

隐式图深度优先搜索 Implicit Graph DFS

课程版本 v6.1

主讲 令狐冲



扫描二维码关注微信/微博
获取最新面试题及权威解答

微信: [ninechapter](#)

微博: <http://www.weibo.com/ninechapter>

知乎: <http://zhuankan.zhihu.com/jiuzhang>

官网: <http://www.jiuzhang.com>

- 请在随课教程中自学如下先修知识：
 - <http://www.jiuzhang.com/tutorial/algorithm/19>
 - 组合类搜索入门问题全子集问题 (Subset) 及其 4 种解法 (课前只需学习递归的 2 种解法)
 - 什么是 Deep copy, 为什么需要 Deep copy?
- 深度优先搜索 DFS 的判断条件 (什么时候用 DFS)
- 递归三要素
- 组合类搜索入门题
- 深度优先搜索的时间复杂度分析

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）

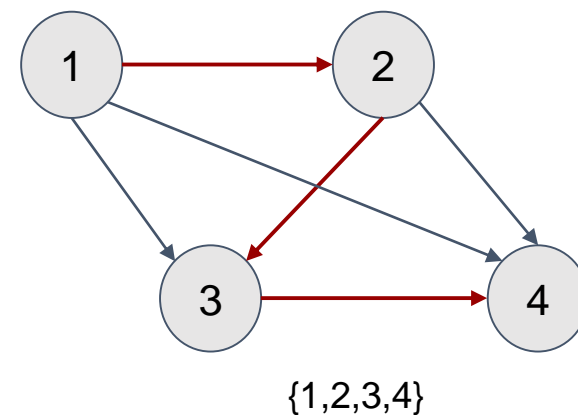
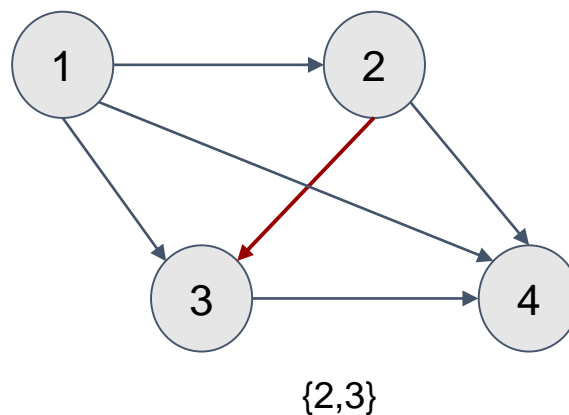
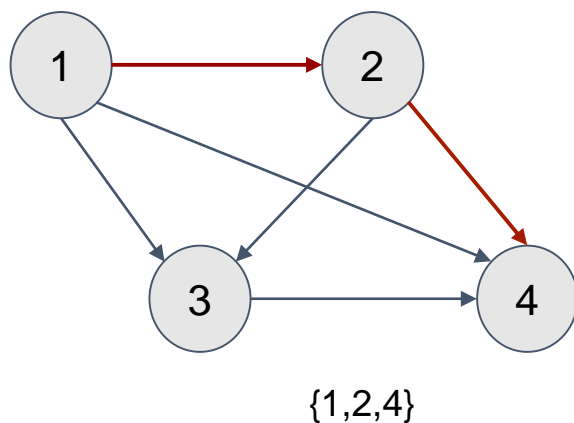
点、边、路径是需要你自己去分析的

