Пузырьковая сортировка

Слёлин А.В.

17 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; случай, когда нужно отсортировать в невозрастающую последовательность аналогичен.

Алгоритм пузырьковой сортировки заключается в том, чтобы поднимать самые «лёгкие» (маленькие по значению) элементы вверх, подобно тому, как пузырёк воздуха, в силу своей несущественной массы, поднимается в воде вверх.

Если мы будем сравнивать в массиве два числа $A[j]$ и $A[j-1]$ с конца (от $n-1$ до 1), и менять их в случае если $A[j] < A[j-1]$, тогда за один такой проход (итерацию) по всем j от $n-1$ до 1 самый «лёгкий пузырёк поднимется вверх», то есть наименьший элемент массива A гарантированно окажется в нулевой ячейке массива.

Далее сделаем тоже самое, уменьшив количество проходимых ячеек j на 1 с начала (от $n-1$ до 2), так как нам уже известно, что в нулевой ячейке находится наименьший элемент. За второй проход по j от $n-1$ до 2 мы получим в начале массива уже наименьший элемент в нулевой ячейке массива A и второй наименьший элемент в первой ячейке массива A.

Далее проделаем такие же действия, уменьшая проход j каждый раз на 1, пока не дойдём до последнего прохода j от $n-1$ до $n-1$ (проверка последней пары). Тогда массив A будет полностью отсортирован.

Во время выполнения итераций может оказаться так, что за одну итерацию ничего не менялось местами, а это возможно только в одном случае - массив уже отсортирован. Для большей оптимизации стоит завести переменную, которая будет следить за тем, были ли обмены в одной итерации или нет.

Пример

Пусть массив $A = [8, 3, 12, 16, 10, 4, 1, 9, 7, 14]$. i - означает итерацию (проход), j числа которые будут сравниваться (запись $j \div a : b$ - означает, что j пробегает целые значения от a до b с шагом 1, если $a < b$, и с шагом -1, если $a > b$).

i = 0:

$j \div 9 : 1$

$$A = [1, 8, 3, 12, 16, 10, 4, 7, 9, 14];$$

i = 1:

$j \div 9 : 2$

$$A = [1, 3, 8, 4, 12, 16, 10, 7, 9, 14];$$

i = 2:

$j \div 9 : 3$

$$A = [1, 3, 4, 8, 7, 12, 16, 10, 9, 14];$$

i = 3:

$j \div 9 : 4$

$$A = [1, 3, 4, 7, 8, 9, 12, 16, 10, 14];$$

i = 4:

$j \div 9 : 5$

$$A = [1, 3, 4, 7, 8, 9, 10, 12, 16, 14];$$

i = 5:

$j \div 9 : 6$

$$A = [1, 3, 4, 7, 8, 9, 10, 12, 14, 16];$$

Заметим, что массив уже отсортирован, то есть в следующих итерациях не будет перестановок (обменов). Флаг, который за этим следит, даст нам об этом знать и мы завершим работу алгоритма. Но предположим, что мы этого не сделали, и доведём работу на всякий случай до конца.

i = 6:

$j \div 9 : 7$

$$A = [1, 3, 4, 7, 8, 9, 10, 12, 14, 16];$$

i = 7:

$j \div 9 : 8$

$$A = [1, 3, 4, 7, 8, 9, 10, 12, 14, 16];$$

i = 8:

$j \div 9 : 9$

$$A = [1, 3, 4, 7, 8, 9, 10, 12, 14, 16].$$

Больше итераций не будет, но очевидно, что в конце массива теперь лежит наибольший элемент и массив уже отсортирован.

Но что если мы хотим получить невозрастающую последовательность? Тогда нам всего лишь нужно изменить порядок j у i итерации, то есть сделать его не от $n - 1$ до $i + 1$, а $j \div 0 : n - i - 1$ и сравнивать числа $A[j]$ и $A[j + 1]$. Но есть ещё более лёгкий способ - просто поменять знак сравнения. Всё бы ничего, только вот после смены знака сортировка перестаёт быть пузырьковой, так как на верх поднимается самый «тяжёлый» элемент (наибольший), что не похоже на всплывающий пузырёк; тем не менее алгоритм работает корректно.

Код сортировки

Листинг 1: bubbleSort

```
1       public static void bubbleSort(int[] A, boolean reverse) {
2           for (int i = 0; i <= A.length - 2; ++i) {
3               boolean swapped = false;
4               for (int j = A.length - 1; j >= i + 1; --j) {
5                   if (reverse ? A[j] > A[j - 1] : A[j] < A[j - 1])
                        {
6                       int tmp = A[j];
7                       A[j] = A[j - 1];
8                       A[j - 1] = tmp;
9
10                      swapped = true;
11                  }
12              }
13
14              if (!swapped) break;
15          }
16      }
```

Нам передаётся ссылка на массив, а также направление, в котором нужно отсортировать. Мы добавляем переменную swapped означающую были ли обмены в i итерации; с помощью неё мы выходим из цикла, если массив уже отсортирован.

В первом цикле мы меняем начало, до которого бежит значение j, а во втором цикле изменяем само значение j.

Исходный код

Представим весь код Main.java для тестирования алгоритма.

Листинг 2: Main

```
1   public class Main {
2       public static void bubbleSort(int[] A, boolean reverse) {
3           for (int i = 0; i <= A.length - 2; ++i) {
4               boolean swapped = false;
5               for (int j = A.length - 1; j >= i + 1; --j) {
6                   if (reverse ? A[j] > A[j - 1] : A[j] < A[j - 1])
                        {
7                       int tmp = A[j];
8                       A[j] = A[j - 1];
9                       A[j - 1] = tmp;
10
11                      swapped = true;
```

```
                }
            }

            if (!swapped) break;
        }
    }

    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 = {8, 3, 12, 16, 10, 4, 1, 9, 7, 14};

        print(A);
        bubbleSort(A, false);
        print(A);
    }
}
```

Сложность

Во-первых, заметим, что мы не используем дополнительную память, не считая некоторых переменных, поэтому сразу можно сказать, что пространственная сложность пузырьковой сортировки состовляет $O(1)$.

Рассмотрим лучший случай - на входе у нас был сразу отсортированный массив. В таком случае нам придётся выполнить только $i = 0$ итерацию, и после того как мы поймём, что у нас не было обменов, выйдем из цикла. То есть один раз мы прошлись линейно по массиву. А это значит, что временная сложность пузырьковой сортировки в лучшем случае равна $\Omega(n)$.

Рассмотрим средний случай - на входе обычный массив (несортированный). В этом случае в первом цикле будет работать второй цикл. Да, с помощью swapped мы сократим какое-то количество итераций, но качественно на асимптотику это не повлияет, следовательно временная сложность пузырьковой сортировки в среднем случае равна $\Theta(n^2)$.

Рассмотрим худший случай - на входе отсортированный в обратную сторону массив. В таком случае нам придётся честно проделать два цикла без выходов, поэтому временная сложность пузырьковой сортировки в худшем случае равна $O(n^2)$. Коэффициенты, скрывающиеся за О-большим больше, чем в среднем случае.

Временная сложность			Пространственная сложность
Худший	Средний	Лучший	Худший
$O(n^2)$	$\Theta(n^2)$	$\Omega(n)$	$O(1)$

Таблица 1: **Резюме**

Комментарии

Пузырьковая сортировка достаточно проста в понимании и не использует дополнительной памяти, но работает, как видно из таблицы, очень медленно для алгоритмов сортировки, поэтому стоит сказать, что данный алгоритм можно использовать только для учебных целей.