王争的算法训练营

习题课：回溯

习题课：回溯

王争的算法训练营

配套习题（15）：

面试题 08.12. 八皇后（困难）
37. 解数独
17. 电话号码的字母组合 （中等）

77. 组合 （中等） 给n个数返回所有k个数的组合
78. 子集（中等） 所有的组合
90. 子集 II（中等）有重复数据
46. 全排列 （中等） 所有排列
47. 全排列 II（中等） 有重复数据

39. 组合总和 （中等） 选出某几个数相加为给定和，无重复数据，可以使用多次，不能有重复答案
40. 组合总和 II（中等）选出某几个数相加为给定和，有重复数据，只能使用一次，不能有重复答案
216. 组合总和 III（中等） 选出k个数相加为给定和，没有重复数据，只能使用一次

131. 分割回文串（中等）
93. 复原 IP 地址（中等）
22. 括号生成（中等）

回溯相关题型/套路

回溯是重点，常考到，并且是另外两个重点：DFS和动态规划的基础。因为回溯用到递归，比较绕，入门很难。入门之后就很简单，因为有模板。

回溯的核心思想：

- 回溯的处理过程是一个**穷举（或者叫枚举）**的过程。枚举所有的解，找出其中满足期望的可行解。为了有规律地枚举所有可能的解，避免遗漏和重复，我们把问题求解的过程归纳为**多阶段决策模型**。每个阶段的决策会对应多个选择，从可选的选择列表中，任意选择一个，然后继续进行下一个阶段的决策。
- 整个决策的过程，如果用图来形象话表示的话，就是一棵**决策树**。回溯穷举所有解来查找可行解的过程，就是**在决策树中进行遍历**的过程。遍历过程中记录的**路径**就是解。
- 回溯一般使**用递归来实现**，递归树就跟决策树完全一样。递的过程进行函数调用，对应到递归树上为从一个节点进入它的子节点，归的过程进行函数调用返回，对应到递归树上是从子节点返回上一层节点。

回溯相关题型/套路

王争的算法训练营

```
#### 回溯代码模板####
result = [ ]
def backtrack(可选列表，决策阶段，路径)
    if 满足结束条件（所有决策都已完成或得到可行解）
        if 路径为可行解: result.add(路径)
        return

    for 选择 in [可选列表]:
        # 做选择
        路径.add(选择)
        backtrack(可选列表，决策阶段+1， 路径)
        # 撤销选择
        路径.remove(选择)
```

```
#### 递归代码模板 ######
def recur(参数) {
    递归结束条件 # 最小子问题
    …前置逻辑...
    recur(参数); #子问题
    是否有现场需要手动恢复（全局变量）
    ...后置逻辑...
}
```

习题课：回溯

王争的算法训练营

配套习题（15）：

面试题 08.12. 八皇后（困难）

37. 解数独

17. 电话号码的字母组合 （中等）

习题课：回溯

面试题 08.12. 八皇后（困难）

设计一种算法，打印 N 皇后在 N × N 棋盘上的各种摆法，其中每个皇后都不同行、不同列，也不在对角线上。这里的“对角线”指的是所有的对角线，不只是平分整个棋盘的那两条对角线。

