第2章　递归与分治策略(1)

递归

内容

2.1 求解递归式

2.2 递归

- 阶乘函数
- Fibonacci数列
- 排列问题
- 整数划分问题
- Hanoi塔问题

2.3 分治

1. 二分搜索技术
2. 合并排序
3. 快速排序
4. 线性时间选择

递归与分治策略总体思想

- 将一个难以直接解决的大问题，分割成一些规模较小的相同子问题。如果这些子问题都可解，并可利用这些子问题的解求出原问题的解，则这种分治法可行。

2.1 求解递归式

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ aT(n/b) + f(n) & \text{if } n > 1 \end{cases}$$

- T(n)表示解规模为n的问题所需的计算时间
- 解规模为1的问题耗费1个单位时间
  - 对于规模一定的问题，所耗费的是常数项时间
- 分治法将规模为**n**的问题分成**a**个规模为**n/b**的子问题去解
- 将原问题分解为a个子问题以及将a个子问题的解合并为原问题的解需用f(n)个单位时间

主方法(Master method)求解递归式

Theorem 4.1 (Master theorem)

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

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

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

例子

- $T(n) = 4T(n/2) + n$ ⟹ $T(n) = \Theta(n^2)$
- $T(n) = 4T(n/2) + n^2$ ⟹ $T(n) = \Theta(n^2 \log n)$
- $T(n) = 4T(n/2) + n^3$ ⟹ $T(n) = \Theta(n^3)$

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \le cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

2.2 递归

1. 阶乘函数
2. Fibonacci数列
3. 排列问题
4. 整数划分问题
5. Hanoi塔问题

2.2 递归

- 直接或间接地调用自身的算法称为递归算法。用函数自身给出定义的函数称为递归函数。
- 由分治法产生的子问题往往是原问题的较小模式，这就为使用递归技术提供了方便。在这种情况下，反复应用分治手段，可以使子问题与原问题类型一致而其规模却不断缩小，最终使子问题缩小到很容易直接求出其解。这自然导致递归过程的产生。
- 分治与递归像一对孪生兄弟，经常同时应用在算法设计之中，并由此产生许多高效算法。

2.2 递归

1. 阶乘函数
2. Fibonacci数列
3. 排列问题
4. 整数划分问题
5. Hanoi塔问题

1 阶乘函数

- 阶乘函数可递归地定义为:

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \times (n-1)! & \text{if } n > 0 \end{cases}$$

边界条件

递归方程

- 边界条件与递归方程是递归函数的二个要素，递归函数只有具备了这两个要素，才能在有限次计算后得出结果。

算法描述

```
FACTORIAL(n)
    if n==0, return 1
    return n*FACTORIAL(n-1)
```

2.2 递归

1. 阶乘函数
2. Fibonacci数列
3. 排列问题
4. 整数划分问题
5. Hanoi塔问题

2 Fibonacci数列

- 无穷数列

  0,1,1,2,3,5,8,13,21,34,55,……

  称为Fibonacci数列。

- 递归定义：

$$
F(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F(n-1) + F(n-2) & \text{if } n > 1 \end{cases}
$$

边界条件

递归方程

算法描述

```
FIBONACCI (n)
    if n==0, return 0
    if n==1, return 1
    return FIBONACCI(n-1) + FIBONACCI(n-2)
```

递归算法

$$T(n) = \begin{cases} \theta(1) & \text{if } n \leq 1 \\ T(n-1) + T(n-2) + \theta(1) & \text{if } n \geq 2 \end{cases}$$

Naive recursive algorithm: $\Omega(\phi^n)$ (exponential time), where $\phi = (1+\sqrt{5})/2$ is the ***golden ratio***.

