KMP-Algorithm

Finding a pattern in a text
The idea is similar to Z-Algorithm
Re-use "information" from previous windows.

Look for pat **in** text[0...]

```
text:   a  a  a  a  a  b  a  a  a  b  a
        0  1  2  3  4  5  6  7  8  9  10

pat:    a  a  a  a
```

pat **found in position 0 of text!**

Now, we start to look for pat **in** text[1...]

KMP-Algorithm

Finding a pattern in a text
The idea is similar to Z-Algorithm
Re-use “information” from previous windows.

Look for pat **in** text[0...]

```
text:  a  a  a  a  a  b  a  a  a  b  a
       0  1  2  3  4  5  6  7  8  9  10

pat:   a  a  a  a
```

pat **found in position 0 of text!**

Now, we start to look for pat **in** text[1...]

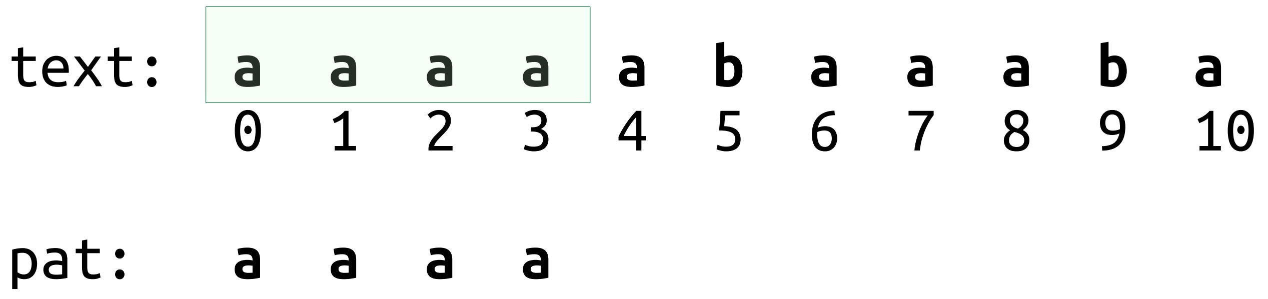
```
text:  a  a  a  a  a  b  a  a  a  b  a
       0  1  2  3  4  5  6  7  8  9  10

pat:      a  a  a  a
```

If we save “information” from previous iteration, we then already know that text[1…3] = pat[0..2]
So we only need to compare text[4] = part[3]

KMP-Algorithm

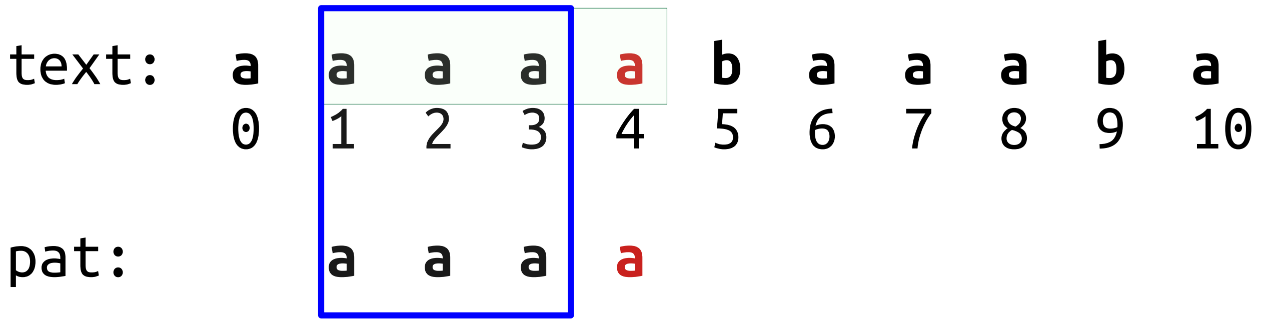
Finding a pattern in a text
The idea is similar to Z-Algorithm
Re-use “information” from previous windows.

