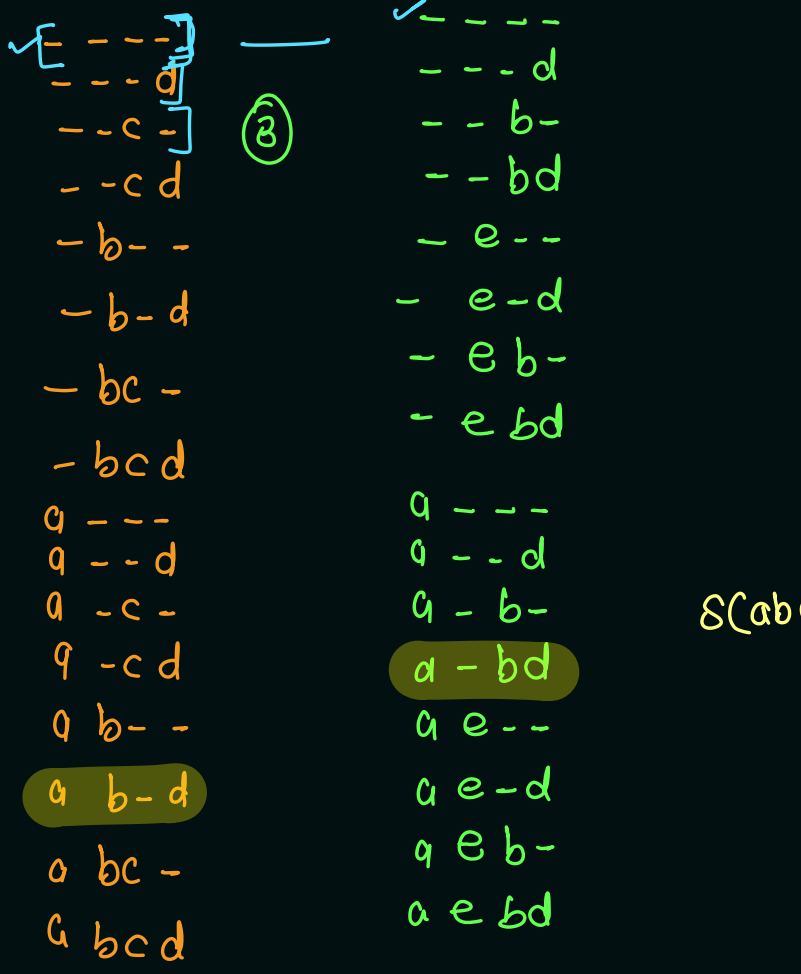
Longest common subseq.:

str1 → abcd → l_1 str2 = aebd → l_2



(3)

complexity of bruteforce :→

generation of subseq. for str1 = 2^{l_1}

" " " str2 = 2^{l_2}

common & longest ⇒

for single subseq = 2^{l_2}

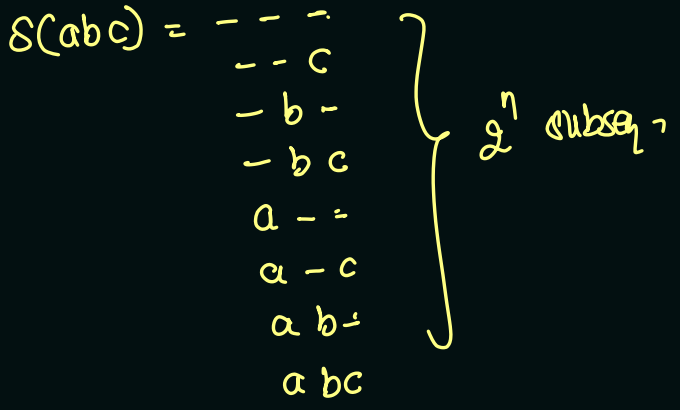
for 2^{l_1} subseq = $2^{l_1} * 2^{l_2} = 2^{l_1+l_2} = 2^n$ (n)

