王争的算法训练营

习题课：DFS

DFS相关题型/套路

王争的算法训练营

重点，特别是DFS，但题型有限，容易掌握，代码模板可以复用。

题型1、二维矩阵搜索或遍历

题型2、最短路径（BFS）

题型3、连通分量/连通性

题型4、拓扑排序

题型5、检测环

习题课：DFS

王争的算法训练营

配套习题（10+4）：

剑指 Offer 13. 机器人的运动范围（中等） **（已讲）**

面试题 08.10. 颜色填充（简单）

面试题 04.01. 节点间通路（中等）

200. 岛屿数量（中等） **（已讲）**

面试题 16.19. 水域大小（中等）

207. 课程表（中等） 拓扑排序

79. 单词搜索（中等）

1306. 跳跃游戏 III（中等）

752. 打开转盘锁（中等） BFS **（已讲）**

面试题 17.22. 单词转换（中等）

以下选做：

面试题 17.07. 婴儿名字（困难） 关系是固定的，并查集或者DFS都能搞定！ 关键在于将数据转化成图结构，也就是建模烦！

529. 扫雷游戏（困难）

127. 单词接龙（困难）

126. 单词接龙 II（困难）

习题课：DFS

王争的算法训练营

剑指 Offer 13. 机器人的运动范围（中等） DFS（已讲）

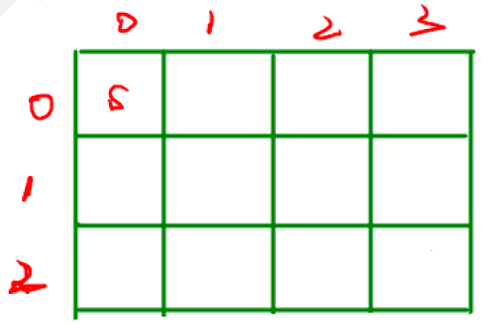
剑指 Offer 13. 机器人的运动范围

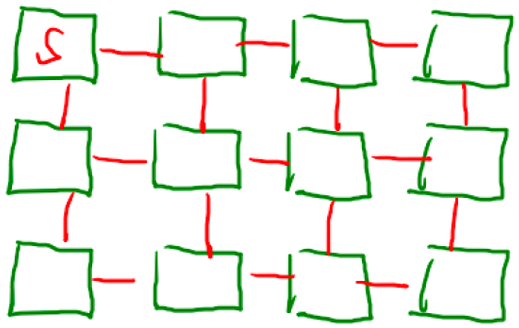
难度 中等　282　收藏　分享　切换为英文　接收动态　反馈

地上有一个m行n列的方格，从坐标 `[0,0]` 到坐标 `[m-1,n-1]` 。一个机器人从坐标 `[0, 0]` 的格子开始移动，它每次可以向左、右、上、下移动一格（不能移动到方格外），也不能进入行坐标和列坐标的数位之和大于k的格子。例如，当k为18时，机器人能够进入方格 [35, 37] ，因为3+5+3+7=18。但它不能进入方格 [35, 38]，因为3+5+3+8=19。请问该机器人能够到达多少个格子？

示例 1：

```
输入：m = 2, n = 3, k = 1
输出：3
```





示例 2：

```
输入：m = 3, n = 1, k = 0
输出：1
```

提示：

- `1 <= n,m <= 100`
- `0 <= k <= 20`

```java
class Solution {
    private boolean[][] visited;
    private int count = 0;
    public int movingCount(int m, int n, int k) {
        visited = new boolean[m][n];
        dfs(0, 0, m, n, k);
        return count;
    }

    private void dfs(int i, int j, int m, int n, int k) {
        visited[i][j] = true;
        count++;
        int[][] directions = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
        for (int di = 0; di < 4; ++di) {
            int newi = i + directions[di][0];
            int newj = j + directions[di][1];
            if (newi >= m || newi < 0 || newj >= n || newj < 0
                || visited[newi][newj] == true
                || check(newi, newj, k) == false) {
                continue;
            }
            dfs(newi, newj, m, n, k);
        }
    }

    private boolean check(int i, int j, int k) {
        int sum = 0;
        while (i != 0) {
            sum += (i%10);
            i /= 10;
        }
        while (j != 0) {
            sum += (j%10);
            j /= 10;
        }
        return sum <= k;
    }
}
```

