
 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 / **Summary-CalNums.java Find file Copy path

 huyilong SUMMARY - ALL numbers cal Problems 20180fd 22 seconds ago

1 contributor

488 lines (419 sloc) 15.5 KB Raw Blame History

```
1 利用递归解决子问题。对于每个段内数来说，最多3位最少1位，
2 所以在每一层可以循环3次，来尝试填段。因为IP地址最多4个分段，
3 当层数是3的时候说明已经尝试填过3个段了，那么把剩余没填的数段接到结尾即可。
4
5 这个过程中要保证的是填的数是合法的，最后拼接的剩余的数也是合法的。
6
7 注意开头如果是0的话要特殊处理，如果开头是0，判断整个串是不是0，
8 不是的话该字符就是非法的。因为001，01都是不对的。
9
10 public class Solution {
11     public List<String> restoreIpAddresses(String s) {
12         ArrayList<String> res = new ArrayList<String>();
13         String item = new String();
14         if (s.length()<4||s.length()>12)
15             return res;
16
17         dfs(s, 0, item, res);
18         return res;
19     }
20
21     public void dfs(String s, int start, String item, ArrayList<String> res){
22         if (start == 3 && isValid(s)) {
23             res.add(item + s);
24             return;
25         }
26         for(int i=0; i<3 && i<s.length()-1; i++){
27             String substr = s.substring(0,i+1);
28             if (isValid(substr))
29                 dfs(s.substring(i+1, s.length()), start+1, item + substr + '.', res);
30         }
31     }
32
33     public boolean isValid(String s){
34         if (s.charAt(0)=='0')
35             return s.equals("0");
36         int num = Integer.parseInt(s);
37
38         if(num <= 255 && num > 0)
39             return true;
40         else
41             return false;
42     }
43 }
44
45 public class Solution {
46     public boolean isHappy(int n) {
47         repeat the process until the number equals 1 (where it will stay),
48         or it loops endlessly in a cycle which does not include 1.
49
50
51         Set<Integer> set = new HashSet<>();
52         int next = nextNum(n);
53         while (!set.contains(next)) {
54             // repeat the process until the number equals 1 (where it will stay), or it loops endlessly in a cycle which does not include 1
55             if (next == 1) return true;
56             set.add(next);
57             next = nextNum(next);
58         }
59         return false;
60     }
61
62     public int nextNum(int n) {
```

```

63     int num = 0;
64     //calculate the sum of each number at each position's square ^2
65     while (n > 0) {
66         //get one bit at a time
67         int last = n % 10;
68         //calculate and accumulate the x^2
69         num += last * last;
70         //remove one bit after done the x^2
71         n = n / 10;
72     }
73     return num;
74 }
75 }
76
77
78 public class Solution {
79     public int myAtoi(String str) {
80         //white space at first then +/-
81         //then illegal conditions "."
82         if(str == null || str.length() == 0){
83             return 0;
84         }
85
86         String trim = str.trim();
87         int len = trim.length();
88         //+ or -
89         int sign=1;
90         int i=0;
91         //we need use ' ' and == for checking char
92         //we need to use .equals("string")
93         if(trim.charAt(0) == '+'){
94             i++;
95         }else if(trim.charAt(0) == '-'){
96             sign = -1;
97             i++;
98         }
99
100
101         //set as long to avoid overflow
102         long result = 0;
103         while(i<len){
104             if(trim.charAt(i) < '0' || trim.charAt(i) > '9'){
105                 break;
106             }
107
108             //////////////////////////////////////
109             result = 10*result + sign*(trim.charAt(i) - '0');
110
111             if(result > Integer.MAX_VALUE){
112                 return Integer.MAX_VALUE;
113             }else if(result < Integer.MIN_VALUE){
114                 return Integer.MIN_VALUE;
115             }
116
117             i++;
118         }
119
120
121         //cant just return sign*(int)result
122         return (int)result;
123     }
124 }
125
126
127 public class Solution {
128     public int reverse(int x) {
129         if(x==Integer.MIN_VALUE) return 0;
130         /*初看这道题觉得先将其转换为字符串然后转置以下就好了，但是仔细一想这种方法存在两种缺陷，一是负号需要单独处理，而是转置后开头的0也需要处理。另一种
131         int reverse = 0;
132         int abs = Math.abs(x);
133         System.out.println("123");
134         //count the length of number 123 ->3
135         // n % 10 ==> the last digit ->3
136         //we need to mod 10 to get last digit of each number
137         //int max_diff = Integer.MAX_VALUE / 10;
138         while(abs!=0){
139             //so we need to check each turn whether it is overflow
140             //int last = ;
141             if(reverse > Integer.MAX_VALUE / 10) return 0;
142             //System.out.println(last);

