

Flexport Coding 面试

Board Game

接口文档

Token Card Games

Implement canPurchase and purchase

Discount with card owned

Gold color

四子棋

手机按键/电话号

IP

valid ip address

generate all valid ip

其他

Board Game

一个棋盘，啥也不说。两边对手，没有block

上面对手只能往下走，往斜对角走。

下面对手只能往上走，斜对角走。

题解：

1.getAllNextStep(char turn, char[][] board)

2.canMove(Position current, Position target)

3.move(Position current, Position target).

接口文档

1、getAllNextStep(boolean isNextPlayer)

获取黑棋或者白棋的当前状态所有可能可以走的步骤。

输入：布尔类型 true表示白棋子在右下角 false表示黑棋左上角

输出: List<int[]> allSteps list结构中包括所有的可能走的步骤

2、canMove(int[] current, int[] target, boolean isNextPlayer)

判断当前位置是否能够到达目标地址位置

输入: 当前位置和目标位置, 是黑棋还是白棋

输出: boolean 布尔值表示是否能够到达

3、move(int[] current, int[] target, boolean isNextPlayer)

移动到当前位置

输入: 当前位置和目标位置, 是黑棋还是白棋

输出: 无

例子:

board 当前状态

		JSON	复制代码
1	1 0 0 0		
2	0 1 0 0		
3	0 0 2 0		
4	0 0 0 2		

1、getAllNextStep(false)

output: [[2,0]]

2、canMove(...)

input: current [1,1] target[3,1]

output: true

3、move()


input: current [1,1] target[3,1]

output:

board:



JSON

 复制代码

```
1  1 0 0 0
2  0 1 0 0
3  1 0 2 0
4  0 1 0 2
```

```
1 public class BoardGame {
2
3     private char[][] board;
4     private char[][] tmpBoard;
5
6     public BoardGame(){
7
8     }
9
10    public void move(int[] current, int[] target, boolean isNextPlayer){
11        if (canMove(current,target,isNextPlayer)){
12            if (isNextPlayer){
13                tmpBoard[target[0]][target[1]] = '2';
14            }else {
15                tmpBoard[target[0]][target[1]] = '1';
16            }
17            board = tmpBoard.clone();
18        }else {
19            System.out.println("Target position couldn't arrival!");
20        }
21    }
22
23    public boolean canMove(int[] current, int[] target, boolean isNextPlayer){
24        tmpBoard = board.clone();
25        int x = current[0];
26        int y = current[1];
27        int targetX = target[0];
28        int targetY = target[1];
29        return moveNextStep(tmpBoard,x,y,targetX,targetY,isNextPlayer);
30    }
31
32    private boolean moveNextStep(char[][] newBoard,
33                                int x,int y,
34                                int targetX,int targetY,
35                                boolean isNextPlayer){
36        // 走出范围
37        // ||白棋: 已经走到x坐标小于目标坐标x了说明无法到达
38        // ||黑棋: 已经走到x坐标大于目标坐标x了说明无法到达
39        if (x==targetX&& y==targetY){
40            return true;
41        }
42        if (!inAreas(x,y) ||
43            (isNextPlayer&&(targetX>x)) ||
44            (!isNextPlayer&&(targetX<x))){
```

```

45
46         return false;
47     }
48
49     boolean ans = false;
50     char tmp = newBoard[x][y];
51     if (isNextPlayer){
52         // 白旗走法
53         if (newBoard[x][y] != '1'){
54             newBoard[x][y] = '2';
55             ans = moveNextStep(newBoard,x-1,y-1,targetX,targetY,isNex
56 tPlayer)||
57             moveNextStep(newBoard,x-1,y+1,targetX,targetY,isN
58 extPlayer);
59             if (!ans){
60                 newBoard[x][y] = tmp;
61             }
62         }else {
63             // 黑旗走法
64             if (newBoard[x][y] != '2'){
65                 newBoard[x][y] = '1';
66                 ans = moveNextStep(newBoard,x+1,y+1,targetX,targetY,isNex
67 tPlayer)||
68                 moveNextStep(newBoard,x+1,y-1,targetX,targetY,isN
69 extPlayer);
70                 if (!ans){
71                     newBoard[x][y] = tmp;
72                 }
73             }
74             return ans;
75         }
76
77     /**
78      *
79      * @param isNextPlayer 那个player的可能要走的steps
80      * @return
81      */
82     public List<int[]> getAllNextStep(boolean isNextPlayer){
83         List<int[]> allNextStep = new ArrayList<>();
84         if (isNextPlayer){
85             getNextStep(board.length-1, board.length-1, allNextStep,isNex
86 tPlayer);
87         }else {
88             getNextStep(0,0, allNextStep,isNextPlayer);