习题课：DFS

王争的算法训练营

面试题 08.10. 颜色填充（简单） DFS

编写函数，实现许多图片编辑软件都支持的「颜色填充」功能。

待填充的图像用二维数组 `image` 表示，元素为初始颜色值。初始坐标点的行坐标为 `sr` 列坐标为 `sc` 。需要填充的新颜色为 `newColor` 。

「周围区域」是指颜色相同且在上、下、左、右四个方向上存在相连情况的若干元素。

请用新颜色填充初始坐标点的周围区域，并返回填充后的图像。

示例：

输入：
`image = [[1,1,1],[1,1,0],[1,0,1]]`
`sr = 1, sc = 1, newColor = 2`
输出：`[[2,2,2],[2,2,0],[2,0,1]]`
解释：
初始坐标点位于图像的正中间，坐标 `(sr,sc)=(1,1)` 。
初始坐标点周围区域上所有符合条件的像素点的颜色都被更改成 `2` 。
注意，右下角的像素没有更改为 `2` ，因为它不属于初始坐标点的周围区域。

提示：

- `image` 和 `image[0]` 的长度均在范围 [1, 50] 内。
- 初始坐标点 `(sr,sc)` 满足 `0 <= sr < image.length` 和 `0 <= sc < image[0].length` 。
- `image[i][j]` 和 `newColor` 表示的颜色值在范围 `[0, 65535]` 内。

习题课：DFS

王争的算法训练营

面试题 08.10. 颜色填充（简单） DFS

```
class Solution {
    public int[][] floodFill(int[][] image, int sr, int sc, int newColor) {
        int n = image.length;
        int m = image[0].length;
        dfs(image, n, m, sr, sc, image[sr][sc], newColor);
        return image;
    }

    private void dfs(int[][] image, int n, int m, int sr, int sc, int color, int newColor) {
        image[sr][sc] = newColor;
        int[][] dirs = {{-1, 0}, {1, 0}, {0, 1}, {0, -1}};
        for (int k = 0; k < 4; ++k) {
            int newr = sr + dirs[k][0];
            int newc = sc + dirs[k][1];
            if (newr < 0 || newr >= n || newc < 0 || newc >= m || image[newr][newc] != color
            || image[newr][newc]==newColor) {
                continue;
            }
            dfs(image, n, m, newr, newc, color, newColor);
        }
    }
}
```

习题课：DFS

王争的算法训练营

[面试题 04.01. 节点间通路](#)（中等）

节点间通路。给定有向图，设计一个算法，找出两个节点之间是否存在一条路径。

示例1:

```
输入：n = 3, graph = [[0, 1], [0, 2], [1, 2], [1, 2]], start = 0, target = 2
输出：true
```

示例2:

```
 输入：n = 5, graph = [[0, 1], [0, 2], [0, 4], [0, 4], [0, 1], [1, 3], [1, 4], [1, 3],
[2, 3], [3, 4]], start = 0, target = 4
 输出 true
```

提示：

1. 节点数量n在[0, 1e5]范围内。
2. 节点编号大于等于 0 小于 n。
3. 图中可能存在自环和平行边。

```java
class Solution {
    private boolean[] visited;
    private HashSet<Integer>[] adj;
    private boolean found = false;
    public boolean findWhetherExistsPath(int n, int[][] graph, int start, int target) {
        visited = new boolean[n];
        adj = new HashSet[n];
        for (int i = 0; i < n; i++) {
            adj[i] = new HashSet<Integer>();
        }
        for (int i = 0; i < n; i++) {
            if (!adj[graph[i][0]].contains(graph[i][1])) {
                adj[graph[i][0]].add(graph[i][1]);
            }
        }
        dfs(start, target);
        return found;
    }

    private void dfs(int cur, int target) {
        if (found) return;
        if (cur == target) {
            found = true;
            return;
        }
        visited[cur] = true;
        for (Integer next : adj[cur]) {
            if (!visited[next]) {
                dfs(next, target);
            }
        }
    }
}
```