注意：本题相对原题做了扩展

示例:

```
 输入：4
 输出：[[".Q..","...Q","Q...","..Q."],["..Q.","Q...","...Q",".Q.."]]
 解释: 4 皇后问题存在如下两个不同的解法。
[
 [".Q..",  // 解法 1
  "...Q",
  "Q...",
  "..Q."],

 ["..Q.",  // 解法 2
  "Q...",
  "...Q",
  ".Q.."]
]
```

```java
class Solution {
  private List<List<String>> result = new ArrayList<>();
  public List<List<String>> solveNQueens(int n) {
    char[][] board = new char[n][n];
    for (int i = 0; i < n; ++i) {
      for (int j = 0; j < n; ++j) {
        board[i][j] = '.';
      }
    }
    backtrack(0, board, n);
    return result;
  }

  // row: 阶段
  // board: 路径，记录已经做出的决策
  // 可选列表：通过board推导出来，没有显式记录
  private void backtrack(int row, char[][] board, int n) {
    // 结束条件，得到可行解
    if (row == n) {
      List<String> snapshot = new ArrayList<>();
      for (int i = 0; i < n; ++i) {
        snapshot.add(new String(board[i]));
      }
      result.add(snapshot);
      return;
    }

    for (int col = 0; col < n; ++col) { //每一行都有n中放法
      if (isOk(board, n, row, col)) { //可选列表
        board[row][col] = 'Q'; //做选择，第row行的棋子放到了col列
        backtrack(row + 1, board, n); //考察下一行
        board[row][col] = '.'; //恢复选择
      }
    }
  }
}
```

```java
//判断row行column列放置是否合适
 private boolean isOk(char[][] board, int n, int row, int col) {
   // 检查列是否有冲突
   for (int i = 0; i < n; i++) {
     if (board[i][col] == 'Q') {
       return false;
     }
   }
   // 检查右上对角线是否有冲突
   int i = row - 1;
   int j = col + 1;
   while (i >= 0 && j < n) {
     if (board[i][j] == 'Q') {
       return false;
     }
     i--;
     j++;
   }
   // 检查左上对角线是否有冲突
   i = row - 1;
   j = col - 1;
   while (i >= 0 && j >= 0) {
     if (board[i][j] == 'Q') {
       return false;
     }
     i--;
     j--;
   }
   return true;
}
```

习题课：回溯

王争的算法训练营

37. 解数独

编写一个程序，通过填充空格来解决数独问题。

数独的解法需 **遵循如下规则**：

1. 数字 `1-9` 在每一行只能出现一次。
2. 数字 `1-9` 在每一列只能出现一次。
3. 数字 `1-9` 在每一个以粗实线分隔的 `3x3` 宫内只能出现一次。（请参考示例图）

数独部分空格内已填入了数字，空白格用 `'.'` 表示。

示例：

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

```java
class Solution {
    private boolean[][] rows = new boolean[9][10];
    private boolean[][] cols = new boolean[9][10];
    private boolean[][][] blocks = new boolean[3][3][10];
    private boolean solved = false;
    public void solveSudoku(char[][] board) {
        for (int i = 0; i < 9; ++i) {
            for (int j = 0; j < 9; ++j) {
                if (board[i][j] != '.') {
                    int num = board[i][j] - '0';
                    rows[i][num] = true;
                    cols[j][num] = true;
                    blocks[i/3][j/3][num] = true;
                }
            }
        }
        backtrack(0, 0, board);
    }
```

```java
private void backtrack(int row, int col, char[][] board) {
    if (row == 9) {
        solved = true;
        return;
    }
    if (board[row][col]!='.') {
        int nextRow = row;
        int nextCol = col+1;
        if (col == 8) {
            nextRow = row+1;
            nextCol = 0;;
        }
        backtrack(nextRow, nextCol, board);
        if (solved) return;
    } else {
        for (int num = 1; num <= 9; ++num) {
            if (!rows[row][num] && !cols[col][num] && !blocks[row/3][col/3][num]) {
                board[row][col] = String.valueOf(num).charAt(0); // 数字转化成字符
                rows[row][num] = true;
                cols[col][num] = true;
                blocks[row/3][col/3][num] = true;
                int nextRow = row;
                int nextCol = col+1;
                if (col == 8) {
                    nextRow = row+1;
                    nextCol = 0;;
                }
                backtrack(nextRow, nextCol, board);
                if (solved) return;
                board[row][col] = '.';
                rows[row][num] = false;
                cols[col][num] = false;
                blocks[row/3][col/3][num] = false;
            }
        }
    }
}
```

习题课：回溯

王争的算法训练营

17. 电话号码的字母组合 （中等）

给定一个仅包含数字 `2-9` 的字符串，返回所有它能表示的字母组合。答案可以按 **任意顺序** 返回。

给出数字到字母的映射如下（与电话按键相同）。注意 1 不对应任何字母。

示例 1：

```
输入：digits = "23"
输出：["ad","ae","af","bd","be","bf","cd","ce","cf"]
```

