Module 12 of 12

String algorithms: KMP, Z and hashing

Linear-time matching with the KMP prefix function and the Z-function, borders and periods, and O(1) substring comparison with a double rolling hash.

Trying every start of a pattern in a text costs O(n · m), which is billions of steps at contest sizes. The algorithms in this module never re-read a character they already understand.

The **prefix function** records, for every prefix of a string, its longest border — a prefix that is also a suffix — and the **KMP** search uses it to fall back without moving backwards in the text. The **Z-function** records how far each position matches the string's own prefix, reusing a box of known matches; on `pattern + '#' + text` it finds every occurrence too. A **rolling hash** turns any substring into a number in O(1), so equality, distinctness and "longest repeated" questions become sorting and comparing — with two moduli so that collisions do not decide the answer.

The exercises are every occurrence of a pattern with KMP and the number of distinct substrings of a given length with a double hash. The timed checkpoint finds a string's longest border, smallest period and repeat count.

CONTENTS

1	KMP, the Z-function and rolling hashes	16 min
✓	Checkpoint — String algorithms	60 min

Exercises in this module compile on Clang 18 · C++20. Reference solutions and the answer key follow each lesson.

LESSON 1 · 16 MIN

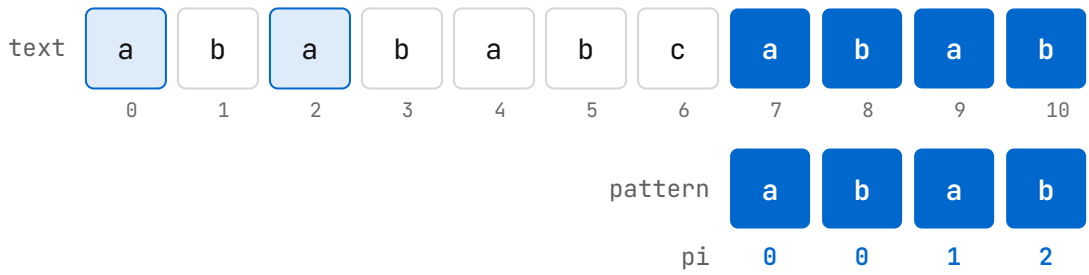
KMP, the Z-function and rolling hashes

Searching a pattern of length m in a text of length n by trying every start is O(n · m) — 2.5×10^9 when both are 5×10^4. Three tools do it in O(n + m) by never re-reading what is already known.

The prefix function and KMP

- `pi[i]` = the length of the longest proper prefix of `s[0..i]` that is also its suffix (its **border**).
- Build it in O(m): try to extend the previous border; on a mismatch fall back to `pi[j - 1]` and try again.
- Match the same way: `j` characters of the pattern are matched; on a mismatch `j = pi[j - 1]`; at `j = m` record a match and fall back to `pi[m - 1]`.
- The text index only moves forward, so the whole search is O(n + m).

KMP: build the prefix function, then never move back in the text



1. pi[i] is the length of the longest proper prefix of the pattern that is also a suffix of its first i + 1 characters. pi[0] is always 0; pi[1] = 0, since "ab" has no border.
2. "aba": "ab" had no border, so the new a is compared with the first character. They match: pi[2] = 1.
3. "abab": the previous border "a" extends by b, so pi[3] = 2 — the pattern's longest border is "ab". The table is built in O(m).
4. Scan the text with j = characters matched. At index 3 all 4 match: an occurrence at index 0.
5. After the last match j fell back to pi[3] = 2: the pattern slid by 2 and "ab" was already matched, so only 2 new characters were compared. Another match at index 2.
6. Index 6: text c ≠ pattern a with j = 2. Instead of restarting, j falls back to pi[1] = 0; the text pointer stays at 6.
7. From j = 0 the pattern is matched afresh, one text character at a time: at index 10 all 4 match, an occurrence at index 7. In all: 3 occurrences, at 0, 2, 7, in one pass of the text.

```cpp
vector<int> prefixFunction(const string& s) {
    vector<int> pi(s.size(), 0);
    for (size_t i = 1; i < s.size(); ++i) {
        int j = pi[i - 1];
        while (j > 0 && s[i] != s[j]) j = pi[j - 1];
        if (s[i] == s[j]) ++j;
        pi[i] = j;
    }
    return pi;
}
```

