

Recursion Concepts & Qns ...

Motivation (भाषण) ...



“ Be very careful on
where you are spending
your time today.
This will impact your
future. Use it wisely. ”

#codestorywithMIK



Facebook
Instagram } → code story with MIK

(Twitter) → CS with MIK



Company :- Google

10. Regular Expression Matching

Hard

Topics

Companies

Given an input string `s` and a pattern `p`, implement regular expression matching with support for `'.'` and `'*'` where:

- `'.'` Matches any single character.
- `'*'` Matches zero or more of the preceding element.

The matching should cover the entire input string (not partial).

`'ab*'`

ab
abb
abbb

Example :- $S = 'aa'$, $P = 'a*'$

Output = True

$S = 'ab'$, $P = '.*'$

Output = True

Thought Process :-

① "ab" ↓

"·a" ↓

False.

② "aa" ↓

"·a" ↓

True

⇒ ③ "ab" ↓

"·*b" ↓
Don't take it (Recursion)
True

④ "abb" ↓

"·*b" ↓
Take it
Don't take. X

↓

"b"

↓

"b"

↑

"·*b" ↓

↓

"b"

↑

•*

True.

⑤

"a b"

↑

↓

Don't take

"a * a b"

↑

take *

"b"

"a * a b"

•*

char *

\downarrow
(True)

"b"

"ab"

Fahr

Summary:-



(c) 0 times $\overline{\text{mzu}}$ \rightarrow `pattern.substr(2)`

(•) first check that (•) $\| \langle ch \rangle == \text{input}[0]$

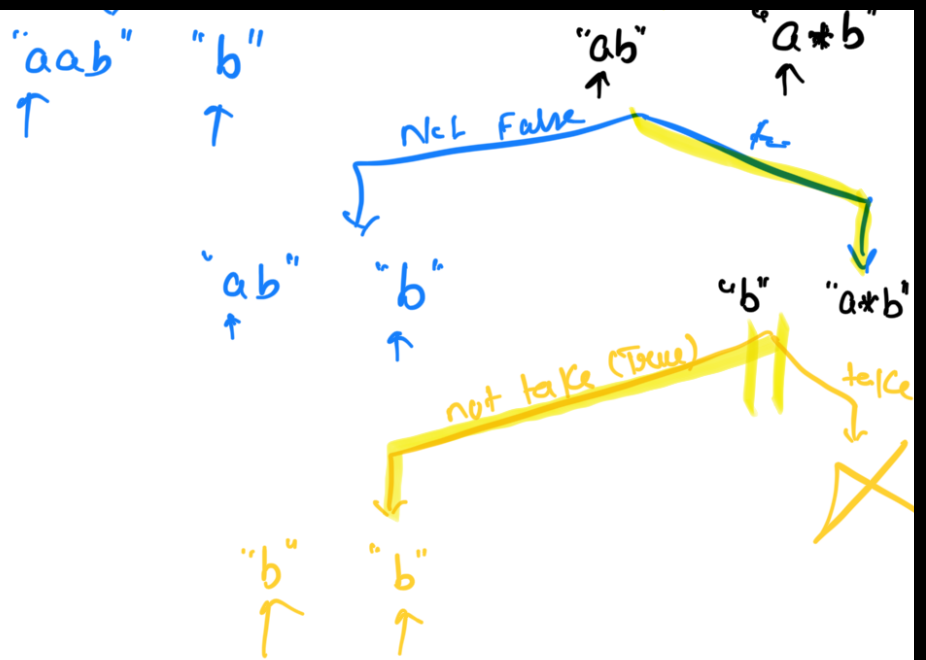
Deep Run:-

Handwritten diagram illustrating the recursive step for finding the longest palindromic substring in "cacaab". The string is enclosed in quotes. Indices 0, 1, 2, and 3 are marked above the characters 'c', 'a', 'c', and 'a' respectively. A yellow circle highlights the first 'c' at index 0. A yellow bracket spans from index 0 to index 3, with the label "S.substr(1)" written below it.

“a * b”

```

graph TD
    Root["take (s.substr(1), pattern)"]
    Root -- "Dn take False" --> Node1["aaab  
↑  
False"]
    Root -- "take" --> Node2["aab  
↑  
(a*) x False"]
    Node2 -- "(a*) x False" --> Node3["aaab  
↑  
False"]
    Node2 -- "take" --> Node4["aaab  
↑  
False"]
    
```



Story Code:



```
bool Solve(s, p) {
```

```
    if (p.length() == 0) {
```

```
        if (s.length() == 0) return True;
```

```
        return False;
```

```
    }
```

```
    bool first-char-matched = p[0] == s[0] || p[0] == '.';
```

```
    if (p[1] == '*') {
```

```
        bool not-take-* = Solve(s, p.substr(2));
```

```
        bool take-* = (p[0] == s[0] || p[0] == '.') &&
```

Solve (S-Subst(1), P) :

} else {

11 $p[0] == '.'$

return (P[0] == s[0] &&

Solve (s-sub(1), p-sub(1))

4

S.C = max depth of tree $\rightarrow O(m)$

$$T.C =$$

$$T(n) = 2T(n-1)$$

↓

$$= 2^n$$



T.C 2^n

Optimisation:-

i
"aba"
↑

j
" . b . "
↑

$s[i] == p[j]$ || $p[j] == '.'$

~~Substr~~



is