Fast pattern matching in strings - KMP

Malte Laukötter

Inhalt

1. Problem
2. Primitive Lösung
3. Idee hinter dem Algorithmus
4. Algorithmus
5. Komplexität
6. Geschichte

Problem

Suche von Pattern in Text

Pattern: *in*

Text: *Dies ist ein Satz.*

Ergebnis: *Dies ist ein Satz.*

Anwendungen

DNA Analyse

Text Editor

Suchmaschienen

Primitive Lösung

Start bei jedem Text Buchstaben

Buchstabe für Buchstabe vergleichen

A	A	A	B	A	A	B	A	A	B	A	C
A	B	A	A	B	A	C					

Primitive Lösung

Start bei jedem Text Buchstaben

Buchstabe für Buchstabe vergleichen

A	A	A	B	A	A	B	A	A	B	A	C
A	B	A	A	B	A	C					

Primitive Lösung

Start bei jedem Text Buchstaben

Buchstabe für Buchstabe vergleichen

A	A	A	B	A	A	B	A	A	B	A	C
A	B	A	A	B	A	C					

Primitive Lösung

Start bei jedem Text Buchstaben

Buchstabe für Buchstabe vergleichen

A	A	A	B	A	A	B	A	A	B	A	C
	A	B	A	A	B	A	C				

Primitive Lösung

Start bei jedem Text Buchstaben

Buchstabe für Buchstabe vergleichen

A	A	A	B	A	A	B	A	A	B	A	C
	A	B	A	A	B	A	C				

Primitive Lösung

Start bei jedem Text Buchstaben

Buchstabe für Buchstabe vergleichen

A	A	A	B	A	A	B	A	A	B	A	C
		A	B	A	A	B	A	C			

Primitive Lösung

Start bei jedem Text Buchstaben

Buchstabe für Buchstabe vergleichen

A	A	A	B	A	A	B	A	A	B	A	C
		A	B	A	A	B	A	C			

Primitive Lösung

Start bei jedem Text Buchstaben

Buchstabe für Buchstabe vergleichen

A	A	A	B	A	A	B	A	A	B	A	C
		A	B	A	A	B	A	C			

Primitive Lösung

Start bei jedem Text Buchstaben

Buchstabe für Buchstabe vergleichen

A	A	A	B	A	A	B	A	A	B	A	C
		A	B	A	A	B	A	C			

Primitive Lösung

Start bei jedem Text Buchstaben

Buchstabe für Buchstabe vergleichen

A	A	A	B	A	A	B	A	A	B	A	C
		A	B	A	A	B	A	C			

Primitive Lösung

Start bei jedem Text Buchstaben

Buchstabe für Buchstabe vergleichen

A	A	A	B	A	A	B	A	A	B	A	C
		A	B	A	A	B	A	C			

Primitive Lösung

Start bei jedem Text Buchstaben

Buchstabe für Buchstabe vergleichen

A	A	A	B	A	A	B	A	A	B	A	C
		A	B	A	A	B	A	C			

Primitive Lösung

Start bei jedem Text Buchstaben

Buchstabe für Buchstabe vergleichen

A	A	A	B	A	A	B	A	A	B	A	C
			A	B	A	A	B	A	C		

Primitive Lösung

Start bei jedem Text Buchstaben

Buchstabe für Buchstabe vergleichen

A	A	A	B	A	A	B	A	A	B	A	C
				A	B	A	A	B	A	C	

Primitive Lösung

Start bei jedem Text Buchstaben

Buchstabe für Buchstabe vergleichen

A	A	A	B	A	A	B	A	A	B	A	C
				A	B	A	A	B	A	C	

Primitive Lösung

Start bei jedem Text Buchstaben

Buchstabe für Buchstabe vergleichen

A	A	A	B	A	A	B	A	A	B	A	C
					A	B	A	A	B	A	C

Primitive Lösung

Start bei jedem Text Buchstaben

Buchstabe für Buchstabe vergleichen

A	A	A	B	A	A	B	A	A	B	A	C
					A	B	A	A	B	A	C

Primitive Lösung

Start bei jedem Text Buchstaben

Buchstabe für Buchstabe vergleichen

A	A	A	B	A	A	B	A	A	B	A	C
					A	B	A	A	B	A	C

Primitive Lösung

Start bei jedem Text Buchstaben

Buchstabe für Buchstabe vergleichen

A	A	A	B	A	A	B	A	A	B	A	C
					A	B	A	A	B	A	C

