

# 1. 绪论

(xa) 局限

不学诗，何以言

不学礼，何以立

邓俊辉

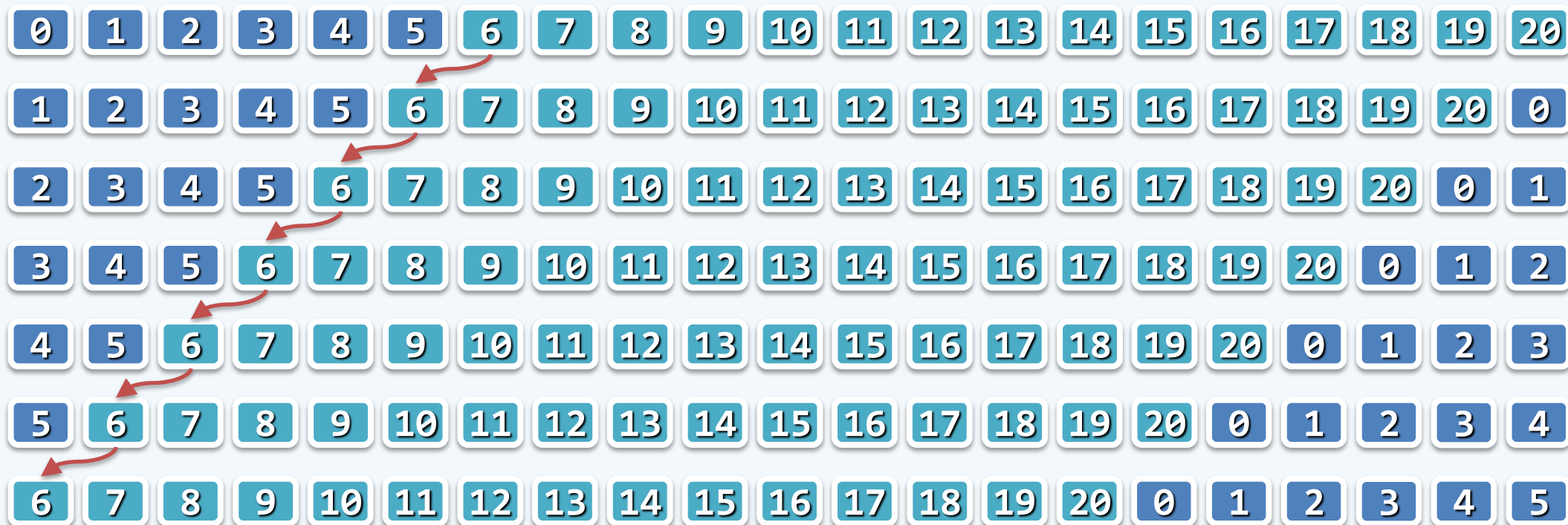
deng@tsinghua.edu.cn

## 循环移位：蛮力版

❖ 将数组A[0, n)中元素向左循环移动k个单元

❖ void shift0( int \* A, int n, int k ) //反复以1为间距循环左移

{ while ( k-- ) shift( A, n, 0, 1 ); } //共迭代k次,  $O(nk)$



## 循环移位：迭代版

```
❖ int shift( int * A, int n, int s, int k ) { //  $O(n / \text{GCD}(n, k))$   
    int b = A[s]; int i = s, j = (s + k) % n; int mov = 0; //mov记录移动次数  
    while ( s != j ) //从A[s]出发，以k为间隔，依次左移k位  
        { A[i] = A[j]; i = j; j = (j + k) % n; mov++; }  
    A[i] = b; return mov + 1; //最后，起始元素转入对应位置  
}
```



