A general approach to backtracking questions in Java (Subsets, Permutations, Combination Sum, Palindrome …

Last Edit: Jun 8, 2016, 7:07 PM

issac3 (/issac3)

This structure might apply to many other backtracking questions, but here I am just going to demonstrate Subsets, Permutations, and Combination Sum.

Subsets : https://leetcode.com/problems/subsets/ (https://leetcode.com/problems/subsets/)

```
public List<List<Integer>> subsets(int[] nums) {
    List<List<Integer>> list = new ArrayList<>();
    Arrays.sort(nums);
    backtrack(list, new ArrayList<>(), nums, 0);
    return list;
}

private void backtrack(List<List<Integer>> list , List<Integer> tempList, int [] nums, int start){
    list.add(new ArrayList<>(tempList));
    for(int i = start; i < nums.length; i++){
        tempList.add(nums[i]);
        backtrack(list, tempList, nums, i + 1);
        tempList.remove(tempList.size() - 1);
    }
}
```

(/problems/combination-sum/discuss/) › A general approach to backtracking questions in Java (Subsets, Permutations, Combination Sum, Palindrome Partitioning)

Subsets II (contains duplicates) : https://leetcode.com/problems/subsets-ii/ (https://leetcode.com/problems/subsets-ii/)

```
public List<List<Integer>> subsetsWithDup(int[] nums) {
    List<List<Integer>> list = new ArrayList<>();
    Arrays.sort(nums);
    backtrack(list, new ArrayList<>(), nums, 0);
    return list;
}

private void backtrack(List<List<Integer>> list, List<Integer> tempList, int [] nums, int start){
    list.add(new ArrayList<>(tempList));
    for(int i = start; i < nums.length; i++){
        if(i > start && nums[i] == nums[i-1]) continue; // skip duplicates
        tempList.add(nums[i]);
        backtrack(list, tempList, nums, i + 1);
        tempList.remove(tempList.size() - 1);
    }
}
```

Permutations : https://leetcode.com/problems/permutations/ (https://leetcode.com/problems/permutations/)

```
public List<List<Integer>> permute(int[] nums) {
   List<List<Integer>> list = new ArrayList<>();
   // Arrays.sort(nums); // not necessary
   backtrack(list, new ArrayList<>(), nums);
   return list;
}

private void backtrack(List<List<Integer>> list, List<Integer> tempList, int [] nums){
   if(tempList.size() == nums.length){
      list.add(new ArrayList<>(tempList));
   } else{
      for(int i = 0; i < nums.length; i++){
         if(tempList.contains(nums[i])) continue; // element already exists, skip
         tempList.add(nums[i]);
         backtrack(list, tempList, nums);
         tempList.remove(tempList.size() - 1);
      }
   }
}
```

Permutations II (contains duplicates) : https://leetcode.com/problems/permutations-ii/ (https://leetcode.com/problems/permutations-ii/)

```
public List<List<Integer>> permuteUnique(int[] nums) {
    List<List<Integer>> list = new ArrayList<>();
    Arrays.sort(nums);
    backtrack(list, new ArrayList<>(), nums, new boolean[nums.length]);
    return list;
}

private void backtrack(List<List<Integer>> list, List<Integer> tempList, int [] nums, boolean [] used){
    if(tempList.size() == nums.length){
        list.add(new ArrayList<>(tempList));
    } else{
        for(int i = 0; i < nums.length; i++){
            if(used[i] || i > 0 && nums[i] == nums[i-1] && !used[i - 1]) continue;
            used[i] = true;
            tempList.add(nums[i]);
            backtrack(list, tempList, nums, used);
            used[i] = false;
            tempList.remove(tempList.size() - 1);
        }
    }
}
```

Combination Sum : https://leetcode.com/problems/combination-sum/ (https://leetcode.com/problems/combination-sum/)

```
public List<List<Integer>> combinationSum(int[] nums, int target) {
    List<List<Integer>> list = new ArrayList<>();
    Arrays.sort(nums);
    backtrack(list, new ArrayList<>(), nums, target, 0);
    return list;
}

private void backtrack(List<List<Integer>> list, List<Integer> tempList, int [] nums, int remain, int start){
    if(remain < 0) return;
    else if(remain == 0) list.add(new ArrayList<>(tempList));
    else{
        for(int i = start; i < nums.length; i++){
            tempList.add(nums[i]);
            backtrack(list, tempList, nums, remain - nums[i], i); // not i + 1 because we can reuse same elements
            tempList.remove(tempList.size() - 1);
        }
    }
}
```

Combination Sum II (can't reuse same element) : https://leetcode.com/problems/combination-sum-ii/ (https://leetcode.com/problems/combination-sum-ii/)

```
public List<List<Integer>> combinationSum2(int[] nums, int target) {
    List<List<Integer>> list = new ArrayList<>();
    Arrays.sort(nums);
    backtrack(list, new ArrayList<>(), nums, target, 0);
    return list;

}

private void backtrack(List<List<Integer>> list, List<Integer> tempList, int [] nums, int remain, int start){
    if(remain < 0) return;
    else if(remain == 0) list.add(new ArrayList<>(tempList));
    else{
        for(int i = start; i < nums.length; i++){
            if(i > start && nums[i] == nums[i-1]) continue; // skip duplicates
            tempList.add(nums[i]);
            backtrack(list, tempList, nums, remain - nums[i], i + 1);
            tempList.remove(tempList.size() - 1);
        }
    }
}
```

