```
利用循环递归解决子问题。对于每个段内数来说，最多3位最少1位，
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，
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。

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。

 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，
 不是的话该字符就是非法的。因为001，01都是不对的。

public class Solution {
    public List<String> restoreIpAddresses(String s) {
        ArrayList<String> res = new ArrayList<String>();
        String item = new String();
        if (s.length()<4||s.length()>12)
        return res;

        dfs(s, 0, item, res);
        return res;
    }

    public void dfs(String s, int start, String item, ArrayList<String> res){
        if (start == 3 && isValid(s)) {
            res.add(item + s);
            return;
        }
        for(int i=0; i<3 && i<s.length()-1; i++){
            String substr = s.substring(0,i+1);
            if (isValid(substr))
                dfs(s.substring(i+1, s.length()), start+1, item + substr + '.', res);
        }
    }

    public boolean isValid(String s){
        if (s.charAt(0)=='0')
            return s.equals("0");
            int num = Integer.parseInt(s);

        if(num <= 255 && num > 0)
            return true;
        else
            return false;
    }

}

public class Solution {
    public boolean isHappy(int n) {
        repeat the process until the number equals 1 (where it will stay),
        or it loops endlessly in a cycle which does not include 1.

        Set<Integer> set = new HashSet<>();
        int next = nextNum(n);
        while (!set.contains(next)) {
            // repeat the process until the number equals 1 (where it will stay), or it loops endlessly in a cycle which does not include 1
            if (next == 1) return true;
            set.add(next);
            next = nextNum(next);
        }
        return false;
    }

    public int nextNum(int n) {
```

```java
        int num = 0;
        //calculate the sum of each number at each position's square ^2
        while (n > 0) {
            //get one bit at a time
            int last = n % 10;
            //calcualte and accumulate the x^2
            num += last * last;
            //remove one bit after done the x^2
            n = n / 10;
        }
        return num;
    }
}


public class Solution {
    public int myAtoi(String str) {
        //white space at first then +/-
        //then illegal conditions "."
        if(str == null || str.length() == 0){
            return 0;
        }

        String trim = str.trim();
        int len =trim.length();
        //+ or -
        int sign=1;
        int i=0;
        //we need use ' ' and  == for checking char
        //we need to use .equals("string")
        if(trim.charAt(0) == '+'){
            i++;
        }else if(trim.charAt(0) == '-'){
            sign = -1;
            i++;
        }


        //set as long to avoid overflow
        long result = 0;
        while(i<len){
            if(trim.charAt(i) < '0' || trim.charAt(i) >'9'){
                break;
            }

            ////////////////////////////////////////////////////////////
            result = 10*result + sign*(trim.charAt(i) - '0');

            if(result > Integer.MAX_VALUE){
                return Integer.MAX_VALUE;
            }else if(result < Integer.MIN_VALUE){
                return Integer.MIN_VALUE;
            }

            i++;
        }


        //cant just return sign*(int)result
        return   (int)result;
    }
}


public class Solution {
    public int reverse(int x) {
        if(x==Integer.MIN_VALUE)  return 0;
        /*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
        int reverse = 0;
        int abs = Math.abs(x);
        System.out.println("!23");
        //count the length of number 123 ->3
        // n % 10 ==> the last digit ->3
        //we need to mod 10 to get last digist of each number
        //int max_diff = Integer.MAX_VALUE / 10;
        while(abs!=0){
            //so we need to check each turn whether it is overflow
            //int last = ;
            if(reverse > Integer.MAX_VALUE / 10) return 0;
            //System.out.println(last);
```

```
            reverse = reverse*10 + abs%10;
            // if(reverse > Integer.MAX_VALUE){
            //     return 0;//overflow
            // }

            /////not x = x%10!!!!!
            //bug free!!!!
            abs= abs/10;
        }

        return (x<0)? -1*reverse : reverse;
    }
}
public class Solution {
    public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
        //What if the digits are stored in regular order instead of reversed order?
        //We can simple reverse the list, calculate the result, and reverse the result.
