Сортировка подсчётом

Слёлин А.В.

18 мая, 2024 г.

Описание алгоритма

Пусть дан массив $A[0..n-1]$, такой что $n = \text{length}[A]$, причём индексация начинается с нуля для облегчения понимания кода.

Рассмотрим случай когда нам нужно отсортировать массив A в неубывающую последовательность.

В сортировке подсчётом предполагается, что все n входных элементов - целые числа, принадлежащие интервалу от min до max. Пусть $k = \max - \min + 1$ - некоторая целая константа.

Основная идея сортировки заключается в том, чтобы для каждого входного элемента x определить количество элементов, которые меньше x. С помощью этой информации элемент x можно разместить на той позиции выходного массива, где он должен находиться. Для этого потребуется еще два массива: в массиве $B[0..n-1]$ будет содержаться отсоритрованная выходная последовательность, а массив $C[0,k]$ служит временным рабочим хранилищем.

Важное свойство алгоритма сортировки подсчётом заключается в том, что он устойчив: элементы с одним и тем же значением находятся в выходном массиве в том же порядке, что и во входном.

Пример

Пусть массив $A = [2,5,3,-1,2,3,-2,3]$. Мы определяем, что элемент $\min = -2$, а $\max = 5$, значит $k = 5 - (-2) + 1 = 8$. Значит создаём массив C, в котором посчитаем количество чисел $i - \min$, то есть в нулевой клетке находится количество min в массиве A, в первой клетке находится количество $\min - 1$ в массиве A, и так далее пока не дойдём до max:

$$C = [1,1,0,0,2,3,0,1].$$

Затем определим сколько элементов для $i - min$ меньше либо равны его значения:

$$C = [1,2,2,2,4,7,7,8].$$

То есть получаем что у нас один элемент меньше либо равен min (сам min), два элемента меньше либо равны $\min - 1$, два элемента меньше либо равны $\min - 2$ и так далее до 8 элементов, которые меньше либо равно max.

Создадим массив B:

$$B = [0,0,0,0,0,0,0,0].$$

Теперь для устойчивости алгоритма начнём обратный цикл и будем брать элемент из A с индексом i (выделено красным цветом), затем смотреть по массиву C куда его нужно ставить (выделено синим цветом), а затем подставлять в сам массив B (выделено зелёным цветом; также жёлтым выделены элементы, которые ещё не менялись).

i = 7:

$$A = [2, 5, 3, -1, 2, 3, -2, 3], \quad C = [1, 2, 2, 2, 4, 7, 7, 8];$$
$$B = [0, 0, 0, 0, 0, 0, 3, 0];$$

i = 6:

$$A = [2, 5, 3, -1, 2, 3, -2, 3], \quad C = [1, 2, 2, 2, 4, 6, 7, 8];$$
$$B = [-2, 0, 0, 0, 0, 0, 3, 0];$$

i = 5:

$$A = [2, 5, 3, -1, 2, 3, -2, 3], \quad C = [0, 2, 2, 2, 4, 6, 7, 8];$$
$$B = [-2, 0, 0, 0, 0, 3, 3, 0];$$

i = 4:

$$A = [2, 5, 3, -1, 2, 3, -2, 3], \quad C = [0, 2, 2, 2, 4, 5, 7, 8];$$
$$B = [-2, 0, 0, 2, 0, , 3, 3, 0];$$

i = 3:

$$A = [2, 5, 3, -1, 2, 3, -2, 3], \quad C = [0, 2, 2, 2, 3, 5, 7, 8];$$
$$B = [-2, -1, 0, 2, 0, 3, 3, 0];$$

i = 2:

$$A = [2, 5, 3, -1, 2, 3, -2, 3], \quad C = [0, 1, 2, 2, 3, 5, 7, 8];$$
$$B = [-2, -1, 0, , 2, 3, 3, 3, 0];$$

i = 1:

$$A = [2, 5, 3, -1, 2, 3, -2, 3], \quad C = [0, 1, 2, 2, 3, 4, 7, 8];$$
$$B = [-2, -1, 0, , 2, 3, 3, 3, 5];$$

i = 0:

$$A = [2, 5, 3, -1, 2, 3, -2, 3], \quad C = [0, 1, 2, 2, 3, 4, 7, 7];$$
$$B = [-2, -1, 2, , 2, 3, 3, 3, 5].$$

В массиве B находится остортированный массив A. Для получения невозрастающей последовательности следует переписать неубывающую последовательность в обратную сторону в цикле.

