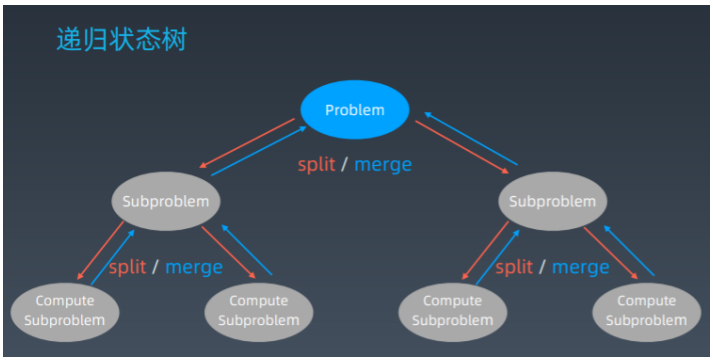
分治

回溯

> 本质上是递归，一种特殊的递归

重复性

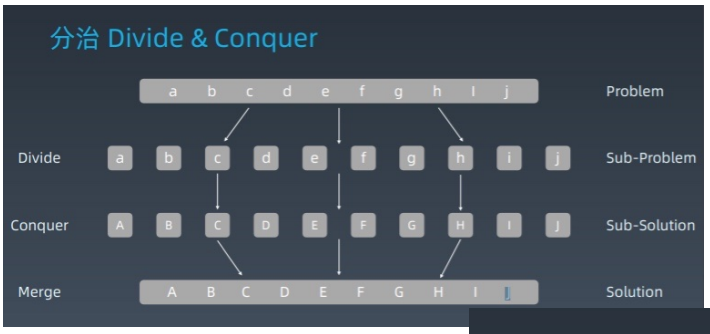
分治 Divide & Conquer



一个问题 → 分解成好几个问题

思路

找重复性 → 分解问题 → 组合每个问题



分治代码模板

```
def divide_conquer(problem, param1, param2, ...):
  # recursion terminator
  if problem is None:
    print_result
    return
  # prepare data (看把这个大问题如何分成子问题)
  data = prepare_data(problem)
  subproblems = split_problem(problem, data)
  # conquer subproblems
  subresult1 = self.divide_conquer(subproblems[0], p1, ...)
  subresult2 = self.divide_conquer(subproblems[1], p1, ...)
  subresult3 = self.divide_conquer(subproblems[2], p1, ...)
  ...
  # process and generate the final result
  result = process_result(subresult1, subresult2, subresult3, …)

  # revert the current level states
```

处理当前层逻辑

下探到下一层

回溯 Backtracking

回溯 Backtracking

回溯法采用试错的思想，它尝试分步的去解决一个问题。在分步解决问题的过程中，当它通过尝试发现现有的分步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步解答再次尝试寻找问题的答案。

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的答案；
- 在尝试了所有可能的分步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。

不断地在每一层去试

(每一层有不同的办法，一个一个去试，
看这个方法行不行)

典型应用：

八皇后，数独