Simple java solution with O(1) space

12

lmx707
Reputation: ★ 23

```java
public int numDecodings(String s) {
    int n1 =1, n2=1, n3=0;
    if(s.length()==0||s.charAt(0)=='0') return 0;
    for(int i=2; i<=s.length(); i++)
    {
        n3=0;
        if(s.charAt(i-1)!='0') n3=n2;
        int num = Integer.parseInt(s.substring(i-2,i));
        if(num>=10 && num<=26) n3+=n1;
        n1=n2;
        n2=n3;
    }
    return n2;
}
```

Recursion

```java
// by recursive
public class Solution {
    public int numDecodings(String s) {
        int len = s.length();
        if (len == 0) return 0;
        if (len == 1 ){
            if (isValid(s.substring(0,1))) return 1;
            else return 0;
        }
        int res = 0;
        // at least have two elements here.
        if (isValid(s.substring(len-1,len))) res += numDecodings(s.substring(0,len-1));
        if (isValid(s.substring(len-2, len))) res += numDecodings(s.substring(0,len-2));

        return res;
    }

    public boolean isValid (String s){
        if (s.length() == 0 ) return false;
        int i = Integer.parseInt(s);
        if ( i >= 1 && i <= 26 ) return true;
        else return false;
    }
}
```

75. Sort Colors

Saturday, February 18, 2017 11:58 PM

Given an array with *n* objects colored red, white or blue, sort them so that objects of the same color are adjacent, with the colors in the order red, white and blue.

Here, we will use the integers 0, 1, and 2 to represent the color red, white, and blue respectively.

Note:

You are not suppose to use the library's sort function for this problem.

click to show follow up.

Follow up:

A rather straight forward solution is a two-pass algorithm using counting sort.

First, iterate the array counting number of 0's, 1's, and 2's, then overwrite array with total number of 0's, then 1's and followed by 2's.

Could you come up with an one-pass algorithm using only constant space?

```java
public class Solution {
    public void sortColors(int[] nums) {
        //两个指针，一个头一个尾，然后还有个index遍历。
        //遇到0就和头指针，即记录当前1的指针交换
        //遇到2就和尾指针，即记录当前2的指针交换

        int p1 = 0, p2 = nums.length-1, index = 0;
        while(index <= p2){
            if(nums[index] == 0){
                nums[index] = nums[p1];
                nums[p1] = 0;
                ++p1;
            }
            if(nums[index] == 2){
                nums[index] = nums[p2];
                nums[p2] = 2;
                --index;
                --p2;
            }
            ++index;
        }
    }
}
```

38. Count and Say

Monday, February 20, 2017 6:56 PM

The count-and-say sequence is the sequence of integers beginning as follows:

`1, 11, 21, 1211, 111221, ...`

`1` is read off as `"one 1"` or `11`.

`11` is read off as `"two 1s"` or `21`.

`21` is read off as `"one 2`, then `one 1"` or `1211`.

Given an integer *n*, generate the n^{th} sequence.

Note: The sequence of integers will be represented as a string.

题目有误导性，其实就是求这个序列的第n个数。序列中第n个数等于第n-1个数的读法。

```
public class Solution {
    public String countAndSay(int n) {
        String s = "1";
        for(int i=1;i<n;++i){
            s = iterate(s);
        }
        return s;
    }

    public String iterate(String s){
        StringBuilder ns = new StringBuilder();
        char c = s.charAt(0);
        int count = 1;
        //Here should start from 1
        for(int i=1; i<s.length(); ++i){
            if(s.charAt(i) == c){
                count ++;
            }else{
                ns.append(count);
                ns.append(s.charAt(i-1));
                count = 1;
                c = s.charAt(i);
            }
        }
        //Do not forget to append the last result
        ns.append(count);
        ns.append(c);
        return ns.toString();
    }
}
```

用迭代法迭代即可

67. Add Binary

Sunday, February 19, 2017 12:20 AM

Given two binary strings, return their sum (also a binary string).

For example,

a = `"11"`

b = `"1"`

Return `"100"`.

```java
public class Solution {
    public String addBinary(String a, String b) {
        //How we use StringBuilder here
        StringBuilder s = new StringBuilder();
        int l1 = a.length()-1, l2 = b.length()-1;
        //Attention, where we initialize we carry, and where
        // we initialize sum
        int carry = 0;
        while(l1>=0||l2>=0){
            int sum = carry;
            if(l1>=0) sum += a.charAt(l1--)-'0';
            if(l2>=0) sum += b.charAt(l2--)-'0';
            s.append(sum%2);
            carry = sum/2;
        }
        //How to append for StringBuilder, even if it is int
        if(carry != 0) s.append(carry);
        return s.reverse().toString();
    }
}
```

278. First Bad Version

Sunday, February 19, 2017 3:50 PM

You are a product manager and currently leading a team to develop a new product. Unfortunately, the latest version of your product fails the quality check. Since each version is developed based on the previous version, all the versions after a bad version are also bad.

Suppose you have `n` versions `[1, 2, ..., n]` and you want to find out the first bad one, which causes all the following ones to be bad.

