《算法设计与分析》

第一章 算法引论

马丙鹏

2023年09月11日

中国科学院大学
University of Chinese Academy of Sciences

第一章 算法引论

- 1.1 算法的定义及特性
- 1.2 复杂性分析初步
- 1.3 递归

1.3 递归

- **1. 定义**
  - 程序调用自身的编程技巧称为递归(**recursion**)。包括直接或间接调用自身。
  - 它通常把一个大型复杂的问题层层转化为一个与原问题相似的规模较小的问题来求解，递归策略只需少量的程序就可描述出解题过程所需要的多次重复计算，大大地减少了程序的代码量。
  - 递归是一种强有力的设计方法
    - 与数学模型一致, 表述简单清晰, 容易实现, 容易证明正确性。
    - 效率较低。

1.3 递归

■ 2. 子程序调用的一般形式

主程序

子程序 A

Call A

1:

主程序

子程序 A

Call A

1:

Call A

2:

主程序

子程序 A

子程序 B

Call B

2:

Call A

1:

主程序

子程序 A

子程序 B

Call A

2:

Call B

1:

递归求阶乘的算法

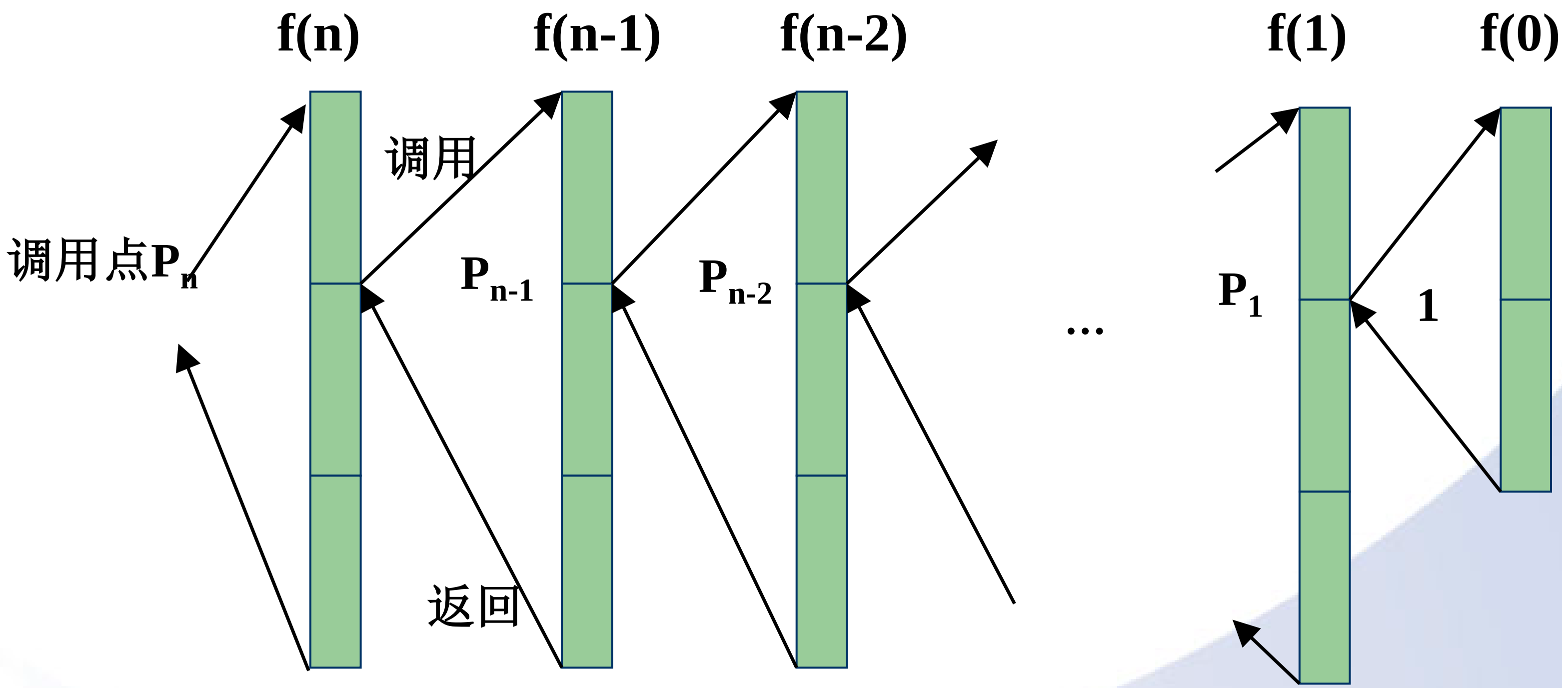
```
procedure f(integer n)
integer y;
    if (n=0) return 1
    y ← f(n–1);
    return(n*y);
end f
```