❖  $[0, n)$  由关于  $k$  的  $g = \text{GCD}(n, k)$  个 **同余类** 组成

//各含  $n/g$  个元素

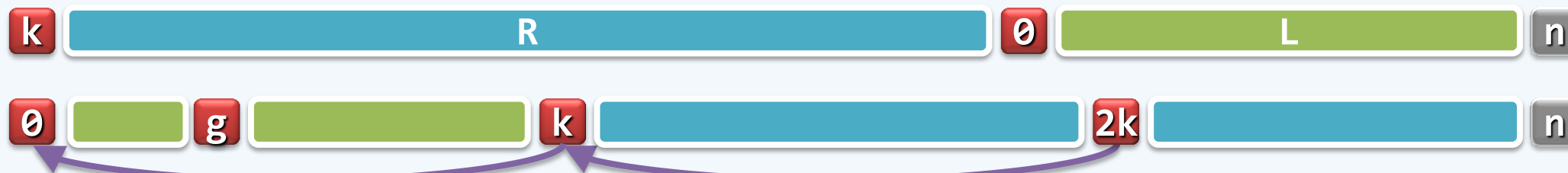
$\text{shift}(s, k)$  能够且只能够使其中之一就位

//即  $s$  所属的同余类

❖ 其它的同余类呢...

## 循环移位：迭代版

```
void shift1(int* A, int n, int k) { //经多轮迭代，实现数组循环左移k位，累计 $O(n)$   
    for (int s = 0, mov = 0; mov < n; s++) // $O(\text{GCD}(n, k)) = O(n*k/\text{LCM}(n, k))$   
        mov += shift(A, n, s, k);  
}
```



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
6	1	2	9	4	5	12	7	8	15	10	11	18	13	14	0	16	17	3	19	20
6	7	2	9	10	5	12	13	8	15	16	11	18	19	14	0	1	17	3	4	20
6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	0	1	2	3	4	5

## 循环移位：倒置版

```
void shift2( int * A, int n, int k ) { //借助倒置算法，将数组循环左移k位， $O(3n)$   
    reverse( A, k ); // $O(3k/2)$   
    reverse( A + k, n - k ); // $O(3(n-k)/2)$   
    reverse( A, n ); // $O(3n/2)$   
}
```



0 1 2 3 4 5 6 7 8

2 1 0 8 7 6 5 4 3

3 4 5 6 7 8 0 1 2

## 幂函数： $O(2^r)$

- ❖ 观察：同一问题的不同算法，复杂度不尽相同
- ❖ 问题：对任何给定的整数  $n > 0$ ，计算  $a^n$  ( $a$  为常数)
- ❖ 平凡实现：  

```
pow = 1; //O(1)  
while ( 0 < n ) { //O(n)  
    { pow *= a; n--; } //O(1) + O(1)
```
- ❖ 复杂度与  $n$  成正比， $T(n) = 1 + n \cdot 2 = O(n)$   
线性？伪线性！
- ❖ 所谓输入规模，准确地应定义为 用以描述输入所需的空间的规模  
对于此类数值计算，即是  $n$  的二进制位数  $r = \log_2(n + 1)$
- ❖ 复杂度与  $r$  成指数关系， $T(r) = O(2^r)$  **//太高了**

$$a^{98765}$$

$$= a^{[9 \times 10^4 + 8 \times 10^3 + 7 \times 10^2 + 6 \times 10^1 + 5 \times 10^0]}$$

$$= (a^{10^4})^9 \times (a^{10^3})^8 \times (a^{10^2})^7 \times (a^{10^1})^6 \times (a^{10^0})^5$$

$$\begin{aligned} &= \text{pow}(\text{pow}(a, 10^4), 9) = \text{pow}(\text{pow}(\text{pow}(a, 10^3), 10), 9) \\ &\times \text{pow}(\text{pow}(a, 10^3), 8) \quad \times \text{pow}(\text{pow}(\text{pow}(a, 10^2), 10), 8) \\ &\times \text{pow}(\text{pow}(a, 10^2), 7) \quad \times \text{pow}(\text{pow}(\text{pow}(a, 10^1), 10), 7) \\ &\times \text{pow}(\text{pow}(a, 10^1), 6) \quad \times \text{pow}(\text{pow}(\text{pow}(a, 10^0), 10), 6) \\ &\times \text{pow}(\text{pow}(a, 10^0), 5) \quad \times \text{pow}(\text{pow}(a, 10^0), 5) \end{aligned}$$

