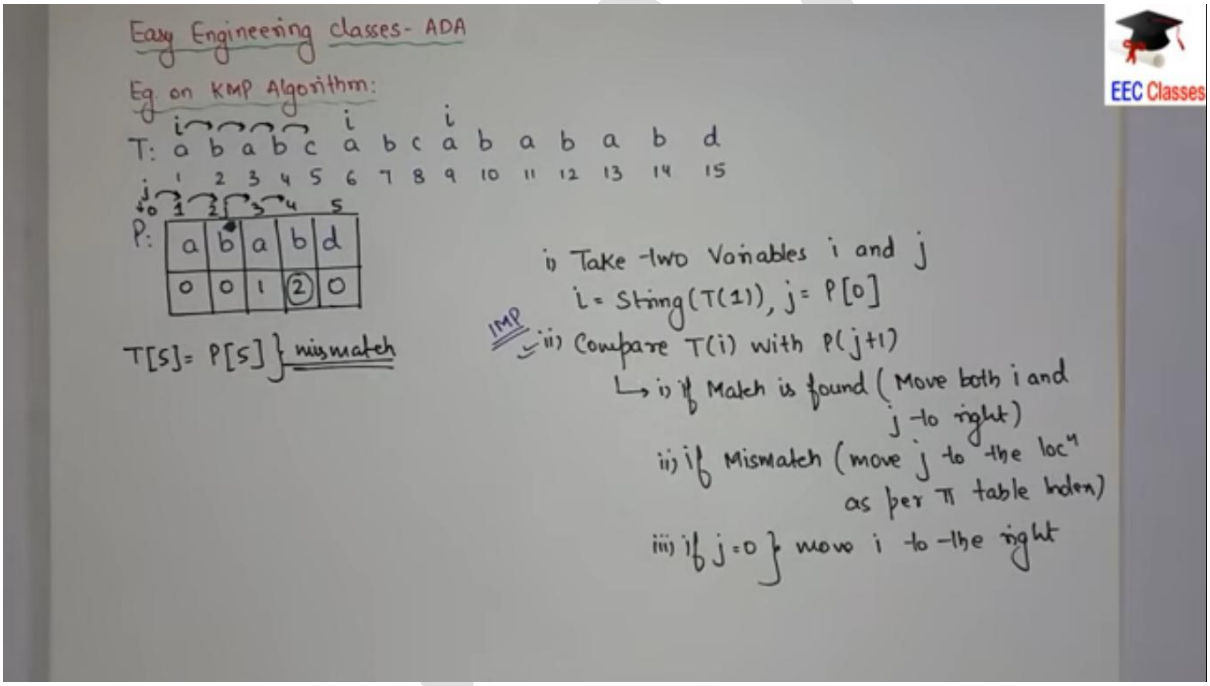
Boyer Moore : check the pdf that I sent u/ attached below.

[Use only last function is there not bad match table (according to sir) so, don't waste your time watching YouTube videos on this, coz max are of bad match table.]

https://www.youtube.com/watch?v=4Oj_ESzSNCk

Brute-Force Pattern Matching: https://www.youtube.com/watch?v=yMJLpdKV0BQ

Knuth-Morris-Pratt Algorithm:



https://www.youtube.com/watch?v=V5-7GzOfADQ [better]

https://www.youtube.com/watch?v=H3HFstXmnP8 [if less time, then see this]

