第十讲

Recursion Strategy

薛浩

2023 年 5 月 4 日

www.stickmind.com

今日话题

话题 1：编程基础 初学编程的新手，一般应该熟练使用函数和库处理字符串相关的编程任务。

话题 2：抽象数据类型的使用 在尝试实现抽象数据类型之前，应该先熟练使用这些工具解决问题。

话题 3：递归和算法分析 递归是一种强有力的思想，一旦掌握就可以解决很多看起来非常难的问题。

话题 4：类和内存管理 使用 C++ 实现数据抽象之前，应先学习 C++ 的内存机制。

话题 5：常见数据结构和算法 在熟练使用抽象数据类型解决常见问题之后，学习如何实现它们是一件很自然的事情。

话题 3：递归和算法分析

递归是一种强有力的思想，一旦掌握就可以解决很多看起来非常难的问题。

- 递归过程
- 算法分析
- 递归回溯
- 排序算法

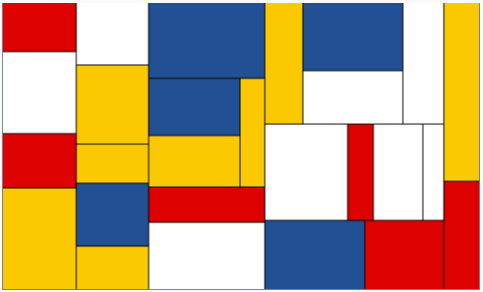


Figure 1: 递归艺术

如何利用递归解决复杂问题?

