极客时间算法训练营
第四课
递归、分治

李煜东

《算法竞赛进阶指南》作者

目录

1. 递归的本质与基本实现形式

2. 递归题目实战

3. 分治：子问题的划分与合并

4. 分治算法的应用

递归的本质与基本实现形式

递归 Recursion

如何理解递归？

- 函数自身调用自身
- 通过函数体来进行的循环
- 以自相似的方法重复进行的过程

递归 Recursion

计算 n!

n! = 1 * 2 * 3 * ... * n

```
def factorial(n):
    if n <= 1:
        return 1
    return n * factorial(n - 1)
```

递归 Recursion

factorial(6)

6 * factorial(5)

6 * (5 * factorial(4))

6 * (5 * (4 * factorial(3)))

6 * (5 * (4 * (3 * factorial(2))))

6 * (5 * (4 * (3 * (2 * factorial(1)))))

6 * (5 * (4 * (3 * (2 * 1))))

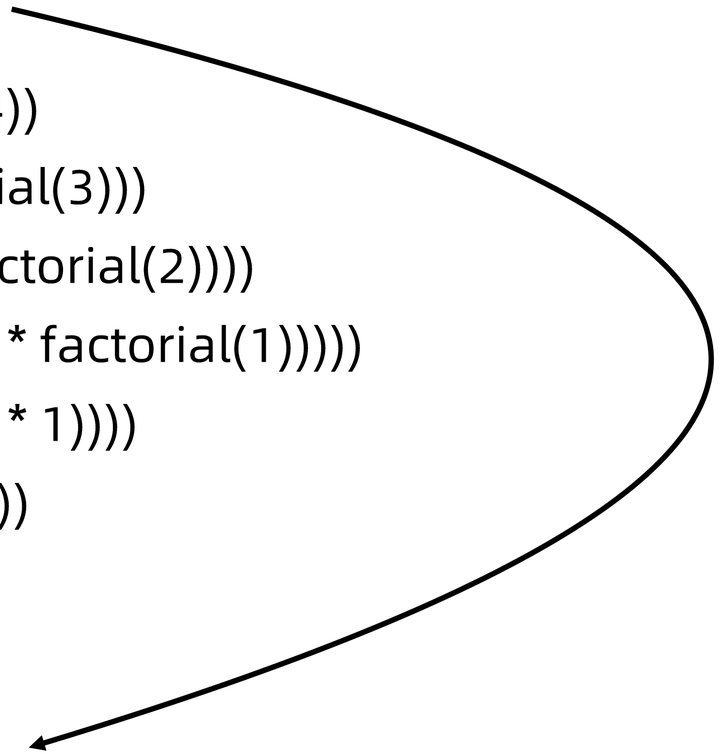
6 * (5 * (4 * (3 * 2)))

6 * (5 * (4 * 6))

6 * (5 * 24)

6 * 120

720



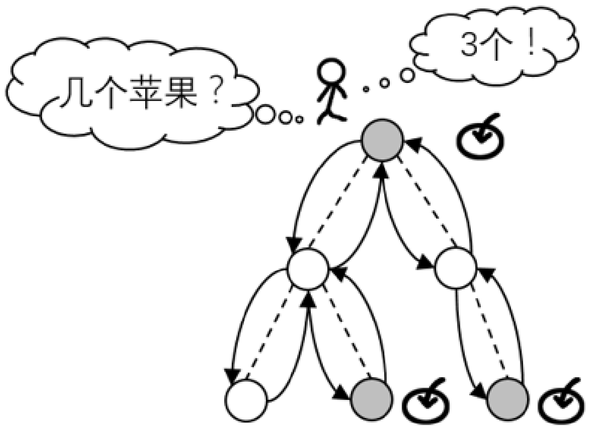
递归 Recursion

啊喂，求 n! 用递推它不香吗.....

