Z-Algorithm

text:	**a**	**a**	**b**	**c**	**a**	**a**	**b**	**x**	**a**	**a**	**z**
k:	0	1	2	3	4	5	6	7	8	9	10

Compute Z function

```
Z[k]: length of longest prefix starting from text[k]
      Which is also a prefix in text[0...k-1]
```

Z-Algorithm

text:	a	a	b	c	a	a	b	x	a	a	z
k:	0	1	2	3	4	5	6	7	8	9	10
Z:	X										

Compute Z function

```
Z[k]: length of longest prefix starting from text[k]
      Which is also a prefix in text[0...k-1]
```

Naive calculation – compare parts

Z[1] = ?

0	1	2	3	4	5	6	7	8	9	10
a	a	b	c	a	a	b	x	a	a	z

a	a	b	c	a	a	b	x	a	a	z
0	1	2	3	4	5	6	7	8	9	10

Z[1] = 1

Z-Algorithm

text:	a	a	b	c	a	a	b	x	a	a	z
k:	0	1	2	3	4	5	6	7	8	9	10
Z:	X	1									

Compute Z function

```
Z[k]: length of longest prefix starting from text[k]
      Which is also a prefix in text[0...k-1]
```

Naive calculation – compare parts

Z[1] = ?

0	1	2	3	4	5	6	7	8	9	10
a	a	b	c	a	a	b	x	a	a	z

a	a	b	c	a	a	b	x	a	a	z
0	1	2	3	4	5	6	7	8	9	10

Z[1] = 1

Z-Algorithm

text:	a	a	b	c	a	a	b	x	a	a	z
k:	0	1	2	3	4	5	6	7	8	9	10
Z:	X	1	0								

Compute Z function

```
Z[k]: length of longest prefix starting from text[k]
      Which is also a prefix in text[0...k-1]
```

Naive calculation – compare parts

Z[2] = ?

0	1	2	3	4	5	6	7	8	9	10
a	a	b	c	a	a	b	x	a	a	z

a	a	b	c	a	a	b	x	a	a	z
0	1	2	3	4	5	6	7	8	9	10

Z[2] = 0

Z-Algorithm

text:	a	a	b	c	a	a	b	x	a	a	z
k:	0	1	2	3	4	5	6	7	8	9	10
Z:	X	1	0	0	3						

Compute Z function

```
Z[k]: length of longest prefix starting from text[k]
      Which is also a prefix in text[0...k-1]
```

Naive calculation – compare parts

Z[4] = ?

