基数排序

杨子凡

Jun 08, 2025

排序算法在计算机科学中占据基础地位，广泛应用于数据库查询优化、大数据处理系统等场景。基数排序作为一种非比较排序算法，具有独特的优势：它能够突破基于比较的排序算法（如快速排序或归并排序）的时间复杂度下界 O(n log n)，在特定场景下实现线性时间复杂度。本文将从底层原理出发，结合 Python 代码实现，深入解析基数排序的工作机制、性能特征及优化策略，帮助读者透彻理解其适用边界与实现细节。

1　基数排序基础概念

基数排序的核心思想是“按位分组排序”，即通过多轮分配与收集操作，从最低位（LSD）或最高位（MSD）开始逐位处理元素。这本质上是桶排序的扩展，利用稳定排序的叠加效应实现全局有序。关键术语包括「数位」—— 指元素的每一位（如数字的个位、十位），「基数」—— 代表数位的取值范围（如十进制基数为 10），以及排序方向的选择：LSD（Least Significant Digit）从最低位开始排序，适用于大多数整数场景；MSD（Most Significant Digit）从最高位开始，行为类似字典树，常用于字符串排序。

2　算法原理深度剖析

LSD 基数排序的流程分为三步：首先计算最大数字的位数 k；然后从最低位到最高位遍历，每轮按当前位分配桶（桶数量等于基数），再按桶顺序收集元素；最终返回有序数组。稳定性在此至关重要 —— 例如对数组 [21，15，12] 按十位排序后（桶分组为 [12] 和 [21，15]），个位排序需保持 21 与 15 的相对顺序，否则结果错误。时间复杂度为 O(d \times (n + b))，其中 d 是最大位数，n 是元素数量，b 是基数。当 d 较小且 n 较大时（如处理 10^7 个 32 位整数），基数排序性能优于快排等算法，因为其避免了比较操作的 log n 因子。

3　代码实现（Python 示例）

```
def radix_sort(arr):
    # 计算最大数字的位数
    max_num = max(arr)
    exp = 1
    while max_num // exp > 0:
        counting_sort(arr, exp)
        exp *= 10
```

```
def counting_sort(arr, exp):
    n = len(arr)
    output = [0] * n
    count = [0] * 10 # 十进制基数

    # 统计当前位出现次数
    for i in range(n):
        index = arr[i] // exp % 10
        count[index] += 1

    # 计算累计位置
    for i in range(1, 10):
        count[i] += count[i-1]

    # 按当前位排序（逆序保证稳定性）
    i = n - 1
    while i >= 0:
        index = arr[i] // exp % 10
        output[count[index] - 1] = arr[i]
        count[index] -= 1
        i -= 1

    # 写回原数组
    for i in range(n):
        arr[i] = output[i]
```

这段代码实现 LSD 基数排序，核心是 radix_sort 函数与子过程 counting_sort。首先，radix_sort 计算最大数字的位数（如 123 的位数为 3），通过变量 exp（初始为 1）控制当前处理的数位（个位、十位等）。exp 在每轮乘以 10，直到覆盖最高位。子过程 counting_sort 是桶排序的优化版本：它统计当前位（由 exp 指定）的出现频率，计算累计位置以确定元素在输出数组中的索引。关键点在于逆序填充（从数组末尾开始处理），这保证了稳定性 —— 当两个元素当前位相同时，原始顺序得以保留。例如，在十位排序轮中，21 和 15 同属桶 1，逆序处理确保 21 先于 15 被放置，维持相对位置。最后，排序结果写回原数组。

4 关键问题与优化

处理负数时有两种常见方案：一是分离正负数组，负数取绝对值排序后反转再合并；二是通过偏移量法将所有数加上最小值转为非负。对于字符串排序，可应用 LSD 方法从右到左按字符分桶（不足位补空字符），例如对 [apple, banana, cherry] 排序时，首轮按末字符分桶。基数选择优化需权衡轮数与桶数：二进制基数（b=2）减少桶数但增加轮数（d 增大），十进制基数（b=10）则相反。理论最优点在 $b \approx n$ 时达到时间复杂度平衡，参考《算法导论》（CLRS）第 8 章分析。

5 性能对比与局限

基数排序在均匀分布的整数或定长字符串（如手机号、IP 地址）场景下表现卓越，尤其当数据量远大于数值范围时（例如 1e6 个 [0, 1000] 的整数）。然而，其空间复杂度 O(n + b) 带来显著桶开销，且不适合浮点数（需处理 IEEE 754 编码）或动态数据结构如链表。基准测试显示，在 1e7 个 32 位整数排序中，基数排序性能稳定，而快速排序在退化情况下（如已有序数组）效率骤降。

6 应用案例

在数据库索引优化中，基数排序支持多字段排序（如按年、月、日分层处理）。MapReduce 模型下可扩展为分布式版本，按数位分片并行处理。计算机图形学中的深度缓冲排序（Z-buffering）也依赖类似机制高效管理像素深度。

基数排序的本质是多轮稳定桶排序与数位分解思想的结合，但其“非比较排序”特性并非万能 —— 需严格匹配数据特性（如元素可分割为离散位）。思考题：对数组 `[(Alice, 25), (Bob, 20), (Alice, 20)]` 按姓名→年龄排序时，可先按年龄（低位）稳定排序，再按姓名（高位）排序，利用稳定性保持同名元素的年龄顺序。