第5章 递归

5.1 什么是递归

5.2 递归算法的设计

5.1 什么是递归

5.1.1 递归的定义

在定义一个过程或函数时，出现直接或者间接调用自己的成分，称之为递归。

- 若直接调用自己，称之为直接递归。
- 若间接调用自己，称之为间接递归。

直接递归函数示例：求*n*!（*n*为正整数）

```
int fun(int n)
{    if (n==1)                          //语句1
          return 1;                     //语句2
     else                               //语句3
          return  n*fun(n-1);           //语句4
}
```

间接递归示例：

```
void f1(…)
{
    …
    f2( …);
    …
}
```

```
void f2(…)
{
    …
    f1( …);
    …
}
```

总可以转换为直接递归函数

如果一个递归函数中递归调用语句是最后一条执行语句，则称这种递归调用为尾递归。

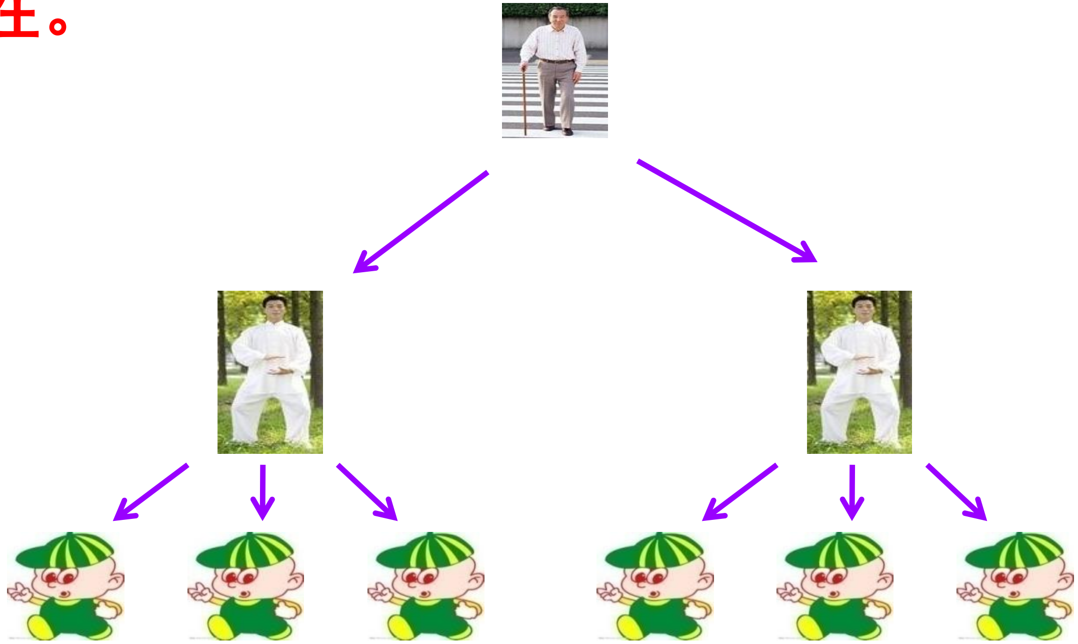
```
int fun(int n)
{     if (n==1)                              //语句1
            return 1;                        //语句2
      else                                   //语句3
            return n*fun(n-1);               //语句4
}
```

直接递归函数、尾递归

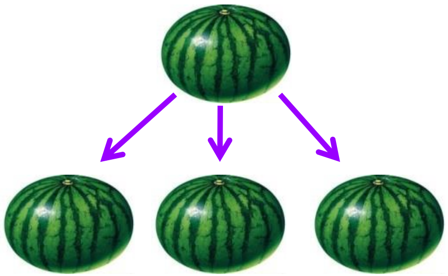
- 尾递归算法：可以用循环语句转换为等价的非递归算法
- 其他递归算法：可以通过栈来转换为等价的非递归算法