The Z-function

- `z[i]` = the length of the longest common prefix of `s` and `s[i..]`.
- Keep the box `[l, r)`: the rightmost segment known to equal a prefix. Inside it, `z[i]` starts at `min(r - i, z[i - l])` for free; only characters past `r` are compared.
- To find a pattern, run Z on `pattern + '#' + text`: every `z[i] = m` is an occurrence.

The Z-function: positions inside the box start from z[i − l]



1. z[i] is how many characters starting at i match the start of s. Comparing from scratch at every i would be O(n²); the box [l, r) of the rightmost match found so far lets most positions start with an answer.
2. i = 1 is not inside the box: compare from scratch; comparing on gives z[1] = 1. The match reaches past the old box: the box becomes [1, 2).
3. i = 2 is not inside the box: compare from scratch; s[2] = b differs from s[0], so z[2] = 0.
4. i = 3 is not inside the box: compare from scratch; s[3] = x differs from s[0], so z[3] = 0.
5. i = 4 is not inside the box: compare from scratch; comparing on gives z[4] = 4. The match reaches past the old box: the box becomes [4, 8).
6. i = 5 is inside the box, which copies s from index 0, so z[5] starts at min(r − i, z[1]) = 1 for free; the next character differs, so z[5] = 1.
7. i = 6 is inside the box, which copies s from index 0, so z[6] starts at min(r − i, z[2]) = 0 for free; the next character differs, so z[6] = 0.
8. i = 7 is inside the box, which copies s from index 0, so z[7] starts at min(r − i, z[3]) = 0 for free; the next character differs, so z[7] = 0.
9. i = 8 is not inside the box: compare from scratch; s[8] = c differs from s[0], so z[8] = 0.
10. i = 9 is not inside the box: compare from scratch; comparing on gives z[9] = 3. The match reaches past the old box: the box becomes [9, 12).
11. i = 10 is inside the box, which copies s from index 0, so z[10] starts at min(r − i, z[1]) = 1 for free; the next character differs, so z[10] = 1.
12. i = 11 is inside the box, which copies s from index 0, so z[11] starts at min(r − i, z[2]) = 0 for free; the next character differs, so z[11] = 0.

Rolling hashes

- Prefix hashes `h[i+1] = h[i] · B + s[i] (mod p)` and powers `B^k`.
- `hash(l, r) = h[r] - h[l] · B^(r-l)`: any substring in O(1), so "are these equal?" is O(1).
- One 32-bit modulus collides often enough to fail anti-hash tests; use **two** moduli with fixed bases (or compare the strings when the hashes agree).
- Hashing answers what KMP cannot directly: distinct substrings, the longest repeated substring with binary search, palindromes by comparing with the reversed string.

More practice

- Find the Index of the First Occurrence in a String
- String Matching: All Occurrences
- Longest Happy Prefix
- Shortest Palindrome
- Repeated String Match

EXERCISE 1

Every occurrence of a pattern

Read a text `t` and a pattern `p` (lowercase letters, one per line). Print the number of positions where `p` occurs in `t` — overlapping occurrences count — and, if there is at least one, a second line with the 1-based start positions in increasing order.

Use **KMP**: build the prefix function of `p`, then scan `t` once. O(|t| + |p|).

