Tutorial on Z Algorithm

AUTH
Vishal Shar

Z algorithm is a linear time string matching algorithm which runs in $O(n)$ complexity. It is used to find all occurrence of a pattern P in a string S, which is common string searching problem.

Algorithm

Given a string S of length n, the Z Algorithm produces an array Z where $Z[i]$ is the length of the longest substring starting from $S[i]$ which is also a prefix of S, i.e. the maximum k such that $S[j] = S[i + j]$ for all $0 \le j < k$. Note that $Z[i] = 0$ means that $S[0] \neq S[i]$. For easier terminology, let's refer to substrings which are also a prefix as prefix-substrings.

The algorithm relies on a single, crucial invariant. Iterate over the letters in the string (index i from 1 to $n - 1$) and maintain an interval $[L,\ R]$ which is the interval with maximum R such that $1 \le L \le i \le R$ and $S[L \ldots R]$ is a prefix-substring (if no such interval exists, just let $L = R = \ -1$). For $i = 1$, simply compute L

and R by comparing $S[0...]$ to $S[1...]$. $Z[1]$ is obtained during this process.

Now, suppose the correct interval $[L, \ R]$ for $i - 1$ and all of the Z values up to $i - 1$. Compute $Z[i]$ and the new $[L, \ R]$ by the following steps:

- If $i > R$, then there does not exist a prefix-substring of S that starts before i and ends at or after i. If such a substring existed, $[L, \ R]$ would have been the interval for that substring rather than its current value. Thus "reset" and compute a new $[L, \ R]$ by comparing $S[0...]$ to $S[i. . .]$ and get $Z[i]$ at the same time ($Z[i] = R - L + 1$).

- Otherwise, $i \leq R$, so the current $[L, \ R]$ extends at least to i. Let $k = i - L$. It is known that $Z[i] \geq min(Z[k], \ R - i + 1)$ because $S[i. . .]$ matches $S[k. . .]$ for at least $R - i + 1$ characters (they are in the $[L, \ R]$ interval which is known to be a prefix-substring).

- If $Z[k] < R - i + 1$, then there is no longer prefix-substring starting at $S[i]$ (or else $Z[k]$ would be larger), meaning $Z[i] = Z[k]$ and $[L, \ R]$ stays the same. The latter is true because $[L, \ R]$ only changes if there is a prefix-substring starting at $S[i]$ that extends beyond R, which is not the case here.

- If $Z[k] \geq R - i + 1$, then it is possible for $S[i...]$ to match $S[0...]$ for more than $R - i + 1$ characters (i.e. past position R). Thus, there's a need to update $[L,\ R]$ by setting $L = i$ and matching from $S[R + 1]$ forward to obtain the new R. Again, $Z[i]$ is obtained during this process.

The process computes all of the Z values in a single pass over the string, so we're done. Correctness is inherent in the algorithm and is pretty intuitively clear.

Time Complexity

The algorithm runs in $O(n)$ time. Characters are never compared at positions less than R, and every time a match is found, R is increased by one, so there are at most n comparisons.

Implementation

Note that the optimization $L = R = i$ is used when $S[0] \neq S[i]$ (it doesn't affect the algorithm since at the next iteration $i > R$ regardless).

```
int L = 0, R = 0;
for (int i = 1; i < n; i++)
{
    if (i > R)
    {
```

```
            L = R = i;
            while (R < n && s[R-L] == s[R])
            {
                R++;
            }
            z[i] = R-L;
            R--;
        }
        else
        {
            int k = i-L;
            if (z[k] < R-i+1)
            {
                z[i] = z[k];
            }
            else
            {
                L = i;
                while (R < n && s[R-L] == s[R])
                {
                    R++;
                }
                z[i] = R-L;
                R--;
            }
        }
}
```

Applications

One application of the Z Algorithm is for the standard string matching problem of finding matches for a pattern T of length m in a string S of length n. This can be done in $O(n + m)$ time by using the Z

Algorithm on the string $T\ \Phi\ S$ (that is, concatenating T, Φ, and S) where Φ is a character that matches nothing. The indices i with $Z[i] = m$ correspond to matches of T in S.

TEST YOUR UNDERSTANDING

Count Substring Occurrences

Given 2 Strings C, S. Find the number of occurrences of C in S. Length of is N, length of C is M.

Input:

Given 2 Strings C, S.

Output:

Print the number of occurrences of C in S.

Constraints:

$1 \leq M \leq N \leq 10^5$.

SAMPLE INPUT

```
abc
abcabdabc
```

SAMPLE OUTPUT

```
2
```

Enter your code or Upload your code as file.

All changes saved

Visual Basic (mono vbnc 4.0.1)

```
Loading...
```

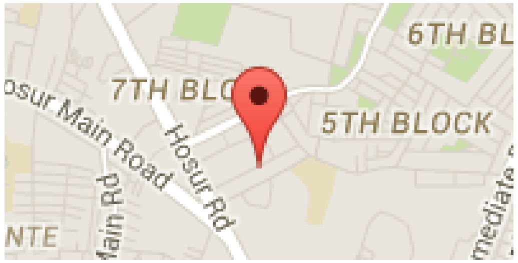
1:1

COMPILE & RUN

SUBMIT

Press ctrl-space for autocomplete suggestions.

POWERED BY codetable



IIIrd Floor,

Salarpuria Business Center,

4th B Cross Road,

5th A Block

Careers

In the Press

Academy

Developer Profile

Resume

Get Badges

Campus Ambassadors

Get Me Hired

Privacy

Terms of Service

Problem Setter Guide

Practice Problems

HackerEarth Challenges

College Challenges

College Ranking

Organise Hackathon

Hackathon Handbook

Competitive Programming

Open Source

FAQs

Customers

Koramangala Industrial Layout, Bangalore, Karnataka 560095, India.

contact@hackerear

+91-80-4155-4695

+1-650-461-4192