L8：递归

吉林大学计算机学院
谷方明
fmgu2002@sina.com

学习目标

- 理解递归的定义
- 利用递归求解问题
- 理解递归的实现机制
- 能够消递归
- 掌握记忆化搜索
- 了解递归与分治法
- 掌握递归树和主定理

例1：引入

- 从前有座山，山上有座庙，庙里有一个老和尚在给小和尚讲故事，故事里说，从前有座山，山上有座庙，庙里有一个老和尚在给小和尚讲故事，故事里说，……

```c
void story( ) {
    printf("从前有座山，山里有座庙，庙里有一个老和尚在给小和尚讲故事，故事里说，");

     story();
}

int main( ) {
    story();
}
```

如何终止

```
#define MAX 10000
void story(int n)
{
   if(n < MAX){
      printf("从前有座山，山里有座庙，庙里有一个老和尚在给小和尚讲故事，故事里说，");
      story( n+1 );
   }
 else printf("都讲%d遍了！你烦不烦哪？\n",MAX);
}
```

递归的定义

- 如果一个对象部分地包含自身，或者利用自身定义自身的方式来定义或描述，则称这个对象是递归的；

- 递归构成：递归定义+递归出口(终止条件)

- 如果一个过程直接或间接地调用自身，则称这个过程是一个递归过程。直接递归和间接递归

例2

- 自然数集合的定义
  - ✓ 1 是 自然数；
  - ✓ 若 s 是自然数，则 s + 1 也是自然数；（+1：后继）

- 数列的定义
  - ✓ $a_{n+1} = a_n + d$
  - ✓ $a_1 = 1$

递归求解问题

- 递归是一种重要的问题求解工具
  - ✓ 魔法
- 适用条件
  - ✓ 问题的定义是递归的；
  - ✓ 问题涉及的数据结构是递归的；
  - ✓ 问题的解法满足递归性质(与问题规模n有关)

1、问题的定义是递归的

- 例： 阶乘 n! = 1*2*……*n

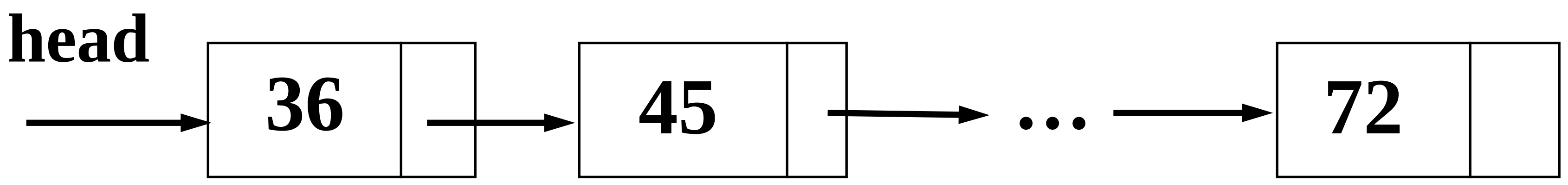
$$n! = \begin{cases} 1 & \mathrm{n} = 0 \\ n*(n-1)! & \mathrm{n} > 0 \end{cases}$$

$$n! = \begin{cases} 1 & \mathbf{n = 0} \\ n*(n-1)! & \mathbf{n > 0} \end{cases}$$

```
long long Fac (long long n)
{
   if (n==0)   return 1;
   else   return n * Fac (n-1);
}
```

2、问题涉及的数据结构是递归的

例:　单链表



单链表的递归定义：

（1）空指针是一个单链表(空单链表);

（2）包含一个非空结点，该结点的指针域指向一个单链表。

打印单链表的所有数据

```
void print（Node  *p）
{
   if（p == NULL）return;
   else {
         printf(“%d\n“, p → data) ;
         print（p → next）；
   }
}
```

打印单链表最后一个数据

```
void findLast (Node * f ) {
   if ( f ==  NULL)  return;
   if ( f →next == NULL )
       printf("%d\n", f → data) ;
   else findLast ( f →next ) ;
}
```