习题课：DFS

王争的算法训练营

200. 岛屿数量（中等） 求连通分量（已讲）

200. 岛屿数量

难度 中等 1146 收藏 分享 切换为英文 接收动态 反馈

给你一个由 `'1'`（陆地）和 `'0'`（水）组成的的二维网格，请你计算网格中岛屿的数量。

岛屿总是被水包围，并且每座岛屿只能由水平方向和/或竖直方向上相邻的陆地连接形成。

此外，你可以假设该网格的四条边均被水包围。

示例 1：

```
输入：grid = [
  ["1","1","1","1","0"],
  ["1","1","0","1","0"],
  ["1","1","0","0","0"],
  ["0","0","0","0","0"]
]
输出：1
```

示例 2：

```
输入：grid = [
  ["1","1","0","0","0"],
  ["1","1","0","0","0"],
  ["0","0","1","0","0"],
  ["0","0","0","1","1"]
]
输出：3
```

```java
class Solution {
    private boolean[][] visited;
    private int h;
    private int w;
    public int numIslands(char[][] grid) {
        h = grid.length;
        w = grid[0].length;
        visited = new boolean[h][w];
        int result  = 0;
        for (int i = 0; i < h; ++i) {
            for (int j = 0; j < w; ++j) {
                if (visited[i][j] != true && grid[i][j] == '1') {
                    result++;
                    dfs(grid, i, j);
                }
            }
        }
        return result;
    }

    private void dfs(char[][] grid, int i, int j) {
        int[][] directions = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
        visited[i][j] = true;
        for (int k = 0; k < 4; ++k) {
            int newi = i + directions[k][0];
            int newj = j + directions[k][1];
            if (newi >= 0 && newi < h && newj >= 0 && newj < w
            && visited[newi][newj] == false && grid[newi][newj]=='1') {
                dfs(grid, newi, newj);
            }
        }
    }
}
```

习题课：DFS

王争的算法训练营

面试题 16.19. 水域大小（中等） 连通性

你有一个用于表示一片土地的整数矩阵 `land`，该矩阵中每个点的值代表对应地点的海拔高度。若值为0则表示水域。由垂直、水平或对角连接的水域为池塘。池塘的大小是指相连接的水域的个数。编写一个方法来计算矩阵中所有池塘的大小，返回值需要从小到大排序。

示例：

输入：

```
[
  [0,2,1,0],
  [0,1,0,1],
  [1,1,0,1],
  [0,1,0,1]
]
```

输出： `[1,2,4]`

提示：

- `0 < len(land) <= 1000`
- `0 < len(land[i]) <= 1000`

```java
class Solution {
    private int count = 0;
    private int n;
    private int m;
    public int[] pondSizes(int[][] land) {
        n = land.length;
        m = land[0].length;
        List<Integer> result = new ArrayList<>();
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < m; ++j) {
                if (land[i][j] == 0) {
                    count = 0;
                    dfs(land, i, j);
                    result.add(count);
                }
            }
        }
        int[] resultArr = new int[result.size()];
        for (int i = 0; i < result.size(); ++i) {
            resultArr[i] = result.get(i);
        }
        Arrays.sort(resultArr);
        return resultArr;
    }

    private void dfs(int[][] land, int curi, int curj) {
        count++;
        land[curi][curj] = 1;
        int[][] dirs = {{-1, 0}, {1, 0}, {0, 1}, {0, -1},
                        {-1, -1}, {1, 1}, {-1, 1}, {1, -1}};
        for (int d = 0; d < 8; ++d) {
            int newi = curi + dirs[d][0];
            int newj = curj + dirs[d][1];
            if (newi >= 0 && newi < n && newj >= 0 && newj < m
                && land[newi][newj] == 0) {
                dfs(land, newi, newj);
            }
        }
    }
}
```

