

LSD и MSD

Лесников Юрий, seagest

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

```
1 public class CountingSort {
2     private static final int MAX = 256;
3     public static void countSort(int[] a, int[] res, int n) {
4         int[] prefCnt = new int[MAX];
5         for (int i = 0; i < n; ++i) {
6             prefCnt[a[i]]++;
7         }
8         for (int i = 1; i < MAX; ++i) {
9             prefCnt[i] += prefCnt[i - 1];
10        }
11        for (int i = n - 1; i >= 0; --i) {
12            res[--prefCnt[a[i]]] = a[i];
13        }
14    }
15 }
```

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 $\mathcal{O}(m(n + K))$, если устойчивая сортировка (в данном случае сортировка подсчётом) имеет время работы $\mathcal{O}(n + K)$. K – максимальное значение цифры в основании системы счисления, по которой число сортируется, m – количество разрядов. Если m – константа, а $K = \mathcal{O}(n)$, то сложность сортировки составляет $\mathcal{O}(n)$. Потребляемая доп. память составляет $\mathcal{O}(n + K)$, поскольку нужен доп. массив для совершения перестановки, и для хранения счетчиков.

Реализация алгоритма LSD на Java

```

1 public class LSDSort {
2     private static final int MAX = 256;
3     public static void lsdSort(int[] a, int n) {
4         int[] res = new int[n];
5         for (int mask = 0xFF, shift = 0; mask > 0; mask <= 8, shift += 8) {
6             int[] prefCnt = new int[MAX];
7             for (int i = 0; i < n; ++i) {
8                 int digit = (a[i] & mask) >> shift;
9                 prefCnt[digit]++;
10            }
11            for (int i = 1; i < MAX; ++i) {
12                prefCnt[i] += prefCnt[i - 1];
13            }
14            for (int i = n - 1; i >= 0; --i) {
15                int digit = (a[i] & mask) >> shift;
16                res[--prefCnt[digit]] = a[i];
17            }
18            System.arraycopy(res, 0, a, 0, n);
19        }
20    }
21 }

```

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. Пусть значения разрядов меньше b , а количество разрядов — k . При сортировке массива из одинаковых элементов MSD-сортировкой на каждом шаге все элементы будут находиться в неубывающей по размеру корзине, а так как цикл идет по всем элементам массива, то получим, что время работы MSD-сортировки оценивается величиной $\mathcal{O}(nk)$, причем это время нельзя улучшить.