Primitive Lösung

Start bei jedem Text Buchstaben

Buchstabe für Buchstabe vergleichen

A	A	A	B	A	A	B	A	A	B	A	C
					A	B	A	A	B	A	C

Primitive Lösung

Start bei jedem Text Buchstaben

Buchstabe für Buchstabe vergleichen

A	A	A	B	A	A	B	A	A	B	A	C
					A	B	A	A	B	A	C

Primitive Lösung

Start bei jedem Buchstaben

Vergleichen und weiterschieben

A	A	A	B	A	A	B	A	A	B	A	C
					A	B	A	A	B	A	C

Idee hinter Knuth-Morris-Pratt

Idee hinter KMP

Überspringen bei gleichen Pattern

A	A	A	B	A	A	B	A	A	B	A	C
		A	B	A	A	B	A	C			

Idee hinter KMP

Überspringen bei gleichen Pattern

A	A	A	B	A	A	B	A	A	B	A	C
					A	B	A	A	B	A	C

Idee hinter KMP

Überspringen bei gleichen Pattern

Tabelle für nächste Pattern Position bei Ungleichheit

A	A	A	B	A	A	B	A	A	B	A	C
					A	B	A	A	B	A	C

pattern	A	B	A	A	B	A	C
next	0	1	0	2	1	0	4

Algorithmus

Primitive Lösung

```
01 j := 1; // position in pattern
02 k := 1; // position in text
03 m := length(pattern);
04 n := length(text);
05
06 while j ≤ m and k ≤ n
07     j := 1;
08     while j ≤ m and text[k+j-1] = pattern[j]
09         j := j + 1;
10     k := k + 1;
11
12 if j > m
13     return k - m
```

Algorithmus

Primitive Lösung

```
01 j := 1; // position in pattern
02 k := 1; // position in text
03 m := length(pattern);
04 n := length(text);
05
06 while j ≤ m and k ≤ n
07   j := 1;
08   while j ≤ m and text[k+j-1] = pattern[j]
09     j := j + 1;
10   k := k + 1;
11
12 if j > m
13   return k - m
```

Knuth-Morris-Pratt

```
01 j := 1; // position in pattern
02 k := 1; // position in text
03 m := length(pattern);
04 n := length(text);
05
06 while j ≤ m and k ≤ n
07   while j > 0 and text[k] ≠ pattern[j]
08     j := next[j];
09   j := j + 1;
10   k := k + 1;
11
12 if j > m
13   return k - m
```

Algorithmus

```
01 j := 1; // position in pattern
02 k := 1; // position in text
03 m := length(pattern); // 7
04 n := length(text); // 12
05
06 while j ≤ m and k ≤ n
07   while j > 0 and text[k] ≠ pattern[j]
08     j := next[j];
09   k := k + 1;
10   j := j + 1;
11
12 if j > m
13   return k - m
```

pattern	A	B	A	A	B	A	C
next	0	1	0	2	1	0	4

j	1
k	1

A	A	A	B	A	A	B	A	A	B	A	C
A	B	A	A	B	A	C					

Algorithmus

```
01 j := 1; // position in pattern
02 k := 1; // position in text
03 m := length(pattern); // 7
04 n := length(text); // 12
05
06 while j ≤ m and k ≤ n
07   while j > 0 and text[k] ≠ pattern[j]
08     j := next[j];
09   k := k + 1;
10   j := j + 1;
11
12 if j > m
13   return k - m
```

pattern	A	B	A	A	B	A	C
next	0	1	0	2	1	0	4

j	1 -> 2
k	1 -> 2

A	A	A	B	A	A	B	A	A	B	A	C
A	B	A	A	B	A	C					

Algorithmus

```
01 j := 1; // position in pattern
02 k := 1; // position in text
03 m := length(pattern); // 7
04 n := length(text); // 12
05
06 while j ≤ m and k ≤ n
07   while j > 0 and text[k] ≠ pattern[j]
08     j := next[j];
09   k := k + 1;
10   j := j + 1;
11
12 if j > m
13   return k - m
```

