DATE
PAGE

Printing Subsequences

1) Exclude then Include

2) Include then Exclude

s = abc

Output

	abc
c	ab
b	ac
bc	a
a	
ac	bc
ab	b
abc	c

Note → If jis data structure me saare subsequence store ho rhe hai

If we first Include any element in that DS.

So During calling again for Exclude, so us DS se wo element pop krke bhejna exclude ke time pr.

DATE
PAGE

- If hum pehle exclude karte hai, means without including any element call function, so include ke time pr per element ko push krenge on particular DS.

Array

Case I Include then Exclude

PrintSeq (index, vector &ds, arr[7, n)

where index → index on original array

vector<> &ds → use ds to store subsequences.

arr[7 → original array

n → size of original array

DATE _____
PAGE _____

Include

ds.push_back(arr[index])
printseq(index+1, ds, arr, n)

Exclude

ds.pop_back(arr[index])
printseq(index+1, ds, arr, n)

Case II : Exclude then Include.

nums = [1,2,3] given array

printSeq(nums, ans, index, ds)
- nums → original array
- ans → invector
- index → traverse on original array.
- ds → 2D vector

{
  if (index >= nums.size())
  {
    ds.push_back(ans)
    return;
  }

DATE
PAGE

Exclude

printseq(nums, ans, index+1, ds)

Include

int element = @nums[index]
ans.push_back(element)
printseq(nums, ans, index+1, ds)

Output of Above 2 Cases

Case I	Case II
[]	321
[1] [2]	32
[1,2]	31
[3]	3
[1,3]	21
[2,3]	2
[1,2,3]	1

DATE
PAGE

String

Case I → Exclude then Include

PrintSeq(str, n, output, index, vector)

given — store subsequence — collection of all subsequence

```
{
  if (i==n){
    vector.pushback(output)
    return;
  }
```

Exclude

PrintSeq(str, n, output, index+1, vector);

Include

output = output + str[i];
printseq(str, n, output, index+1, vector)

}

DATE _____
PAGE _____

Case-2 → Include the Exclude

Include

PrintSeq(str, n, out+str[i], index+1, vector)

Exclude

PrintSeq(str, n, output, index+1, vector)

DATE
PAGE

Print subsequences with Bit Masking

vector<vector<int>> Result;
store collection of subsequences

if n = [1 2 3] array

Its total size $2^3 = 8$
{ [ ], [1], [2] [3] [1,2] [2 3]
[1,3] [1,2,3] }

Here i used also
1<<n → n times multiply by 2

1<<3 → 2x2x2 (three times 2 multiply)

ye loop chalega jitne vector me data jaana hai, n=3 so data size = $2^3 = 8$

for (i=0; i<pow(2,n); i++)

{

vector<int> ans

Ander wala loop value ki bits par chalega upto n

for (j=0; j<n; j++)

{ if ((1<<j) & i)

ans.push back(nums[j])
}

result.push_back(ans)

Ex — given Array

i = 5

i = 5 → 101 → [1, 2, 3]

Working of Inner loop

```
            j
    1   0   1
&   1   0   1     ← i = 5
    ---------
            1     [1, 2, 3]
```

1 means element need to push into ans (3)

then leftshift 1<<j

```
        j
    1   0   1
&   1   0   1
    ---------
[1 ~~2~~ 3]    0 → means no need to insert
```

leftshift 1<<j