0	1	2	3	4	5	6	7	8	9	10
a	a	b	c	a	a	b	x	a	a	z

a	a	b	c	a	a	b	x	a	a	z
0	1	2	3	4	5	6	7	8	9	10

Z[4] = 3

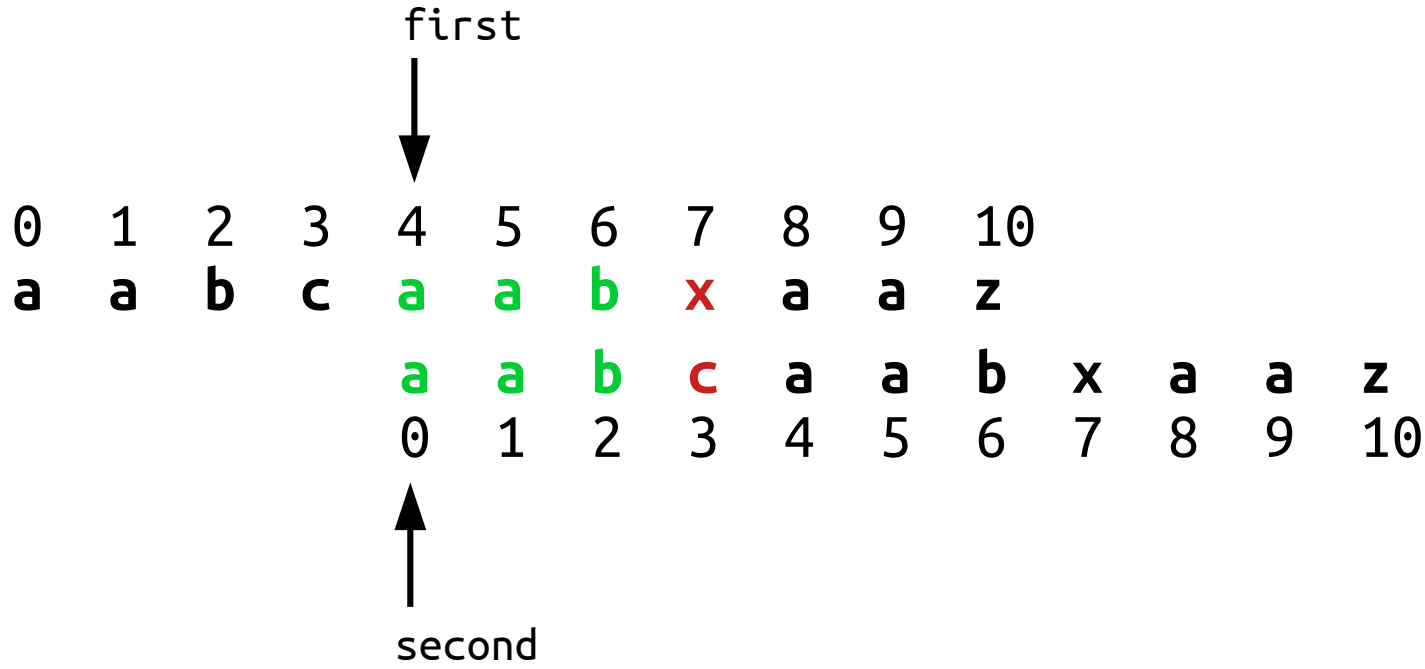
Z-Algorithm

Naive calculation for an element Z[k]

```
int compareParts(const std::string& s, const int first, const int second)
{
    int cnt = 0; // number of matched symbols!

    while(first + cnt < s.size() && second + cnt < s.size() && s[first + cnt] == s[second + cnt])
        cnt++;

    return cnt;
}
```



Z-Algorithm

How to use Z function to find a pattern in a text?

```
int find(const std::string& text, const std::string& pattern)
{
    std::string joined = pattern + "$" + text;

    std::vector<int> z = computeZ(joined);

    for(int i = pattern.size() + 1; i < joined.size(); i++)
    {
        if(z[i] == pattern.size())
        {
            return i - pattern.size() - 1;
        }
    }

    return -1;
}
```

text: aabcaabxaaz
pattern: aa

joined:	a	a	$	a	a	b	c	a	a	b	x	a	a	z
i:	0	1	2	3	4	5	6	7	8	9	10	11	12	13

Z-Algorithm

How to use Z function to find a pattern in a text?

```
int find(const std::string& text, const std::string& pattern)
{
    std::string joined = pattern + "$" + text;

    std::vector<int> z = computeZ(joined);

    for(int i = pattern.size() + 1; i < joined.size(); i++)
    {
        if(z[i] == pattern.size())
        {
            return i - pattern.size() - 1;
        }
    }

    return -1;
}
```

text: aabcaabxaaz
pattern: aa

joined:	**a**	**a**	**$**	**a**	**a**	**b**	**c**	**a**	**a**	**b**	**x**	**a**	**a**	**z**
i:	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Z:	X	1	0	**2**	1	0	0	**2**	1	0	0	**2**	1	0

Z-Algorithm

How to use Z function to find a pattern in a text?

```
int find(const std::string& text, const std::string& pattern)
{
    std::string joined = pattern + "$" + text;

    std::vector<int> z = computeZ(joined);

    for(int i = pattern.size() + 1; i < joined.size(); i++)
    {
        if(z[i] == pattern.size())
        {
            return i - pattern.size() - 1;
        }
    }

    return -1;
}
```

text: aabcaabxaaz
pattern: aa

joined:	**a**	**a**	**$**	**a**	**a**	**b**	**c**	**a**	**a**	**b**	**x**	**a**	**a**	**z**
i:	0	1	2	**3**	4	5	6	**7**	8	9	10	**11**	12	13
Z:	X	1	0	**2**	1	0	0	**2**	1	0	0	**2**	1	0

text:	**a**	**a**	b	c	**a**	**a**	b	x	**a**	**a**	z
	0	1	2	3	4	5	6	7	8	9	10

Z-Algorithm

How to use Z function to find a pattern in a text?

```
int find(const std::string& text, const std::string& pattern)
{
    std::string joined = pattern + "$" + text;

    std::vector<int> z = computeZ(joined);

    for(int i = pattern.size() + 1; i < joined.size(); i++)
    {
        if(z[i] == pattern.size())
        {
            return i - pattern.size() - 1;
        }
    }

    return -1;
}
```

If all elements in Z are calculated naively, then time complexity of computeZ will be quadratic!