You are given an API `bool isBadVersion(version)` which will return whether `version` is bad. Implement a function to find the first bad version. You should minimize the number of calls to the API.

```java
/* The isBadVersion API is defined in the parent class VersionControl.
      boolean isBadVersion(int version); */

public class Solution extends VersionControl {
    public int firstBadVersion(int n) {
        int start = 1;
        int end = n;
        while(start<end){
            //mid = start + (end-start)/2, not just (end-start)/2
            int mid = start+(end-start)/2;
            if(isBadVersion(mid)) end = mid;
            else start = mid+1;
        }

        //Understand what we should return
        return start;
    }
}
```

78. Subsets

Sunday, February 19, 2017 4:10 PM

Given a set of **distinct** integers, *nums*, return all possible subsets.

Note: The solution set must not contain duplicate subsets.

For example,

If ***nums*** = `[1,2,3]` , a solution is:

```
[
  [3],
  [1],
  [2],
  [1,2,3],
  [1,3],
  [2,3],
  [1,2],
  []
]
```

```java
public class Solution {
    public List<List<Integer>> subsets(int[] nums) {
        //How to initialize 2D array, here it must be ArrayList<>, not ArrayList<Integer>, as it's 2D!
        List<List<Integer>> result = new ArrayList<>();
        //Do not forget to sort
        Arrays.sort(nums);
        List<Integer> current = new ArrayList<Integer>();
        backTracking(result, current, nums, 0);
        return result;
    }

    public void backTracking (List<List<Integer>> result, List<Integer> current, int[] nums, int start){
        //Here it must be new ArrayList<>(current), you cannot direct add current as it is list not ArrayList
        result.add(new ArrayList<>(current));
        for(int i=start; i<nums.length; i++){
            current.add(nums[i]);
            backTracking(result,current,nums,i+1);
            current.remove(current.size()-1);
        }
    }
}
```

90. Subsets II

Sunday, February 19, 2017 4:15 PM

Given a collection of integers that might contain duplicates, ***nums***, return all possible subsets.

Note: The solution set must not contain duplicate subsets.

For example,

If ***nums*** = `[1,2,2]`, a solution is:

```
[
  [2],
  [1],
  [1,2,2],
  [2,2],
  [1,2],
  []
]
```

输入数组可能有重复，但是要求输出的组合不能有相同的。

Subsets II (contains duplicates) : 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);
    }
}
```

和78题几乎完全一样，只要加一个判断然后跳过重复的即可。

46. Permutations

Sunday, February 19, 2017 4:22 PM

Given a collection of **distinct** numbers, return all possible permutations.

For example,

`[1,2,3]` have the following permutations:

```
[
  [1,2,3],
  [1,3,2],
  [2,1,3],
  [2,3,1],
  [3,1,2],
  [3,2,1]
]
```

```java
public class Solution {
    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++){
         //Skip element already exit.
         //Erase possible cases like (1,1,1)
         if(tempList.contains(nums[i])) continue;
         tempList.add(nums[i]);
         backtrack(list, tempList, nums);
         tempList.remove(tempList.size() - 1);
      }
   }
}
}
```

47. Permutations II

Sunday, February 19, 2017 4:40 PM

Given a collection of numbers that might contain duplicates, return all possible unique permutations.

For example,

`[1,1,2]` have the following unique permutations:

```
[
  [1,1,2],
  [1,2,1],
  [2,1,1]
]
```

```java
public class Solution {
    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 already used, skip
            //Or although Used, however current one is same as previous one and previous one is not used
            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);
        }
    }
}
}
```

39. Combination Sum

Sunday, February 19, 2017 4:56 PM

Given a **set** of candidate numbers (***C***) **(without duplicates)** and a target number (***T***), find all unique combinations in ***C*** where the candidate numbers sums to ***T***.

The **same** repeated number may be chosen from ***C*** unlimited number of times.

Note:

- All numbers (including target) will be positive integers.
- The solution set must not contain duplicate combinations.

For example, given candidate set `[2, 3, 6, 7]` and target `7`,

A solution set is:

```
[
  [7],
  [2, 2, 3]
]
```

```
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);
        }
    }
}
```

377. Combination Sum IV

Sunday, February 19, 2017 5:06 PM

Given an integer array with all positive numbers and no duplicates, find the number of possible combinations that add up to a positive integer target.

Example:

```
nums = [1, 2, 3]
target = 4

The possible combination ways are:
(1, 1, 1, 1)
(1, 1, 2)
(1, 2, 1)
(1, 3)
(2, 1, 1)
(2, 2)
(3, 1)

Note that different sequences are counted as different combinations.

Therefore the output is 7.
```

Follow up:

What if negative numbers are allowed in the given array?

How does it change the problem?

What limitation we need to add to the question to allow negative numbers?

```java
public class Solution {
    private int[] dp;

    public int combinationSum4(int[] nums, int target) {
        //Here we need target + 1, not nums.length +1
        dp = new int [target+1];
        Arrays.fill(dp, -1);
        dp[0] = 1;
        return helper(nums, target);
    }

    public int helper(int[] nums, int target){
        if(dp[target] != -1) return dp[target];
        int res = 0;
        for(int i=0; i<nums.length; ++i){
            if(target>=nums[i]){
                res += helper(nums, target - nums[i]);
            }
        }
        dp[target] = res;
        return res;
    }

}
```