

26 December 2022 15:58

26 December 2022 15:58

que 2º-

Medium Accuracy: 68.1% Submissions: 238+ Points: 4

 Land your Dream Job with Mega Job-a-thon. Register Now! 

A binary string is that string which contains only 0 and 1.

Example 1:

Input:

N = 3

Output:
000 , 001 , 010 , 100 , 101

Explanation:

None of the above strings contain consecutive 1s. "110" is not an answer as it has '1's occurring consecutively.

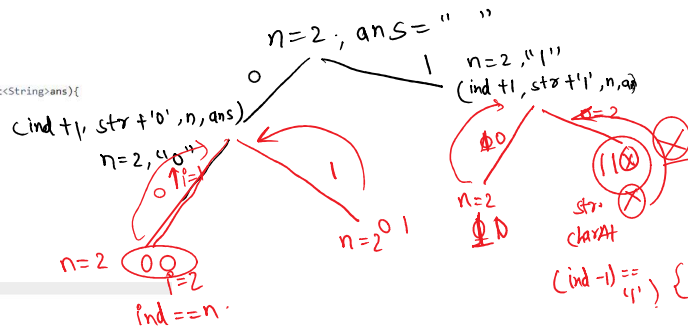
Screen clipping taken: 26-12-2022 16:51

```

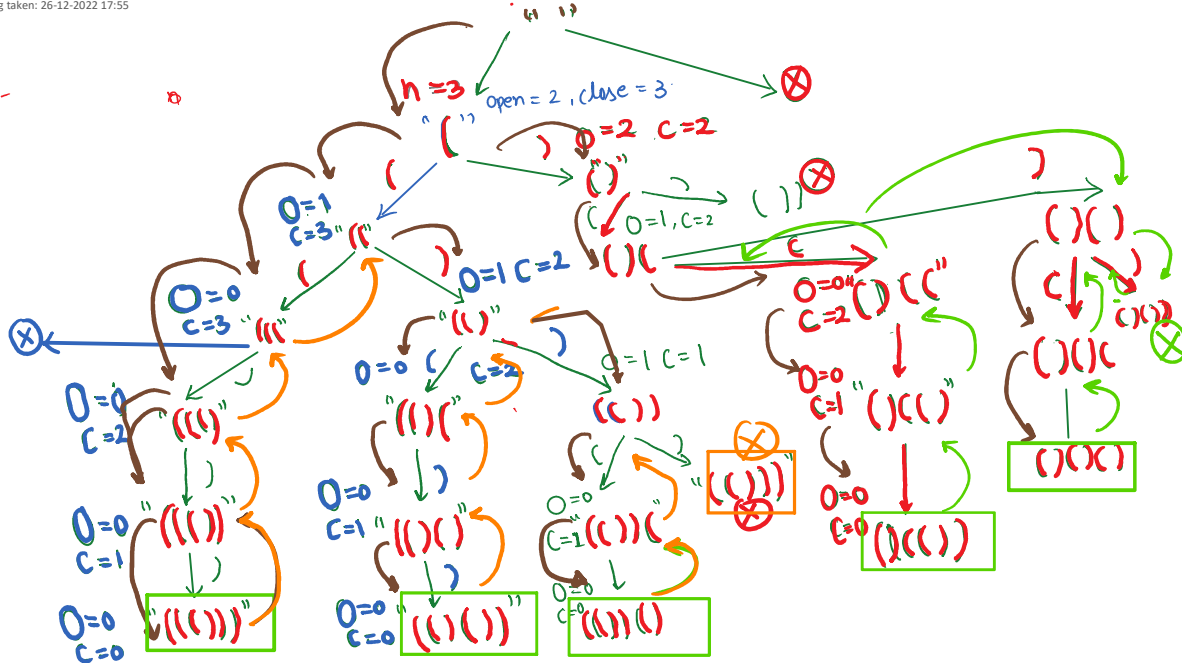
26 //User function template for Java
27
28 class Solution {
29     private static void getList(int ind, String str, int n, List<String>ans){
30         if(ind == n){
31             ans.add(str);
32             return;
33         }
34         getList(ind +1, str + '0', n, ans);
35
36         if(ind >0 && str.charAt(ind-1) == '1'){
37
38             return;
39         }
40         getList(ind +1, str + '1', n, ans);
41     }
42     public static List<String> generateBinaryStrings(int n) {
43         // code here
44         List<String>ans = new ArrayList<>();
45         getList(0, "", n, ans);
46
47         return ans;
48     }
49 }
50
51
52