3、问题的解法满足递归性质

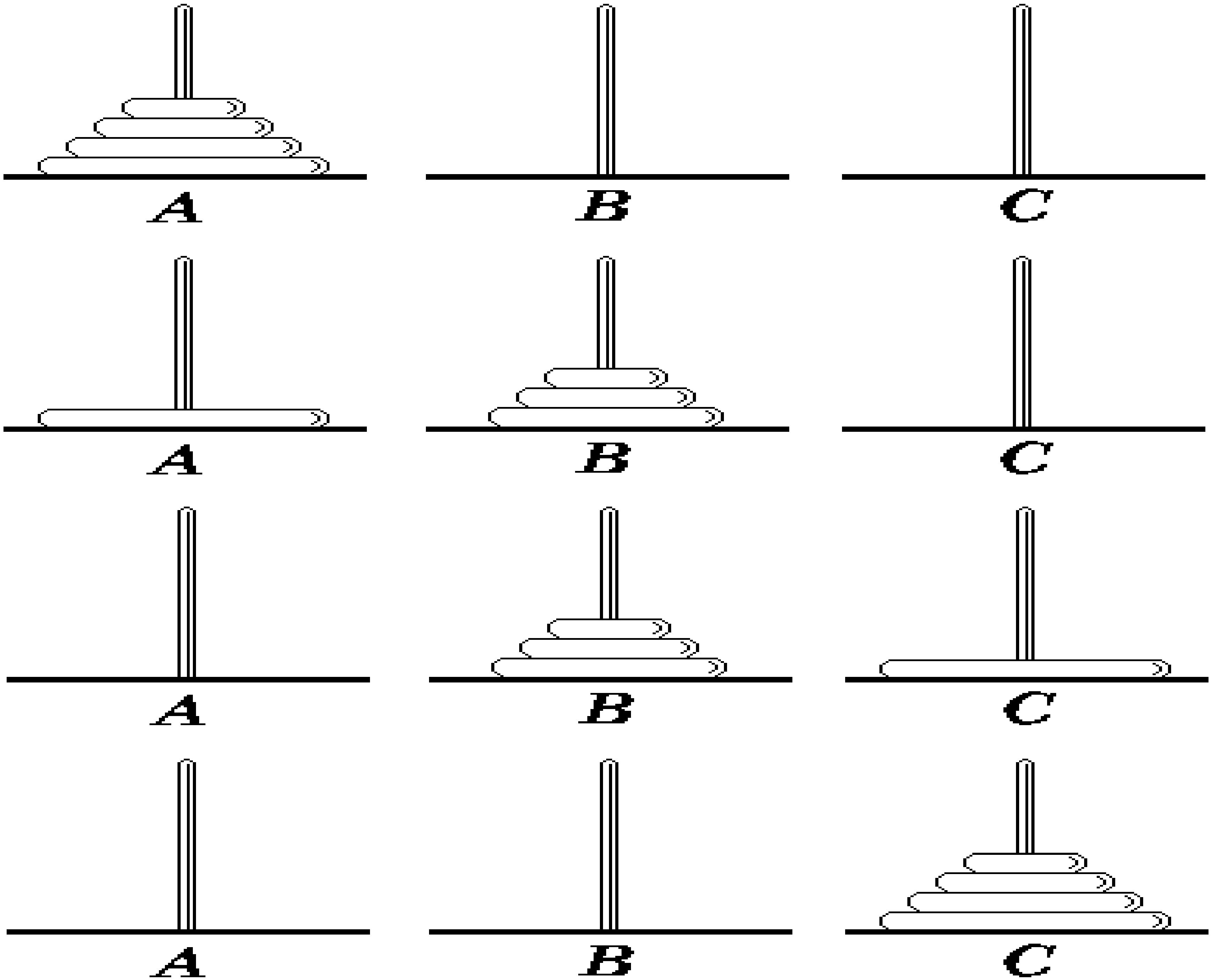
- **例：汉诺塔(Tower of Hanoi)问题**

在世界中心贝拿勒斯（印度北部）的圣庙里，**一块黄铜板上插着三根宝石针**。印度教的主神梵天创世时，**在其中一根针上从下到上地穿好了由大到小的64片金片，这就是所谓的汉诺塔**。不论白天黑夜，总有一个僧侣在按照下面的法则移动这些金片：**一次只移动一片，不管在哪根针上，小片必须在大片上面。**

僧侣们**预言**，当所有的金片都从梵天穿好的那根针上移到另外一根针上时，世界就将在一声霹雳中消灭，而梵塔、庙宇和众生也都将同归于尽。


团队烧脑游戏
汉诺塔
5块毛毡圆板+1张游戏布+袋子


HEARING!
ONE POINT OR TWO


A
B
C
A
B
C
A
B
C
A
B
C

解决规模为n的汉诺塔

- 即将 n 个金片 从 A 移到 C 利用 B

- 第一步：将 n-1 个金片从 A 移到 B 利用 C；
- 第二步：将第 n 个金片从 A 移到 C；
- 第三步：将 n-1个 金片从 B 移到 C 利用A ；

汉诺塔的实现

```
void hanoi (int  n, char a, char b, char c)
{
     if(n>0){
          hanoi(n-1,a,c,b);
          printf("%c -> %c\n", a , c );
          hanoi(n-1,b,a,c);
     }
}
```

证明：所述方法移动的步数最少

- 数学归纳法
- 当n=1时，所述方法只要1步完成，步数最少。
- 设当n=k时，所述方法移动的步数最少。

  当n=k+1时，必然要出现一种局面：最大的盘子在A柱上，其它k个盘子在B柱上，C柱为空。根据假设，所述方法把A上k个盘子利用C柱移动到B柱的步数最少；最大的盘子移动到C上需要1步；所述方法把B柱上的k个盘子利用A柱移动到C柱的步数也最少，因此所述方法在k+1个盘子的情况下移动步数最少。