$l_1+l_2=n$

$O(2^n)$

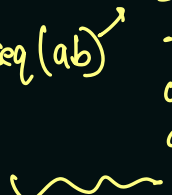
Allowed = $O(n^2)$

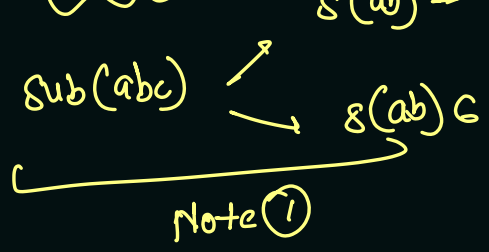
S(abc) =



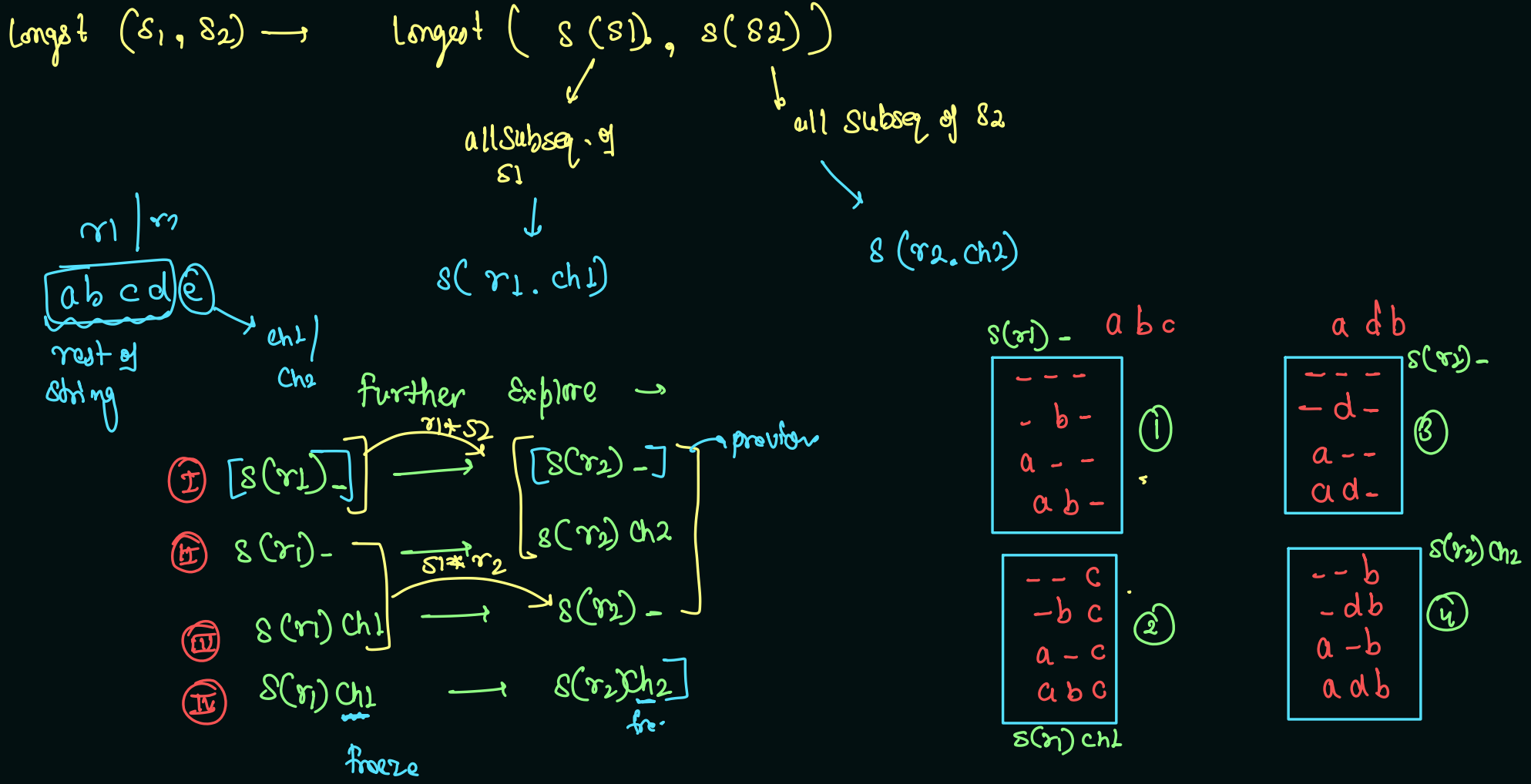
2^n subseq

subseq(ab)





longst (s1, s2) → generate all subseq. & compare and update longst length.