How to calculate elements in Z efficiently?

Z-Algorithm

k

text:	a	a	b	c	a	a	b	x	a	a	z
	0	1	2	3	4	5	6	7	8	9	10
	L	R									

Z: X

```
std::vector<int> computeZ(const std::string& s)
{
    std::vector<int> z(s.size(), 0);

    int L = 0, R = 0;

    for(int k = 1; k < s.size(); k++)
    {
        if(k > R)
        {
            int cnt = compareParts(s, k, 0);
            z[k] = cnt;
            L = k;
            R = k + cnt - 1;
```

Z-Algorithm

k

text:	a	a	b	c	a	a	b	x	a	a	z
	0	1	2	3	4	5	6	7	8	9	10
	L	R									

Z: X

0	1	2	3	4	5	6	7	8	9	10
a	a	b	c	a	a	b	x	a	a	z
	a	a	b	c	a	a	b	x	a	a	z
	0	1	2	3	4	5	6	7	8	9	10

cnt = 1

```
std::vector<int> computeZ(const std::string& s)
{
    std::vector<int> z(s.size(), 0);

    int L = 0, R = 0;

    for(int k = 1; k < s.size(); k++)
    {
        if(k > R)
        {
            int cnt = compareParts(s, k, 0);
            z[k] = cnt;
            L = k;
            R = k + cnt - 1;
```

Z-Algorithm

k

text:	a	a	b	c	a	a	b	x	a	a	z
	0	1	2	3	4	5	6	7	8	9	10
	L		R								

Z:	X	1

0	1	2	3	4	5	6	7	8	9	10
a	a	b	c	a	a	b	x	a	a	z

a	a	b	c	a	a	b	x	a	a	z
0	1	2	3	4	5	6	7	8	9	10

cnt = 1

```
std::vector<int> computeZ(const std::string& s)
{
    std::vector<int> z(s.size(), 0);

    int L = 0, R = 0;

    for(int k = 1; k < s.size(); k++)
    {
        if(k > R)
        {
            int cnt = compareParts(s, k, 0);
            z[k] = cnt;
            L = k;
            R = k + cnt - 1;
```

Z-Algorithm

k

text:	a	a	b	c	a	a	b	x	a	a	z
	0	1	2	3	4	5	6	7	8	9	10
	L		R								

Z: X 1

```
std::vector<int> computeZ(const std::string& s)
{
    std::vector<int> z(s.size(), 0);

    int L = 0, R = 0;

    for(int k = 1; k < s.size(); k++)
    {
        if(k > R)
        {
            int cnt = compareParts(s, k, 0);
            z[k] = cnt;
            L = k;
            R = k + cnt - 1;
```

Z-Algorithm

k

text:	a	a	b	c	a	a	b	x	a	a	z
	0	1	2	3	4	5	6	7	8	9	10
		R	L								
Z:	X	1	0								

0	1	2	3	4	5	6	7	8	9	10
a	a	b	c	a	a	b	x	a	a	z

a	a	b	c	a	a	b	x	a	a	z
0	1	2	3	4	5	6	7	8	9	10

cnt = 0

```
std::vector<int> computeZ(const std::string& s)
{
    std::vector<int> z(s.size(), 0);

    int L = 0, R = 0;

    for(int k = 1; k < s.size(); k++)
    {
        if(k > R)
        {
            int cnt = compareParts(s, k, 0);
            z[k] = cnt;
            L = k;
            R = k + cnt - 1;
```

