Last update: November 14, 2023 Translated From: e-maxx.ru

Finding repetitions

Given a string s of length n.

A **repetition** is two occurrences of a string in a row. In other words a repetition can be described by a pair of indices $i < j$ such that the substring $s[i \dots j]$ consists of two identical strings written after each other.

The challenge is to **find all repetitions** in a given string s. Or a simplified task: find **any** repetition or find the **longest** repetition.

The algorithm described here was published in 1982 by Main and Lorentz.

Example

Consider the repetitions in the following example string:

$$acababaee$$

The string contains the following three repetitions:

- $s[2 \dots 5] = abab$
- $s[3 \dots 6] = baba$
- $s[7 \dots 8] = ee$

Another example:

$$abaaba$$

Here there are only two repetitions

- $s[0 \dots 5] = abaaba$
- $s[2 \dots 3] = aa$

Number of repetitions

In general there can be up to $O(n^2)$ repetitions in a string of length n. An obvious example is a string consisting of n times the same letter, in this case any substring of even length is a repetition. In general any periodic string with a short period will contain a lot of repetitions.

On the other hand this fact does not prevent computing the number of repetitions in $O(n \log n)$ time, because the algorithm can give the repetitions in compressed form, in groups of several pieces at once.

There is even the concept, that describes groups of periodic substrings with tuples of size four. It has been proven that we the number of such groups is at most linear with respect to the string length.

Also, here are some more interesting results related to the number of repetitions:

- The number of primitive repetitions (those whose halves are not repetitions) is at most $O(n \log n)$.
- If we encode repetitions with tuples of numbers (called Crochemore triples) $(i,\ p,\ r)$ (where i is the position of the beginning, p the length of the repeating substring, and r the number of repetitions), then all repetitions can be described with $O(n \log n)$ such triples.
- Fibonacci strings, defined as

  $$t_0 = a,$$

  $$t_1 = b,$$

  $$t_i = t_{i-1} + t_{i-2},$$

  are "strongly" periodic. The number of repetitions in the Fibonacci string f_i, even in the compressed with Crochemore triples, is $O(f_n \log f_n)$. The number of primitive repetitions is also $O(f_n \log f_n)$.

Main-Lorentz algorithm

The idea behind the Main-Lorentz algorithm is **divide-and-conquer**.

It splits the initial string into halves, and computes the number of repetitions that lie completely in each halve by two recursive calls. Then comes the difficult part. The algorithm finds all repetitions starting in the first half and ending in the second half (which we will call **crossing repetitions**). This is the essential part of the Main-Lorentz algorithm, and we will discuss it in detail here.

The complexity of divide-and-conquer algorithms is well researched. The master theorem says, that we will end up with an $O(n \log n)$ algorithm, if we can compute the crossing repetitions in $O(n)$ time.

Search for crossing repetitions

So we want to find all such repetitions that start in the first half of the string, let's call it u, and end in the second half, let's call it v:

$$s = u + v$$

Their lengths are approximately equal to the length of s divided by two.

Consider an arbitrary repetition and look at the middle character (more precisely the first character of the second half of the repetition). I.e. if the repetition is a substring $s[i \dots j]$, then the middle character is $(i + j + 1)/2$.

We call a repetition **left** or **right** depending on which string this character is located - in the string u or in the string v. In other words a string is called left, if the majority of it lies in u, otherwise we call it right.

We will now discuss how to find **all left repetitions**. Finding all right repetitions can be done in the same way.

Let us denote the length of the left repetition by $2l$ (i.e. each half of the repetition has length l). Consider the first character of the repetition falling into the string v (it is at position $|u|$ in the string s). It coincides with the character l positions before it, let's denote this position $cntr$.

We will fixate this position $cntr$, and **look for all repetitions at this position $cntr$**.

For example:

$$c \ \underset{cntr}{a} \ c \mid a \, d \, a$$

The vertical lines divides the two halves. Here we fixated the position $cntr = 1$, and at this position we find the repetition $caca$.

It is clear, that if we fixate the position $cntr$, we simultaneously fixate the length of the possible repetitions: $l = |u| - cntr$. Once we know how to find these repetitions, we will iterate over all possible values for $cntr$ from 0 to $|u| - 1$, and find all left crossover repetitions of length $l = |u|, \ |u| - 1, \ \dots, 1$.

Criterion for left crossing repetitions

Now, how can we find all such repetitions for a fixated $cntr$? Keep in mind that there still can be multiple such repetitions.

Let's again look at a visualization, this time for the repetition $abcabc$:

$$\overbrace{a}^{l_1} \ \overbrace{\underset{cntr}{b} \ c}^{l_2} \ \overbrace{a}^{l_1} \mid \overbrace{b \, c}^{l_2}$$

Here we denoted the lengths of the two pieces of the repetition with l_1 and l_2: l_1 is the length of the repetition up to the position $cntr - 1$, and l_2 is the length of the repetition from $cntr$ to the end of the half of the repetition. We have $2l = l_1 + l_2 + l_1 + l_2$ as the total length of the repetition.