pattern	A	B	A	A	B	A	C
next	0	1	0	2	1	0	4

j	2 -> 1
k	2

A	A	A	B	A	A	B	A	A	B	A	C
A	B	A	A	B	A	C					

Algorithmus

```
01 j := 1; // position in pattern
02 k := 1; // position in text
03 m := length(pattern); // 7
04 n := length(text); // 12
05
06 while j ≤ m and k ≤ n
07   while j > 0 and text[k] ≠ pattern[j]
08     j := next[j];
09   k := k + 1;
10   j := j + 1;
11
12 if j > m
13   return k - m
```

pattern	A	B	A	A	B	A	C
next	0	1	0	2	1	0	4

j	1 -> 2
k	2 -> 3

A	A	A	B	A	A	B	A	A	B	A	C
	A	B	A	A	B	A	C				

Algorithmus

```
01 j := 1; // position in pattern
02 k := 1; // position in text
03 m := length(pattern); // 7
04 n := length(text); // 12
05
06 while j ≤ m and k ≤ n
07   while j > 0 and text[k] ≠ pattern[j]
08     j := next[j];
09   k := k + 1;
10   j := j + 1;
11
12 if j > m
13   return k - m
```

pattern	A	B	A	A	B	A	C
next	0	1	0	2	1	0	4

j	2 -> 1
k	3

A	A	A	B	A	A	B	A	A	B	A	C
	A	B	A	A	B	A	C				

Algorithmus

```
01 j := 1; // position in pattern
02 k := 1; // position in text
03 m := length(pattern); // 7
04 n := length(text); // 12
05
06 while j ≤ m and k ≤ n
07   while j > 0 and text[k] ≠ pattern[j]
08     j := next[j];
09   k := k + 1;
10   j := j + 1;
11
12 if j > m
13   return k - m
```

pattern	A	B	A	A	B	A	C
next	0	1	0	2	1	0	4

j	1 -> 2
k	3 -> 4

A	A	A	B	A	A	B	A	A	B	A	C
		A	B	A	A	B	A	C			

Algorithmus

```
01 j := 1; // position in pattern
02 k := 1; // position in text
03 m := length(pattern); // 7
04 n := length(text); // 12
05
06 while j ≤ m and k ≤ n
07   while j > 0 and text[k] ≠ pattern[j]
08     j := next[j];
09   k := k + 1;
10   j := j + 1;
11
12 if j > m
13   return k - m
```

pattern	A	B	A	A	B	A	C
next	0	1	0	2	1	0	4

j	2 -> 3
k	4 -> 5

A	A	A	B	A	A	B	A	A	B	A	C
		A	B	A	A	B	A	C			

Algorithmus

```
01 j := 1; // position in pattern
02 k := 1; // position in text
03 m := length(pattern); // 7
04 n := length(text); // 12
05
06 while j ≤ m and k ≤ n
07   while j > 0 and text[k] ≠ pattern[j]
08     j := next[j];
09   k := k + 1;
10   j := j + 1;
11
12 if j > m
13   return k - m
```

pattern	A	B	A	A	B	A	C
next	0	1	0	2	1	0	4

j	3 -> 4
k	5 -> 6

A	A	A	B	A	A	B	A	A	B	A	C
		A	B	A	A	B	A	C			

Algorithmus

```
01 j := 1; // position in pattern
02 k := 1; // position in text
03 m := length(pattern); // 7
04 n := length(text); // 12
05
06 while j ≤ m and k ≤ n
07   while j > 0 and text[k] ≠ pattern[j]
08     j := next[j];
09   k := k + 1;
10   j := j + 1;
11
12 if j > m
13   return k - m
```

pattern	A	B	A	A	B	A	C
next	0	1	0	2	1	0	4

j	4 -> 5
k	6 -> 7

A	A	A	B	A	A	B	A	A	B	A	C
		A	B	A	A	B	A	C			

Algorithmus

```
01 j := 1; // position in pattern
02 k := 1; // position in text
03 m := length(pattern); // 7
04 n := length(text); // 12
05
06 while j ≤ m and k ≤ n
07   while j > 0 and text[k] ≠ pattern[j]
08     j := next[j];
09   k := k + 1;
10   j := j + 1;
11
12 if j > m
13   return k - m
```