移动的最少步数f_n

- 移动的最少步数f_n形成一个数列

- $f_1 = 1$
- $f_n = f_{n-1} + 1 + f_{n-1}$
- 解得： $f_n = 2^n – 1$

- $f_{64} = 1.84467440*10^{19}$
- 假设每秒移动1次，超过5800亿年

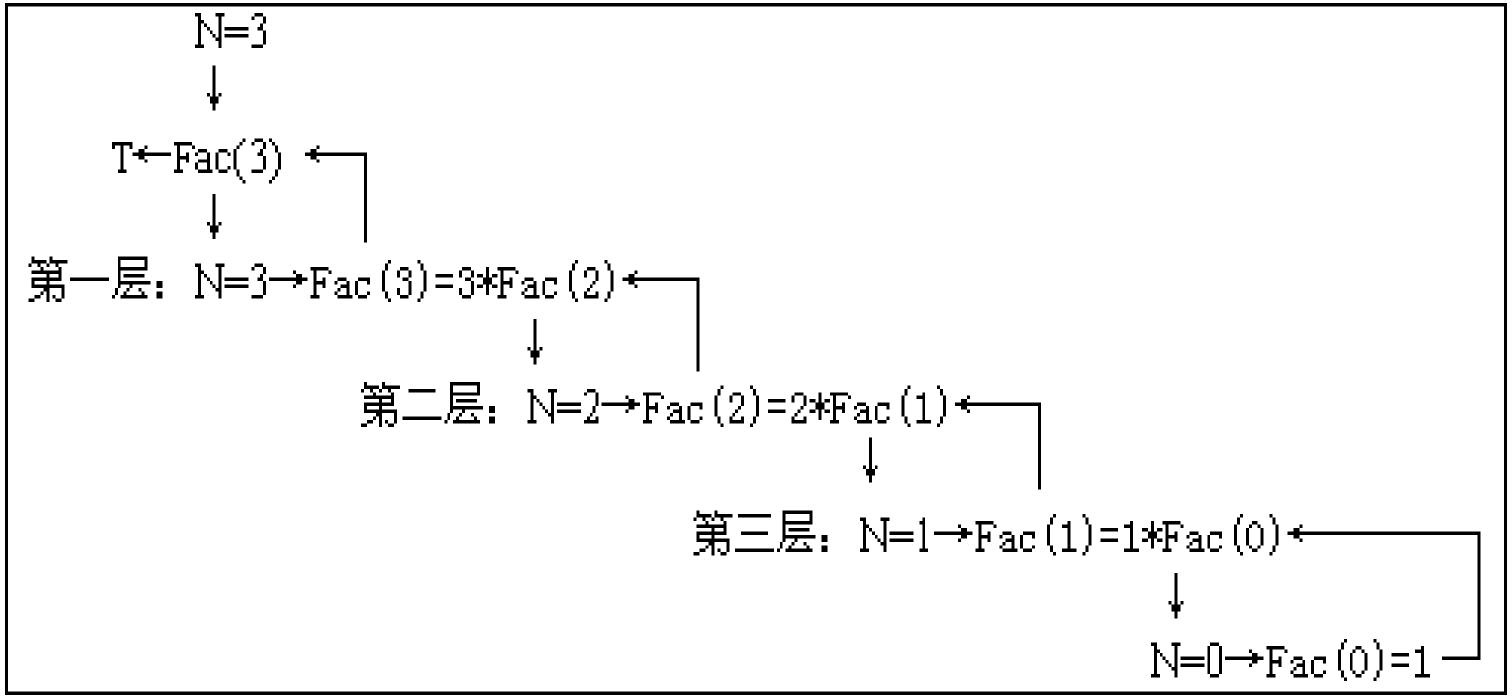
课堂练习

- 输入一个整数，用递归算法将整数倒序输出。
- $x^n = x \times x \times \ldots \times x$ （n 个x 连乘）
  - ✓ $x^n = x^{n-1} \times x$
  - ✓ $x^n = x^{n/2} \times x^{n/2}$ n是偶数

    $= x^{n/2} \times x^{n/2} \times x$ n是奇数
- gcd (a , b)
  - ✓ **gcd(a,b) = gcd(b, a%b)**
  - ✓ **gcd(a,0) = a**