```

```

88     }
89     return allNextStep;
90 }
91
92 private void getNextStep(int i,int j,List<int[]> steps,boolean isNext
93 Player){
94     if (!inAreas(i,j)){
95         return;
96     }
97     if (board[i][j]=='0'){
98         steps.add(new int[]{i,j});
99         return;
100    }
101    if (isNextPlayer){
102        // 白旗走法
103        if (board[i][j]=='2'){
104            getNextStep(i-1,j-1,steps,isNextPlayer);
105            getNextStep(i-1,j+1,steps,isNextPlayer);
106        }
107    }else {
108        // 黑旗走法
109        if (board[i][j]=='1'){
110            getNextStep(i+1,j+1,steps,isNextPlayer);
111            getNextStep(i+1,j-1,steps,isNextPlayer);
112        }
113    }
114 }
115
116 private boolean inAreas(int i,int j){
117     return 0<=i&&i< board.length&&
118         0<=j&&j<board.length;
119 }
120
121 public static void printArr(char[][] board){
122     StringBuilder ans = new StringBuilder();
123     ans.append("[");
124     for (int i=0;i<board.length;i++){
125
126         for (int j=0;j<board.length;j++){
127             ans.append(board[i][j]);
128             if (j==board.length-1){
129                 ans.append(',');
130             };
131         }
132     }
133     ans.append("]");
134     System.out.println(new String(ans));

```

```

135     }
136
137     public static void main(String[] args) {
138         BoardGame bg = new BoardGame();
139         int n = 4;
140         bg.board = new char[][]{
141             {'1','0','0','0'},
142             {'0','1','0','0'},
143             {'0','0','2','0'},
144             {'0','0','0','2'}
145         };
146         List<int[]> black = bg.getAllNextStep(false);
147         System.out.println(JSONObject.toJSONString(black));
148         int[] current = {1,1};
149         int[] target = {3,1};
150         boolean ans = bg.canMove(current,target,false);
151         System.out.println(ans);
152
153         bg.move(current,target,false);
154         printArr(bg.board);
155     }
156 }
157

```

Token Card Games

Coding Card game, step by step. Money is represented in number of each color: black, blue, green, green, white. A card has properties of how much it requires by it. Implement a canPurchase() method to decide whether a certain amount of money can buy a card. Implement the purchase method, which should update the money and cards owned. Discount with card owned. Each card has a property of color. If you own cards of a color. Next time purchase of another card, the price of this color can be discounted by the number of card you owned. Update the canPurchase and purchase method.

Implement canPurchase and purchase

```
1  /**
2   * 1、使用map定义key是颜色，value是所需要的数量，然后对比card和player的对应颜色的数
   量是否一致问题
3   * 2、定义Tokens结构，从结构里面获取各个颜色的数量对比是否一致
4   * @Author: Twiss
5   * @Date: 2022/8/20 12:14 下午
6   */
7  public class TokensCardGames {
8
9      private Map<String,Integer> card;
10     private Map<String,Integer> player;
11     private Map<String,Integer> playerCards;
12
13     public TokensCardGames(){
14         this.card = new HashMap<>();
15         this.player = new HashMap<>();
16         this.playerCards = new HashMap<>();
17     }
18
19     public boolean canPurchase() {
20         Map<String,Integer> resultCard = new HashMap<>();
21         for (String tokenCard : player.keySet()) {
22             if (!card.containsKey(tokenCard)) {
23                 return false;
24             } else {
25                 resultCard.put(tokenCard, card.get(tokenCard) - player.get
   (tokenCard));
26             }
27         }
28         for (String tokenCard : resultCard.keySet()) {
29             if (resultCard.get(tokenCard) > 0) {
30                 return false;
31             }
32         }
33         return true;
34     }
35
36     public void purchase(Map<String,Map<String,Integer>> cards, Map<String
   ,Integer> player) {
37         for (String cardName:cards.keySet()){
38             card = cards.get(cardName);
39             if (canPurchase()){
40                 for (String tokenCard:player.keySet()){
41                     player.put(tokenCard,player.get(tokenCard)-card.get(to
   kenCard));
```



```

42         }
43         playerCards.put(cardName,playerCards.getOrDefault(cardName
44         ,0)+1);
45     }
46 }
47
48 public static void main(String[] args) {
49     TokensCardGames tg = new TokensCardGames();
50     Map<String,Map<String,Integer>> cards = new HashMap<>();
51     Map<String,Integer> cardA = new HashMap<>();
52     cardA.put("WH",2);
53     cardA.put("BL",1);
54     cardA.put("B",4);
55     cards.put("cardA",cardA);
56
57     Map<String,Integer> player = new HashMap<>();
58     player.put("WH",4);
59     player.put("BL",2);
60     player.put("B",5);
61     tg.player = player;
62     for (String cardName:cards.keySet()){
63         tg.card = cards.get(cardName);
64         boolean valid = tg.canPurchase();
65         System.out.println(valid);
66     }
67
68     tg.purchase(cards, player);
69     System.out.println(JSONObject.toJSONString(tg.playerCards));
70 }
71 }
72

