

算法设计与分析

Youtube频道: [hwdong](https://www.youtube.com/hwdong)

博客: hwdong-net.github.io

算法设计与分析

- 算法(algorithm):

什么是算法、算法解决问题的步骤、算法有什么用?

- 算法的设计(Design):

如何设计算法?

- 算法的分析(analysis):

分析算法的性能: 时间、空间、正确性

什么是算法？

算法定义、算法的设计过程、算法的作用

算法的定义

- 算法是解决一类问题或某个计算的过程（方法），算法包含有限步可行的明确的操作。
- 问题：2个整数相乘

$$\begin{array}{r} 4265 \\ 3718 \\ \hline 34120 \\ 4265 \\ 29855 \\ 12795 \\ \hline 15857270 \end{array}$$

算法的定义

问题：长 a ，宽 b 的矩形的周长是多少？

$$2*(a+b)$$

问题：求一组数 (a_1, a_2, \dots, a_n) 中的最大值。

$\text{max} = a_1$

for $i=2$ to n :

if $a_i > \text{max}$:

$\text{max} = a_i$

return max

算法解决问题的过程

- 问题：求一组数的最大值。
- 问题的理解：一组数、最大值的概念
- 数学建模：用 (a_1, a_2, \dots, a_n) 表示一组数，最大值 \max 必须满足：

$$\max \geq a_i \quad 1 \leq i \leq n$$

- 设计算法：

$\max = a_1$

for $i=2$ to n :

← $n-1$

if $a_i > \max$:

$\max = a_i$

return \max

- 分析算法性能：时间、空间、正确性

算法有什么用？

1. 是计算机学科的主干：每个计算机科学分支都以算法为核心。

❖ operating systems and compiles: 进程调度、词法分析
networking: 如路由算法、搜索引擎

❖ machine learning and AI: 各种机器学习算法如神经网络层、随机森林、支持向量机、智能算法

❖ cryptography: 密码算法、数论算法

❖ computational biology: 同源性分析（序列比对）

❖ computer Graphics: 计算几何、光照渲染、流体仿真、动画、

算法有什么用？

2. 算法实际应用

- ❖ 大数据处理要求计算速度。计算速度取决于硬件和算法。
- ❖ 深度学习/现代人工智能：人脸识别、下棋程序、自动驾驶等
- ❖ 电子商务平台：推荐算法。
- ❖ 自媒体：信息喂料，算法决定思维、洗脑、舆论。

算法有什么用？

3. 算法有趣

❖ 创造性的数学活动，有趣也有挑战，有挑战才激动人心。如 a^n 、大整数乘法。

$$\begin{array}{r} 4265 \\ 3718 \end{array} \longrightarrow \begin{array}{r} 42 \quad 65 \\ 37 \quad 18 \end{array}$$

$$65 * 18 = 1170 \quad 42 * 37 = 1554$$

$$42 * 18 + 37 * 65 = 756 + 2405 = 3161$$

$$\begin{array}{r} 1170 \\ 316100 \\ 15540000 \\ \hline 15857270 \end{array}$$

算法的设计

如何设计高效的、正确的算法？

- 技术（设计模式）、艺术（创造性思维）

$$\begin{array}{r}
 4265 \\
 3718 \\
 \hline
 34120 \\
 4265 \\
 29855 \\
 12795 \\
 \hline
 15857270
 \end{array}$$

$$\begin{array}{r}
 65 * 18 = 1170 \quad 42 * 37 = 1554 \\
 42 * 18 + 37 * 65 = 756 + 2405 = 3161 \\
 1170 \\
 316100 \\
 15540000 \\
 \hline
 15857270
 \end{array}$$

分治法

- 技术（设计模式）、艺术（创造性思维）

$$1+2+3+\dots+100 = (\dots((1+2)+3)+\dots+100) = 5050 \quad \text{迭代法}$$

$$1+2+3+\dots+100 = (1+100)+(2+99)+\dots+(50+51) = 101*50 = 5050$$

算法设计的策略（模式、技术）

- ❖ 穷举法：考虑所有可能情况。
- ❖ 分治法：分而治之，将大问题分解成小问题。
- ❖ 动态规划：避免重复子问题的重复计算。
- ❖ 贪婪法：选择当前局部最优的策略。

❖**穷举法**: 列举所有可能情况/元素。


求最大值: 和每个数去比较。

28	3	21	69	17
----	---	----	----	----

❖穷举法：列举所有可能情况/元素。

求最大值：和每个数去比较。

28	3	21	69	17
----	---	----	----	----




$\max = a_1$

❖穷举法：列举所有可能情况/元素。

求最大值：和每个数去比较。

28	3	21	69	17
----	---	----	----	----




$\max = a_1$

❖穷举法：列举所有可能情况/元素。

求最大值：和每个数去比较。

28	3	21	69	17
----	---	----	----	----




$\max = a_1$

❖穷举法：列举所有可能情况/元素。

求最大值：和每个数去比较。

28	3	21	69	17
----	---	----	----	----




$\max = a_1$

❖穷举法：列举所有可能情况/元素。

求最大值：和每个数去比较。

28	3	21	69	17
----	---	----	----	----



$\max = a_4$

❖穷举法：列举所有可能情况/元素。

求最大值：和每个数去比较。

28	3	21	69	17
----	---	----	----	----



$\max = a_4$

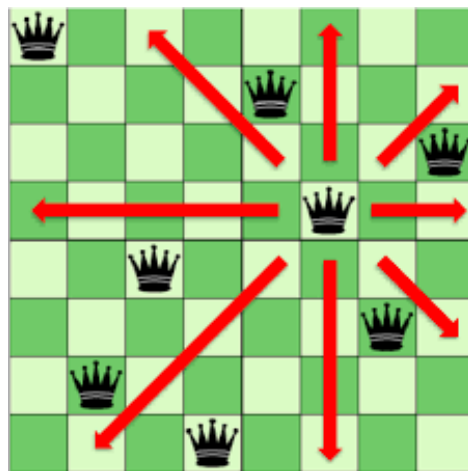
❖ 穷举法：列举所有可能情况/元素。

求最大值：和每个数去比较。

八皇后问题：

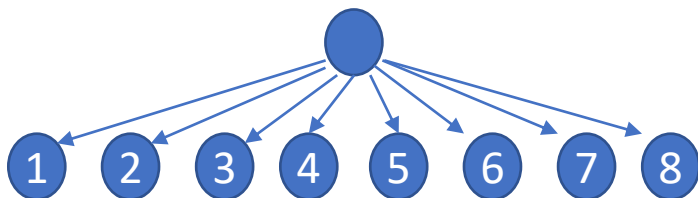
8个皇后分别在每一行，
不能同行、同列、同对角线

线性穷举（迭代法）

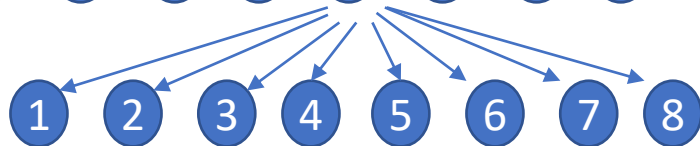


树型穷举（树型搜索）

第1行皇后



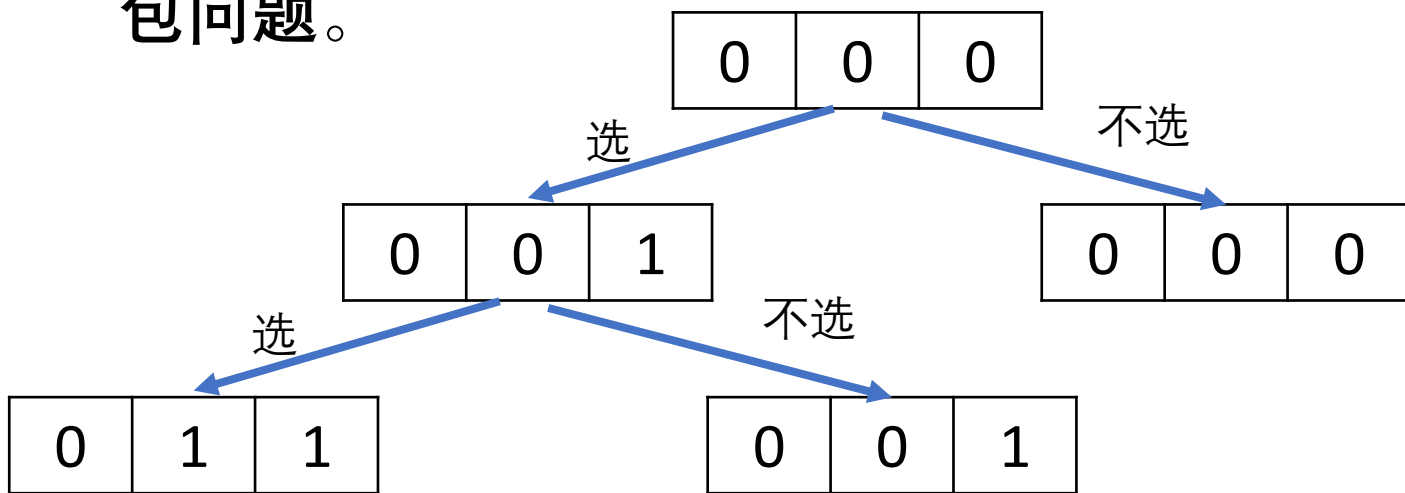
第2行皇后



搜索树

背包问题 (Knapsack problem) : 给定一组物品, 每种物品都有自己的重量和价格, 背包有限定的总重量, 如何选择, 使装入背包物品的总价格最高。

如果限定每种物品只能选择0个或1个, 则问题称为**0-1背包问题**。



❖分治法: 大问题分解成小问题 (Divid) , 解决小问题 (Conqour)、组合小问题解为大问题的解(Combine)

1) $n! = n * (n-1)!$

2) 斐波拉契数列:

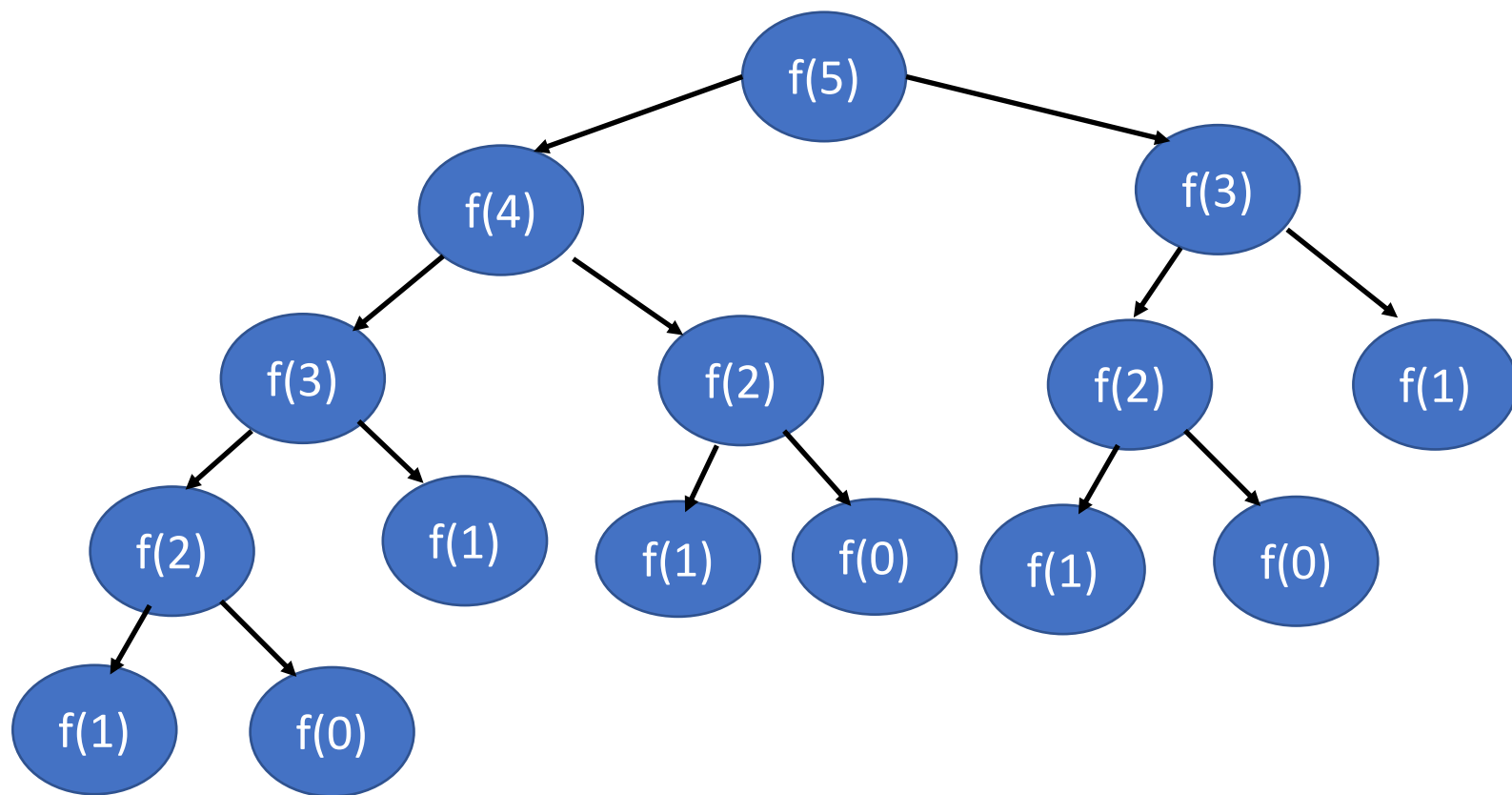
0、1、1、2、3、5、8、13、21、34、...

$$F_0 = 0$$

$$F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2} \quad (n \geq 2)$$

斐波拉契数列



3) 大整数相乘: $ab * cd = a * c * 10^n + (a * d + b * c) 10^{n/2} + b * d$

4) $a^n = a^{n/2} * a^{n/2}$

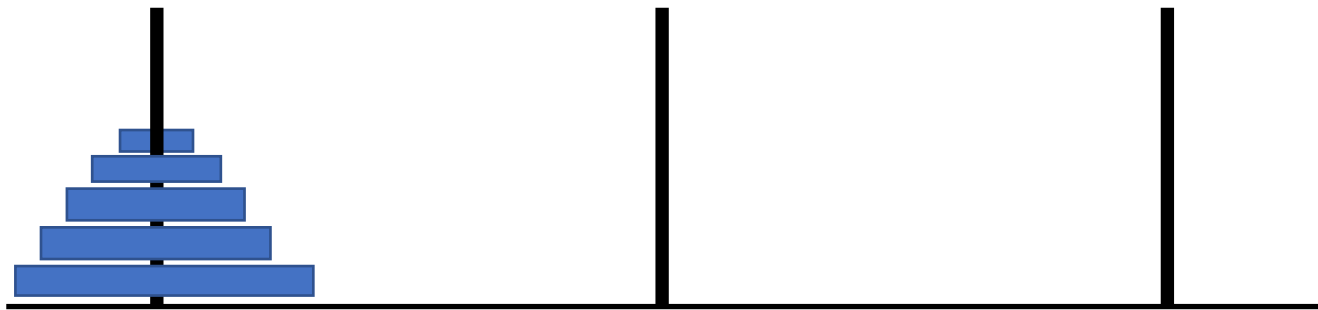
5) 选择排序、冒泡排序、快速排序、归并排序

$a_1 \ a_2 \ a_3 \ \dots \ a_n$

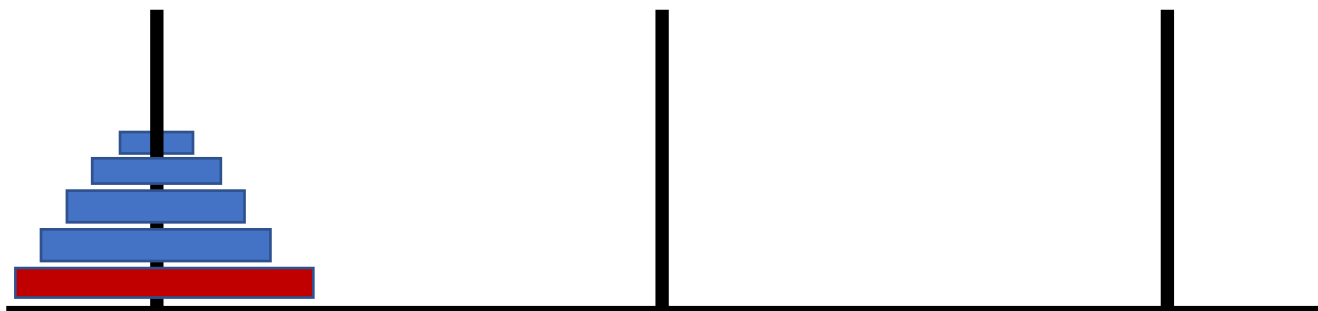
$b_1 \ a_2' \ a_3' \ \dots \ a_n'$

$b_1 \ b_2 \ a_3'' \ \dots \ a_n''$

6) 汉诺塔 n 个盘子移动问题



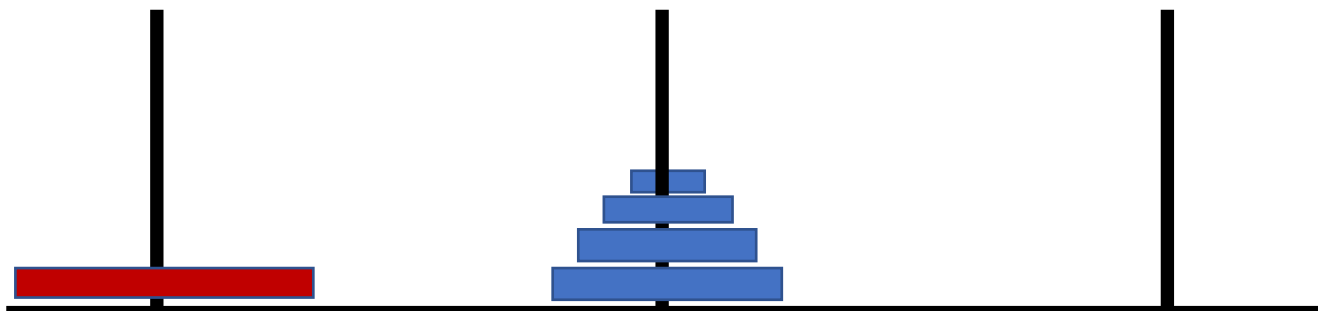
6) 汉诺塔 n 个盘子移动问题：转化为最下面盘子和上面的 $n-1$ 个盘子移动问题



