Z algorithm

Ulises Méndez Martínez

Algorist Weekly Talks

ulisesmdzmtz@gmail.com

August 7, 2015

Overview

1. Motivation
   - Exact string matching
   - Prefix, suffix problem
2. Two Pointers
   - With two list
   - In a single list
3. Z algorithm
   - Definition
   - Algorithm
   - Sample
   - Implementation

At the end of the day we will be able to solve...

Exact string matching

Given a (long) string **S** of length **n** and a shorter string pattern **P** of length **m** find all occurrences of **P** in **S**.
Occurrences of **P** are allowed to overlap.

CF Beta Round 93. Problem B. Password

Given a string **S** find the longest substring which is both prefix and suffix of **S** and also appears inside **S**.

Two Pointer Technique

The 2 pointer technique is mostly applicable in sorted arrays where we try to perform a search in $\mathbf{O(N)}$.

Two Pointer Technique

The 2 pointer technique is mostly applicable in sorted arrays where we try to perform a search in $\mathbf{O}(\mathbf{N})$.

With two list

Given two arrays ($\mathbf{A}$ and $\mathbf{B}$) sorted in ascending order and an integer $\mathbf{X}$, we need to find $\mathbf{i}$ and $\mathbf{j}$, such that $\mathbf{a}[\mathbf{i}] + \mathbf{b}[\mathbf{j}]$ is equal to $\mathbf{X}$.

Two Pointer Technique

The 2 pointer technique is mostly applicable in sorted arrays where we try to perform a search in $\mathbf{O}(\mathbf{N})$.

With two list

Given two arrays (**A** and **B**) sorted in ascending order and an integer **X**, we need to find **i** and **j**, such that $\mathbf{a}[\mathbf{i}] + \mathbf{b}[\mathbf{j}]$ is equal to **X**.

Solution

```
i = 0; j = b.size() - 1;
while( i < a.size() )
{
  while(a[i]+b[j]>X && j>0) j--;
  if(a[i]+b[j]==X) processAnswer(i,j);
  ++i;
}
```

In a single list

Given a list of **N** integers, your task is to select **K** integers from the list such that its unfairness is minimized.

If $(\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}, ..., \mathbf{x_k})$ are **K** numbers selected from the list **N**, the unfairness is defined as $\mathbf{max}(\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_k}) - \mathbf{min}(\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_k})$ where max denotes the largest integer among the elements of **K**, and min denotes the smallest integer among the elements of **K**.

In a single list

Given a list of **N** integers, your task is to select **K** integers from the list such that its unfairness is minimized.

If $(\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}, ..., \mathbf{x_k})$ are **K** numbers selected from the list **N**, the unfairness is defined as $\mathbf{max}(\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_k}) - \mathbf{min}(\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_k})$ where max denotes the largest integer among the elements of **K**, and min denotes the smallest integer among the elements of **K**.

Solution

```
mn = INT_MAX;
sort(a,a+n);
for(i=0;i<=n-k;i++)
{
  mn = min(mn , a[i+k-1]-a[i]);
}
cout<<mn<<endl;
```

Z function

Definition

Given a string $\mathbf{S}$ of length $\mathbf{n}$, the **Z Algorithm** produces an array $\mathbf{Z}$ where $\mathbf{Z}[\mathbf{i}]$ is the length of the longest substring starting from $\mathbf{S}[\mathbf{i}]$ which is also a **prefix** of $\mathbf{S}$, i.e. the maximum $\mathbf{k}$ such that $\mathbf{S}[\mathbf{j}] = \mathbf{S}[\mathbf{i}+\mathbf{j}]$ for all $\mathbf{0} \leq \mathbf{j} < \mathbf{k}$. Note that $\mathbf{Z}[\mathbf{i}] = \mathbf{0}$ means that $\mathbf{S}[\mathbf{0}] \neq \mathbf{S}[\mathbf{i}]$.