递归算法

Naive recursive algorithm: $\Omega(\phi^n)$ (exponential time), where $\phi = (1+\sqrt{5})/2$ is the ***golden ratio***.

```
FIBONACCI (n)
    if n==0, return 0
    if n==1, return 1
    return FIBONACCI(n-1) + FIBONACCI(n-2)
```

如何改进？

非递归算法

- **Bottom-up:**
  - Compute $F_0, F_1, \ldots, F_n$ in order, forming each number by summing the two previous.
  - Running time: $\Theta(n)$

```
FIBONACCI(n)
    let a[0..n] be a new vector
    a[0] = 0
    a[1] = 1
    for i=2 to n
        a[i] = a[i-1] + a[i-2]
    return a[n]
```

2.2 递归

1. 阶乘函数
2. Fibonacci数列
3. 排列问题
4. 整数划分问题
5. Hanoi塔问题

3 排列问题

- **设计一个递归算法生成 n 个元素的全排列。**
- 例：3个元素：1,2,3的全排列为：

```
[1 2 3]
[1 3 2]
[2 1 3]
[2 3 1]
[3 2 1]
[3 1 2]
```

```
1 2 3 4
1 2 4 3
1 3 2 4
1 3 4 2
1 4 3 2
1 4 2 3
2 1 3 4
2 1 4 3
2 3 1 4
2 3 4 1
2 4 3 1
2 4 1 3
3 2 1 4
3 2 4 1
3 1 2 4
3 1 4 2
3 4 1 2
3 4 2 1
4 2 3 1
4 2 1 3
4 3 2 1
4 3 1 2
4 1 3 2
4 1 2 3
```

3 排列问题

- $R=\{r_1,r_2,\ldots,r_n\}$为要进行排列的n个元素集合
- $R_i=R-\{r_i\}$
- 集合 R 中元素的全排列记为 $perm(R)$
- $(r_i)perm(R_i)$ 表示在全排列 $perm(R_i)$ 的每一个排列前加上前缀 r_i 得到的排列。

$$R=\{1,2,3\}$$

$$R_1=R-\{1\}=\{2,3\}$$

排列问题定义

- R的全排列可归纳定义如下：
  - 当$n=1$时，$perm(R)=(r)$，r是集合R中唯一的元素
  - 当$n>1$时，$perm(R)$由以下构成：