目录

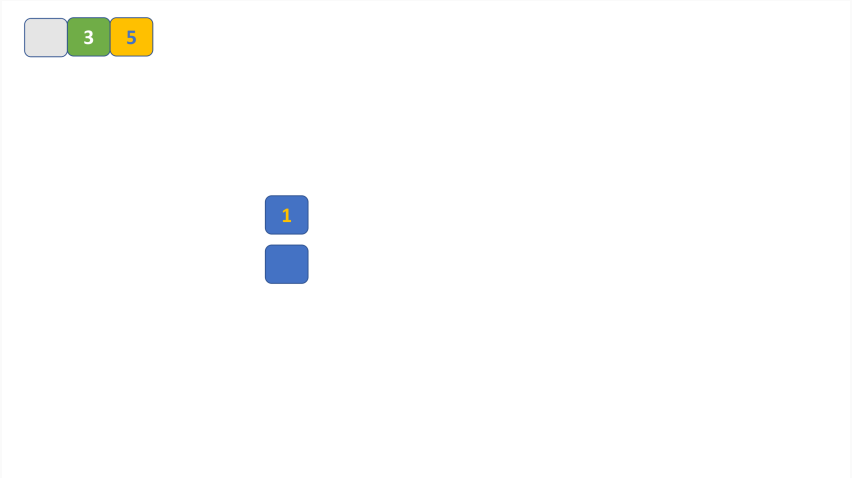
递归子集

练习：ListSubsets

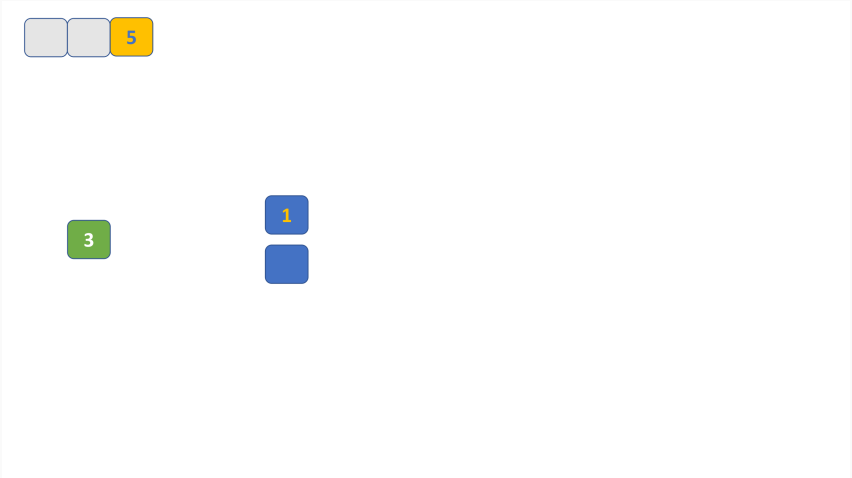
练习：ListSubsets

练习：ListSubsets

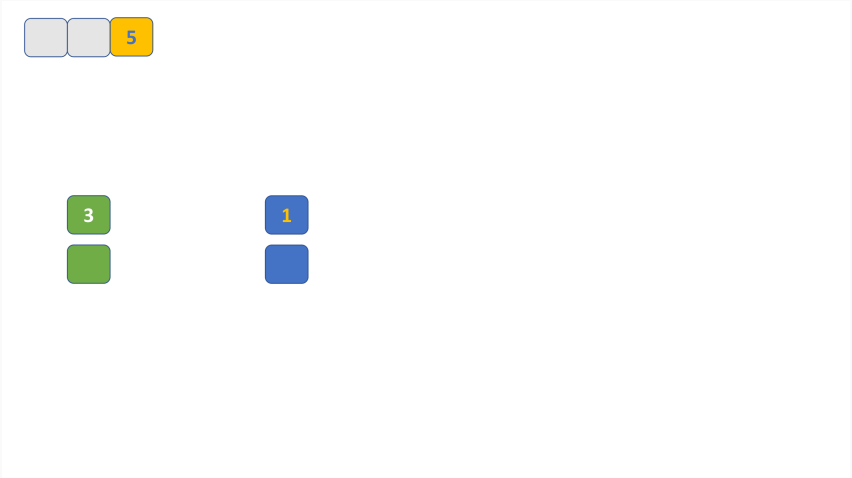
练习：ListSubsets



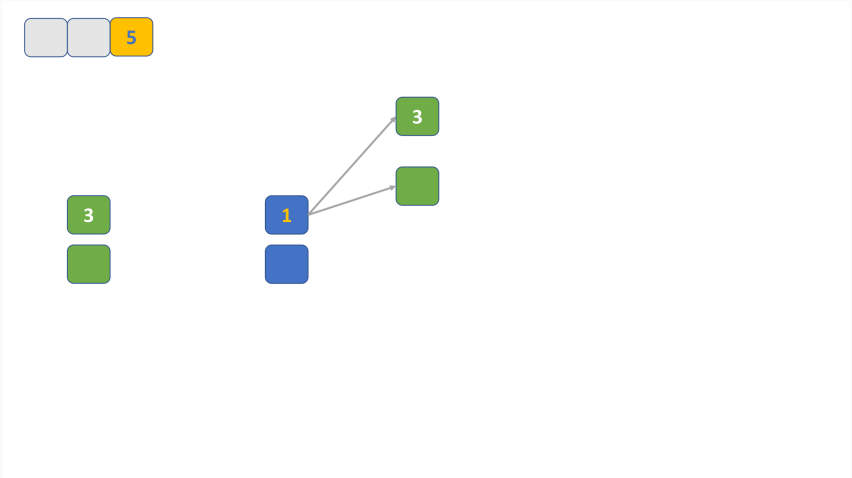
