Unit-6 :-

String Matching :-

1. Naive string matching

Ex: Text T = AABAACAADAABAABAA
pattern P = AABA The pattern is found at index using Naive string matching.

sol: Text T = A A B A A C A A D A A B A A B A A
pattern P = A A B A pattern is found at index 1.

step 2:
i
A A B A A C A A D A A B A A B A A
x
j A A B A pattern is not matching.
i

step 3:
A A B A A C A A D A A B A A B A A
x
j A B A A Not matched.

Then shift one step to right side.

i
stepu: A A B A A C A A D A A B A A B A A
x x
j A A B A Not matching, then shift into one step right.

steps:
A A B A A C A A D A A B A A B A A
A A B A Not matching, then shift into one step right

step 6:
A A B A A C A A D A A B A A B A A
x
A A B A Not matching.

step 7:
A A B A A C A A D A A B A A B A A
x
A A B A Not matching. then shift to right.

step 8:
A A B A A C A D A A B A A B A A
x
A A B A Not matching.

step 9:
A A B A A C A D A A B A A B A A
x
A A B A Not maching.

Step 10:- A A B A A C A A D A A B A A

pattern is found at index 9

Step 11:- A A B A A C A A D A A B A A

Not matching.

Step 12:- A A B A A C A A D A A B A A

Not matching.

Step 13:- A A B A A C A A D A A B A A

pattern is found at Index 13.

Step 14:- A A B A A C A A D A A B A A

Not matching

the window is shifting to one step to right there is no text.

→ pattern is found at index 1, 10, 13.

Algorithm:-

```
Algorithm Naive ()  ——0
{                   ——0
n ← length (text)   ——1
m ← length (pattern) —1
for s ← 1 to n-m do ——0 (n-m+1)
do P[1....m] = T[s+1.....s+m] —1
Then pattern "pattern occurs with shift" —1
}  —0
```

O(n-m+1)

Best case:-

Text : A A B A A C A A

pattern : F A A — O(n)

worst case

Text: A A A A A A A A → no. of windows

pattern:- A A A A

O(m × (n-m+1))

↳ length of pattern window.

Text: a c a a b c

pattern: aab

Find the pattern in a given text using Naive algorithm.

* Tries: Also called as Radix trie, Digital trie, prefix trie.

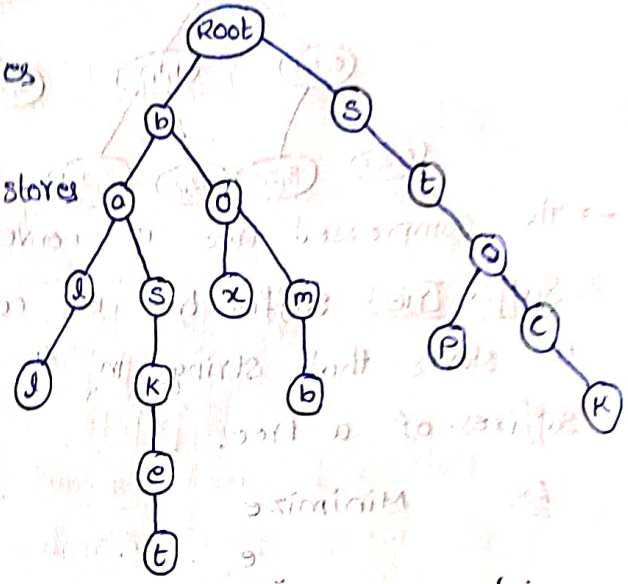
→ There are three types of tries. → Information Retrieval data structure.

1. Standard Trie :-

string s = {ball, box, bomb, basket, stock, stop}
0 1 2 3 4 5

→ Standard trie is a tree which stores the strings.

→ Each and every node in a tree stores the letter from the given string.