$$(r_1)perm(R_1),(r_2)perm(R_2),\ldots,(r_n)perm(R_n)$$

$R=\{1,2,3\}$

$R_1=\{2,3\}\quad R_2=\{1,3\}\quad R_3=\{1,2\}$

$perm(R) \Rightarrow (1)perm(R_1),(2)perm(R_2),(3)perm(R_3)$

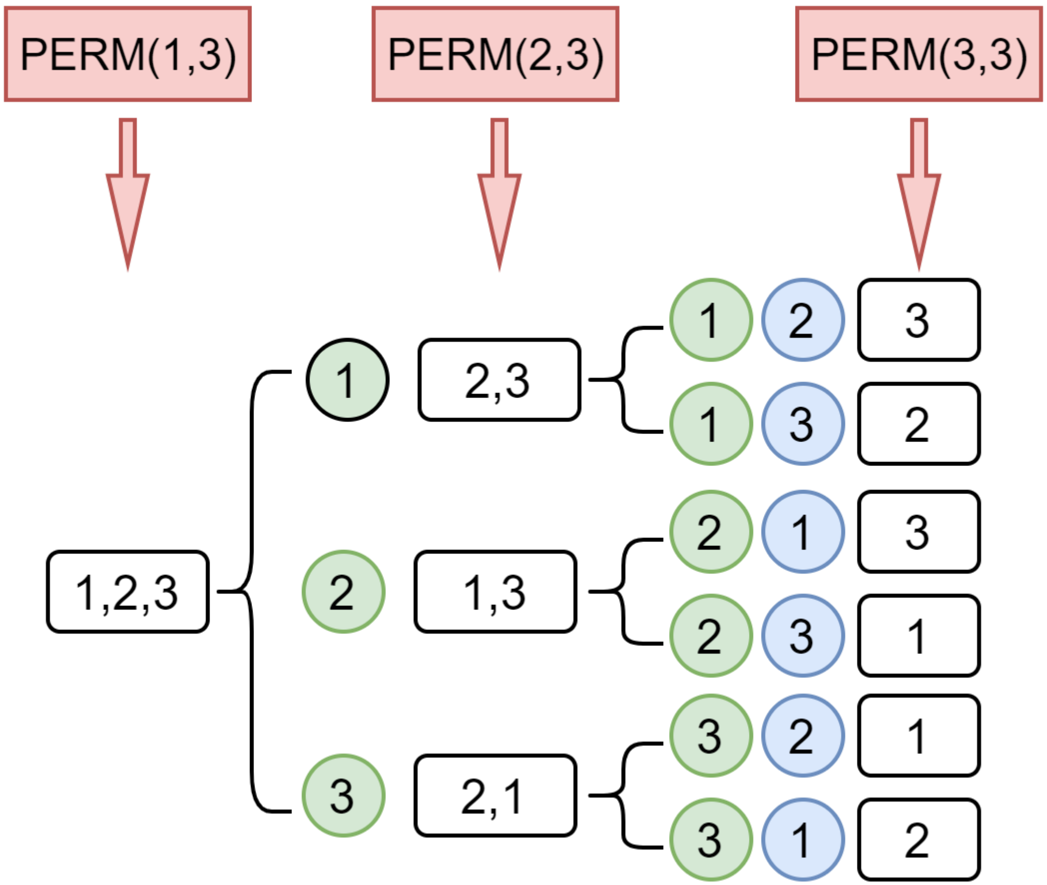
排列问题算法

```
PERM(A, first, last)
  if first==last
    PRINT A
  else
    for i=first to last
      SWAP(A[first], A[i])
      PERM(A, first+1, last)
      SWAP(A[first], A[i])
```

- PERM(A, first, last)递归地产生所有前缀是A[1..first-1], 后缀是A[first..last]的所有排列。
- 调用PERM(A, 1, n)则产生A[1..n]的全排列。

排列问题算法图示

```
PERM(A, first, last)
  if first==last
    PRINT A
  else
    for i=first to last
      SWAP(A[first], A[i])
      PERM(A, first+1, last)
      SWAP(A[first], A[i])
```



2.2 递归

1. 阶乘函数
2. Fibonacci数列
3. 排列问题
4. 整数划分问题
5. Hanoi塔问题

4 整数划分问题

- 将正整数n表示成一系列正整数之和：

$$n = n_1 + n_2 + \cdots + n_k$$

其中 $n_1 \geqslant n_2 \geqslant \cdots \geqslant n_k \geqslant 1, k \geqslant 1$

- 正整数n的这种表示称为正整数n的划分。
- 整数划分问题：求正整数n的不同的划分个数p(n)。

例：正整数6的划分

6

5+1

4+2, 4+1+1

3+3, 3+2+1, 3+1+1+1

2+2+2, 2+2+1+1, 2+1+1+1+1

1+1+1+1+1+1

分析

- 前面的几个例子中，问题本身都具有比较明显的递归关系，因而容易用递归函数直接求解。
- 在本例中，如果设**p(n)**为正整数n的划分数，则难以找到递归关系，因此考虑增加一个自变量：
  - 将最大加数n_1不大于m的划分个数记作**q(n,m)**

递归关系

1. **q(n,1)=1,n≥1**
   - 当最大加数n_1不大于1时，任何正整数n只有一种划分形式，$n = \overbrace{1+1+\cdots+1}^{n}$
2. **q(n,m)=q(n,n),m≥n**
   - 最大加数n_1实际上不能大于n
   - q(1,m)=1