习题课：DFS

王争的算法训练营

207. 课程表（中等） 拓扑排序，看是否存在环

你这个学期必须选修 `numCourses` 门课程，记为 `0` 到 `numCourses - 1` 。

在选修某些课程之前需要一些先修课程。 先修课程按数组 `prerequisites` 给出，其中 `prerequisites[i] = [a_i, b_i]` ，表示如果要学习课程 a_i 则 **必须** 先学习课程 b_i 。

- 例如，先修课程对 `[0, 1]` 表示：想要学习课程 `0` ，你需要先完成课程 `1` 。

请你判断是否可能完成所有课程的学习？如果可以，返回 `true` ；否则，返回 `false` 。

示例 1：

```
输入：numCourses = 2, prerequisites = [[1,0]]
输出：true
解释：总共有 2 门课程。学习课程 1 之前，你需要完成课程 0 。这是可能的。
```

示例 2：

```
输入：numCourses = 2, prerequisites = [[1,0],[0,1]]
输出：false
解释：总共有 2 门课程。学习课程 1 之前，你需要先完成课程 0 ；并且学习课程 0 之前，你还应先完成课程 1 。这是不可能的。
```

提示：

- $1 <= numCourses <= 10^5$
- $0 <= prerequisites.length <= 5000$
- $prerequisites[i].length == 2$
- $0 <= a_i, b_i < numCourses$
- $prerequisites[i]$ 中的所有课程对 **互不相同**

```java
class Solution {
    public boolean canFinish(int numCourses, int[][] prerequisites) {
        List<Integer> adjs[] = new List[numCourses];
        for (int i = 0; i < numCourses; ++i) {
            adjs[i] = new ArrayList<Integer>();
        }
        int[] indegrees = new int[numCourses];
        for (int i = 0; i < prerequisites.length; i++) {
            adjs[prerequisites[i][1]].add(prerequisites[i][0]);
            indegrees[prerequisites[i][0]]++;
        }
        List<Integer> zeroInDegrees = new LinkedList<>();
        for (int i = 0; i < indegrees.length; ++i) {
            if (indegrees[i] == 0) {
                zeroInDegrees.add(i);
            }
        }
        int zeroInDegreesCount = 0;
        while (!zeroInDegrees.isEmpty()) {
            int coursei = zeroInDegrees.remove();
            zeroInDegreesCount++;
            for (Integer coursej : adjs[coursei]) {
                indegrees[coursej]--;
                if (indegrees[coursej] == 0) {
                    zeroInDegrees.add(coursej);
                }
            }
        }
        return zeroInDegreesCount == numCourses;
    }
}
```