示例 2：

```
输入：digits = ""
输出：[]
```

示例 3：

```
输入：digits = "2"
输出：["a","b","c"]
```

```java
class Solution {
    private List<String> result = new ArrayList<>();
    public List<String> letterCombinations(String digits) {
        if (digits.length() == 0) return Collections.emptyList();
        String[] mappings = new String[10];
        mappings[2] = "abc";
        mappings[3] = "def";
        mappings[4] = "ghi";
        mappings[5] = "jkl";
        mappings[6] = "mno";
        mappings[7] = "pqrs";
        mappings[8] = "tuv";
        mappings[9] = "wxyz";
        char[] path = new char[digits.length()];
        backtrack(mappings, digits, 0, path);
        return result;
    }

    // k表示阶段
    // path路径
    // digits[k]+mappings确定当前阶段的可选列表
    private void backtrack(String[] mappings, String digits, int k, char[] path) {
        if (k == digits.length()) {
            result.add(new String(path));
            return;
        }
        String mapping = mappings[digits.charAt(k)-'0'];
        for (int i = 0; i < mapping.length(); ++i) {
            path[k] = mapping.charAt(i);
            backtrack(mappings, digits, k+1, path);
        }
    }
}
```

配套习题（15）：

77. 组合 （中等） 给n个数返回所有k个数的组合
78. 子集（中等） 所有的组合
90. 子集 II（中等）有重复数据
46. 全排列 （中等） 所有排列
47. 全排列 II（中等） 有重复数据

习题课：回溯

王争的算法训练营

77. 组合 （中等） 给n个数返回所有k个数的组合

给定两个整数 *n* 和 *k*，返回 1 ... *n* 中所有可能的 *k* 个数的组合。

示例:

```
输入: n = 4, k = 2
输出:
[
  [2,4],
  [3,4],
  [2,3],
  [1,2],
  [1,3],
  [1,4],
]
```

```java
class Solution {
    private List<List<Integer>> result = new ArrayList<>();

    public List<List<Integer>> combine(int n, int k) {
        backtrack(n, k, 1, new ArrayList<Integer>());
        return result;
    }

    // n,k必须的参数
    // step阶段
    // path路径
    // step选与不选-可选列表
    private void backtrack(int n, int k, int step, List<Integer> path) {
        if (path.size()==k) {
            result.add(new ArrayList(path));
            return;
        }
        if (step == n+1) {
            return;
        }

        backtrack(n, k, step+1, path);
        path.add(step);
        backtrack(n, k, step+1, path);
        path.remove(path.size()-1);
    }
}
```

习题课：回溯

王争的算法训练营

78. 子集（中等） 所有的组合

给你一个整数数组 `nums` ，数组中的元素 **互不相同** 。返回该数组所有可能的子集（幂集）。

解集 **不能** 包含重复的子集。你可以按 **任意顺序** 返回解集。

示例 1：

```
输入：nums = [1,2,3]
输出：[[],[1],[2],[1,2],[3],[1,3],[2,3],[1,2,3]]
```

示例 2：

```
输入：nums = [0]
输出：[[],[0]]
```

提示：

- `1 <= nums.length <= 10`
- `-10 <= nums[i] <= 10`
- `nums` 中的所有元素 **互不相同**

习题课：回溯

王争的算法训练营

78. 子集（中等） 所有的组合

```
class Solution {
    private List<List<Integer>> result = new ArrayList<>();

    public List<List<Integer>> subsets(int[] nums) {
        backtrack(nums, 0, new ArrayList<Integer>());
        return result;
    }

    // k阶段
    // path路径
    // nums[k]选或不选-可选列表
    private void backtrack(int[] nums, int k, List<Integer> path) {
        if (k == nums.length) {
            result.add(new ArrayList(path));
            return;
        }
        backtrack(nums, k+1, path);
        path.add(nums[k]);
        backtrack(nums, k+1, path);
        path.remove(path.size()-1);
    }
}
```

