This repository Search Pull requests Issues Gist

huyilong / **OOP-Design** Unwatch 1 Unstar 1 Fork 0

Code Issues 0 Pull requests 0 Wiki Pulse Graphs Settings

Branch: master **OOP-Design** / ********3Sum2PtrsWindow.java** Find file Copy path

huyilong SUMMARY - ALLLL **3Sum && 2Ptrs && Window** Problems ecd32d3 a minute ago

1 contributor

619 lines (522 sloc) 21.6 KB Raw Blame History

```java
public class Solution {
    public void nextPermutation(int[] nums) {
        //If such arrangement is not possible, it must rearrange it as the lowest possible order (ie, sorted in ascending order).
        // rearranges numbers into the lexicographically next greater permutation of numbers.
        //(2,3,6,5,4,1))
        /*1.从后往前，找到第一个 A[i-1] < A[i]的。也就是第一个排列中的  6那个位置，可以看到A[i]到A[n-1]这些都是单调递减序列。
          2.从 A[n-1]到A[i]中找到一个比A[i-1]大的值（也就是说在A[n-1]到A[i]的值中找到比A[i-1]大的集合中的最小的一个值）
          3.交换 这两个值，并且把A[n-1]到A[i+1]排序，从小到大。*/
         /*
         1) 先从后往前找到第一个不是依次增长的数，记录下位置p。比如例子中的3，对应的位置是1;
2) 接下来分两种情况:
    (1) 如果上面的数字都是依次增长的，那么说明这是最后一个排列，下一个就是第一个，其实把所有数字反转过来即可(比如(6,5,4,3,2,1)下一个是(1,2,3,4,5,6));
    (2) 否则，如果p存在，从p开始往后找，找找找，找到其中比他大的数里面最小的数，然后两个调换位置，比如例子中的4。调换位置后得到(2,4,6,5,3,1)。最后把p之

*/
            //2,3,6,5,4,1
            //scan from last 1 -> find the first not increasing number -> 3
            //then from 3 search back 6,5,4,1 find the {6,5,4} is larger than 3 and the last larger than 3 is 4
            //swap 3 and 4 then becomes 2,4, 6,5,3,1
            //sort from i+1 to last   2,4,1,3,5,6 -> this is the next permutation.

            if(nums == null || nums.length==0 || nums.length==1){
                return;
            }
            int i =nums.length -2;//we need to search the not 1,6,5,4,3 search for 1
            //from last to before -> find the first not increasing number
            while(i>=0 &&nums[i] >= nums[i+1] ){
                i--;
            }
            int length = nums.length;

            if(i<0){
                //reverse the whole permutation
                reverse(nums, 0, length-1);
            }
            //i<0 the jumps out the while means that this is the last one 6,5,4,3,2,1
            //we just need to reverse the whole nums
            if(i>=0){
                int j=i+1;//we need to find the minimum of the set of numbers that are larger than nums[i]
                while(j<nums.length && nums[j] > nums[i]){
                    //because they are increasing say, 2,6,5,4,1 -> we need to find 4
                    //swap 2,4
                    j++;
                }

                //j-- because the loop up is out terminate until nums[j] <= nums[i] we need to go back
                j--;
                swap(nums, i, j);
                //reverse the following numbers after swapping
                //2,4,5,6,1 -> 2,4,1,5,6
                reverse(nums, i+1, length-1);
            }
            //swap and reverse -- inplace
            //the worst case is to search the array for 3 times  -> time complexity O(n)
    }

    //we need to pass the nums as well otherwise the swapping not working
    private void swap(int[] nums, int i, int j){
        int tmp = nums[i];
        nums[i] = nums[j];
        nums[j] = tmp;
    }
    private void reverse(int[] nums, int i, int j){
        while(i<j){
            swap(nums, i++, j--);
        }
    }
}


Given an array of n positive integers and a positive integer s,
find the minimal length of a subarray of which the sum ≥ s.