练习：ListSubsets



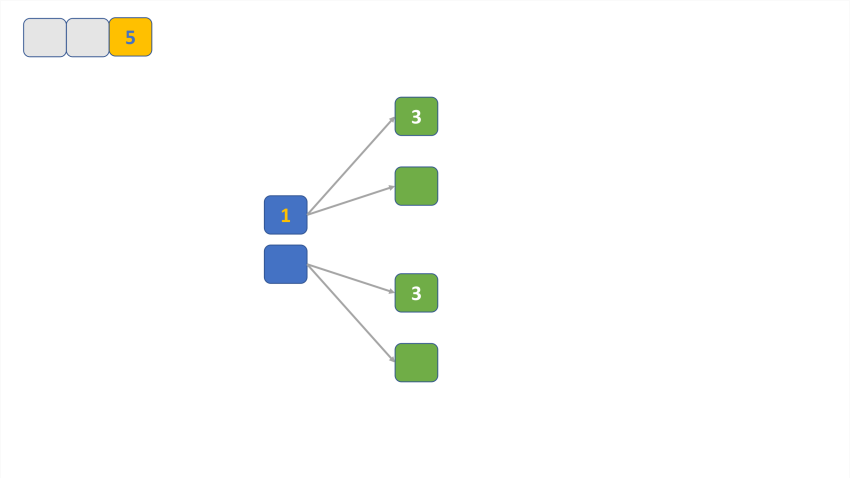
练习：ListSubsets



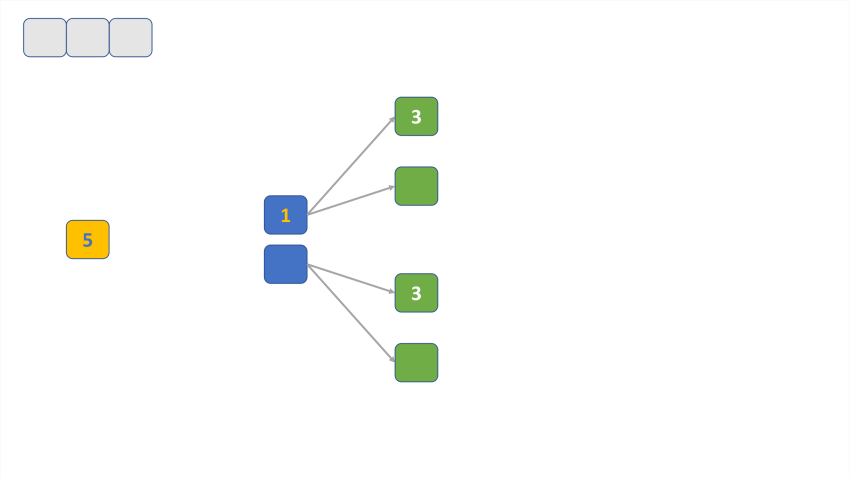
练习：ListSubsets



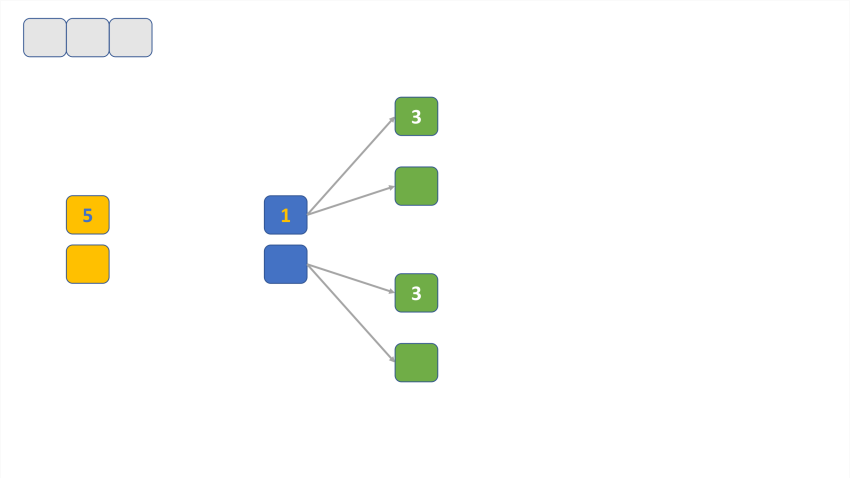
练习：ListSubsets



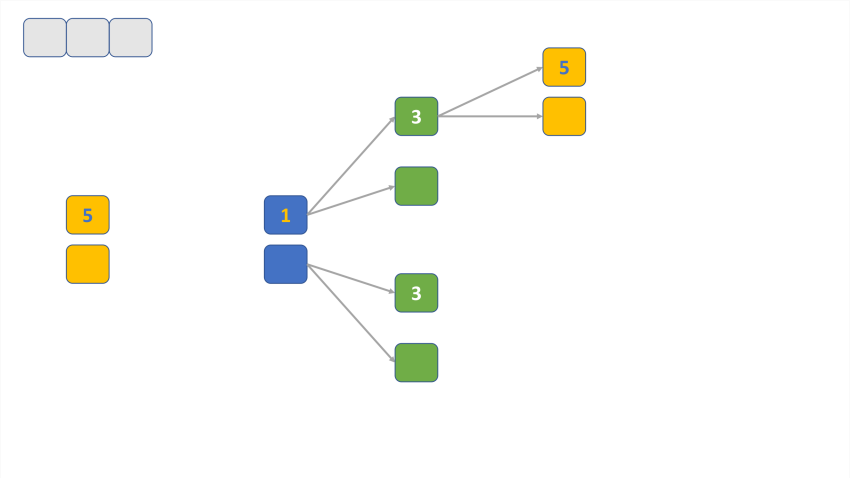