习题课：回溯

王争的算法训练营

90. 子集 II（中等）有重复数据

给你一个整数数组 `nums` ，其中可能包含重复元素，请你返回该数组所有可能的子集（幂集）。

解集 **不能** 包含重复的子集。返回的解集中，子集可以按 **任意顺序** 排列。

示例 1：

```
输入：nums = [1,2,2]
输出：[[],[1],[1,2],[1,2,2],[2],[2,2]]
```

示例 2：

```
输入：nums = [0]
输出：[[],[0]]
```

提示：

- `1 <= nums.length <= 10`
- `-10 <= nums[i] <= 10`

```java
class Solution {
    private List<List<Integer>> result = new ArrayList<>();
    public List<List<Integer>> subsetsWithDup(int[] nums) {
        HashMap<Integer, Integer> hm = new HashMap<>();
        for (int i = 0; i < nums.length; ++i) {
            int count = 1;
            if (hm.containsKey(nums[i])) {
                count += hm.get(nums[i]);
            }
            hm.put(nums[i], count);
        }
        int n = hm.size();
        int[] uniqueNums = new int[n];
        int[] counts = new int[n];
        int k = 0;
        for (int i = 0; i < nums.length; ++i) {
            if (hm.containsKey(nums[i])) {
                uniqueNums[k] = nums[i];
                counts[k] = hm.get(nums[i]);
                k++;
                hm.remove(nums[i]);
            }
        }
        backtrack(uniqueNums, counts, 0, new ArrayList<Integer>());
        return result;
    }

    private void backtrack(int[] uniqueNums, int[] counts, int k, List<Integer> path) {
        if (k == uniqueNums.length) {
            result.add(new ArrayList<>(path));
            return;
        }
        for (int count = 0; count <= counts[k]; ++count) {
            for (int i = 0; i < count; ++i) {
                path.add(uniqueNums[k]);
            }
            backtrack(uniqueNums, counts, k+1, path);
            for (int i = 0; i < count; ++i) {
                path.remove(path.size()-1);
            }
        }
    }
}
```

习题课：回溯

王争的算法训练营

46. 全排列 （中等） 所有排列

给定一个不含重复数字的数组 `nums` ， 返回其 **所有可能的全排列** 。你可以 **按任意顺序** 返回答案。

示例 1：

```
输入：nums = [1,2,3]
输出：[[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]
```

示例 2：

```
输入：nums = [0,1]
输出：[[0,1],[1,0]]
```

示例 3：

```
输入：nums = [1]
输出：[[1]]
```

提示：

- `1 <= nums.length <= 6`
- `-10 <= nums[i] <= 10`
- `nums` 中的所有整数 **互不相同**

```java
class Solution {
  private List<List<Integer>> result = new ArrayList<>();

  public List<List<Integer>> permute(int[] nums) {
    List<Integer> path = new ArrayList<>();
    backtrack(nums, 0, path);
    return result;
  }

  // 路径：记录在path中
  // 决策阶段：k
  // 可选列表：nums中除掉存在于path中的数据
  private void backtrack(int[] nums, int k, List<Integer> path) {
    // 结束条件
    if (k == nums.length) {
      result.add(new ArrayList<>(path));
      return;
    }

    for (int i = 0; i < nums.length; ++i) {
      if (path.contains(nums[i])) {
        continue;
      }
      // 做选择
      path.add(nums[i]);
      // 递归
      backtrack(nums, k+1, path);
      // 撤销选择
      path.remove(path.size()-1);
    }
  }
}
```

习题课：回溯

王争的算法训练营

47. 全排列 II（中等） 有重复数据

给定一个可包含重复数字的序列 `nums` ， **按任意顺序** 返回所有不重复的全排列。

示例 1：

```
输入：nums = [1,1,2]
输出：
[[1,1,2],
 [1,2,1],
 [2,1,1]]
```

示例 2：

```
输入：nums = [1,2,3]
输出：[[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]
```