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



找N个数组成的全排列

动动手，该如何构建图？

如 123 的全排列有 6 个

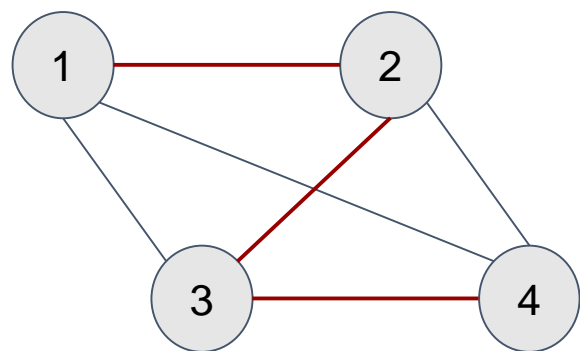
123, 132, 213, 231, 312, 321

案例二：求出 N 个数组成的全排列

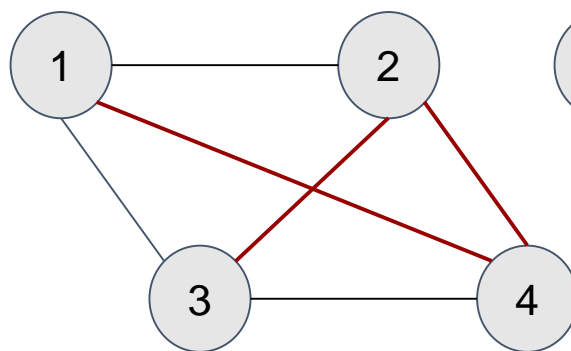
点：每个数为一个点

边：任意两两点之间都有连边，且为无向边

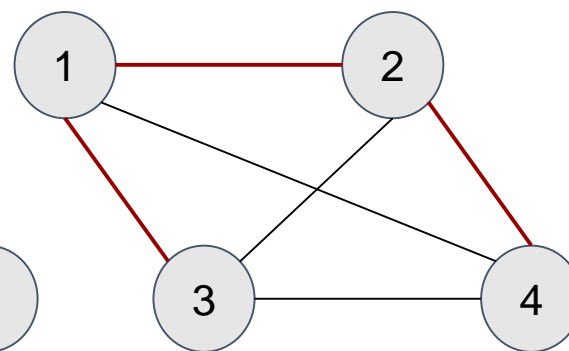
路径：= 排列 = 从任意点出发到任意点结束经过每个点一次且仅一次的路径



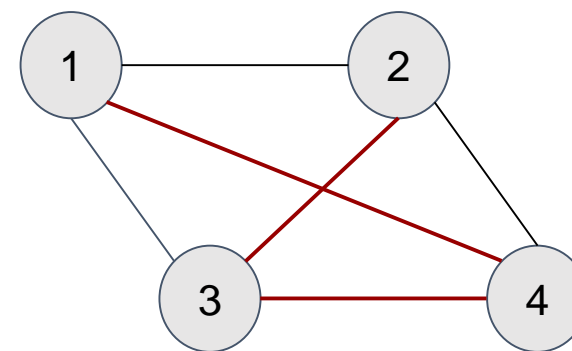
1234



1432



3124



2341

BFS vs DFS

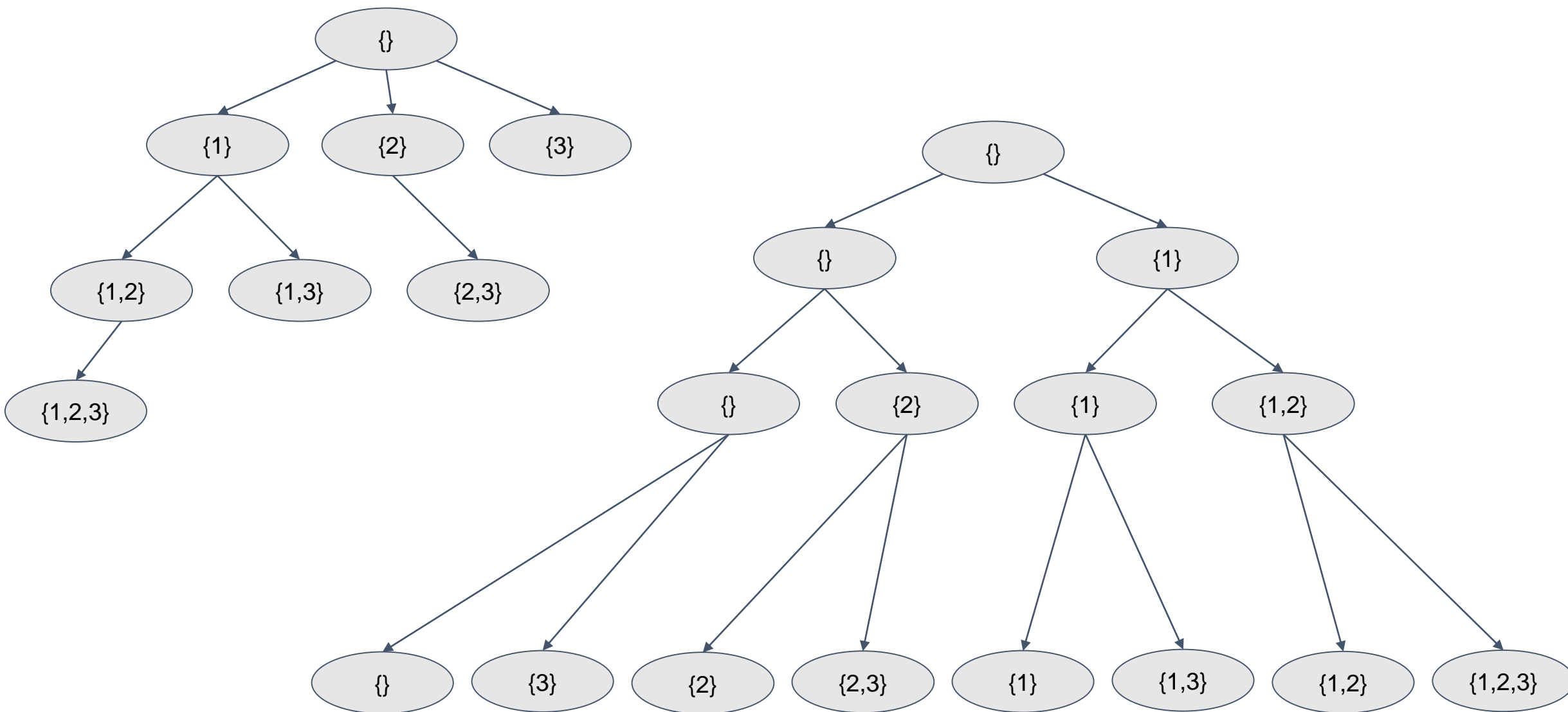
找所有方案的问题是否可以使用 BFS?

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）



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

```
1 ▾ if n == 1:
2 ▾     for i in range(1, n + 1):
3 ▾         ...
4
5 ▾ if n == 2:
6 ▾     for i in range(1, n + 1):
7 ▾         for j in range(1, n + 1):
8 ▾             if i != j:
9 ▾                 ...
10
11 ▾ if n == 3:
12 ▾     for i in range(1, n + 1):
13 ▾         for j in range(1, n + 1):
14 ▾             if i != j:
15 ▾                 for k in range(1, n + 1):
16 ▾                     if k != i and k != j:
17 ▾                         ...
```

```
1 ▾ def recursion(self, n, visited, path):
2 ▾     if len(path) == n:
3 ▾         # do something
4 ▾         return
5
6 ▾     for i in range(1, n + 1):
7 ▾         if i not in visited:
8 ▾             path.append(i)
9 ▾             self.recursion(n, visited, path)
10 ▾             path.pop()
```

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。

通用的DFS时间复杂度计算公式

$O(\text{答案个数} * \text{构造每个答案的时间})$

<http://www.jiuzhang.com/qa/2994/>

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？

- 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

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

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

休息 5 分钟

Take a break

N Queens

<http://www.lintcode.com/problem/n-queens/>

<http://www.jiuzhang.com/solutions/n-queens/>

另一种问法：问方案总数 (N Queens II)

下一个排列

<http://www.lintcode.com/problem/next-permutation/>

<http://www.lintcode.com/problem/next-permutation-ii/>

排列的顺序

<http://www.lintcode.com/problem/permutation-index/>

<http://www.lintcode.com/problem/permutation-index-ii/>

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？

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?

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

Word Search II

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

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

矩阵 (Matrix) 也是图

Word Ladder II

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

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

求**所有**的**最短**路径