Burnside's lemma / Pólya enumeration theorem

Table of Contents

Burnside's lemma

Burnside's lemma was formulated and proven by **Burnside** in 1897, but historically it was already discovered in 1887 by **Frobenius**, and even earlier in 1845 by **Cauchy**. Therefore it is also sometimes named the **Cauchy-Frobenius lemma**.

Burnside's lemma allows us to count the number of equivalence classes in sets, based on internal symmetry.

Objects and representations

We have to clearly distinguish between the number of objects and the number of representations.

Different representations can correspond to the same objects, but of course any representation corresponds to exactly one object. Consequently the set of all representations is divided into equivalence classes. Our task is to compute the number of objects, or equivalently, the number of equivalence classes. The following example will make the difference between object and representation clearer.

Example: coloring of binary trees

Suppose we have the following problem. We have to count the number of ways to color a rooted binary tree with n vertices with two colors, where at each vertex we do not distinguish between the left and the right children.

Here the set of objects is the set of different colorings of the tree.

We now define the set of representations. A representation of a coloring is a function $f(v)$, which assigns each vertex a color (here we use the colors 0 and 1). The set of representations is the set containing all possible functions of this kind, and its size is obviously equal to 2^n.

At the same time we introduce a partition of this set into equivalence classes.

For example, suppose $n = 3$, and the tree consists of the root 1 and its two children 2 and 3. Then the following functions f_1 and f_2 are considered equivalent.

$$\begin{aligned} f_1(1) &= 0 \quad & f_2(1) &= 0 \\ f_1(2) &= 1 \quad & f_2(2) &= 0 \\ f_1(3) &= 0 \quad & f_2(3) &= 1 \end{aligned}$$

Invariant permutations

Why do these two function f_1 and f_2 belong to the same equivalence class? Intuitively this is understandable - we can rearrange the children of vertex 1, the vertices 2 and 3, and after such a transformation of the function f_1 it will coincide with f_2.

But formally this means that there exists an **invariant permutation** π (i.e. a permutation which does not change the object itself, but only its representation), such that:

$$f_2 \pi \equiv f_1$$

So starting from the definition of objects, we can find all the invariant permutations, i.e. all permutations which do not change the object when applying the permutation to the representation. Then we can check whether two functions f_1 and f_2 are equivalent (i.e. if they correspond to the same object) by checking the condition $f_2 \pi \equiv f_1$ for each invariant permutation (or equivalently $f_1 \pi \equiv f_2$). If at least one permutation is

found for which the condition is satisfied, then f_1 and f_2 are equivalent, otherwise they are not equivalent.

Finding all such invariant permutations with respect to the object definition is a key step for the application of both Burnside's lemma and the Pólya enumeration theorem. It is clear that these invariant permutations depend on the specific problem, and their finding is a purely heuristic process based on intuitive considerations. However in most cases it is sufficient to manually find several "basic" permutations, with which all other permutations can be generated (and this part of the work can be shifted to a computer).

It is not difficult to understand that invariant permutations form a **group**, since the product (composition) of invariant permutations is again an invariant permutation. We denote the **group of invariant permutations** by G.

The statement of the lemma

For the formulation of the lemma we need one more definition from algebra. A **fixed point** f for a permutation π is an element that is invariant under this permutation: $f \equiv f\pi$. For example in out example the fixed points are those functions f, which correspond to colorings that do not change when the permutation π is applied to them (i.e. they do not change in the formal sense of the equality of functions). We denote by $I(\pi)$ the **number of fixed points** for the permutation π.

Then **Burnside's lemma** goes as follows: the number of equivalence classes is equal to the sum of the numbers of fixed points with respect to all permutations from the group G, divided by the size of this group:

$$|\text{Classes}| = \frac{1}{|G|} \sum_{\pi \in G} I(\pi)$$

Although Burnside's lemma itself is not so convenient to use in practice (it is unclear how to quickly look for the value $I(\pi)$, it most clearly reveals the mathematical essence on which the idea of calculating equivalence classes is based.

Proof of Burnside's lemma

The proof of Burnside's lemma described here is not important for the practical applications, so it can be skipped on the first reading.

The proof here is the simplest known, and does not use group theory. The proof was published by Kenneth P. Bogart in 1991.

