Backtracking, DFS

#G面试准备

Subsets

Subsets - LeetCode

- helper(nums, i + 1, subset, results)
- 错：startIndex + 1 ➡ i + 1

Subsets II

Subsets II - LeetCode

> [1,2,2']什么时候我们应该避免重复呢?
> 遇到2'的时候,如果2'之前有2,而且2没有在已有的集合中,此时加2',就会重复

- 要记得sort!
- 去重判断：[1,2,2] ➡],[1],[1,2],[1,2,2],[2],[2,2
- 判断当前数是否和前面数一致
- 且该数不是在startIndex，在startIndex的数还是可以取
- 不加 i > startIndex的结果：],[1],[1,2],[2

Combination Sum

Combination Sum - LeetCode

- 加一个变量 remain 离target还剩多少
- startIndex 从i开始而不是i + 1开始 同一个数可以选多次
- helper头要判断remain是否0 符合条件可加入results
- for循环里要判断remain减完是否还是 >= 0 不是的话直接跳过
- 还可以做的：去重 因为一个数可以取多次 遇到跟上一个数字重复可以跳过

Combination Sum II

Combination Sum II - LeetCode

用到subsets ii中的去重：

- i != startIndex
- 且nums[i] == nums[i -1] 就跳过

Palindrome Partitioning

Palindrome Partitioning - LeetCode

- 本质上是切割点的组合
- 每个切割点取或不取
- 终止条件是startIndex等于s.length()，因为substring的后一个参数是excluded的
- 所谓的切割点在代码里是 `s.substring(startIndex, i + 1)` 中的i + 1，就是不取的那个index

Permutations

Permutations - LeetCode

- 只有个数达到数组长度才输出
- for (int i = 0; i < nums.length; i++)
  - if (permutation.contains(nums[i])) continue;

Word Search

Word Search - LeetCode

- 两层for循环 如果首字母相同的话进入helper
- helper的参数：
  - board
  - 当前x y
  - 当前所查char的start index
  - string word
- 退出条件：start = word.length() 也就是start等于最大index+1
- 原来把往四边DFS的边界条件写在了for循环4次里 不满足就continue
  - 这样如果整个board只有一个字母的话 没法进入下一层递归使得start == word.length() 结果就错了
  - 讨论版里的做法是把边界判断、visited判断和字符是否相等都放在了外面的开头 递归出口的下面 全部为false
- 要记得写一个hash变量 来记录该点走过没有 避免重复查找 然后helper尾再把该点visited改回false

Evaluate Division

Evaluate Division

- 面经题
- 把公式关系抽象成图
  - a / b = 2.0, b / c = 3.0
  - a ➡ b 值为2
  - b ➡ a 值为1/2
- 我的做法是邻接表
  - HashMap<String, Map<String, Double>>
  - 对每一个点存其指向的邻居及其值
  - 对每一个公式 a / b = 2.0
    - 存成 a: (b, 2.0) b: (a, 0.5)
- 一个很容易忽视的错: Map<String, Double> 里面的Double要大写 像int和Integer
- 用DFS backtrack
- 走没走过的hash用set存 这个set每个query开始前都要清空 否则后面的结果都是得不到 ➡ 错在这里

- DFS helper函数
  - 参数: start, end, result, map, set
  - start每次都改变
  - 退出条件: start和end是同一个string 可以输出result
  - 不符合的退出条件(返回0.0):
    - set已经visited
    - map里没有start这个值 ➡ 想不通 建邻接表的时候应该每个node都有的
      - 试了一下不加这条: 处理query里输入的不存在表里的值 没有的话会出问题
  - set记录visited这个点
  - 到map里找到start对应的value: 邻居列表
  - 遍历邻居 把邻居带的值value乘上result 递归调用自己
    - neighbor, end, result*value, map, set
  - 如果返回的不是0.0 就是我们要的结果了
  - 记得set把visited的这个点挪掉
- 外部调用helper
  - query的起点 终点 1.0(用来乘)

113. Path Sum II

Path Sum II - LeetCode

- 外部调用的时候记得先把root.val加入路径里 否则结果集都会少一个root的value

```
ArrayList<Integer> path = new ArrayList<>();
path.add(root.val);
// helper(root, root.val, sum, new ArrayList<Integer>(), results);
helper(root, root.val, sum, path, results);
```

17. Letter Combinations of a Phone Number

Letter Combinations of a Phone Number - LeetCode

- 还是写helper做backtracking
- 用String[]建一个dictionary index 0和1对应空 2开始存可选字母
- 把输入的String数字 “23” 转换成int[]

```
for (int i = 0; i < int_digits.length; i++) {
  int_digits[i] = digits.charAt(i) - '0';
}
```

- helper参数: index, digits数组, 结果集results, StringBuilder, 字典数组
- helper递归出口: sb.length()到了输入数字的长度 可以把结果加入results
- 得到当前数字对应的candidates 2 ➡ “abc”
- sb.append
- helper
- sb.deleteCharAt(sb.length() - 1)
- 还是犯了个蠢的错误: String类的长度获取是 s.length() 不是s.length

10. Regular Expression Matching

Regular Expression Matching - LeetCode

- 参考了答案的递归写法
- 判断pattern空没 如果空 返回s空没空
- s空不空的判断再firstMatch变量里
- firstMatch =
  - !s.isEmpty()
  - &&
  - ((s.charAt(0) == p.charAt(0)) || p.charAt(0) == '.')
- 如果pattern长度大于等于2 且第二位有 *
  - * 表示字符a出现0次或者无数次
  - 两种情况取或
    - n个match: 如果当前位能match 看s的下一位还是不是继续match (firstMatch && isMatch(s.substring(1), p))
    - 0个match: 无论当前位match不match 都直接跳过 `[a*]` 这个组合 isMatch(s, p.substring(2))
  - 不能写成if firstMatch else的形式 报错 test case: `aaa a*a`
    - 即使firstMatch是true 也可以走跳过*的这条路才对
- 否则
  - 返回firstMatch && 递归两个字符各进一位
  - 用&&不用||: 如果当前两位不对应 就没有看下去的必要了 后面的结果都不重要了

140. Word Break II

Word Break II - LeetCode

- DFS + hash map
- hashmap的作用是避免重复做已经解析过的substring 防止TLE
  - 有点像找重复子树 找到一棵子树先序列化存进map里

- helper
  - 参数: 当前的substring, dict, map
  - map里如果已经有当前string的对应结果 就不需要再继续递归了 直接输出结果
  - 如果当前子串为空 往results里加一个空string return
  - for dict里的每一个词word 查当前string s有没有以该词开头
    - 有的话就helper进s.substring(word.length())
    - helper返回该substring的所有可能输出
    - 对于每一条输出string 在前面append当前word
      - 要注意判断加空格 " " 还是加空string ""
  - 记得把当前helper输出的results加入map

329. Longest Increasing Path in a Matrix

Longest Increasing Path in a Matrix - LeetCode

- 普通DFS
- 判断上下左右是否能走通 如果下一个值valid/hash=false 且值比当前值大 就可以递归helper下去
- 一个点的四条路可能有四个candidate路线 记录最长的一条 加上自己 更新擂台全局变量maxLen
- 优化: 加一个lengths二维数组 helper做完返回当前点开始的最长length之前 把这个length存在数组内对应坐标上 这样别的点如果碰到这条路可以走 就可以省去重新做递归的时间 直接取表里已经记录的该点开始的最长长度