Z 函数（扩展 KMP）

约定：字符串下标以 0 为起点。

定义

对于一个长度为 n 的字符串 s，定义函数 $z[i]$ 表示 s 和 $s[i, n-1]$（即以 $s[i]$ 开头的后缀）的最长公共前缀（LCP）的长度，则 z 被称为 s 的 **Z 函数**。特别地，$z[0] = 0$。

国外一般将计算该数组的算法称为 **Z Algorithm**，而国内则称其为 **扩展 KMP**（exKMP）。

这篇文章介绍在 $O(n)$ 时间复杂度内计算 Z 函数的算法以及其各种应用。

解释

下面若干样例展示了对于不同字符串的 Z 函数：

- $z(\mathtt{aaaaa}) = [0, 4, 3, 2, 1]$
- $z(\mathtt{aaabaab}) = [0, 2, 1, 0, 2, 1, 0]$
- $z(\mathtt{abacaba}) = [0, 0, 1, 0, 3, 0, 1]$

朴素算法

Z 函数的朴素算法复杂度为 $O(n^2)$：

实现

C++

```cpp
vector<int> z_function_trivial(string s) {
  int n = (int)s.length();
  vector<int> z(n);
  for (int i = 1; i < n; ++i)
    while (i + z[i] < n && s[z[i]] == s[i + z[i]]) ++z[i];
  return z;
}
```

Python

```python
def z_function_trivial(s):
    n = len(s)
    z = [0] * n
    for i in range(1, n):
        while i + z[i] < n and s[z[i]] == s[i + z[i]]:
            z[i] += 1
    return z
```

线性算法

如同大多数字符串主题所介绍的算法，其关键在于，运用自动机的思想寻找限制条件下的状态转移函数，使得可以借助之前的状态来加速计算新的状态。

在该算法中，我们从 1 到 $n-1$ 顺次计算 $z[i]$ 的值（$z[0]=0$）。在计算 $z[i]$ 的过程中，我们会利用已经计算好的 $z[0],\dots,z[i-1]$。

对于 i，我们称区间 $[i,i+z[i]-1]$ 是 i 的 **匹配段**，也可以叫 Z-box。

算法的过程中我们维护右端点最靠右的匹配段。为了方便，记作 $[l,r]$。根据定义，$s[l,r]$ 是 s 的前缀。在计算 $z[i]$ 时我们保证 $l\le i$。初始时 $l=r=0$。

在计算 $z[i]$ 的过程中：

- 如果 $i\le r$，那么根据 $[l,r]$ 的定义有 $s[i,r]=s[i-l,r-l]$，因此 $z[i]\ge \min(z[i-l],r-i+1)$。这时：
  - 若 $z[i-l]<r-i+1$，则 $z[i]=z[i-l]$。
  - 否则 $z[i-l]\ge r-i+1$，这时我们令 $z[i]=r-i+1$，然后暴力枚举下一个字符扩展 $z[i]$ 直到不能扩展为止。
- 如果 $i>r$，那么我们直接按照朴素算法，从 $s[i]$ 开始比较，暴力求出 $z[i]$。
- 在求出 $z[i]$ 后，如果 $i+z[i]-1>r$，我们就需要更新 $[l,r]$，即令 $l=i,r=i+z[i]-1$。

可以访问 这个网站 来看 Z 函数的模拟过程。

实现

C++

```cpp
vector<int> z_function(string s) {
  int n = (int)s.length();
  vector<int> z(n);
  for (int i = 1, l = 0, r = 0; i < n; ++i) {
    if (i <= r && z[i - l] < r - i + 1) {
      z[i] = z[i - l];
    } else {
      z[i] = max(0, r - i + 1);
      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;
}
```

Python

```python
def z_function(s):
    n = len(s)
    z = [0] * n
    l, r = 0, 0
    for i in range(1, n):
        if i <= r and z[i - l] < r - i + 1:
            z[i] = z[i - l]
        else:
            z[i] = max(0, r - i + 1)
            while i + z[i] < n and s[z[i]] == s[i + z[i]]:
                z[i] += 1
        if i + z[i] - 1 > r:
            l = i
            r = i + z[i] - 1
    return z
```