递归关系

3. **$q(n,n)=1+q(n,n-1)$**

- 正整数n的划分由$n_1=n$的划分和$n_1 \le n-1$的划分组成。
- 例：$q(6,6)=1+q(6,6-1)=1+q(6,5)$

6

5+1

4+2, 4+1+1

3+3, 3+2+1, 3+1+1+1

2+2+2, 2+2+1+1, 2+1+1+1+1

1+1+1+1+1+1

递归关系

4. **$q(n,m)=q(n-m,m)+q(n,m-1), n>m>1$**

- 正整数n的最大加数n_1不大于m的划分由$n_1=m$的划分和$n_1\le m-1$ 的划分组成。
- 例：$q(6,4)=q(6-4,4)+q(6,4-1)=q(2,4)+q(6,3)$

$\boxed{4+2,\ 4+1+1}$

3+3, 3+2+1, 3+1+1+1
2+2+2, 2+2+1+1, 2+1+1+1+1
1+1+1+1+1+1

递归式

$$q(n,m)=\begin{cases}1 & \text{if } n=1, m=1\\ q(n,n) & \text{if } n<m\\ 1+q(n,n-1) & \text{if } n=m\\ q(n,m-1)+q(n-m,m) & \text{if } n>m>1\end{cases}$$

正整数n的划分数p(n)=q(n,n)。

算法描述

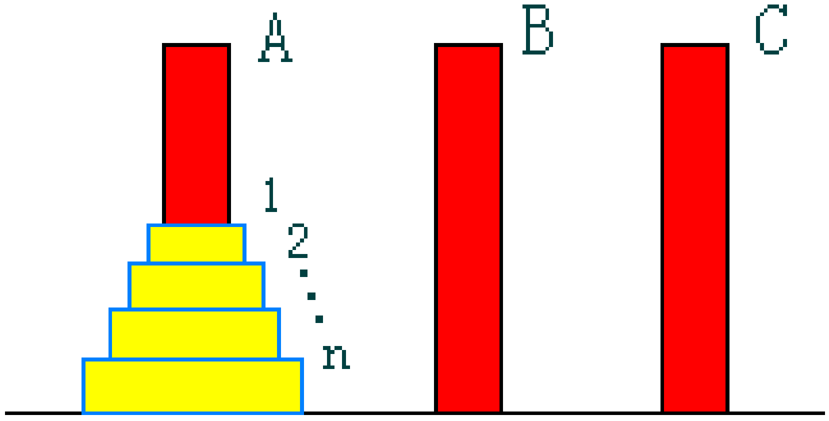
对于正整数n，求最大数不大于m的划分个数：

```
Q(n, m)
   if n<1 or m<1
      return 0
   if n==1 or m==1
      return 1
   if n<m
      return Q(n, n)
   if n==m
      return Q(n,m-1)+1
   return Q(n, m-1) + Q(n-m, m)
```

