paladin8's blog

Z Algorithm

By paladin8, 4 years ago,

This is the first post in my Codeforces blog! I plan to use this as a place where I can write down new algorithms I learned, problems that I found interesting, or stupid mistakes I made during contests.

Today, the topic will be the Z Algorithm, which I learned as a result of failing to solve Problem B of Beta Round 93 (http://codeforces.com/contest/126/problem/B). There are some other solutions like binary search + hashing, but this one is quite nice. Anyway, first, a description of the algorithm and why it works; it is simple and makes a lot of sense (as all good algorithms are).

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 $S[0] \ne S[i]$. For easier terminology, we will refer to substrings which are also a prefix as prefix-substrings.

The algorithm relies on a single, crucial invariant. As we iterate over the letters in the string (index i from 1 to $n - 1$), we maintain an interval $[L, R]$ which is the interval with maximum R such that $1 \le L \le i \le R$ and $S[L...R]$ is a prefix-substring (if no such interval exists, just let $L = R = -1$). For $i = 1$, we can simply compute L and R by comparing $S[0...]$ to $S[1...]$. Moreover, we also get $Z[1]$ during this.

Now suppose we have the correct interval $[L, R]$ for $i - 1$ and all of the Z values up to $i - 1$. We will 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 we "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, $i \le R$, so the current $[L, R]$ extends at least to i. Let $k = i - L$. We know that $Z[i] \ge min(Z[k], R - i + 1)$ because $S[i...]$ matches $S[k...]$ for at least $R - i + 1$ characters (they are in the $[L, R]$ interval which we know to be a prefix-substring). Now we have a few more cases to consider.
- 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 we know is not the case here.
- If $Z[k] \ge R - i + 1$, then it is possible for $S[i...]$ to match $S[0...]$ for more than $R - i + 1$ characters (i.e. past position R). Thus we need to update $[L, R]$ by setting $L = i$ and matching from $S[R + 1]$ forward to obtain the new R. Again, we get $Z[i]$ during this.

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.

Analysis

We claim that the algorithm runs in $O(n)$ time, and the argument is straightforward. We never compare characters at positions less than R, and every time we match a character R increases by one, so there are at most n comparisons there. Lastly, we can only mismatch once for each i (it causes R to stop increasing), so that's another at most n comparisons, giving $O(n)$ total.

Code

Simple and short. 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--;
    }
  }
}
```

Application

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. We can do this 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.

Lastly, to solve Problem B of Beta Round 93, we simply compute Z for the given string S, then iterate from i to n - 1. If $Z[i] = n$ - i then we know the suffix from $S[i]$ is a prefix, and if the largest Z value we've seen so far is at least n - i, then we know some string inside also matches that prefix. That gives the result.

```
int maxz = 0, res = 0;
for (int i = 1; i < n; i++) {
  if (z[i] == n-i && maxz >= n-i) { res = n-i; break; }
  maxz = max(maxz, z[i]);
}
```

algorithm, beta round 93, string, tutorial, zalgorithm

+107 paladin8 4 years ago 47

Comments (47)

Write comment?

4 years ago, # | +21

The are many good blogs on Codeforces about algorithm. If they are post in same place (as tutorial) is so great.

→ Reply

thphong

4 years ago, # ^ | -11

so you dont like?

→ Reply

kvtoraman

4 years ago, # ^ | +2

He simply said that there should be a place where all useful blog posts like this are categorized

→ Reply

wil93

Olympiad in Informatics — Open Contest

lperovskaya → Yandex.Algorithm 2015: Qualification round is now!

Egor → Dress rehearsal LIVE

seland → Codeforces Round #303 (Div.2)

fushar → Asia-Pacific Informatics Olympiad (APIO) 2015 (online mirror available!)

Kerim.K → USACO problem

allleksssa → Serbian National Olympiad

SAKT → Bug of contest registeration (Just for fun)

rng_58 → World Finals Prediction Game

ahmed_aly → Teams for ACM ICPC World Finals 2015 — Marrakesh, Morocco

zxqfl → SRM 659 Editorial

Diguised → POI Subway Problem

eatmore → Yandex.Algorithm 2015 Qualification Round

MikhailRubinchik → Suffix tree that is two times shorter than Ukkonen's algorithm?

codolove → insertion in a linked list

ypizarroza → Fast Fibonacci algorithm (fast doubling method)

aropan → Codeforces Beta Round #92 - Analysis

llaki → Rockethon 2015 Editorial

MikeMirzayanov → Codeforces Round #277.5 (Div. 2) Editorial [A-D for now]

HolkinPV → Codeforces Round #171 (Div. 2)

WIL → flow algorithm?

vatsa_09 → CF Round #151 Div2 D

Detailed →