主程序：

```
integer fn;
fn ← f(4);
```

为了保证递归调用的正确性，需要保存调用点的现场(返回地址、局部变量、被调用函数的参数等)，以便正确地返回，并且按先进后出的原则来管理这些信息。

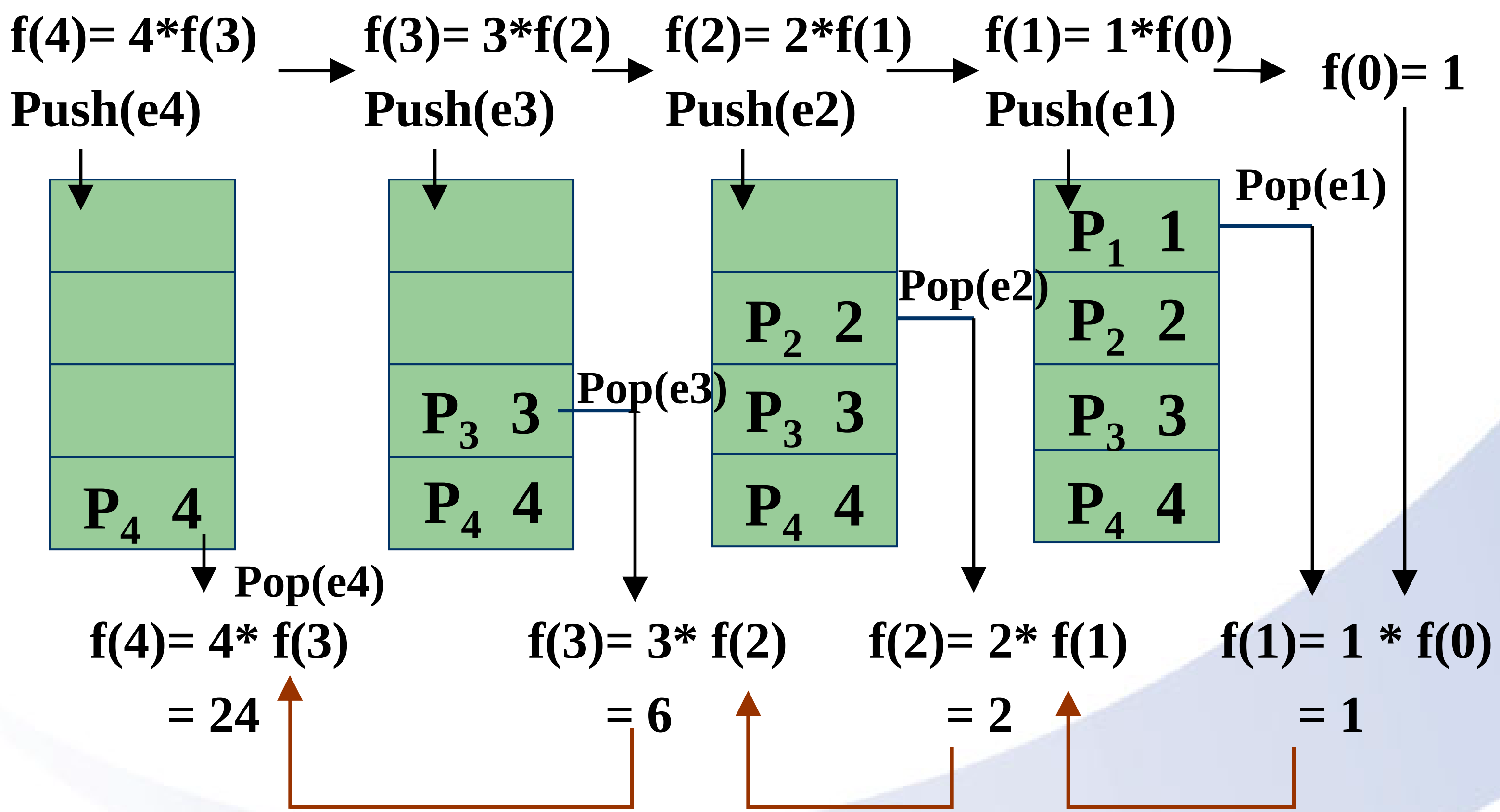
高级语言编译程序是利用栈来实现的。



调用时执行入栈操作保存现场，返回时执行出栈操作恢复现场。

计算 4 ! 递归过程图示:

下图中 P_i 代表现场信息，栈元素由现场信息和参数构成



1.3 递归

- **3. 递归算法设计**
  - 如果待解决的问题具备下列两个特性，就可以考虑使用递归:
    ① 复杂的问题可以转换为简单些的一个或几个子问题；
    ② 最简单的问题可以直接解决。

1.3 递归

- 3. 递归算法设计
  - 定义: 树是一个或多个结点的有限集合，它使得:
    - 有一个指定为根(root)的结点
    - 剩余结点被划分成m≥0个不相交的集合：T_1，…，T_m，这些集合的每一个又都是一棵树，并称T_1，…，T_m为根的子树。

```
int   P(参数表)
{
    if  (递归出口)
        简单操作
    else
    {
        简单操作;
        P;
        简单操作;
    }
}
```

初始情况

递归部分

1.3 递归

- **3. 递归算法设计**
  - 定义:二元树是结点的有限集合,
    - 它或者为空,
    - 或者由一个根和两棵称为左子树和右子树的不相交二元树所组成
  - 二分检索树(略)

```
int   P(参数表)
{
    if  (递归出口)
        简单操作
    else
    {
        简单操作;
        P;
        简单操作;
    }
}
```

初始情况

递归部分

1.3 递归

■ 3. 递归算法设计

```
int  P(参数表)
{
   if (递归出口)
      简单操作
    else
   {
      简单操作;
      P;
      简单操作;
    }
}
```

规模或与规模相关的参数

初始情况

降低规模的处理

对递归结果的处理