递归的执行过程

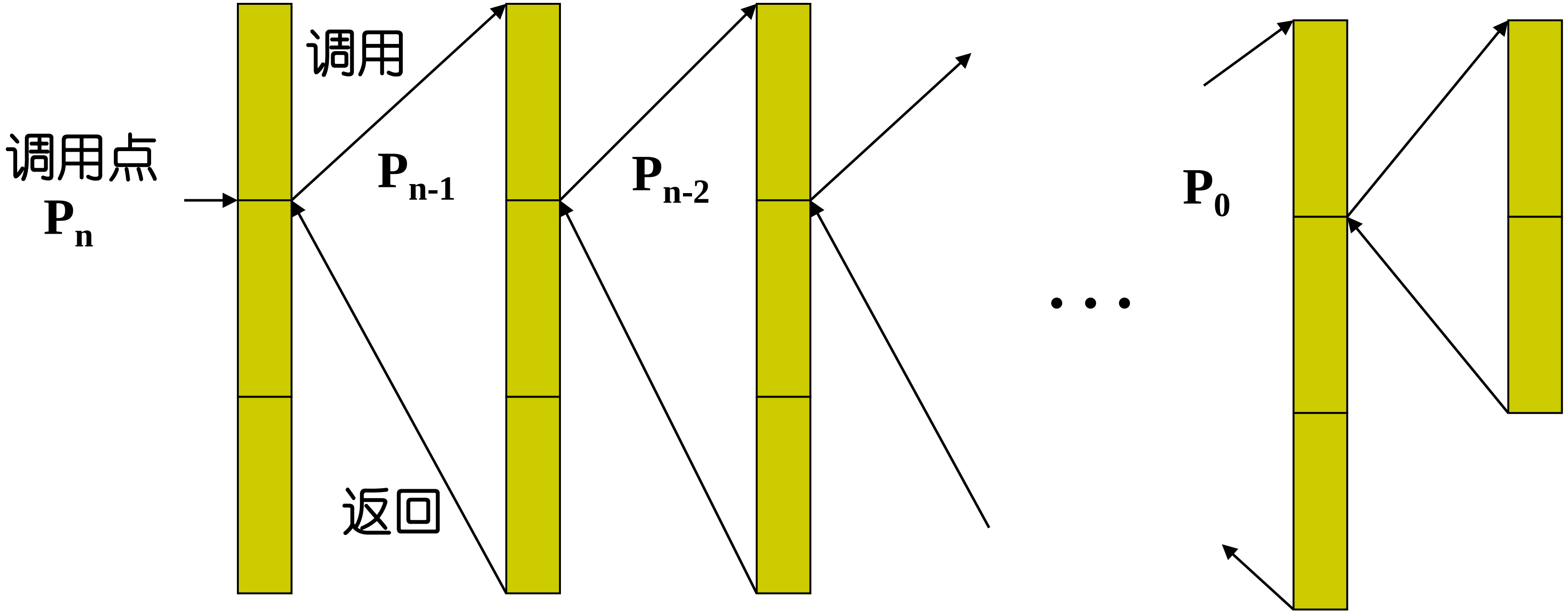
- 以阶乘为例



递归过程分递推和回归两个阶段

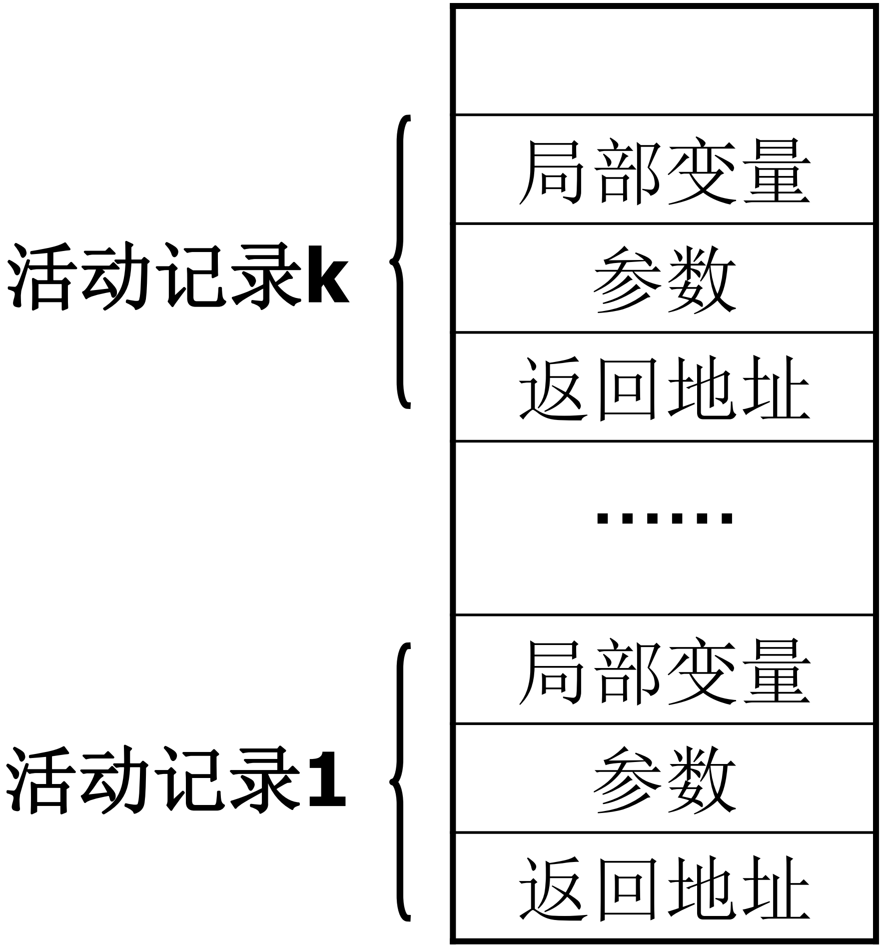
- 在递推阶段，把较复杂问题（规模为n）的求解推到比原问题简单的问题（规模小于n）来求解。
  - 例：求n!转化为求(n-1)!,依次类推，直至计算到n=0时；在递推阶段，必须有终止的情况。
- 在回归阶段，当获得最简单问题的解后，逐级返回，依次得到稍复杂问题的解。
  - 例如知道0！=1，可以得到1！=1，2！=2，…，直到n!