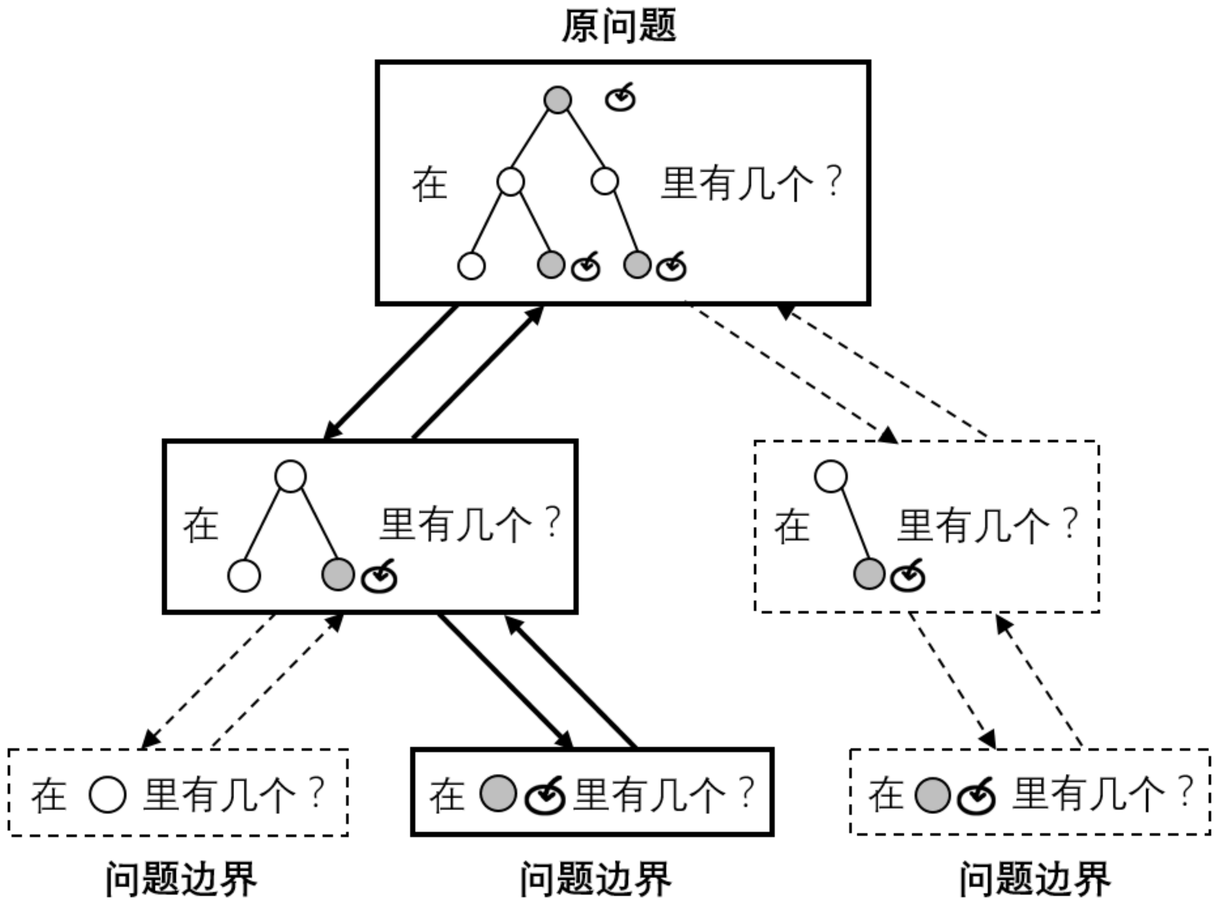
确实，因为 n! 的推导路径我们已经知道了，for 一遍就好了。

但如果是不太容易找到推导路径的问题呢？

比如下面的问题，一棵树，在根节点的时候，我们是不知道下边长什么样的.....



递归 Recursion



递归 Recursion

变量：参数变量 与 全局的共享变量

递归的三个关键：

- 定义需要递归的问题（重叠子问题）——数学归纳法思维
- 确定递归边界
- 保护与还原现场

C++ / Java 代码模板

```
void recursion(int level, int param) {
  // terminator 边界
  if (level > MAX_LEVEL) {
    // process result
    return;
  }

  // process logic in current level
  process(level, param); 每一次的处理

  // drill down
  recur(level + 1, new_param); 递归调用

  // restore the current level status
}   还原
```

Python 代码模板

```
def recursion(level, param1, param2, ...):
    # recursion terminator
    if level > MAX_LEVEL:
      # process result
      return

    # process logic in current level
    process(level, data...)

    # drill down
    self.recursion(level + 1, new_param1, ...)

    # restore the current level status if needed
```

