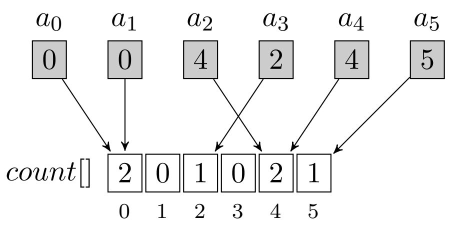
Chapter 2

Counting elements

A numerical sequence can be stored in an array in various ways. In the standard approach, the consecutive numbers $a_0, a_1, \ldots, a_{n-1}$ are usually put into the corresponding consecutive indexes of the array:

$$A[0] = a_0 \quad A[1] = a_1 \quad \ldots \quad A[n-1] = a_{n-1}$$

We can also store the data in a slightly different way, by making an array of counters. Each number may be counted in the array by using an index that corresponds to the value of the given number.



Notice that we do not place elements directly into a cell; rather, we simply count their occurrences. It is important that the array in which we count elements is sufficiently large. If we know that all the elements are in the set $\{0, 1, \ldots, x\}$, then the array used for counting should be of size $x + 1$.

The limitation here may be available memory. Usually, we are not able to create arrays of 10^9 integers, because this would require more than one gigabyte of available memory.

Counting the number of negative integers can be done in two ways. The first method is to add some big number to each value: so that, all values would be greater than or equal to zero. That is, we shift the representation of zero by some arbitrary amount to accommodate all the negative numbers we need. In the second method, we simply create a second array for counting negative numbers.

2.1. Exercises

Problem: You are given an integer m such that $(1 \leqslant m \leqslant 1\,000\,000)$ and a non-empty, zero-indexed array A of n integers: $a_0, a_1, \ldots, a_{n-1}$ such that $(0 \leqslant a_i \leqslant m)$. Count the number of occurrences of the values $0, 1, \ldots, m$.

Solution: The simple way is to iterate through the whole array, searching for each value $0, 1, \ldots, m$ separately, but that produces a time complexity of $O(n \cdot m)$. The better approach is to count the elements in the array.

2.1: Counting elements.

```
def counting(A, m):
    n = len(A)
    count = [0] * (m + 1)
    for k in xrange(n):
        count[A[k]] += 1
    return count
```

With this approach, the time complexity is $O(n + m)$.