- main()　f(n)　f(n-1)　f(1)　f(0)



递归的实现

- 为保证递归或函数调用执行正确，系统使用 **栈** 来管理实现。
- 调用（进层）
  - ✓ 保存本层参数和返回地址
  - ✓ 分配空间，参数传递
  - ✓ 程序转移到被调函数入口
- 返回(退层)
  - ✓ 保存计算结果
  - ✓ 释放空间，恢复上层参数
  - ✓ 依照返回地址转移



消递归

- 递归由系统的工作栈管理实现；因此，使用递归解题简洁；

- 但有些情况下，要实现递归与非递归的转换
  - ✓ 系统空间栈崩溃（ **64K** ）
  - ✓ 不是所有的语言都支持递归（升级前的**FORTRAN**）
  - ✓ 提高程序的效率 （函数调用和返回代价略大）

消递归原理

- 简单的，直接使用循环结构代替
  - ✓ 尾递归：递归发生在最后一步；
  - ✓ 例：阶乘

- 更多的，要基于栈的方式，按照递归机制模拟
  - ✓ 方法一：规则法
  - ✓ 方法二：技巧法

```
void hanoi_1(int n,int a,int b,int c){
        int t,addr;
        top=0;
L1: if(n>0){
          push(n,a,b,c); push(2); n--;t=b;b=c;c=t;
          goto L1;
        L2:printf("%d->%d\n",a,c);
          push(n,a,b,c);push(3); n--;t=a;a=b;b=t;
          goto L1;
        }
L3: if(top){
                pop(addr); pop(n,a,b,c);
                if(addr==2) goto L2;
                else if(addr==3) goto L3;
        }
}
```

使用转换技巧

Hanoi (n， a， b， c)

问题分解

Hanoi（n-1，a，c，b）

MOVE（a，c） => Hanoi（1，a，b，c）

Hanoi（n-1，b，a，c）

消递归压栈

S⇐（n-1，b，a，c）.

S⇐（1，a，b，c） .

S⇐（n-1，a，c，b） .

汉诺塔非递归

算法HI(n)

HI1[建立堆栈]

```
    CREATEStack（S）
```

HI2[堆栈初始化]

```
    S⇐（n， a， b， c）
```

HI3[利用栈实现递归]

```
    while( ！ StackEmpty(S) ) {
            (n,a,b,c）  ⇐ S;
            if (n == 1) MOVE ( a, c );
            else {     S⇐(n-1, b, a, c);
                       S⇐(1, a, b, c);
                       S⇐(n-1, a, c, b) ; }    }
```

```
void hanoi_2(int n,int a,int b,int c){
        top=0;
        push(n,a,b,c);
        while(top){
                pop(n,a,b,c);
                if(n==1) printf("%d->%d\n",a,c);
                else{
                        push(n-1,b,a,c);
                        push(1,a,b,c);
                        push(n-1,a,c,b);
                }
        }
}
```