6) 汉诺塔

n 个盘子移动问题：转化为最下面盘子和上面的 $n-1$ 个盘子移动问题

归结为更小问题： **$n-1$ 个盘子移动问题**

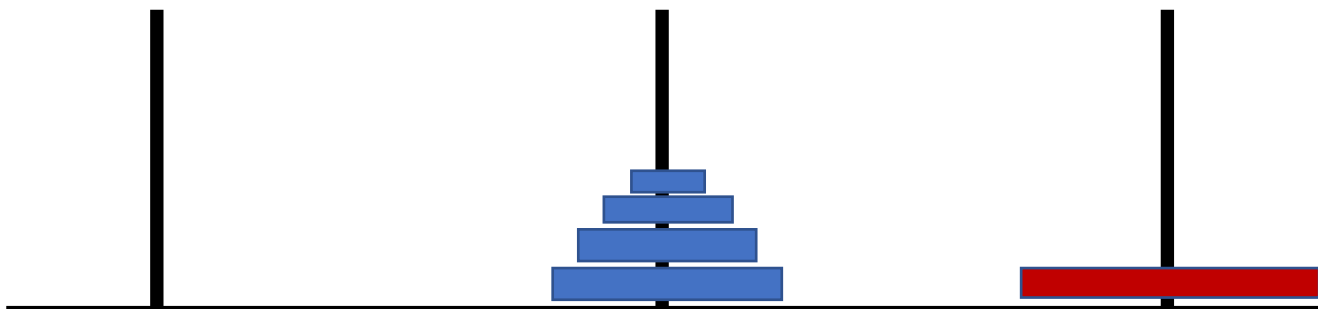


6) 汉诺塔

n 个盘子移动问题：转化为最下面盘子和上面的 $n-1$ 个盘子移动问题

归结为更小问题： $n-1$ 个盘子移动问题

直接移动最下面盘子



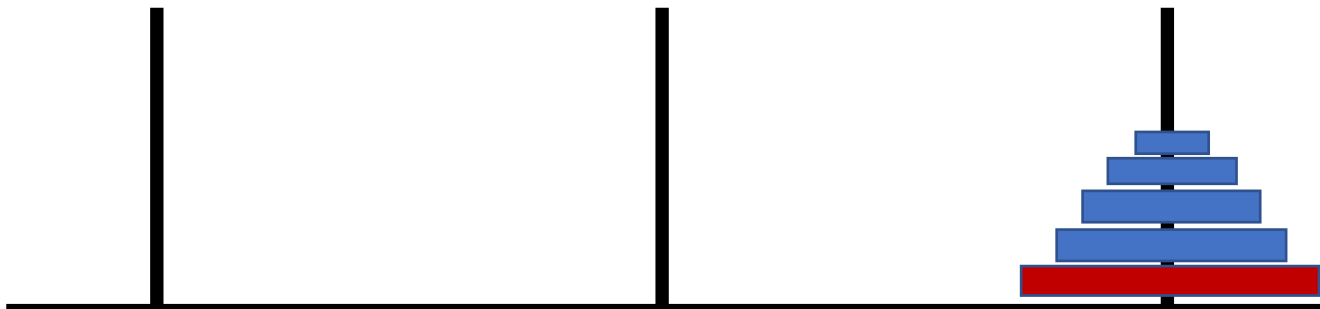
6) 汉诺塔

n 个盘子移动问题：转化为最下面盘子和上面的 $n-1$ 个盘子移动问题

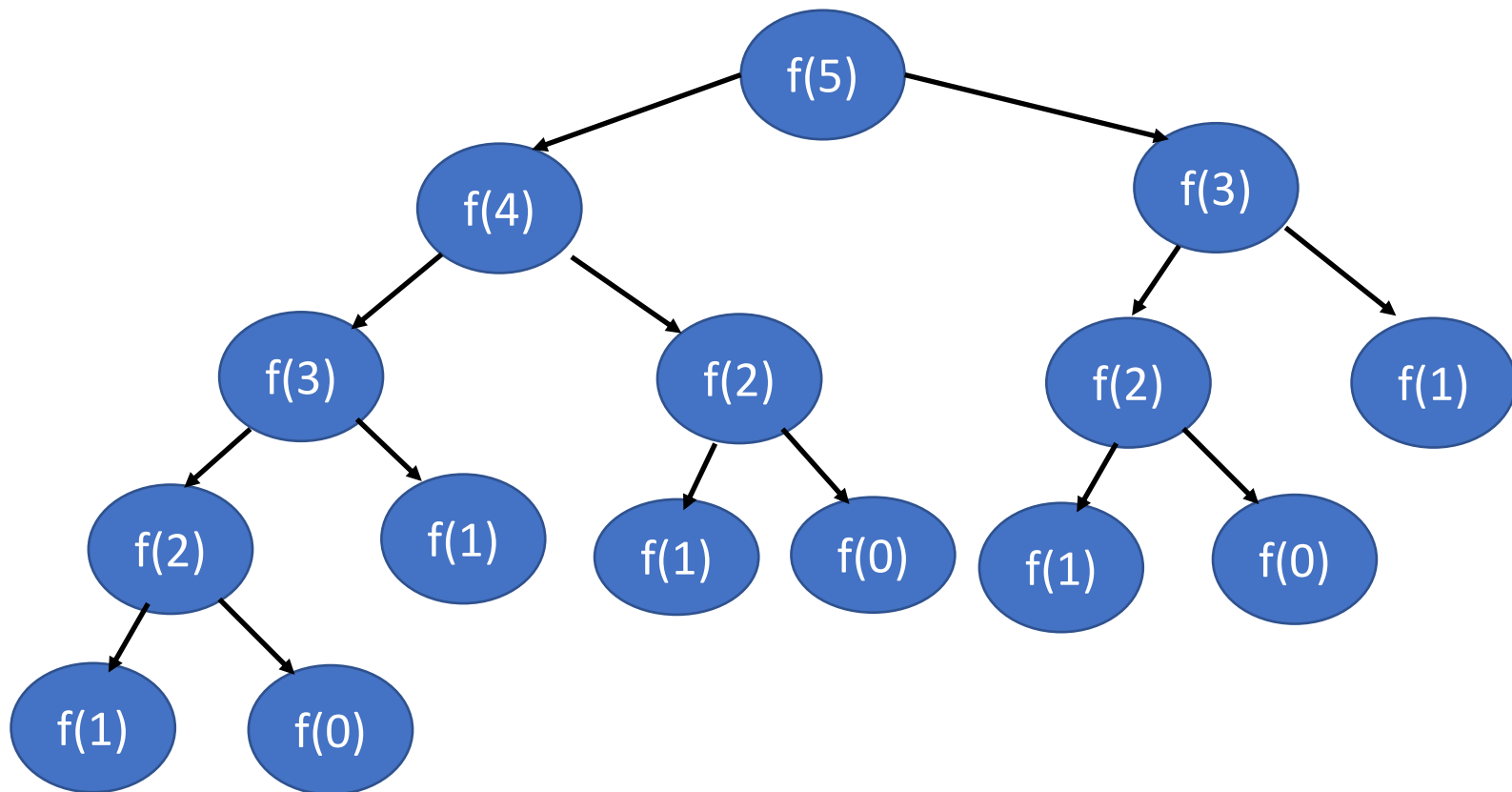
归结为更小问题： $n-1$ 个盘子移动问题

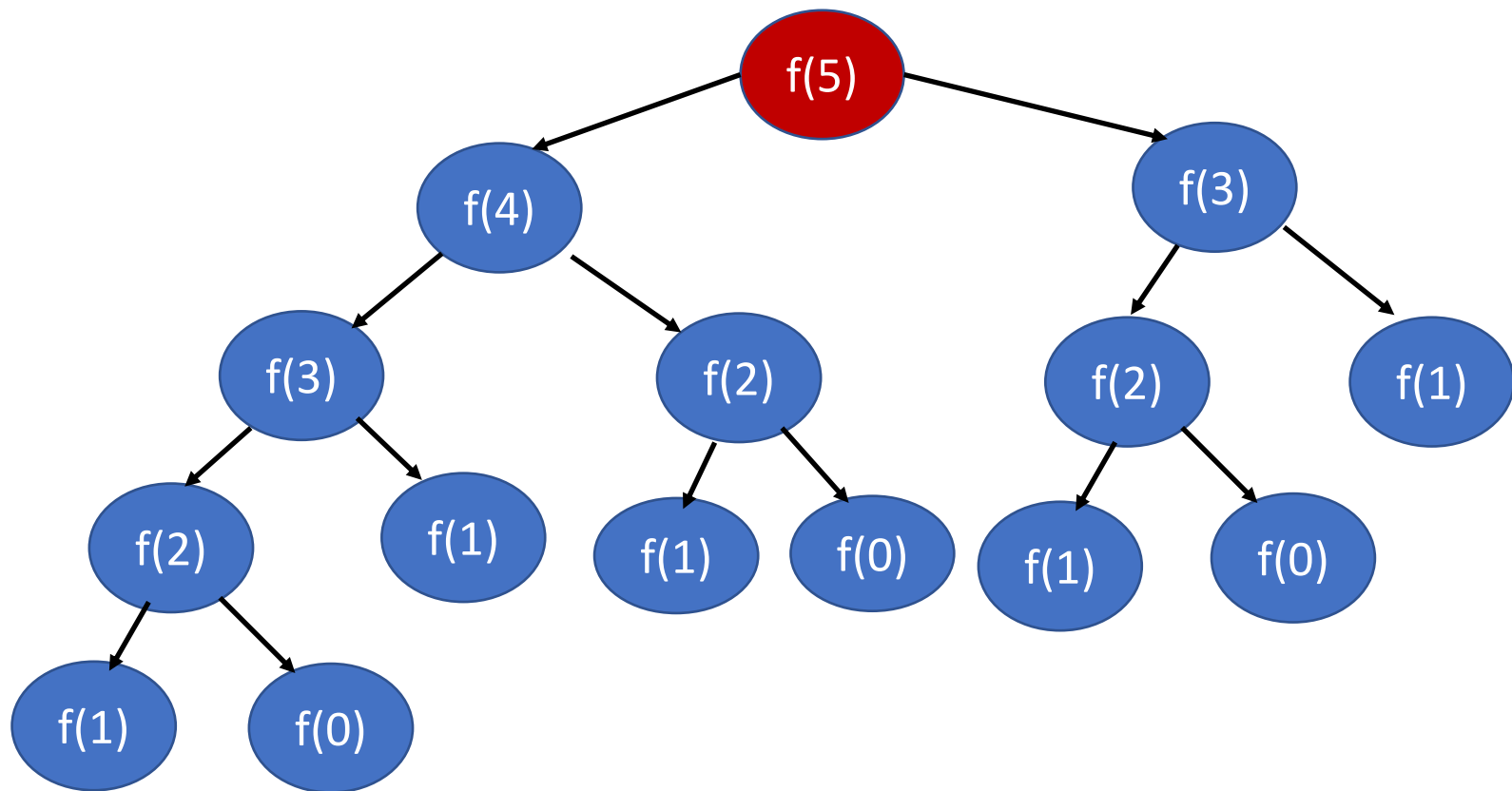
直接移动最下面盘子

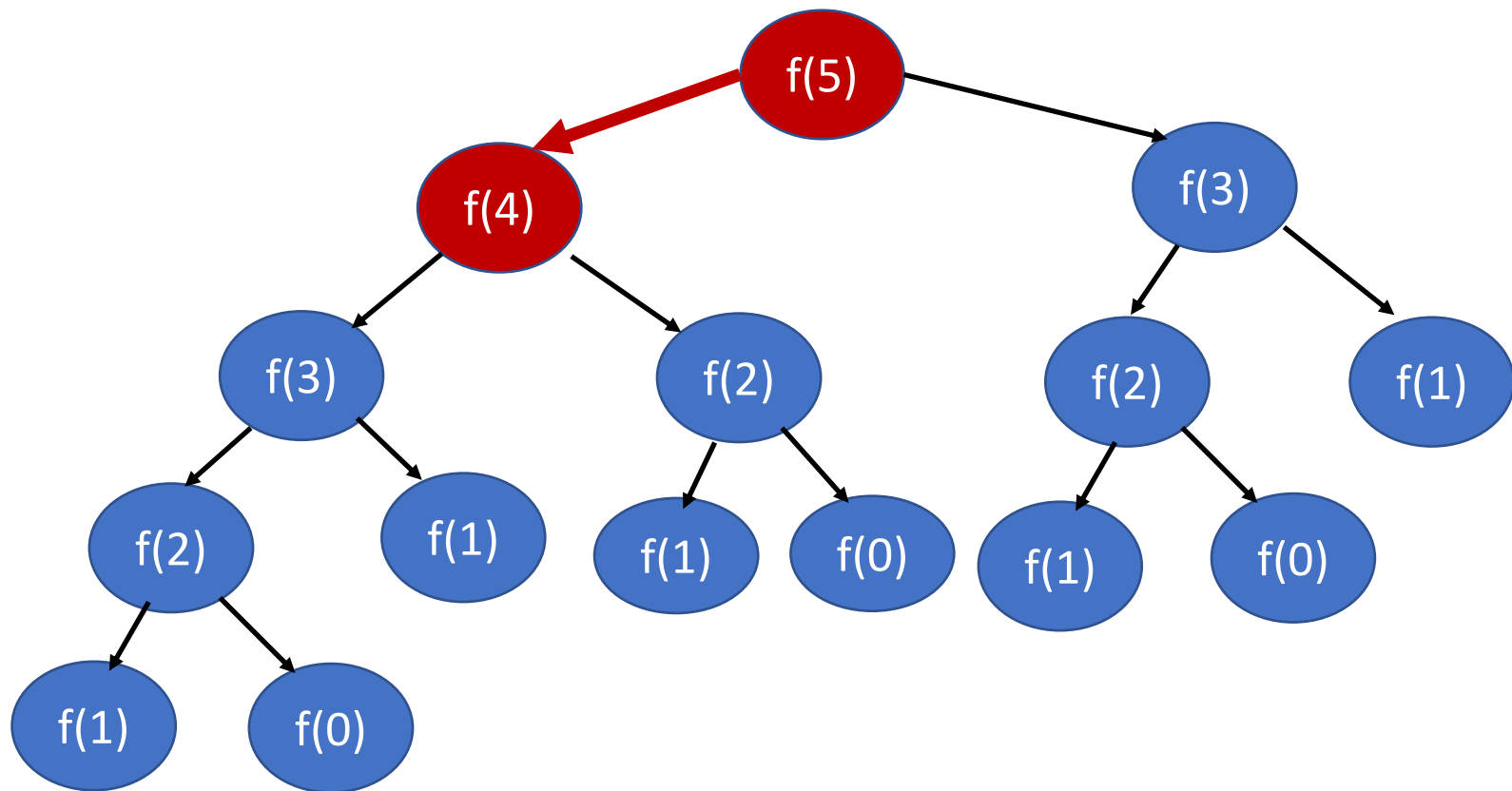
$n-1$ 个盘子移动问题

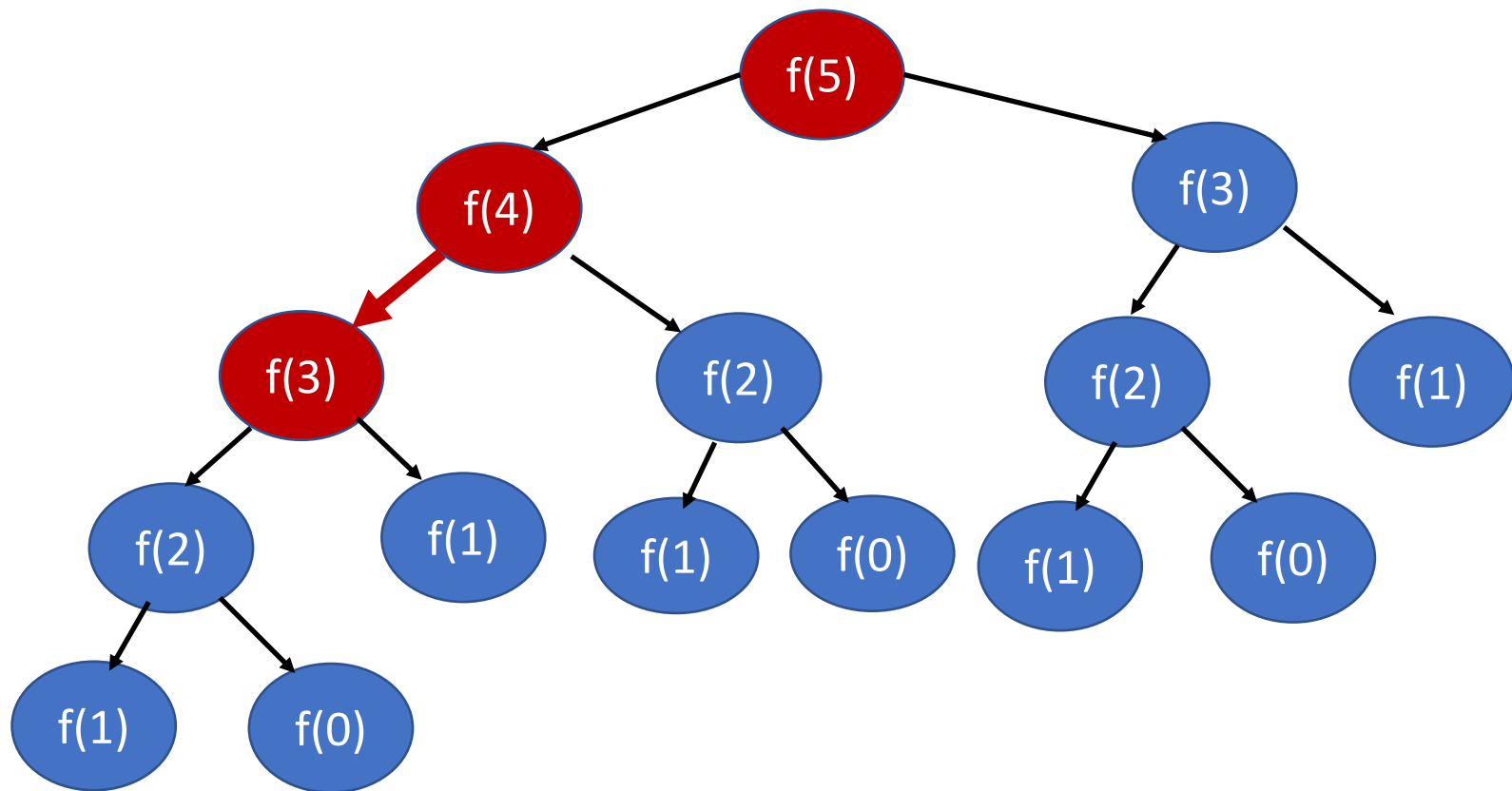


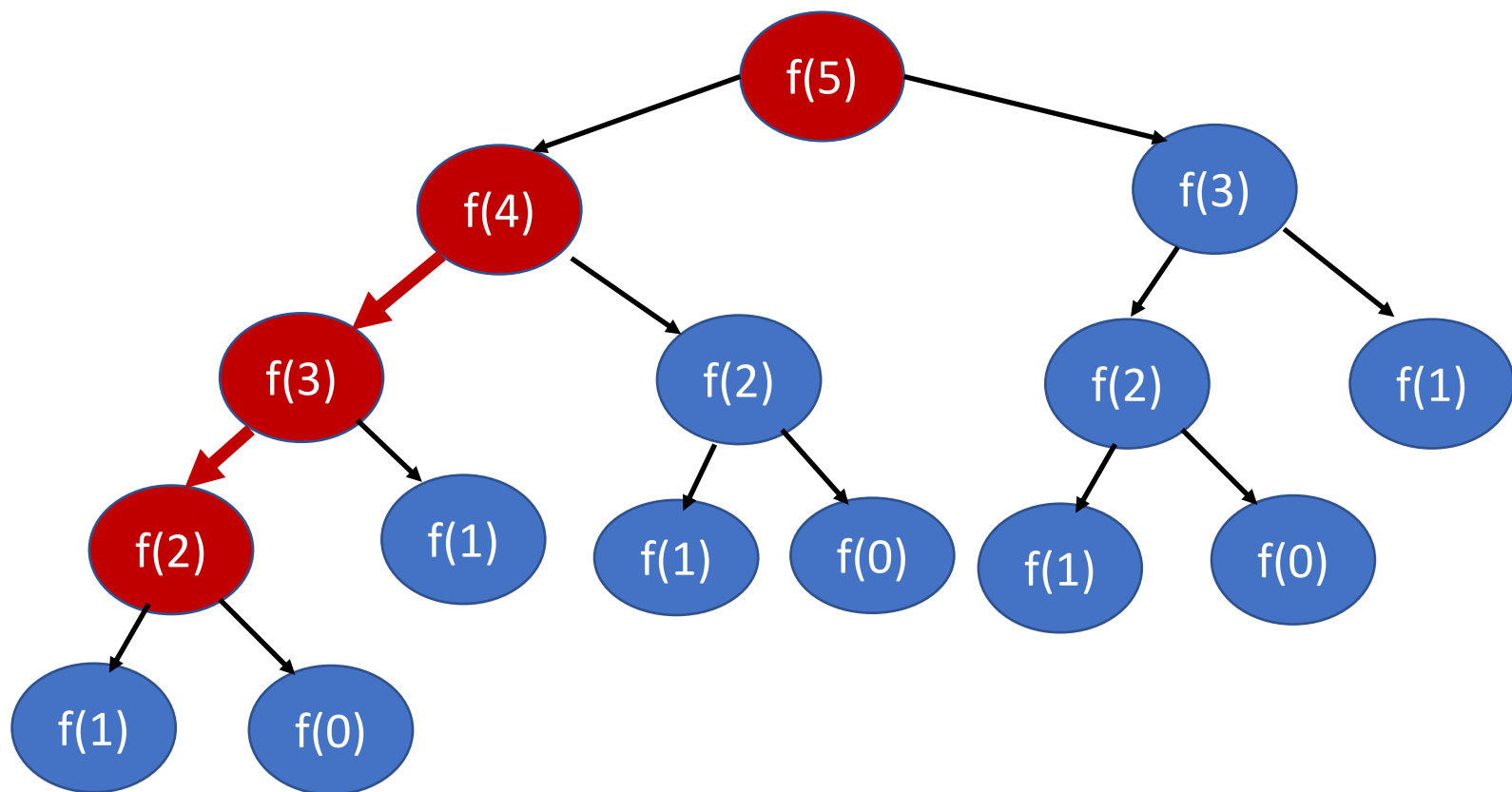
❖ 动态规划：避免重复子问题的计算

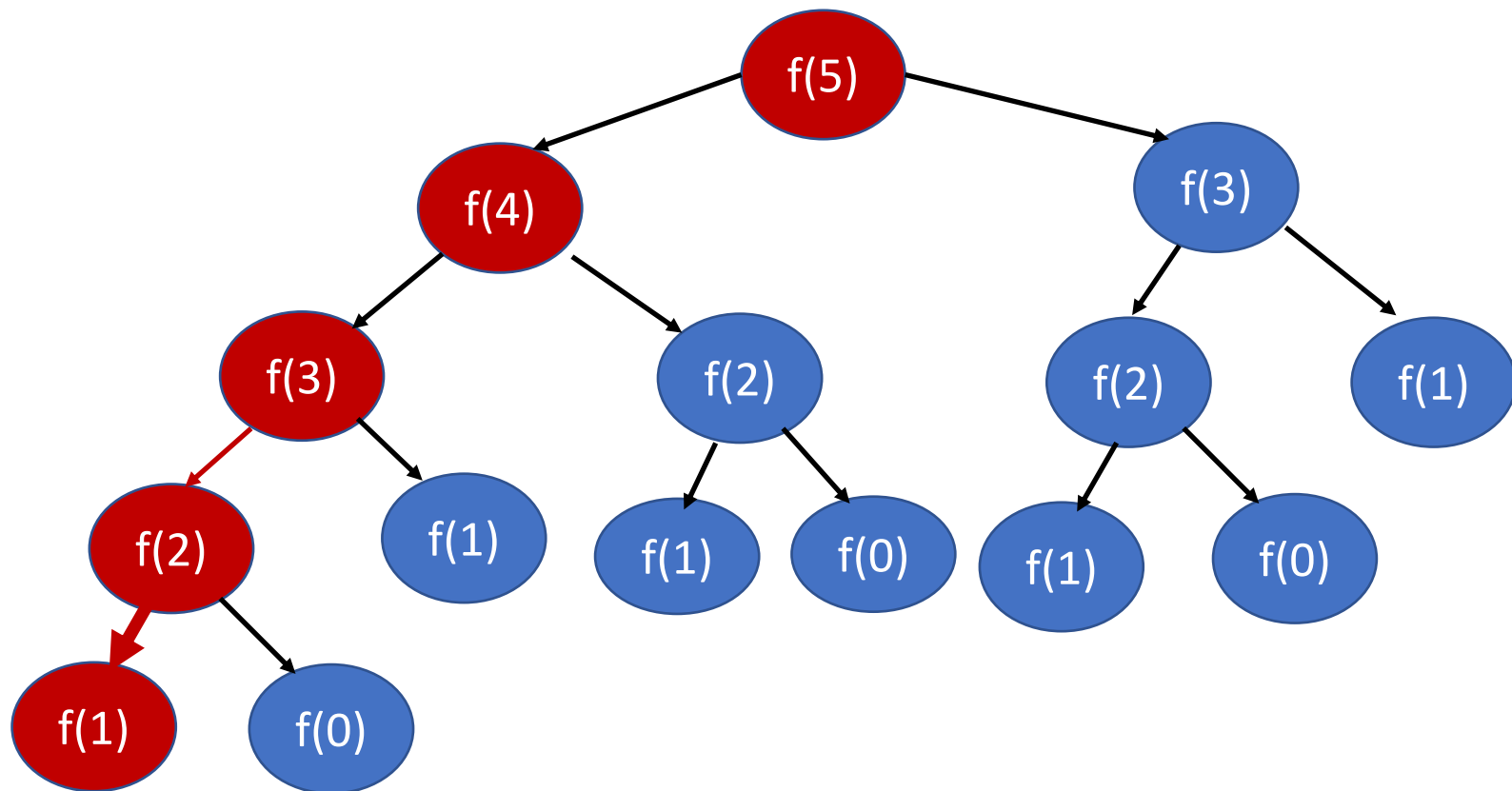


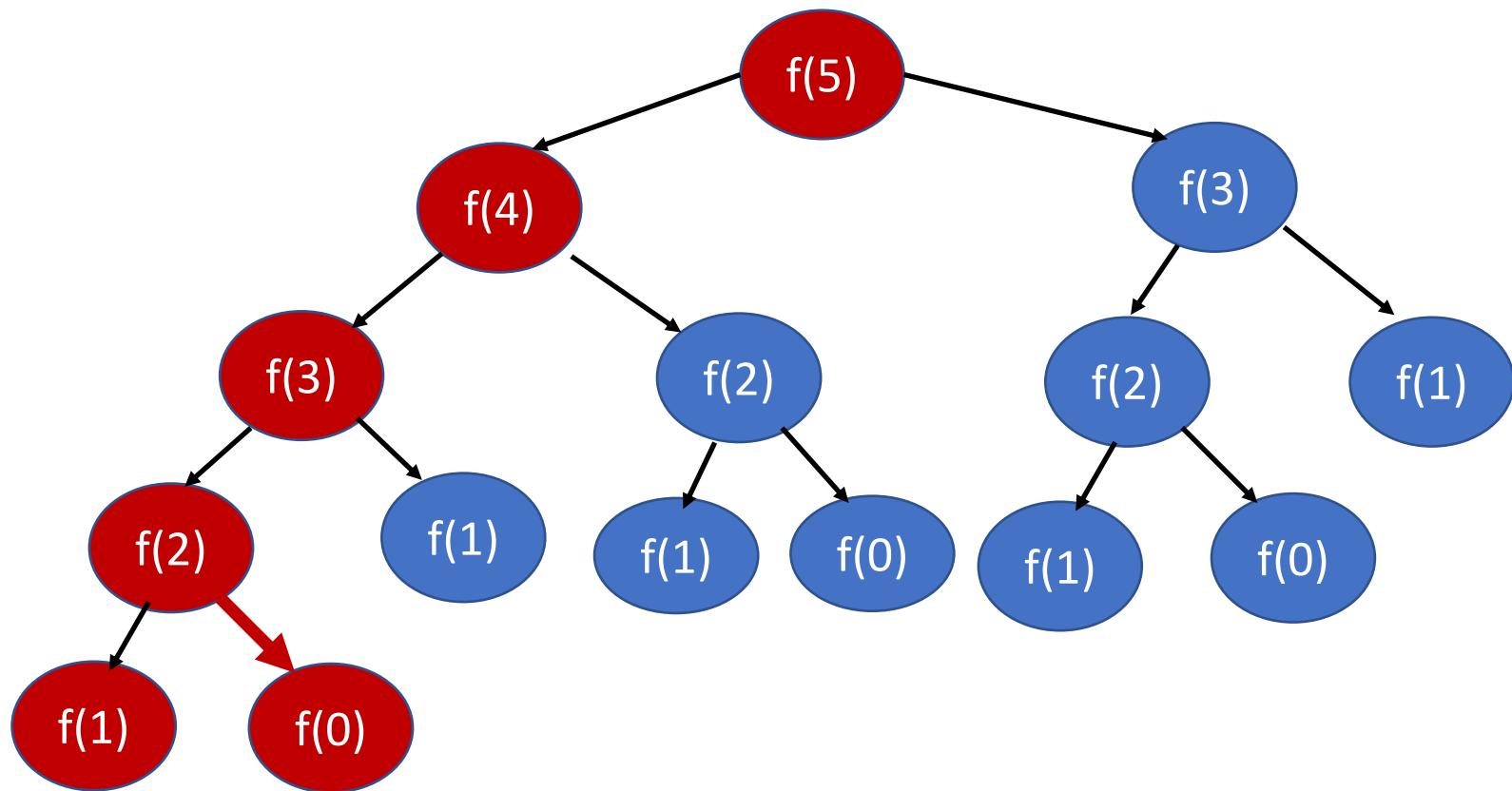