Constraints: `1 ≤ |t|, |p| ≤ 10^6` (the pattern may be longer than the text). **Output:** the count, then the positions (no second line when the count is 0).

```
abababcabab
abab
```

prints

```
3
1 3 8
```

INPUT 1

```
abababcabab
abab
```

OUTPUT 1

```
3
1 3 8
```

INPUT 2

```
hello
world
```

OUTPUT 2

```
0
```

REFERENCE SOLUTION

```cpp
#include <bits/stdc++.h>
using namespace std;

vector<int> prefixFunction(const string& s) {
    vector<int> pi(s.size(), 0);
    for (size_t i = 1; i < s.size(); ++i) {
        int j = pi[i - 1];
        while (j > 0 && s[i] != s[j]) j = pi[j - 1];
        if (s[i] == s[j]) ++j;
        pi[i] = j;
    }
    return pi;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    string t, p;
    cin >> t >> p;
    const int n = static_cast<int>(t.size()), m = static_cast<int>(p.size());
    vector<int> pi = prefixFunction(p), found;
    int j = 0;
    for (int i = 0; i < n; ++i) {
        while (j > 0 && t[i] != p[j]) j = pi[j - 1];
        if (t[i] == p[j]) ++j;
        if (j == m) {
            found.push_back(i - m + 2);   // 1-based start
            j = pi[m - 1];                // keep the border: occurrences may overlc
        }
    }
    cout << found.size() << '\n';
    for (size_t k = 0; k < found.size(); ++k) cout << found[k] << (k + 1 == found.si
    return 0;
}
```

EXERCISE 2

Distinct substrings of each length

Read a string `s` of lowercase letters, then `q` lengths. For each length `L`, print how many **distinct** substrings of length `L` the string has.

Hash every window of length `L` in O(1) with a **double rolling hash** (two moduli, fixed bases), sort the hashes and count the distinct ones: O(n log n) per query.

Constraints: `1 ≤ |s| ≤ 2 × 10^5`, `1 ≤ q ≤ 10`, `1 ≤ L ≤ |s|`. **Input:** `s`, then `q`, then the `q` lengths. **Output:** one count per line.

```
abab
4
1 2 3 4
```

prints `2`, `2`, `2`, `1` — {a, b}, {ab, ba}, {aba, bab}, {abab}.