We need to prove the following statement:

$$|\text{Classes}| \cdot |G| = \sum_{\pi \in G} I(\pi)$$

The value on the right side is nothing more than the number of "invariant pairs" (f, π), i.e. pairs such that $f\pi \equiv f$. It is obvious that we can change the order of

summation. We let the sum iterate over all elements f and sum over the values $J(f)$ - the number of permutations for which f is a fixed point.

$$|\text{Classes}| \cdot |G| = \sum_f J(f)$$

To prove this formula we will compose a table with columns labeled with all functions f_i and rows labeled with all permutations π_j. And we fill the cells with $f_i \pi_j$. If we look at the columns in this table as sets, then some of them will coincide, and this means that the corresponding functions f for these columns are also equivalent. Thus the numbers of different (as sets) columns is equal to the number of classes. Incidentally, from the standpoint of group theory, the column labeled with f_i is the orbit of this element. For equivalent elements the orbits coincides, and the number of orbits gives exactly the number of classes.

Thus the columns of the table decompose into equivalence classes. Let us fix a class, and look at the columns in it. First, note that these columns can only contain elements f_i of the equivalence class (otherwise some permutation π_j moved one of the functions into a different equivalence class, which is impossible since we only look at invariant permutations). Secondly each element f_i will occur the same number of times in each columns (this also follows from the fact that the columns correspond to equivalent elements). From this we can conclude, that all the columns within the same equivalence class coincide with each other as multisets.

Now fix an arbitrary element f. On the one hand, it occurs in its column exactly $J(f)$ times (by definition). On the other hand, all columns within the same equivalence class are the same as multisets. Therefore within each column of a given equivalence class any element g occurs exactly $J(g)$ times.

Thus if we arbitrarily take one column from each equivalence class, and sum the number of elements in them, we obtain on one hand $|\mathrm{Classes}| \cdot |G|$ (simply by multiplying the number of columns by the number of rows), and on the other hand the sum of the quantities $J(f)$ for all f (this follows from all the previous arguments):

$$|\mathrm{Classes}| \cdot |G| = \sum_f J(f)$$

Pólya enumeration theorem

The Pólya enumeration theorem is a generalization of Burnside's lemma, and it also provides a more convenient tool for finding the number of equivalence classes. It should noted that this theorem was already discovered before Pólya by Redfield in 1927, but his publication went unnoticed by mathematicians. Pólya independently came to the same results in 1937, and his publication was more successful.

Here we discuss only a special case of the Pólya enumeration theorem, which will turn out very useful in

practice. The general formula of the theorem will not be discussed.

We denote by $C(\pi)$ the number of cycles in the permutation π. Then the following formula (a **special case of the Pólya enumeration theorem**) holds:

$$|\text{Classes}| = \frac{1}{|G|} \sum_{\pi \in G} k^{C(\pi)}$$

k is the number of values that each representation element can take, in the case of the coloring of a binary tree this would be $k = 2$.

Evidence

This formula is a direct consequence of Burnside's lemma. To get it, we just need to find an explicit expression for $I(\pi)$, which appears in the lemma. Recall, that $I(\pi)$ is the number of fixed points in the permutation π.

Thus we consider a permutation π and some element f. During the application of π, the elements in f move via the cycles in the permutation. Since the result should obtain $f \equiv f\pi$, the elements touched by one cycle must all be equal. At the same time different cycles are independent. Thus for each permutation cycle π we can choose one value (among k possible) and thus we get the number of fixed points:

$$I(\pi) = k^{C(\pi)}$$

Application: Coloring necklaces

The problem "Necklace" is one of the classical combinatorial problems. The task is to count the number of different necklaces from n beads, each of which can be painted in one of the k colors. When comparing two necklaces, they can be rotated, but not reversed (i.e. a cyclic shift is permitted).

In this problem we can immediately find the group of invariant permutations:

$$\begin{aligned}\pi_0 &= 123\ldots n\\ \pi_1 &= 23\ldots n1\\ \pi_2 &= 3\ldots n12\\ &\ldots\\ \pi_{n-1} &= n123\ldots\end{aligned}$$

Let us find an explicit formula for calculating $C(\pi_i)$. First we note, that the permutation π_i has at the j-th position the value $i+j$ (taken modulo n). If we check the cycle structure for π_i. We see that 1 goes to $1+i$, $1+i$ goes to $1+2i$, which goes to $1+3i$, etc., until we come to a number of the form $1+kn$. Similar statements can be mode for the remaining elements. Hence we see that all cycles have the same length, namely $\frac{\operatorname{lcm}(i,n)}{i} = \frac{n}{\gcd(i,n)}$. Thus the number of cycles in π_i will be equal to $\gcd(i,n)$.