        //we need to think about carry!!!
        //we need to be careful
        //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


        //this is easy because --  The digits are stored in reverse order and each of their nodes contain a single digit. and we do not nee
        int carry = 0;
        int digit = 0;
        //we need to create a new linkedlist from the scratch to return the result
        ListNode head = null;
        //we need to create pre because we finally need to keep head at first and not use head to going through and carrying on
        ListNode scanner = null;
        //we need to take care of the conditions when there is no node created yet!!!
        while(l1!=null && l2!=null){
            //we go through the two lists at the same time
            digit = (l1.val + l2.val + carry)%10;
            //carry is always int and not be a node created ever
            carry = (l1.val + l2.val + carry)/10;
            ListNode result = new ListNode(digit);
            if(head == null){
                head =result;
            }else{
                //there is a head
                scanner.next = result;
            }
            //after we link the result we need to move scanner up toward there
            scanner = result;
            l1 = l1.next;
            l2 = l2.next;
        }

        //we need to think about what if there is one remaining
        while(l1!=null){
            digit = (l1.val+carry)%10;
            carry = (l1.val + carry)/10;
            ListNode result = new ListNode(digit);
            if(head == null){
                head =result;
            }else{
                //there is a head
                scanner.next = result;
            }
            scanner = result;
            l1 = l1.next;
        }

        while(l2!=null){
            digit = (l2.val+carry)%10;
            carry = (l2.val + carry)/10;
            ListNode result = new ListNode(digit);
            if(head == null){
                head =result;
            }else{
                //there is a head
                scanner.next = result;
            }
            scanner = result;
            l2 = l2.next;
        }
        //finally if there is still a carry after traversing two linked list
        if(carry>0){
            ListNode remained = new ListNode(carry);
            scanner.next = remained;
```

```java
        }
        return head;
    }
}

public class Solution {
    public String addBinary(String a, String b) {
        if(a==null || a.length()==0) return b;
        if(b==null || b.length()==0) return a;

        //if two string are not equal we need to padding the string with 0s
        StringBuilder sb = new StringBuilder();
        int alen = a.length();
        int blen = b.length();
        //carry should be declared outside the loop - global
        int carry =0;
        for(int ia = alen-1, ib=blen-1; ia>=0 || ib>=0 ; ia--,ib--){
            int anum = (ia<0 )? 0:a.charAt(ia) - '0';///if a is shorter then ia<0 then padding with 0
            int bnum = (ib<0)?0:b.charAt(ib) - '0';
            //int val = (anum+bnum)%2;
            //carry = carry +

            int val = (anum + bnum + carry)%2;
            carry = (anum + bnum + carry)/2;
            sb.append(val);
        }

        if(carry ==1){
            sb.append(1);
        }
        return sb.reverse().toString();
    }
}

public class Solution {
    public int[] plusOne(int[] digits) {
        int len= digits.length;
        int carry =1;
        int digit=0;
        for(int i = len-1; i>=0; i--){
            digit = (digits[i] + carry)%10;
            carry = (digits[i] +carry)/10;
            digits[i] = digit;
            System.out.println("1231");
            System.out.println(carry);
        }

        if(carry>0){
            System.out.println("123");
            int[] expand = new int[len+1];
            expand[0] = carry;
            for(int j=1; j<len+1; j++){
                expand[j] = digits[j-1];
            }
            return expand;
        }
        return digits;

    }
}

public class Solution {
    public void merge(int[] nums1, int m, int[] nums2, int n) {
        //You may assume that nums1 has enough space (size that is greater or equal to m + n)
        int index1 = m-1, index2 = n-1, index = m+n-1;
        while(index1>=0 && index2>=0){
            //we need to compare from last to the first
            if(nums1[index1] > nums2[index2]){
                nums1[index--] = nums1[index1--];

            }else{
                nums1[index--] = nums2[index2--];
            }
        }

        while(index1>=0){
            nums1[index--] = nums1[index1--];
        }
        while(index2>=0){
            nums1[index--] = nums2[index2--];
        }
```

```java



    }
}

public class Solution {
    public int addDigits(int num) {
       // Given num = 38, the process is like: 3 + 8 = 11, 1 + 1 = 2. Since 2 has only one digit, return it.