pattern	A	B	A	A	B	A	C
next	0	1	0	2	1	0	4

j	5 -> 6
k	7 -> 8

A	A	A	B	A	A	B	A	A	B	A	C
		A	B	A	A	B	A	C			

Algorithmus

```
01 j := 1; // position in pattern
02 k := 1; // position in text
03 m := length(pattern); // 7
04 n := length(text); // 12
05
06 while j ≤ m and k ≤ n
07   while j > 0 and text[k] ≠ pattern[j]
08     j := next[j];
09   k := k + 1;
10   j := j + 1;
11
12 if j > m
13   return k - m
```

pattern	A	B	A	A	B	A	C
next	0	1	0	2	1	0	4

j	6 -> 7
k	8 -> 9

A	A	A	B	A	A	B	A	A	B	A	C
		A	B	A	A	B	A	C			

Algorithmus

```
01 j := 1; // position in pattern
02 k := 1; // position in text
03 m := length(pattern); // 7
04 n := length(text); // 12
05
06 while j ≤ m and k ≤ n
07   while j > 0 and text[k] ≠ pattern[j]
08     j := next[j];
09   k := k + 1;
10   j := j + 1;
11
12 if j > m
13   return k - m
```

pattern	A	B	A	A	B	A	C
next	0	1	0	2	1	0	4

j	7 -> 4
k	9

A	A	A	B	A	A	B	A	A	B	A	C
		A	B	A	A	B	A	C			

Algorithmus

```
01 j := 1; // position in pattern
02 k := 1; // position in text
03 m := length(pattern); // 7
04 n := length(text); // 12
05
06 while j ≤ m and k ≤ n
07   while j > 0 and text[k] ≠ pattern[j]
08     j := next[j];
09   k := k + 1;
10   j := j + 1;
11
12 if j > m
13   return k - m
```

pattern	A	B	A	A	B	A	C
next	0	1	0	2	1	0	4

j	4 -> 5
k	9 -> 10

A	A	A	B	A	A	B	A	A	B	A	C
					A	B	A	A	B	A	C

Algorithmus

```
01 j := 1; // position in pattern
02 k := 1; // position in text
03 m := length(pattern); // 7
04 n := length(text); // 12
05
06 while j ≤ m and k ≤ n
07   while j > 0 and text[k] ≠ pattern[j]
08     j := next[j];
09   k := k + 1;
10   j := j + 1;
11
12 if j > m
13   return k - m
```

pattern	A	B	A	A	B	A	C
next	0	1	0	2	1	0	4

j	5 -> 6
k	10 -> 11

A	A	A	B	A	A	B	A	A	B	A	C
					A	B	A	A	B	A	C

Algorithmus

```
01 j := 1; // position in pattern
02 k := 1; // position in text
03 m := length(pattern); // 7
04 n := length(text); // 12
05
06 while j ≤ m and k ≤ n
07   while j > 0 and text[k] ≠ pattern[j]
08     j := next[j];
09   k := k + 1;
10   j := j + 1;
11
12 if j > m
13   return k - m
```

pattern	A	B	A	A	B	A	C
next	0	1	0	2	1	0	4

j	6 -> 7
k	11 -> 12

A	A	A	B	A	A	B	A	A	B	A	C
					A	B	A	A	B	A	C

Algorithmus

```
01 j := 1; // position in pattern
02 k := 1; // position in text
03 m := length(pattern); // 7
04 n := length(text); // 12
05
06 while j ≤ m and k ≤ n
07   while j > 0 and text[k] ≠ pattern[j]
08     j := next[j];
09   k := k + 1;
10   j := j + 1;
11
12 if j > m
13   return k - m
```

pattern	A	B	A	A	B	A	C
next	0	1	0	2	1	0	4

j	7 -> 8
k	12 -> 13

A	A	A	B	A	A	B	A	A	B	A	C
					A	B	A	A	B	A	C

Algorithmus

```
01 j := 1; // position in pattern
02 k := 1; // position in text
03 m := length(pattern); // 7
04 n := length(text); // 12
05
06 while j ≤ m and k ≤ n
07   while j > 0 and text[k] ≠ pattern[j]
08     j := next[j];
09   k := k + 1;
10   j := j + 1;
11
12 if j > m
13   return k - m
```