递归：无处不在。

实例1：家谱



实例2：种瓜得瓜



第一年种瓜

第n年种瓜

5.1.2 何时使用递归

在以下三种情况下，常常要用到递归的方法。

1、定义是递归的

有许多数学公式、数列等的定义是递归的。

例如，求n!和Fibonacci数列等。这些问题的求解过程可以将其递归定义直接转化为对应的递归算法。

思考题：

请你给出正整数的定义。

- 1是正整数。
- 如果n是正整数，则$n+1$也是正整数。

2、数据结构是递归的

有些数据结构是递归的。例如，第2章中介绍过的单链表就是一种递归数据结构，其节点类型定义如下：

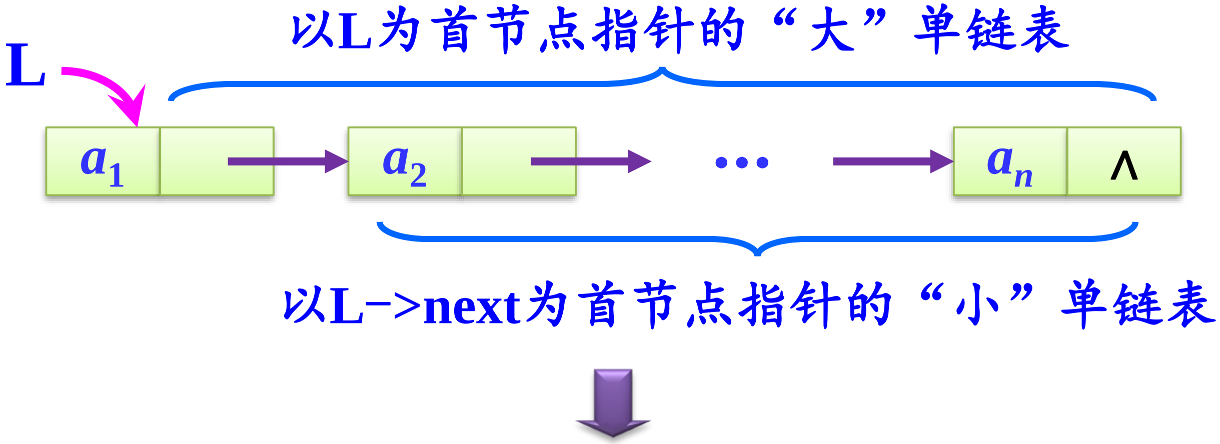
```
typedef  struct LNode
{      ElemType data;
       struct LNode *next;
}  LinkList;
```

→ 指向同类型节点的指针

↓

递归数据结构

不带头节点单链表示意图



体现出这种单链表的递归性。

思考：如果带有头节点又会怎样呢？？？

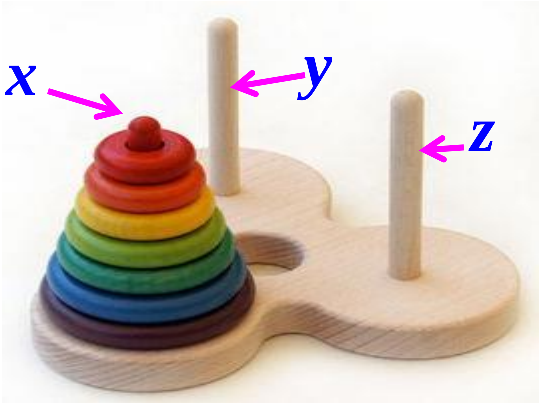
3、问题的求解方法是递归的

Hanoi问题：*X*、*Y*和*Z*的塔座，在塔座*X*上有*n*个直径各不相同，从小到大依次编号为1～*n*的盘片。要求将*X*塔座上的*n*个盘片移到塔座*Z*上。

移动规则：

- 每次只能移动一个盘片；
- 盘片可以插在*X*、*Y*和*Z*中任一塔座上；
- 任何时候都不能将一个较大的盘片放在较小的盘片上方。

设Hanoi(n， x， y， z)表示将n个盘片从x通过y移动到z上。



Hanoi(n， x， y， z)

Hanoi(n−1， x， z， y);
move(n， x， z):将第n个圆盘从x移到z;
Hanoi(n−1， y， x， z)

