刷人利器——深度优先搜索

主讲人 令狐冲
课程版本 v7.0

什么时候使用 DFS？

在之前的课程中，我们知道了 Binary Tree 的问题大部分都是 DFS

今天的课程中，我们将更深入的讨论 DFS 的使用场景

独孤九剑 —— 破索式

碰到让你找所有方案的题，基本可以确定是 DFS

除了二叉树以外的 90% DFS 的题，要么是排列，要么是组合

找所有满足某个条件的方案

找到图中的所有满足条件的**路径**

路径 = 方案 = 图中节点的排列组合

很多题不像二叉树那样直接给你一个图（二叉树也是一个图）

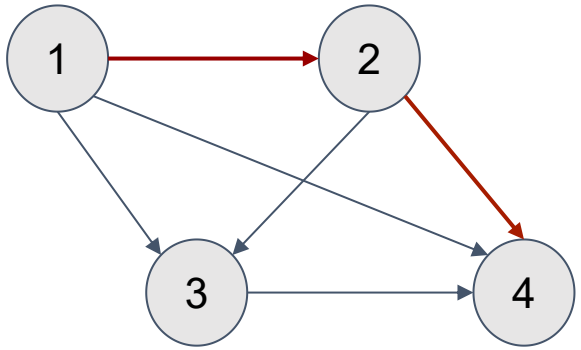
点、边、路径是需要你自己去分析的

案例一：找出一个集合的所有子集

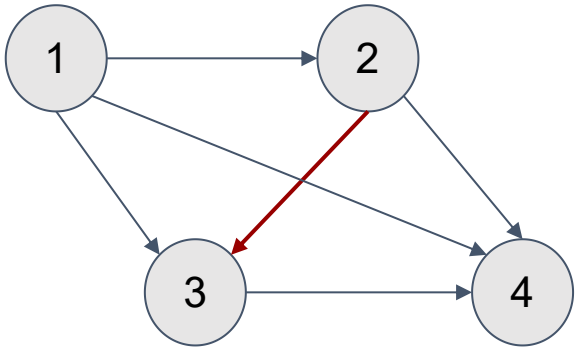
点：集合中的元素

边：元素与元素之间用**有向边**连接，小的点指向大的点（为了避免选出 12 和 21 这种重复集合）

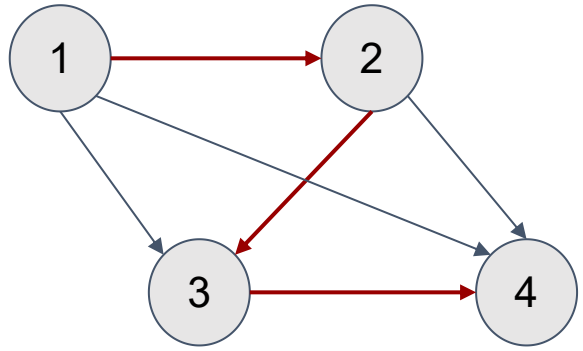
路径：= 子集 = 图中任意点出发到任意点结束的一条路径



{1,2,4}



{2,3}



{1,2,3,4}

找N个数组成的全排列
Permutation

动动手，该如何构建图？

如 123 的全排列有 6 个

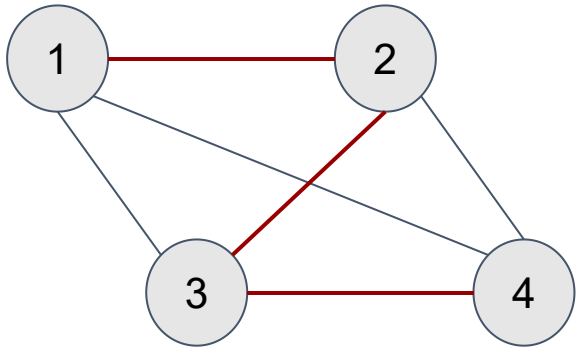
123, 132, 213, 231, 312, 321

案例二：求出 N 个数组成的全排列

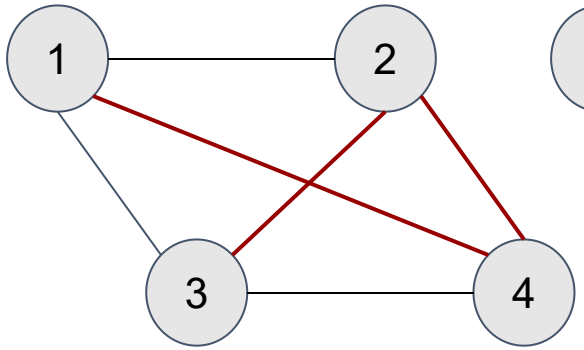
点：每个数为一个点

边：任意两两点之间都有连边，且为无向边

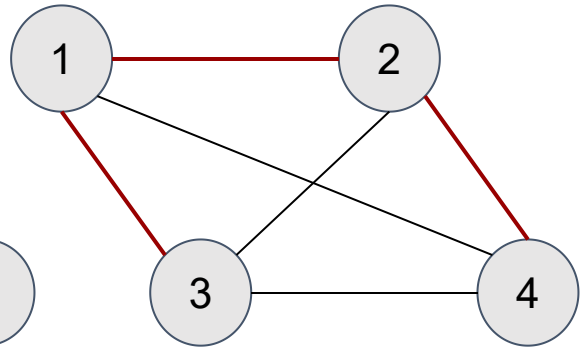
路径：= 排列 = 从任意点出发到任意点结束经过每个点一次且仅一次的路径



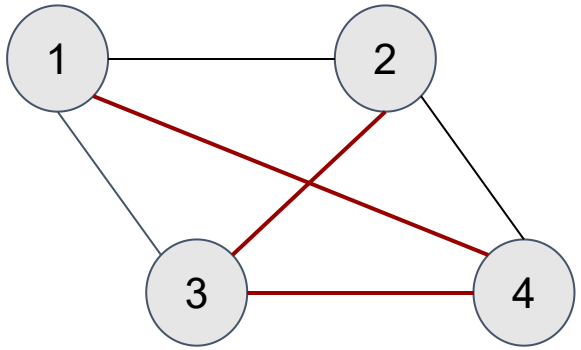
1234



1432



3124



2341

BFS vs DFS

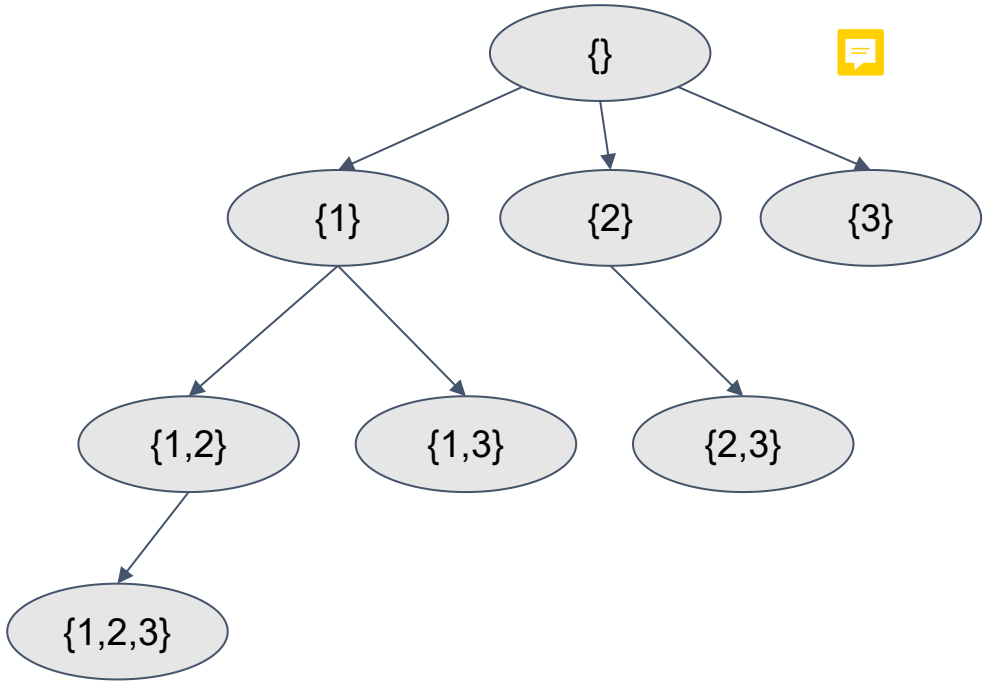
找所有方案的问题是否可以使用 BFS？

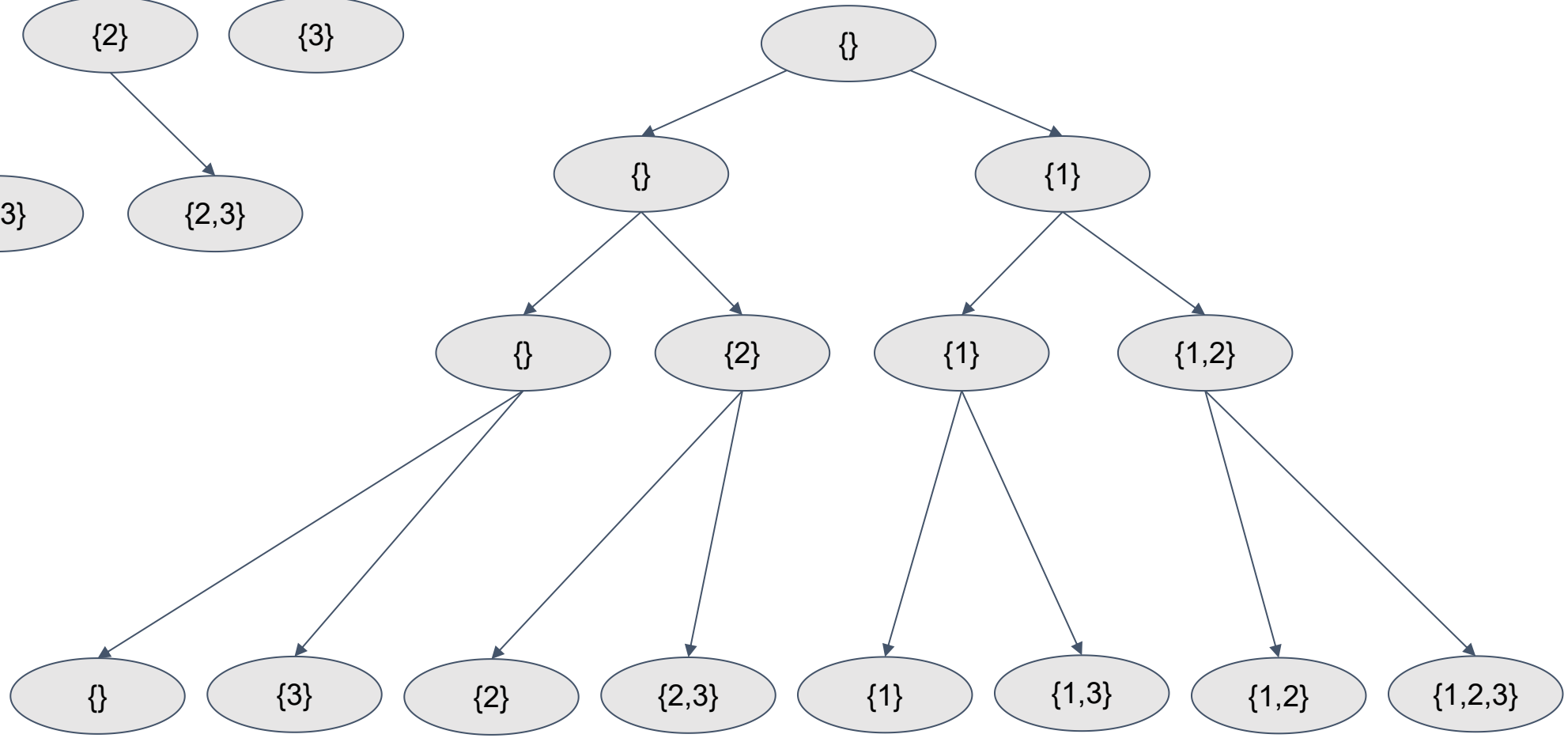
找路径 → 找点

改变“点”的定义

可以将找所有路径问题变为找所有点的问题

全子集问题的另外两种画图方法（找所有点）





BFS vs DFS

宽度优先搜索的空间复杂度取决于宽度

深度优先搜索的空间复杂度取决于深度

什么是递归 Recursion？