        while (num >= 10) {
            //new num will store the result from the new equation
            int newNum = 0;


            while (num > 0) {
                newNum += num % 10;
                num /= 10;
            }
            num = newNum;
        }
        return num;

    }
}

public class Solution {
    public int[] productExceptSelf(int[] nums) {
        //Solve it without division and in O(n).
        int[] result = new int[nums.length];
        int[] t1 = new int[nums.length];
        int[] t2 = new int[nums.length];

        //set the first element and the last one both to 1
        t1[0] = 1;
        t2[nums.length -1]=1;

        for(int i=0; i<nums.length-1;i++){
            t1[i+1] = nums[i] * t1[i];
        }

        for(int i=nums.length-1; i>0; --i){
            t2[i-1] = t2[i] *nums[i];
        }
        for(int i=0; i<nums.length; i++){
            result[i] = t1[i] * t2[i];
        }

        return result;

    }
}
public class Solution {
    public double myPow(double x, int n) {
        // if (x == 0) return 0;
        // if (n == 0) return 1;
        // if(n<0){
        //     //turn it into positive number
        //     return 1 / myPow(x, -n);
        // }

        // return x * myPow(x, n - 1);
        //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
        if(n == 0){
            return 1.0;
        }

        double half = myPow(x, n/2);
        //x^n/2 * x^n/2 = x^n
        if(n%2==0){
            return half * half;
        }
        //n cannot be divided by 2 which means it is 3
        else if(n>0){
            //(n+1)/2 + (n+1)/2 = half * half * x = which means add one
            return x*half*half;
        }
        else{
            return (half*half)/x;
        }
```

```java


    }

public class Solution {
    public int mySqrt(int x) {
        //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
        if(x<0) return -1;
        if(x==0) return 0;

        int l=1;
        int r=x/2+1;
        while(l<=r){
            //calculate the middle of the number
            int m = (l+r)/2;

            if(m<=x/m && x/(m+1) < m+1)
                return m;


            if(x/m<m){
                //the target is less than middle
                r = m-1;
             }else/* if(x/m > m)*/{
                //the target is larger than middle
                l = m+1;
             }
            // }else{
            //     return m;
            // }
        }

        //if not found
        return 0;
    }
}
public class Solution {
    public boolean isPowerOfTwo(int n) {
        //power of 2 -- then the bit manipulation must just have only one 1
        //all others are 0s
        //x-1&x == 0

        if(n<1)
        return false;
        // if(n==1){
        //     return true;
        // }

        return (n&(n-1))==0;
        // System.out.println(n<<31);
        // return (n<<31 == 0)? true:false;

        //if a number's binary is power of 2 -- not power!!!!
        //then the bit representation should be the last bit -- 0
        //if power of 8 // mulptiple   ---000

        //how to check whether x's last bit is 0?
        //x & (x<<1) == 0?
        // e.g 01    01<<1 -- 00   01&00 ==0
    }
}

public class Solution {
    public boolean isPalindrome(int x) {
        // / define negative integers as non-palindromes.
        if(x<0){
            return false;
        }
        int check = x;
        char[] a = Integer.toString(check).toCharArray();
        int len = a.length;
        for(int i= 0 ;i<=len/2; i++){
            if(a[i] != a[len-i-1]){
                return false;
            }
        }
        return true;
    }

}
```

```
public boolean isPalindrome(int x) {
    //do it without extra space
        //negative numbers are not palindrome
        if (x < 0)
            return false;

        // initialize how many zeros
        int div = 1;
        while (x / div >= 10) {
            div *= 10;
        }

        while (x != 0) {
            int left = x / div;
            int right = x % 10;

            if (left != right)
                return false;

            x = (x % div) / 10;
            div /= 100;
        }

        return true;
    }
```