“大问题”转化为若干个“小问题”求解

5.1.3 递归模型

递归模型是递归算法的抽象，它反映一个递归问题的递归结构。

例如，求n! 递归算法对应的递归模型如下：

fun(1)=1		❶	递归出口
fun(n)=n*fun(n-1)	n>1	❷	递归体

一般地，一个递归模型是由递归出口和递归体两部分组成。

- 递归出口确定递归到何时结束。
- 递归体确定递归求解时的递推关系。

递归出口的一般格式如下：

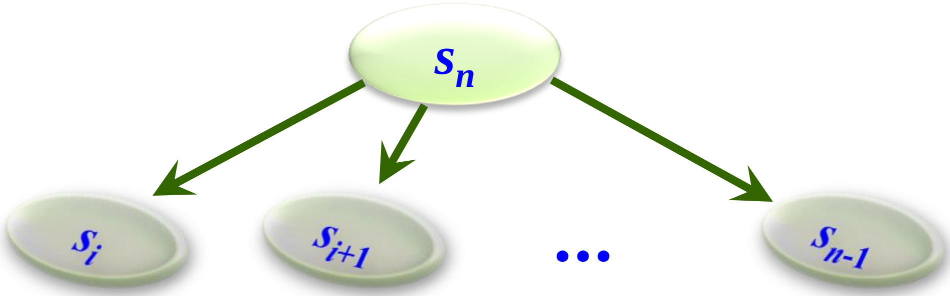
$$f(s_1) = m_1$$

这里的s_1与m_1均为常量，有些递归问题可能有几个递归出口。

递归体的一般格式如下：

$$f(s_n)=g(f(s_i),\ f(s_{i+1}),\ \ldots,\ f(s_{n-1}),\ \underbrace{c_j,\ c_{j+1},\ \ldots,\ c_m}_{\text{常量}})$$

