Сортировки для особых случаев

Далее будут рассмотрены некоторые сортировки, которые применимы в строго определенных случаях. Однако, из-за наложенных на их применимость ограничений скорость их работы довольно высокая.

1 Сортировка подсчетом

Эта сортировка используется в случаях, когда количество уникальных сортируемых элементов является заведомо небольшим. В таком случае их можно упорядочить за время $O(n)$. Например, будем сортировать n целых чисел из отрезка $[0; k]$, где k – некоторая заранее известная константа. Тогда просто пройдем по массиву чисел и посчитаем, сколько раз встретился каждый элемент. После этого просто построим отсортированный массив, записывая туда элементы от 0 до k в нужном количестве.

```
def counting_sort(a, k):
    c = [0] * (k + 1)
    for i in a:
        c[i] += 1
    b = []
    for i in range(k + 1):
        b += [i] * c[i]
    return b
```

2 Цифровая сортировка

Пусть наш массив состоит из N элементов, которые можно разделить на разряды (например, числа или строки). Будем считать, что все элементы массива содержат M разрядов (числа мы всегда сможем дополнить лидирующими нулями, а строки – 0). Стоит уточнить, что сравнение символов строки происходит в соответствии с таблицей кодировки символов (например, ASCII). Рассмотрим алгоритм, которая выполняет сортировку элементов двигаясь по их разрядам. Такая сортировка будет работать за $O(N)$, считая $N \gg M$, или $O(NM)$ в общем случае. Цифровая сортировка имеет две возможные реализации: сортировка со старшего разряда (Most Significant Digit radix sort, MSD) и сортировка с младшего разряда (Least Significant Digit radix sort). Далее в примерах реализации сортировки написаны для массива чисел, хранящихся в строках. Все строки одинаковой длины за счет лидирующих нулей. Значение M можно посчитать в процессе считывания массива чисел, значение K – количество возможных значений разрядов – известно из постановки задачи. В нашем случае $K = 10$. В обоих реализация понадобится вспомогательная функция digit, которая возвращает значение указанного разряда.

```
def digit(e, i):
    return int(e[i])
```

2.1 LSD

В процессе работы алгоритма вспомогательным является любая другая устойчивая сортировка. Обычно, это сортировка подсчетом, т.к. K – маленькое число. Алгоритм содержит внешний цикл, который указывает на текущий разряд, по которому выполняется сортировка. На каждой итерации цикла выполняется сортировка элементов по текущему разряду выбранной устойчивой сортировкой.

```
def lsd(a):
    n = len(a)
    m = 3
    for i in range(m-1, -1, -1):
        c = [[] for _ in range(10)]
        for e in a:
            d = digit(e, i)
            c[d].append(e)
        a = sum(c, [])
    return a
```

2.2 MSD

Такой вариант цифровой сортировки представлен рекурсией. Для начала массив разбивается на части по значению старшего разряда каждого элемента. Такие части называют «корзинами». Далее, каждая корзина – массив элементов, у которых одинаковый старший разряд. Запустим для каждой корзины эту же функцию цифровой сортировки, только теперь используем следующий разряд элементов. Критерии останова рекурсии: мы рассмотрели все разряды чисел или в корзине меньше двух элементов. Изначально функция запускается с аргументами msd(a, 0, len(a), 1).

```
def msd(a, left, right, i):
    if i >= m or left >= right - 1:
        return
    c = [[] for _ in range(k)]
    for j in range(left, right):
        d = digit(a[j], i)
        c[d].append(a[j])
    j = left
    for basket in c:
        for e in basket:
            a[j] = e
```

```
            j += 1
    j = left
    for basket in c:
        msd(a, j, j + len(basket), i + 1)
        j += len(basket)
```

Список литературы

Кормен, Томас и др. (2013). *Алгоритмы. Построение и анализ. Третье издание*. Издательский дом «Вильямс».