```

Screen clipping taken: 26-12-2022 17:55



$n=3$ ans open close:



call stack

0 = 2 " ("

①

Observation:-

Well formed parenthesis means-

each "(" has a corresponding ")" so "(" valid but

) (not valid -

② Observation:- no of "(" \geq ")" = n. and the total length of generated string will be $n \times 2$.

At each point in our string, we are faced with 3 choices

- ① If the length of string = $n \times 2$ → if yes then show it as result.
- ② Is the no. of "(" less than n → if yes then add another "("
- ③ Is the no of ")" less than n and less than "(" . If yes then add another ")"
if none of the condition match then poped

if (str.length == $2 \times n$) //→ Add to our result.

if (open < n) → str + "(" and recursively call the fn.

if (close < open) → str + ")" and —:

```

35 //User function Template for Java
36
37 class Solution {
38
39 public void backtracking( List<String>ans, int n, String curr, int open, int close){
40     if(curr.length() == 2 * n){
41         ans.add(curr);
42         return;
43     }
44     if(open < n){
45         backtracking(ans, n, curr + '(', open + 1, close);
46     }
47     if(close < open){
48         backtracking(ans, n, curr + ')', open, close + 1);
49     }
50 }
51
52 public List<String> AllParenthesis(int n)
53 {
54     // Write your code here
55     List<String>ans = new ArrayList<String>();
56     backtracking(ans, n, "", 0, 0);
57     return ans;
58 }
59

```

Screen clipping taken: 26-12-2022 19:40

que:- power set:-

Power Set

Easy Accuracy: 43.3% Submissions: 38K+ Points: 2

Land your Dream Job with Mega Job-a-thon. Register Now!

Given a string S, Find all the possible subsequences of the String in lexicographically-sorted order.

Example 1:

Input : str = "abc"

Output: a ab abc ac b bc c

Explanation : There are 7 subsequences that can be formed from abc.

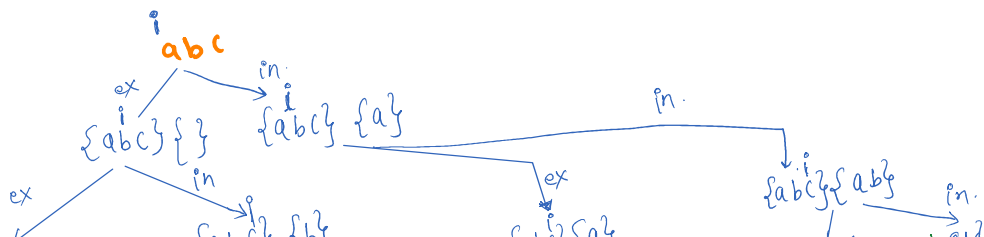
Example 2:

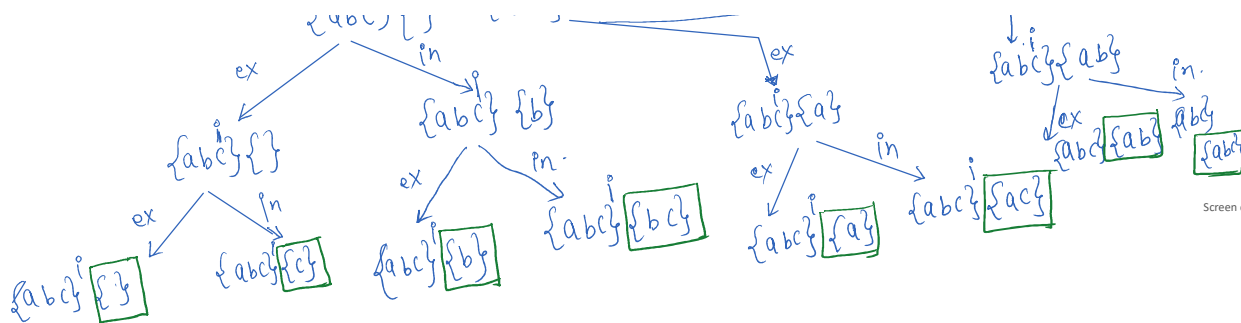
Input: str = "aa"