Substituting these values into the Pólya enumeration theorem, we obtain the solution:

$$\frac{1}{n}\sum_{i=1}^{n} k^{\gcd(i,n)}$$

You can leave this formula in this form, or you can simplify it even more. Let transfer the sum so that it iterates over all divisors of n. In the original sum there will be many equivalent terms: if i is not a divisor of n, then such a divisor can be found after computing $\gcd(i,n)$. Therefore for each divisor $d \mid n$ its term $k^{\gcd(d,n)} = k^d$ will appear in the sum multiple times, i.e. the answer to the problem can be rewritten as

$$\frac{1}{n}\sum_{d \mid n} C_d k^d,$$

where C_d is the number of such numbers i with $\gcd(i,n) = d$. We can find an explicit expression for this value. Any such number i has the form $i = dj$ with $\gcd(j, n/d) = 1$ (otherwise $\gcd(i,n) > d$). So we can count the number of j with this behavior. Euler's phi function gives us the result $C_d = \phi(n/d)$, and therefore we get the answer:

$$\frac{1}{n}\sum_{d \mid n} \phi\left(\frac{n}{d}\right) k^d$$

Application: Coloring a torus

Quite often we cannot obtain an explicit formula for the number of equivalence classes. In many problems the

number of permutations in a group can be too large for manual calculations and it is not possible to compute analytically the number of cycles in them.

In that case we should manually find several "basic" permutations, so that they can generate the entire group G. Next we can write a program that will generate all permutations of the group G, count the number of cycles in them, and compute the answer with the formula.

Consider the example of the problem for coloring a torus. There is a checkered sheet of paper $n \times m$ ($n < m$), some of the cells are black. Then a cylinder is obtained from this sheet by gluing together the two sides with lengths m. Then a torus is obtained from the cylinder by gluing together the two circles (top and bottom) without twisting. The task is to compute the number of different colored tori, assuming that we cannot see the glued lines, and the torus can be turned and turned.

We again start with a piece of $n \times m$ paper. It is easy to see that the following types of transformations preserve the equivalence class: a cyclic shift of the rows, a cyclic shift of the columns, and a rotation of the sheet by 180 degrees. It is also easy to see, that these transformations can generate the entire group of invariant transformations. If we somehow number the cells of the paper, then we can write three permutations p_1, p_2, p_3 corresponding to these types of transformation.

Next it only remains to generate all permutations obtained as a product. It is obvious that all such permutations have the form $p_1^{i_1} p_2^{i_2} p_3^{i_3}$ where $i_1 = 0 \dots m-1, i_2 = 0 \dots n-1, i_3 = 0 \dots 1$.

Thus we can write the implementations to this problem.

```
using Permutation = vector<int>;

void operator*=(Permutation& p, Permutation co
    Permutation copy = p;
    for (int i = 0; i < p.size(); i++)
        p[i] = copy[q[i]];
}

int count_cycles(Permutation p) {
    int cnt = 0;
    for (int i = 0; i < p.size(); i++) {
        if (p[i] != -1) {
            cnt++;
            for (int j = i; p[j] != -1;) {
                int next = p[j];
                p[j] = -1;
                j = next;
            }
        }
    }
    return cnt;
}

int solve(int n, int m) {
    Permutation p(n*m), p1(n*m), p2(n*m), p3(n
```

```
    for (int i = 0; i < n*m; i++) {
        p[i] = i;
        p1[i] = (i % n + 1) % n + i / n * n;
        p2[i] = (i / n + 1) % m * n + i % n;
        p3[i] = (m - 1 - i / n) * n + (n - 1 -
    }

    set<Permutation> s;
    for (int i1 = 0; i1 < n; i1++) {
        for (int i2 = 0; i2 < m; i2++) {
            for (int i3 = 0; i3 < 2; i3++) {
                s.insert(p);
                p *= p3;
            }
            p *= p2;
        }
        p *= p1;
    }

    int sum = 0;
    for (Permutation const& p : s) {
        sum += 1 << count_cycles(p);
    }
    return sum / s.size();
}
```

(c) 2014-2019 translation by http://github.com/e-maxx-eng