pattern	A	B	A	A	B	A	C
next	0	1	0	2	1	0	4

j	8
k	13

A	A	A	B	A	A	B	A	A	B	A	C
					A	B	A	A	B	A	C

Vorberechnung

Ziel: Tabelle mit nächster zu testenden Position

Weg: Suche im Pattern nach Übereinstimmungen

pattern	A	B	A	A	B	A	C
next	0	1	0	2	1	0	4

Vorberechnung

```
01 pos := 1;
02 cnd := 0;
03 next[1] := 0;
04
05 while pos < length(pattern)
06   while cnd > 0 and pattern[pos] ≠ pattern[cnd]
07     cnd := next[cnd];
08   cnd := cnd + 1;
09   pos := pos + 1;
10   if pattern[pos] = pattern[cnd]
11     next[pos] := next[cnd];
12   else
13     next[pos] := cnd;
```

Vorberechnung

```
01 pos := 1;
02 cnd := 0;
03 next[1] := 0;
04
05 while pos < length(pattern)
06   while cnd > 0 and pattern[pos] ≠ pattern[cnd]
07     cnd := next[cnd];
08   cnd := cnd + 1;
09   pos := pos + 1;
10   if pattern[pos] = pattern[cnd]
11     next[pos] := next[cnd];
12   else
13     next[pos] := cnd;
```

pattern	A	B	A	A	B	A	C
next	0						

pos	1
cnd	0

Vorberechnung

```
01 pos := 1;
02 cnd := 0;
03 next[1] := 0;
04
05 while pos < length(pattern)
06   while cnd > 0 and pattern[pos] ≠ pattern[cnd]
07     cnd := next[cnd];
08   cnd := cnd + 1;
09   pos := pos + 1;
10   if pattern[pos] = pattern[cnd]
11     next[pos] := next[cnd];
12   else
13     next[pos] := cnd;
```

pattern	A	B	A	A	B	A	C
next	0						

pos	1 -> 2
cnd	0 -> 1

Vorberechnung

```
01 pos := 1;
02 cnd := 0;
03 next[1] := 0;
04
05 while pos < length(pattern)
06   while cnd > 0 and pattern[pos] ≠ pattern[cnd]
07     cnd := next[cnd];
08   cnd := cnd + 1;
09   pos := pos + 1;
10   if pattern[pos] = pattern[cnd]
11     next[pos] := next[cnd];
12   else
13     next[pos] := cnd;
```

pattern	A	B	A	A	B	A	C
next	0	1					

pos	2
cnd	1

Vorberechnung

```
01 pos := 1;
02 cnd := 0;
03 next[1] := 0;
04
05 while pos < length(pattern)
06   while cnd > 0 and pattern[pos] ≠ pattern[cnd]
07     cnd := next[cnd];
08   cnd := cnd + 1;
09   pos := pos + 1;
10   if pattern[pos] = pattern[cnd]
11     next[pos] := next[cnd];
12   else
13     next[pos] := cnd;
```

pattern	A	B	A	A	B	A	C
next	0	1					

pos	2
cnd	1 -> 0

Vorberechnung

```
01 pos := 1;
02 cnd := 0;
03 next[1] := 0;
04
05 while pos < length(pattern)
06   while cnd > 0 and pattern[pos] ≠ pattern[cnd]
07     cnd := next[cnd];
08   cnd := cnd + 1;
09   pos := pos + 1;
10   if pattern[pos] = pattern[cnd]
11     next[pos] := next[cnd];
12   else
13     next[pos] := cnd;
```

pattern	A	B	A	A	B	A	C
next	0	1					

pos	2 -> 3
cnd	0 -> 1

Vorberechnung

```
01 pos := 1;
02 cnd := 0;
03 next[1] := 0;
04
05 while pos < length(pattern)
06   while cnd > 0 and pattern[pos] ≠ pattern[cnd]
07     cnd := next[cnd];
08   cnd := cnd + 1;
09   pos := pos + 1;
10   if pattern[pos] = pattern[cnd]
11     next[pos] := next[cnd];
12   else
13     next[pos] := cnd;
```