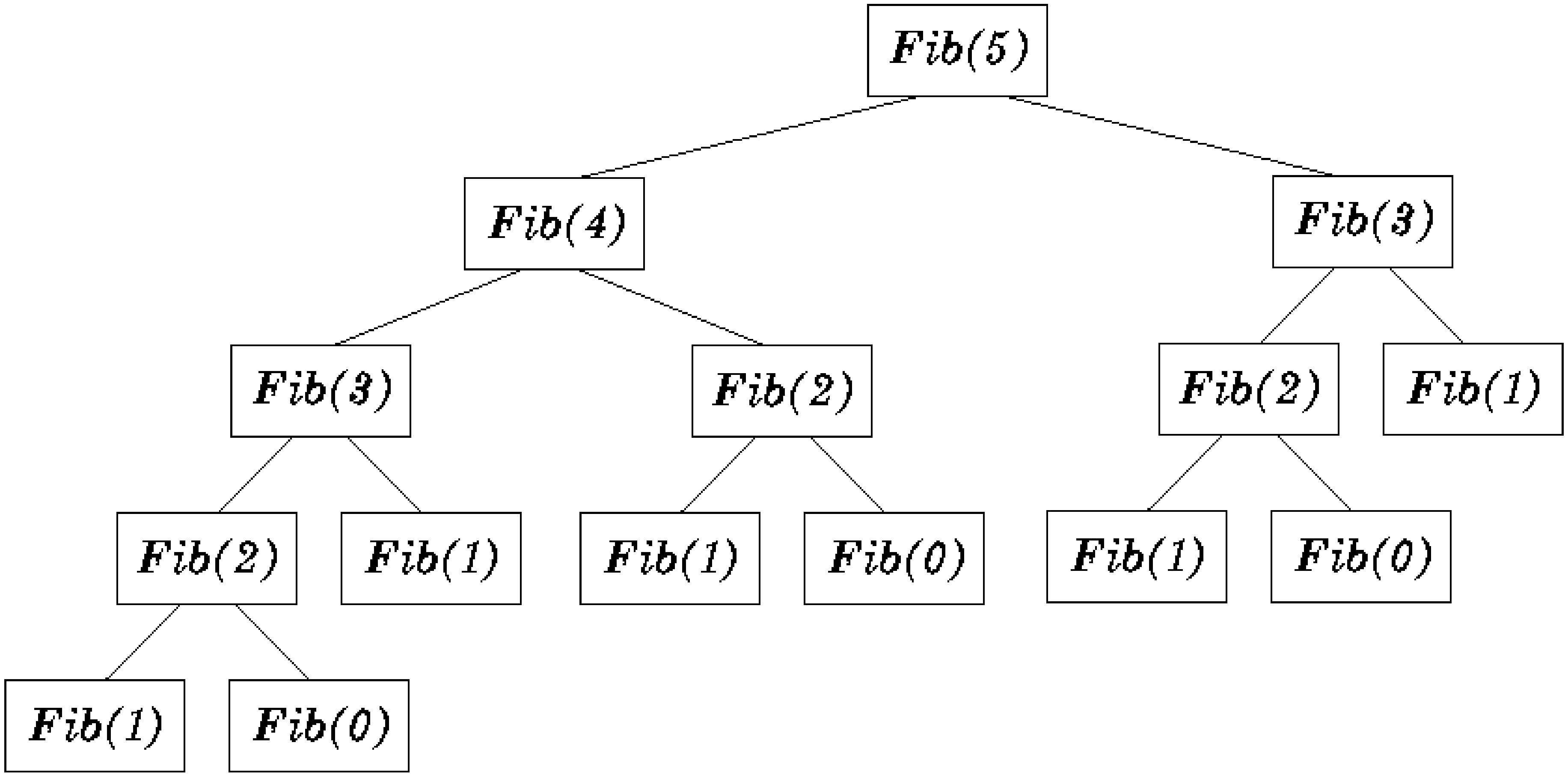
递归的效率

- 递归方法在解决某些问题时是最直观、最方便的方法，但未必是高效的方法。
- 例 斐波那契数列

假定一对大兔子每一个月可以生一对小兔子，而小兔子出生后两个月就有生殖能力．问从一对大兔子开始，一年后能繁殖成多少对兔子？

$$Fib(n) = \begin{cases} n, & n = 0,1 \\ Fib(n-1) + Fib(n-2), & n > 1 \end{cases}$$

```
long Fib ( long n ) {
    if ( n <= 1 ) return n;
    else return Fib (n-1) + Fib (n-2); }
```



效率：相同子问题的重复计算

- 斐波那契数列的递归调用树，调用次数 O(2^k)
- 使用循环迭代法，只需要 O(n)

  long long $f_1 = 1$， $f_2 = 0$， $f = 0$；
  for (int i=2； i<=n； i++) { $f=f_1+f_2$； $f_2=f_1$； $f_1=f$； }

- 当 **n = 35** 时，斐波那契数迭代函数需进行**33**次加法，而递归函数需要进行**185万**次函数调用！

原理：用空间换时间

```
long Fib ( long n ) {
    if ( n <= 1 ) return n;
    else return Fib (n-1) + Fib (n-2); }
```

```
int f[MAXN];//初始化为-1.
long fibo (long n)
{
     if(f[n]>=0) return f[n];
     if (n < = 1)  return f[n]=n；
     else return f[n]=fibo(n-1)+fibo(n-2)；
}
```

- 记忆化搜索

递归和分治法

- 分治法的基本设计思想（分而治之）
  1. 分解：将原问题分解为若干个规模较小、相互独立、与原问题形式相同的子问题。
  2. 治理：求解各个子问题。
  3. 合并：将子问题的解合并构成原问题的解。
- 子问题与原问题形式相同，只是规模较小，可以用递归方法解决
- 适用条件：能有效的分解和合并。

例：用分治法求n个数中的最大值

- 设计策略
  - ✓ 分解：中间划分，形成左右两个相同子问题；
  - ✓ 治理：递归求解两个子问题；
  - ✓ 合并：左右两个子问题的结果的最大值就是原问题的最大值。

例：用分治法求n个数中的最大值

- 算法描述(ADL-C)

算法RangeMax (A, i, j. ma)

RM1. [递归出口]

```
    if(i>=j) return ma = A[i];
```

RM2. [分治]

```
    mid = (i + j) / 2;
    RangeMax(A, i, mid, ma1);
    RangeMax(A, mid+1, j, ma2);
    return ma = max(ma1,ma2) ▌
```