Look for pat **in** text[0...]



pat **found in position 0 of text!**

Now, we start to look for pat **in** text[1...]



If we save “information” from previous iteration, we then already know that text[1…3] = pat[0..2]
So we only need to compare text[4] = part[3]

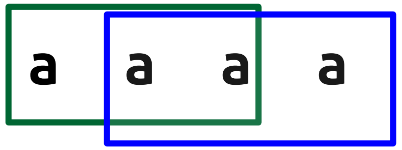
KMP-Algorithm

How many characters to skip?
To know this, we construct a vector `lps` **of** `pat.size()`

```
lps[i] = size of longest prefix of pat[0...i] which is also
a suffix of pat[0...i]
```

```
pat:   a  a  a  a
```

lps[3] = **longest prefix** of lps[0...3] which is also a suffix of pat[0...3]



```
lps[3] = 3
```

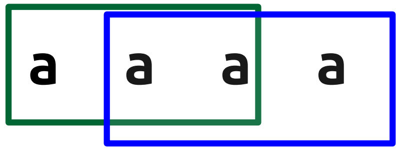
KMP-Algorithm

How many characters to skip?
To know this, we construct a vector `lps` **of** `pat.size()`

```
lps[i] = size of longest prefix of pat[0...i] which is also
a suffix of pat[0...i]
```

```
pat:   a  a  a  a
```

lps[3] = **longest prefix** of lps[0...3] which is also a suffix of pat[0...3]



```
lps[3] = 3
```

i	longest prefix/suffix in pat[0..i]	size
0		lps[0] =
1		lps[1] =
2		lps[2] =
3		lps[3] =

KMP-Algorithm

How many characters to skip?
To know this, we construct a vector lps **of** pat.size()

lps[i] = size of longest prefix of pat[0...i] which is also a suffix of pat[0...i]

pat:	a	a	b	a	a	c	a	a	b	a	a
	0	1	2	3	4	5	6	7	8	9	10

i	longest prefix/suffix in pat[0..i]	size
0		lps[0] =
1		lps[1] =
2		lps[2] =
3		lps[3] =
…		…
8		lps[8] =
9		lps[9] =
10		lps[10] =

KMP-Algorithm

```
pat:   a  a  a  c  a  a  a  a
       0  1  2  3  4  5  6  7
lps:
```

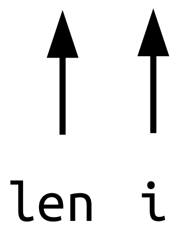
```
std::vector<int> computeLPS(const std::string& pat)
{
    std::vector<int> lps;
    for(int i = 0; i < pat.size(); i++)
        lps.push_back(0);

    // length of longest prefix/suffix in pat
    int len = 0;

    lps[0] = 0;

    int i = 1;

    while( i < pat.size() )
    {
        if( pat[i] == pat[len] )
        {
            len++;
            lps[i] = len;
            i++;
        }
        else // pat[i] != pat[len]
        {
            if(len != 0)
            {
                len = lps[len - 1];
            }
            else // len == 0
            {
                lps[i] = 0;
                i++;
            }
        }
    }

    return lps;
}
```

KMP-Algorithm

```
pat:    a  a  a  c  a  a  a  a
        0  1  2  3  4  5  6  7
lps:    0
```

i=1:

```
pat:    a  a  ..
        0  1
```



```
pat[i] = pat[len]
len = 1
lps[i] = 1
i++;
```

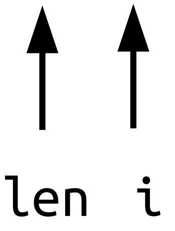
```cpp
std::vector<int> computeLPS(const std::string& pat)
{
    std::vector<int> lps;
    for(int i = 0; i < pat.size(); i++)
        lps.push_back(0);

    // length of longest prefix/suffix in pat
    int len = 0;

    lps[0] = 0;

    int i = 1;

    while( i < pat.size() )
    {
        if( pat[i] == pat[len] )
        {
            len++;
            lps[i] = len;
            i++;
        }
        else // pat[i] != pat[len]
        {
            if(len != 0)
            {
                len = lps[len - 1];
            }
            else // len == 0
            {
                lps[i] = 0;
                i++;
            }
        }
    }

    return lps;
}
```