函数 Function 自己调用自己

Recursion 是代码的实现方式，并不算是一种算法

（也有一些书籍认为递归是算法，但是这样说并不准确）

递归在算法中干了啥

递归就是当多重循环层数不确定的时候

一个更优雅的实现多重循环的方式

递归干了啥


```
if n == 1:
    for i in range(1, n + 1):
        ...

if n == 2:
    for i in range(1, n + 1):
        for j in range(1, n + 1):
            if i != j:
                ...

if n == 3:
    for i in range(1, n + 1):
        for j in range(1, n + 1):
            if i != j:
                for k in range(1, n + 1):
                    if k != i and k != j:
                        ...
```

```
def recursion(self, n, visited, path):
    if len(path) == n:
        # do something
        return

    for i in range(1, n + 1):
        if i not in visited:
            path.append(i)
            visited.add(i)
            self.recursion(n, visited, path)
            path.pop()
            visited.remove(i)
```

递归干了啥


```
if (n == 1) {
    for (int i = 1; i <= n; i++) {
        ...
    }
}

if (n == 2) {
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            if (i != j) {
                ...
            }
        }
    }
}

if (n == 3) {
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            if (i != j) {
                for (int k = 1; k <= n; k++) {
                    if (k != i && k != j) {
                        ...
                    }
                }
            }
        }
    }
}
```

```
private void recursion(int n, Set<Integer> visited, List<Integer> path) {
    if (path.size() == n) {
        // do something
        return;
    }

    for (int i = 1; i <= n; i++) {
        if (!visited.contains(i)) {
            path.add(i);
            visited.add(i);
            recursion(n, visited, path);
            path.remove(path.size() - 1);
            visited.remove(i);
        }
    }
}
```

递归三要素

一般来说，如果面试官不特别要求的话，DFS都可以使用递归(Recursion)的方式来实现。

递归三要素是实现递归的重要步骤：

- 递归的定义
- 递归的拆解
- 递归的出口

组合问题

问题模型：求出所有满足条件的“组合”。

判断条件：组合中的元素是顺序无关的。

时间复杂度：与 2^n 相关。

排列问题

问题模型：求出所有满足条件的“**排列**”。

判断条件：组合中的元素是顺序“**相关**”的。

时间复杂度：与 n! 相关。

DFS 时间复杂度通用计算公式

O(方案个数 * 构造每个方案的时间)

排列问题 = O(n! * n)

组合问题 = O(2^n * n)

String Permutation II

www.lintcode.com/problem/string-permutation-ii

www.jiuzhang.com/solutions/string-permutation-ii

字母换数字，换汤不换药

搜索去重的诀窍 —— 选代表

错误方法：把所有方案都放在 HashSet 里去重

正确方法：在每组重复方案中选出代表方案

代码


```java
public List<String> stringPermutation2(String str) {
    char[] chars = str.toCharArray();
    Arrays.sort(chars);
    boolean[] visited = new boolean[chars.length];
    List<String> permutations = new ArrayList<>();

    dfs(chars, visited, "", permutations);
    return permutations;
}
```

```java
// 递归的定义：找到所有 permutation 开头的排列
private void dfs(char[] chars,
                 boolean[] visited,
                 String permutation,
                 List<String> permutations) {
    // 递归的出口：当我找到一个完整的排列
    if(chars.length == permutation.length()){
        permutations.add(permutation);
        return;
    }

    // 递归的拆解：基于当前的前缀，下一个字符放啥
    for(int i = 0; i < chars.length; i++) {
        if(visited[i]) {
            continue;
        }
        // 去重：不同位置的同样的字符，必须按照顺序用。
        // a' a" b
        // => a' a" b => √
        // => a" a' b => x
        // 不能跳过一个a选下一个a
        if(i > 0 && chars[i] == chars[i -1] && !visited[i-1]) {
            continue;
        }

        // make changes
        visited[i] = true;

        // 找到所有 permutation 开头的排列
        // 找到所有 "a" 开头的
        dfs(chars, visited, permutation + chars[i], permutations);

        // backtracking
        visited[i] = false;
    }
}
```

