Theory: Finding max and min in arrays

18 minutes 0 / 5 problems solved Skip this topic Start practicing

444 users solved this topic. Latest completion was about 2 hours ago.

Sometimes we need to find the maximum or minimum value in an array. The array can store integer numbers, floating-point numbers, characters, or even more complex objects such as strings or dates. The only restriction is that the array elements should be comparable.

The problem of finding the min/max element in an array can be formulated in different ways:

- find the value of the min/max element;
- find the index of the min/max element;
- find the first/last min/max element;
- and so on.

Current topic:

Finding max and min in arrays

Topic depends on:

× Algorithms in Java

§1. Finding the index of the maximum in an array

Let's write a method to find the maximum element in an array of ints, more specifically the index of the element.

The most natural approach is to take the index of the first element as the index of maximum and to compare the value with this index with the values of all other elements. Once we find a greater element we save its index and continue the comparisons with this. Finally, we find the index of the maximum.

The algorithm performs $n-1$ comparisons. It has the complexity $O(n)$. It's impossible to reduce the complexity If we do not have more information about the elements of the array. This is because the maximum can be either the first element or the last one or any other.

Below is an implementation of the algorithm in Java:

```
public static int findIndexOfMax(int[] numbers) {
    int index = 0;
    for (int i = 1; i < numbers.length; i++) {
        if (numbers[i] > numbers[index]) {
            index = i;
        }
    }
    return index;
}
```

The method takes an array of ints and returns the index of the max element. The input array can be either ordered or unordered.

Let's describe the algorithm with words to better understand it:

1. suppose the number with the index 0 (the first element) is a candidate to be the maximum;
2. in the `for` loop, we consider the numbers of the array from the second to the last one and do (3). When the end of the array is reached, we perform (5).
3. if the current number is greater than the candidate, we save its index (found a new candidate)
4. go to step (2);
5. return the index of a found candidate

Let's call the method passing various arrays:

```
findIndexOfMax(new int[] {1, 5, 3, 2, 3}); // 1
findIndexOfMax(new int[] {6, 8, 7, 9})); // 3
findIndexOfMax(new int[] {99}); // 0, the algorithm processes single-
element arrays as well
```

Table of contents:

It may seem, the algorithm works well. But what if the array contains several numbers which are equals to the maximum? Also, the array can be empty.

Actually, the presented algorithm finds the first maximum in the array:

```
findIndexOfMax(new int[] {3, 2, 8, 8, 1}); // 2
```

Sometimes we may need the first one, sometimes the last one is needed. It depends on the problem you are solving. Anyway, the time complexity of the algorithm will be the $O(n)$, where n is the size of an array.

If a passed array is empty, the method returns `0` as the index of the maximum number.

```
findIndexOfMax(new int[] {}); // 0
```

It may be better to return the special value `-1`. It means no maximum found in the array. Let's add the special condition in the method:

```
if (numbers.length == 0) {
    return -1;
}
```

So, we can write a method that works for empty, single and multi-element arrays to find the first/last min/max value in an array. All these methods work in $O(n)$ for an array of size n.

§2. Finding the value of the min/max element in a sorted array

We can find the value of the min/max element more efficient if the given array is sorted in ascending or descending order. If the array is sorted in ascending order, the first element is the minimum, the last element is the maximum:

```
int[] sortedArray = { 1, 2, 3, 4, 6, 8, 9 }; // sortedArray[0] - min, sortedArray[
6] - max
```

In this case, we can find the min/max element in $O(1)$.

So, if we have additional information about an input array, we can search for the minimum/maximum more efficiently. But in general case, the time complexity is $O(n)$.

Report a typo

31 users liked this theory. **0** didn't like it. **What about you?**

😍 🙂 😐 🙁 😡

Start practicing

Comments (0) Hints (0) Useful links (0) Show discussion