Z-Algorithm

k

text:	a	a	b	c	a	a	b	x	a	a	z
	0	1	2	3	4	5	6	7	8	9	10
		R	L								
Z:	X	1	0								

```
std::vector<int> computeZ(const std::string& s)
{
    std::vector<int> z(s.size(), 0);

    int L = 0, R = 0;

    for(int k = 1; k < s.size(); k++)
    {
        if(k > R)
        {
            int cnt = compareParts(s, k, 0);
            z[k] = cnt;
            L = k;
            R = k + cnt - 1;
```

Z-Algorithm

k

text:	a	a	b	c	a	a	b	x	a	a	z
	0	1	2	3	4	5	6	7	8	9	10
			R	L							
Z:	X	1	0	0							

```
std::vector<int> computeZ(const std::string& s)
{
    std::vector<int> z(s.size(), 0);

    int L = 0, R = 0;

    for(int k = 1; k < s.size(); k++)
    {
        if(k > R)
        {
            int cnt = compareParts(s, k, 0);
            z[k] = cnt;
            L = k;
            R = k + cnt - 1;
```

Z-Algorithm

k

text:	a	a	b	c	a	a	b	x	a	a	z
	0	1	2	3	4	5	6	7	8	9	10
			R	L							
Z:	X	1	0	0							

0	1	2	3	4	5	6	7	8	9	10
a	a	b	c	a	a	b	x	a	a	z

a	a	b	c	a	a	b	x	a	a	z
0	1	2	3	4	5	6	7	8	9	10

cnt = 3

```
std::vector<int> computeZ(const std::string& s)
{
    std::vector<int> z(s.size(), 0);

    int L = 0, R = 0;

    for(int k = 1; k < s.size(); k++)
    {
        if(k > R)
        {
            int cnt = compareParts(s, k, 0);
            z[k] = cnt;
            L = k;
            R = k + cnt - 1;
```

Z-Algorithm

k

text:	a	a	b	c	a	a	b	x	a	a	z
	0	1	2	3	4	5	6	7	8	9	10
					L		R				
Z:	X	1	0	0	3						

0	1	2	3	4	5	6	7	8	9	10
a	a	b	c	a	a	b	x	a	a	z
				a	a	b	c	a	a	b	x	a	a	z
				0	1	2	3	4	5	6	7	8	9	10

cnt = 3

```
std::vector<int> computeZ(const std::string& s)
{
    std::vector<int> z(s.size(), 0);

    int L = 0, R = 0;

    for(int k = 1; k < s.size(); k++)
    {
        if(k > R)
        {
            int cnt = compareParts(s, k, 0);
            z[k] = cnt;
            L = k;
            R = k + cnt - 1;
```

Z-Algorithm

k

text:	a	a	b	c	a	a	b	x	a	a	z
	0	1	2	3	4	5	6	7	8	9	10
					L		R				
Z:	X	1	0	0	3						

```
}else // k <= R
{
    if(z[k - L] < R-k+1)
    {
        z[k] = z[k - L];
    }else
    {
        int cnt = compareParts(s, R+1, R-L+1);
        z[k] = (R - k + 1) + cnt;
        L = k;
        R = R + z[k] - 1;
    }
}
```

Re-use previous z! :-)

Z-Algorithm

		k-L				k					
text:	a	a	b	c	a	a	b	x	a	a	z
	0	1	2	3	4	5	6	7	8	9	10
					L		R				
Z:	X	1	0	0	3						

```
}else // k <= R
{
    if(z[k - L] < R-k+1)
    {
        z[k] = z[k - L];
    }else
    {
        int cnt = compareParts(s, R+1, R-L+1);
        z[k] = (R - k + 1) + cnt;
        L = k;
        R = R + z[k] - 1;
    }
}
```

Re-use previous z! :-)

Z-Algorithm

		k-L				k					
text:	a	a	b	c	a	a	b	x	a	a	z
	0	1	2	3	4	5	6	7	8	9	10
					L		R				
Z:	X	1	0	0	3	1					

```
}else // k <= R
{
    if(z[k - L] < R-k+1)
    {
        z[k] = z[k - L];
    }else
    {
        int cnt = compareParts(s, R+1, R-L+1);
        z[k] = (R - k + 1) + cnt;
        L = k;
        R = R + z[k] - 1;
    }
}
```