代码


```python
def stringPermutation2(self, str):
    chars = sorted(list(str))
    visited = [False] * len(chars)
    permutations = []
    self.dfs(chars, visited, [], permutations)
    return permutations
```

```python
# 递归的定义：找到所有 permutation 开头的排列
def dfs(self, chars, visited, permutation, permutations):
    # 递归的出口：当我找到一个完整的排列
    if len(chars) == len(permutation):
        permutations.append(''.join(permutation))
        return

    # 递归的拆解：基于当前的前缀，下一个字符放啥
    for i in range(len(chars)):
        # 同一个位置上的字符用过不能在用
        if visited[i]:
            continue
        # 去重：不同位置的同样的字符，必须按照顺序用。
        # a' a" b
        # => a' a" b => √
        # => a" a' b => x
        # 不能跳过一个a选下一个a
        if i > 0 and chars[i] == chars[i - 1] and not visited[i - 1]:
            continue

        # make changes
        visited[i] = True
        permutation.append(chars[i])

        # 找到所有 permutation 开头的排列
        # 找到所有 "a" 开头的
        self.dfs(chars, visited, permutation, permutations)

        # backtracking
        permutation.pop()
        visited[i] = False
```

Combination Sum

http://www.lintcode.com/problem/combination-sum/

http://www.jiuzhang.com/solutions/combination-sum/

问：和 subsets 的区别有哪些？

与 Subsets 比较

- Combination Sum 限制了组合中的数之和
  - 加入一个新的参数来限制
- Subsets 无重复元素，Combination Sum 有重复元素
  - 需要先去重
- Subsets 一个数只能选一次，Combination Sum 一个数可以选很多次
  - 搜索时从 index 开始而不是从 index + 1

回溯与硬拷贝


```python
def combinationSum(self, candidates, target):
    candidates = sorted(list(set(candidates)))
    results = []
    self.dfs(candidates, target, 0, [], results)
    return results

def dfs(self, candidates, target, start, combination, results):
    if target < 0:
        return
    if target == 0:
        return results.append(list(combination))

    for i in range(start, len(candidates)):
        combination.append(candidates[i])
        self.dfs(
            candidates,
            target - candidates[i],
            i,
            combination,
            results,
        )
        combination.pop()
```

```python
def combinationSum(self, candidates, target):
    candidates = sorted(list(set(candidates)))
    results = []
    self.dfs(candidates, target, 0, [], results)
    return results

def dfs(self, candidates, target, start, combination, results):
    if target < 0:
        return
    if target == 0:
        return results.append(combination)

    for i in range(start, len(candidates)):
        self.dfs(
            candidates,
            target - candidates[i],
            i,
            combination + [candidates[i]],
            results,
        )
```

代码


```java
public List<List<Integer>> combinationSum(int[] candidates, int target) {
    List<List<Integer>> results = new ArrayList<>();
    // 集合为空
    if (candidates.length == 0) {
        return results;
    }
    // 排序和去重
    candidates = removeDuplicates(candidates);
    // dfs
    dfs(candidates, target, 0, new ArrayList<Integer>(), results);
    return results;
}

private int[] removeDuplicates(int[] candidates) {
    //排序
    Arrays.sort(candidates);
    //去重
    int index = 0;
    for (int i = 0; i < candidates.length; i++) {
        if (candidates[i] != candidates[index]) {
            candidates[++index] = candidates[i];
        }
    }
    int[] candidatesNew = new int[index + 1];
    for (int i = 0; i < index + 1; i++) {
        candidatesNew[i] = candidates[i];
    }
    return candidatesNew;
}
```

```java
private void dfs(int[] candidates, int target, int start, List<Integer>
    combination, List<List<Integer>> results) {
    // 到达边界
    if (target == 0) {
        results.add(new ArrayList<Integer>(combination));
        return;
    }
    // 递归的拆解：挑一个数放入current
    for (int i = start; i < candidates.length; i++) {
        // 剪枝
        if (target < candidates[i]) {
            break;
        }
        combination.add(candidates[i]);
        dfs(candidates, target - candidates[i], i, combination, results);
        combination.remove(combination.size() - 1);
    }
}
```