复杂度分析

对于内层 `while` 循环，每次执行都会使得 r 向后移至少 1 位，而 $r < n - 1$，所以总共只会执行 n 次。

对于外层循环，只有一遍线性遍历。

总复杂度为 $O(n)$。

应用

我们现在来考虑在若干具体情况下 Z 函数的应用。

这些应用在很大程度上同 前缀函数 的应用类似。

匹配所有子串

为了避免混淆，我们将 t 称作 **文本**，将 p 称作 **模式**。所给出的问题是：寻找在文本 t 中模式 p 的所有出现（occurrence）。

为了解决该问题，我们构造一个新的字符串 $s = p + \diamond + t$，也即我们将 p 和 t 连接在一起，但是在中间放置了一个分割字符 $\diamond$（我们将如此选取 $\diamond$ 使得其必定不出现在 p 和 t 中）。

首先计算 s 的 Z 函数。接下来，对于在区间 $[0, |t| - 1]$ 中的任意 i，我们考虑以 $t[i]$ 为开头的后缀在 s 中的 Z 函数值 $k = z[i + |p| + 1]$。如果 $k = |p|$，那么我们知道有一个 p 的出现位于 t 的第 i 个位置，否则没有 p 的出现位于 t 的第 i 个位置。

其时间复杂度（同时也是其空间复杂度）为 $O(|t| + |p|)$。

本质不同子串数

给定一个长度为 n 的字符串 s，计算 s 的本质不同子串的数目。

考虑计算增量，即在知道当前 s 的本质不同子串数的情况下，计算出在 s 末尾添加一个字符后的本质不同子串数。

令 k 为当前 s 的本质不同子串数。我们添加一个新的字符 c 至 s 的末尾。显然，会出现一些以 c 结尾的新的子串（以 c 结尾且之前未出现过的子串）。

设串 t 是 $s + c$ 的反串（反串指将原字符串的字符倒序排列形成的字符串）。我们的任务是计算有多少 t 的前缀未在 t 的其他地方出现。考虑计算 t 的 Z 函数并找到其最大值 $z_{\max}$。则 t 的长度小于等于 $z_{\max}$ 的前缀的反串在 s 中是已经出现过的以 c 结尾的子串。

所以，将字符 c 添加至 s 后新出现的子串数目为 $|t| - z_{\max}$。

算法时间复杂度为 $O(n^2)$。

值得注意的是，我们可以用同样的方法在 $O(n)$ 时间内，重新计算在端点处添加一个字符或者删除一个字符（从尾或者头）后的本质不同子串数目。

字符串整周期

给定一个长度为 n 的字符串 s，找到其最短的整周期，即寻找一个最短的字符串 t，使得 s 可以被若干个 t 拼接而成的字符串表示。

考虑计算 s 的 Z 函数，则其整周期的长度为最小的 n 的因数 i，满足 $i + z[i] = n$。

该事实的证明同应用 前缀函数 的证明一样。

练习题目

- luogu P5410【模板】扩展 KMP/exKMP（Z 函数）
- luogu P7114【NOIP2020】字符串匹配
- CF126B Password
- UVa # 455 Periodic Strings
- UVa # 11022 String Factoring
- UVa 11475 - Extend to Palindrome
- Codechef - Chef and Strings
- Codeforces - Prefixes and Suffixes
- Leetcode 2223 - Sum of Scores of Built Strings

本页面最近更新：2025/8/30 13:58:15，更新历史
发现错误？想一起完善？ 在 GitHub 上编辑此页！
本页面贡献者：sshwy, StudyingFather, Enter-tainer, LeoJacob, countercurrent-time, H-J-Granger, minghu6, NachtgeistW, iamtwz, Ir1d, weiyong1024, AngelKitty, CCXXXI, cjsoft, diauweb, Early0v0, ezoixx130, GekkaSaori, Konano, LovelyBuggies, Makkiy, mgt, P-Y-Y, PotassiumWings, SamZhangQingChuan, Suyun514, Xeonacid, amlhdsan, c-forrest, Dfkuaid, ethanrao, GavinZhengOI, Gesrua, gi-b716, HeRaNO, ksyx, kxccc, lychees, Marcythm, Menci, ouuan, Peanut-Tang, pengxurui, shawlleyw, shuzhouliu, SukkaW, Tiphereth-A, TrisolarisHD