2.2 递归

1. 阶乘函数
2. Fibonacci数列
3. 排列问题
4. 整数划分问题
5. Hanoi塔问题

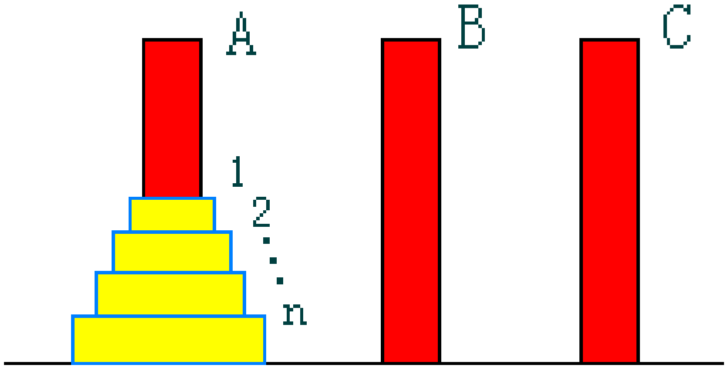
5 Hanoi塔问题



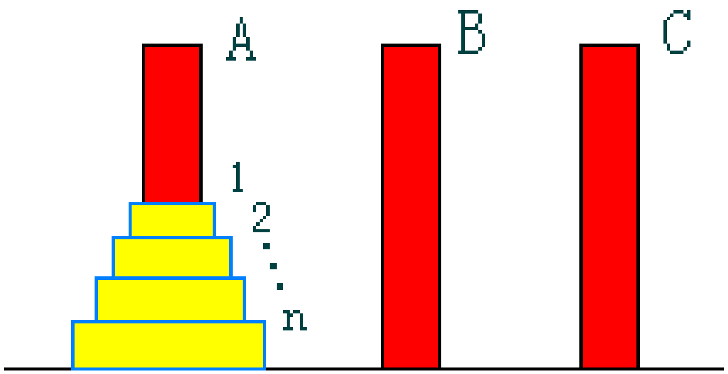
- 设A,B,C是3个塔座。开始时，在塔座A上有一叠共n个圆盘，这些圆盘自下而上，由大到小地叠在一起。各圆盘从小到大编号为1,2,…,n,现要求将塔座A上的这一叠圆盘移到塔座B上，并仍按同样顺序叠置。在移动圆盘时应遵守以下移动规则：
  - 规则1：每次只能移动1个圆盘；
  - 规则2：任何时刻都不允许将较大的圆盘压在较小的圆盘之上；
  - 规则3：在满足移动规则1和2的前提下，可将圆盘移至A,B,C中任一塔座上。

5 Hanoi塔问题

- 在问题规模较大时，较难找到好的方法，因此我们尝试用递归技术来解决这个问题。



5 Hanoi塔问题



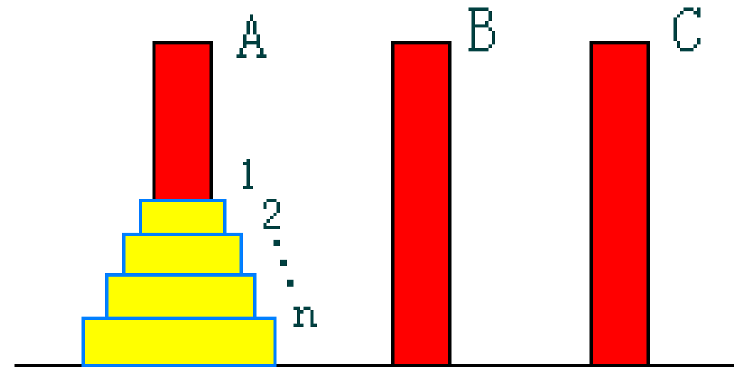
- 当n=1时，问题比较简单。此时，只要将编号为1的圆盘从塔座A直接移至塔座B上即可。
- 当n＞1时，需要利用塔座C作为辅助塔座。
  1. 此时若能设法将n-1个较小的圆盘依照移动规则从塔座A移至塔座C
  2. 然后，将剩下的最大圆盘从塔座A移至塔座B
  3. 最后，再设法将n-1个较小的圆盘依照移动规则从塔座C移至塔座B。
- 由此可见，n个圆盘的移动问题可分为2次n-1个圆盘的移动问题，这又可以递归地用上述方法来做。