k Sum II

http://www.lintcode.com/problem/k-sum-ii/

http://www.jiuzhang.com/solution/k-sum-ii/

找出所有 k 个数之和 = target 的组合

代码


```python
def kSumII(self, A, k, target):
    A = sorted(A)
    subsets = []
    self.dfs(A, 0, k, target, [], subsets)
    return subsets

def dfs(self, A, index, k, target, subset, subsets):
    if k == 0 and target == 0:
        subsets.append(list(subset))
        return

    if k == 0 or target <= 0:
        return

    for i in range(index, len(A)):
        subset.append(A[i])
        self.dfs(A, i + 1, k - 1, target - A[i], subset, subsets)
        subset.pop()
```

```java
public List<List<Integer>> kSumII(int[] A, int k, int target) {
    Arrays.sort(A);
    List<List<Integer>> subsets = new ArrayList();

    dfs(A, 0, k, target, new ArrayList<Integer>(), subsets);

    return subsets;
}

private void dfs(int[] A, int index, int k, int target,
                 List<Integer> subset,
                 List<List<Integer>> subsets) {
    if (k == 0 && target == 0) {
        subsets.add(new ArrayList<Integer>(subset));
        return;
    }
    if (k == 0 || target <= 0) {
        return;
    }

    for (int i = index; i < A.length; i++) {
        subset.add(A[i]);
        dfs(A, i + 1, k - 1, target - A[i], subset, subsets);
        subset.remove(subset.size() - 1);
    }
}
```

休息一会儿

Letter Combinations of Phone Number

http://www.lintcode.com/problem/letter-combinations-of-a-phone-number/

http://www.jiuzhang.com/solution/letter-combinations-of-a-phone-number/

什么是点？什么是边？什么是路径？

代码


```python
KEYBOARD = {
    '2': 'abc',
    '3': 'def',
    '4': 'ghi',
    '5': 'jkl',
    '6': 'mno',
    '7': 'pqrs',
    '8': 'tuv',
    '9': 'wxyz',
}

class Solution:
    """
    @param digits: A digital string
    @return: all posible letter combinations
    """
    def letterCombinations(self, digits):
        if not digits:
            return []

        results = []
        self.dfs(digits, 0, [], results)

        return results

    def dfs(self, digits, index, chars, results):
        if index == len(digits):
            results.append(''.join(chars))
            return

        for letter in KEYBOARD[digits[index]]:
            chars.append(letter)
            self.dfs(digits, index + 1, chars, results)
            chars.pop()
```

常量表

```java
public List<String> letterCombinations(String digits) {
    List<String> combinations = new ArrayList();
    if (digits == null || digits.length() == 0) {
        return combinations;
    }

    dfs(digits, 0, "", combinations);
    return combinations;
}

private void dfs(String digits,
                 int index,
                 String combination,
                 List<String> combinations) {
    if (index == digits.length()) {
        combinations.add(combination);
        return;
    }

    int digit = digits.charAt(index) - '0';
    for (int i = 0; i < KEYBOARD[digit].length(); i++) {
        dfs (
            digits,
            index + 1,
            combination + KEYBOARD[digit].charAt(i),
            combinations
        );
    }
}
```

```java
public static String[] KEYBOARD = {
    "",
    "",
    "",
    "abc",
    "def",
    "ghi",
    "jkl",
    "mno",
    "pqrs",
    "tuv",
    "wxyz",
};
```