1.3 递归

- **3. 递归算法设计**
  - **斐波那契(Fibonacci)序列**
    - **兔子的问题: 假设小兔子每一个月长成大兔子,大兔子每一个月生一个小兔子,第一个月有一只小兔子,不考虑兔子的寿命,求n个月后有多少只兔子?**
  - **例1.3 斐波那契(Fibonacci)序列:**

$$F_0 = F_1 = 1$$

$$F_i = F_{i-1} + F_{i-2} \ (i>1)$$

1.3 递归

■ 3. 递归算法设计

□斐波那契(Fibonacci)序列

```
procedure  F(n)
//返回第n个斐波那契数//
    integer n
    if n ≤ 1
        then return(1)
    else
        return(F(n-1) + F(n-2))
    endif
```

$$\begin{aligned}F(5) &= F(4) + F(3)\\ &= (F(3) + F(2)) + ((F(2) + F(1))\\ &= ((F(2) + F(1)) + (F(1) + F(0))) + ((F(1) + F(0)) +1)\\ &= (((F(1) + F(0)) + 1) + (1 +1)) +(1 + 1) +1)\end{aligned}$$

F1(5) = F2(2, 5, 1, 1) = F2(3, 5, 1, 2)
= F2(4, 5, 2, 3) = F2(5, 5, 3, 5)
= F2(6, 5, 5, 8)

■ 3. 递归算法设计

- 斐波那契(Fibonacci)序列
  - 算法效率：对F(n-1)、F(n-2)存在大量的重复计算
  - 改进：保存中间结果

改进的算法

```
procedure F1(n)
    //返回第n个斐波那契数//
    integer n
    if n<2 then return(1)
      else return(F2(2, n, 1, 1))
    endif
end F1
```

```
procedure F2(i, n, x, y)
    if i≤ n then
        call F2(i+1, n, y, x+y)
    endif
    return(y)
end F2
```