练习：ListSubsets



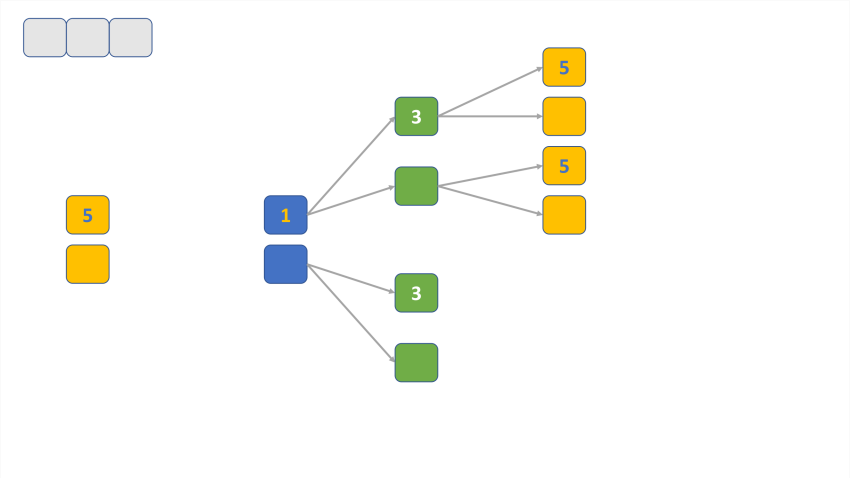
练习：ListSubsets



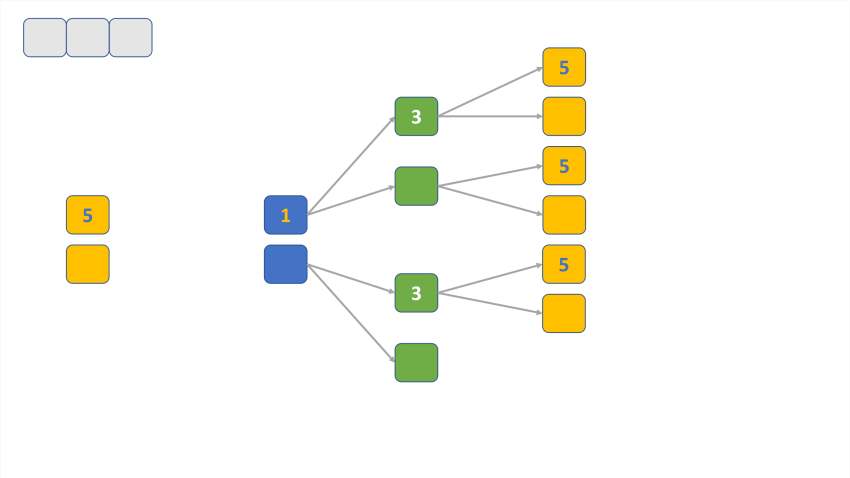
练习：ListSubsets



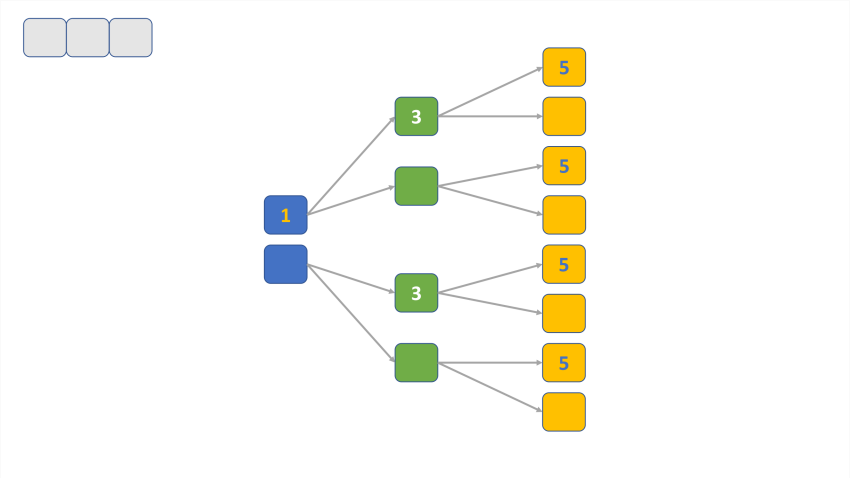
练习：ListSubsets



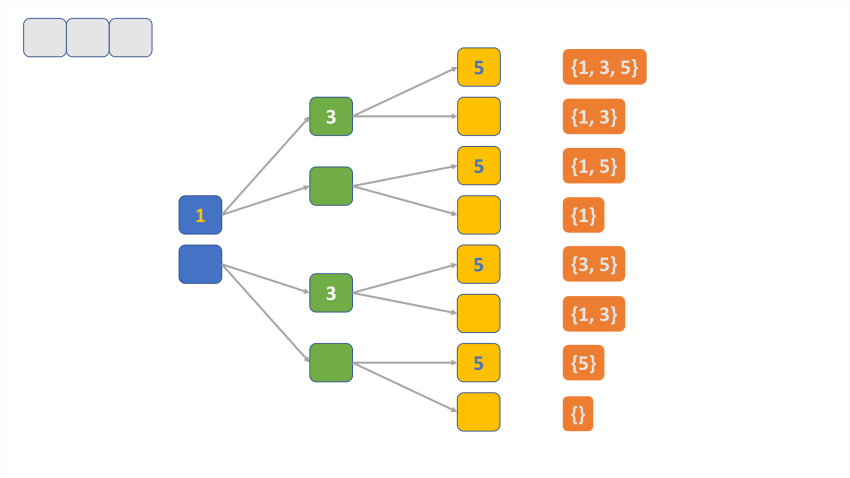