If there isnt one, return 0 instead.

For example, given the array [2,3,1,2,4,3] and s = 7,
the subarray [4,3] has the minimal length under the problem constraint.

public class Solution {
    public int minSubArrayLen(int s, int[] nums) {
        //we just need to return the length of sub array!!!
        //two pointers make a window!!!!  TIME COMPLEXITY O(N)  OR WE COULD USE BINARY SERACH O(NlogN)
        if(nums == null || nums.length == 0){
            return 0;//not found
        }
        int l =0, r=0;//window is 0 for the beginning
        int sum=0, res = Integer.MAX_VALUE;
        while(r<nums.length){
            //here we firstly increse window size to satisfy that sum>s
            //we use while!!!!
            while(sum<s && r<nums.length){
                sum = sum+nums[r];
                r++;
            }

            //then we possibly decrease the length of window
            while(sum>=s){
                res = Math.min(res, r-l);
                sum = sum-nums[l];
                l++;
            }
        }

        return res == Integer.MAX_VALUE ? 0 :res;
    }
}

longest common prefix
123
12
1
we use a flag to mark whether all first index of the all strings are same
if true, append to final result
stop condition -- find one of them differs from str[0].charAt(0) -- random standard
OR the index is already exceed the length of the shortest string among all these


most common 2sum
we store the distance to go as key and the index of result as value
the index of second value -- distance is just the scanner i!!!
public class Solution {
    public int[] twoSum(int[] nums, int target) {
        HashMap<Integer, Integer> map = new HashMap<>();
        //distance from target --> index
        int[] res = new int[2];
        //two index

        for(int i=0 ; i<nums.length; i++){
            if(!map.containsKey(nums[i])){

                index and the res has a difference of 1!!!

                map.put(target - nums[i], i + 1);
            }else{
                res[0] = map.get(nums[i]);
                res[1] = i +1;

                break!!!;

            }

        }

        return res;
    }
}



3 sum --- we need to sort the array!!!! so that we could use left++/right--;
i =0 -- n
left=i+1
right=num.length-1  for(left<right)


public class Solution{
    public Arraylist<Arraylist<Integer>> threeSum(int[] num){
        List<List<Integer>> res = new List<Arraylist<Integer>>();
        Arrays.sort(num);
        for(int i=0; i<num.length-2;i++){

            int left = i+1;
            int right = num.length-1;

            while(left<right){

                int sum = num[left] + num[right] + num[i];
                /////////
                if(sum ==0){
                    Arraylist<Integer> sub = new Arraylist<>();
                    sub.add(num[left]);
                    sub.add(num[right]);
                    sub.add(num[i]);
                    res.add(sub);


                    //dont forget!!!!!! we need
                    left++;
                    right--;
                    //////////////////////

                    while(left<right && num[left] == num[left-1]){
                        left++;
                    }
                    while(left<right && num[right] == num[right+1]){
                        right--;
                    }

                }else if(sum < 0){
                    left++;
                }else{
                    right--;
                }
            }
        }
    }
}

3 sum closest!!

public class Solution{
    public Arraylist<Arraylist<Integer>> threeSum(int[] num, int target){
        if(num == null || num.length<3){
            return Integer.MAX_VALUE;
        }

        List<List<Integer>> res = new List<Arraylist<Integer>>();
        Arrays.sort(num);
        int closest = Integer.MAX_VALUE/2;
        for(int i=0; i<num.length-2; i++){
            int left = i+1;
            int right = num.length-1;
            while(left<right){
                int sum = num[left] + num[right]+ num[i];
                if(sum == target){
                    return sum;
                    //no left++ and right-- here
                    //because we do not need to find all sub solutions
                }else if(sum < target){
                    left++;

                }else{
                    right--;
                }


                //the later two conditions needs to update the closest
                //according to the diff btw the new sum and target
                closest = Math.abs(sum-target) < Math.abs(closest-target) ?