实战

三种基本的递归模板！！！

子集

https://leetcode-cn.com/problems/subsets/

组合

https://leetcode-cn.com/problems/combinations/

全排列

https://leetcode-cn.com/problems/permutations/

全排列 II（Homework）

https://leetcode-cn.com/problems/permutations-ii/

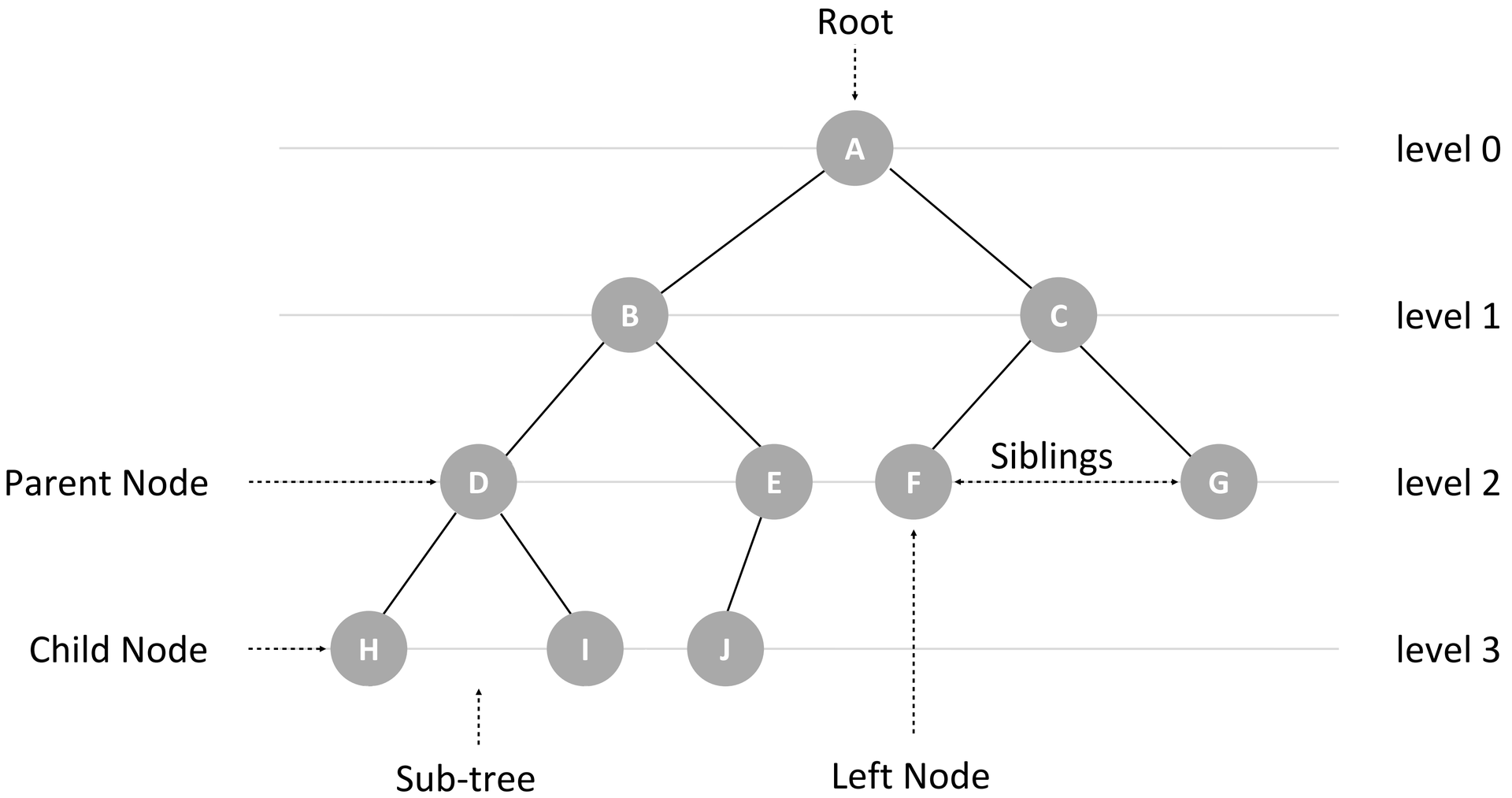
递归基本形式总结

以上三个问题都是递归实现的“暴力搜索”（或者叫枚举、回溯等）

可以总结为以下三种基本形式

递归形式	时间复杂度规模	问题举例
指数型	k^n	子集、大体积背包
排列型	$n!$	全排列、旅行商、N 皇后
组合型	$\frac{n!}{m!\,(n-m)!}$	组合选数

树


Root
A
B
C
D
E
F
G
H
I
J
level 0
level 1
level 2
level 3
Parent Node
Child Node
Siblings
Sub-tree
Left Node

实战

翻转二叉树

https://leetcode-cn.com/problems/invert-binary-tree/description/

验证二叉搜索树

https://leetcode-cn.com/problems/validate-binary-search-tree/

重叠子问题：翻转 or 验证左、右子树

当前层逻辑：翻转 or 验证大小关系

递归边界：叶子节点（无子树）

实战

递归信息统计

二叉树的最大/最小深度

https://leetcode-cn.com/problems/maximum-depth-of-binary-tree/

https://leetcode-cn.com/problems/minimum-depth-of-binary-tree/

思路一（自底向上统计信息，分治思想）

最大深度 = max(左子树最大深度，右子树最大深度) + 1

思路二（自顶向下维护信息）

把“深度”作为一个全局变量——一个跟随结点移动而动态变化的信息

递归一层，变量+1，在叶子处更新答案

这种写法需要注意保护和还原现场

分治算法

分治

分治，即“分而治之”