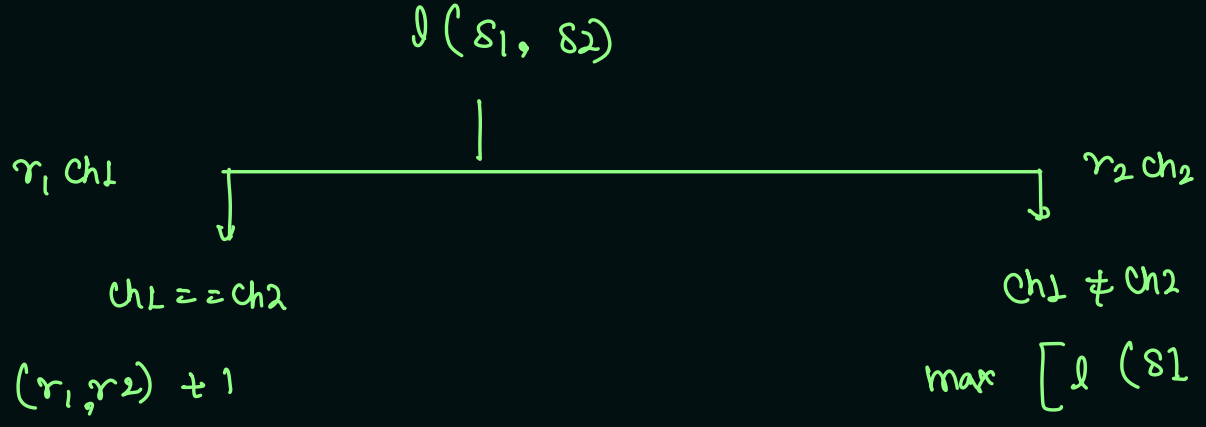
$ch1 == ch2$	$ch1 \neq ch2$
Result → (IV)	Result (I), (II) & (III)
$longst(s_1, s_2) = longst(r_1, r_2) + 1$	$longst(s_1, s_2) = \max(longst(s_1, r_2), longst(r_1, s_2))$