```

```

143         reverse = reverse*10 + abs%10;
144         // if(reverse > Integer.MAX_VALUE){
145         //     return 0;//overflow
146         // }
147
148         ////not x = x%10!!!!
149         //bug free!!!!
150         abs= abs/10;
151     }
152
153     return (x<0)? -1*reverse : reverse;
154 }
155 }
156 public class Solution {
157     public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
158         //What if the digits are stored in regular order instead of reversed order?
159         //We can simple reverse the list, calculate the result, and reverse the result.
160         //we need to think about carry!!!
161         //we need to be careful
162         //because we need to think that if there is a carry in the end we need to create a new head appending in the beginning
163
164
165         //this is easy because -- The digits are stored in reverse order and each of their nodes contain a single digit. and we do not need
166         int carry = 0;
167         int digit = 0;
168         //we need to create a new linkedlist from the scratch to return the result
169         ListNode head = null;
170         //we need to create pre because we finally need to keep head at first and not use head to going through and carrying on
171         ListNode scanner = null;
172         //we need to take care of the conditions when there is no node created yet!!!
173         while(l1!=null && l2!=null){
174             //we go through the two lists at the same time
175             digit = (l1.val + l2.val + carry)%10;
176             //carry is always int and not be a node created ever
177             carry = (l1.val + l2.val + carry)/10;
178             ListNode result = new ListNode(digit);
179             if(head == null){
180                 head =result;
181             }else{
182                 //there is a head
183                 scanner.next = result;
184             }
185             //after we link the result we need to move scanner up toward there
186             scanner = result;
187             l1 = l1.next;
188             l2 = l2.next;
189         }
190
191         //we need to think about what if there is one remaining
192         while(l1!=null){
193             digit = (l1.val+carry)%10;
194             carry = (l1.val + carry)/10;
195             ListNode result = new ListNode(digit);
196             if(head == null){
197                 head =result;
198             }else{
199                 //there is a head
200                 scanner.next = result;
201             }
202             scanner = result;
203             l1 = l1.next;
204         }
205
206         while(l2!=null){
207             digit = (l2.val+carry)%10;
208             carry = (l2.val + carry)/10;
209             ListNode result = new ListNode(digit);
210             if(head == null){
211                 head =result;
212             }else{
213                 //there is a head
214                 scanner.next = result;
215             }
216             scanner = result;
217             l2 = l2.next;
218         }
219         //finally if there is still a carry after traversing two linked list
220         if(carry>0){
221             ListNode remained = new ListNode(carry);
222             scanner.next = remained;

```

```

223     }
224     return head;
225 }
226 }
227
228 public class Solution {
229     public String addBinary(String a, String b) {
230         if(a==null || a.length()==0) return b;
231         if(b==null || b.length()==0) return a;
232
233         //if two string are not equal we need to padding the string with 0s
234         StringBuilder sb = new StringBuilder();
235         int alen = a.length();
236         int blen = b.length();
237         //carry should be declared outside the loop - global
238         int carry =0;
239         for(int ia = alen-1, ib=blen-1; ia>=0 || ib>=0 ; ia--,ib--){
240             int anum = (ia<0)?0:a.charAt(ia) - '0';//if a is shorter then ia<0 then padding with 0
241             int bnum = (ib<0)?0:b.charAt(ib) - '0';
242             //int val = (anum+bnum)%2;
243             //carry = carry +
244
245             int val = (anum + bnum + carry)%2;
246             carry = (anum + bnum + carry)/2;
247             sb.append(val);
248         }
249
250         if(carry ==1){
251             sb.append(1);
252         }
253         return sb.reverse().toString();
254     }
255 }
256
257 public class Solution {
258     public int[] plusOne(int[] digits) {
259         int len= digits.length;
260         int carry =1;
261         int digit=0;
262         for(int i = len-1; i>=0; i--){
263             digit = (digits[i] + carry)%10;
264             carry = (digits[i] +carry)/10;
265             digits[i] = digit;
266             System.out.println("1231");
267             System.out.println(carry);
268         }
269
270         if(carry>0){
271             System.out.println("123");
272             int[] expand = new int[len+1];
273             expand[0] = carry;
274             for(int j=1; j<len+1; j++){
275                 expand[j] = digits[j-1];
276             }
277             return expand;
278         }
279         return digits;
280     }
281 }
282 }
283
284 public class Solution {
285     public void merge(int[] nums1, int m, int[] nums2, int n) {
286         //You may assume that nums1 has enough space (size that is greater or equal to m + n)
287         int index1 = m-1, index2 = n-1, index = m+n-1;
288         while(index1>=0 && index2>=0){
289             //we need to compare from last to the first
290             if(nums1[index1] > nums2[index2]){
291                 nums1[index--] = nums1[index1--];
292             }else{
293                 nums1[index--] = nums2[index2--];
294             }
295         }
296     }
297
298     while(index1>=0){
299         nums1[index--] = nums1[index1--];
300     }
301     while(index2>=0){
302         nums1[index--] = nums2[index2--];
303     }

