Theory: Counting sort in Java

17 minutes 0 / 3 problems solved

Counting sort is not a comparison-based algorithm for sorting small integers. The algorithm counts the number of occurrences of distinct numbers in the input array and then output the values from the minimum to the maximum according to their counts. To store counts, the algorithm uses another array.

The algorithm can be **stable** and **unstable**. We will consider both versions written in Java.

§1. Implementing unstable counting sort

Given below is an implementation of unstable counting sort in Java:

```
public static int[] countingSort(int[] numbers) {
    int maxVal = 10; // we suppose the maximum is 10
    int k = maxVal + 1; // the length of the array containing counts

int[] counts = new int[k]; // it stores 11 zeros with indexes from 0 to k-1

    /* in this loop we count distinct numbers in the input array */
    for (int i = 0; i < numbers.length; i++) {
        counts[numbers[i]]++;
    }



    int pos = 0; // a position in the numbers array



    /* in this loop we modify the input array to make it sorted */

    for (int num = 0; num < k; num++) { // get the next element

        int count = counts[num]; // get the count of the element

        while (count > 0) {

            numbers[pos] = num; // write it in the numbers array

            pos++;

            count--;

        }

    }



    return numbers;

}
```

We assumed that the maximum element is 10, but in fact, the best strategy can be to pass the possible maximum as a parameter to the method or to determine it by traversing the input array with the time complexity $O(n)$.

Here is an example of how the method can be used:

```
countingSort(new int[] {2, 0, 3, 5, 0, 3, 3, 1}); // [0, 0, 1, 2, 3, 3, 3, 5]
```

§2. Implementing stable counting sort

Current topic:

Counting sort in Java

Topic depends on:

✓ Counting sort

✗ Algorithms in Java

Table of contents:

This algorithm also can be stable that may be required when sorting keys. To make it stable, we will calculate cumulative counts.

```
  public static int[] stableCountingSort(int[] numbers, int max) {
      int k = max + 1; // the length of the array containing counts

 int[] counts = new int[k]; // it stores counts with indexes from 0 to k-1

      for (int i = 0; i < numbers.length; i++) {
          counts[numbers[i]]++;
      }

      for (int i = 1; i < counts.length; i++) {

          counts[i] = counts[i - 1] + counts[i]; // cumulative counts

      }



      int[] sortedNumbers = new int[numbers.length];



      for (int i = numbers.length - 1; i >= 0; i-
-) {  // go through input array in backward

    int rightmostIndex = counts[numbers[i]] - 1; // get the rightmost index

          sortedNumbers[rightmostIndex] = numbers[i];

          counts[numbers[i]]-
-; // decrease the index to calculate the previous occurrence

      }



      return sortedNumbers;

  }
```

The method takes an array of ints and a possible maximum value as parameters. It returns a new sorted array.

Here is how this algorithm can be used:

```

stableCountingSort(new int[] {2, 0, 3, 5, 0, 3, 3, 1}, 10); // [0, 0, 1, 2, 3, 3,
3, 5]
```

As you can see, it can sort the same array of ints. If you do not understand the algorithm, try to start it in the debug mode and sort different arrays.

Report a typo

21 users liked this theory. **2** didn't like it. **What about you?**

Start practicing

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