Output: a a aa

Explanation : There are 3 subsequences that can be formed from aa.

Screen clipping taken: 26-12-2022 19:59





```

0 //User function Template for Java
1
2 class Solution
3 {
4     public static void allpossible( List<String>ans , String str, String s , int ind){
5         if(ind >= s.length()){
6             if(str.length() != 0)
7                 ans.add(str);
8             return;
9         }
10        allpossible(ans, str + s.charAt(ind), s,ind +1);
11        allpossible(ans, str,s, ind +1 );
12    }
13
14    return;
15 }
16
17 public List<String> AllPossibleStrings(String s)
18 {
19     // Code here
20     List<String>ans = new ArrayList<>();
21     int ind = 0;
22     allpossible(ans, "" , s,ind);
23     Collections.sort(ans);
24     return ans;
25 }
26
27 }
28
29

```

que :- Combination sum - I :-

39. Combination Sum

Medium

14.5K

290

Companies

Given an array of **distinct** integers `candidates` and a target integer `target`, return a *list of all **unique combinations** of `candidates` where the chosen numbers sum to `target`*. You may return the combinations in **any order**.

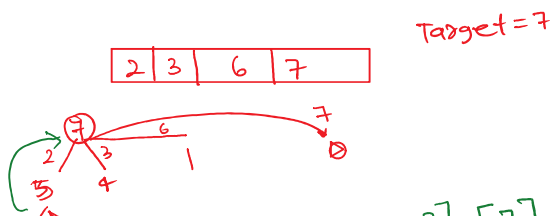
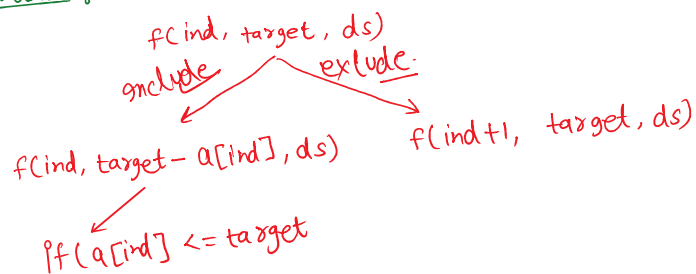
The **same** number may be chosen from `candidates` an **unlimited number of times**. Two combinations are unique if the **frequency** of at least one of the chosen numbers is different.

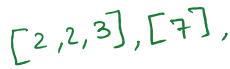
The test cases are generated such that the number of unique combinations that sum up to `target` is less than **150** combinations for the given input.

Example 1:

Input: `candidates = [2,3,6,7], target = 7`
Output: `[[2,2,3],[7]]`
Explanation:
2 and 3 are candidates, and $2 + 2 + 3 = 7$. Note that 2 can be used multiple times.
7 is a candidate, and $7 = 7$.
These are the only two combinations.

DAY Run the Problem:-





Handwritten recursive tree for finding the minimum element in a sorted array. The tree shows recursive calls with parameters (start, end, mid) and returns the minimum value. Annotations include 'POP' for stack operations, '90%' for a specific path, and 'a[ind] > target' for a base case. Red and green arrows highlight different parts of the tree.

Key annotations and steps:

- POP**: Indicated on the left side of the tree, showing the stack state during recursion.
- 90%**: A red arrow pointing to a specific node, indicating a high probability or a specific path.
- a[ind] > target**: A condition used to determine the next step in the recursion.
- Red and Green Arrows**: Highlight different parts of the tree, likely representing different branches or the final path to the minimum element.

