Increasing Subsequences

Given an integer array, your task is to find all the different possible increasing subsequences of the given array, and the length of an increasing subsequence should be at least 2 .

Example:

```
Input: [4, 6, 7, 7]
Output: [[4, 6], [4, 7], [4, 6, 7], [4, 6, 7, 7], [6, 7], [6, 7, 7], [7,7], [4,7,7]
]
```

Note:

1. The length of the given array will not exceed 15.
2. The range of integer in the given array is [-100,100].
3. The given array may contain duplicates, and two equal integers should also be considered as a special case of increasing sequence.

Solution 1

```java
public List<List<Integer>> findSubsequences(int[] nums) {
 List<List<Integer>> res = new ArrayList<>();
 helper(res, new ArrayList<Integer>(), nums, 0);
 return res;
}

private void helper(List<List<Integer>> res, List<Integer> list, int[] nums, int
id) {
 if (list.size() > 1) res.add(new ArrayList<Integer>(list));
 List<Integer> unique = new ArrayList<Integer>();
 for (int i = id; i < nums.length; i++) {
  if (id > 0 && nums[i] < nums[id-1]) continue; // skip non-increase
  if (unique.contains(nums[i])) continue; // skip duplicate
  unique.add(nums[i]);
  list.add(nums[i]);
  helper(res, list, nums, i+1);
  list.remove(list.size()-1);
 }
}
```

written by shawloatrchen original link here

Solution 2

```
public class Solution {

     public List<List<Integer>> findSubsequences(int[] nums) {
         Set<List<Integer>> res= new HashSet<List<Integer>>();
         List<Integer> holder = new ArrayList<Integer>();
         findSequence(res, holder, 0, nums);
         List result = new ArrayList(res);
         return result;
     }

    public void findSequence(Set<List<Integer>> res, List<Integer> holder, int in
dex, int[] nums) {
        if (holder.size() >= 2) {
            res.add(new ArrayList(holder));
        }
        for (int i = index; i < nums.length; i++) {
            if(holder.size() == 0 || holder.get(holder.size() - 1) <= nums[i]) {
                holder.add(nums[i]);
                findSequence(res, holder, i + 1, nums);
                holder.remove(holder.size() - 1);
            }
        }
    }
}
```

written by adarshkp original link here

Solution 3

```
public class Solution {
    public List<List<Integer>> findSubsequences(int[] nums) {
        List<List<Integer>> res = new LinkedList<>();
        helper(new LinkedList<Integer>(), 0, nums, res);
        return res;
    }
    private void helper(LinkedList<Integer> list, int index, int[] nums, List<Lis
t<Integer>> res){
        if(list.size()>1) res.add(new LinkedList<Integer>(list));
        Set<Integer> used = new HashSet<>();
        for(int i = index; i<nums.length; i++){
            if(used.contains(nums[i])) continue;
            if(list.size()==0 || nums[i]>=list.peekLast()){
                used.add(nums[i]);
                list.add(nums[i]);
                helper(list, i+1, nums, res);
                list.remove(list.size()-1);
            }
        }
    }
}
```

Pretty straightforward. Maybe one thing is: while nums is not necessarily sorted but we have to skip duplicates in each recursion, so we use a hash set to record what we have used in this particular recursion.

written by chidong original link here