Non Decreasing Sequence - *QOTD 20 jan*

---

leetcode link : [Click](Non-decreasing Subsequences - LeetCode)

★ $Approach - 1$ (recursion and backtracking)

note : if we used ordered map, then time complexity will become $O(2^(n * logn))$

Time : $O(2^n)$

> O(2^N) because we have 2 choices for each element i.e either we pick it or we do not pick it, and n is total number of elements

Space : $O(2^n)$

> for unordered map & rec stack

Approach / steps :-

```
/* ✔️ ⭐Approach - 1 (recursion and backtrack)

        Explanation :-

                -> // Fun.2 : findSeqRec(nums, result, ans, map, index)
                void findSeqRec(vector<int> &nums, vector<vector<int>> &result,
vector<int> &ans, unordered_map<string,bool> &map, int index){

                    step 1 : base case : if the index == n then,
                        step 1.1 : if the ans vector size is more then 1
                            step 1.1.1 : create a string 'temp' now, convert ans into
string and store it into temp, now check if it is present in the map, if yes then
return the function, if no then push it into map and push ans into result 2d vetor and
return

                            // note : these above steps are performed so that we do
not add duplicates to the result

                    recursive calls
                    1. when we do not pick the element
                    2. when we do pick the element, check if element is greater or
equal to the last element of the ans


                -> // Fun.1 : main function
                    step 1 : create a 2d vector 'res' , and a 1d vector 'ans' , int
index = 0
                    step 2 : create a unordered map<string, bool>
                    step 3 : call fun.2 findSeqRec(nums, result, ans, map, index) this
function will fill the 2d 'result', so return it at the end


    */
```

Code :-

```
private:
    // Fun.2 : findSeqRec(nums, result, ans, map, index)
    void findSeqRec(vector<int> &nums, vector<vector<int>> &result, vector<int> &ans, unordered_map<string,bool> &map, int index){

        // step 1 : base case : if the index == n then,
        if(index == nums.size()){

            // step 1.1 : if the ans vector size is more then 1
            if(ans.size() > 1){

                // step : create a string 'temp' now, convert ans into string and store it into temp, now check if it is present in the map, if yes then return the function, if no then push it into map and push ans into result 2d vetor and return
                string temp = "";
                for(int i:ans)
                    temp += i;

                if(map.count(temp) == 1) return;
                else{
                    map[temp] = true;
                    result.push_back(ans);
                    return;
                }

                // note : these above steps are performed so that we do not add duplicates to the result
            }

            return;
        }

        // recursive calls
        // 1. when we do not pick the element
        findSeqRec(nums, result, ans, map, index + 1);

        //2. when we do pick the element, check if element is greater or equal to the last element of the ans
        if(ans.size() == 0 || nums[index] >= ans.back()){
            ans.push_back(nums[index]);
            findSeqRec(nums, result, ans, map, index + 1);
            ans.pop_back();
        }

    }
public:
    // Fun.1 : main function
    vector<vector<int>> findSubsequences(vector<int>& nums) {

        // step 1 : create a 2d vector 'res' , and a 1d vector 'ans' , int index = 0
```

```
        vector<vector<int>> result;
        vector<int> ans;
        int index = 0;

        // step 2 : create a unordered map<string, bool>
        unordered_map<string, bool> map;

        // step 3 : call fun.2 findSeqRec(nums, result, ans, map, index) this function
will fill the 2d 'result', so return it at the end
        findSeqRec(nums, result, ans, map, index);

        return result;

    }
};
```