KMP-Algorithm

```
pat:    a  a  a  c  a  a  a  a
        0  1  2  3  4  5  6  7
lps:    0  1
```

i=2:

```
pat:    a  a  a ..
        0  1  2
```



```
pat[i] = pat[len]
len = 2
lps[i] = 2
i++;
```

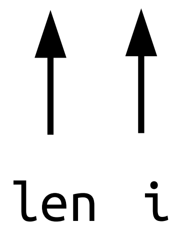
```cpp
std::vector<int> computeLPS(const std::string& pat)
{
    std::vector<int> lps;
    for(int i = 0; i < pat.size(); i++)
        lps.push_back(0);

    // length of longest prefix/suffix in pat
    int len = 0;

    lps[0] = 0;

    int i = 1;

    while( i < pat.size() )
    {
        if( pat[i] == pat[len] )
        {
            len++;
            lps[i] = len;
            i++;
        }
        else // pat[i] != pat[len]
        {
            if(len != 0)
            {
                len = lps[len - 1];
            }
            else // len == 0
            {
                lps[i] = 0;
                i++;
            }
        }
    }

    return lps;
}
```

KMP-Algorithm

```
pat:    a  a  a  c  a  a  a  a
        0  1  2  3  4  5  6  7
lps:    0  1  2
```

i=3:

```
pat:    a  a  a  c  ..
        0  1  2  3
```



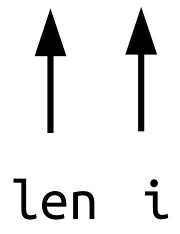
pat[i] != pat[len]

```cpp
std::vector<int> computeLPS(const std::string& pat)
{
    std::vector<int> lps;
    for(int i = 0; i < pat.size(); i++)
        lps.push_back(0);

    // length of longest prefix/suffix in pat
    int len = 0;

    lps[0] = 0;

    int i = 1;

    while( i < pat.size() )
    {
        if( pat[i] == pat[len] )
        {
            len++;
            lps[i] = len;
            i++;
        }
        else // pat[i] != pat[len]
        {
            if(len != 0)
            {
                len = lps[len - 1];
            }
            else // len == 0
            {
                lps[i] = 0;
                i++;
            }
        }
    }

    return lps;
}
```

KMP-Algorithm

```
pat:    a  a  a  c  a  a  a  a
        0  1  2  3  4  5  6  7
lps:    0  1  2
```

```
i=3:

pat:    a  a  a  c  ..
        0  1  2  3
```



```
pat[i] != pat[len]

len = lps[len - 1] = 1
```

Move back pointer “len” to the last character of previous longest/prefix suffix

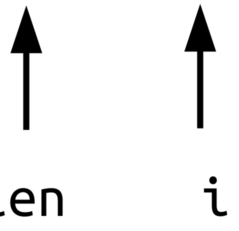
```cpp
std::vector<int> computeLPS(const std::string& pat)
{
    std::vector<int> lps;
    for(int i = 0; i < pat.size(); i++)
        lps.push_back(0);

    // length of longest prefix/suffix in pat
    int len = 0;

    lps[0] = 0;

    int i = 1;

    while( i < pat.size() )
    {
        if( pat[i] == pat[len] )
        {
            len++;
            lps[i] = len;
            i++;
        }
        else // pat[i] != pat[len]
        {
            if(len != 0)
            {
                len = lps[len - 1];
            }
            else // len == 0
            {
                lps[i] = 0;
                i++;
            }
        }
    }

    return lps;
}
```