■ 3. 递归算法设计

□阶乘函数

➢阶乘函数定义为：$n! = \begin{cases} 1 & n = 0 \\ n(n-1)! & n > 0 \end{cases}$

```
procedure Factorial(n)
{
    if n==0 then
        return(1)
    else
        return(n*Factorial(n-1))
    endif
}
```

1.3 递归

■ 3. 递归算法设计

□最大公因数

- 已知两个非负整数a和b，且a＞b≥0，求这两个数的最大公因数。
- 辗转相除法：
  - ✓若b=0，则a和b的最大公因数等于a；
  - ✓若b＞0，则a和b的最大公因数等于b和用b除a的余数的最大公因数。

$$GCD(a,b)=\begin{cases} a & b=0 \\ GCD(b,a\%b) & b>0 \end{cases}$$

GCD(22, 8)=GCD(8, 22%8)=GCD(8, 6)=GCD(6, 8%6)
=GCD(6, 2)=GCD(2, 6%2)=GCD(2, 0)=2

1.3 递归

- 3. 递归算法设计
  - 最大公因数
    - 求最大公因数

```
procedure  GCD(a, b)
      // 约定a>b //
         if b=0 then return(a)
               else return(GCD(b, a mod b))
         endif
end GCD
```

GCD(22, 8)=GCD(8, 22%8)=GCD(8, 6)=GCD(6, 8%6)
=GCD(6, 2)=GCD(2, 6%2)=GCD(2, 0)=2

1.3 递归

- **4. 递归在非数值算法设计中的应用**
  - 已知元素x，判断x是否在A(1:n)中。
  - 在A(1:n)中检索x

```
procedure  SEARCH(i)
  //如果在A(1：n)中有一元素
   A(k)=x，则将其第一次出现的
   下标k返回，否则返回0//
    global n, x, A(1:n)
    case
      :i>n: return(0)
      :A(i) = x; return(i)
      :else: return(SEARCH(i+1))
    endcase
end SEARCH
```

$$T(A, i, n, x)= \begin{cases} 0 & i>n \\ i & A[i]=x,\ i\le n \\ T(A, i+1, n, x) & A[i]\neq x,\ i\le n \end{cases}$$

1.3 递归

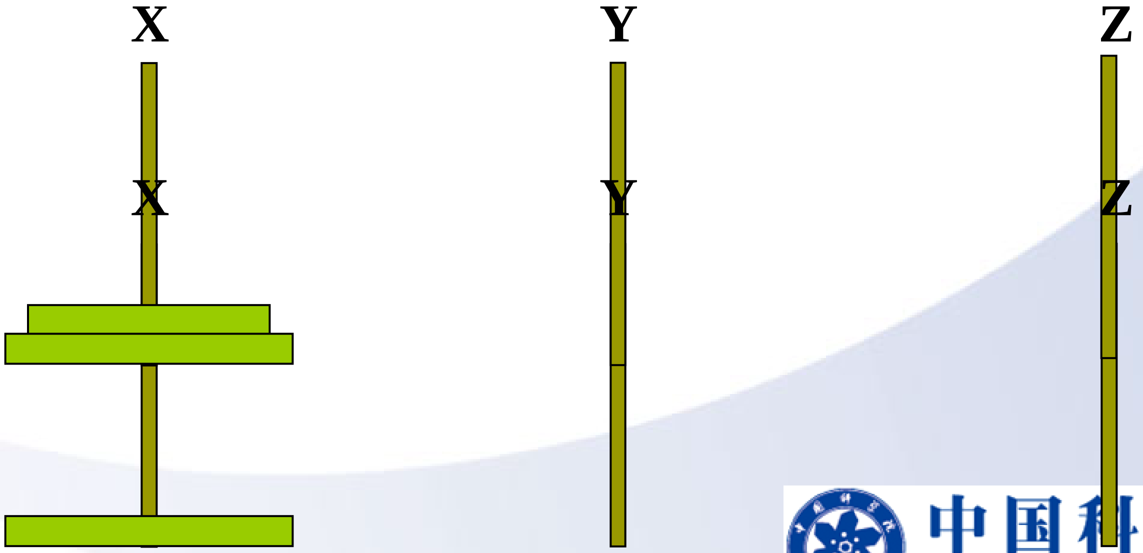
- 4. 递归在非数值算法设计中的应用
  - n阶Hanoi问题
    - X，Y，Z是三个塔座，开始时有n个盘子依其半径大小套在柱子X上，其中半径大的在下面。现要将X上的圆盘移到Z上，并仍按同样顺序叠置。
    - 移动规则：
      ① 每次只能移动1个圆盘；
      ② 任何时刻都不允许将半径大的圆盘压在半径小的圆盘之上；