练习：ListSubsets



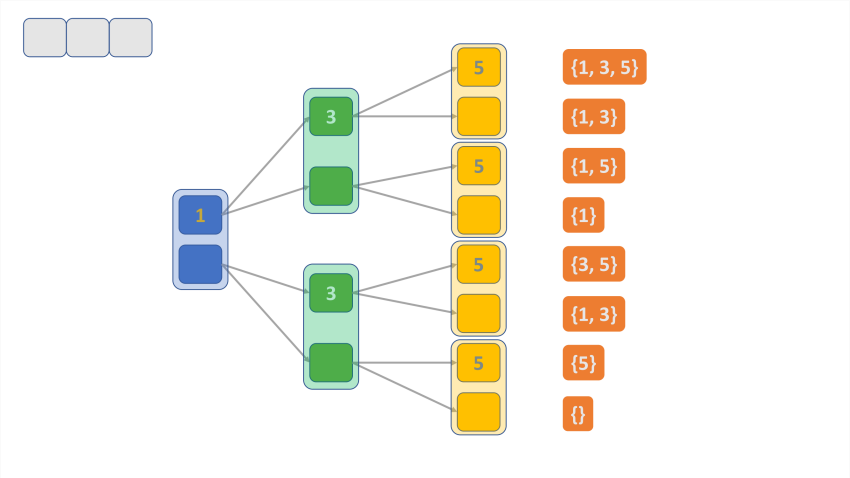
练习：ListSubsets



练习：ListSubsets



练习：ListSubsets



递归范式：包含/排除模式

```
if (问题最简单的形式) {
    无需递归，直接处理并返回结果
} else {
    选择其中一个元素，降低问题规模
    包含该元素，作一条递归调用
    不包含该元素，作另一条递归调用
}
```

