

词典

基数排序：算法与实现

11-E1

邓俊辉

deng@tsinghua.edu.cn

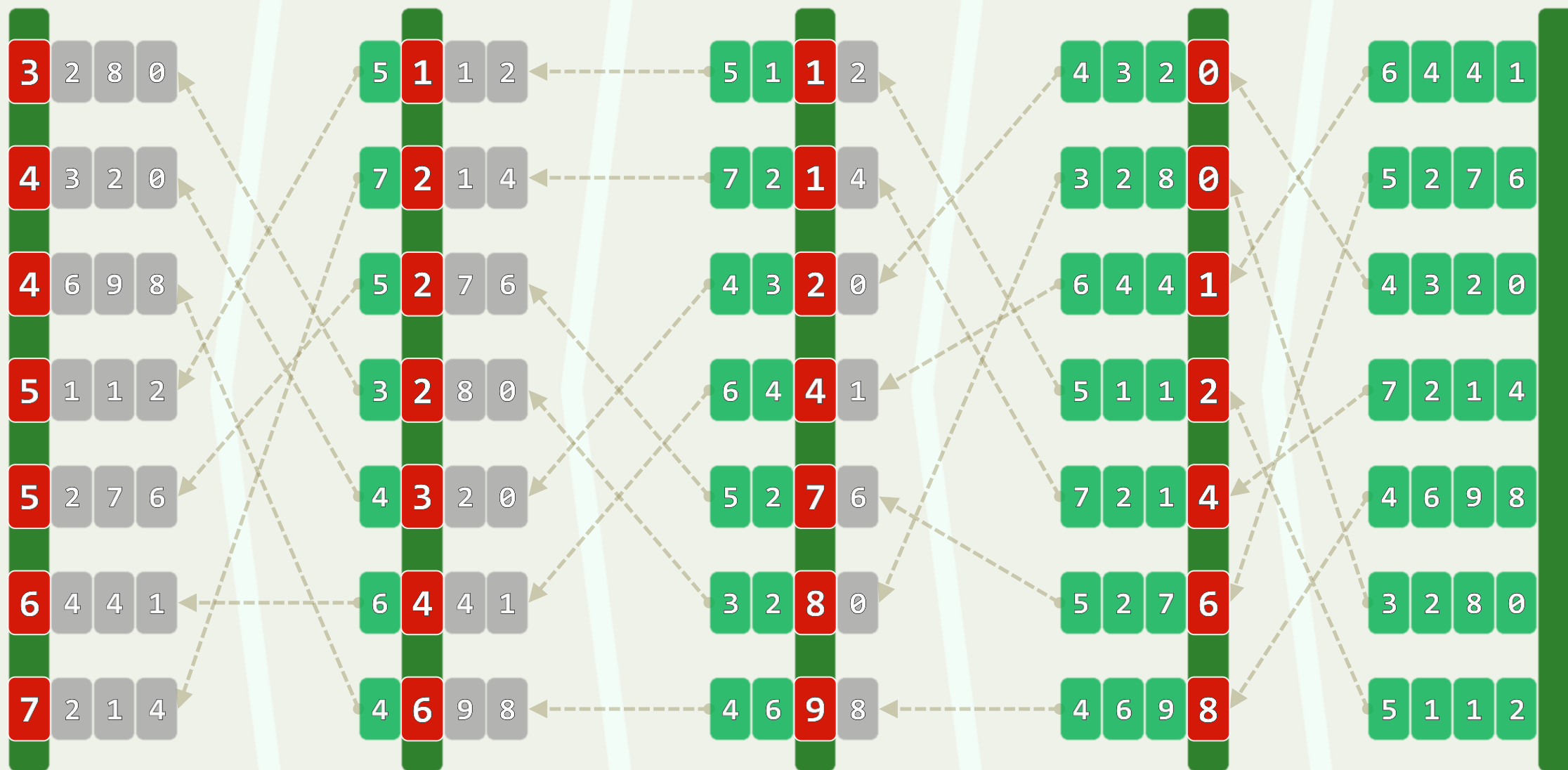
词典序

❖ 有时，关键码由多个域组成： k_t , k_{t-1} , ... , k_1

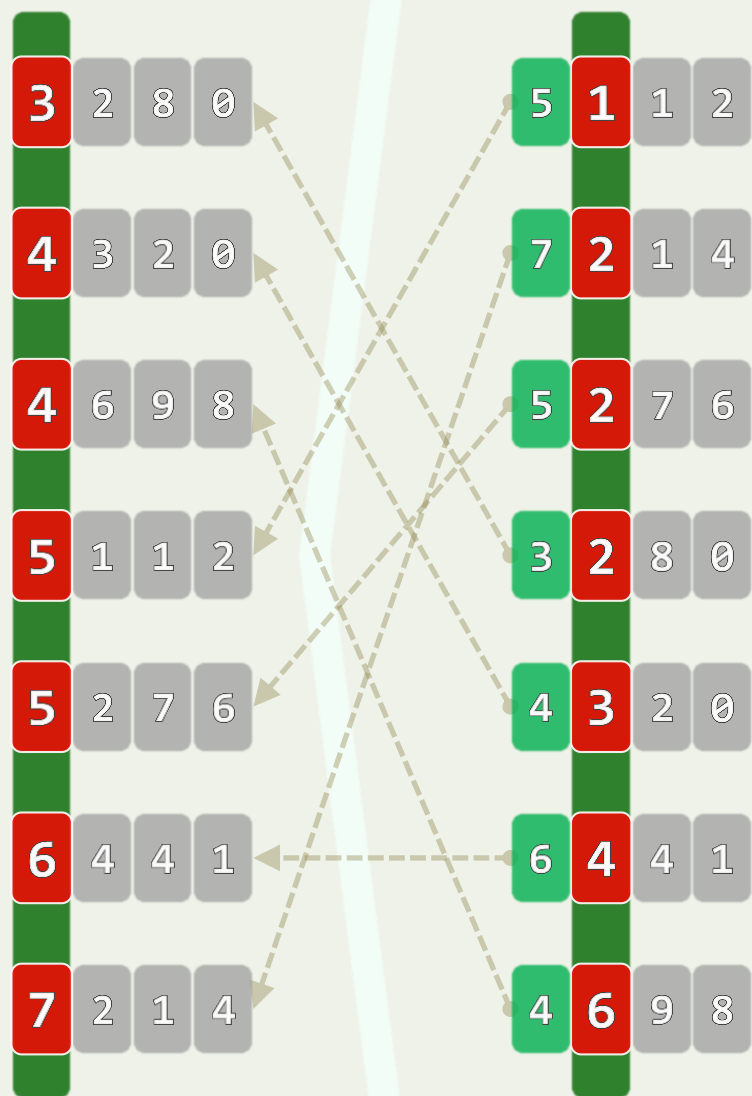
❖ 若将各域视作字母，则关键码即单词——按词典的方式排序 (lexicographic order)