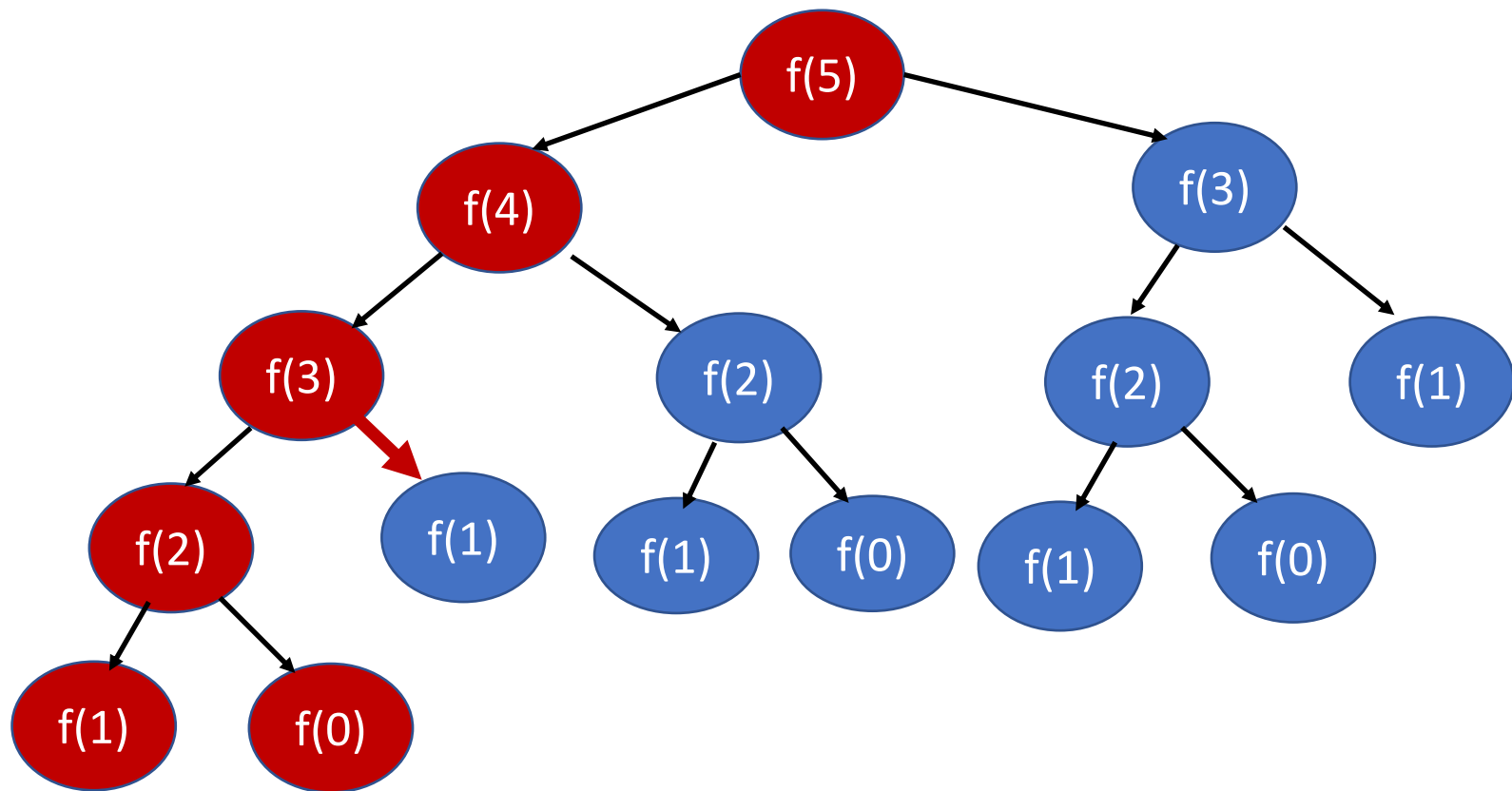


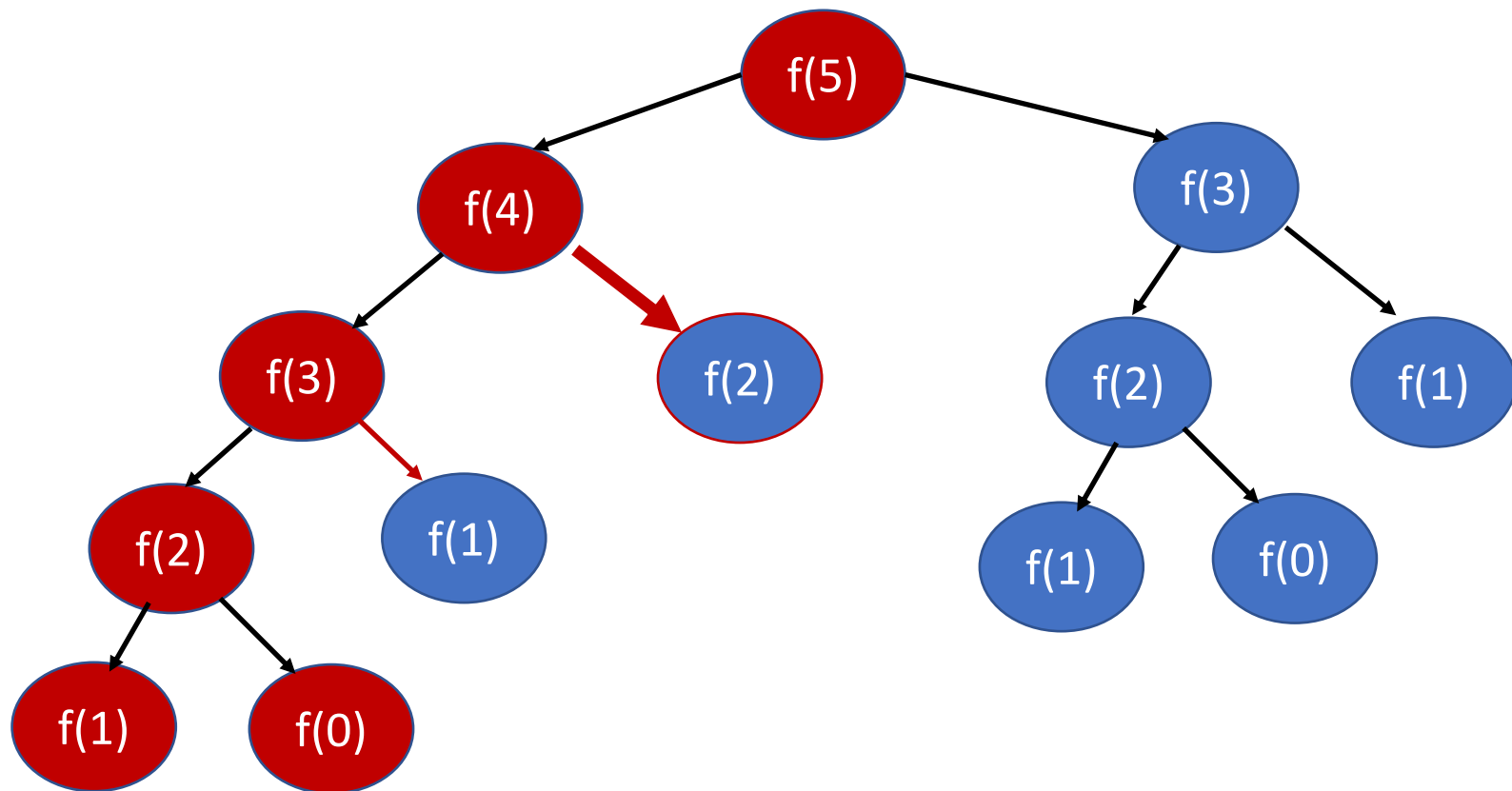


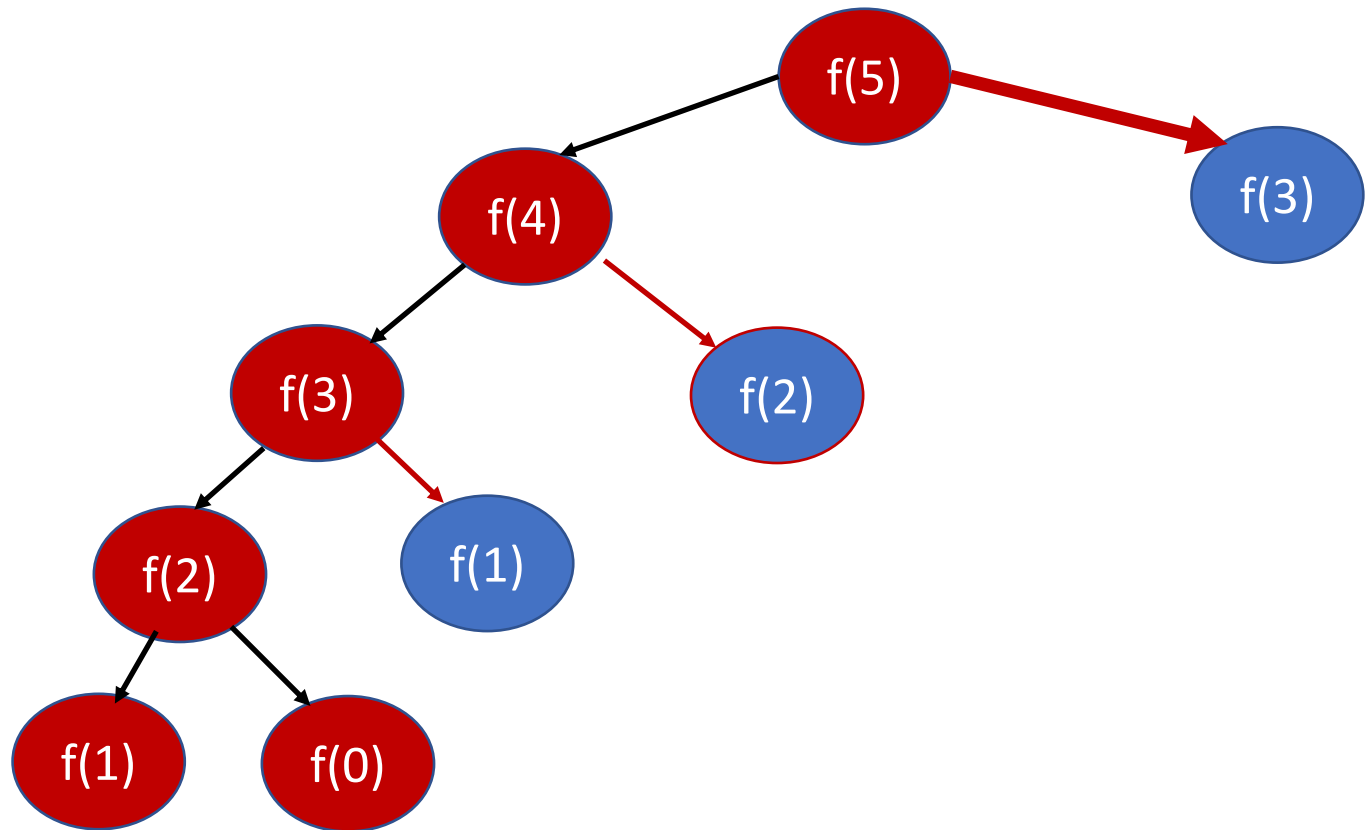




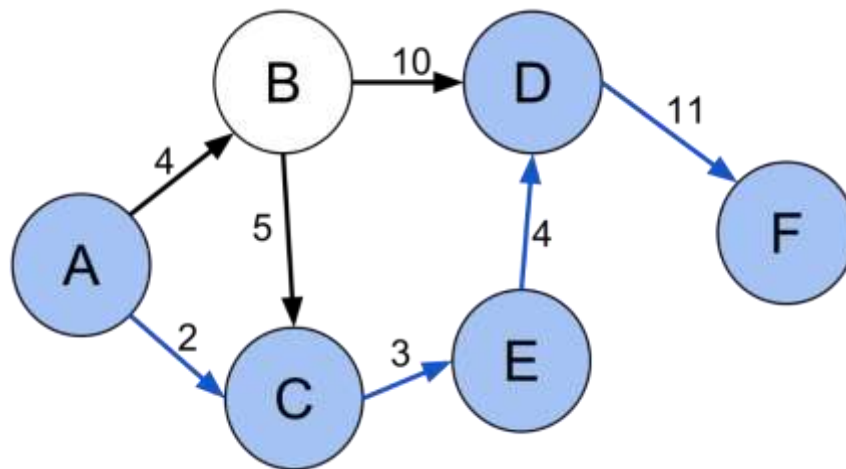








❖ 贪婪法：图的Dijkstra最短路径算法



❖贪婪法：找零钱

如果硬币的面值是 c_0, c_1, \dots, c_k ，求找价值 M 所需要的最少硬币数。

如硬币面值为 $\langle 1, 2, 5, 10 \rangle$ ，那么如果要找33块钱的话，最少的零钱数是 $10+10+10+2+1$ ，一共5个硬币。

算法的表示

- ❖ 自然语言: 用人类语言描述。
- ❖ 伪代码: 介于自然语言和编程语言之间。
- ❖ 编程语言: 用编程语言实现算法

- 问题：求一组数的最大值
- 自然语言的算法表示：

假设第一个数是最大值；

剩余的每个数和当前最大值比较：

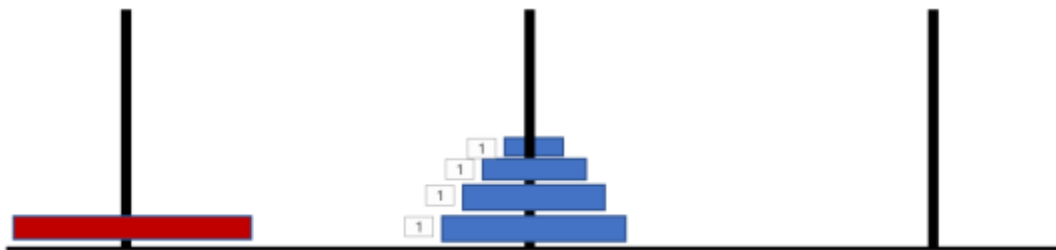
如果比当前最大值还要大，则更新最大值为这个数。

- 问题：求一组数的最大值
- 伪代码：

```
max =  $a_1$   
for i=2 to n:  
    if  $a_i > \text{max}$ :  
        max =  $a_i$   