g是一个非递归函数



大问题求解

$\Big\downarrow$ 转化

若干个相似子问题求解

递归思路

把一个不能或不好直接求解的“大问题”转化成一个或几个“小问题”来解决；

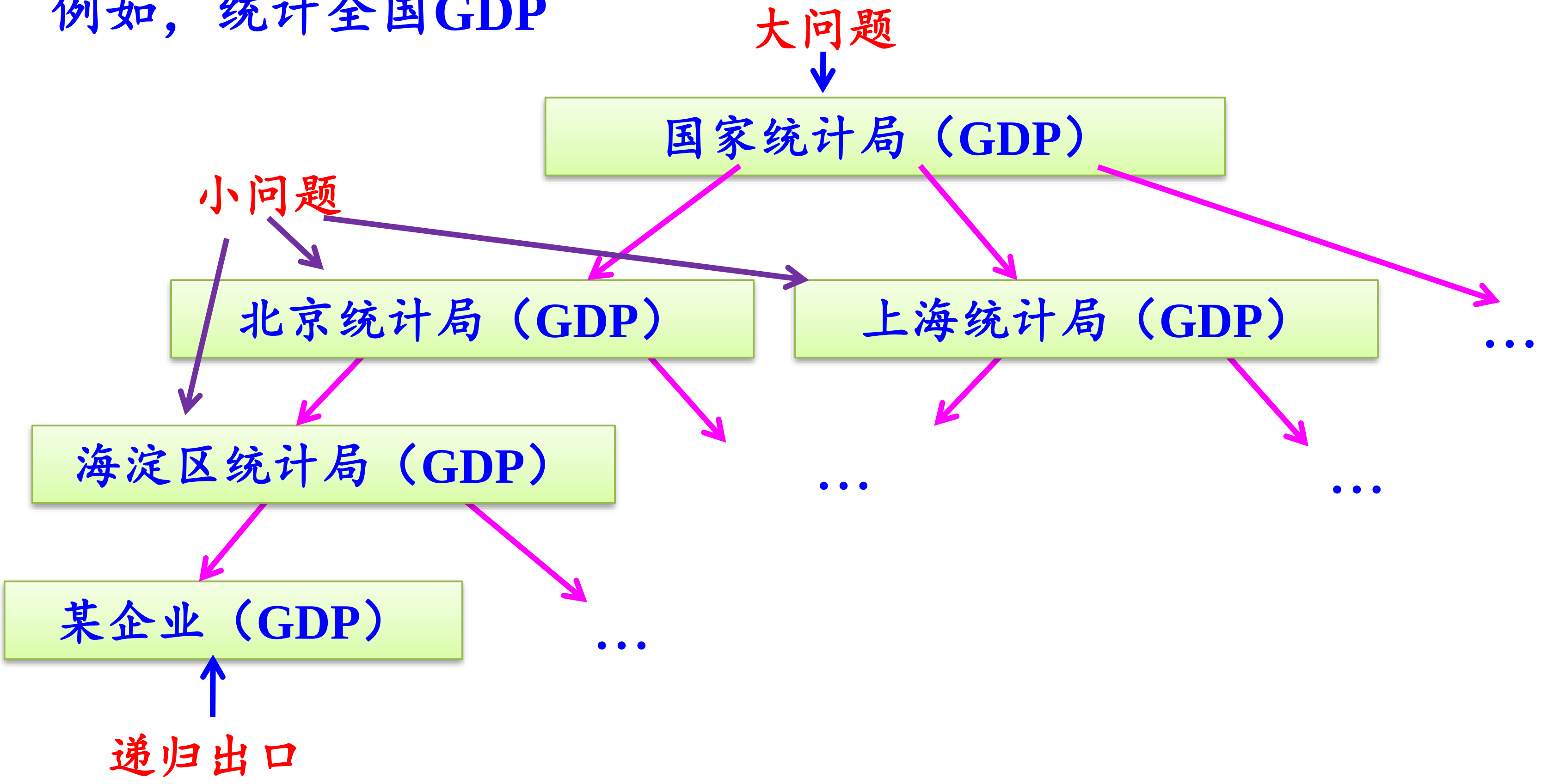
再把这些“小问题”进一步分解成更小的“小问题”来解决。

直到

每个“小问题”都可以直接解决（此时分解到递归出口）

但递归分解不是随意的分解，递归分解要保证“大问题”与“小问题”相似，即求解过程与环境都相似。

例如，统计全国GDP



为了讨论方便，简化上述递归模型为：

$$f(s_1)=m_1$$

$$f(s_n)=g(f(s_{n-1}),\ c_{n-1})$$

求$f(s_n)$的分解过程如下：

$$f(s_n)$$
$$\downarrow$$
$$f(s_{n-1})$$
$$\downarrow$$
$$\cdots$$
$$\downarrow$$
$$f(s_2)$$
$$\downarrow$$
$$f(s_1)$$