5 Hanoi塔问题

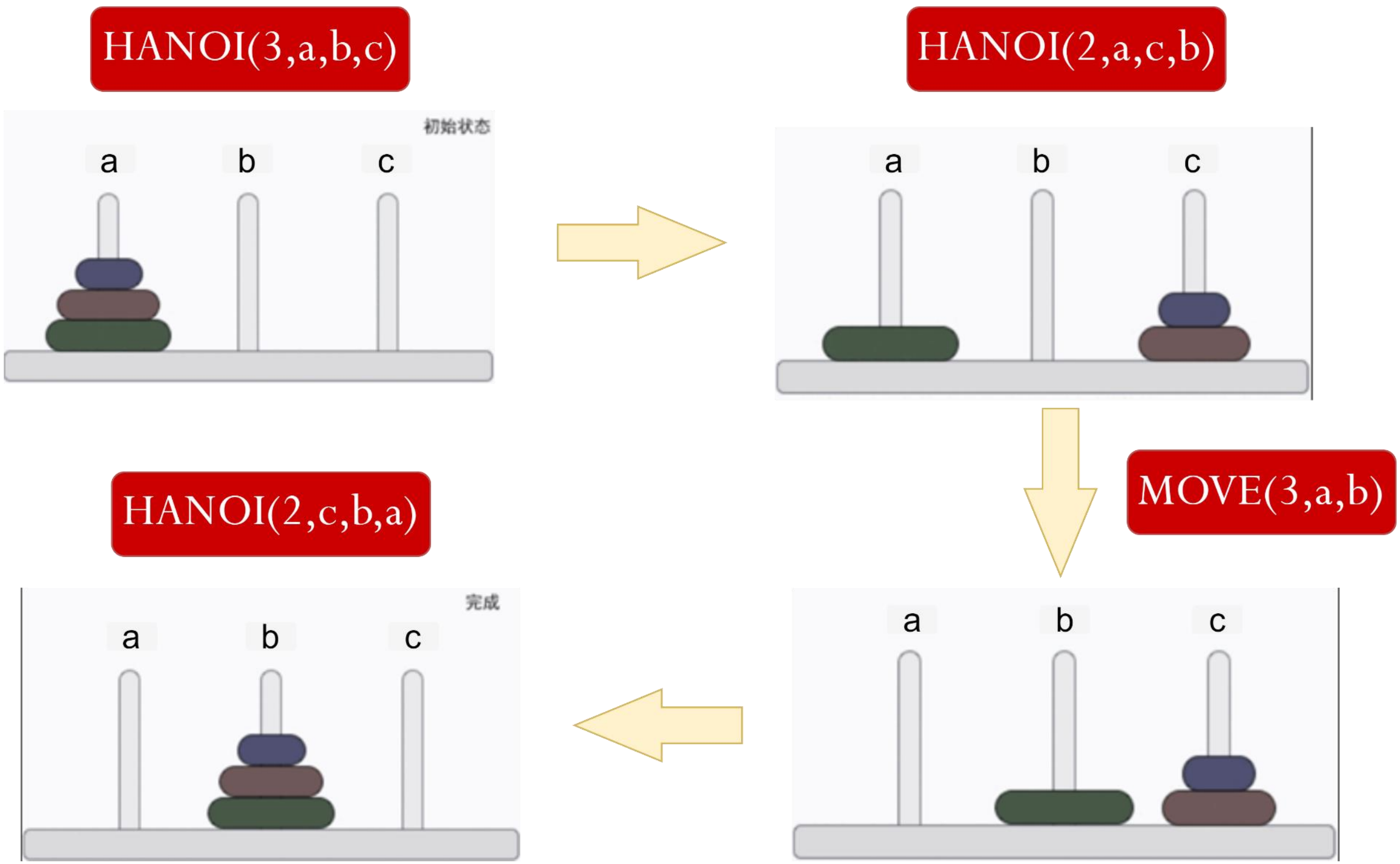
通过c将n个圆盘从a移至b

解Hanoi塔问题的递归算法:

```
HANOI(n, a, b, c)
   if n==1
      MOVE(n, a, b)
   else
      HANOI(n-1, a, c, b)
      MOVE(n, a, b)
      HANOI(n-1, c, b, a)
```



Hanoi塔问题算法图示



递归小结

- 优点：结构清晰，可读性强，而且容易用数学归纳法来证明算法的正确性，因此它为设计算法、调试程序带来很大方便。
- 缺点：递归算法的运行效率较低，无论是耗费的计算时间还是占用的存储空间都比非递归算法要多。

内容

2.1 求解递归式

2.2 递归

1. 阶乘函数
2. Fibonacci数列
3. 排列问题
4. 整数划分问题
5. Hanoi塔问题

2.3 分治

1. 二分搜索技术
2. 合并排序
3. 快速排序
4. 线性时间选择

End