# 递推与递归

19 计科朱俊杰，内容部分引自19信安张华睿

对于一个待求解的问题，当它局限在某处边界、某个小范围或者某种特殊情形下时， 答案往往是已知的。如果能够将该解答的应用场景扩大到原问题的状态空间，并且扩展过程的每个步骤具有相似性，就可以考虑使用递推或者递归求解。递推与递归是可以相互转化的，递归在某些问题上更容易理解。

> 以已知的“问题边界”为起点向“原问题”正向推导的方式就是递推。
>
> 以“原问题”为起点尝试寻找把状态空间缩小到已知的“问题边界”的路线，再通过该 路线反向回溯的遍历方法就是递归。

--《李煜东：算法竞赛进阶指南》

## 递推

## 例1

### 求前缀和

给定n，求$a_1+a_2+a_3+...+a_n$。

递归边界：$sum_1=a_1$

递推公式：$sum_n=sum_{n-1}+a_n$

```
int a[N],sum[N];
for(int i=1;i<=n;i++){
    sum[i]=sum[i-1]+a[i];
}
```

## 例2

### 卡特兰数

给定一个凸多边形，有n条边$(n >= 4)$，用n-3条不相交的对角线把它分成n-2个三角形，求不同的方法数目。

假设每一个顶点都有一个编号，分别为$V_1, V_2 \ldots V_n$,对于任意选择的$V_1, V_n$两个节点，然后枚举顶点$V_k$，$V_k$可以去从$V_2$到$V_{n-1}$的n-2个顶点。会构成一个以{1,2，…，k}为顶点的多边形和一个以{k，k+1,…，n}为顶点的多边形。

递推边界：$f_2 = f_3 = 1$

递推公式：$f_n = f_2 * f_{n-1} + f_3 * f_{n-2} + \ldots + f_{n-1} * f_2$

## 例3

**硬币下棋游戏**

棋盘上标有第0站，第1站...第100站，一开始棋子在第0站，棋手每次投一次硬币，若硬币正面向上，则往前跳两站；否则，往前跳一站...直到棋子跳到第99站（胜利大本营），第100站（失败大本营）时，游戏结束。如果硬币出现正反面的概率均为0.5，分别求出棋子到达胜利大本营和失败大本营的概率。

假设$f_i$表示从第i站开始到达第100站的概率。有0.5的概率从第i站可以到达第i+1站或者第i+2站。得到$f_i = 0.5 * f_{i+1} + 0.5 * f_{i+2}$。

结束位置即为递推边界，$f_{100} = 1.0, f_{99} = 0.0$。

## 例4

**传球游戏**

现有四个人做传球游戏，要求接球后马上传给别人。由甲先传，并作为第一次传球。求经过10次传球，球仍回到发球人甲手中的传球方式的种数。

这道题我们只要考虑甲，所以我们定义两个状态。

(1)当前球在甲手上，经过i次传球之后球仍在甲上，此状况记为f，其传球方式的种数为$f_i$；

(2)当前球不在甲手上，经过i次传球之后球在甲手上，此状态记为g，其传球方式的种数为$g_i$。

对于状态 (1)，$f_i$,球在甲的手上，所以球只能传给其他三人，状态就转变为$g_{i-1}$,所以$f_i = 3 * g_{i-1}$。

对于状态(2),$g_i$，此时球可以传给甲，状态转移为$f_{i-1}$，或者其他两个人$g_{i-1}$，所以
$g_i = f_{i-1} + 2 * g_{i-1}$。

递推边界，$f_1 = 0, g_1 = 1$,球在甲手上他不可能一次传回给自己，求在其他人手上一次传回甲只有一种方式。

这种递推也有人称它为间接递推

```
int g[N],f[N];
f[1]=0,g[1]=0;
for(int i=2;i<=n;i++){
    f[i]=3*g[i-1];
    g[i]=f[i-1]+2*g[i-1];
}
```

## 空间复杂度

在讲解递归之前，我给大家讲解一下空间复杂度的概念。对于一个算法的好坏的评价的标准是根据这个算法需要的空间和时间来决定的。之前就有师哥给你们讲过时间复杂度，现在我来给你们讲一下空间复杂度。

算法的存储量包括：

1.程序本身所占空间。

2.输入数据所占空间。

3.辅助变量所占空间。

一个int型变量在内存中的大小为4B（字节）

```
int a[N];//所需要的空间大小为4N B
int b[N][M];//所需的空间大小为4NM B
```