习题课：DFS

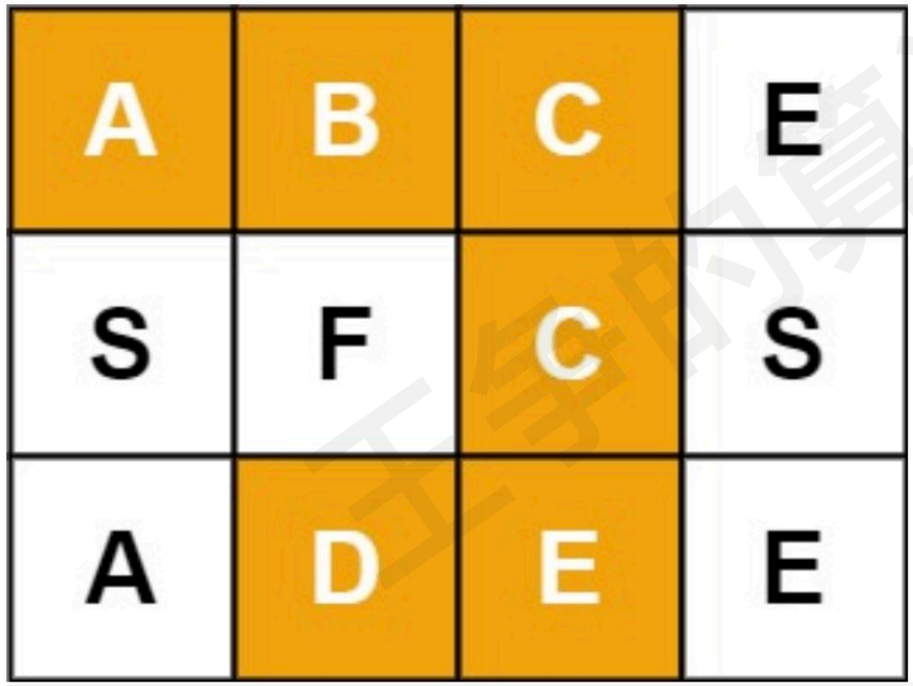
王争的算法训练营

79. 单词搜索（中等）DFS的稍微升级

给定一个 `m x n` 二维字符网格 `board` 和一个字符串单词 `word` 。如果 `word` 存在于网格中，返回 `true` ；否则，返回 `false` 。

单词必须按照字母顺序，通过相邻的单元格内的字母构成，其中“相邻”单元格是那些水平相邻或垂直相邻的单元格。同一个单元格内的字母不允许被重复使用。

示例 1：




```
输入：board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "ABCCED"
输出：true
```

```java
class Solution {
    private boolean existed = false;
    private int h;
    private int w;
    public boolean exist(char[][] board, String word) {
      h = board.length;
      w = board[0].length;
      for (int i = 0; i < h; ++i) {
        for (int j = 0; j < w; ++j) {
          boolean[][] visited = new boolean[h][w];
          dfs(board, word, i, j, 0, visited);
          if (existed) return true;
        }
      }
      return false;
    }

    private void dfs(char[][] board, String word, int i, int j, int k, boolean[][] visited) {
      if (existed == true) return;
      if (word.charAt(k) != board[i][j]) {
        return;
      }
      visited[i][j] = true;
      if (k == word.length()-1) {
        existed = true;
        return;
      }

      int[][] directions = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
      for (int di = 0; di < 4; di++) {
        int nexti = i + directions[di][0];
        int nextj = j + directions[di][1];
        if (nexti>=0 && nexti < h && nextj>=0 && nextj < w
            && !visited[nexti][nextj]) {
          dfs(board, word, nexti, nextj, k+1, visited);
        }
      }
      visited[i][j] = false;
    }
}
```

习题课：DFS

王争的算法训练营

1306. 跳跃游戏 III（中等） DFS，看着不像，实际上是

这里有一个非负整数数组 `arr`，你最开始位于该数组的起始下标 `start` 处。当你位于下标 `i` 处时，你可以跳到 `i + arr[i]` 或者 `i - arr[i]`。

请你判断自己是否能够跳到对应元素值为 0 的 **任一** 下标处。

注意，不管是什么情况下，你都无法跳到数组之外。

示例 1：

```
输入：arr = [4,2,3,0,3,1,2], start = 5
输出：true
解释：
到达值为 0 的下标 3 有以下可能方案：
下标 5 -> 下标 4 -> 下标 1 -> 下标 3
下标 5 -> 下标 6 -> 下标 4 -> 下标 1 -> 下标 3
```

示例 2：

```
输入：arr = [4,2,3,0,3,1,2], start = 0
输出：true
解释：
到达值为 0 的下标 3 有以下可能方案：
下标 0 -> 下标 4 -> 下标 1 -> 下标 3
```