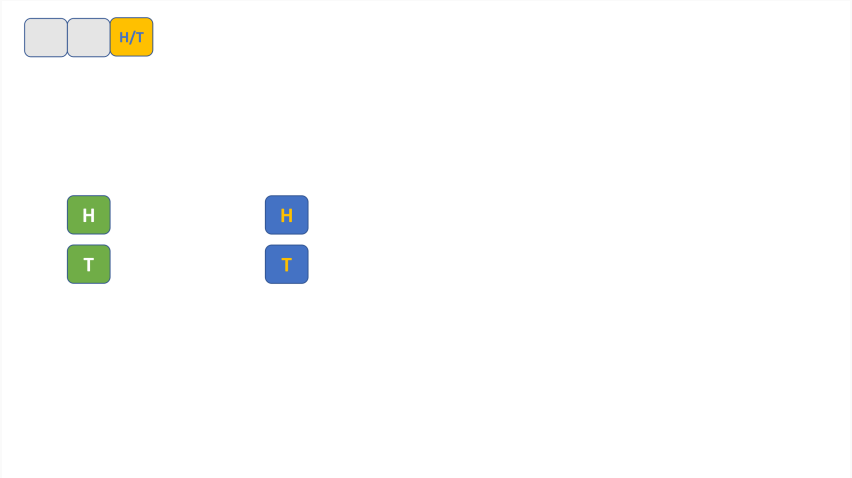
递归排列

练习：ListPermutation

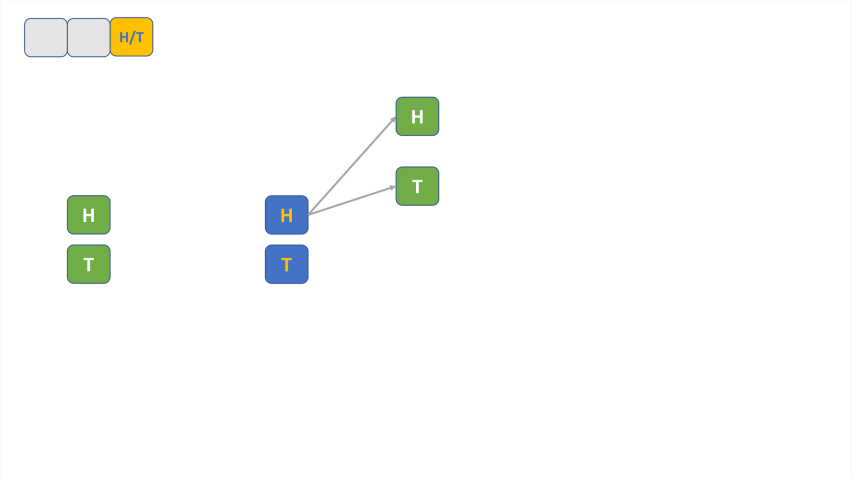
练习：ListPermutation

练习：ListPermutation

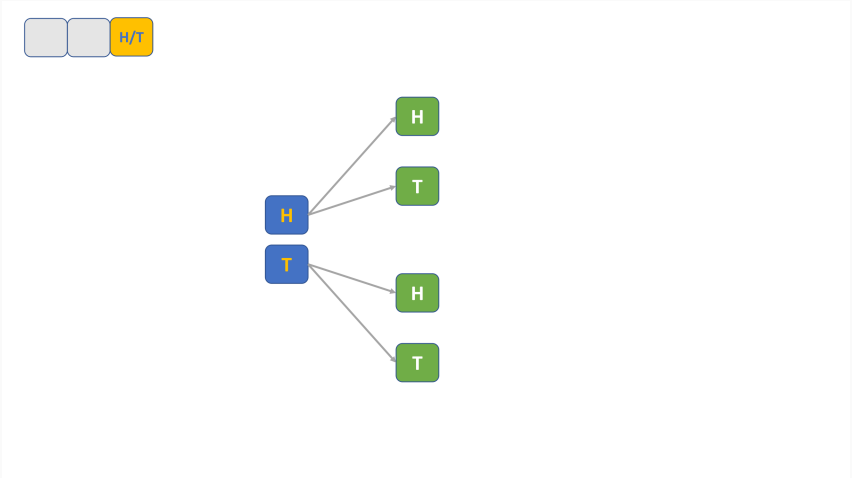
练习：ListPermutation



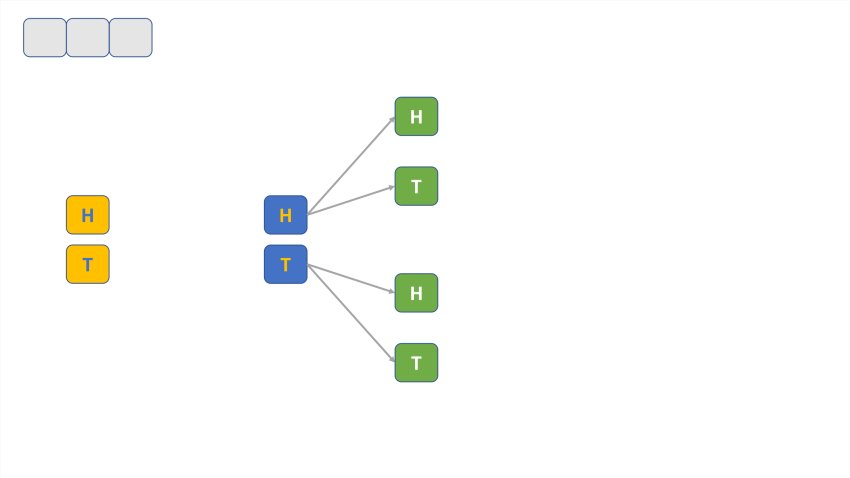
练习：ListPermutation



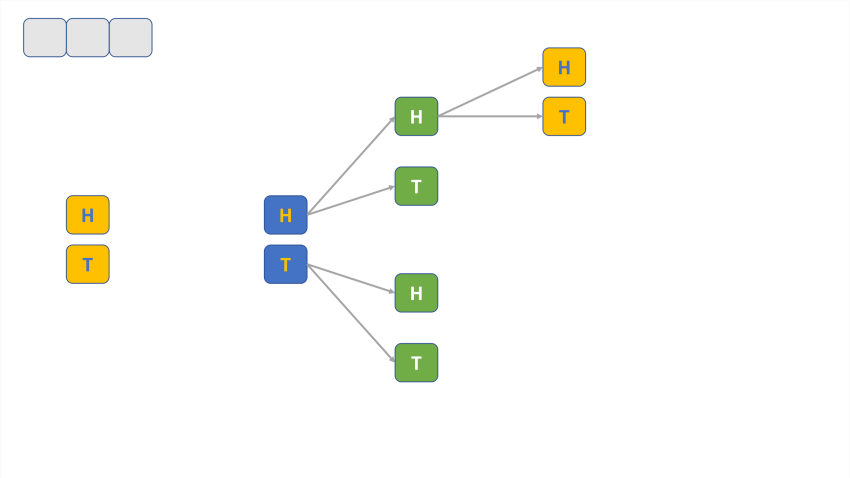
练习：ListPermutation



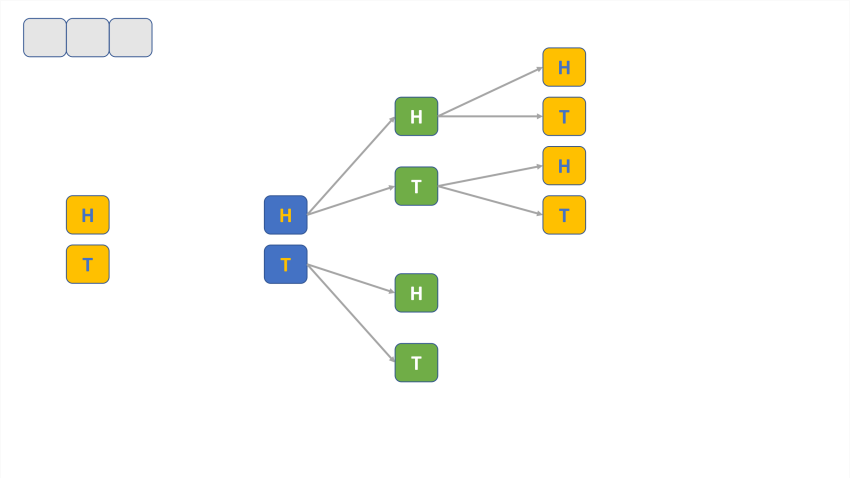
练习：ListPermutation



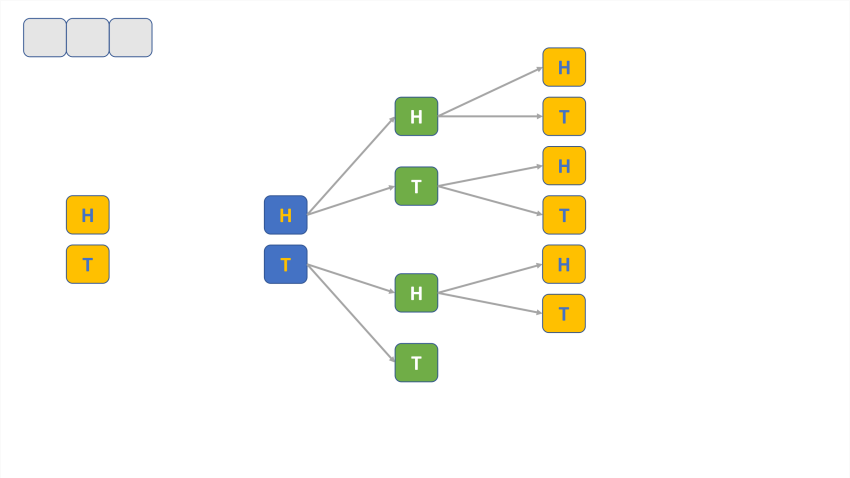
练习：ListPermutation



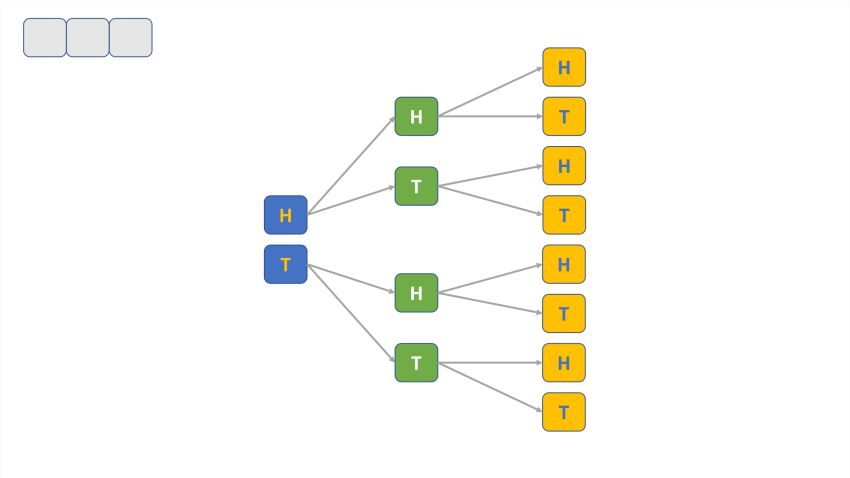
练习：ListPermutation



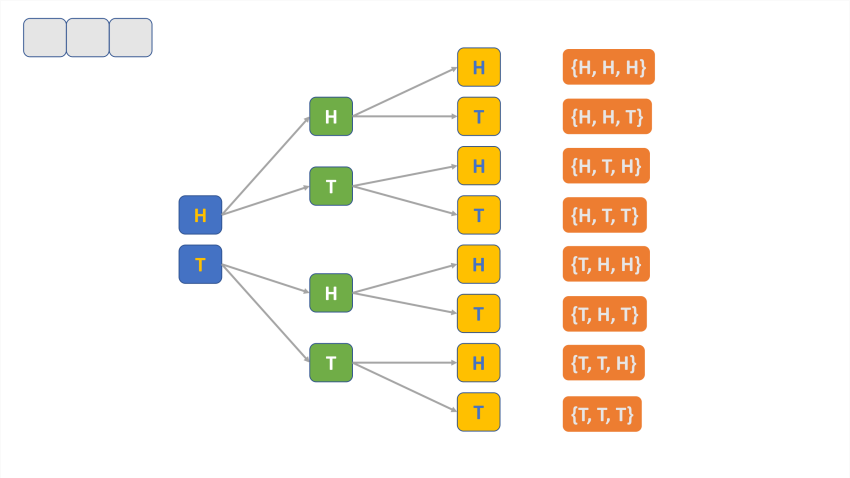