KMP-Algorithm

```
pat:    a  a  a  c  a  a  a  a
        0  1  2  3  4  5  6  7
lps:    0  1  2
```

i=3:

```
pat:    a  a  a  c  ..
        0  1  2  3
```



pat[i] != pat[len]

len = lps[len - 1] = 0

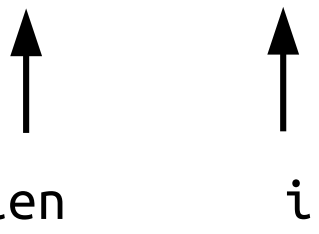
```cpp
std::vector<int> computeLPS(const std::string& pat)
{
    std::vector<int> lps;
    for(int i = 0; i < pat.size(); i++)
        lps.push_back(0);

    // length of longest prefix/suffix in pat
    int len = 0;

    lps[0] = 0;

    int i = 1;

    while( i < pat.size() )
    {
        if( pat[i] == pat[len] )
        {
            len++;
            lps[i] = len;
            i++;
        }
        else // pat[i] != pat[len]
        {
            if(len != 0)
            {
                len = lps[len - 1];
            }
            else // len == 0
            {
                lps[i] = 0;
                i++;
            }
        }
    }

    return lps;
}
```

KMP-Algorithm

```
pat:  a  a  a  c  a  a  a  a
      0  1  2  3  4  5  6  7
lps:  0  1  2  0
```

i=3:

```
pat:  a  a  a  c  ..
      0  1  2  3
```



pat[i] != pat[len] and len == 0

lps[3] = 0

```cpp
std::vector<int> computeLPS(const std::string& pat)
{
    std::vector<int> lps;
    for(int i = 0; i < pat.size(); i++)
        lps.push_back(0);

    // length of longest prefix/suffix in pat
    int len = 0;

    lps[0] = 0;

    int i = 1;

    while( i < pat.size() )
    {
        if( pat[i] == pat[len] )
        {
            len++;
            lps[i] = len;
            i++;
        }
        else // pat[i] != pat[len]
        {
            if(len != 0)
            {
                len = lps[len - 1];
            }
            else // len == 0
            {
                lps[i] = 0;
                i++;
            }
        }
    }

    return lps;
}
```

KMP-Algorithm

```
pat:   a  a  a  c  a  a  a  a
       0  1  2  3  4  5  6  7
lps:   0  1  2  0  1
```

```
i=4:

pat:   a  a  a  c  a  ...
       0  1  2  3  4
       ↑           ↑
      len          i
```

```
pat[i] = pat[len]
len = 1
lps[i] = 1
i++;
```

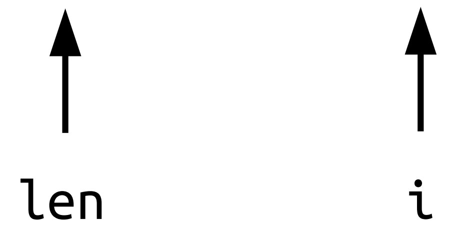
```cpp
std::vector<int> computeLPS(const std::string& pat)
{
    std::vector<int> lps;
    for(int i = 0; i < pat.size(); i++)
        lps.push_back(0);

    // length of longest prefix/suffix in pat
    int len = 0;

    lps[0] = 0;

    int i = 1;

    while( i < pat.size() )
    {
        if( pat[i] == pat[len] )
        {
            len++;
            lps[i] = len;
            i++;
        }
        else // pat[i] != pat[len]
        {
            if(len != 0)
            {
                len = lps[len - 1];
            }
            else // len == 0
            {
                lps[i] = 0;
                i++;
            }
        }
    }

    return lps;
}
```