```

Discount with card owned

Discount with card owned. Each card has a property of color. If you own cards of a color. Next time you purchase another card, the price of this color can be discounted by the number of cards you owned. Update the canPurchase and purchase method. (比如手里有三张 Red Card, 下一张待购买的卡片 cost 需要 N 个 Red Money, 实际购买时 只需要支付 N-3 个 Red Money 就行)

```
1 public class TokensCardGamesII {
2
3     private Map<String,Integer> card;
4     private Map<String,Integer> player;
5     private Map<String,Integer> playerCards;
6
7     public TokensCardGamesII(){
8         this.card = new HashMap<>();
9         this.player = new HashMap<>();
10        this.playerCards = new HashMap<>();
11    }
12
13    public boolean canPurchase() {
14        Map<String,Integer> resultCard = new HashMap<>();
15        for (String tokenCard : player.keySet()) {
16            if (card.containsKey(tokenCard)) {
17
18                resultCard.put(tokenCard, card.get(tokenCard)
19                    - player.get(tokenCard)
20                    -playerCards.getDefault(tokenCard,0));
21                for (String resultCardName : resultCard.keySet()) {
22                    if (resultCard.get(resultCardName) <= 0) {
23                        return true;
24                    }
25                }
26            }
27        }
28        return false;
29    }
30
31    public void purchase(Map<String,Map<String,Integer>> cards, Map<String
32    ,Integer> player) {
33        for (String cardName:cards.keySet()){
34            card = cards.get(cardName);
35            if (canPurchase()){
36                for (String tokenCard:card.keySet()){
37                    player.put(tokenCard,player.get(tokenCard)
38                        -card.get(tokenCard)
39                        +playerCards.getDefault(tokenCard,0));
40                }
41                playerCards.put(cardName,playerCards.getDefault(cardName
42    ,0)+1);
43            }
44        }
45    }
46 }
```

```

44
45 public static void print(Map<String,Integer> map){
46
47     StringBuilder res = new StringBuilder();
48     res.append("{");
49     int idx = 0;
50     for (String key:map.keySet()){
51         res.append("\n").append(key).append("\n").append(":").append(m
52 ap.get(key));
53         if (idx<map.size()-1){
54             res.append(",");
55         }
56         idx++;
57     }
58     res.append("}");
59     System.out.println(res);
60 }
61
62 /**
63  * 假设:
64  * card:{"Red":4}
65  * player:{"Red":6,"Blue":7}
66  * playerCards:{"Red":1}
67  * @param args
68  */
69 public static void main(String[] args) {
70     TokensCardGamesII tg = new TokensCardGamesII();
71     Map<String,Map<String,Integer>> cards = new HashMap<>();
72     Map<String,Integer> redCard = new HashMap<>();
73     redCard.put("Red",4);
74     cards.put("Red",redCard);
75
76     Map<String,Integer> player = new HashMap<>();
77     player.put("Red",7);
78     player.put("Blue",2);
79     player.put("Black",5);
80     tg.player = player;
81     for (String cardName: cards.keySet()){
82         tg.card = cards.get(cardName);
83         boolean valid = tg.canPurchase();
84         System.out.println(valid);
85     }
86     tg.purchase(cards, player);
87     print(tg.player);
88     print(tg.playerCards);
89     tg.purchase(cards, player);
90     print(tg.player);
91     print(tg.playerCards);