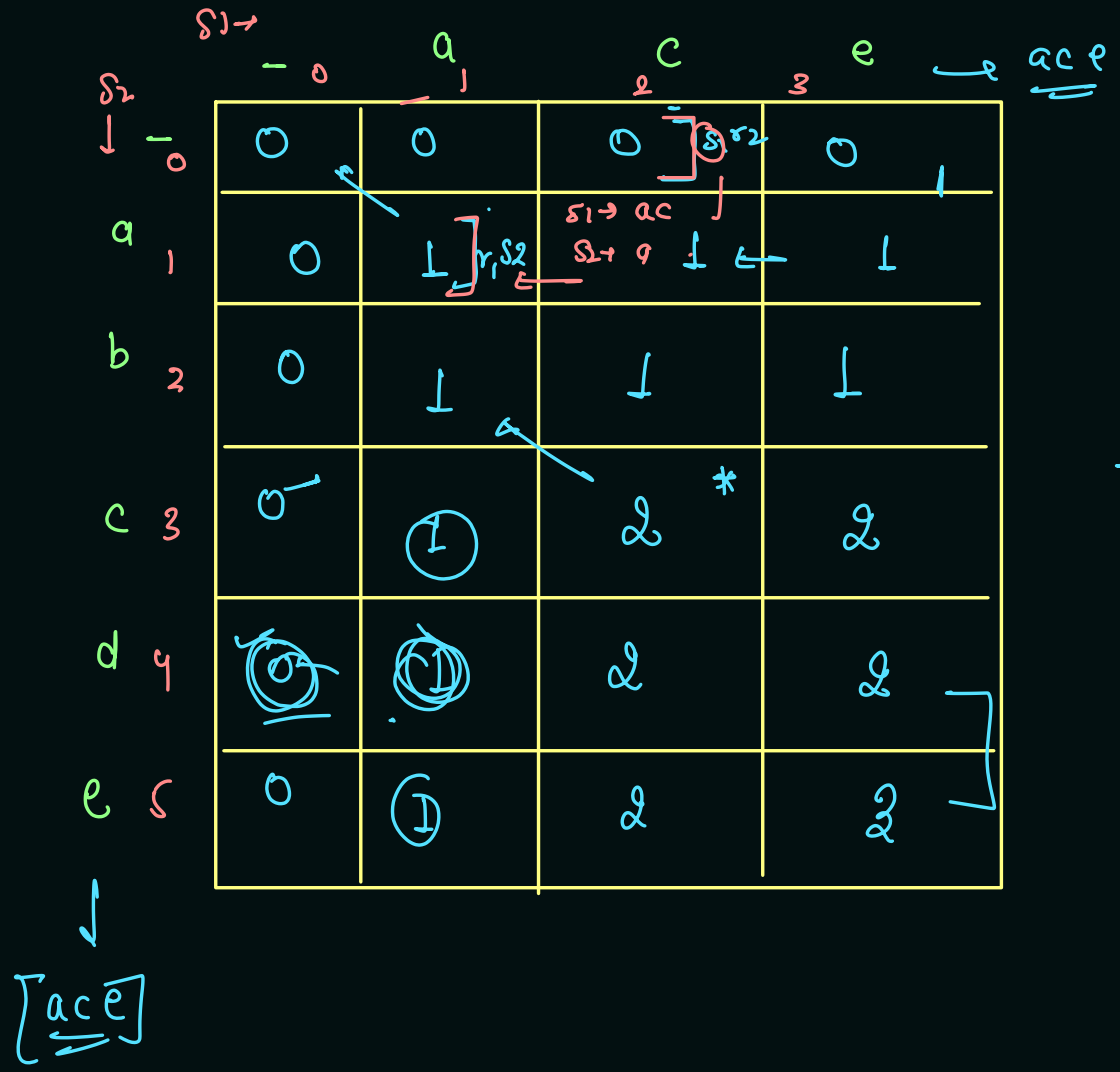
$s_i = r_i\, ch_i$

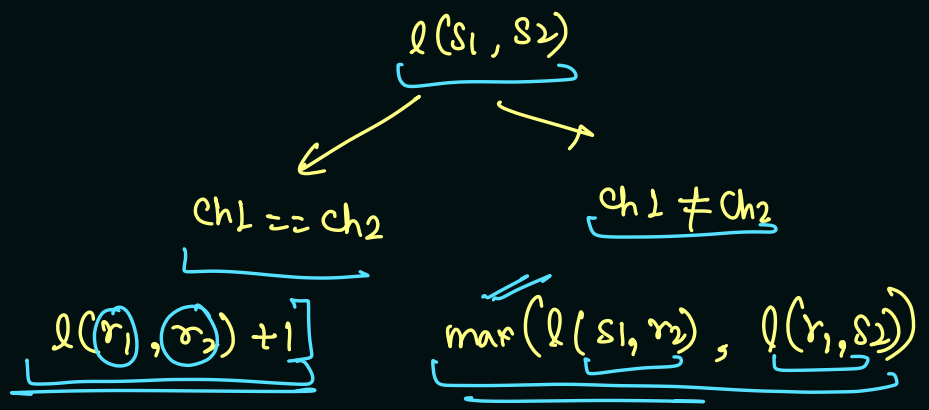
$longst(s_1, s_2)$ → max from (III) & (I) equation

$longst(r_1, s_2)$ → max from (I) & (II)

max → (I), (II), & (IV)