pattern	A	B	A	A	B	A	C
next	0	1	0				

pos	3
cnd	1

Vorberechnung

```
01 pos := 1;
02 cnd := 0;
03 next[1] := 0;
04
05 while pos < length(pattern)
06   while cnd > 0 and pattern[pos] ≠ pattern[cnd]
07     cnd := next[cnd];
08   cnd := cnd + 1;
09   pos := pos + 1;
10   if pattern[pos] = pattern[cnd]
11     next[pos] := next[cnd];
12   else
13     next[pos] := cnd;
```

pattern	A	B	A	A	B	A	C
next	0	1	0				

pos	3 -> 4
cnd	1 -> 2

Vorberechnung

```
01 pos := 1;
02 cnd := 0;
03 next[1] := 0;
04
05 while pos < length(pattern)
06   while cnd > 0 and pattern[pos] ≠ pattern[cnd]
07     cnd := next[cnd];
08   cnd := cnd + 1;
09   pos := pos + 1;
10   if pattern[pos] = pattern[cnd]
11     next[pos] := next[cnd];
12   else
13     next[pos] := cnd;
```

pattern	A	B	A	A	B	A	C
next	0	1	0	2			

pos	4
cnd	2

Vorberechnung

```
01 pos := 1;
02 cnd := 0;
03 next[1] := 0;
04
05 while pos < length(pattern)
06   while cnd > 0 and pattern[pos] ≠ pattern[cnd]
07     cnd := next[cnd];
08   cnd := cnd + 1;
09   pos := pos + 1;
10   if pattern[pos] = pattern[cnd]
11     next[pos] := next[cnd];
12   else
13     next[pos] := cnd;
```

pattern	A	B	A	A	B	A	C
next	0	1	0	2			

pos	4
cnd	2 -> 1

Vorberechnung

```
01 pos := 1;
02 cnd := 0;
03 next[1] := 0;
04
05 while pos < length(pattern)
06   while cnd > 0 and pattern[pos] ≠ pattern[cnd]
07     cnd := next[cnd];
08   cnd := cnd + 1;
09   pos := pos + 1;
10   if pattern[pos] = pattern[cnd]
11     next[pos] := next[cnd];
12   else
13     next[pos] := cnd;
```

pattern	A	B	A	A	B	A	C
next	0	1	0	2			

pos	4 -> 5
cnd	1 -> 2

Vorberechnung

```
01 pos := 1;
02 cnd := 0;
03 next[1] := 0;
04
05 while pos < length(pattern)
06   while cnd > 0 and pattern[pos] ≠ pattern[cnd]
07     cnd := next[cnd];
08   cnd := cnd + 1;
09   pos := pos + 1;
10   if pattern[pos] = pattern[cnd]
11     next[pos] := next[cnd];
12   else
13     next[pos] := cnd;
```

pattern	A	B	A	A	B	A	C
next	0	1	0	2	1		

pos	5
cnd	2

Vorberechnung

```
01 pos := 1;
02 cnd := 0;
03 next[1] := 0;
04
05 while pos < length(pattern)
06   while cnd > 0 and pattern[pos] ≠ pattern[cnd]
07     cnd := next[cnd];
08   cnd := cnd + 1;
09   pos := pos + 1;
10   if pattern[pos] = pattern[cnd]
11     next[pos] := next[cnd];
12   else
13     next[pos] := cnd;
```

pattern	A	B	A	A	B	A	C
next	0	1	0	2	1		

pos	5 -> 6
cnd	2 -> 3

Vorberechnung

```
01 pos := 1;
02 cnd := 0;
03 next[1] := 0;
04
05 while pos < length(pattern)
06   while cnd > 0 and pattern[pos] ≠ pattern[cnd]
07     cnd := next[cnd];
08   cnd := cnd + 1;
09   pos := pos + 1;
10   if pattern[pos] = pattern[cnd]
11     next[pos] := next[cnd];
12   else
13     next[pos] := cnd;
```

pattern	A	B	A	A	B	A	C
next	0	1	0	2	1	0	

pos	6
cnd	3

Vorberechnung

```
01 pos := 1;
02 cnd := 0;
03 next[1] := 0;
04
05 while pos < length(pattern)
06   while cnd > 0 and pattern[pos] ≠ pattern[cnd]
07     cnd := next[cnd];
08   cnd := cnd + 1;
09   pos := pos + 1;
10   if pattern[pos] = pattern[cnd]
11     next[pos] := next[cnd];
12   else
13     next[pos] := cnd;
```