return max
```

Hanoi汉诺塔的分治法

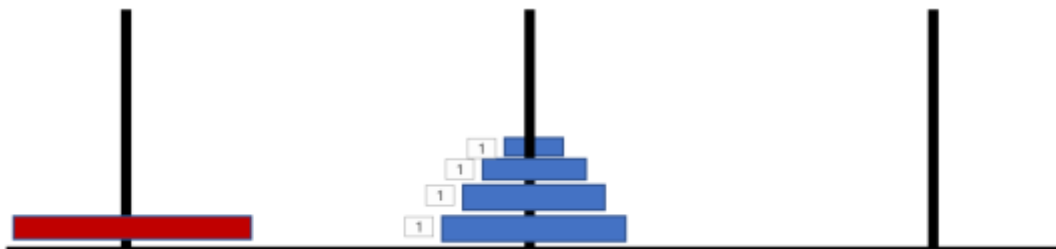
- 问题： n 个盘子借助于B柱从A柱移到C柱子
- 自然语言描述：



如果 n 为1， 直接从A柱移到C柱， 否则
将A柱上面的 $n-1$ 个盘子借助于C柱从A柱移到B柱
直接将最下面的 n 号盘子从A柱移到C柱
将B柱上面的 $n-1$ 个盘子借助于A柱从B柱移到C柱

Hanoi汉诺塔的分治法

- 问题： n 个盘子借助于B柱从A柱移到C柱子
- 伪代码：



```
Hanoi(n,A,B,C)
```

```
    if n==1:    move(n, A,C)
```

```
    Hanoi(n-1,A,C,B)
```

```
    move(n, A,C)
```

```
    Hanoi(n-1,B,A,C)
```

算法的分析

分析算法的性能：时间、空间、正确性

同一问题可以有不同算法

- 求长 a 宽 b 的矩形周长：

算法1: $2*(a+b)$

算法2: $2*a+2*b$

哪个效率更高（速度快）？

同一问题可以有不同算法

大整数相乘问题：

$$\begin{array}{r} 4265 \\ 3718 \\ \hline 34120 \\ 4265 \\ \hline 29855 \\ 12795 \\ \hline 15857270 \end{array}$$