```

```

303     }
304 }
305
306 }
307
308
309 public class Solution {
310     public int addDigits(int num) {
311         // Given num = 38, the process is like: 3 + 8 = 11, 1 + 1 = 2. Since 2 has only one digit, return it.
312
313         while (num >= 10) {
314             //new num will store the result from the new equation
315             int newNum = 0;
316
317             while (num > 0) {
318                 newNum += num % 10;
319                 num /= 10;
320             }
321             num = newNum;
322         }
323         return num;
324     }
325 }
326
327 }
328
329 public class Solution {
330     public int[] productExceptSelf(int[] nums) {
331         //Solve it without division and in O(n).
332         int[] result = new int[nums.length];
333         int[] t1 = new int[nums.length];
334         int[] t2 = new int[nums.length];
335
336         //set the first element and the last one both to 1
337         t1[0] = 1;
338         t2[nums.length - 1] = 1;
339
340         for(int i=0; i<nums.length-1;i++){
341             t1[i+1] = nums[i] * t1[i];
342         }
343
344         for(int i=nums.length-1; i>0; --i){
345             t2[i-1] = t2[i] * nums[i];
346         }
347         for(int i=0; i<nums.length; i++){
348             result[i] = t1[i] * t2[i];
349         }
350
351         return result;
352     }
353 }
354
355 public class Solution {
356     public double myPow(double x, int n) {
357         // if (x == 0) return 0;
358         // if (n == 0) return 1;
359         // if(n<0){
360         //     //turn it into positive number
361         //     return 1 / myPow(x, -n);
362         // }
363
364         // return x * myPow(x, n - 1);
365         //接下来我们介绍二分法的解法，如同我们在Sqrt(x)的方法。不过这道题用递归来解比较容易理解，把x的n次方划分成两个x的n/2次方相乘，然后递归求解子问题
366         if(n == 0){
367             return 1.0;
368         }
369
370         double half = myPow(x, n/2);
371         //x^n/2 * x^n/2 = x^n
372         if(n%2==0){
373             return half * half;
374         }
375         //n cannot be divided by 2 which means it is 3
376         else if(n>0){
377             //(n+1)/2 + (n+1)/2 = half * half * x = which means add one
378             return x*half*half;
379         }
380         else{
381             return (half*half)/x;
382         }
383     }
384 }

```

```

383
384
385     }
386
387 public class Solution {
388     public int mySqrt(int x) {
389         //这是一道数值处理的题目，和Divide Two Integers不同，这道题一般采用数值中经常用的另一种方法：二分法。基本思路是跟二分查找类似，要求是知道结果的
390         if(x<0) return -1;
391         if(x==0) return 0;
392
393         int l=1;
394         int r=x/2+1;
395         while(l<=r){
396             //calculate the middle of the number
397             int m = (l+r)/2;
398
399             if(m<=x/m && x/(m+1) < m+1)
400                 return m;
401
402
403             if(x/m<m){
404                 //the target is less than middle
405                 r = m-1;
406             }else/* if(x/m > m)*/{
407                 //the target is larger than middle
408                 l = m+1;
409             }
410             // }else{
411             //     return m;
412             // }
413         }
414
415         //if not found
416         return 0;
417     }
418 }
419
420 public class Solution {
421     public boolean isPowerOfTwo(int n) {
422         //power of 2 -- then the bit manipulation must just have only one 1
423         //all others are 0s
424         //x-1&x == 0
425
426         if(n<1)
427             return false;
428         // if(n==1){
429         //     return true;
430         // }
431
432         return (n&(n-1))==0;
433         // System.out.println(n<<31);
434         // return (n<<31 == 0)? true:false;
435
436         //if a number's binary is power of 2 -- not power!!!
437         //then the bit representation should be the last bit -- 0
438         //if power of 8 // multiple ---000
439
440         //how to check whether x's last bit is 0?
441         //x & (x<<1) == 0?
442         // e.g 01    01<<1 -- 00    01&00 ==0
443     }
444 }
445
446 public class Solution {
447     public boolean isPalindrome(int x) {
448         // / define negative integers as non-palindromes.
449         if(x<0){
450             return false;
451         }
452         int check = x;
453         char[] a = Integer.toString(check).toCharArray();
454         int len = a.length;
455         for(int i= 0 ;i<=len/2; i++){
456             if(a[i] != a[len-i-1]){
457                 return false;
458             }
459         }
460         return true;
461     }
462 }

```

```
463 public boolean isPalindrome(int x) {
464     //do it without extra space
465     2     //negative numbers are not palindrome
466     3     if (x < 0)
467         4         return false;
468     5
469     6     // initialize how many zeros
470     7     int div = 1;
471     8     while (x / div >= 10) {
472         9         div *= 10;
473     10    }
474     11
475     12    while (x != 0) {
476         13        int left = x / div;
477         14        int right = x % 10;
478         15
479         16        if (left != right)
480             17            return false;
481         18
482         19        x = (x % div) / 10;
483         20        div /= 100;
484         21    }
485     22
486     23    return true;
487     24 }
```