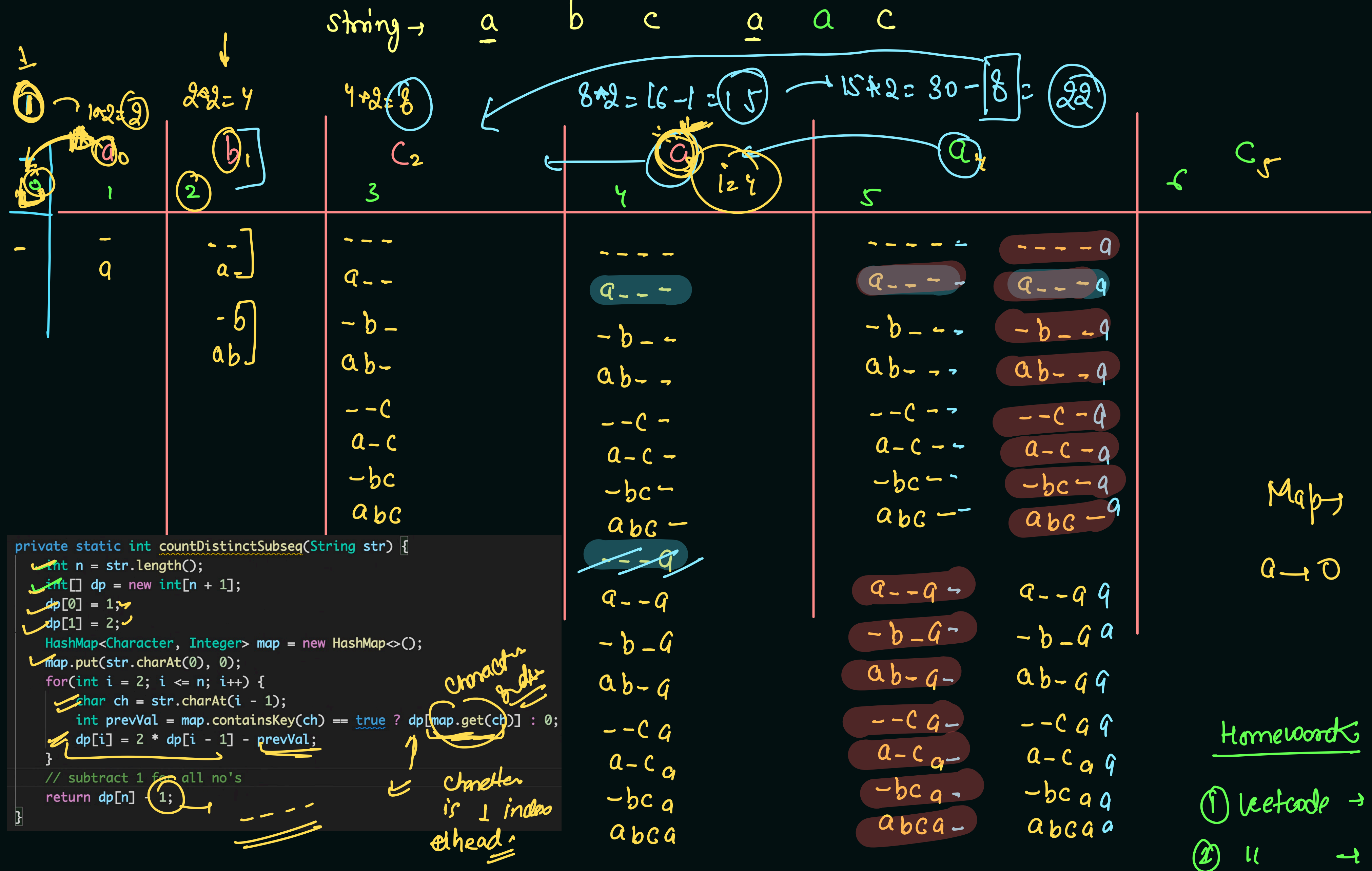
a

time complexity → $O(n^2)$

space complexity → $O(n^2)$ → $O(n)$

↳ 1D- array.

count distinct subseq :→




```java
private static int countDistinctSubseq(String str) {
    int n = str.length();
    int[] dp = new int[n + 1];
    dp[0] = 1;
    dp[1] = 2;
    HashMap<Character, Integer> map = new HashMap<>();
    map.put(str.charAt(0), 0);
    for(int i = 2; i <= n; i++) {
        char ch = str.charAt(i - 1);
        int prevVal = map.containsKey(ch) == true ? dp[map.get(ch)] : 0;
        dp[i] = 2 * dp[i - 1] - prevVal;
    }
    // subtract 1 for all no's
    return dp[n] - 1;
}
```

Homework

1. leetcode → 741
2. " → 403
3. → 64