示例 3：

```
输入：arr = [3,0,2,1,2], start = 2
输出：false
解释：无法到达值为 0 的下标 1 处。
```

```java
class Solution {
    private boolean[] visited;
    private boolean reached = false;
    public boolean canReach(int[] arr, int start) {
        int n = arr.length;
        visited = new boolean[n];
        dfs(arr, start);
        return reached;
    }

    private void dfs(int[] arr, int curi) {
        if (reached) return;
        if (arr[curi]==0) {
            reached = true;
            return;
        }
        visited[curi] = true;
        int move2left = curi-arr[curi];
        if (move2left>=0 && move2left<arr.length
           && visited[move2left]==false) {
            dfs(arr, move2left);
        }
        int move2right = curi+arr[curi];
        if (move2right>=0 && move2right<arr.length
           && visited[move2right]==false) {
            dfs(arr, move2right);
        }
    }
}
```

习题课：DFS

王争的算法训练营

752. 打开转盘锁（中等） BFS（已讲）

752. 打开转盘锁

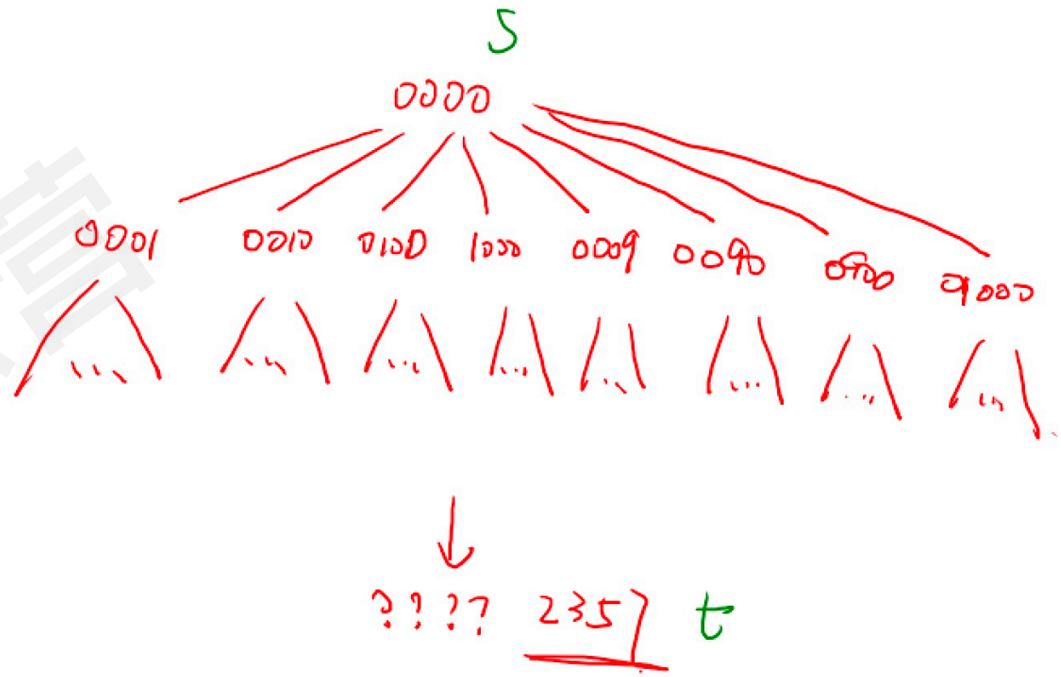
难度 中等 250 收藏 分享 切换为英文 接收动态 反馈

你有一个带有四个圆形拨轮的转盘锁。每个拨轮都有10个数字： `'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'` 。每个拨轮可以自由旋转：例如把 `'9'` 变为 `'0'`， `'0'` 变为 `'9'` 。每次旋转都只能旋转一个拨轮的一位数字。

锁的初始数字为 `'0000'` ，一个代表四个拨轮的数字的字符串。

列表 `deadends` 包含了一组死亡数字，一旦拨轮的数字和列表里的任何一个元素相同，这个锁将会被永久锁定，无法再被旋转。

字符串 `target` 代表可以解锁的数字，你需要给出最小的旋转次数，如果无论如何不能解锁，返回 -1。



示例 1:

```
输入: deadends = ["0201","0101","0102","1212","2002"], target = "0202"
输出: 6
解释:
可能的移动序列为 "0000" -> "1000" -> "1100" -> "1200" -> "1201" -> "1202" -> "0202"。
注意 "0000" -> "0001" -> "0002" -> "0102" -> "0202" 这样的序列是不能解锁的，
因为当拨动到 "0102" 时这个锁就会被锁定。
```