Strings with their Z values

aaaaa	**aaabaab**	**abacaba**
$z[0] = 0$	$z[0] = 0$	$z[0] = 0$
$z[1] = 4$	$z[1] = 2$	$z[1] = 0$
$z[2] = 3$	$z[2] = 1$	$z[2] = 1$
$z[3] = 2$	$z[3] = 0$	$z[3] = 0$
$z[4] = 1$	$z[4] = 2$	$z[4] = 3$
	$z[5] = 1$	$z[5] = 0$
	$z[6] = 0$	$z[6] = 1$

Table : Example of Z function

Algorithm

Z-Box

The algorithm relies on a single, crucial invariant. As we iterate over the letters in the string (index $\mathbf{i}$ from $\mathbf{1}$ to $\mathbf{n-1}$), we maintain an interval $[\mathbf{L}, \mathbf{R}]$ which is the interval with **maximum** $\mathbf{R}$ such that $\mathbf{1 \leq L \leq i \leq R}$ and $\mathbf{S[L...R]}$ is a **prefix-substring**.

Procedure

Now suppose we have the correct interval $[\mathbf{L}, \mathbf{R}]$ for $\mathbf{i} - \mathbf{1}$ and all of the $\mathbf{Z}$ values up to $\mathbf{i} - \mathbf{1}$. We will compute $\mathbf{Z}[\mathbf{i}]$ and the new $[\mathbf{L}, \mathbf{R}]$ by the following steps:

Procedure

Now suppose we have the correct interval $[\mathbf{L}, \mathbf{R}]$ for $\mathbf{i} - \mathbf{1}$ and all of the $\mathbf{Z}$ values up to $\mathbf{i} - \mathbf{1}$. We will compute $\mathbf{Z}[\mathbf{i}]$ and the new $[\mathbf{L}, \mathbf{R}]$ by the following steps:

If $\mathbf{i} > \mathbf{R}$

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

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

Now we have a few more cases to consider.

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

Now we have a few more cases to consider.

If $\mathbf{Z}[\mathbf{k}] < \mathbf{R} - \mathbf{i} + \mathbf{1}$

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

$\mathbf{Z[k] \geq R - i + 1}$

Then it is possible for $\mathbf{S[i\ldots]}$ to match $\mathbf{S[0\ldots]}$ for more than $\mathbf{R - i + 1}$ characters (i.e. past position $\mathbf{R}$). Thus we need to update $\mathbf{[L, R]}$ by setting $\mathbf{L = i}$ and matching from $\mathbf{S[R + 1]}$ forward to obtain the new $\mathbf{R}$. Again, we get $\mathbf{Z[i]}$ during this.

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

Example $\mathbf{S} = \mathbf{aabcaabxaaaz}$

v \i	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
S_i	a	a	b	c	a	a	b	x	a	a	a	z
L	0	1	2	3	4	4	4	7	8	9	9	11
R	0	1	1	2	6	6	6	6	9	10	10	11
k	-	-	-	-	-	1	2	-	-	-	1	-
$Z[k]$	-	-	-	-	-	1	0	-	-	-	1	-
$Z[i]$	0	1	0	0	3	1	0	0	2	2	1	0

Table : Example of Z algorithm

See algorithm running step by step in the following ▸ Link

Algorithm Code

Example (C++ Implementation)

```
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--;
    }
  }
}
```

Solution to our original problems

String matching

We can do this in $\mathbf{O}(\mathbf{n}+\mathbf{m})$ time by using the $\mathbf{Z}$ Algorithm on the string $\mathbf{P\$S}$ (that is, concatenating $\mathbf{P}$, \$, and $\mathbf{S}$) where \$ is a character that matches nothing. The indices $\mathbf{i}$ with $\mathbf{Z[i]}=\mathbf{m}$ correspond to matches of $\mathbf{P}$ in $\mathbf{S}$.

Problem B

We simply compute $\mathbf{Z}$ for the given string $\mathbf{S}$, then iterate from $\mathbf{i}$ to $\mathbf{n}-\mathbf{1}$. If $\mathbf{Z[i]}=\mathbf{n}-\mathbf{i}$ then we know the suffix from $\mathbf{S[i]}$ is a prefix, and if the largest $\mathbf{Z}$ value we've seen so far is at least $\mathbf{n}-\mathbf{i}$, then we know some string inside also matches that prefix.

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

Q & A

References

- http://codeforces.com/blog/entry/3107
- https://www.hackerrank.com/challenges/pairs/topics/two-pointer-technique
- http://e-maxx-eng.github.io/string/z-function.html