INPUT 1

```
abab
4
1 2 3 4
```

OUTPUT 1

```
2
2
2
1
```

INPUT 2

```
z
1
1
```

OUTPUT 2

```
1
```

REFERENCE SOLUTION

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

const ll M1 = 1'000'000'007, M2 = 1'000'000'009, B1 = 131, B2 = 137;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    string s;
    int q;
    cin >> s >> q;
    const int n = static_cast<int>(s.size());
    vector<ll> h1(n + 1, 0), h2(n + 1, 0), p1(n + 1, 1), p2(n + 1, 1);
    for (int i = 0; i < n; ++i) {
        ll c = s[i] - 'a' + 1;
        h1[i + 1] = (h1[i] * B1 + c) % M1;
        h2[i + 1] = (h2[i] * B2 + c) % M2;
        p1[i + 1] = p1[i] * B1 % M1;
        p2[i + 1] = p2[i] * B2 % M2;
    }
    auto get = [&](const vector<ll>& h, const vector<ll>& p, ll mod, int l, int r) {
        return ((h[r] - h[l] * p[r - l]) % mod + mod) % mod;
    };
    while (q--) {
        int L;
        cin >> L;
        vector<unsigned long long> keys;
        keys.reserve(n - L + 1);
        for (int i = 0; i + L <= n; ++i) {
            unsigned long long a = static_cast<unsigned long long>(get(h1, p1, M1, i
            unsigned long long b = static_cast<unsigned long long>(get(h2, p2, M2, i
            keys.push_back(a << 32 | b);   // both halves are below 2^30
        }
        sort(keys.begin(), keys.end());
        cout << unique(keys.begin(), keys.end()) - keys.begin() << '\n';
    }
    return 0;
}
```

QUIZ

1. What is the prefix function of `aabaaab`?

A. `0 1 0 1 2 2 3`

B. `0 1 0 1 2 3 3`

C. `0 0 1 0 1 2 3`

D. `1 2 0 1 2 2 3`

2. In KMP, why does the text index never move backwards?

A. On a mismatch the pattern is shifted by falling back to a border, and the border's characters are already known to match the text

B. Because the pattern is reversed first

C. Because every mismatch restarts the pattern at index 0

D. It does move back, by pi[j] characters

3. The Z-function is at `i` inside the box `[l, r)` with `z[i - l] = 5` and `r - i = 3`. Where does `z[i]` start?

A. At 3, then extends by comparing characters from position r

B. At 5, final

C. At 0

D. At 8

4. Why do contest solutions hash with **two** moduli (or a 64-bit one)?

A. With one ~10^9 modulus and ~10^5 substrings, a collision among them is likely by the birthday bound, and fixed bases can be attacked

B. Because one modulus overflows `long long`

C. Two moduli make the hash computation faster

D. To handle uppercase letters

ANSWER KEY · KMP, THE Z-FUNCTION AND ROLLING HASHES

1. A a → 0; aa → 1 ("a"); aab → 0; aaba → 1; aabaa → 2 ("aa"); aabaaa → 2 ("aa": "aab" ≠ "aaa"); aabaaab → 3 ("aab").

2. A The last j text characters equal p[0..j); its longest border p[0..pi[j−1]) therefore also matches the text's last characters, so matching resumes from there without re-reading anything. The pointer into the text only increases — n steps, plus amortised fallbacks.

3. A Inside the box s[i..r) equals s[i−l..r−l), so z[i] ≥ min(z[i − l], r − i) = 3. Beyond r nothing is known, so the comparison resumes there — that is how each character past r is compared once.

4. A 10^5 values give about 5 × 10^9 pairs against 10^9 hash values — a collision is expected. Two independent moduli push the chance per pair to about 10^-18.

CHECKPOINT · 60 MIN

Checkpoint — String algorithms

A timed checkpoint on the module: building and using the prefix function, KMP matching, the Z-function's box, rolling hashes and their collisions, and borders and periods.

How it works. Sixty minutes from the moment you start the timer: ten questions and one program. 70% on the questions and the program accepted clears the module.

Before you start, make sure you can answer these from memory:

- The prefix function of `aabaaab`, and what `pi[n - 1]` means.
- Where `j` falls back to after a full match in KMP, and why the text index never moves back.
- What the Z-function's box `[l, r)` lets you skip.
- How to read the hash of `s[l..r)` from prefix hashes, and why one modulus is not enough.

The program prints a string's longest border, its smallest period and how many times that period repeats.

EXERCISE

Border, period and repeats

Read a string `s` of lowercase letters (length `n`). Print one line:

`border=<b> period=<p> repeats=<r>`

- `b` — the length of the longest **proper** prefix of `s` that is also its suffix;
- `p = n - b` — the smallest period (`s[i] = s[i + p]` for every valid `i`);
- `r = n / p` when `p` divides `n` (then `s` is `r` copies of its first `p` letters), otherwise `1`.

Use the **prefix function**: `b = pi[n - 1]`. O(n).

Constraints: `1 ≤ n ≤ 10^6`.

```
abcabcabc
```

prints `border=6 period=3 repeats=3`; `aabaa` prints `border=2 period=3 repeats=1`.

INPUT 1

```
abcabcabc
```

OUTPUT 1

```
border=6 period=3 repeats=3
```

INPUT 2

```
aabaa
```

OUTPUT 2

```
border=2 period=3 repeats=1
```

REFERENCE SOLUTION

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    string s;
    cin >> s;
    const int n = static_cast<int>(s.size());
    vector<int> pi(n, 0);
    for (int i = 1; i < n; ++i) {
        int j = pi[i - 1];
        while (j > 0 && s[i] != s[j]) j = pi[j - 1];
        if (s[i] == s[j]) ++j;
        pi[i] = j;
    }
    int b = pi[n - 1];
    int p = n - b;
    int repeats = n % p == 0 ? n / p : 1;
    cout << "border=" << b << " period=" << p << " repeats=" << repeats << '\n';
    return 0;
}
```