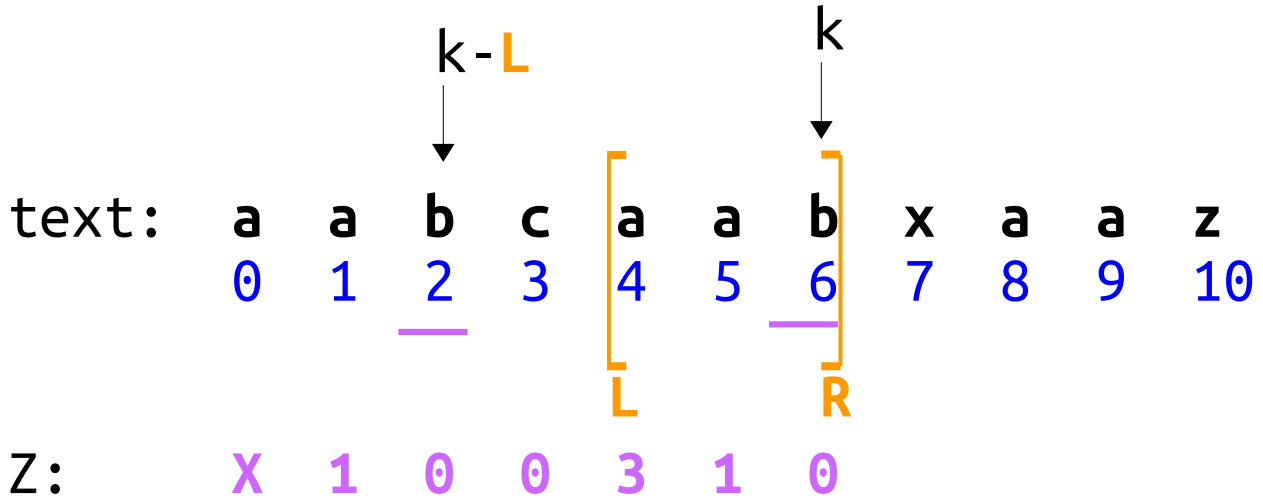
Re-use previous z! :-)

Z-Algorithm

			k-L				k				
text:	a	a	b	c	a	a	b	x	a	a	z
	0	1	2	3	4	5	6	7	8	9	10
					L		R				
Z:	X	1	0	0	3	1					

```
}else // k <= R
{
    if(z[k - L] < R-k+1)
    {
        z[k] = z[k - L]; Re-use previous z! :-)
    }else
    {
        int cnt = compareParts(s, R+1, R-L+1);
        z[k] = (R - k + 1) + cnt;
        L = k;
        R = R + cnt - 1;
    }
}
```

Z-Algorithm




```
}else // k <= R
{
    if(z[k - L] < R-k+1)
    {
        z[k] = z[k - L];
    }else
    {
        int cnt = compareParts(s, R+1, R-L+1);
        z[k] = (R - k + 1) + cnt;
        L = k;
        R = R + z[k] - 1;
    }
}
```

Re-use previous z! :-)

Z-Algorithm

k

text:	a	a	b	c	a	a	b	x	a	a	z
	0	1	2	3	4	5	6	7	8	9	10
					L		R				
Z:	X	1	0	0	3	1	0	0			

0	1	2	3	4	5	6	7	8	9	10
a	a	b	c	a	a	b	x	a	a	z

a	a	b	c	a	a	b	x	a	a	z
0	1	2	3	4	5	6	7	8	9	10

cnt = 0

```
std::vector<int> computeZ(const std::string& s)
{
    std::vector<int> z(s.size(), 0);

    int L = 0, R = 0;

    for(int k = 1; k < s.size(); k++)
    {
        if(k > R)
        {
            int cnt = compareParts(s, k, 0);
            z[k] = cnt;
            L = k;
            R = k + cnt - 1;
```

Z-Algorithm

k

text:	a	a	b	c	a	a	b	x	a	a	z
	0	1	2	3	4	5	6	7	8	9	10
							R	L			
Z:	X	1	0	0	3	1	0	0			