Код сортировки

Листинг 1: COUNTINGSORT

```
1       public static int[] COUNTING_SORT(int[] A, int min, int max)
            {
2           int[] C = new int[max - min + 1], B = new int[A.length];
3           for (int i = 0; i < A.length; ++i)
4               ++C[A[i] - min];
5           for (int i = 1; i <= max - min; ++i)
6               C[i] += C[i - 1];
7
8           for (int i = A.length - 1; i >= 0; --i)
9               B[--C[A[i] - min]] = A[i];
10          return B;
11      }
```

Заметим, что обратный цикл для данного алгоритма нужен только для устойчивости.

Исходный код

Представим весь код Main.java для тестирования алгоритма.

Листинг 2: Main

```
1   public class Main {
2       public static int[] COUNTING_SORT(int[] A, int min, int max)
            {
3           int[] C = new int[max - min + 1], B = new int[A.length];
4           for (int i = 0; i < A.length; ++i)
5               ++C[A[i] - min];
6           for (int i = 1; i <= max - min; ++i)
7               C[i] += C[i - 1];
8
9           for (int i = A.length - 1; i >= 0; --i)
10              B[--C[A[i] - min]] = A[i];
11          return B;
12      }
13
14      public static int[] SORT(int[] A, boolean reverse) {
15          int max = Integer.MIN_VALUE, min = Integer.MAX_VALUE;
16          for (int i = 0; i < A.length; ++i) {
17              max = Math.max(max, A[i]);
18              min = Math.min(min, A[i]);
19          }
20
21          int[] B = COUNTING_SORT(A, min, max);
22          if (reverse) {
23              for (int i = 0; i < B.length / 2; ++i) {
```

```
                int tmp = B[i];
                B[i] = B[B.length - i - 1];
                B[B.length - i - 1] = tmp;
            }
        }

        return B;
    }

    public static void PRINT(int[] A) {
        for (int i = 0; i < A.length; ++i)
            System.out.print(A[i] + " ");

        System.out.println();
    }

    public static void main(String[] args) {
        int[] A = {-2, 5, 3, 2, 2, 3, 5, 3, 3, 6, 0, -1};

        PRINT(A);
        int[] B = SORT(A, false);
        PRINT(B);
    }
}
```

Сложность

Сначала оценим пространственную сложность. Мы создаём два массива C и B длинной соответственно $k+1$ и n. Поэтому просторанственная сложность сортировки подсчётом $O(n+k)$.

Рассмотрим сразу все случаи для определния временной сложности - лучший (отсортированный массив), средний (неотсортированный массив), худший (отсортированный массив в обратную сторону). Во всех трёх ситуациях нам нужно линейно в цикле проходится по массиву длинной n и длинной k. Нахождения минимума и максимума асимптотику не меняет, как и создание обратной последовательности при значении reverse = true, но увеличивает константы, которые скрываются за O. Поэтому временная сложность алгоритма сортировки подсчётом в лучшем, среднем и худшем случаях равна соответственно $\Omega(n+k)$, $\Theta(n+k)$, $O(n+k)$.

Временная сложность			Пространственная сложность
Худший	Средний	Лучший	Худший
$O(n+k)$	$\Theta(n+k)$	$\Omega(n+k)$	$O(n+k)$

Таблица 1: **Резюме**

Комментарии

Данный алгоритм стоит использовать если мы точно знаем, что $k = O(n)$, тогда временная асимптотика получается $O(n)$, а пространственная $O(n)$, что является хорошим показателем. В остальных случаях ($k = O(n^2)$) данную сортировку нет смысла использовать. Этот вид сортировки даст скорость только с «хорошими» входными данными.