一般的题目数组在全局变量中最多可以开1e8。

# 递归

在计算机科学中，如果一个函数的实现中，出现对函数自身的调用语句，则该函数称为递归函数。
递推算法可以用递归函数来实现。一般来说循环递推算法比递归函数要快，但递归函数的可读性更棒。
以已知的“问题边界”为起点向“原问题”正向推导的方式就是递推。 以“原问题”为起点尝试寻找把状态空间缩小到已知的“问题边界”的路线，再通过该路线 反向回溯的遍历方法就是递归。

递归要求“原问题”与“问题边界”之间的每个变换步骤具有相似性。这样我们才能够设计 一段程序实现这个步骤，将其重复作用于问题之中。换句话说，程序在每个步骤上应该 面对相同种类的问题，并且能够使用“求解原问题的程序”进行求解。这些问题都是原问 题的一个子问题，可能仅在规模或者某些限制条件上有所区别，并且能够使用“求解问题 的程序”进行求解。 --《李煜东：算法竞赛进阶指南》

我们就可以设计程序在每个变换步骤种执行三个操作： 1. 缩小问题状态空间的规模。 2. 尝试求解规模缩小以后的问题，结果可能是成功，也可能是失败。 3. 如果成功，即找到了规模缩小后的问题的答案，那么将答案扩展到当前问题。如果失败，那么重新回到当前问题，程序可能会继续寻找当前问题的其他变换路线，直至最终确定当前问题无法求解。

## 例1

### 求阶乘

递归边界：$f_1 = f_0 = 0$

递归公式：$f_n = f_{n-1} * n$

```
int f(int n){
    if(n==1||n==0) return 1;
    return n*f(n-1);
}
```

## 例2

### 爬楼梯

假设你正在爬楼梯。需要 n 阶你才能到达楼顶。每次你可以爬 1 或 2 个台阶。你有多少种不同的方法可以爬到楼顶呢?

递归边界:$f_1 = 1, f_2 = 2$

递归公式:$f_n = f_{n-1} + f_{n-2}$

```
int f(int n){
    if(n==1) return 1;
    if(n==2) return 2;
    return f(n-1)+f(n-2);
}
```

## 例3

### 汉诺塔

汉诺塔问题是一个经典的问题。汉诺塔（Hanoi Tower），又称河内塔，源于印度一个古老传说。大梵天创造世界的时候做了三根金刚石柱子，在一根柱子上从下往上按照大小顺序摞着64片黄金圆盘。大梵天命令婆罗门把圆盘从下面开始按大小顺序重新摆放在另一根柱子上。并且规定，任何时候，在小圆盘上都不能放大圆盘，且在三根柱子之间一次只能移动一个圆盘。问应该如何操作？
假设三根柱子分别为A，B，C，起初我们一共有$n$个圆盘放在A柱上，我们要把n个圆盘移到C上，首先就要把A最下面的大圆盘放在C上面，所以需要把先把n-1个柱子放在B上，再把最大的圆盘移到C上,之后问题就等价于将n-1个圆盘从B柱上移到C柱上....

递归边界：$f_1 = 1$

递归公式：$f_n = 2 * f_{n-1} + 1$

```
int f(int n){
    if(n==1) return 1;
    return 2*f(n-1)+1;
}
```

## 递归的空间问题

递归与递推相比往往需要更多的空间，递归的函数的空间复杂度。

```
int fib(int n){
    if(n==1||n==2) return 1;
    return f(n-1)+f(n-2);
}
```

它需要的递归次数大约是$2^n$,所以它的空间复杂度$O(2^n)$。

## 递推的时间问题

之前的**zrz楼梯**那道题有部分同学使用递归超时，我们来回忆一下那道题，上楼梯一次可以跨三步，两步，一步。

递归边界：$f_1 = 1, f_2 = 2, f_3 = 4$

递归公式：$f_n = f_{n-1} + f_{n-2} + f_{n-3}$

```
//超时写法
long long f(int n){
    if(n==1) return 1;
    if(n==2) return 2;
    if(n==3) return 4;
    return f(n-1)+f(n-2)+f(n-3);
}
```

为什么会超时？存在**重复计数**！！！

这个代码如果你需要计算$f_{50}$，你需要计算$f_{49}+f_{48}+f_{47}$,它需要一直展开，每次得到一个新的式子，从它的最左边再次展开，最终它需要计算的次数会非常多，时间复杂度接近于$O(3^{47})$,肯定会超时，而且你会发现它会计算相同的项，计算$f_{49}$的过程中你会计算$f_{48}$,所以我们需要把之前计算的结果保留，才不会超时。

```
long long ans[N]={0,1,2,4};//存储信息的数组
long long f(int n){
    if(f[n]!=0) return f[n];//如果之前计算过这个值，那么就直接返回
    return f[n]=f(n-1)+f(n-2)+f(n-3);//之前没有计算过这个值，返回计算出的值，并将值保留下来。
}
```