pattern	A	B	A	A	B	A	C
next	0	1	0	2	1	0	

pos	6 -> 7
cnd	3 -> 4

Vorberechnung

```
01 pos := 1;
02 cnd := 0;
03 next[1] := 0;
04
05 while pos < length(pattern)
06   while cnd > 0 and pattern[pos] ≠ pattern[cnd]
07     cnd := next[cnd];
08   cnd := cnd + 1;
09   pos := pos + 1;
10   if pattern[pos] = pattern[cnd]
11     next[pos] := next[cnd];
12   else
13     next[pos] := cnd;
```

pattern	A	B	A	A	B	A	C
next	0	1	0	2	1	0	4

pos	7
cnd	4

Vorberechnung

```
01 pos := 1;
02 cnd := 0;
03 next[1] := 0;
04
05 while pos < length(pattern)
06   while cnd > 0 and pattern[pos] ≠ pattern[cnd]
07     cnd := next[cnd];
08   cnd := cnd + 1;
09   pos := pos + 1;
10   if pattern[pos] = pattern[cnd]
11     next[pos] := next[cnd];
12   else
13     next[pos] := cnd;
```

pattern	A	B	A	A	B	A	C
next	0	1	0	2	1	0	4

pos	7
cnd	4

Komplexität

Vorberechnung: O(Patternlänge)

Suche: O(Textlänge)

Gesamt: O(Patternlänge + Textlänge)

Komplexität - Vergleich

	Vorberechnung	Suche
Naive Suche	keine	$\Theta(nm)$
Knuth-Morris-Pratt	$\Theta(m)$	$\Theta(n)$
Boyer-Moore	$\Theta(m+k)$	Bester Fall: $\Omega(n/m)$ Schlechtester Fall: $O(nm)$

Länge des Patterns: m
Länge des Textes: n
Elemente im Alphabet: k

Demo

Vorteile

Lineare Laufzeit

-> Real-Time Variante vorhanden

Kein zurückgehen im Text

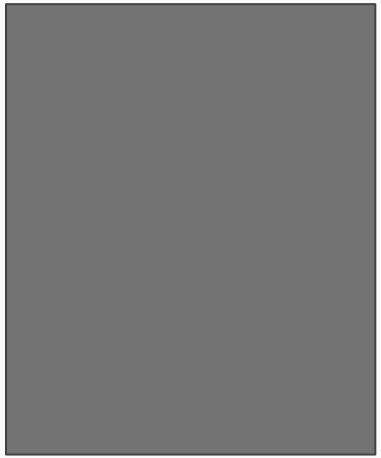
Nachteile

Vorberechnung

Geschichte

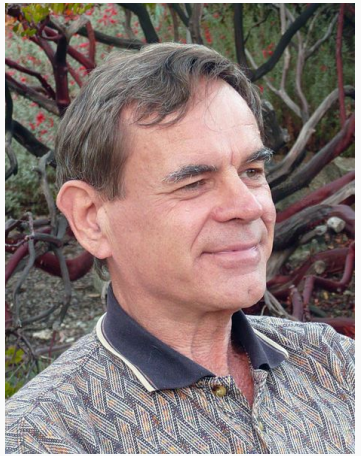
By Flickr user Jacob Appelbaum, uploaded to en.wikipedia by users BeSherman, Duozmo (Flickr.com (via en.wikipedia)) CC BY-SA 2.5

Donald Knuth



https://web.stanford.edu/class/ee380/Abstracts/090422-jhmorris.jpg

James H. Morris



By Vaughan Pratt (Photograph taken and owned by Vaughan Pratt) CC BY-SA 3.0

Vaughan Pratt

Geschichte

Knuth und Pratt in 1970

- auf Basis von Cook’s Theorem

Morris in 1969

- unverständlich

Quellen / Nachweise

Knuth, Donald E., James H. Morris, Jr, and Vaughan R. Pratt. "Fast pattern matching in strings." SIAM journal on computing 6.2 (1977): 323-350.

Singla, Nimisha, and Deepak Garg. "String matching algorithms and their applicability in various applications." International journal of soft computing and engineering 1.6 (2012): 218-222.

“Donald Knuth - The Knuth-Morris-Pratt Algorithm” WEB of STORIES, https://www.webofstories.com/play/donald.knuth/92.