Screen clipping taken: 27-12-2022 10:06

40. Combination Sum II

Medium

183

Companies

Given a collection of candidate numbers (`candidates`) and a target number (`target`), find all unique combinations in `candidates` where the candidate numbers sum to `target`.

Each number in `candidates` may only be used **once** in the combination.

Note: The solution set must not contain duplicate combinations.

Example 1:

Input: `candidates = [10,1,2,7,6,1,5]`, `target = 8`

if (array)

0	1	2	3	4
2	5	2	1	2

Target = 4

if(arr[i] > target) break;

```
Input: candidates = [10,1,2,7,6,1,5], target = 8
Output:
[
  [1,1,6],
  [1,2,5],
  [1,7],
  [2,6]
]
```

```

f(idx, target, ds, ans) ~v1~
{
    if (i > target)
        break;
    ds.add(arr[i])
    f(idx+1, target-arr[i], ds, ans)
    ds.remove(arr[i])
}

```

```
if (target == 0)
    ans.add(ds);
```

Handwritten diagram illustrating a recursive tree for the function $f(0, 4, [])$. The root node is $f(0, 4, [])$. It has four children: $f(0, 3, [1])$, $f(1, 2, [1])$, $f(2, 2, [1])$, and $f(3, 2, [2])$. The nodes $f(1, 2, [1])$, $f(2, 2, [1])$, and $f(3, 2, [2])$ are marked with a circled 'X' and labeled '1', '2', and '3' respectively, indicating they are not part of the optimal solution. The node $f(0, 3, [1])$ is labeled '1' and has a child $f(1, 2, [1])$, which is also marked with a circled 'X' and labeled '1'. The diagram shows the recursive calls and the pruning of non-optimal branches.

```

1 class Solution {
2     public void findCombination(int ind, int[] candidates, int target, List<List<Integer>> ans,
3     ArrayList<Integer> list){
4         if(target ==0){
5             ans.add(new ArrayList<>(list));
6             return;
7         }
8         for(int i=ind; i<candidates.length; i++){
9             if(i > ind && candidates[i-1] == candidates[i]){
10                 continue;
11             }
12             if(candidates[i] > target){
13                 break;
14             }
15             list.add(candidates[i]);
16             findCombination(i+1, candidates, target - candidates[i], ans, list);
17             list.remove(list.size()-1);
18         }
19     }
20     public List<List<Integer>> combinationSum2(int[] candidates, int target) {
21         List<List<Integer>> ans = new ArrayList<>();
22         Arrays.sort(candidates);
23         findCombination(0, candidates, target, ans, new ArrayList<>());
24         return ans;
25     }
26 }

```

que 3:- Combination sum - 3

Screen clipping taken: 27-12-2022 11:35

- Only numbers 1 through 9 are used.
- Each number is used **at most once**.

Example 1:

```
Input: k = 3, n = 7
Output: [[1,2,4]]
Explanation:
1 + 2 + 4 = 7
There are no other valid combinations.
```

Example 2:

Input: $k = 3, n = 9$
Output: $[[1, 2, 6], [1, 3, 5], [2, 3, 4]]$

```

1 class Solution {
2     public static void solve(List<Integer>>ans, int start, int k, int n, ArrayList<Integer>>list){
3         if(n < 0) return;
4         if(k==0 && n==0){
5             ans.add(new ArrayList<>(list));
6             return;
7         }
8         for(int i=start; i<=n; i++){
9             list.add(i);
10            solve(ans, i+1, k-1, n-i, list);
11            list.remove(list.size()-1);
12        }
13    }
14    public List<List<Integer>> combinationSum(int k, int n) {
15        List<List<Integer>>ans = new ArrayList<>();
16
17        solve(ans, 1, k, n, new ArrayList<>());
18
19        return ans;
20    }
21 }

```

Screen clipping taken: 27-12-2022 11:35

que:- Subset sum :-

Subset Sums

Basic Accuracy: 72.55% Submissions: 53K+ Points: 1

Land your Dream Job with Mega Job-a-thon. Register Now!