KMP-Algorithm

```
pat:    a  a  a  c  a  a  a  a
        0  1  2  3  4  5  6  7
Lps:    0  1  2  0  1  2
```

i=5:

```
pat:    a  a  a  c  a  a ...
        0  1  2  3  4  5
```



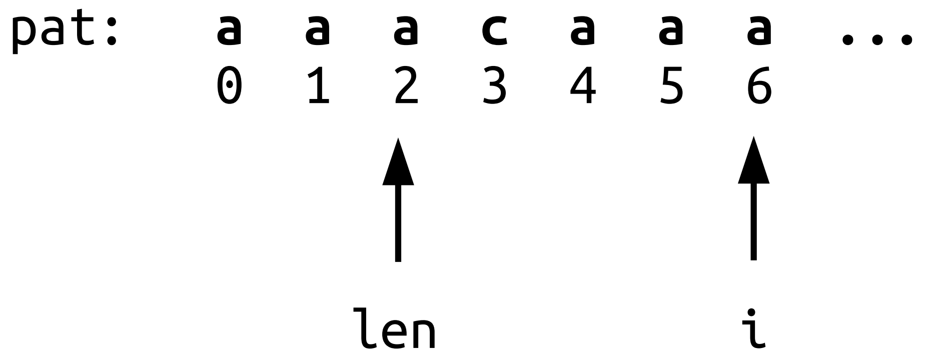
```
pat[i] = pat[len]
len = 2
lps[i] = 2
i++;
```

```cpp
std::vector<int> computeLPS(const std::string& pat)
{
    std::vector<int> lps;
    for(int i = 0; i < pat.size(); i++)
        lps.push_back(0);

    // length of longest prefix/suffix in pat
    int len = 0;

    lps[0] = 0;

    int i = 1;

    while( i < pat.size() )
    {
        if( pat[i] == pat[len] )
        {
            len++;
            lps[i] = len;
            i++;
        }
        else // pat[i] != pat[len]
        {
            if(len != 0)
            {
                len = lps[len - 1];
            }
            else // len == 0
            {
                lps[i] = 0;
                i++;
            }
        }
    }

    return lps;
}
```

KMP-Algorithm

pat:	a	a	a	c	a	a	a	a
	0	1	2	3	4	5	6	7
lps:	0	1	2	0	1	2	3	

i=6:



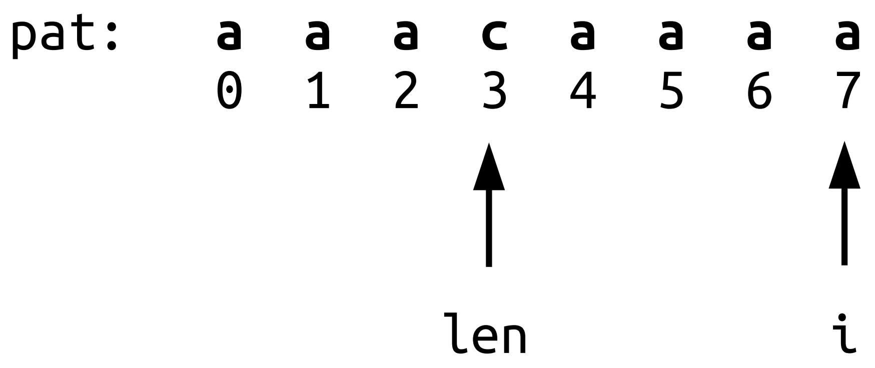
```
pat[i] = pat[len]
len = 3
lps[i] = 3
i++;
```

```cpp
std::vector<int> computeLPS(const std::string& pat)
{
    std::vector<int> lps;
    for(int i = 0; i < pat.size(); i++)
        lps.push_back(0);

    // length of longest prefix/suffix in pat
    int len = 0;

    lps[0] = 0;

    int i = 1;

    while( i < pat.size() )
    {
        if( pat[i] == pat[len] )
        {
            len++;
            lps[i] = len;
            i++;
        }
        else // pat[i] != pat[len]
        {
            if(len != 0)
            {
                len = lps[len - 1];
            }
            else // len == 0
            {
                lps[i] = 0;
                i++;
            }
        }
    }

    return lps;
}
```

