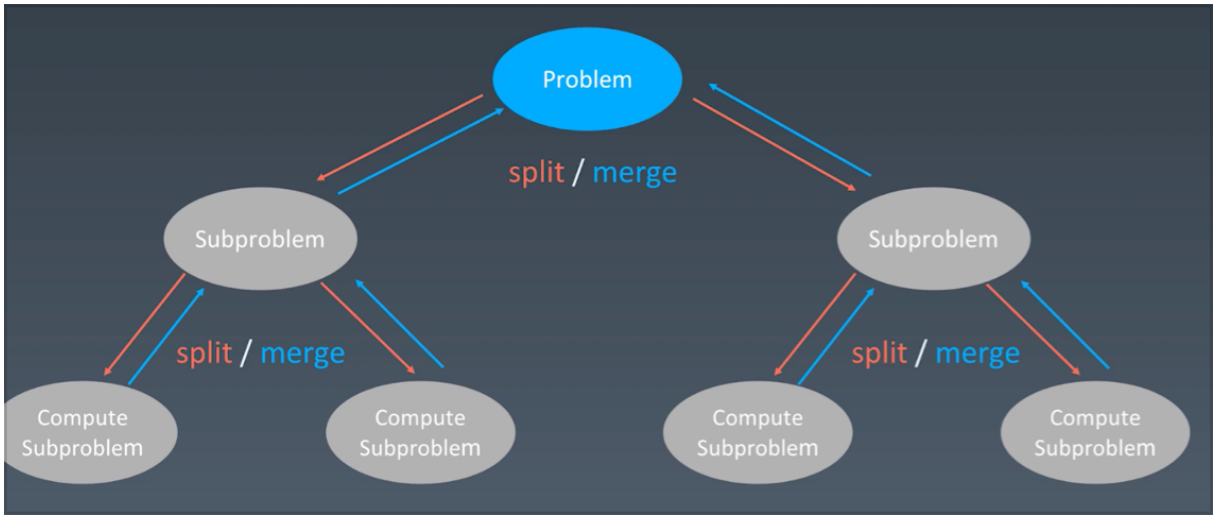
9. 分治 (Divide & Conquer)、回溯

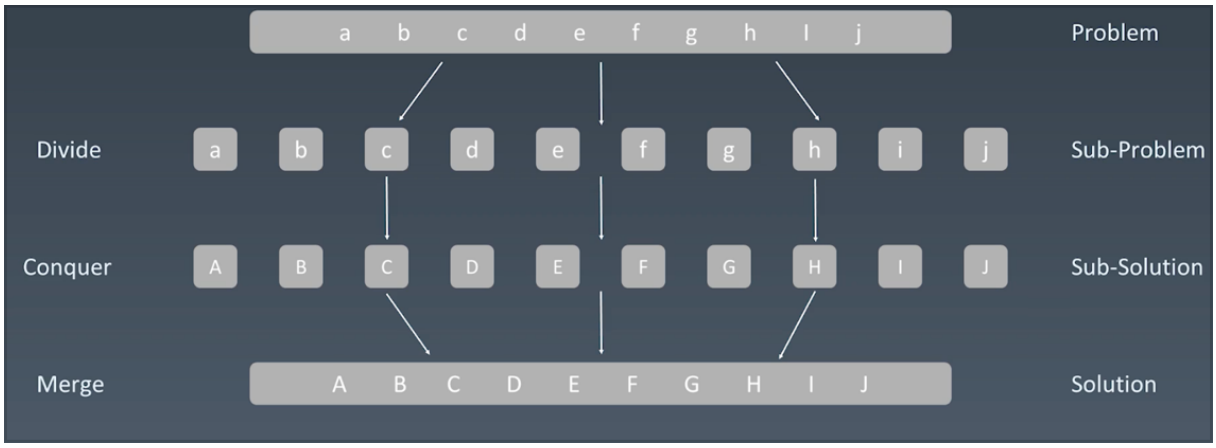
> 分解子问题、找重复性、组合结果

分治 Divide & Conquer

递归状态树



例子



分治代码模版（与递归类似）

注意：当前层只考虑当前层的问题，不下下探太多，人脑不擅长人肉递归

```
def divide_conquer(problem, param1, param2, ...):
```

```
    # recursion terminator 递归结束条件
    if problem is None:
        print_result
        return

    # prepare data 处理当前逻辑
    data = prepare_data(problem)
    subproblems = split_problem(problem, data)

    # conquer subproblems
   # drill down 下探一层
    subresult1 = self.divide_counquer(subproblems[0], p1, ...)
    subresult2 = self.divide_counquer(subproblems[1], p1, ...)
    subresult3 = self.divide_counquer(subproblems[2], p1, ...)

    ...

    # process and generate the final reuslt
    return = process_result(subresult1, subresult2, subresult3, ...)
```

回溯 Backtracking

回溯法采用试错的思想，它尝试分布的去解决一个问题。在分布解决问题的过程中，当它通过尝试发现现有的分布答案不能得到有效的正确的解答的时候，它将取消上一步甚至是上几步的计算，再通过其它的可能的分布解答再次尝试寻找问题答案。

回溯法通常用最简单的递归方法来实现，在反复重复上述的步骤后可能出现两种情况：

- 找到一个可能存在的正确答案；
- 在尝试了所有可能的分布方法后宣告该问题没有答案.

在最坏的情况下，回溯法会导致一次复杂度为指数时间的计算.

#Algorithm/Part II : Theory/Algorithm#