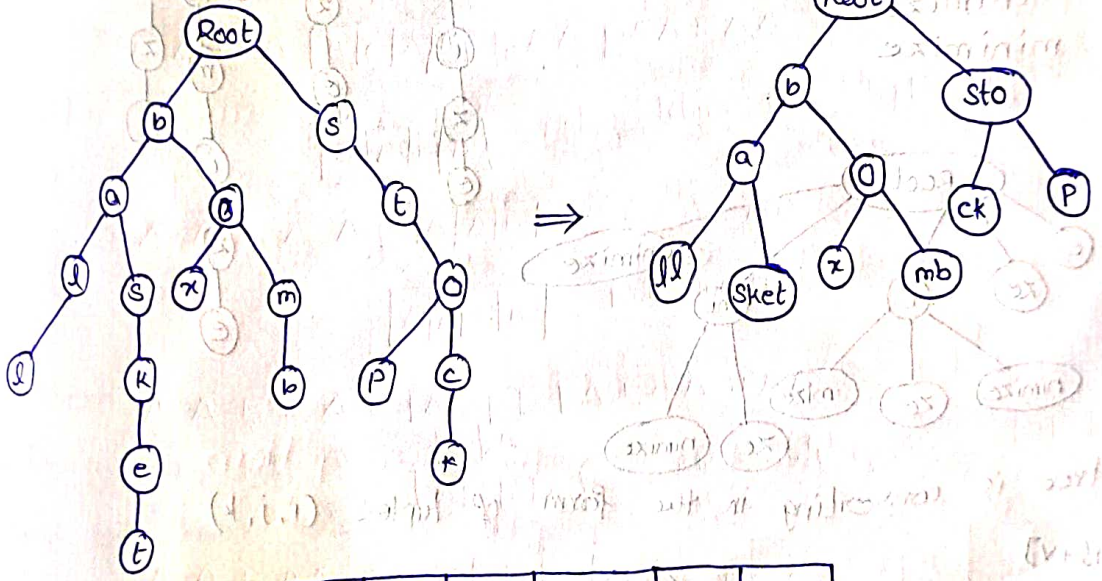


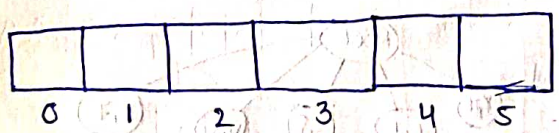
2. Compressed trie :-

→ Advanced version of standard trie which stores the strings in the form of tree structure.

Each and every node stores the letters from the given string.

→ So we are merging the leaf nodes which contains one node or group of nodes.





s(0) = b a l l

s(1) = b o x

s(2) = b o m b

s(3) = b a s k e t

s(4) = s t o c k

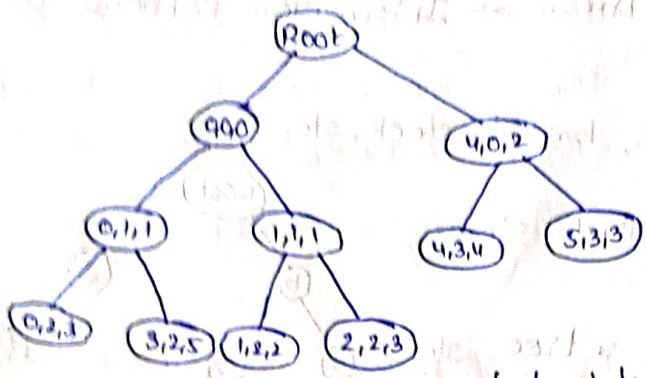
s(5) = s t o p

Tuple (i, j, k)

i is the string present in the index.

j is the position which is starting at index. [ctrl+c]

k is the prefix which is ending at index.

