Порядковая статика

Слёлин А.В.

18 мая, 2024 г.

Описание алгоритма

i-ой порядковой статикой множества, состоящего из n элементов, называется i-ый элемент в порядке возрастания. Например, минимум такого множества - это первая порядковая статика, а его максимум - это n-ая порядковая статика. Медиана неформально обозначает середину множества. При $i = \lfloor (n+1)/2 \rfloor$ - нижняя медиана, а при $i = \lceil (n+1)/2 \rceil$ - верхняя медиана.

Нахождение порядковой статики можно решить за время $O(n \lg n)$. Для этого достаточно выполнить сортировку, а затем извлечь элемент выходного массива с индексом i. Однако есть и более быстрые алгоритмы нахождения порядковой статики.

Заметим, что если нам нужно найти просто максимум и (или) минимум, то его поиск очень прост: линейно сравниваем со всеми элементами. Отдельный код показан в Листинге 1: getMinMax.

Общая задача выбора оказывается более сложной, чем задача поиска максимума (минимума). Однако время решения в асимптотическом пределе ведёт себя одинаково - $\Theta(n)$. Алгоритм основан на парадигме «разделяй и властвуй» и разработан по аналогии с алгоритмом быстрой сортировки. Однако в отличие от быстрой сортировки, в которой рекурсивно обрабатываются обе части разбиения, этот алгоритм работает только с одной частью.

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

Пусть дан подмассив $A[p..r]$. Необходимо найти i-ую статику в данном подмассиве. Разобъём его на два подмассива с помощью функции разбиения, которую мы использовали в быстрой сортировке: $L = A[p..q-1]$ и $R = A[q+1..r]$, q - опорный элемент. Тогда номер статики элемента с индексом q равен $k = q-p+1$. Если $i = k$, то мы уже нашли нашу порядковую статику под индексом q. Если $i < k$, тогда получается, что наша порядковая статика находится уже в L, а если $i > k$, тогда в R, причём для R это уже будет не i-ая статика, а $i-k$-ая статика.

Пример

Пусть массив $A = [8, 3, 12, 16, 10, 4, 1, 9, 7, 14]$. Необходимо найти 5-ю порядковую статику. count будет обозначать номер вызова рекурсии, tmp - рандомное число, которое указывает на элемент (выделено красным цветом), относительно которого будет производиться разбиение массива input на подмассивы L (выделено синим цветом) и R (выделено жёлтым цветом).

count = 0:

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

count = 1:

$$\text{input} = [8, 3, 12, 16, 10, 4, 1, 9, 7, 14];$$
$$\text{tmp} = \text{RANDOM}(0, 9) = 4;$$
$$\text{modified} = [8, 3, 4, 1, 9, 7, 10, 12, 16, 14];$$
$$q = 6, \quad k = 6 - 0 + 1 = 7,$$

Мы нашли 7-ю статику, а нужно 4-ю, значит вызываем процедуру на массиве L.

count = 2:

$$\text{input} = [8, 3, 4, 1, 9, 7];$$
$$\text{tmp} = \text{RANDOM}(0, 5) = 5;$$
$$\text{modified} = [3, 4, 1, 7, 8, 9];$$
$$q = 3, \quad k = 3 - 0 + 1 = 4,$$

Мы нашли 4-ю статику, а нужно 5-ю, значит вызываем процедуру на массиве R, но там мы уже ищем 1-ю статику.

count = 3:

$$\text{input} = [8, 9];$$
$$\text{tmp} = \text{RANDOM}(0, 1) = 0,$$
$$\text{modified} = [8, 9];$$
$$q = 0, \quad k = 0 - 0 + 1 = 1,$$

Мы нашли 1-ю статику для $[8, 9]$, значит для A мы нашли 5-ю статику. Алгоритм работает корректно.

Код

Если необходим просто поиск максимума и (или) минимума лучше использовать такой код.