$$65 * 18 = 1170 \quad 42 * 37 = 1554$$

$$42 * 18 + 37 * 65 = 756 + 2405 = 3161$$

$$\begin{array}{r} 1170 \\ 316100 \\ 15540000 \\ \hline 15857270 \end{array}$$

哪个效率更高（速度快）？

- 空间：除输入数据外的额外空间， $O(1)$

```
max =  $a_1$ 
```

```
for i=2 to n:
```

```
    if  $a_i > \text{max}$ :
```

```
        max =  $a_i$ 
```

```
return max
```

- **空间：** 除输入数据外的额外空间， $O(1)$
- **时间：** 算法执行的时间

事后分析： 编写程序， 在某机器上运行
事前分析



C/C++ , Java,Python

- 事先分析：选择基本操作（Flops），统计基本操作执行的次数。
- 基本操作执行次数，称为**频度**。
- 依赖问题规模 n ,

频度可表示为 n 的函数

```
max = a1
for i=2 to n:
    if ai > max:
        max = ai
return max
```

- 插入排序:

18	49	25	97	76	13	27
----	----	----	----	----	----	----

18	49	25	97	76	13	27
----	----	----	----	----	----	----

18	49	25	97	76	13	27
----	----	----	----	----	----	----

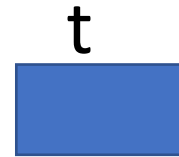
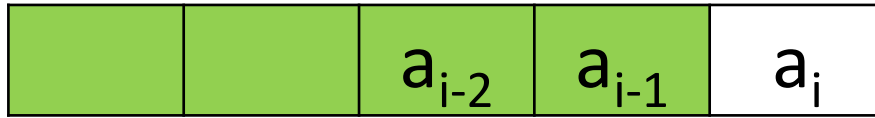
18	25	49	97	76	13	27
----	----	----	----	----	----	----

18	25	49	97	76	13	27
----	----	----	----	----	----	----

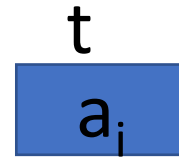
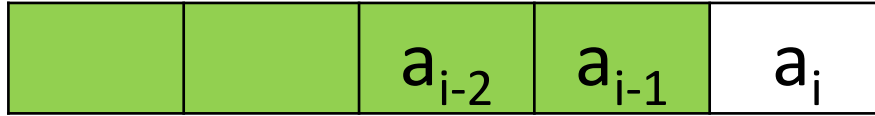
18	25	49	76	97	13	27
----	----	----	----	----	----	----

13	18	25	49	76	97	27
----	----	----	----	----	----	----

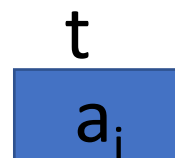
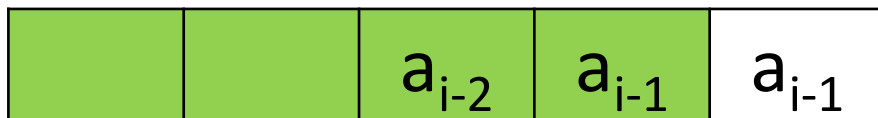
13	18	25	27	49	76	97
----	----	----	----	----	----	----