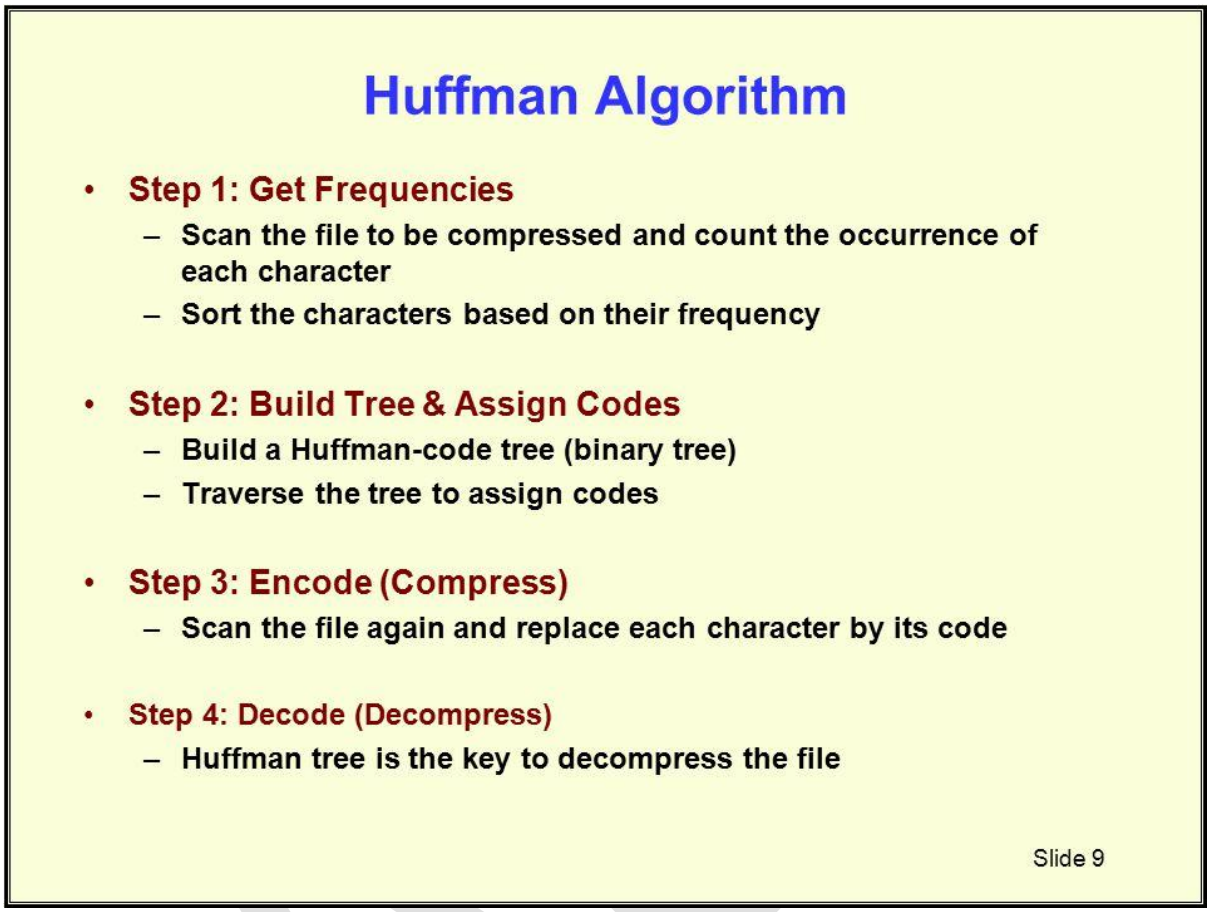
Failure function visualization: https://www.educative.io/edpresso/what-is-the-knuth-morris-pratt-algorithm

Standard Tries, Compressed Tries, Suffix Tries: TRIES PPT {UPLOADED IN DRIVE} [prefer this]

https://drive.google.com/file/d/1saZrFjk5mWUdw9q0PS9apCina4DX6Yct/view?usp=sharing

https://www.geeksforgeeks.org/types-of-tries/#:~:text=A%20Compressed%20Trie%20is%20an,of%20redundant%20nodes%20is%20performed. [2nd resource]

Huffman coding:



https://www.youtube.com/watch?v=co4_ahEDCho&t=158s [For understanding]

https://www.programiz.com/dsa/huffman-coding [Algorithm]

The Longest Common Subsequence Problem (LCS):

```
LONGEST-COMMON-SUBSEQUENCE(x, m, y, n)
 1  for i ← −1 to m − 1
 2       do T[i, −1] ← 0
 3  for j ← −1 to n − 1
 4       do T[−1, j] ← 0
 5  for i ← 0 to m − 1
 6       do for j ← 0 to n − 1
 7             do if x_i = y_j
 8                   then T[i, j] ← T[i − 1, j − 1] + 1
 9                   else T[i, j] ← max(T[i, j − 1],
                                        T[i − 1, j])
10  return T
```

https://www.youtube.com/watch?v=sSno9rV8Rhg

Applying Dynamic Programming to the LCS Problem.

