LSD и MSD

ceagest

1 Сортировка подсчётом

Пусть есть массив положительных чисел, элементы которого ограничены неким числом K. Хотим его отсортировать.

- Почему бы просто не сосчитать какое значение вошло в него сколько раз?
- Затем выписать элементы по возрастанию нужное число раз!

Подводным камнем такой сортировки является отсутствие стабильности, но отчаиваться рано, ведь можно сделать эту сортировку стабильной.

1.1 Устойчивая сортировка подсчётом

- Будем для каждого значения хранить информацию о том, сколько элементов меньше либо равны его самого (префикс сумма массива счета).
- Пойдем с конца исходного массива и будем с помощью массива префикс суммы массива счета вычислять позицию текущего элемента.
- Поскольку мы идем с конца, то при первой встрече значения a_i он получит позицию $\text{prefCnt}[a_i]$
- После установки элемента в итоговый массив, $\text{prefCnt}[a_i]$ нужно уменьшить на 1.

Устойчивая сортировка подсчётом на Java

```java
public class CountingSort {
    private static final int MAX = 256;
    public static void countSort(int[] a, int[] res, int n) {
        int[] prefCnt = new int[MAX];
        for (int i = 0; i < n; ++i) {
            prefCnt[a[i]]++;
        }
        for (int i = 1; i < MAX; ++i) {
            prefCnt[i] += prefCnt[i - 1];
        }
        for (int i = n - 1; i >= 0; --i) {
            res[--prefCnt[a[i]]] = a[i];
        }
    }
}
```

2 Least Significant Digit Sort (LSD)

- Поймем, что целое число это набор байт.
- Что будет, если сортировать от последнего разряда к первому?
- Тут важно заметить, что нам КРАЙНЕ важна стабильность сортировки — без нее перемешаются старые результаты для менее значащих разрядов.

Время сортировки составляет $\mathcal{O}(n+K)$, где K – максимальное значение цифры в основании системы счисления, по которой число сортируется. Потребляемая доп. память составляет $\mathcal{O}(n+K)$, поскольку нужен доп. массив для совершения перестановки, и для хранения счетчиков.

Реализация алгоритма LSD на Java

```java
public class LSDSort {
    private static final int MAX = 256;
    public static void lsdSort(int[] a, int n) {
        int[] res = new int[n];
        for (int mask = 0xFF, shift = 0; mask > 0; mask <<= 8, shift += 8) {
            int[] prefCnt = new int[MAX];
            for (int i = 0; i < n; ++i) {
                int digit = (a[i] & mask) >> shift;
                prefCnt[digit]++;
            }
            for (int i = 1; i < MAX; ++i) {
                prefCnt[i] += prefCnt[i - 1];
            }
            for (int i = n - 1; i >= 0; --i) {
                int digit = (a[i] & mask) >> shift;
                res[--prefCnt[digit]] = a[i];
            }
            System.arraycopy(res, 0, a, 0, n);
        }
    }
}
```

3 Most Significant Digit Sort (MSD)

- А если пойти от самого значимого разряда?
- Тогда мы последними будем сортировать младшие разряды и результат для более важных разрядов может перемешаться...
- Давайте после сортировки разряда рекурсивно вызывать сортировку для последовательностей элементов, которые оказались равными в текущем разряде.
- Можно сказать, что мы разбили элементы на корзины.
- Такую сортировку часто называют «корзинной сортировкой».

Алгоритм требует $\mathcal{O}(n+K)$ памяти, и также $\mathcal{O}(n+K)$ времени (в случае чисел). Можно также использовать для сортировки строк.