Palindrome Partitioning : https://leetcode.com/problems/palindrome-partitioning/ (https://leetcode.com/problems/palindrome-partitioning/)

```
public List<List<String>> partition(String s) {
   List<List<String>> list = new ArrayList<>();
   backtrack(list, new ArrayList<>(), s, 0);
   return list;
}

public void backtrack(List<List<String>> list, List<String> tempList, String s, int start){
   if(start == s.length())
      list.add(new ArrayList<>(tempList));
   else{
      for(int i = start; i < s.length(); i++){
         if(isPalindrome(s, start, i)){
            tempList.add(s.substring(start, i + 1));
            backtrack(list, tempList, s, i + 1);
            tempList.remove(tempList.size() - 1);
         }
      }
   }
}

public boolean isPalindrome(String s, int low, int high){
   while(low < high)
      if(s.charAt(low++) != s.charAt(high--)) return false;
   return true;
}
```

Comments: 33

Sort By

Click here to reply
(https://discuss.leetcode.com/topic/46161)

hide (/hide) Jun 30, 2016, 11:17 AM
I don't think sorting the nums array is useful in your Combination Sum solution. Your for loop condition should also contain '&& nums[i] <= remain'
17

l78 (/l78) Nov 15, 2016, 3:58 PM
Hi, in the first one, `list.add(new ArrayList<>(tempList));`
If I use `list.add(templist);` directly, then it's emptly. But if I print the elements, they are in the templist. Why this happens? Thanks.
Got it. The templist will change later.
8

hide (/hide) Jul 17, 2016, 4:23 PM
@HaitaoSun Say remain is 7, and you have numbers 8,9,10,11,12,13,14,15,16… up to a huge number.
If you put the condition in the for loop, it'll break out as soon as you see '8'. In OP's case, it'll check through all the numbers even when it could have just stopped after 8.
So actually, sorting the numbers CAN be useful, if you add the condition I stated to the for loop. Otherwise, sorting the numbers is useless because you aren't using the fact that they're sorted to prune your possible results.
8

humachine (/humachine) Dec 28, 2016, 8:27 PM
@smallskysky11 You need to create a COPY of templist before adding it. As you see templist keeps getting passed on as an argument and keeps getting modified. If you just add templist, all the modifications you make to templist get reflected in the final list.
4

sabrinasong (/sabrinasong) Sep 19, 2016, 8:19 PM
Nice post!
But in permutation,

```
if(tempList.contains(nums[i])) continue;
```

is O(n) complexity, right?
This makes the total complexity n^n instead of n! ?
3

shenzhu0127 (/shenzhu0127) Jul 13, 2016, 10:30 AM

great summarization!

3

ranhar (/ranhar) Feb 14, 2017, 1:11 AM

@poojasunder27 I think it`s, O(k * 2 ^ n), where k is the average length of all the combinations and n is the size of nums

2

ramanan2 (/ramanan2) Jul 25, 2016, 6:42 PM

To generate anagrams of a Given String using same structure

```
List<String> anagrams (String input){
        char[] inp = input.toCharArray();
        List<String> result = new ArrayList();
        backtrack(inp,result,inp.length,"");
        return result;
}
void backtrack(char[] input,List result,final int size,String str) {
        if(str.length()==size){  // or can be input.length
                result.add(new String(str));
        }
        else{
                for(int i =0 ;i<input.length;i++){
                        if(str.contains(""+input[i])){
                                continue;
                        }
                        String newStr = str + input[i];
                        backtrack(input,result,size,newStr);
                }
        }
}
```

2

congchengchina (/congchengchina) Apr 26, 2017, 4:46 AM

@hide you are right, you can safely do it without sorting the array first. But, actually, you can take advantege of sorted array.
Whenever you find `remain < 0`, it suggests that `nums[i]` is too large, and it also means the same thing for `nums[j]` while `j` is a valid index and `j > i` in sorted array, so the solution below can be a choice if you sort the array first:

```
public List<List<Integer>> combinationSum(int[] candidates, int target) {
     List<List<Integer>> res = new ArrayList<>();
     Arrays.sort(candidates);
     if (candidates[0] > target) return res;
     dfs(candidates, target, 0, res, new ArrayList<>());
     return res;
}

private void dfs(int[] candidates,
                 int target,
                 int start,
                 List<List<Integer>> res,
                 List<Integer> tracker)
{
     if (target == 0) res.add(new ArrayList<>(tracker));
     else
     {
         for (int i = start; i < candidates.length; i++)
         {
              if (candidates[i] > target) return;
              tracker.add(candidates[i]);
              dfs(candidates, target - candidates[i], i, res, tracker);
              tracker.remove(tracker.size() - 1);
         }
     }
}
```

1

gokullu (/gokullu) Mar 23, 2017, 1:50 AM

@nksharath you need '!used[i-1]' to differentiate whether we are processing the tempList which starts with 'i' or with 'i-1'.
If we started with 'i-1' then used[i-1] is still true in that case even though 'nums[i] == nums[i-1]' we should not skip it.
once we have finished processing 'i-1', used[i-1] is set to false and now processing 'i' doesn't make sense if nums[i]==nums[i-1] as it will generate the same results as 'i-1'

1

Notes