30-01-2022 | STRING

Subsets :- Non-adjacent collection.
:: It means taking n no. of elements, may or may not be adjacent.

see iterative method in video

Example -

Str = "abc"

ans = ["a", "b", "c", "ab", "ac", "bc", "abc"]

→ we cannot take "ba" or "ca" as we are only taking sets and cannot change orders.

→ "ab" & "ba" are the same thing as no. of characters and type of characters are same.

Here we are removing some elements and taking some elements.

N.V. imp

This pattern of taking some elements & removing some is known as **subset pattern**.

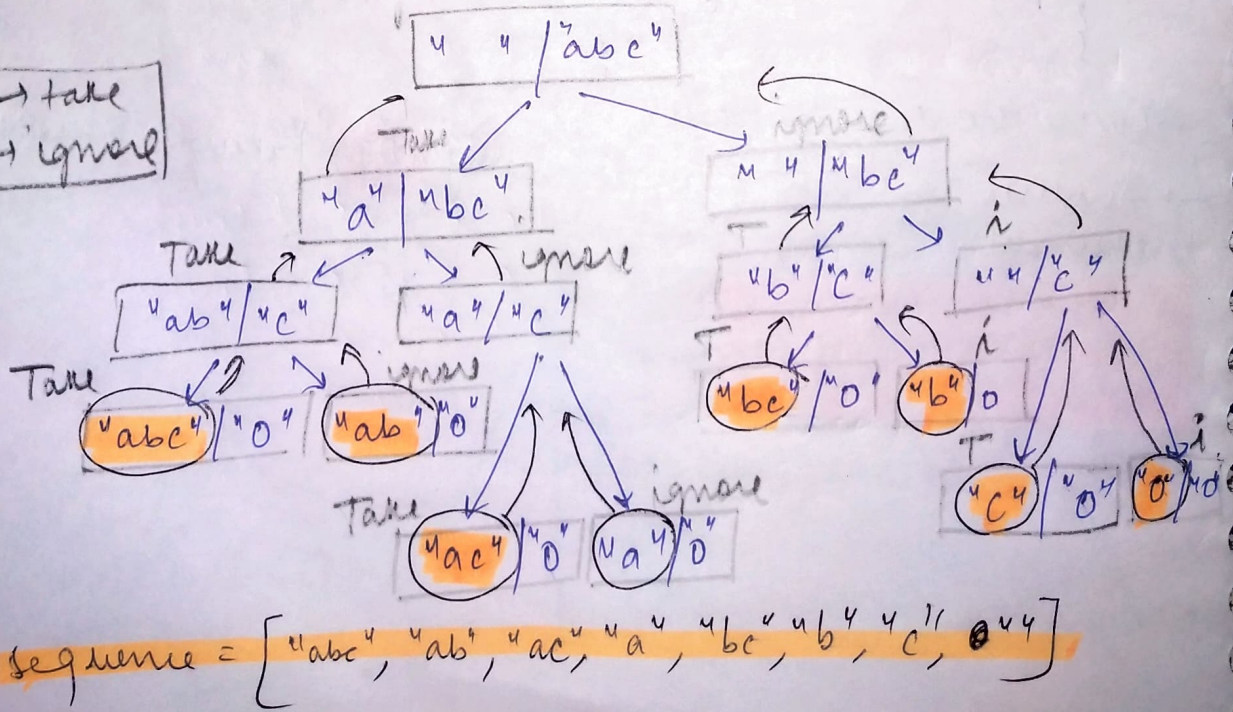
Subsequence of String :- sequence that can be derived from the given string by deleting zero or more elements without changing the order of remaining elements.

Example -

→ we will take two strings processed (resultant) and unprocessed (given string)

→ Two recursive calls → take & ignore

T → take
i → ignore



sequence = ["abc", "ab", "ac", "a", "bc", "b", "c", ""]

Time complexity Analysis :-

* Time taken at each level → n
* Total no. of subsets → at every level subsets are getting doubled (tree)
  → 2^n no. of subsets

$\therefore O(n * 2^n)$

Space complexity :- $O(2^n * n)$ — 2^n: total subsets; n: space taken by each subset

Recursive Implementation

```
void sequence(String p, String up){
if (up == "") { // base condition
sout(p);
return ;}
char ch = up.charAt(0);
sequence(p+ch, up.substring(1));
sequence(p, up.substring(1));
}
```