```

```
91     }  
92   }  
93
```

Gold color

第四问:Money 增加一种 Gold color, 万能色, 可以在买卡的时候冲抵任何一种颜色。比如 下一张待购买的卡片 cost 需要 N 个 Red Money。手里只有 N-1 个 Red Money, 但是有超过一个 Gold Money, 那也买得起

```

1 public class TokensCardGamesIII {
2
3     private Map<String,Integer> card;
4     private Map<String,Integer> player;
5     private Map<String,Integer> playerCards;
6     private final static String GOLD_COLOR = "Gold";
7
8     public TokensCardGamesIII(){
9         this.card = new HashMap<>();
10        this.player = new HashMap<>();
11        this.playerCards = new HashMap<>();
12    }
13
14    public boolean canPurchase() {
15        Map<String,Integer> resultCard = new HashMap<>();
16        for (String tokenCard : player.keySet()) {
17            if (card.containsKey(tokenCard)) {
18
19                resultCard.put(tokenCard, card.get(tokenCard)
20                    - player.get(tokenCard));
21                for (String resultCardName : resultCard.keySet()) {
22                    if (resultCard.get(resultCardName) <= 0) {
23                        return true;
24                    }else {
25                        if (player.containsKey(GOLD_COLOR)){
26                            int goldNums = player.get(GOLD_COLOR);
27                            while (goldNums>0){
28                                resultCard.put(resultCardName,resultCard.
29                                    get(resultCardName)-1);
30                                goldNums--;
31                                if (resultCard.get(resultCardName)<=0){
32                                    return true;
33                                }
34                            }
35                        }
36                    }
37                }
38            }
39            return false;
40        }
41
42        public void purchase(Map<String,Map<String,Integer>> cards, Map<String,Integer> player) {
43            for (String cardName:cards.keySet()){

```

```

44         card = cards.get(cardName);
45         if (canPurchase()){
46             for (String tokenCard:card.keySet()){
47                 int remainderCard = player.get(tokenCard)
48                     -card.get(tokenCard);
49                 if(remainderCard>=0){
50                     player.put(tokenCard,remainderCard);
51                 }else {
52                     player.put(tokenCard,0);
53                     player.put(GOLD_COLOR, player.get(GOLD_COLOR)+rem
54 ainderCard);
55                 }
56             }
57             playerCards.put(cardName,playerCards.getOrDefault(cardNam
58 e,0)+1);
59         }
60     }
61     public static void print(Map<String,Integer> map){
62
63         StringBuilder res = new StringBuilder();
64         res.append("{");
65         int idx = 0;
66         for (String key:map.keySet()){
67             res.append("\\"").append(key).append("\\"").append(":").append(
68 map.get(key));
69             if (idx<map.size()-1){
70                 res.append(",");
71             }
72             idx++;
73         }
74         res.append("}");
75         System.out.println(res);
76     }
77     /**
78      * 假设:
79      * card:{"Red":4}
80      * player:{"Red":6,"Blue":7}
81      * playerCards:{"Red":1}
82      * @param args
83      */
84     public static void main(String[] args) {
85         TokensCardGamesIII tg = new TokensCardGamesIII();
86         Map<String,Map<String,Integer>> cards = new HashMap<>();
87         Map<String,Integer> redCard = new HashMap<>();
88         redCard.put("Red",4);

```

```

89         cards.put("Red", redCard);
90
91         Map<String,Integer> player = new HashMap<>();
92         player.put("Red",6);
93         player.put("Blue",2);
94         player.put("Gold",5);
95         tg.player = player;
96         for (String cardName: cards.keySet()){
97             tg.card = cards.get(cardName);
98             boolean valid = tg.canPurchase();
99             System.out.println(valid);
100         }
101         tg.purchase(cards, player);
102         System.out.print("player:");
103         print(tg.player);
104         System.out.print("playerCards:");
105         print(tg.playerCards);
106         tg.purchase(cards, player);
107         System.out.print("player:");
108         print(tg.player);
109         System.out.print("playerCards:");
110         print(tg.playerCards);
111     }
112 }
113

```

四子棋

```

          R
R         B
R         B
-----
-4 -3 -2 -1 0  1  2  3  4

```

每次按照位置进行叠加，直到连续4个颜色一直为赢

put(color, pos) -> 返回bool true表示输入的颜色赢

可以使用

{"-4":"RR"}结构去做put

手机按键/电话号

给一个map, {"0":["a","b"], "1":["c"]}

和一个string, "001"

输出所有可能的组合["aac","abc","bac","bbc"]