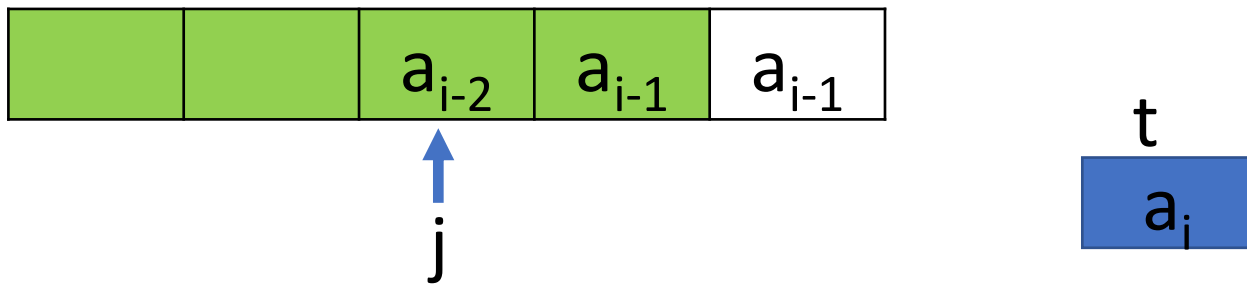
```
if a[i]<a[i-1]:
```



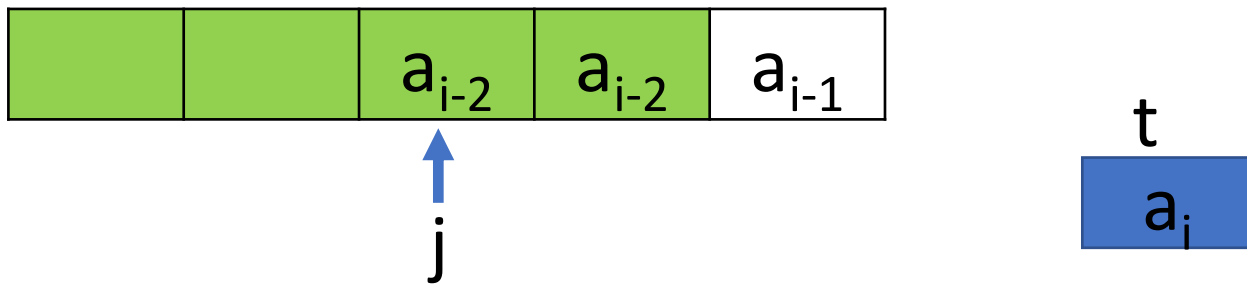
```
if a[i]<a[i-1]:  
    t  = a[i]
```

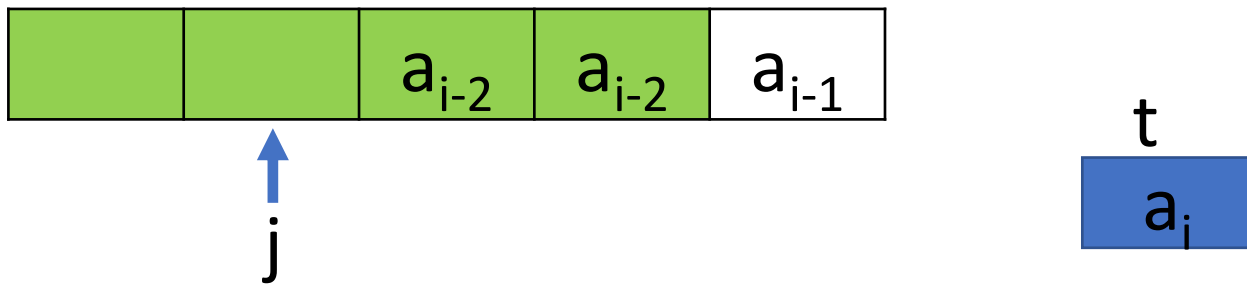
```
if a[i]<a[i-1]:  
    t  = a[i]  
    a[i] = a[i-1]  //后移
```



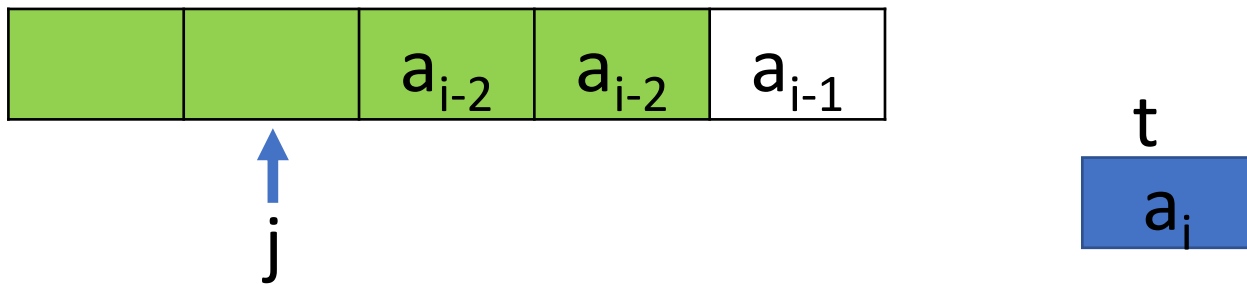
```
if a[i]<a[i-1]:  
    t = a[i]  
    a[i] = a[i-1]    //后移  
    j = i-2  
    while j>0 and t<a[j]:  
        a[j+1] = a[j];    //a[j]后移
```



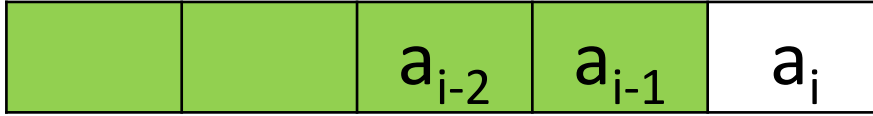
```
if a[i]<a[i-1]:  
    t = a[i]  
    a[i] = a[i-1]    //后移  
    j = i-2  
    while j>0 and t<a[j]:  
        a[j+1] = a[j];    //a[j]后移
```



```
if a[i]<a[i-1]:  
    t = a[i]  
    a[i] = a[i-1]    //后移  
    j = i-2  
    while j>0 and t<a[j]:  
        a[j+1] = a[j]    //a[j]后移  
        j = j-1
```



```
if a[i]<a[i-1]:  
    t = a[i]  
    a[i] = a[i-1]    //后移  
    j = i-2  
    while j>0 and t<a[j]:  
        a[j+1] = a[j]    //a[j]后移  
        j = j-1  
    a[j+1] = t
```



```
for i = 2 to n
  if a[i]<a[i-1]:
    t = a[i]
    a[i] = a[i-1]  //后移
    j = i-2
    while j>0 and t<a[j]:
      a[j+1] = a[j]  //a[j]后移
      j = j-1
    a[j+1] = t
```

- 时间复杂度分析:

```
for i = 2 to n
```

```
    if a[i]<a[i-1]:
```

```
        t = a[i]
```

```
        a[i] = a[i-1]    //后移
```

```
        j = i-2
```

```
        while j>0 and t<a[j]:
```

```
            a[j+1] = a[j]    //a[j]后移
```

```
            j = j-1
```

```
        a[j+1] = t
```

n

n-1

?

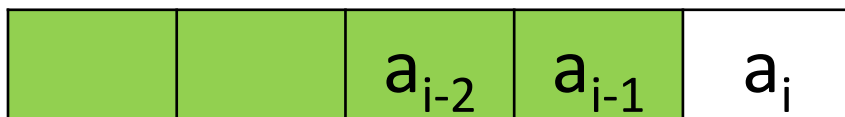
?

?

?

- 时间复杂度分析:

最好情况: 每次插入值比较1次, 赋值0次



1 2 3 4 5 6 7

1 2 3 4 5 6 7

1 2 3 4 5 6 7

1 2 3 4 5 6 7

1 2 3 4 5 6 7

1 2 3 4 5 6 7

1 2 3 4 5 6 7

```

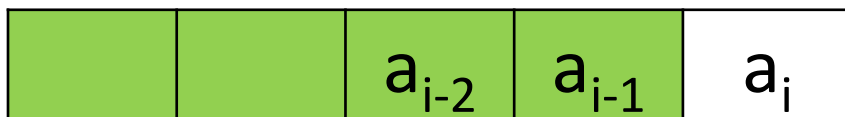
if a[i] < a[i-1]:
    t = a[i]
    a[i] = a[i-1]
    j = i-2
    while j > 0 and t < a[j]:
        a[j+1] = a[j]
        j = j-1
    a[j+1] = t

```

$$\sum_{i=2}^n 1 = (n-1)$$

- 时间复杂度分析:

最差情况: 每次插入值比较 $2i$ 次, 赋值 $4+2(i-1) : 4i+3$



7 6 5 4 3 2 1

6 7 5 4 3 2 1

5 6 7 4 3 2 1

4 5 6 7 3 2 1

3 4 5 6 7 2 7

2 3 4 5 6 7 1

1 2 3 4 5 6 7

```

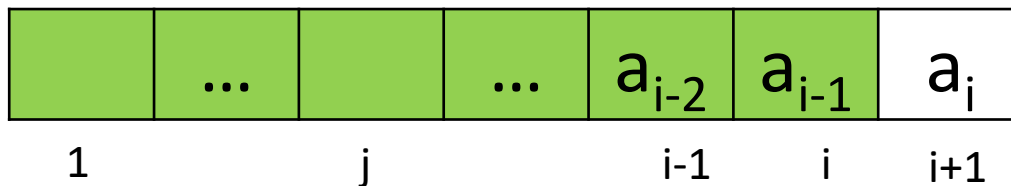
if a[i] < a[i-1]:
    t = a[i]
    a[i] = a[i-1]
    j = i-2
    while j > 0 and t < a[j]:
        a[j+1] = a[j]
        j = j-1
    a[j+1] = t

```

$$\sum_{i=2}^n (4i + 3) = (4n + 3 + 8 + 3)(n - 1)/2$$

- 时间复杂度分析:

平均情况: 基本操作: 元素的比较



$$\sum_{j=i}^1 \frac{i-j+1}{i} = \frac{1+2+\dots+i}{i} = (i+1)/2$$

$$\sum_{i=2}^n \frac{i+1}{2} = (n+4)(n-1)/4$$

- 时间复杂度分析:

基本操作执行次数表示为问题规模 n 的函数。如 $T(n) = n-1$ 。

最坏、平均情况下的时间复杂度

- 时间复杂度分析:

分治法的时间复杂度函数表示为一个递归式。

```
Hanoi(n,A,B,C)
  if n==1:    move(n, A,C)
  Hanoi(n-1,A,C,B)
  move(n, A,C)
  Hanoi(n-1,B,C,A)
```

$$T(n) = 2T(n-1)+1$$

$$T(1) = 1$$



$$T(n) = 2^n - 1$$

参考书：

- 屈婉玲：算法设计与分析，清华大学出版社

附：伪代码描述约定

- 赋值：

a = b

- 条件：

if exp:

...

if exp:

...

else:

...

if exp:

...

else if exp2:

...

else:

...

- 迭代:

for i = 2 to n:

...

while exp:

...

do{:

...

} **while** (exp)

- 注释:

// 代码注释

关注我

博客: hwdong.net

Youtube频道



hwdong

@hwdong · 5.01K subscribers · 558 videos

博客: <https://hwdong-net.github.io> >

youtube.com/c/4kRealSound and 4 more links