提示：

- `1 <= nums.length <= 8`
- `-10 <= nums[i] <= 10`

```java
class Solution {
  private List<List<Integer>> result = new ArrayList<>();
  public List<List<Integer>> permuteUnique(int[] nums) {
    HashMap<Integer, Integer> hm = new HashMap<>();
    for (int i = 0; i < nums.length; ++i) {
      int count = 1;
      if (hm.containsKey(nums[i])) {
        count += hm.get(nums[i]);
      }
      hm.put(nums[i], count);
    }
    int n = hm.size();
    int[] uniqueNums = new int[n];
    int[] counts = new int[n];
    int k = 0;
    for (int i = 0; i < nums.length; ++i) {
      if (hm.containsKey(nums[i])) {
          uniqueNums[k] = nums[i];
          counts[k] = hm.get(nums[i]);
          k++;
          hm.remove(nums[i]);
      }
    }
    backtrack(uniqueNums, counts, 0, new ArrayList(), nums.length);
    return result;
  }

  private void backtrack(int[] uniqueNums, int[] counts, int k, List<Integer> path, int n) {
    if (k == n) {
      result.add(new ArrayList<>(path));
      return;
    }
    for (int i = 0; i < uniqueNums.length; ++i) {
      if (counts[i] == 0) continue;
      path.add(uniqueNums[i]); // 添加选择
      counts[i]--;
      backtrack(uniqueNums, counts, k+1, path, n);
      path.remove(path.size()-1); // 撤销选择
      counts[i]++;
    }
  }
}
```

习题课：回溯

王争的算法训练营

配套习题（15）：

39. 组合总和 （中等） 选出某几个数相加为给定和，无重复数据，可以使用多次，不能有重复答案

40. 组合总和 II（中等）选出某几个数相加为给定和，有重复数据，只能使用一次，不能有重复答案

216. 组合总和 III（中等） 选出k个数相加为给定和，没有重复数据，只能使用一次

习题课：回溯

王争的算法训练营

39. 组合总和 （中等） 选出某几个数相加为给定和，无重复数据，可以使用多次，不能有重复答案

给定一个**无重复元素**的数组 `candidates` 和一个目标数 `target` ，找出 `candidates` 中所有可以使数字和为 `target` 的组合。

`candidates` 中的数字可以无限制重复被选取。

说明：

- 所有数字（包括 `target`）都是正整数。
- 解集不能包含重复的组合。

示例 1：

```
输入: candidates = [2,3,6,7], target = 7,
所求解集为:
[
  [7],
  [2,2,3]
]
```

示例 2：

```
输入: candidates = [2,3,5], target = 8,
所求解集为:
[
  [2,2,2,2],
  [2,3,3],
  [3,5]
]
```

提示：

- `1 <= candidates.length <= 30`

习题课：回溯

王争的算法训练营

```java
class Solution {
    private List<List<Integer>> result = new ArrayList<>();
    public List<List<Integer>> combinationSum(int[] candidates, int target) {
        backtrack(candidates, 0, target, new ArrayList<>());
        return result;
    }

    private void backtrack(int[] candidates, int k, int left, List<Integer> path) {
        if (left == 0) {
            result.add(new ArrayList<>(path));
            return;
        }
        if (k == candidates.length) {
            return;
        }
        for (int i = 0; i <= left/candidates[k]; ++i) {
            for (int j = 0; j < i; ++j) {
                path.add(candidates[k]);
            }
            backtrack(candidates, k+1, left-i*candidates[k], path);
            for (int j = 0; j < i; ++j) {
                path.remove(path.size()-1);
            }
        }
    }
}
```

习题课：回溯

王争的算法训练营

40. 组合总和 II（中等）选出某几个数相加为给定和，有重复数据，只能使用一次，不能有重复答案

给定一个数组 `candidates` 和一个目标数 `target` ，找出 `candidates` 中所有可以使数字和为 `target` 的组合。

`candidates` 中的每个数字在每个组合中只能使用一次。

说明：

- 所有数字（包括目标数）都是正整数。
- 解集不能包含重复的组合。

示例 1:

```
输入: candidates = [10,1,2,7,6,1,5], target = 8,
所求解集为:
[
  [1, 7],
  [1, 2, 5],
  [2, 6],
  [1, 1, 6]
]
```

