

week6

1) 牢记递归模版

- 终止条件
- 处理当前层逻辑
- 下探到下一层
- 恢复到下一层

递归代码模版

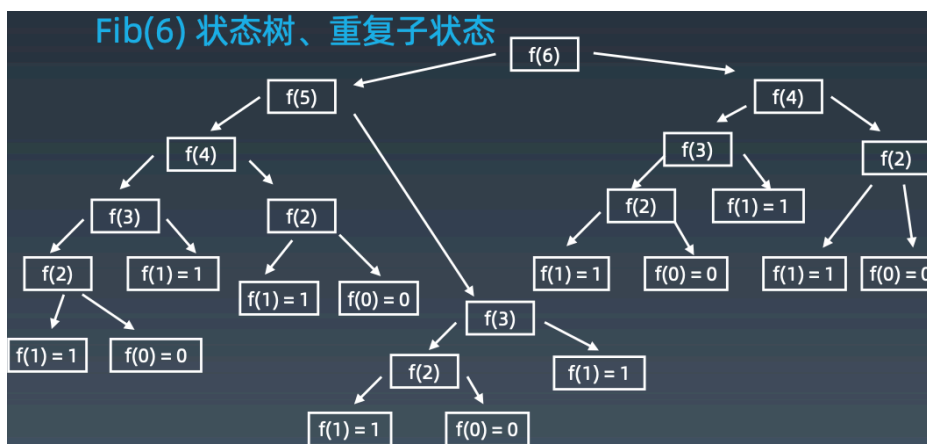
```
public void recur(int level, int param) {  
    // terminator  
    if (level > MAX_LEVEL) {  
        // process result  
        return;  
    }  
  
    // process current logic  
    process(level, param);  
  
    // drill down  
    recur( level: level + 1, newParam);  
  
    // restore current status  
}
```

2) 分治代码模版

分治代码模板

```
def divide_conquer(problem, param1, param2, ...):  
    # recursion terminator  
    if problem is None:  
        print_result  
        return  
  
    # prepare data  
    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
```

- 3) 一般思路：找到最近最简方法，将问题拆解成可重复解决的问题
- 数学归纳法
 - 寻找重复性-> 计算机指令集能做的有限 (if else, for, recursion)
 - eg :



4) dp = 分治 + 最优子结构

- 分治过程中，每一步不需要保存所有状态，只需要保存**最优**的状态
- 动态规划 和 递归或者 **分治** 没有根本上的区别 ([关键看有无最优子结构](#))
 - 共性：找到重复子问题
 - 差异性：最优子结构，中途可以淘汰次优解

- **自底向上** 写循环：

Bottom Up

- $F[n] = F[n-1] + F[n-2]$
- $a[0] = 0, a[1] = 1;$
for (int i = 2; i <= n; ++i) {
 $a[i] = a[i-1] + a[i-2];$
}
- $a[n]$
- 0, 1, 1, 2, 3, 5, 8, 13,

fib (n) = fib (n-1) + fib (n-2)
fib (0) = 0
fib (1) = 1

```
int fib (int n, int[] memo) {
    if (n <= 1) {
        return n;
    }
    if (memo[n] == 0) {
        memo[n] = fib (n - 1) + fib (n - 2);
    }
    return memo[n];
}
```

Memoization
↳ $O(n)$

memo[2] = 1
[3] = 2
[4] = 3
[5] = 5

-dp 关键点：

- 最优子结构 $opt[n] = best_of(opt[n-1], opt[n-2], \dots)$
- 储存中间状态： $opt[i]$
- 递推公式（状态转移方程）
 - Fib: $opt[i] = opt[i-2] + opt[i-1]$
 - 二维路径： $opt[i,j] = opt[i+1][j] + opt[i][j+1]$ （且判断 $a[i,j]$ 是否空地）