练习：ListPermutation



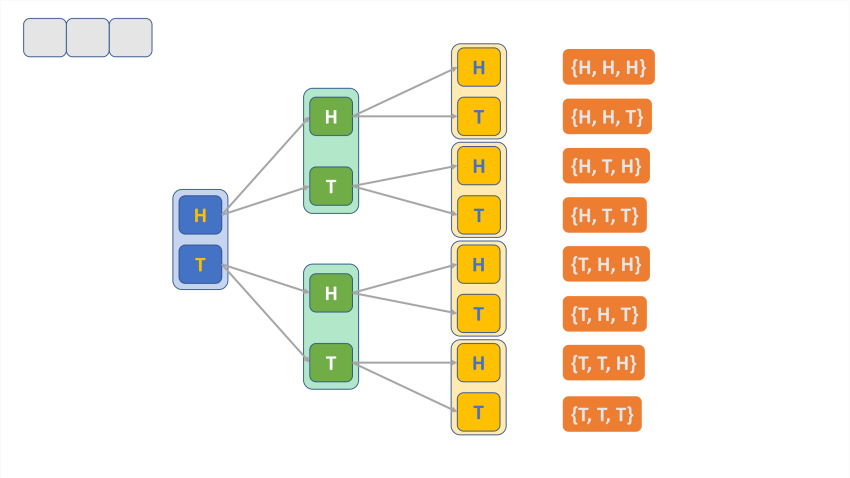
练习：ListPermutation



练习：ListPermutation



练习：ListPermutation



递归范式：选择/探索/切换选项模式

```
if (问题最简单的形式) {
    无需递归，直接处理并返回结果
} else {
    for (每一个可能的选项) {
        选择一个元素
        探索当前元素，作一条递归调用
        切换下一个选项
    }
}
```

BFS vs DFS

练习：Unlock

BFS 宽度优先搜索

```
void generatePasswords(Vector<string>& passwords) {
    Queue<string> todolist;
    todolist.enqueue("");
    while (!todolist.isEmpty()) {
        string current = todolist.dequeue();
        if (current.length() == kMaxLength) {
            passwords.add(current);
        }
        if (current.length() < kMaxLength) {
            for (char ch = '0'; ch <= '9'; ch++)
                todolist.enqueue(current + ch);
        }
    }
}
```

DFS 深度优先搜索

```
Vector<string> generatePasswords(const string& sofar) {
    if (sofar.size() == kMaxLength) {
        return {sofar};
    } else {
        Vector<string> passwords;
        for (char ch = '0'; ch <= '9'; ch++) {
            passwords += generatePasswords(sofar + ch);
        }
        return passwords;
    }
}
```

如何利用递归解决复杂问题?