算法：自 k_1 到 k_t （**低位优先**），依次以各域为序做一趟**桶排序**

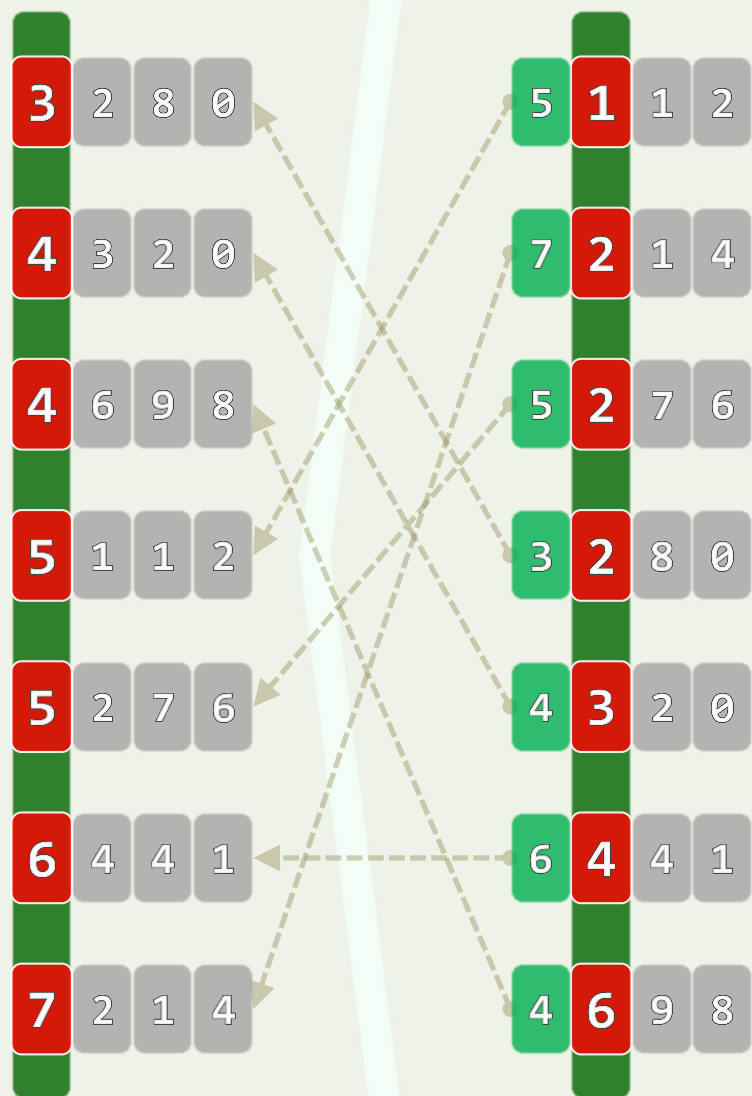


正确性



- ❖ 归纳假设：前 i 趟排序后，所有词条关于低 i 位有序（第1趟显然）
- ❖ 假设前 $i-1$ 趟均成立，现考查第 i 趟排序之后的时刻
- ❖ 无非两种情况
 - 凡第 i 位不同的词条
即便此前曾是逆序，现在亦必已转为有序
 - 凡第 i 位相同的词条
得益于桶排序的稳定性，必保持原有次序

时间成本



= 各趟桶排序所需时间之和

$$= n + 2m_1$$

$$+ n + 2m_2$$

$$+ \dots$$

$$+ n + 2m_t \quad // m_k \text{ 为各域的取值范围}$$

$$= O(t \times (n + m))$$

$$// m = \max\{m_1, \dots, m_t\}$$

❖ 当 $m = O(n)$ 且 t 可视作常数时, $O(n)$!

❖ 在一些特定场合, Radixsort 非常高效, 比如...

实现（以二进制无符号整数为例）

```
❖ typedef unsigned int U; //约定：类型T或就是U；或可转换为U，并依此定序

❖ template <typename T> void List<T>::radixSort( ListNodePosi(T) p, int n ) {
    ListNodePosi(T) head = p->pred; ListNodePosi(T) tail = p;

    for ( int i = 0; i < n; i++ ) tail = tail->succ; //待排序区间为(head, tail)

    for ( U radixBit = 0x1; radixBit && (p = head); radixBit <<= 1 ) //以下反复地
        for ( int i = 0; i < n; i++ ) //根据当前基数位，将所有节点
            radixBit & U (p->succ->data) ? //分拣为前缀（0）与后缀（1）
                insert( remove( p->succ ), tail ) : p = p->succ;
    } //为避免remove()、insert()的低效率，可实现List::moveB(t,p)接口，将节点p挪至t之前
```

实例