Given a list arr of N integers, print sums of all subsets in it.

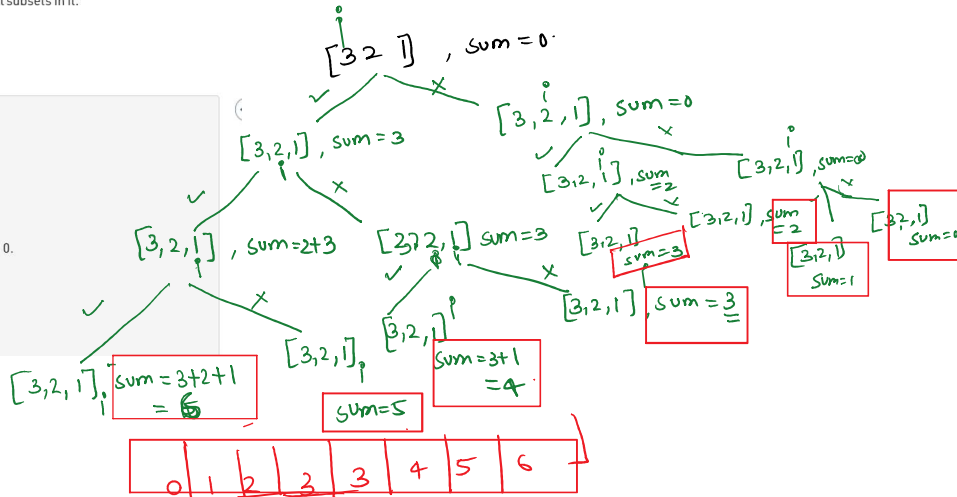
Example 1:

Input:
N = 2
arr[] = {2, 3}

Output:
0 2 3 5

Explanation:
When no elements is taken then Sum = 0.
When only 2 is taken then Sum = 2.
When only 3 is taken then Sum = 3.
When element 2 and 3 are taken then Sum = 2+3 = 5.

Screen clipping taken: 27-12-2022 11:37



Time complexity $O(2^n) + O(2^n \log(2^n))$
SC:- $O(2^n)$

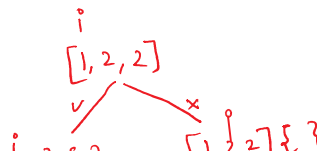
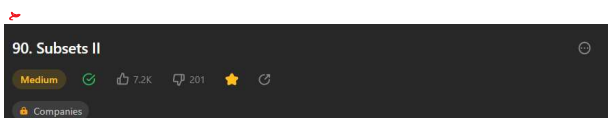
```

32
33
34 //User function Template for Java
35 class Solution{
36     public static void solve(ArrayList<Integer>arr, int n, ArrayList<Integer>ans, int ind, int sum){
37         if(ind == n){
38             ans.add(sum);
39             return;
40         }
41         solve(arr, n, ans, ind+1, sum + arr.get(ind)); //pick the element
42         solve(arr, n, ans, ind+1, sum);
43     }
44     ArrayList<Integer> subsetSums(ArrayList<Integer> arr, int N){
45         // code here
46         ArrayList<Integer>ans = new ArrayList<>();
47         solve(arr, N, ans, 0, 0);
48         Collections.sort(ans);
49         return ans;
50     }
51 }
52

```

Screen clipping taken: 27-12-2022 11:53

que:- Subset sum-II :-



90. Subsets II

Medium 7.2K 201

Companies

Given an integer array `nums` that may contain duplicates, return *all possible subsets* (the power set).
The solution set **must not** contain duplicate subsets. Return the solution in **any order**.

