

Z-Function

Prefix matching in $O(n)$ — the Z-array ■

● MEDIUM

■ Hook

The Z-function is a powerful string preprocessing tool. $z[i]$ = length of the longest substring starting at position i that is also a **prefix** of the string. Used for pattern matching, finding all occurrences of a pattern in a text in $O(n+m)$.

■ Core Intuition — Z-Window

■ The Z-box $[l, r]$

Maintain a window $[l, r]$ — the rightmost Z-box (substring that matches a prefix). For position i : if $i < r$, use previously computed $z[i-l]$ as a head start. Then extend naively. This amortization gives $O(n)$ total.

■ Z-Array — 'aabxaaabxaa'

```
s = a a b x a a a b x a a
i = 0 1 2 3 4 5 6 7 8 9 10
z = 0 1 0 0 6 1 0 3 0 2 1
```

↑

↑

$z[1]=1$: $s[1..1]='a'$ matches prefix 'a' of length 1

$z[4]=6$: $s[4..9]='aaabxa'?$ no... 'aaabxa' vs 'aabxaa'?

Actually $z[4]=6$ means $s[4..9]$ matches $s[0..5]='aabxaa'$

PATTERN MATCHING: concat pattern+'\$'+text, compute Z.

Where $z[i] == \text{len}(\text{pattern}) \rightarrow$ pattern found at that position.

■ Code

```

int[] zFunction(String s) {
    int n = s.length();
    int[] z = new int[n];
    int l=0, r=0;
    for (int i=1; i<n; i++) {
        if (i < r) z[i] = Math.min(r-i, z[i-l]);
        while (i+z[i]<n && s.charAt(z[i])==s.charAt(i+z[i]))
            z[i]++;
        if (i+z[i] > r) { l=i; r=i+z[i]; }
    }
    return z;
}

// Pattern search: find all occurrences of pat in text
List<Integer> search(String pat, String text) {
    String combined = pat + '$' + text;
    int[] z = zFunction(combined);
    List<Integer> res = new ArrayList<>();
    for (int i=pat.length()+1; i<combined.length(); i++)
        if (z[i] == pat.length()) res.add(i - pat.length() - 1);
    return res;
}

```

■ TIME

O(n)

■ SPACE

O(n)

Practice

#	Problem	Diff	Pattern
1	Z-Function (build and use)	Medium	Z-Algorithm
2	Find All Occurrences of Pattern	Medium	Z or KMP
3	Minimum Period of String	Medium	Z-Function
4	Count Occurrences of Anagrams (LC 438)	Medium	Sliding Window
5	Repeated Substring Pattern (LC 459)	Easy	Z / KMP

- ✓ $z[i]$ = length of longest substring starting at i matching prefix
- ✓ Maintain Z-box $[l, r]$: rightmost matching window
- ✓ Use $z[i-l]$ as head start when $i < r$ (amortized $O(n)$)
- ✓ Pattern search: concat $pat + '$' + text$, find positions where $z[i] == \text{len}(pat)$
- ✓ Time: $O(n)$ | Space: $O(n)$

■ MENTAL MODEL — Remember This Forever

The Z-box $[l, r]$ is your cache. It says: 'I already know that $s[l..r]$ equals $s[0..r-l]$.' When processing position i inside the box, you get a free head start using $z[i-l]$. You only do new work (naive extension) at the right boundary — which can only move rightward, giving $O(n)$ total.