时间复杂度： $T(n) = n$

问题的划分

- 原问题可以一分为二（二分法）、一分为三、……

- 原问题划分成多少个子问题合适？
  - ✓ 通常采取二分法，这么划分既简单又均匀。

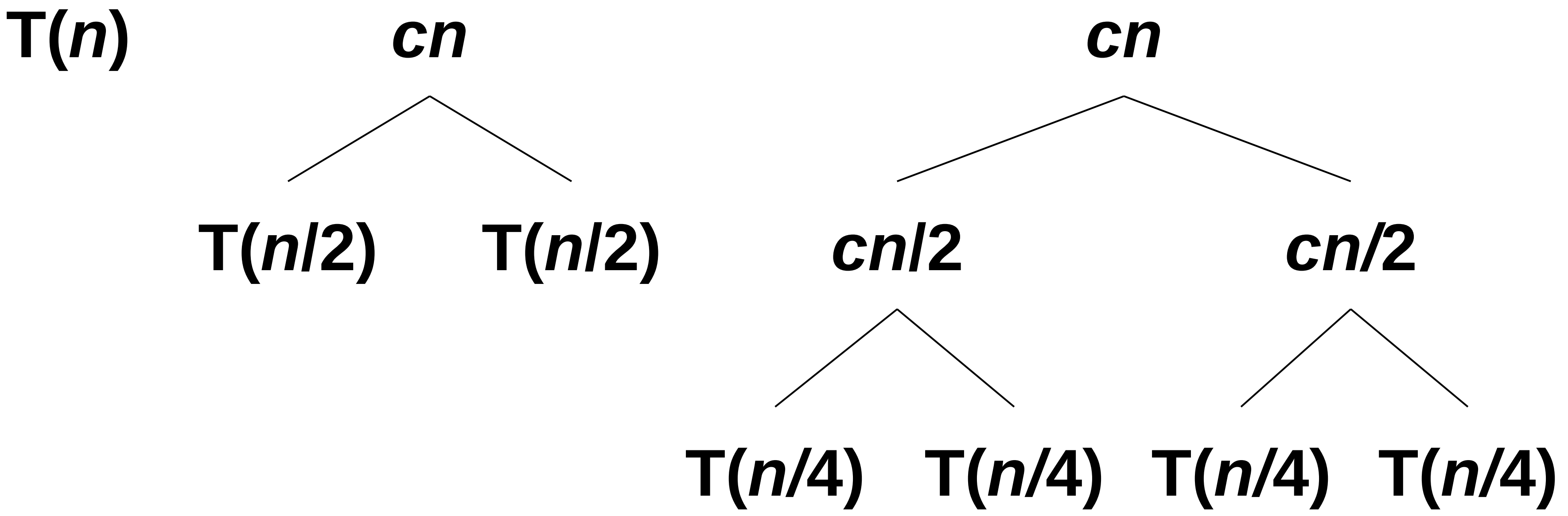
数据结构常用算法

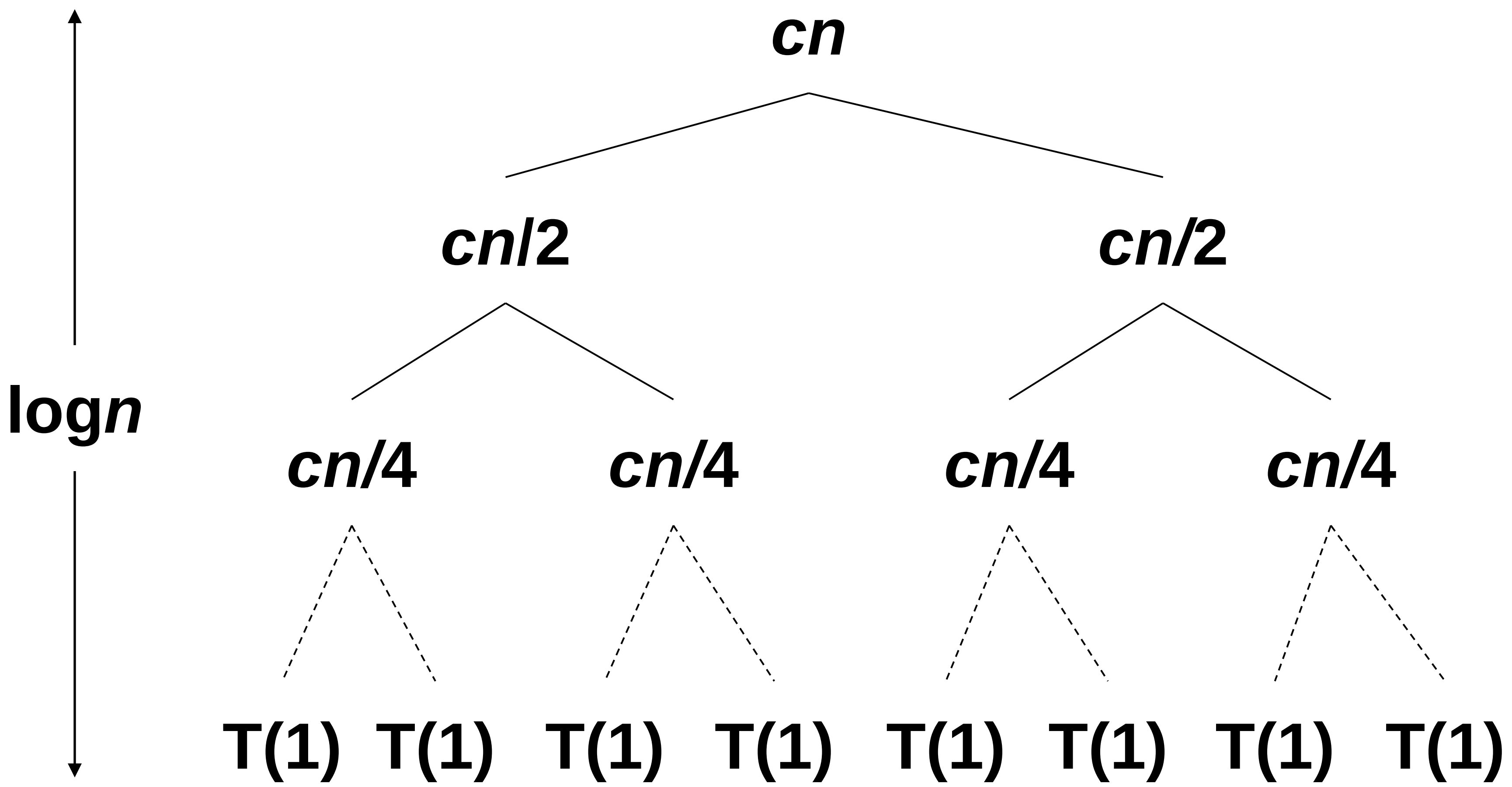
- 枚举法
- 模拟法
- 贪心法
- 分治法
- 递归法
- 递推法
- 搜索（基本）
- 动态规划（基本）
- ……