CHECKPOINT QUESTIONS

1. What is `pi` for `abab`?

A. `0 0 1 2`

B. `0 1 2 3`

C. `0 0 0 2`

D. `1 1 2 2`

2. KMP has just found a full match (j = m). What happens to j?

A. `j = pi[m - 1]`

B. `j = 0`

C. `j = m - 1`

D. The search stops

3. Overall time of KMP matching for text length n and pattern length m?

A. O(n + m)

B. O(n · m)

C. O(n log m)

D. O(m²) + O(n)

4. To find pattern `p` in text `t` with the Z-function, you compute Z of…

A. `p + '#' + t`, and report each i with `z[i] == |p|`

B. `t + p`

C. `p` alone, then compare with `t`

D. `t` reversed

5. What is `z[0]` usually defined as?

A. 0 (or n by convention) — the whole string trivially matches itself, so it carries no information

B. 1

C. Always n − 1

D. Undefined behaviour

6. Prefix hashes `h[i+1] = h[i]·B + s[i] (mod p)`. The hash of `s[l..r)` is…

A. `h[r] - h[l] · B^(r-l)` mod p

B. `h[r] - h[l]` mod p

C. `h[r] / h[l]`

D. `h[r-l]`

7. Why map letters to 1..26 rather than 0..25 for a polynomial hash?

A. With a = 0 and b = 1, "b" and "ab" both hash to 1 — leading zeros vanish

B. Because 0 is not a valid `char`

C. It makes the modulus prime

D. It does not matter

8. The smallest period of `abcabcab`?

A. 3

B. 8

C. 5

D. 2

9. On a hash hit for two substrings of equal length, how do you make the answer certain?

A. Compare the substrings character by character

B. Hash them again with the same base

C. Trust the hash — collisions cannot happen

D. Compare their lengths

10. Which question is answered most directly by the Z-function rather than the prefix function?

A. For every i, how long the string's prefix that starts again at i is

B. The longest border of the whole string

C. The number of distinct substrings

D. The lexicographically smallest rotation

ANSWER KEY · CHECKPOINT — STRING ALGORITHMS

1. A a → 0, ab → 0, aba → 1 ("a"), abab → 2 ("ab"). A border must be proper, so pi[0] is always 0.

2. A Falling back to the pattern's longest border keeps the characters that can start the next, possibly overlapping, occurrence. Resetting to 0 would miss "aa" at position 2 of "aaa".

3. A Building pi is O(m) and the scan is O(n): j rises by at most 1 per character and each fallback lowers it, so the total number of fallbacks is bounded by the number of rises.

4. A The separator, a character in neither string, caps every z value at |p|, so z[i] = |p| means p starts at position i − |p| − 1 of t.

5. A Most implementations leave z[0] = 0 and loop from i = 1; some set it to n. Either way it is never used for matching.

6. A h[r] holds h[l] shifted left by r − l digits plus the substring's own value; subtract the shifted h[l]. Plain h[r] − h[l] mixes in positional weights.

7. A A polynomial hash reads the string as a number in base B; a zero digit at the front adds nothing, so strings differing only by leading 'a's collide every time.

8. A pi[7] = 5 ("abcab"), so p = 8 − 5 = 3. 3 does not divide 8, so the string is not a whole number of copies — but s[i] = s[i + 3] still holds throughout.

9. A Equal strings always hash equal; unequal strings rarely do. A direct comparison on the hits removes the doubt, and it is cheap when hits are few.

10. A That is the definition of z[i]. The longest border is pi[n − 1]; distinct substrings need hashing or suffix structures.

CodeKairo · Competitive Programming study plan · Module 12: String algorithms: KMP, Z and hashing. Solve the exercises online to track your progress and earn XP.