                            sum : closest;
            }
        }

        return closest;
    }
}

3Sum Smaller

Given an array of n integers nums and a target, find the number of index triplets i, j, k with 0 <= i < j < k < n that satisfy the conditio

For example, given nums = [-2, 0, 1, 3], and target = 2.

Return 2. Because there are two triplets which sums are less than 2:

[-2, 0, 1]
[-2, 0, 3]
Follow up: Could you solve it in O(n2) runtime?
排序法

复杂度

时间 O(N^2) 空间 O(1)

思路

解题思路和3SUM一样，也是先对整个数组排序，然后一个外层循环确定第一个数，
然后里面使用头尾指针left和right进行夹逼，得到三个数的和。如果这个和大于或者等于目标数，
说明我们选的三个数有点大了，就把尾指针right向前一位（因为是排序的数组，所以向前肯定会变小）。
如果这个和小于目标数，那就有right - left个有效的结果。为什么呢?
因为假设我们此时固定好外层的那个数，还有头指针left指向的数不变，那把尾指针向左移0位一直到
左移到left之前一位，这些组合都是小于目标数的。

代码

public class Solution {
    public int threeSumSmaller(int[] nums, int target) {
        // 先将数组排序
        Arrays.sort(nums);
        int cnt = 0;
        for(int i = 0; i < nums.length - 2; i++){
            int left = i + 1, right = nums.length - 1;
            while(left < right){
                int sum = nums[i] + nums[left] + nums[right];
                // 如果三个数的和大于等于目标数，那将尾指针向左移
                if(sum >= target){
                    right--;
                // 如果三个数的和小于目标数，那将头指针向右移
                } else {
                    // right - left个组合都是小于目标数的
                    cnt += right - left;
                    left++;
                }
            }
        }
        return cnt;
    }
}

public class Solution {
    public int[] twoSum(int[] nums, int target) {
        HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();
        int[] result = new int[2];
        //because it is two sum

        for(int i=0; i<nums.length; ++i){
            if(!map.containsKey(nums[i])){
                //the key in the map is remaing target
                //the value in the map is the index *arr+1*
                map.put(target - nums[i], i+1);

            }else{
                int smallindex = map.get(nums[i]);
```

```java

                result[0] = smallindex;
                //we have converted it by adding 1
                result[1] = i+1;
                break;
                //do not need to loop anymore
            }
        }
        return result;
    }
}



public class Solution {
    public String longestPalindrome(String s) {
        String res = new String();
        String temp = new String();
        int len = s.length();
        for(int i=0; i<len; i++){
            for(int j=i+1; j<len; j++){
                temp = s.substring(i,j);
                if(temp.length() > res.length() && helper(temp)){
                    res = temp;
                }
            }
        }

        return res;
    }

    private boolean helper(String s){
        if(s == null || s.length() == 0){
            return true;
        }
        int input_len = s.length();
        for(int i=0; i<(input_len/2); i++){
            if(s.charAt(i)!=s.charAt(input_len-i-1)){
                return false;
            }
        }

        //
        return true;
    }
}

public class Solution {
    public int lengthOfLongestSubstring(String s) {
        if(s.length() == 0 || s ==null){
            return 0;
        }
        // HashSet<Character> hs = new HashSet<>();
        // int max = 0;
        // int len  = s.length();
        // for(int i=0; i<len; i++){
        //     int temp=0
        //     if(!hs.contains(s.charAt(i))){
        //       hs.add(s.charAt(i));
        //       temp++;
        //     }else{

        //     }
        // }
        HashSet<Character> hs = new HashSet<>();
        int max = 0;
        int len  = s.length();
        //at least O(n^2)
        for(int i=0; i<len; i++){
            int temp = 0;
            for(int j=i; j<len; j++){
                if(!hs.contains(s.charAt(j))){
                    hs.add(s.charAt(j));
                    temp++;
                    //System.out.println(temp);
                    if(j == len-1){
                         //max = Math.max(max, temp);