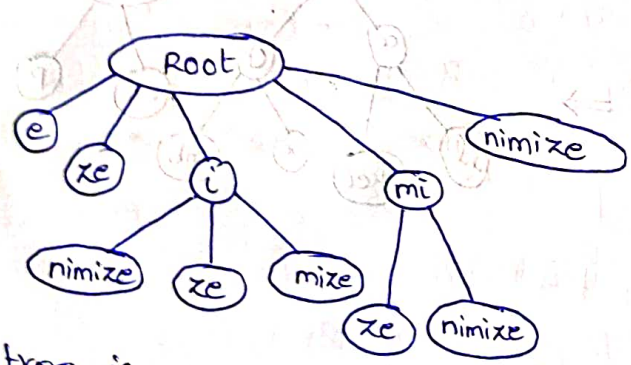


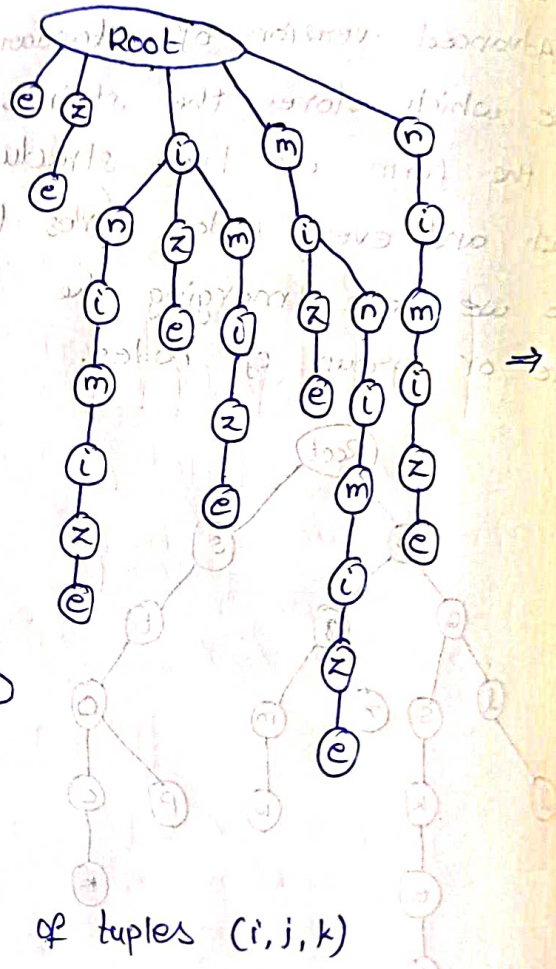
→ The compressed tree is converted into the in form of tuple (i,j,k)

3. Suffix Trie: Suffix trie is compressed tree which can be used to stores that strings in the forom of tree stultures by using suffixes of a tree.

Ex: Minimize

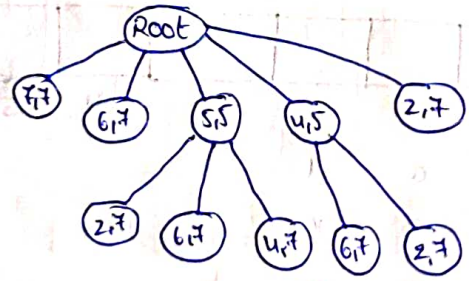
e
ze
ize
mize
imize
nimize
inimize
minimize





→ The tree is converting in the form of tuples (i, j, k)

[ctrl+v]



* Rabin - Karp Algorithm [1987] :-

→ Rabin - Karp algorithm is finding or matching in the given text by using hash value.

→ pattern characters are converted into hash values that is matching with text window hash value, both the hash values are same then only we can check the characters.

Ex: Text T = A B C C D D A E F G

pattern P = C C D

n is the size of pattern

m is the size of text

n = 3 m = 10, d = 10

d no. of character in a given text

A - 1
B - 2
C - 3
D - 4
E - 5
F - 6
G - 7
H - 8
I - 9
J - 10

→ Find the hash value of pattern

$$h(P) = \sum (v \times d^{n-1}) \bmod 13$$

$$= (3\times10^{3-1} + 3\times10^{3-2} + 4\times10^{3-3}) \bmod 13$$

$$= (3\times10^{2} + 3\times10 + 4) \bmod 13$$

$$= 334 \bmod 13$$

$$= 9$$

→ Find the hash value of Text window

hash (Text) = A B C

$$\text{hash}(T) = (1\times10^{3-1} + 2\times10^{3-2} + 3\times10^{3-3}) \bmod 13$$

$$= (10^{2} + 2\times10 + 3) \bmod 13$$

$$= (100 + 20 + 3) \bmod 13$$

$$= (129) \bmod 13$$

$$= 6$$

pattern hash value and text window hash value both are not matching so shift the text one step right.