1.3 递归

■ 4. 递归在非数值算法设计中的应用

□n阶Hanoi问题

- □ Hanoi(n, X, Y, Z)
- ➢ n=1, X→Z
- ➢ n=2, X→Y, X→Z, Y→Z



1.3 递归

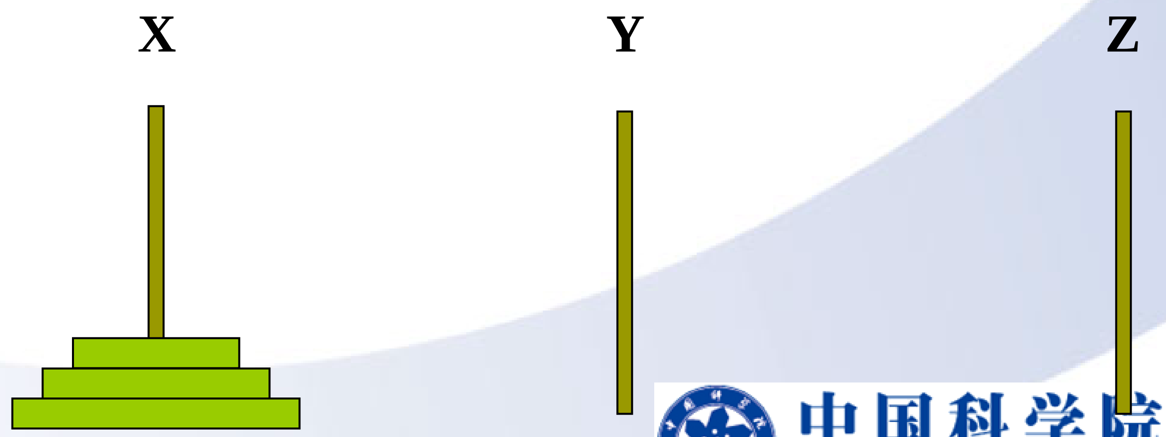
■ 4. 递归在非数值算法设计中的应用

□n阶Hanoi问题

- □ Hanoi(n, X, Y, Z)
- ➢ n=3, Hanoi(2, X, Z, Y)

  $X\xrightarrow{3}Z$

  Hanoi(2, Y, X, Z)



1.3 递归

■ 4. 递归在非数值算法设计中的应用

□n阶Hanoi问题

```
procedure Hanoi(n, X, Y, Z)
  if (n = 1) then move(X, Z)
   else
      Hanoi(n–1, X, Z, Y)
      move(X, Z)
      Hanoi(n–1, Y, X, Z)
  endif
end Hanoi
```

