深入理解并实现基本的基数排序（Radix Sort）算法

叶家炜

Aug 04, 2025

排序算法在计算机科学中占据基础地位，广泛应用于数据处理、数据库索引、搜索算法等多个领域。常见的排序算法可分为比较排序（如快速排序、归并排序）和非比较排序两大类。基数排序作为非比较排序的代表，以其线性时间复杂度的特性脱颖而出，特别适用于整数或字符串等可分解键值的数据类型。本文旨在透彻解析基数排序的原理，通过手把手实现代码加深理解，并分析其性能与适用边界，帮助读者掌握这一高效算法。

1　基数排序的核心思想

基数排序的核心在于「基数」的概念，即键值的进制基数，如十进制中基数为 10。排序过程通过按位进行，从最低位到最高位（LSD 方式），在多轮「分桶-收集」操作中完成。这类似于整理扑克牌时，先按花色分桶，再按点数排序。关键特性是每轮排序必须保持稳定性，即相同键值的元素在排序后保持原顺序，这对算法的正确性至关重要。稳定性确保在后续高位排序时，低位排序的结果不被破坏。

2　算法步骤详解

基数排序的算法步骤包括预处理和核心循环。预处理阶段需要确定数组中最大数字的位数 d，这决定了排序轮数。核心循环中，每轮针对一个位进行分桶与收集操作。具体步骤为：首先创建 10 个桶（对应数字 0 到 9）；然后按当前位数字将元素分配到相应桶中，确保分配过程稳定；接着按桶顺序（从 0 到 9）收集所有元素回原数组；最后更新当前位向高位移动，重复此过程直至最高位。以数组 $[170, 45, 75, 90, 802, 24, 2, 66]$ 为例，第一轮按个位分桶：桶 0 包含 170 和 90，桶 2 包含 802 和 2，桶 4 包含 24，桶 5 包含 45 和 75，桶 6 包含 66；收集后数组为 $[170, 90, 802, 2, 24, 45, 75, 66]$；第二轮按十位分桶：桶 0 包含 802 和 2，桶 6 包含 66，桶 7 包含 170 和 75，桶 9 包含 90；收集后数组为 $[802, 2, 24, 66, 170, 75, 90, 45]$；第三轮按百位分桶后收集，最终得到有序数组 $[2, 24, 45, 66, 75, 90, 170, 802]$。

3　时间复杂度与空间复杂度分析

基数排序的时间复杂度为 $O(d \cdot (n + k))$，其中 d 是最大位数，n 是元素数量，k 是基数（桶的数量）。与比较排序如快速排序的 $O(n \log n)$ 相比，当 d 较小且 k 不大时，基数排序效率更高，尤其在数据规模大但位数少的场景。空间复杂度为 $O(n + k)$，主要来自桶的额外存储。稳定性是算法成立的前提，因为每轮排序必须是稳定的，以保证高位排序时低位顺序不被破坏；如果某轮排序不稳定，整体结果可能出错。

4 基数排序的局限性

尽管高效，基数排序有显著局限性。它主要适用于整数、定长字符串（需补位）或前缀可比较的数据类型，不直接处理浮点数或可变长数据（需额外转换）。当基数 k 较大时，如处理 Unicode 字符串，空间开销显著增加。此外，如果位数 d 接近元素数 n，算法可能退化为 $O(n^2)$ 效率，例如在大范围稀疏数据中。

5 代码实现（Python 示例）

以下是用 Python 实现的基数排序代码：

```
def radix_sort(arr):
    # 1. 计算最大位数
    max_digits = len(str(max(arr)))

    # 2. LSD 排序循环
    for digit in range(max_digits):
        # 创建 10 个桶
        buckets = [[] for _ in range(10)]

        # 按当前位分配元素
        for num in arr:
            current_digit = (num // (10 ** digit)) % 10
            buckets[current_digit].append(num)

        # 收集元素（保持桶内顺序）
        arr = [num for bucket in buckets for num in bucket]

    return arr

# 测试
arr = [170, 45, 75, 90, 802, 24, 2, 66]
print("排序前␣:", arr)
print("排序后␣:", radix_sort(arr))
```

代码解读：首先，在预处理阶段，`max_digits = len(str(max(arr)))` 计算数组中最大数字的位数，例如最大数 802 的位数为 3。在 LSD 循环中，变量 `digit` 表示当前处理的位索引（从 0 开始，0 为个位）。对于每个数字 `num`，`current_digit = (num // (10 ** digit)) % 10` 提取当前位数字：例如当 `digit=0` 时，170 的个位为 $(170//10^0)\%10 = 170\%10 = 0$。元素被分配到 `buckets` 列表中，桶使用列表的列表实现，确保稳定性（相同当前位数字的元素保持原顺序）。收集操作 `arr = [num for bucket in buckets for num in bucket]` 通过列表推导式将所有桶中的元素扁平化回数组，保持桶内顺序。测试部分输出排序前后数组，验

证算法正确性。

6 变体与优化

基数排序有多个变体与优化方向。MSD（最高位优先）基数排序采用递归方式，先按最高位分桶，再对每个桶递归排序，适合字符串处理。在桶的实现上，可用链表代替动态数组以减少内存分配开销；尝试原地排序虽复杂但可能节省空间，但需牺牲稳定性。对于负数处理，可分离正负数分别排序，或通过添加偏移量（如加 1000）将负数转为正数处理后再排序，最后还原符号。

7 实际应用场景

基数排序在实际中常用于大范围整数排序，如数据库索引构建或大规模 ID 排序，其中数据量大但位数有限。它也适用于定长字符串的字典序排序，例如车牌号或 ISBN 号的快速处理。此外，基数排序可扩展至混合键值排序场景，如先按日期（高位）再按 ID（低位）的多级排序，充分利用其稳定性优势。

基数排序的核心优势在于其线性时间复杂度 $O(d \cdot (n + k))$，突破比较排序的下限 $O(n \log n)$。使用时需满足键值可分解、排序过程稳定且空间充足等前提。学习基数排序不仅掌握一种高效算法，更体现了非比较排序的设计思想和空间换时间的经典权衡，为处理特定数据类型提供优化方案。