示例 2:

```
输入: deadends = ["8888"], target = "0009"
输出: 1
解释:
把最后一位反向旋转一次即可 "0000" -> "0009"。
```

```java
class Solution {
    public int openLock(String[] deadends, String target) {
        HashSet<String> deadset = new HashSet();
        for (String d: deadends) {
            deadset.add(d);
        }
        if (deadset.contains("0000")) return -1;
        Queue<String> queue = new LinkedList();
        Set<String> visited = new HashSet();
        queue.offer("0000");
        visited.add("0000");
        int depth = 0;
        while (!queue.isEmpty()) {
            int size = queue.size();
            int k = 0;
            while (k < size) {
                String node = queue.poll();
                k++;
                if (node.equals(target)) {
                    return depth;
                }
                List<String> newnodes = genNewNode(node);
                for (String newnode : newnodes) {
                    if (visited.contains(newnode)
                        || deadset.contains(newnode)) {
                        continue;
                    }
                    queue.add(newnode);
                    visited.add(newnode);
                }
            }
            depth++;
        }
        return -1;
    }
}
```

```java
private List<String> genNewNode(String node) {
    List<String> newnodes = new ArrayList<>();
    int[] change = {-1, 1};
    for (int i = 0; i < 4; ++i) {
        for (int k = 0; k < 2; ++k) {
            char[] newNode = new char[4];
            for (int j = 0; j < i; ++j) {
                newNode[j] = node.charAt(j);
            }
            for (int j = i+1; j < 4; ++j) {
                newNode[j] = node.charAt(j);
            }
            String newC = (((node.charAt(i)-'0') + change[k] + 10) % 10) + "";
            newNode[i] = newC.charAt(0);
            newnodes.add(new String(newNode));
        }
    }
    return newnodes;
}
```

习题课：DFS

王争的算法训练营

面试题 17.22. 单词转换（中等）

给定字典中的两个词，长度相等。写一个方法，把一个词转换成另一个词， 但是一次只能改变一个字符。每一步得到的新词都必须能在字典中找到。

编写一个程序，返回一个可能的转换序列。如有多个可能的转换序列，你可以返回任何一个。

示例 1:

```
输入:
beginWord = "hit",
endWord = "cog",
wordList = ["hot","dot","dog","lot","log","cog"]

输出:
["hit","hot","dot","lot","log","cog"]
```

示例 2:

```
输入:
beginWord = "hit"
endWord = "cog"
wordList = ["hot","dot","dog","lot","log"]

输出: []

解释: endWord "cog" 不在字典中，所以不存在符合要求的转换序列。
```

```java
class Solution {
    private Set<String> visited = new HashSet<>();
    private List<String> resultPath = new ArrayList<>();
    private boolean found = false;
    public List<String> findLadders(String beginWord, String endWord, List<String> wordList) {
        dfs(beginWord, endWord, new ArrayList<>(), wordList);
        return resultPath;
    }

    private void dfs(String curWord, String endWord, List<String> path, List<String> wordList) {
        if (found) return;

        path.add(curWord);
        visited.add(curWord);
        if (curWord.equals(endWord)) {
            resultPath.addAll(path);
            found = true;
            return;
        }
        for (int i = 0; i < wordList.size(); ++i) {
            String nextWord = wordList.get(i);
            if (visited.contains(nextWord) || !isValidChange(curWord, nextWord)) {
                continue;
            }
            dfs(nextWord, endWord, path, wordList);
        }
        path.remove(path.size()-1);
    }

    private boolean isValidChange(String word1, String word2) {
        int diff = 0;
        for (int i = 0; i < word1.length(); ++i) {
            if (word1.charAt(i) != word2.charAt(i)) {
                diff++;
            }
        }
        return diff == 1;
    }
}
```

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