0	1	2	3	4	5	6	7	8	9	10
a	a	b	c	a	a	b	x	a	a	z

a	a	b	c	a	a	b	x	a	a	z
0	1	2	3	4	5	6	7	8	9	10

cnt = 0

```
std::vector<int> computeZ(const std::string& s)
{
    std::vector<int> z(s.size(), 0);

    int L = 0, R = 0;

    for(int k = 1; k < s.size(); k++)
    {
        if(k > R)
        {
            int cnt = compareParts(s, k, 0);
            z[k] = cnt;
            L = k;
            R = k + cnt - 1;
```

Z-Algorithm

k

text:	a	a	b	c	a	a	b	x	a	a	z
	0	1	2	3	4	5	6	7	8	9	10
							R	L			
Z:	X	1	0	0	3	1	0	0	2		

0	1	2	3	4	5	6	7	8	9	10
a	a	b	c	a	a	b	x	a	a	z

a	a	b	c	a	a	b	x	a	a	z
0	1	2	3	4	5	6	7	8	9	10

cnt = 2

```
std::vector<int> computeZ(const std::string& s)
{
    std::vector<int> z(s.size(), 0);

    int L = 0, R = 0;

    for(int k = 1; k < s.size(); k++)
    {
        if(k > R)
        {
            int cnt = compareParts(s, k, 0);
            z[k] = cnt;
            L = k;
            R = k + cnt - 1;
```

Z-Algorithm

k

text:	a	a	b	c	a	a	b	x	a	a	z
	0	1	2	3	4	5	6	7	8	9	10
									L	R	
Z:	X	1	0	0	3	1	0	0	2		

0	1	2	3	4	5	6	7	8	9	10
a	a	b	c	a	a	b	x	a	a	z

a	a	b	c	a	a	b	x	a	a	z
0	1	2	3	4	5	6	7	8	9	10

cnt = 2

```
std::vector<int> computeZ(const std::string& s)
{
    std::vector<int> z(s.size(), 0);

    int L = 0, R = 0;

    for(int k = 1; k < s.size(); k++)
    {
        if(k > R)
        {
            int cnt = compareParts(s, k, 0);
            z[k] = cnt;
            L = k;
            R = k + cnt - 1;
```

Z-Algorithm

k-L k

text:	a	a	b	c	a	a	b	x	a	a	z
	0	1	2	3	4	5	6	7	8	9	10
									L	R	
Z:	X	1	0	0	3	1	0	0	2		

```
}else // k <= R
{
    if(z[k - L] < R-k+1)
    {
        z[k] = z[k - L];
    }else
    {
        int cnt = compareParts(s, R+1, R-L+1);
        z[k] = (R - k + 1) + cnt;
        L = k;
        R = R + z[k] - 1;
    }
}
```

We fall in this case! =>

Z-Algorithm

k-L

k

text:	a	a	b	c	a	a	b	x	a	a	z
	0	1	2	3	4	5	6	7	8	9	10
										L	R
Z:	X	1	0	0	3	1	0	0	2	1	

R+1

0	1	2	3	4	5	6	7	8	9	10
a	a	b	c	a	a	b	x	a	a	z
				a	a	b	c	a	a	b	x	a	a	z
				0	1	2	3	4	5	6	7	8	9	10

R-L+1

cnt = 0

We fall in this case! =>

```
}else // k <= R
{
    if(z[k - L] < R-k+1)
    {
        z[k] = z[k - L];
    }else
    {
        int cnt = compareParts(s, R+1, R-L+1);
        z[k] = (R - k + 1) + cnt;
        L = k;
        R = R + z[k] - 1;
    }
}
```

Z[9] = (9 - 9 + 1) + 0 = 1

Z-Algorithm

k

text:	a	a	b	c	a	a	b	x	a	a	z
	0	1	2	3	4	5	6	7	8	9	10
										L	R
Z:	X	1	0	0	3	1	0	0	2	1	0

```
std::vector<int> computeZ(const std::string& s)
{
    std::vector<int> z(s.size(), 0);

    int L = 0, R = 0;

    for(int k = 1; k < s.size(); k++)
    {
        if(k > R)
        {
            int cnt = compareParts(s, k, 0);
            z[k] = cnt;
            L = k;
            R = k + cnt - 1;
```