就是把原问题划分成若干个同类子问题，分别解决后，再把结果合并起来

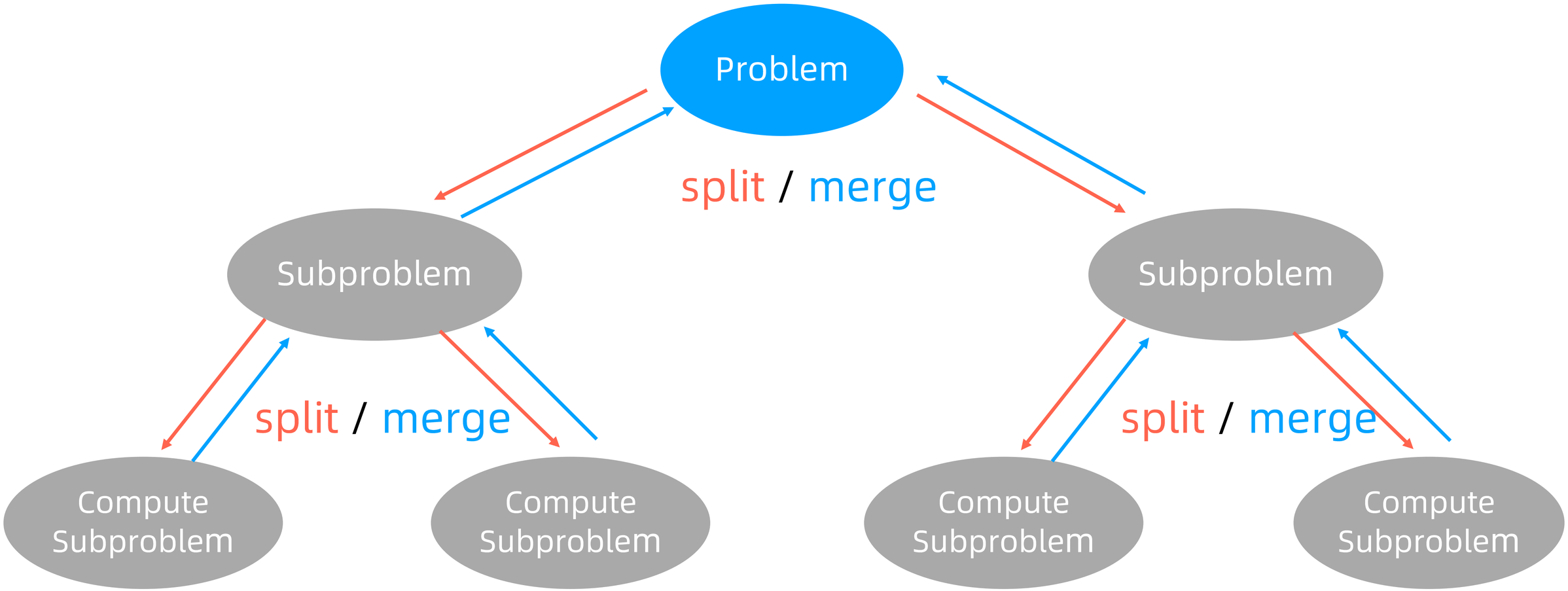
关键点：

原问题和各个子问题都是重复的（同类的）——递归定义

除了向下递归“问题”，还要向上合并“结果”

分治算法一般用递归实现

分治算法的“递归状态树”


Problem
split / merge
Subproblem
Subproblem
split / merge
split / merge
Compute Subproblem
Compute Subproblem
Compute Subproblem
Compute Subproblem

实战

Pow(x, n)

https://leetcode-cn.com/problems/powx-n/

- n 为偶数：Pow(x, n) = Pow(x, n/2) * Pow(x, n/2)
- n 为奇数：Pow(x, n) = Pow(x, n/2) * Pow(x, n/2) * x
- 递归边界：Pow(x, 1) = x
- O(log n)

实战

括号生成

https://leetcode-cn.com/problems/generate-parentheses/

分治划分子问题的标准：不重不漏

（（））（）（）

正确的划分：

- (a)b —— （（））（）（）

错误的划分：

- abc —— （（））（）（） 时间复杂度爆炸
- ab —— （（））（）（） 与 （（））（）（） 重复

Homework

合并K个升序链表

https://leetcode-cn.com/problems/merge-k-sorted-lists/

要求：用分治实现，不要用堆

THANKS