# 回溯法

* 1. 是一个类似枚举的搜索尝试过程，在搜索尝试过程中寻找问题的解，若发现不满足条件则回溯返回，尝试别的路径
  2. 在包含问题所有解的解空间树中，根节点出发根据深度优先搜索策略进行探索
  3. 探索到某一结点时，判断结点是否包含问题的解，若包含，则继续探索，若不包含，则逐层向其祖先结点回溯。
  4. 递归形式举例：
     1. 仿照手机9键输入法，digits字符串包含顺序按下的按键所对应的数字，要求返回所有可能的字母组合
        1. @param {string} digits
        2. @return {string[]}
        3. var letterCombinations = function(digits) {
        4. if(digits === '') return []
        5. let res = []
        6. let map = {2:'abc',3:'def',4:'ghi',5:'jkl',6:'mno',7:'pqrs',8:'tuv',9:'wxyz'}
        7. let dfs = (str, i)=>{
        8. if (i > digits.length - 1){
        9. res.push(str)
        10. return
        11. }
        12. for (let letter of map[digits[i]]){
        13. dfs(str + letter,i+1)
        14. }
        15. }
        16. dfs('',0)
        17. return res
        18. }

# 双指针法

# 哑结点

* 1. 创建一个dummyNode指向头结点，使得空链表的处理方式与非空链表相同

# 动态规划

* 1. 可用动态规划求解的问题分为两大类，一种是求最优解类，典型问题是背包问题，另一种就是计数类，比如统计方案数的问题，它们都存在一定的递推性质。前者的递推性质还有一个名字，叫做 「最优子结构」 ——即当前问题的最优解取决于子问题的最优解，后者类似，当前问题的方案数取决于子问题的方案数。

# 滑动窗口

* 1. 用法：
     1. 在字符串 S 中使用双指针，初始化 left = right = 0，把索引闭区间 [left, right] 称为一个「窗口」。
     2. 先不断地增加 right 指针扩大窗口 [left, right]，直到窗口中的字符串符合要求（包含了 T 中的所有字符）。
     3. 此时，我们停止增加 right，转而不断增加 left 指针缩小窗口 [left, right]，直到不能继续缩小。更新一轮结果。
     4. 将left自增1，此时窗口不满足条件。再重复2、3步。直至right超过右边界