递归树方法

- 在递归树中，每个结点表示一个单一子问题的代价，每个子问题对应某次递归调用。
- 将树中每层中的代价求和，得到每层代价。
- 然后将所有层的代价求和，得到所有层次的递归调用的总代价。
- 主要用途
  - ✓ 猜测/估算递归方法的渐进上界
  - ✓ 准确计算递归方法的时间复杂度

例：T(n) = 2T(n/2) + O(n) 上界





设T(1)是常量，O(n)用cn(精度损失常数项)代替，

$T(n) \leqslant cn * \log n + \Theta(n) = O(n\log n)$

- 正确性：存在d，*n*足够大,使得T(n)≤dnlog*n*.

  T(n) = 2T(n/2) + O(n)

  ≤ 2*d*(n/2)*log(n/2) + cn

  ≤dnlogn + (c-d)n

  当d≥c时，T (n) ≤dnlogn

- 代入法
  - ✓ 猜测解的形式（递归树等方法）
  - ✓ 用数学归纳法等证明解是正确的

例：BS算法——递归树

- 为尽量精确易算，递归树最好是满二叉树。
- 左右子树结点最多差1，最下一层有两种情况
  - ✓ 由T(2)和T(3)构成
  - ✓ 由T(3)和T(4)构成
- 第一种情况
  - ✓ $2i + 3j = n$
  - ✓ $i + j = 2^k$
  - ✓ $T(n) = 2+\ldots+2^k + i + 3j$

    $\leqslant (5/3)n - 2$

主定理（Master Theorem）

- 令 a ≥ 1 和 b>1 是常数，f(n) 是一个函数，T(n) 是定义在非负整数上的递归式：

$$T(n) = a\,T(n/b) + f(n)$$

可将n/b解释为 $\lfloor n/b \rfloor$或 $\lceil n/b \rceil$。那么：

1. 有常数ε>0, 使$f(n)=O(n^{\log_b a-\varepsilon})$, 则$T(n)=\Theta(n^{\log_b a})$
2. 若 $f(n)=\Theta(n^{\log_b a})$, 则$T(n) = \Theta(n^{\log_b a}\log n)$
3. 有常数ε>0, 使$f(n)=\Omega(n^{\log_b a+\varepsilon})$, 且存在c<1和 所有足够大的n, 有 $af(n/b) \le cf(n)$, 则$T(n)=\Theta(f(n))$

例：主方法（Master Method）

- **$T(n) = 9T(n/3) + n$**
  - ✓ $n^{\log_b a} = n^2$, $f(n) = n = O(n^{\log_b a - \varepsilon})$， $T(n) = \Theta(n^2)$
- **$T(n) = T(3n/2) + 1$**
  - ✓ $n^{\log_b a} = 1$, $f(n) = 1 = \Theta(n^{\log_b a})$， $T(n) = \Theta(\log n)$
- **$T(n) = 3T(n/4) + n\log n$**
  - ✓ $n^{\log_b a} = O(n^{0.793})$，取$c = 3/4$，正则条件成立，$T(n)= \Theta(n\log n)$
- **$T(n) =2T(n/2) + n\log n$**
  - ✓ 没有适合情况。可证明$T(n)= \Theta(n\log^2 n)$

总结

- 递归的定义（递归定义+递归出口）
- 递归求解问题
  - ✓ 问题的定义是递归的；
  - ✓ 问题涉及的数据结构是递归的；
  - ✓ 问题的解法满足递归性质消递归
- 消递归（转换规则和技巧）
- 递归的效率（尾递归和记忆化搜索）
- 递归算法的时效分析工具（递归树、主定理）