我一开始就说遍历之后, 每个可能的都选一次, 然后把所有组成的string加入一个列表返回。当时觉得和permutation很想, 就说用backtracking。但是写的时候又写成了recursion 加string concatenation。而且recursion外面还套了个iteration。。。后来debug了好久, 屏幕都被print满了才发现recursion外面不应该用iteration, 二者取其一就行。于是改对了。此处用了30分钟。。。


```
1 public class LetterCombinations {
2
3     public LetterCombinations(){
4     }
5
6     public List<String> getCombinations(String digits){
7         List<String> combinations = new ArrayList<String>();
8         if (digits.length() == 0) {
9             return combinations;
10        }
11        Map<String,String> reflectMap = new HashMap<>();
12        reflectMap.put("0","ab");
13        reflectMap.put("1","c");
14        System.out.println(JSONObject.toJSONString(reflectMap));
15        backtrack(combinations,reflectMap,digits,0,new StringBuilder());
16        return combinations;
17    }
18
19    private void backtrack(List<String> combinations,
20                           Map<String,String> reflectMap,
21                           String digits,
22                           int index,
23                           StringBuilder path){
24        if (index>=digits.length()){
25            combinations.add(new String(path));
26        }else {
27            String digit = String.valueOf(digits.charAt(index));
28            String reflect = reflectMap.getOrDefault(digit,"");
29            int count = reflect.length();
30            for (int i=0;i<count;i++){
31                path.append(reflect.charAt(i));
32                backtrack(combinations,reflectMap,digits,index+1,
33                        path);
34                path.deleteCharAt(index);
35            }
36        }
37    }
38
39    public static void main(String[] args) {
40        String digits = "001";
41        List<String> ans = new LetterCombinations().getCombinations(digits
42    );
43        System.out.println(JSONObject.toJSONString(ans));
44    }
```

follow up

如果map的key不止一位，如加入一个"01":"z",

输出应该是["aac","abc","bac","bbc","az","bz"]

<https://leetcode.cn/problems/letter-combinations-of-a-phone-number/>

```
1 public class LetterCombinationsII {
2
3     public LetterCombinationsII(){
4     }
5
6     public List<String> getCombinations(String digits){
7         List<String> combinations = new ArrayList<String>();
8         if (digits.length() == 0) {
9             return combinations;
10        }
11        Map<String,String> reflectMap = new HashMap<>();
12        reflectMap.put("0","ab");
13        reflectMap.put("1","c");
14        reflectMap.put("01","z");
15        System.out.println(JSONObject.toJSONString(reflectMap));
16        backtrack(combinations,reflectMap,digits,0,new StringBuilder());
17        return combinations;
18    }
19
20    private void backtrack(List<String> combinations,
21                           Map<String,String> reflectMap,
22                           String digits,
23                           int index,
24                           StringBuilder path){
25        if (index>=digits.length()){
26            combinations.add(new String(path));
27        }else {
28            for (int idx = 1;idx<digits.length()-index+1;idx++){
29                String digit = digits.substring(index,index+idx);
30                String reflect = reflectMap.getOrDefault(digit,"");
31                int count = reflect.length();
32                for (int i=0;i<count;i++){
33                    path.append(reflect.charAt(i));
34                    backtrack(combinations,reflectMap,digits,index+idx,
35                             path);
36                    path.deleteCharAt(index);
37                }
38            }
39        }
40    }
41
42    public static void main(String[] args) {
43        String digits = "001";
44        List<String> ans = new LetterCombinationsII().getCombinations(digits);
```

```
45         System.out.println(JSONObject.toJSONString(ans));  
46     }  
47 }
```

IP

valid ip address

IP V4:

- 1、个数是不是4个
- 2、每个长度不在 1-3之间
- 3、有前导0 并且长度不为1
- 4、计算数字

不是数字

数字超过255的

IP V6:

- 1、数量不是8个
- 2、每个串 长度不在1-4之间
- 3、遍历

不是数字并且字母不在 a-f之间

```
1 public class ValidIp {
2
3     public String validIPAddress(String queryIP) {
4         if (queryIP.indexOf('.') >= 0) {
5             return isIPv4(queryIP) ? "IPv4" : "Neither";
6         } else {
7             return isIPv6(queryIP) ? "IPv6" : "Neither";
8         }
9     }
10
11     public boolean isIPv4(String queryIP) {
12         //加-1是防止出现空字符串无法计数 比如192.168.1.1. 后边多了一个点, 不加-1会
        被忽略后边的空串
13         String[] split = queryIP.split("\\.", -1);
14         //个数不是4个
15         if (split.length != 4) {
16             return false;
17         }
18         for (String s : split) {
19             //每个长度不在 1-3之间
20             if (s.length() > 3 || s.length() == 0) {
21                 return false;
22             }
23             //有前导0 并且长度不为1
24             if (s.charAt(0) == '0' && s.length() != 1) {
25                 return false;
26             }
27             //计算数字
28             int ans = 0;
29             for (int j = 0; j < s.length(); j++) {
30                 char c = s.charAt(j);
31                 //不是数字
32                 if (!Character.isDigit(c)) {
33                     return false;
34                 }
35                 ans = ans * 10 + (c - '0');
36             }
37             //数字超过255
38             if (ans > 255) {
39                 return false;
40             }
41         }
42         return true;
43     }
44 }
```

```

45     public boolean isIPv6(String queryIP) {
46         String[] split = queryIP.split(":",-1);
47         //数量不是8个
48         if (split.length != 8) {
49             return false;
50         }
51         for (String s : split) {
52             //每个串 长度不在1-4之间
53             if (s.length() > 4 || s.length() == 0) {
54                 return false;
55             }
56             for (int i = 0; i < s.length(); i++) {
57                 char c = s.charAt(i);
58                 //不是数字并且字母不在 a-f之间
59                 if (!Character.isDigit(c) && !(Character.toLowerCase(c) >=
60 'a')
61                 || !(Character.toLowerCase(c) <= 'f')) {
62                     return false;
63                 }
64             }
65             return true;
66         }
67     }
68     public static void main(String[] args) {
69         String ip= "192.168.1.1";
70         String ans = new ValidIp().validIPAddress(ip);
71         System.out.println(ans);
72     }
73 }
74

```

generate all valid ip

给一个数字string, 它可以组成多少个valid的 IP, 输出 eg: input "00123" output: "0.0.1.23" "0.0.12.3"

```
1 public class RestoreIp {
2     public List<String> restoreIpAddresses(String s) {
3         int len = s.length();
4         List<String> res = new ArrayList<>();
5         // 如果长度不够, 不搜索
6         if (len < 4 || len > 12) {
7             return res;
8         }
9
10        Deque<String> path = new ArrayDeque<>(4);
11        int splitTimes = 0;
12        dfs(s, len, splitTimes, 0, path, res);
13        return res;
14    }
15
16    /**
17     * 判断 s 的子区间 [left, right] 是否能够成为一个 ip 段
18     * 判断的同时顺便把类型转了
19     *
20     * @param s
21     * @param left
22     * @param right
23     * @return
24     */
25    private int judgeIfIpSegment(String s, int left, int right) {
26        int len = right - left + 1;
27
28        // 大于 1 位的时候, 不能以 0 开头
29        if (len > 1 && s.charAt(left) == '0') {
30            return -1;
31        }
32
33        // 转成 int 类型
34        int res = 0;
35        for (int i = left; i <= right; i++) {
36            res = res * 10 + s.charAt(i) - '0';
37        }
38
39        if (res > 255) {
40            return -1;
41        }
42        return res;
43    }
44
45 }
```

```

46     private void dfs(String s, int len, int split, int begin, Deque<String
47 > path, List<String> res) {
48         if (begin == len) {
49             if (split == 4) {
50                 res.add(String.join(".", path));
51             }
52             return;
53         }

54         // 看到剩下的不够了, 就退出 (剪枝), len - begin 表示剩余的还未分割的字符串
55         的位数
56         if (len - begin < (4 - split) || len - begin > 3 * (4 - split)) {
57             return;
58         }

59         for (int i = 0; i < 3; i++) {
60             if (begin + i >= len) {
61                 break;
62             }

63             int ipSegment = judgeIfIpSegment(s, begin, begin + i);
64             if (ipSegment != -1) {
65                 // 在判断是 ip 段的情况下, 才去做截取
66                 path.addLast(ipSegment + "");
67                 dfs(s, len, split + 1, begin + i + 1, path, res);
68                 path.removeLast();
69             }
70         }
71     }
72 }
73
74 public static void main(String[] args) {
75     String words = "00123";
76     List<String> ans = new RestoreIp().restoreIpAddresses(words);
77     System.out.println(JSONObject.toJSONString(ans));
78 }
79
80 }

```

其他

LC17, LC 62, LC91, LC93, LC468, LC469.