Iterative Implementation

```
List<List<Integer>> subset(int[] arr){
List<List<Integer>> outer = new ArrayList<>();
outer.add(new ArrayList<>());
for(int num : arr){
  int n = outer.size();
  for(int i=0; i<n; i++){
   List<Integer> internal = new ArrayList<>(outer.get(i));
  internal.add(num);
  outer.add(internal);
  }}
  return outer;}
```

Return an ArrayList of String

```
ArrayList<String> sequence(String p, String up){
if (up.isEmpty()){
ArrayList<String> list = new ArrayList<>();
list.add(p);
return list;}
char ch = up.charAt(0);
ArrayList<String> left = sequence(p+ch, up.substring(1));
ArrayList<String> right = sequence(p, up.substring(1));
left.addAll(right);
return left;
}
```

*duplicate element subset video last

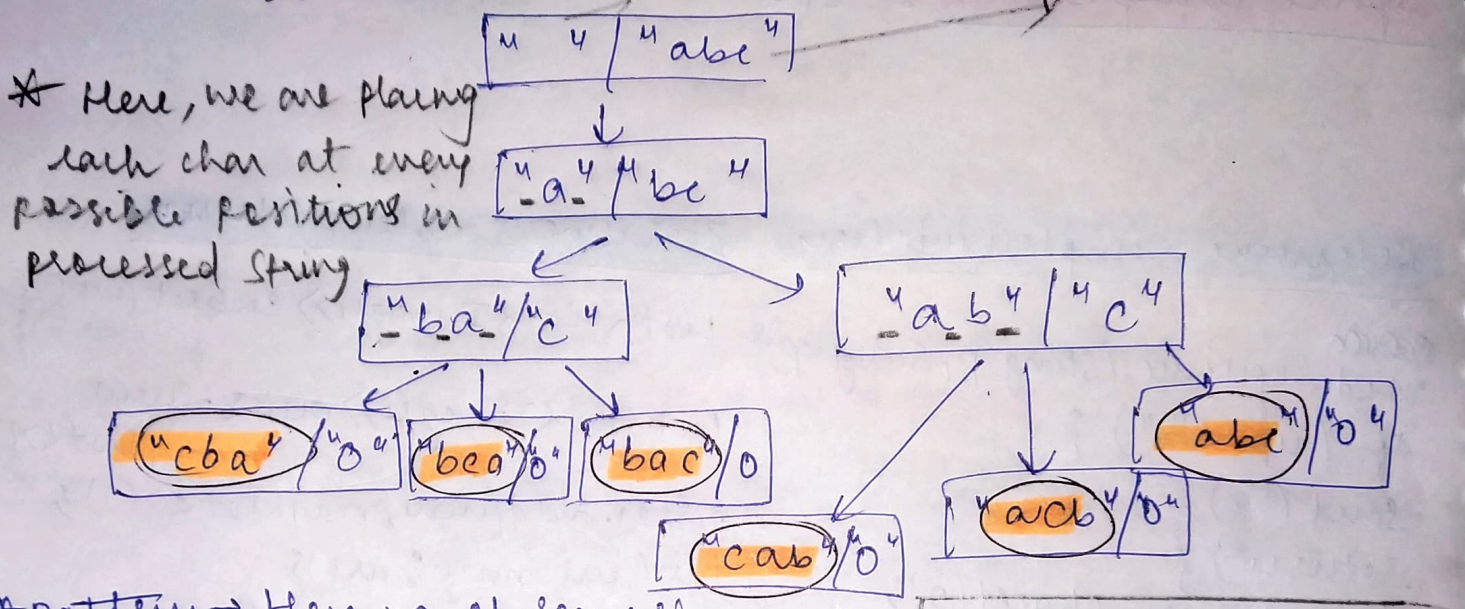
PERMUTATIONS :- Elements position can be at multiple places but we do not ignore elements like we did in subsets.

Example —

str = "abc"

ans = ["abc", "bac", "bca", "cab", "acb", "cba"]

2 variables = processed & unprocessed



*pattern → Here no. of recursive calls are diffrent at every level. They are dependent on size of processed string.

Total no. of permutation = 3! = 6
Factorial

level 1 → size → 1
level 2 → size → 2
level 3 → size → 3

At every level, recursive calls are increased by one of processed string's size (tree)

Time complexity - O(n * n!)
for loop, recursive call