                        return Math.max(max, temp);
                    }
                }else{
                    max = Math.max(max, temp);
                    hs.clear();
                    break;//current loop does not need to continue
                }
            }
        }

        return max;
    }
}

public class Solution {
    public List<List<String>> groupAnagrams(String[] strs) {
        List<List<String>> res = new ArrayList<List<String>>();
        if(strs.length == 0 || strs == null){
            return res;
        }


        //["eat", "tea", "tan", "ate", "nat", "bat"]
        HashMap<String, ArrayList<Integer>> map = new HashMap<>();
        for(int i=0; i<strs.length; i++){
            char[] arr = strs[i].toCharArray();
            Arrays.sort(arr);
            String sorted = String.valueOf(arr);
            if(!map.containsKey(sorted)){
                ArrayList<Integer> index = new ArrayList<>();
                index.add(i);
                map.put(sorted, index);
            }else{
                //it already has a anagram existing
                map.get(sorted).add(i);
            }
        }

        for(ArrayList<Integer> list : map.values()){
            List<String> sub = new ArrayList<>();
            for(int j : list){
                sub.add(strs[j]);
            }
            Collections.sort(sub);
            res.add(sub);
        }

        return res;
    }
}

给的例子太不具说明性了。应该举这个例子：

["eqdf", "qcpr"]。

((‘q’ - 'e') + 26) % 26 = 12, ((‘d’ - 'q') + 26) % 26 = 13, ((‘f’ - 'd') + 26) % 26 = 2

((‘c’ - 'q') + 26) % 26 = 12, ((‘p’ - 'c') + 26) % 26 = 13, ((‘r’ - 'p') + 26) % 26 = 2

所以"eqdf"和"qcpr"是一组shifted strings。

public class Solution {
    public List<List<String>> groupStrings(String[] strings) {
        List<List<String>> result = new ArrayList<List<String>>();
        HashMap<String, List<String>> d = new HashMap<>();
        for(int i = 0; i < strings.length; i++) {
            StringBuffer sb = new StringBuffer();
            for(int j = 0; j < strings[i].length(); j++) {
                sb.append(Integer.toString(((strings[i].charAt(j) - strings[i].charAt(0)) + 26) % 26));
                sb.append(" ");
            }
            String shift = sb.toString();
            if(d.containsKey(shift)) {
                d.get(shift).add(strings[i]);
            } else {
                List<String> l = new ArrayList<>();
                l.add(strings[i]);
                d.put(shift, l);
            }
        }

        for(String s : d.keySet()) {
            Collections.sort(d.get(s));
            result.add(d.get(s));
        }
        return result;
    }
}

public class Solution {
    public void sortColors(int[] nums) {

        //{0,1,2,0,0,2,1} -> {0,0,0,1,1,2,2}
        /*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

            Could you come up with an one-pass algorithm using only constant space?*/
         //we are creating a new array of the size of the number of different colors
         //then we could just count[0] ++ means the color 0 has one more instance

         if(nums == null || nums.length ==1){
             return;//here we just have one color and therefore we do not need to sort
         }

         int[] count = new int[3];
         for(int i=0; i<nums.length; i++){
             count[nums[i]] ++;
             //which just has three conditions count[0-2]
         }

         //now we need to overwrite the array with the new result
         int i=0; //this is the cursor to overwrite the old array
        // int j=2; // j is to tracking how many kind of colors left -- if the count for one color -> 0 then decrease j
        //because we need to put 0 at first
        int j=0;
         while(j<=2){
             if(count[j] !=0 ){
                 //this kind of color still have instances // not copied thoroughly yet
                 nums[i++] = j;
                 //we continuously copy 0 then next copy1s..