1.3 递归

■ 4. 递归在非数值算法设计中的应用

□棋子的移动问题

有2n个棋子(n≥4)排成一行，白子用0表示，黑子用1表示，例如n=5时初始状态为

0 0 0 0 0 1 1 1 1 1 _ _(后面至少有两个空位)，要求通过棋子移动最终成为

0 1 0 1 0 1 0 1 0 1

移动规则：

① 每次同时移动相邻两个棋子，颜色不限，移动方向不限，

② 每次移动必须跳过若干棋子，

③ 不能调换两个棋子的位置。

1.3 递归

■ 4. 递归在非数值算法设计中的应用

□棋子的移动问题

➢n=4时

0 0 0 0 1 1 1 1 _ _

① 0 0 0 _ _ 1 1 1 0 1

② 0 0 0 1 0 1 1 _ _ 1

③ 0 _ _ 1 0 1 1 0 0 1

④ 0 1 0 1 0 1 _ _ 0 1

⑤ _ _ 0 1 0 1 0 1 0 1

1.3 递归

- 4. 递归在非数值算法设计中的应用
  - 棋子的移动问题
    - n=5时

      0 0 0 0 0 1 1 1 1 1_ _

      0 0 0 0 _ _ 1 1 1 1 0 1

      0 0 0 0 1 1 1 1 _ _ 0 1

    - n=6时

      0 0 0 0 0 0 1 1 1 1 1 1_ _

      0 0 0 0 0 _ _ 1 1 1 1 1 0 1

      0 0 0 0 0 1 1 1 1 1 _ _ 0 1

1.3 递归

■ 4. 递归在非数值算法设计中的应用

□棋子的移动问题

递归出口：n=4

move (4, 5) →(9, 10)	0 0 0 _ _ 1 1 1 0 1
move (8, 9) →(4, 5)	0 0 0 1 0 1 1 _ _ 1
move (2, 3) →(8, 9)	0 _ _ 1 0 1 1 0 0 1
move (7, 8) →(2, 3)	0 1 0 1 0 1 _ _ 0 1
move (1, 2) →(7, 8)	_ _ 0 1 0 1 0 1 0 1

2n个棋子的移动：chess(n)

move (n, n+1) →(2n+1, 2n+2)

move (2n-1, 2n) →(n, n+1)

call chess(n-1)

简单操作
降低问题规模

1.3 递归

```
procedure Chess(n)
   if n=4 then
      move  (4, 5) →(9, 10)
      move  (8, 9) →(4, 5)
      move  (2, 3) →(8, 9)
      move  (7, 8) →(2, 3)
      move  (1, 2) →(7, 8)
   else
      move  (n, n+1) →(2n+1, 2n+2)
      move  (2n–1, 2n) →(n, n+1)
      call  Chess(n-1)
   endif
end Chess
```

递归出口

A(9)←A(4)
A(10)←A(5)

递归调用

1.3 递归

■ 4. 递归在非数值算法设计中的应用

□简单的0/1背包问题

- ➢背包可容纳物品的最大质量为m，现有n件物品，质量分别为$m_1, m_2, \ldots, m_n$，m_i均为正整数，要从n件物品中挑选若干件，使放入背包的质量之和正好为m。
- ➢例：$m=20, n=5, (m_1, m_2, m_3, m_4, m_5)=(3, 5, 8, 9, 10)$
  $(x_1, x_2, x_3, x_4, x_5)=(1, 0, 1, 1, 0)$
- ➢m=18?
- ➢m=28?
- ➢注：对于第i件物品要么取，要么舍，不能取一部分，因此这个问题可能有解，也可能无解。

问题分析 knap(m, n)

初始： m　m_1　m_2　……　m_{n-1}　m_n

m_n=m　m　m_1　m_2　……　m_{n-1}　true

m_n<m　m　m_1　m_2　……　m_{n-1}

n>1,即还有可选物品 ⇩ $knap(m-m_n, n-1)$

$$\begin{cases} \text{有解} & knap(m, n) \leftarrow knap(m-m_n, n-1) \\ \text{无解} & knap(m, n) \leftarrow knap(m, n-1) \end{cases}$$

m_n>m　m　m_1　m_2　……　m_{n-1}

n>1 ⟹ knap(m, n–1)

否则　false

1.3 递归

■ 4. 递归在非数值算法设计中的应用

□简单的0/1背包问题

```
function knap(m, n)
    case
    :m–m_n=0: knap ←true
    :m–m_n>0: if  n>1 then
                   if knap(m–m_n, n–1)=true then
                              knap ←true
                     else   knap ←knap(m, n–1)  endif
                 else knap ←false  endif
    :m–m_n<0: if n>1 then knap ←knap(m, n–1)
                              else  knap ←false;  endif
    endcase
end knap
```

- **End**