Листинг 1: getMinMax

```java
    public static int[] getMinMax(int[] A) {
        int max = Integer.MIN_VALUE, min = Integer.MAX_VALUE;
        for (int i = 0; i < A.length; ++i) {
            max = Math.max(max, A[i]);
            min = Math.min(min, A[i]);
        }

        return new int[]{min, max};
    }
```

Листинг 2: getStatics

```
public static int getStatics(int[] A, int i) throws Exception {
    if (i < 0 || i > A.length)
        throw new Exception("Неверный ввод данных.");

    return randomizedSelect(A, 0, A.length - 1, i);
}
```

Листинг 3: randomizedSelect

```
public static int randomizedSelect(int[] A, int p, int r, int i) {
    if (p == r)
        return A[p];

    int q = randomizedPartition(A, p, r);
    int k = q - p + 1;
    if (i == k)
        return A[q];
    else if (i < k)
        return randomizedSelect(A, p, q - 1, i);
    else
        return randomizedSelect(A, q + 1, r, i - k);
}
```

Листинг 4: randomizedPartition

```
public static int randomizedPartition(int[] A, int p, int r) {
    int i = ThreadLocalRandom.current().nextInt(p, r + 1);

    int tmp = A[i];
    A[i] = A[r];
    A[r] = tmp;

    return partition(A, p, r);
}
```

Листинг 5: partition

```
public static int partition(int[] A, int p, int r) {
    int x = A[r];
    int i = p - 1;
    for (int j = p; j < r; ++j) {
```

```java
            if (A[j] <= x) {
                int tmp = A[++i];
                A[i] = A[j];
                A[j] = tmp;
            }
        }

        int tmp = A[++i];
        A[i] = A[r];
        A[r] = tmp;

        return i;
    }
```

Исходный код

Представим весь код Main.java для тестирования алгоритма.

Листинг 6: Main

```java
import java.util.concurrent.ThreadLocalRandom;

public class Main {
    public static int[] getMinMax(int[] A) {
        int max = Integer.MIN_VALUE, min = Integer.MAX_VALUE;
        for (int i = 0; i < A.length; ++i) {
            max = Math.max(max, A[i]);
            min = Math.min(min, A[i]);
        }

        return new int[]{min, max};
    }

    public static int partition(int[] A, int p, int r) {
        int x = A[r];
        int i = p - 1;
        for (int j = p; j < r; ++j) {
            if (A[j] <= x) {
                int tmp = A[++i];
                A[i] = A[j];
                A[j] = tmp;
            }
        }

        int tmp = A[++i];
        A[i] = A[r];
        A[r] = tmp;
```

```java

        return i;
    }

    public static int randomizedPartition(int[] A, int p, int r)
         {
        int i = ThreadLocalRandom.current().nextInt(p, r + 1);

        int tmp = A[i];
        A[i] = A[r];
        A[r] = tmp;

        return partition(A, p, r);
    }

    public static int randomizedSelect(int[] A, int p, int r,
        int i) {
        if (p == r)
            return A[p];

        int q = randomizedPartition(A, p, r);
        int k = q - p + 1;
        if (i == k)
            return A[q];
        else if (i < k)
            return randomizedSelect(A, p, q - 1, i);
        else
            return randomizedSelect(A, q + 1, r, i - k);
    }

    public static int getStatics(int[] A, int i) throws
        Exception {
        if (i < 0 || i > A.length)
            throw new Exception("Неверный ввод данных.");

        return randomizedSelect(A, 0, A.length - 1, i);
    }

    public static void main(String[] args) throws Exception {
        int[] A = new int[]{8, 3, 12, 16, 10, 4, 1, 9, 7, 14};

        System.out.println(getStatics(A, 4));
    }
}
```

Сложность

Во-первых, заметим, что мы не используем дополнительную память, исключая некоторые переменные, поэтому пространственная сложность алгоритма порядковой статики $O(1)$.

Рассмотрим случай когда на вход подаётся отсортированный массив. Тогда метод partition будет работать за время $\Omega(n)$.

Теперь рассмотрим случай когда на вход подаётся неотсортированный массив. Тогда в асимптотическом пределе временная сложность алгоритма порядковой статики равна $\Theta(n)$.

И последний случай - на вход подаётся отсортированный в обратную сторону массив. Тогда время работы существенно увеличивается и равно $O(n^2)$.

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

Таблица 1: **Резюме**

Комментарии

Алгоритм порядковой статики очень хорош и может быть использован из-за своей скорости в среднем случае и памяти $O(1)$. То есть асимптотически мы будем искать порядковую статику как и обычный минимум или максимум - $\Theta(n)$.