示例 2:

```
输入: candidates = [2,5,2,1,2], target = 5,
所求解集为:
[
  [1,2,2],
  [5]
]
```

```java
class Solution {
    private List<List<Integer>> result = new ArrayList<>();
    public List<List<Integer>> combinationSum2(int[] candidates, int target) {
        HashMap<Integer, Integer> hashTable = new HashMap<>();
        for (int i = 0; i < candidates.length; ++i) {
            if (!hashTable.containsKey(candidates[i])) {
                hashTable.put(candidates[i], 1);
            } else {
                hashTable.put(candidates[i], hashTable.get(candidates[i])+1);
            }
        }
        List<Integer> nums = new ArrayList<>();
        List<Integer> counts = new ArrayList<>();
        for (int i = 0; i < candidates.length; ++i) {
            if (hashTable.containsKey(candidates[i])) {
                nums.add(candidates[i]);
                counts.add(hashTable.get(candidates[i]));
                hashTable.remove(candidates[i]);
            }
        }
        backtrack(nums, counts, 0, target, new ArrayList<>());
        return result;
    }
}
```

```java
private void backtrack(List<Integer> nums, List<Integer> counts, int k, int left, List<Integer> path) {
    if (left == 0) {
        result.add(new ArrayList<>(path));
        return;
    }
    if (left < 0 || k == nums.size()) {
        return;
    }
    for (int count = 0; count <= counts.get(k); ++count) {
        for (int i = 0; i < count; ++i) {
            path.add(nums.get(k));
        }
        backtrack(nums, counts, k+1, left-count*nums.get(k), path);
        for (int i = 0; i < count; ++i) {
            path.remove(path.size()-1);
        }
    }
}
```

习题课：回溯

王争的算法训练营

216. 组合总和 III（中等） 选出k个数相加为给定和，没有重复数据，只能使用一次

找出所有相加之和为 ***n*** 的 ***k*** 个数的组合。组合中只允许含有 1 - 9 的正整数，并且每种组合中不存在重复的数字。

说明：

- 所有数字都是正整数。
- 解集不能包含重复的组合。

示例 1:

```
输入: k = 3, n = 7
输出: [[1,2,4]]
```

示例 2:

```
输入: k = 3, n = 9
输出: [[1,2,6], [1,3,5], [2,3,4]]
```

习题课：回溯

王争的算法训练营

216. 组合总和 III（中等） 选出k个数相加为给定和，没有重复数据，只能使用一次

```
class Solution {
    private List<List<Integer>> result = new ArrayList<>();
    // 1~9，选k个数，和为n
    public List<List<Integer>> combinationSum3(int k, int n) {
        backtrack(k, n, 1, 0, new ArrayList<>());
        return result;
    }

    private void backtrack(int k, int n, int step, int sum, List<Integer> path) {
        if (sum == n && path.size() == k) {
            result.add(new ArrayList<>(path));
            return;
        }
        if (sum > n || path.size() > k || step > 9) {
            return;
        }
        backtrack(k, n, step+1, sum, path);
        path.add(step);
        backtrack(k, n, step+1, sum+step, path);
        path.remove(path.size()-1);
    }
}
```

习题课：回溯

王争的算法训练营

配套习题（15）：

131. 分割回文串（中等）

93. 复原 IP 地址（中等）

22. 括号生成（中等）

习题课：回溯

王争的算法训练营

131. 分割回文串（中等）

给你一个字符串 `s`，请你将 `s` 分割成一些子串，使每个子串都是 **回文串** 。返回 `s` 所有可能的分割方案。

回文串 是正着读和反着读都一样的字符串。

示例 1：

```
输入：s = "aab"
输出：[["a","a","b"],["aa","b"]]
```

示例 2：

```
输入：s = "a"
输出：[["a"]]
```

提示：

- `1 <= s.length <= 16`
- `s` 仅由小写英文字母组成

aabbbc

```java
class Solution {
    private List<List<String>> result = new ArrayList<>();
    public List<List<String>> partition(String s) {
        backtrack(s, 0, new ArrayList<>());
        return result;
    }

    private void backtrack(String s, int k, List<String> path) {
        if (k == s.length()) {
            result.add(new ArrayList<>(path));
            return;
        }
        for (int end = k; end < s.length(); ++end) {
            if (ispalindrome(s, k, end)) {
                path.add(s.substring(k, end+1));
                backtrack(s, end+1, path);
                path.remove(path.size()-1);
            }
        }
    }

    private boolean ispalindrome(String s, int p, int r) {
        int i = p;
        int j = r;
        while (i <= j) {
            if (s.charAt(i) != s.charAt(j)) return false;
            i++;
            j--;
        }
        return true;
    }
}
```