❖ 若能在 $\mathcal{O}(1)$ 时间内得到 $\text{pow}(x, n)$ ,  $0 \leq n \leq 10$

则自右（低）向左（高），每个数位只需 $\mathcal{O}(1)$ 时间

$a^{10110b}$

$$= a^{[1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0]}$$

$$= (a^{2^4})^1 \times (a^{2^3})^0 \times (a^{2^2})^1 \times (a^{2^1})^1 \times (a^{2^0})^0$$

$$\begin{aligned} &= \text{pow}(\text{pow}(a, 2^4), 1) &= \text{pow}(\text{sqr}(\text{pow}(a, 2^3)), 1) \\ &\times \text{pow}(\text{pow}(a, 2^3), 0) &\times \text{pow}(\text{sqr}(\text{pow}(a, 2^2)), 0) \\ &\times \text{pow}(\text{pow}(a, 2^2), 1) &\times \text{pow}(\text{sqr}(\text{pow}(a, 2^1)), 1) \\ &\times \text{pow}(\text{pow}(a, 2^1), 1) &\times \text{pow}(\text{sqr}(\text{pow}(a, 2^0)), 1) \\ &\times \text{pow}(\text{pow}(a, 2^0), 0) &\times \text{pow}(\text{pow}(a, 2^0), 0) \end{aligned}$$

❖  $\text{pow}(x, 0) = 1$ 、 $\text{pow}(x, 1) = x$ 、 $\text{pow}(x, 2) = \text{sqr}(x)$

都可在 $\mathcal{O}(1)$ 时间内得到

❖ 故对应于每个数位，只需 $\mathcal{O}(1)$ 时间！



## 幂函数： $O(r)$

```
❖ int power( int a, int n ) {  
    int pow = 1; int p = a; //O(1 + 1)  
    while (0 < n) { //O(logn)  
        if (n & 1) pow *= p; //O(1 + 1)  
        n >>= 1; p *= p; //O(1 + 1)  
    }  
    return pow; //O(1)  
}
```

❖ 输入规模 =  $n$  的二进制位数 =  $r = \lceil \log_2(n+1) \rceil$

❖ 复杂度主要取决于循环次数,  $T(r) = 1 + 1 + 4 \times r + 1 = O(r)$

❖ 从  $O(2^r)$  到  $O(r)$  的改进, 实际效果如何?

## 幂函数：悖论

❖ 观察： $\text{power}(n) = a^n$ 的二进制展开宽度，可以度量为 $\Theta(n)$

❖ 根据以上算法，可在 $\mathcal{O}(r = \log n)$ 时间内计算出 $\text{power}(n)$

❖ 然而，即便是直接打印 $\text{power}(n)$ ，也至少需要 $\Omega(n)$ 时间

...哪里错了？

❖ RAM模型

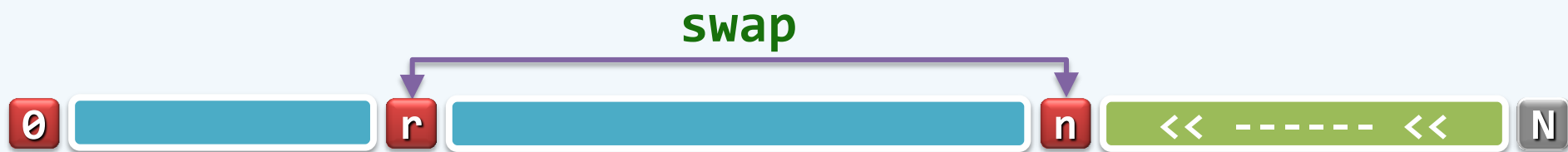
常数代价准则 (uniform cost criterion)

对数代价准则 (logarithmic cost criterion)

## 随机置乱

- ❖ 任给一个数组  $A[0, n)$  , 理想地将其中元素的次序**随机**打乱
- ❖ `//[R. Fisher & F. Yates, 1938], [R. Durstenfeld, 1964], [D. E. Knuth, 1969]`

```
void shuffle( int A[], int n )  
    { while ( 1 < n ) swap( A[ rand() % n ], A[ --n ] ); }
```



- ❖ 策略：自后向前，依次将各元素与随机选取的某一先驱（含自身）交换
- ❖ 的确可以等概率地生成所有  $n!$  种排列？