Follow up

如果有一个词典（Dictionary）
要求组成的单词都是词典里的
如何优化？

面试评分标准

Strong Hire: 两问的 DFS 都能写出来，第二问使用 Trie 或者 Hash 都可以，无需提示

Hire / Weak Hire: 写完第一问的 DFS，第二问给出正确思路和方法，但是没写完，需要部分提示

No Hire: 第一问没写完，或者 bug 很多

Strong No: 没思路不会做

矩阵上的 DFS

http://www.lintcode.com/problem/word-search-ii/

http://www.jiuzhang.com/solution/word-search-ii/

矩阵（Matrix）也是图

给出一个由小写字母组成的矩阵和一个字典。找出所有同时在字典和矩阵中出现的单词。

使用哪种搜索策略

A: for 词典里的每个单词 { check 单词是否在矩阵里 }

B: for 矩阵的每个单词 { check 单词是否在词典里}

代码


```java
public List<String> wordSearchII(char[][] board, List<String> words) {
    if (board == null || board.length == 0) {
        return new ArrayList<>();
    }
    if (board[0] == null || board[0].length == 0) {
        return new ArrayList<>();
    }

    boolean[][] visited = new boolean[board.length][board[0].length];
    Map<String, Boolean> prefixIsWord = getPrefixSet(words);
    Set<String> wordSet = new HashSet<>();

    for (int i = 0; i < board.length; i++) {
        for (int j = 0; j < board[i].length; j++) {
            visited[i][j] = true;
            dfs(board, visited, i, j, String.valueOf(board[i][j]), prefixIsWord, wordSet);
            visited[i][j] = false;
        }
    }

    return new ArrayList<String>(wordSet);
}

private Map<String, Boolean> getPrefixSet(List<String> words) {
    Map<String, Boolean> prefixIsWord = new HashMap<>();
    for (String word : words) {
        for (int i = 0; i < word.length() - 1; i++) {
            String prefix = word.substring(0, i + 1);
            if (!prefixIsWord.containsKey(prefix)) {
                prefixIsWord.put(prefix, false);
            }
        }
        prefixIsWord.put(word, true);
    }

    return prefixIsWord;
}
```

```java
public static int[] DX = {0, 1, -1, 0};
public static int[] DY = {1, 0, 0, -1};

private boolean inside(char[][] board, int x, int y) {
    return x >= 0 && x < board.length && y >= 0 && y < board[0].length;
}

private void dfs(char[][] board,
                 boolean[][] visited,
                 int x,
                 int y,
                 String word,
                 Map<String, Boolean> prefixIsWord,
                 Set<String> wordSet) {
    if (!prefixIsWord.containsKey(word)) {
        return;
    }

    if (prefixIsWord.get(word)) {
        wordSet.add(word);
    }

    for (int i = 0; i < 4; i++) {
        int adjX = x + DX[i];
        int adjY = y + DY[i];

        if (!inside(board, adjX, adjY) || visited[adjX][adjY]) {
            continue;
        }

        visited[adjX][adjY] = true;
        dfs(board, visited, adjX, adjY, word + board[adjX][adjY], prefixIsWord, wordSet);
        visited[adjX][adjY] = false;
    }
}
```

代码


```python
DX = [0, 1, -1, 0]
DY = [1, 0, 0, -1]

class Solution:
    def wordSearchII(self, board, words):
        if board is None or len(board) == 0:
            return []
        if board[0] is None or len(board[0]) == 0:
            return []

        visited = [[0] * len(board[0]) for _ in range(len(board))]
        prefix_is_word = self.get_prefix_set(words)
        word_set = set()

        for i in range(len(board)):
            for j in range(len(board[i])):
                visited[i][j] = True
                self.dfs(board, visited, i, j, board[i][j], prefix_is_word, word_set)
                visited[i][j] = False

        return list(word_set)

    def get_prefix_set(self, words):
        prefix_is_word = {}
        for word in words:
            for i in range(len(word)):
                prefix = word[:i + 1]
                if prefix not in prefix_is_word:
                    prefix_is_word[prefix] = False
                prefix_is_word[word] = True

        return prefix_is_word
```

```python
def inside(self, board, x, y):
    return x >= 0 and x < len(board) and y >= 0 and y < len(board[0])

def dfs(self, board, visited, x, y, word, prefix_is_word, word_set):
    if word not in prefix_is_word:
        return

    if prefix_is_word[word]:
        word_set.add(word)

    for i in range(4):

        adjX = x + DX[i]
        adjY = y + DY[i]
        if not self.inside(board, adjX, adjY) or visited[adjX][adjY]:
            continue

        visited[adjX][adjY] = True
        self.dfs(board,
                 visited,
                 adjX,
                 adjY,
                 word + board[adjX][adjY]
                 , prefix_is_word, word_set)
        visited[adjX][adjY] = False
```

如何使用 hashset 来做前缀查询？

wordSet = {“hello”, “world”}

prefixSet = {“h”, “he”, “hel”, “hell”, “hello”,

“w”, “wo”, “wor”, “worl”, “world”}

Follow up

https://www.lintcode.com/problem/word-search-iii/

https://www.lintcode.com/problem/boggle-game/ (Airbnb)