遇到递归出口发生"质变"，即原递归问题便转化成可以直接求解的问题。求值过程：

$$f(s_1)=m_1$$

$$\downarrow$$

$$f(s_2)=g(f(s_1),c_1)$$

$$\downarrow$$

$$f(s_3)=g(f(s_2),c_2)$$

$$\downarrow$$

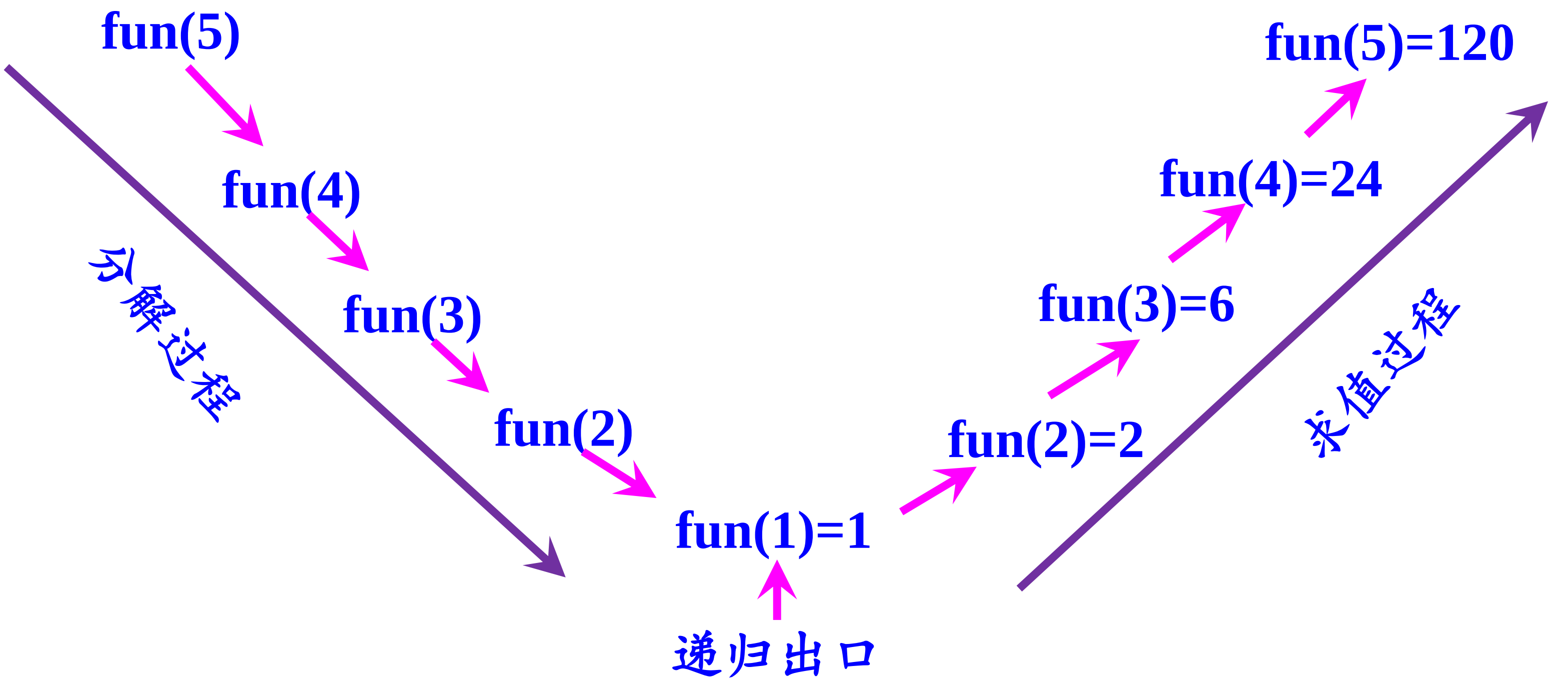
$$\cdots$$

$$\downarrow$$

$$f(s_n)=g(f(s_{n-1}),c_{n-1})$$

这样$f(s_n)$便计算出来了，因此递归的执行过程由分解和求值两部分构成。

求解fun(5)即5!的过程如下：

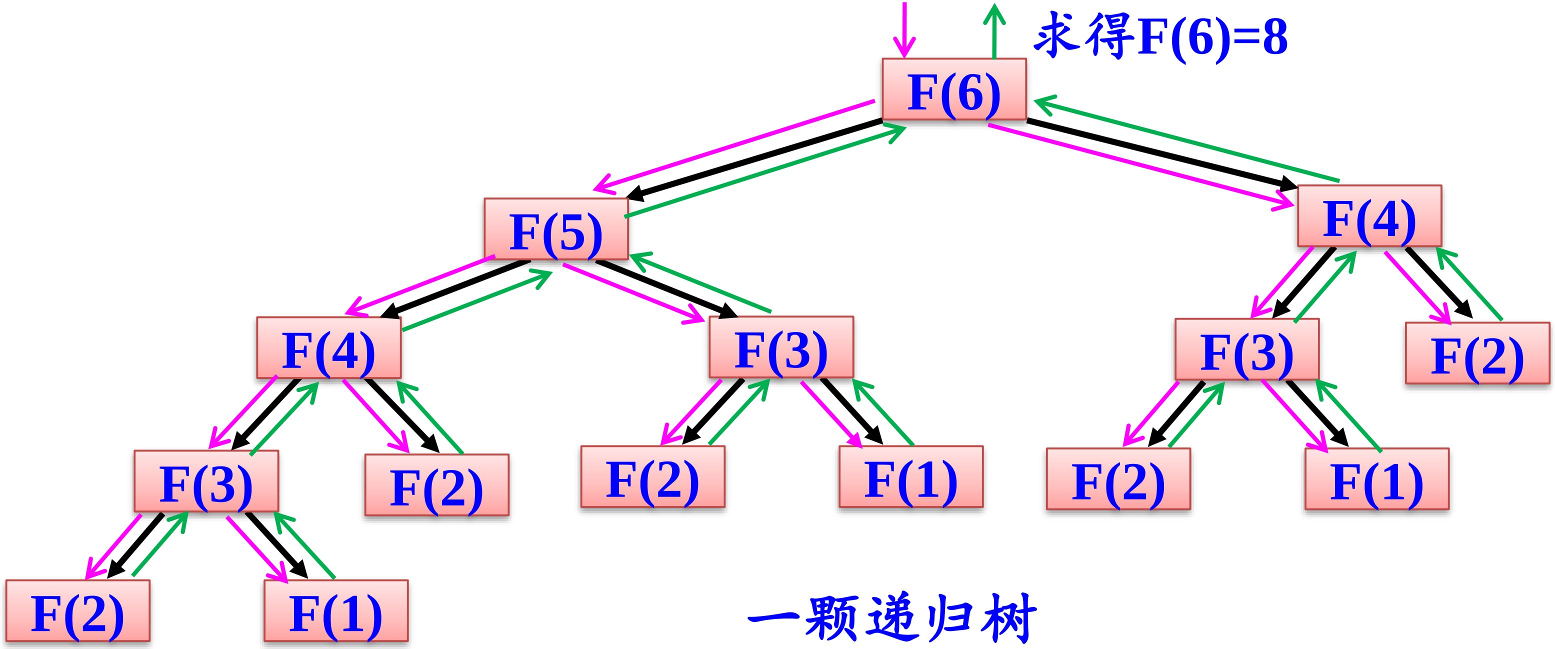


对于复杂的递归问题，在求解时需要进行多次分解和求值。

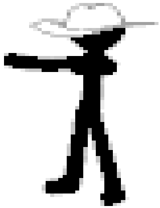
例如：

$F(1)=1$，$F(2)=1$

$F(n)=F(n-1)+F(n-2)\quad n>2$　　　求$F(6)$ = ?



一颗递归树



——本讲完——