1.绪论

(x1) 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为间隔循环左移

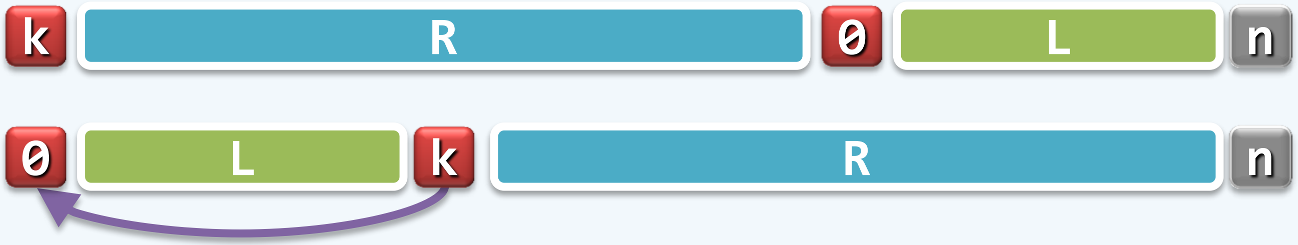
```
{ while (k--) shift(A, n, 0, 1); } //共迭代k次，O(nk)
```

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

❖ //从A[s]出发，以k为间隔循环左移，$\mathcal{O}$(n/GCD(n,k) = LCM(n,k)/k)

```
int shift(int* A, int n, int s, int k) {
   int b = A[s]; int i = s, j = (s + k) % n; int mov = 0;
   while (s != j) //从首元素出发，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 = GCD(n, k)个同余类组成 //各含n/g个元素

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...

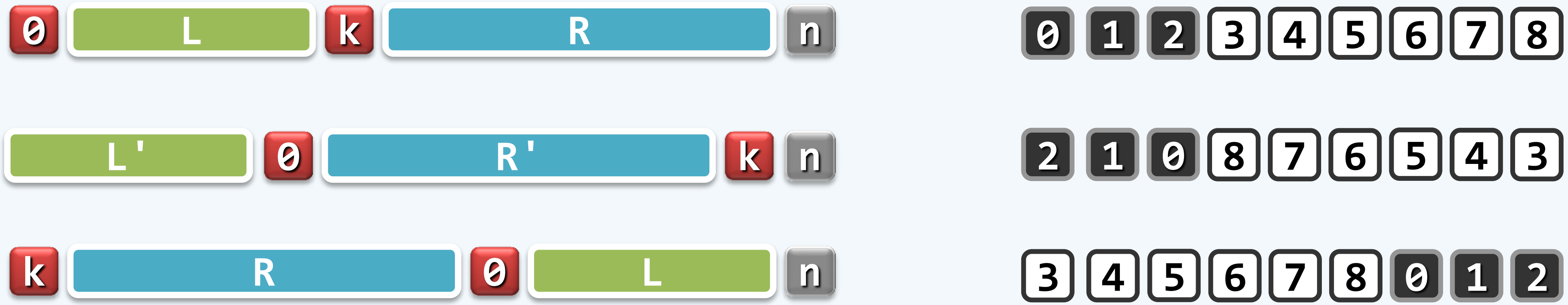
循环移位：迭代版

```
//通过g = GCD(n, k)轮迭代，将数组循环左移k位，O(n)
  void shift1(int* A, int n, int k) {
    for (int s = 0, mov = 0; mov < n; s++) //O(GCD(n, k)) = O(n*k/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

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

a^{98765}

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

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

= pow(pow(a, 10^4), 9)	= pow(pow(pow(a, 10^3), 10), 9)
× pow(pow(a, 10^3), 8)	× pow(pow(pow(a, 10^2), 10), 8)
× pow(pow(a, 10^2), 7)	× pow(pow(pow(a, 10^1), 10), 7)
× pow(pow(a, 10^1), 6)	× pow(pow(pow(a, 10^0), 10), 6)
× pow(pow(a, 10^0), 5)	× pow(pow(a, 10^0) , 5)

❖ 若能在O(1)时间内得到pow(x, n)，$0 \le n \le 10$

则自右（低）向左（高），每个数位只需$O(1)$时间

a^{10110b}

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

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

= pow($\underline{\text{pow(a, } 2^4\text{)}}$, 1)	= pow(sqr($\underline{\text{pow(a, } 2^3\text{)}}$), 1)
× pow($\underline{\text{pow(a, } 2^3\text{)}}$, 0)	× pow(sqr($\underline{\text{pow(a, } 2^2\text{)}}$), 0)
× pow($\underline{\text{pow(a, } 2^2\text{)}}$, 1)	× pow(sqr($\underline{\text{pow(a, } 2^1\text{)}}$), 1)
× pow($\underline{\text{pow(a, } 2^1\text{)}}$, 1)	× pow(sqr($\underline{\text{pow(a, } 2^0\text{)}}$), 1)
× pow($\underline{\text{pow(a, } 2^0\text{)}}$, 0)	× pow($\underline{\text{pow(a, } 2^0\text{)}}$, 0)

❖ pow(x, 0) = 1、pow(x, 1) = x、pow(x, 2) = 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)

❖ 算法

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

❖ 复杂度

循环次数

$= n$的二进制位数

$= r$

$= \lceil \log_2(n+1) \rceil$

$T(r)$

$= 1 + 1 + 4 \times r + 1$

$= \mathcal{O}(r)$

❖ 从$\mathcal{O}$(2^r)到$\mathcal{O}$(r)的改进

实际效果如何？

幂函数：悖论

❖ 观察：power(n) = a^n的二进制展开宽度，可以度量为Θ(n)

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

❖ 然而，即便是直接打印power(n)，也至少需要Ω(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前驱（含自身）交换

❖ 的确可以等概率地生成所有n!种排列？