Z Algorithm

This algorithm finds all occurrences of a pattern in a input text in linear time. Let length of text be n and of pattern be m, then total time taken is $O(m+n)$ with *linear space complexity*.

Definition of prefix

Example
In word *apple* the prefix can be *apple* (or) *appl* (or) *app* (or) *ap* (or) *a*.
In word *banana* the prefix can be *banana* (or) *banan* (or) *bana* (or) *ban* (or) *ba* (or) *b*.

Explanation of prefix: Any substring *S* of a string *T* that matches from starting of the string *T* till end of string *T* or before the end is called as prefix.

How to build Z array

Lets take this example text : *a a b c a a b x a a a z*

Index	0	1	2	3	4	5	6	7	8	9
Text	a	a	b	$	b	a	a	b	a	a
Z values	X	1	0	0	0	3	1	0	2	1

Algorithm

Step 1:
At *index 0*, substring from 0th index till end is also prefix of given text.

a a b $ b a a b a a => of length 10 is the longest substring which is also a prefix of the text. But this will not help in pattern matching so we make it as **X** (or 0) in Z array.

Step 2:
At *index 1*, longest substring starting from position 1 till end which is also a prefix of the text are as follows

```
"a"         => prefix of the text "a a b $ b a a b a a" and length is 1.
"a b"       => Not a prefix
"a b $"     => Not a prefix
"a b $ b"   => Not a prefix
"a b $ b a" => Not a prefix
and so ...
```

Here the only longest substring which is also a prefix is "a" and its length is 1. It is stored in Z array.

At *index 2*, longest substring starting from position 2 till end which is also a prefix of the text are as follows

```
"b" => Not a prefix
"b $" => Not a prefix
"b $ b" => Not a prefix
and so on ...
```

Here there is no substring which is also a prefix of text T. So we are storing zero at index 2 in Z array.

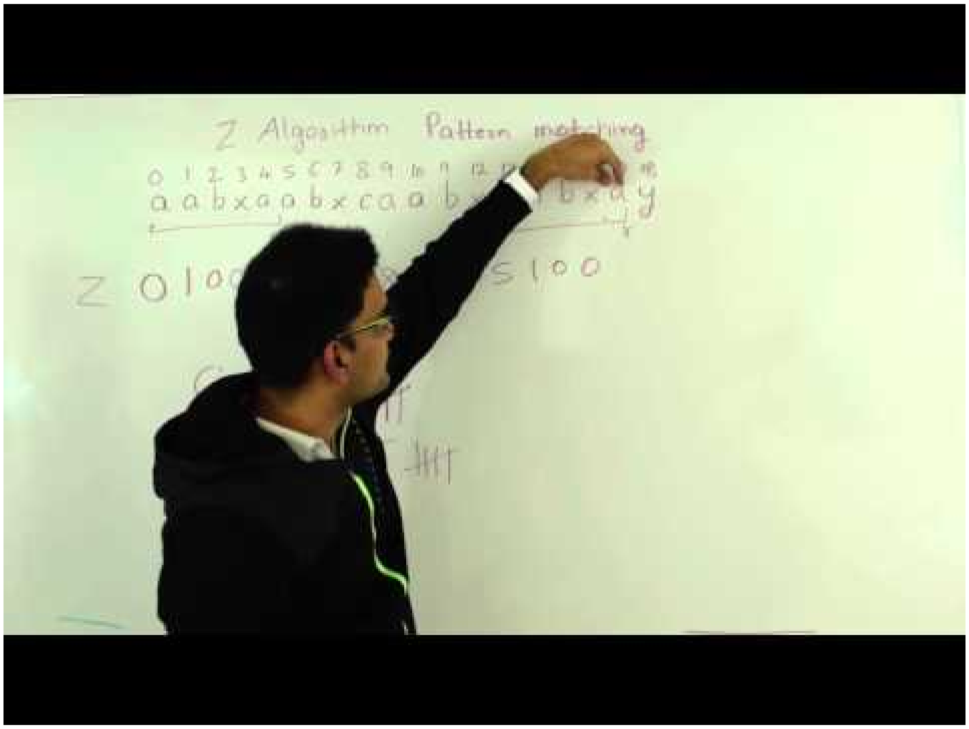
At *index 5* substrings are as follows

```
"a" => prefix of text "a a b $ b a a b a a" and length is 1.
"a a" => prefix of text "a a b $ b a a b a a" and length is 2.
"a a b" => prefix of text "a a b $ b a a b a a" and length is 3.
"a a b a"=> Not a prefix
and so on ...
```

Here the longest substring which is also a prefix of text T is "a a b" of length 3. So we store 3 at index 5 in Z array.

Step 3:
Finally, If any value in Z array is same as the length of the pattern then that pattern is present in the text T.

Video reference



Time and Space Complexity

Time and space complexity is same as KMP algorithm but this algorithm is simpler to understand.