KMP-Algorithm

pat:	a	a	a	c	a	a	a	a
	0	1	2	3	4	5	6	7
lps:	0	1	2	0	1	2	3	

i=7:



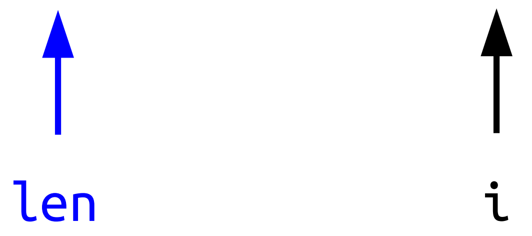
pat[i] != pat[len]

```cpp
std::vector<int> computeLPS(const std::string& pat)
{
    std::vector<int> lps;
    for(int i = 0; i < pat.size(); i++)
        lps.push_back(0);

    // length of longest prefix/suffix in pat
    int len = 0;

    lps[0] = 0;

    int i = 1;

    while( i < pat.size() )
    {
        if( pat[i] == pat[len] )
        {
            len++;
            lps[i] = len;
            i++;
        }
        else // pat[i] != pat[len]
        {
            if(len != 0)
            {
                len = lps[len - 1];
            }
            else // len == 0
            {
                lps[i] = 0;
                i++;
            }
        }
    }

    return lps;
}
```

KMP-Algorithm

```
pat:    a  a  a  c  a  a  a  a
        0  1  2  3  4  5  6  7
lps:    0  1  2  0  1  2  3
```

```
i=7:

pat:    a  a  a  c  a  a  a  a
        0  1  2  3  4  5  6  7
```



```
pat[i] != pat[len]

len = lps[len - 1] = 1
    = lps[ 3  - 1] = 2
```

Move back pointer “len” to the
last character of previous
longest/prefix suffix

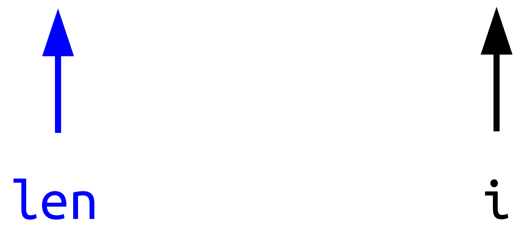
```cpp
std::vector<int> computeLPS(const std::string& pat)
{
    std::vector<int> lps;
    for(int i = 0; i < pat.size(); i++)
        lps.push_back(0);

    // length of longest prefix/suffix in pat
    int len = 0;

    lps[0] = 0;

    int i = 1;

    while( i < pat.size() )
    {
        if( pat[i] == pat[len] )
        {
            len++;
            lps[i] = len;
            i++;
        }
        else // pat[i] != pat[len]
        {
            if(len != 0)
            {
                len = lps[len - 1];
            }
            else // len == 0
            {
                lps[i] = 0;
                i++;
            }
        }
    }

    return lps;
}
```

KMP-Algorithm

pat:	a	a	a	c	a	a	a	a
	0	1	2	3	4	5	6	7
Lps:	0	1	2	0	1	2	3	3

i=7:

pat:	a	a	a	c	a	a	a	a
	0	1	2	3	4	5	6	7



```
pat[i] = pat[len]
len = 3
lps[i] = 3
i++;
```

```cpp
std::vector<int> computeLPS(const std::string& pat)
{
    std::vector<int> lps;
    for(int i = 0; i < pat.size(); i++)
        lps.push_back(0);

    // length of longest prefix/suffix in pat
    int len = 0;

    lps[0] = 0;

    int i = 1;

    while( i < pat.size() )
    {
        if( pat[i] == pat[len] )
        {
            len++;
            lps[i] = len;
            i++;
        }
        else // pat[i] != pat[len]
        {
            if(len != 0)
            {
                len = lps[len - 1];
            }
            else // len == 0
            {
                lps[i] = 0;
                i++;
            }
        }
    }

    return lps;
}
```