→ Find the hash value of text window

hash (text) = B C C

$$= (2\times10^{3-1} + 3\times10^{3-2} + 3\times10^{3-3}) \bmod 13$$

$$= (2\times10^{2} + 3\times10 + 3) \bmod 13$$

$$= (200 + 30 + 3) \bmod 13$$

$$= (233) \bmod 13$$

$$= (233-1) \bmod 13$$

232 mod 13 = 12

13) 232 (17
13
102
91
11

pattern hash value and text window value not matching then text window is moving one step to right side.

→ Text T = CCD

pattern p = CCD

Find the hash value of text window "CCD"

$$\text{hash}(T) = (3\times10^{3-1} + 3\times10^{3-2} + 4\times10^{3-3}) \bmod 13$$

$$= (3\times10^{2} + 3\times10 + 4) \bmod 13$$

$$= (334) \bmod 13$$

$$= 9$$

∴ pattern = 9

pattern hash value and text window hash value are same then compare characters.

Text T = A B C C D D A E F G

C C D

pattern is matching at index 3

* Knauth-Morris pratt Algorithm [KMP]

prefix table, called as pi table (π)

→ prefix is same as suffix.

Text: a b a b c a b a

prefix: a, ab, aba, abab....

suffix: a, ba, aba, abac,...

	1	2	3	4	5	6	7	4
	a	b	a	b	c	a	b	a
0	0	1	2	0	1	2	1	

→ Matching the pattern in a given text by using prefix table.

→ Main idea of KMP algorithm is reducing the no. of comparisons.

is there any prefix values is same as suffix values.

Ex: Text T = a b ab ab d

pattern p = ab ab d

sol: First prepare prefix table for the given pattern

	0	1	2	3	4	5
		a	b	a	b	d
	0	0	0	1	2	0

→ prefix table

* Boyer Moore Algorithm:-

→ Boyer Moore Algorithm is the fastest way to finding or mat the pattern in a string.

→ If the characters are not matching then we can check Bad match table.

→ The bad match table is deciding. How many gems the patte is moving. From current position to the next matching positi depends on values of the bad match table.

Ex:- Text T = THIS IS A TEST

pattern P = TEST Find the pattern using boymoore Algorithm.

sol:- step1:- First create mad match table for the pattern "TEST"

letter	T	E	S	*
value	3	2	1	4

Find the values of each and every letter in the pattern

Value = length of the pattern - index of letter -1

Value (T) = 4-0-1 = 3

Value (E) = 4-1-1 = 2

Value (S) = 4-2-1 = 1

step2:-

T	H	I	S		I	S		A		T	E	S	T

Text

pattern =

T	E	S	T

x Not matching

The letter s is available in bad match table

→ the value of S is =1 means the pattern y jumping one position to the right.

Step3:- Text

T	H	I	S		I	S		A		T	E	S	T

pattern

T	E	S	T

x Not matching

The empty letter is not available in the bad match table then consider "*" symbol. the value of "*" is 4.

step4:- Text:

T	H	I	S		I	S		A		T	E	S	T

pattern:-

T	E	S	T

x Not matching.

→ the letter not available in the bad match table, then consider "*" value = 4. pattern jumping 4 positions to right.

step5:- Text:

T	H	I	S		I	S		A		T	E	S	T

T	E	S	T

x not matching.

→ S is available in bad match ~~tab~~ table. The value of s = 1

(text window letter)

step6:- Text :-

T	H	I	S		I	S		A		T	E	S	T

T	E	S	T

Ex:- Text T :- WELCOME TO MY WORLD

pattern P :- WORLD

W	O	R	L	D	*
4	3	2	1	0	5

step1:- WELCOME TO MY WORLD

WORLD x Not matching

step2:- WELCOME TO MY WORLD

WORLD x Not matching

step3:- WELCOME TO MY WORLD

WORLD x Not matching.

step4:-

WELCOME TO MY WORLD

↓ ↓ ↓ ↓ ↓

WORLD