习题课：回溯

王争的算法训练营

123524121

93. 复原 IP 地址（中等）

给定一个只包含数字的字符串，用以表示一个 IP 地址，返回所有可能从 `s` 获得的 **有效 IP 地址** 。你可以按任何顺序返回答案。

有效 IP 地址 正好由四个整数（每个整数位于 0 到 255 之间组成，且不能含有前导 `0`），整数之间用 `'.'` 分隔。

例如："0.1.2.201" 和 "192.168.1.1" 是 **有效** IP 地址，但是 "0.011.255.245"、"192.168.1.312" 和 "192.168@1.1" 是 **无效** IP 地址。

示例 1：

```
输入：s = "25525511135"
输出：["255.255.11.135","255.255.111.35"]
```

示例 2：

```
输入：s = "0000"
输出：["0.0.0.0"]
```

示例 3：

```
输入：s = "1111"
输出：["1.1.1.1"]
```

示例 4：

```
输入：s = "010010"
输出：["0.10.0.10","0.100.1.0"]
```

```java
class Solution {
    private List<String> result = new ArrayList<>();
    public List<String> restoreIpAddresses(String s) {
        backtrack(s, 0, 0, new ArrayList<>());
        return result;
    }

    private void backtrack(String s, int k,
                           int step, List<Integer> path) {
        if (k == s.length() && step == 4) {
            StringBuilder sb = new StringBuilder();
            for (int i = 0; i < 3; ++i) {
                sb.append(path.get(i) + ".");
            }
            sb.append(path.get(3));
            result.add(sb.toString());
            return;
        }
        if (step > 4) {
            return;
        }
        if (k == s.length()) {
            return;
        }

        int val = 0;
        // 1位数
        if (k < s.length()) {
            val = val*10+(s.charAt(k)-'0');
            path.add(val);
            backtrack(s, k+1, step+1, path);
            path.remove(path.size()-1);
        }

        if (s.charAt(k) == '0') {//前导0不行
            return;
        }
        // 2位数
        if (k+1 < s.length()) {
            val = val*10 + (s.charAt(k+1)-'0');
            path.add(val);
            backtrack(s, k+2, step+1, path);
            path.remove(path.size()-1);
        }
        // 3位数
        if (k+2 < s.length()) {
            val = val*10 + (s.charAt(k+2)-'0');
            if (val <= 255) {
                path.add(val);
                backtrack(s, k+3, step+1, path);
                path.remove(path.size()-1);
            }
        }
    }
}
```

习题课：回溯

王争的算法训练营

22. 括号生成（中等）

数字 `n` 代表生成括号的对数，请你设计一个函数，用于能够生成所有可能的并且 **有效的** 括号组合。

示例 1：

```
输入：n = 3
输出：["((()))","(()())","(())()","()(())","()()()"]
```

示例 2：

```
输入：n = 1
输出：["()"]
```

提示：

- `1 <= n <= 8`

习题课：回溯

王争的算法训练营

22. 括号生成（中等）

```
class Solution {
    private List<String> result = new ArrayList<>();
    public List<String> generateParenthesis(int n) {
        char[] path = new char[2*n];
        backtrack(n, 0, 0, 0, path);
        return result;
    }

    private void backtrack(int n, int leftUsed, int rightUsed, int k, char[] path) {
        if (k == 2*n) {
            result.add(String.valueOf(path));
            return;
        }
        if (leftUsed < n) {
            path[k] = '(';
            backtrack(n, leftUsed+1, rightUsed, k+1, path);
        }
        if (leftUsed > rightUsed) {
            path[k] = ')';
            backtrack(n, leftUsed, rightUsed+1, k+1, path);
        }
    }
}
```

提问环节

关注微信公众号“**小争哥**”，

后台回复“**PDF**”获取独家算法资料