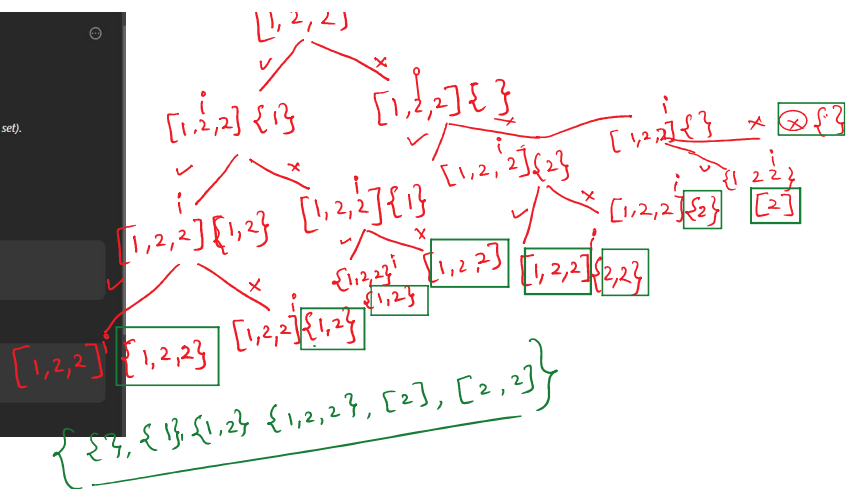
Screen clipping taken: 27-12-2022 12:14

Example 1:

Input: `nums = [1,2,2]`
Output: `[[], [1], [1,2], [1,2,2], [2], [2,2]]`

Example 2:

Input: `nums = [0]`
Output: `[[], [0]]`



Screen clipping taken: 27-12-2022 11:54

```

1 class Solution {
2     public static void solve(int ind, int[] nums, ArrayList<Integer> list, List<List<Integer>> ans){
3         ans.add(new ArrayList<>(list));
4         for(int i = ind; i < nums.length; i++){
5             if(i != ind && nums[i] == nums[i-1]) continue;
6             list.add(nums[i]);
7             solve(i+1, nums, list, ans);
8             list.remove(list.size()-1);
9         }
10    }
11    public List<List<Integer>> subsetsWithDup(int[] nums) {
12        Arrays.sort(nums);
13        List<List<Integer>> ans = new ArrayList<>();
14        solve(0, nums, new ArrayList<>(), ans);
15        return ans;
16    }
17 }

```

TC - $K \times 2^{N-1}$
SC - $K \times 2^{N-1}$

que:- letter combination of a phone number:-

17. Letter Combinations of a Phone Number

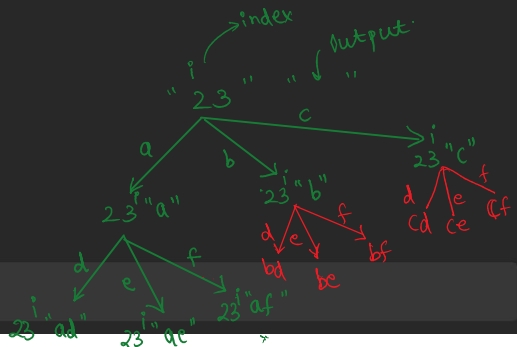
Medium 13.6K 790

Companies

Given a string containing digits from `2-9` inclusive, return all possible letter combinations that the number could represent. Return the answer in **any order**.
A mapping of digits to letters (just like on the telephone buttons) is given below. Note that `1` does not map to any letters.

Example 1:

Input: `digits = "23"`
Output: `["ad","ae","af","bd","be","bf","cd","ce","cf"]`



Screen clipping taken: 27-12-2022 12:16

```

1 class Solution {
2     private String[] keyword = new String[]{"", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"};
3     public void solve(String digits, int ind, String str, List<String> ans){
4         if(digits.length() == ind){
5             ans.add(str);
6             return;
7         }
8         String s = keyword[digits.charAt(ind) - '0'];
9         for(char ch : s.toCharArray()){
10             solve(digits, ind+1, str+ch, ans);
11         }
12    }
13    public List<String> letterCombinations(String digits) {
14        List<String> ans = new ArrayList<>();
15        if(digits.length() == 0){
16            return ans;
17        }
18        solve(digits, 0, "", ans);
19        return ans;
20    }
21 }
22 }

```