在矩阵上同时能够找出最多多少个不重叠的单词？

本题不作为面试要求

找所有的最短路径

http://www.lintcode.com/problem/word-ladder-ii/

http://www.jiuzhang.com/solutions/word-ladder-ii/

算法1: 直接 DFS

通过打擂台的方式记录下所有最短路径

缺点是什么？

算法2: 直接 BFS

队列里存什么？

A: 单词节点

B: 单词路径

算法2: 直接 BFS

队列里存什么？

A: 单词节点

B: 单词路径

如果使用 BFS 找所有的路径，那么队列里放的就是路径

最优算法: BFS + DFS

BFS: 求出所有点到终点的距离

DFS: 沿着离终点越来越近的路线找到所有路径

代码


```java
public List<List<String>> findLadders(String start, String end, Set<String> dict) {
    dict.add(start);
    dict.add(end);
    Map<String, Set<String>> indexes = buildIndexes(dict);

    Map<String, Integer> distance = bfs(end, indexes);

    List<List<String>> results = new ArrayList<List<String>>();
    List<String> path = new ArrayList<String>();
    path.add(start);
    dfs(start, end, distance, indexes, path,results);

    return results;
}

private Map<String, Set<String>> buildIndexes(Set<String> dict) {
    Map<String, Set<String>> indexes = new HashMap<String, Set<String>>();
    for (String word : dict) {
        for (int i = 0; i < word.length(); i++) {
            String key = word.substring(0, i) + "%" + word.substring(i +  1);
            if (indexes.containsKey(key)) {
                indexes.get(key).add(word);
            } else {
                indexes.put(key, new HashSet<>());
                indexes.get(key).add(word);
            }
        }
    }
    return indexes;
}
```

```java
private Map<String, Integer> bfs(String end, Map<String, Set<String>> indexes) {
    Map<String, Integer> distance = new HashMap<String, Integer>();
    distance.put(end, 0);
    Queue<String> queue = new ArrayDeque<String>();
    queue.offer(end);
    while (!queue.isEmpty()) {
        String word = queue.poll();
        for (String nextWord : getNextWords(word, indexes)) {
            if (!distance.containsKey(nextWord)) {
                distance.put(nextWord, distance.get(word) + 1);
                queue.offer(nextWord);
            }
        }
    }
    return distance;
}

private List<String> getNextWords(String word, Map<String, Set<String>> indexes) {
    List<String> words = new ArrayList<String>();
    for (int i = 0; i < word.length(); i++) {
        String key = word.substring(0, i) + "%" + word.substring(i + 1);
        for (String w : indexes.get(key)) {
            words.add(w);
        }
    }
    return words;
}
```

代码


```
private void dfs(String curt,
                 String target,
                 Map<String, Integer> distance,
                 Map<String, Set<String>> indexes,
                 List<String> path,
                 List<List<String>> results) {
    if (curt.equals(target)) {
        results.add(new ArrayList<String>(path));
        return;
    }

    for (String word : getNextWords(curt, indexes)) {
        if (distance.get(word) != distance.get(curt) - 1) {
            continue;
        }
        path.add(word);
        dfs(word, target, distance, indexes, path, results);
        path.remove(path.size() - 1);
    }

}
```

代码


```python
from collections import deque
class Solution:
    def findLadders(self, start, end, dict):
        dict.add(start)
        dict.add(end)
        indexes = self.build_indexes(dict)

        distance = self.bfs(end, indexes)

        results = []
        self.dfs(start, end, distance, indexes, [start], results)

        return results

    def build_indexes(self, dict):
        indexes = {}
        for word in dict:
            for i in range(len(word)):
                key = word[:i] + '%' + word[i + 1:]
                if key in indexes:
                    indexes[key].add(word)
                else:
                    indexes[key] = set([word])
        return indexes
```

```python
def bfs(self, end, indexes):
    distance = {end: 0}
    queue = deque([end])
    while queue:
        word = queue.popleft()
        for next_word in self.get_next_words(word, indexes):
            if next_word not in distance:
                distance[next_word] = distance[word] + 1
                queue.append(next_word)
    return distance

def get_next_words(self, word, indexes):
    words = []
    for i in range(len(word)):
        key = word[:i] + '%' + word[i + 1:]
        for w in indexes.get(key, []):
            words.append(w)
    return words

def dfs(self, curt, target, distance, indexes, path, results):
    if curt == target:
        results.append(list(path))
        return

    for word in self.get_next_words(curt, indexes):
        if distance[word] != distance[curt] - 1:
            continue
        path.append(word)
        self.dfs(word, target, distance, indexes, path, results)
        path.pop()
```

后续学习

- 在第 30 章的互动课中学习非递归的方式实现排列组合类 DFS
  - 如何求下一个排列
  - 通用的非递归算法
- 在第 31 章的互动课中学习两道经典的 DFS 题
  - N 皇后问题及程序结构的艺术
  - 数独及搜索顺序的优化算法