11/2/21:
Thursday

The Boyer Moore Algorithm:

Text: a b ac a a b a dc a b a c a b aa bb

pattern: a b ac a b

Maximum index of 'a' in pattern: 4
" " " 'b' " " : 5
" " " 'c' " " : 3
" " " 'd' " " : 0

If 'd' is encountered in text, the entire pattern until 'd' is skipped.

a b ac a a b a dc a b a c a b aa bb
→ a b ac a b
⟶ a ba c a b
⟶ a b a c a b
⟶ a b a c a b
a b a c a b

Since 'd' is not present in pattern, the pattern is not compared with 'd'. And hence we skip '6' positions.

⟶ a b a c a b
13 12 11 10 9 8

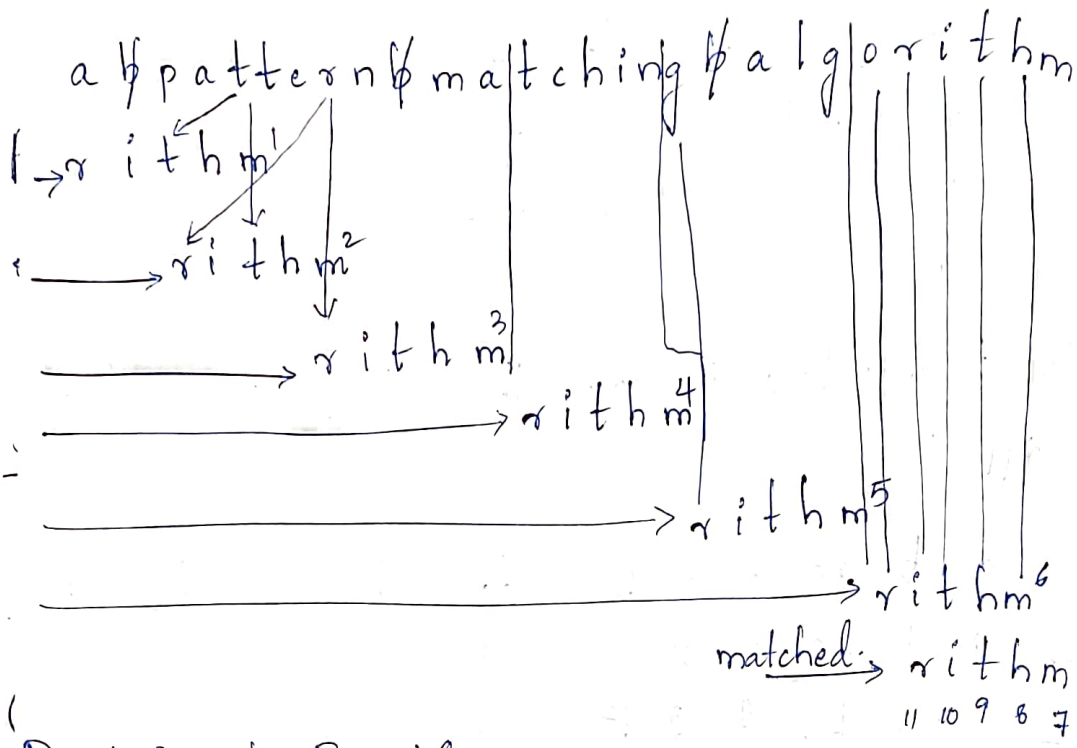
ϕ → blank space

Text: a ϕ pattern ϕ matching ϕ algorithm

pattern: rithm

a ϕ pattern ϕ matching ϕ algorithm

char	a	p	t	e	r	n	m	i	h	g	l	o
last(char)												



Drawback with Boyer Moore

If no. of symbols are more; and pattern length is less then less non-zero values. Less skips.

Text = 0 1 2 3 4 5 6
a a a a a a a

P = 0 1 2 3 4
b a a a a
b a a a a
b a a a a

Postfix matches and prefix unmatches.
And so the shift occurs only by one position.

Compute the last function.

$i \leftarrow m-1$
$j \leftarrow m-1$
repeat
if $P(j) == T(i)$ then

```
        if (j == 0) return i
        else
            i ← i - 1
            j ← j - 1
    else
        i ← i + m - min(j, 1 + last(T(i)))
        j ← m - 1
until i > n - 1
return 'pattern not in text'
```

Boyer Moore for Σ {-, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z}

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v
30	9	7	33	24	14	35	27	6	17	8	29	19	13	34	20	4	10	21	26	18	23

w	x	y	z	-
12	16	32	31	28

String: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
The quick brown fox jumps over the lazy dog
33 34 35