Let us generate **necessary and sufficient** conditions for such a repetition at position $cntr$ of length $2l = 2(l_1 + l_2) = 2(|u| - cntr)$:

- Let k_1 be the largest number such that the first k_1 characters before the position $cntr$ coincide with the last k_1 characters in the string u:

$$u[cntr - k_1 \dots cntr - 1] = u[|u| - k_1 \dots |u| - 1]$$

- Let k_2 be the largest number such that the k_2 characters starting at position $cntr$ coincide with the first k_2 characters in the string v:

$$u[cntr \dots cntr + k_2 - 1] = v[0 \dots k_2 - 1]$$

- Then we have a repetition exactly for any pair $(l_1,\ l_2)$ with

$$l_1 \le k_1,$$

$$l_2 \le k_2.$$

To summarize:

- We fixate a specific position $cntr$.
- All repetition which we will find now have length $2l = 2(|u| - cntr)$. There might be multiple such repetitions, they depend on the lengths l_1 and $l_2 = l - l_1$.
- We find k_1 and k_2 as described above.
- Then all suitable repetitions are the ones for which the lengths of the pieces l_1 and l_2 satisfy the conditions:

$$l_1 + l_2 = l = |u| - cntr$$

$$l_1 \le k_1,$$

$$l_2 \le k_2.$$

Therefore the only remaining part is how we can compute the values k_1 and k_2 quickly for every position $cntr$. Luckily we can compute them in $O(1)$ using the Z-function:

- To can find the value k_1 for each position by calculating the Z-function for the string $\overline{u}$ (i.e. the reversed string u). Then the value k_1 for a particular $cntr$ will be equal to the corresponding value of the array of the Z-function.
- To precompute all values k_2, we calculate the Z-function for the string $v + \# + u$ (i.e. the string u concatenated with the separator character $\#$ and the string v). Again we just need to look up the corresponding value in the Z-function to get the k_2 value.

So this is enough to find all left crossing repetitions.

Right crossing repetitions

For computing the right crossing repetitions we act similarly: we define the center $cntr$ as the character corresponding to the last character in the string u.

Then the length k_1 will be defined as the largest number of characters before the position $cntr$ (inclusive) that coincide with the last characters of the string u. And the length k_2 will be defined as the largest number of characters starting at $cntr + 1$ that coincide with the characters of the string v .

Thus we can find the values k_1 and k_2 by computing the Z-function for the strings $\overline{u} + \# + \overline{v}$ and v.

After that we can find the repetitions by looking at all positions $cntr$, and use the same criterion as we had for left crossing repetitions.

Implementation

The implementation of the Main-Lorentz algorithm finds all repetitions in form of peculiar tuples of size four: $(cntr,\ l,\ k_1,\ k_2)$ in $O(n \log n)$ time. If you only want to find the number of repetitions in a string, or only want to find the longest repetition in a string, this information is enough and the runtime will still be $O(n \log n)$.

Notice that if you want to expand these tuples to get the starting and end position of each repetition, then the runtime will be the runtime will be $O(n^2)$ (remember that there can be $O(n^2)$ repetitions). In this implementation we will do so, and store all found repetition in a vector of pairs of start and end indices.

```
vector<int> z_function(string const& s) {
    int n = s.size();
    vector<int> z(n);
    for (int i = 1, l = 0, r = 0; i < n; i++) {
        if (i <= r)
            z[i] = min(r-i+1, z[i-l]);
        while (i + z[i] < n && s[z[i]] == s[i+z[i]])
            z[i]++;
        if (i + z[i] - 1 > r) {
            l = i;
            r = i + z[i] - 1;
        }
    }
    return z;
}

int get_z(vector<int> const& z, int i) {
    if (0 <= i && i < (int)z.size())
        return z[i];
    else
        return 0;
}

vector<pair<int, int>> repetitions;

void convert_to_repetitions(int shift, bool left, int cntr, int l, int k1, int k2) {
    for (int l1 = max(1, l - k2); l1 <= min(l, k1); l1++) {
        if (left && l1 == l) break;
        int l2 = l - l1;
        int pos = shift + (left ? cntr - l1 : cntr - l - l1 + 1);
        repetitions.emplace_back(pos, pos + 2*l - 1);
    }
}

void find_repetitions(string s, int shift = 0) {
    int n = s.size();
    if (n == 1)
        return;

    int nu = n / 2;
    int nv = n - nu;
    string u = s.substr(0, nu);
    string v = s.substr(nu);
    string ru(u.rbegin(), u.rend());
    string rv(v.rbegin(), v.rend());

    find_repetitions(u, shift);
    find_repetitions(v, shift + nu);

    vector<int> z1 = z_function(ru);
    vector<int> z2 = z_function(v + '#' + u);
    vector<int> z3 = z_function(ru + '#' + rv);
    vector<int> z4 = z_function(v);

    for (int cntr = 0; cntr < n; cntr++) {
        int l, k1, k2;
        if (cntr < nu) {
            l = nu - cntr;
            k1 = get_z(z1, nu - cntr);
            k2 = get_z(z2, nv + 1 + cntr);
        } else {
            l = cntr - nu + 1;
            k1 = get_z(z3, nu + 1 + nv - 1 - (cntr - nu));
            k2 = get_z(z4, (cntr - nu) + 1);
        }
        if (k1 + k2 >= l)
            convert_to_repetitions(shift, cntr < nu, cntr, l, k1, k2);
    }
}
```

Contributors:

jakobkogler (91.29%)   adamant-pwn (7.58%)   mhayter (0.38%)   deji725 (0.38%)   wikku (0.38%)