                 count[j]--;
             }else{
                 //we finished copy 0s we then copy count[1] 1s and then count[2] ge 2s
                 j++;
             }
         }
    }
}
----------------convert int to string --- use String s = Sring.valueOf(num[i])----------

public class Solution {
    public String largestNumber(int[] nums) {
        String[] strs = new String[nums.length];
        for(int i=0; i<nums.length; i++){
          strs[i] = String.valueOf(nums[i]);
        }

    //we are sorting the whole strs array with calling Arrays
    //we are sorting the whole linkedlist with calling Collections
    //we are overwriting the sortig algorithm with Comparator<String> and redefine the compare function
    //int if compare >0 then
         Arrays.sort(strs, new Comparator<String>(){
             //The value 0 if the argument is a string lexicographically equal to this string; a value less than 0 if the argument is a str
          public int compare(String s1, String s2){
            String leftRight = s1+s2;
            String rightLeft = s2+s1;
            //System.out.println(leftRight.compareTo(rightLeft));
            //less than zero if rightleft > leftright
            //bigger than zero fi leftright > rightleft
            //we can think compareTo as default as "<" whether is true
            return -leftRight.compareTo(rightLeft);

            //this means if the combination 1,2 is smaller than 2,1 then return 1 means put 1,2 after 2,1
            //means put 1 after 2
        //Note that the magnitude of the number doesn't matter. The aim isn't to say "how different" the two objects are, just in which dir
         }
    });

    StringBuilder sb = new StringBuilder();
    for(String s: strs){
        sb.append(s);
    }

    //delete the leading zeros if the number itself is not 0!!!!!!
    while(sb.charAt(0)=='0' && sb.length()>1){
        sb.deleteCharAt(0);
    }

    return sb.toString();
    }
}

Two strings are isomorphic if the characters in s can be replaced to get t.

All occurrences of a character must be replaced with another character while preserving the order of characters. No two characters may map

public class Solution {
    public boolean isIsomorphic(String s, String t) {
        //this is a one-to-one relationship!!!1
        //we could use two maps!!!
        public static boolean check(String s,String t){
        if(s.length()!=t.length()) return false;
        HashMap<Character,Character> map1=new HashMap<Character, Character>();
        HashMap<Character,Character> map2=new HashMap<Character, Character>();

        for(int i=0;i<s.length();i++){
            char c1=s.charAt(i);
            char c2=t.charAt(i);
            if(map1.containsKey(c1)){
                if(map1.get(c1)!=c2) return false;
            }
            if(map2.containsKey(c2)){
                if(map2.get(c2)!=c1) return false;
            }

            map1.put(c1, c2);
            map2.put(c2, c1);
        }
        return true;
    }
 ----------------------or we could use one map-----------------------------
        if(s.length()!= t.length()){
            return false;
        }

        HashMap<Character, Character> dict = new HashMap<>();
        for(int i=0; i<s.length(); i++){
            if(!dict.containsKey(t.charAt(i)) /*&& !dict.containsValue(s.charAt(i))*/){
                if(dict.containsValue(s.charAt(i))){
                    return false;
                }
                dict.put(t.charAt(i), s.charAt(i));
                //dict.put(s.charAt(i), t.charAt(i));
            }else{
                if(dict.get(t.charAt(i)) != s.charAt(i)){
                    return false;
                }
            }
        }
        return true;
    }
}

isSubtree
The approach is fundamentally flawed. If youre going to do it this way, you need two methods:

public boolean equals(Node n1, Node n2) {
    if (n1 == n2) return true; here we are using  == strong equal means the reference needs to be equal
    we should not use .equals() here
    if (n1 == null || n2 == null) return false;
    if (n1.data != n2.data) return false; // Should use .equals if Node.data isn't primitive
    return equals(n1.left, n2.left) && equals(n1.right, n2.right);
}

public boolean isSubtree(Node n1, Node n2) {
    if (n2 == null) return true;
    if (n1 == null) return false;
    return equals(n1, n2) || isSubtree(n1.left, n2) || isSubtree(n1.right, n2);
}
```