17-电话号码的字母组合

题述

17. 电话号码的字母组合

难度 中等 1876

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"]
```

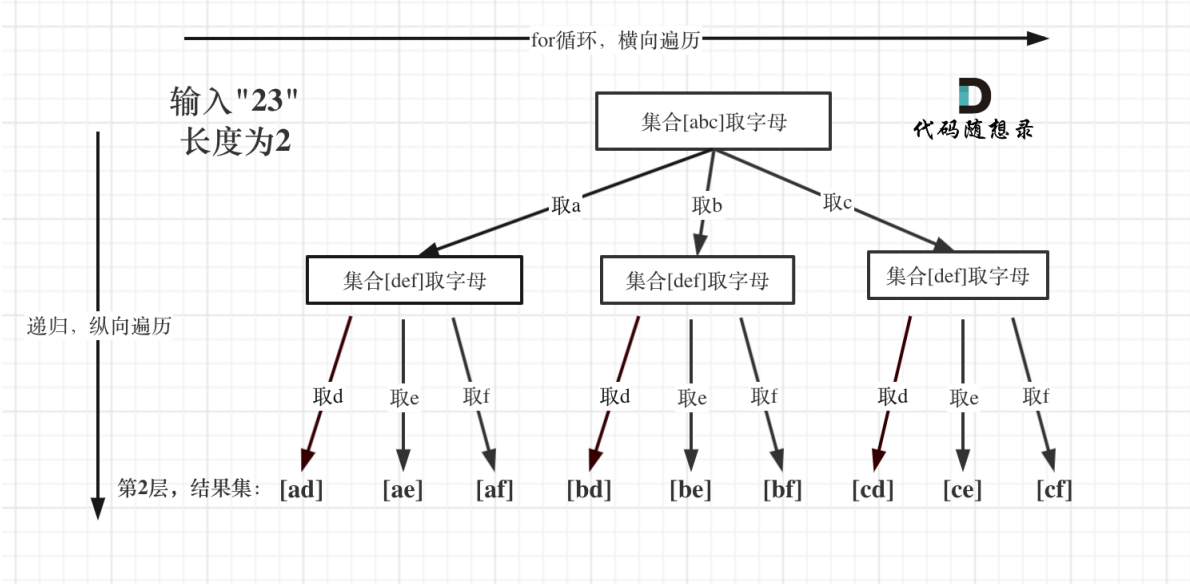
思路

数字和字母的映射关系

定义一个map或者二维数组 来表示映射关系

```
const string letterMap[10] = {
    "", // 0
    "", // 1
    "abc", // 2
    "def", // 3
    "ghi", // 4
    "jkl", // 5
    "mno", // 6
    "pqrs", // 7
    "tuv", // 8
    "wxyz", // 9
};
```

回溯构建



图中可以看出遍历的深度，就是输入"23"的长度，而叶子节点就是我们要收集的结果，输出["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"]。

- 确定回溯函数参数
  - 首先需要一个字符串s来收集叶子节点的结果，然后用一个字符串数组result保存起来，这两个变量我依然定义为全局。

    再来看参数，参数指定是有题目中给的string digits，然后还要有一个参数就是int型的index

    ```
    vector<string> result;
    string s;
    void backtracking(const string& digits, int index)
    ```

- 确定终止条件
  - 例如输入用例"23"，两个数字，那么根节点往下递归两层就可以了，叶子节点就是要收集的结果集。

    那么终止条件就是如果index 等于 输入的数字个数（digits.size）了（本来index就是用来遍历digits的）。

    然后收集结果，结束本层递归。

```
if (index == digits.size()) {
    result.push_back(s);
    return;
}
```

- 确定单层遍历逻辑
  - 首先要取index指向的数字，并找到对应的字符集（手机键盘的字符集）。
  - ```
    int digit = digits[index] - '0';        // 将index指向的数字转为int
    string letters = letterMap[digit];      // 取数字对应的字符集
    for (int i = 0; i < letters.size(); i++) {
        s.push_back(letters[i]);            // 处理
        backtracking(digits, index + 1);    // 递归，注意index+1，一下层要处理下一个数字了
        s.pop_back();                       // 回溯
    }
    ```

题解

Python回溯

```
class Solution:
    def __init__(self):
        self.answers = []
        self.answer = ''    #类似组合中的path和result
        self.letterMap = {
            '2' : 'abc' ,
            '3' : 'def' ,
            '4' : 'ghi' ,
            '5' : 'jkl' ,
            '6' : 'mno' ,
            '7' : 'pqrs' ,
            '8' : 'tuv' ,
            '9' : 'wxyz'
        }

    def backTracking(self, digits: str, index: int) -> None:
        #回溯子过程函数
        #终止条件
        if index == len(digits):
            #当本层遍历结束时
            self.answers.append(self.answer)
            return

        #单层递归
        letters = self.letterMap[digits[index]]
        for letter in letters:
            self.answer += letter #字母处理
            self.backTracking(digits,index + 1)
```
```

```
            self.answer = self.answer[:-1]  #回溯

    def letterCombinations(self, digits: str) -> List[str]:
        #回溯入口
        self.answers.clear()
        if not digits : return []  #处理特殊情况
        self.backTracking(digits,0)
        return self.answers
```

思考

回溯三部曲

递归函数参数、递归终止条件、树结构的联想及横纵遍历