String Algorithm

Pelatnas 1 TOKI 2019

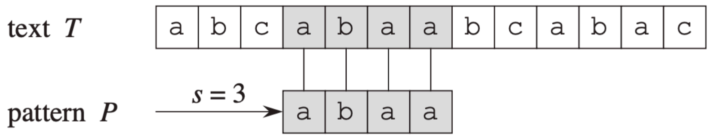
Kamis, 11 Oktober 2018 | Auzi Asfarian | Leonardi Fabianto

Yang Kita Ulas

- String matching
- Knuth-Morris-Prath algorithm
- Longest common subsequence
- Longest increasing subsequence

String Matching

- Diberikan sebuah string $T[1 \ldots n]$ dan sebuah string $P[1 \ldots m]$, carilah kemunculan P di dalam T.
- Asumsi:
    - T dan P terdiri atas alfabet/karakter yang sama.
    - P terdapat dalam T dengan pergeseran sebanyak s jika $P[1 \ldots m] = T[s+1 \ldots s+m]$.
    - $1 \leq m \leq n$
- Keluaran:
    - Seluruh nilai s yang memenuhi.
- Trivia:
    - P biasa disebut sebagai *needle* dan T disebut *haystack*.



Contoh

- *Input*:
  - T: akuakudiadiakamukamuaku
  - P: aku
- *Output*:
  - 1, 4, 21

Naïve Solution

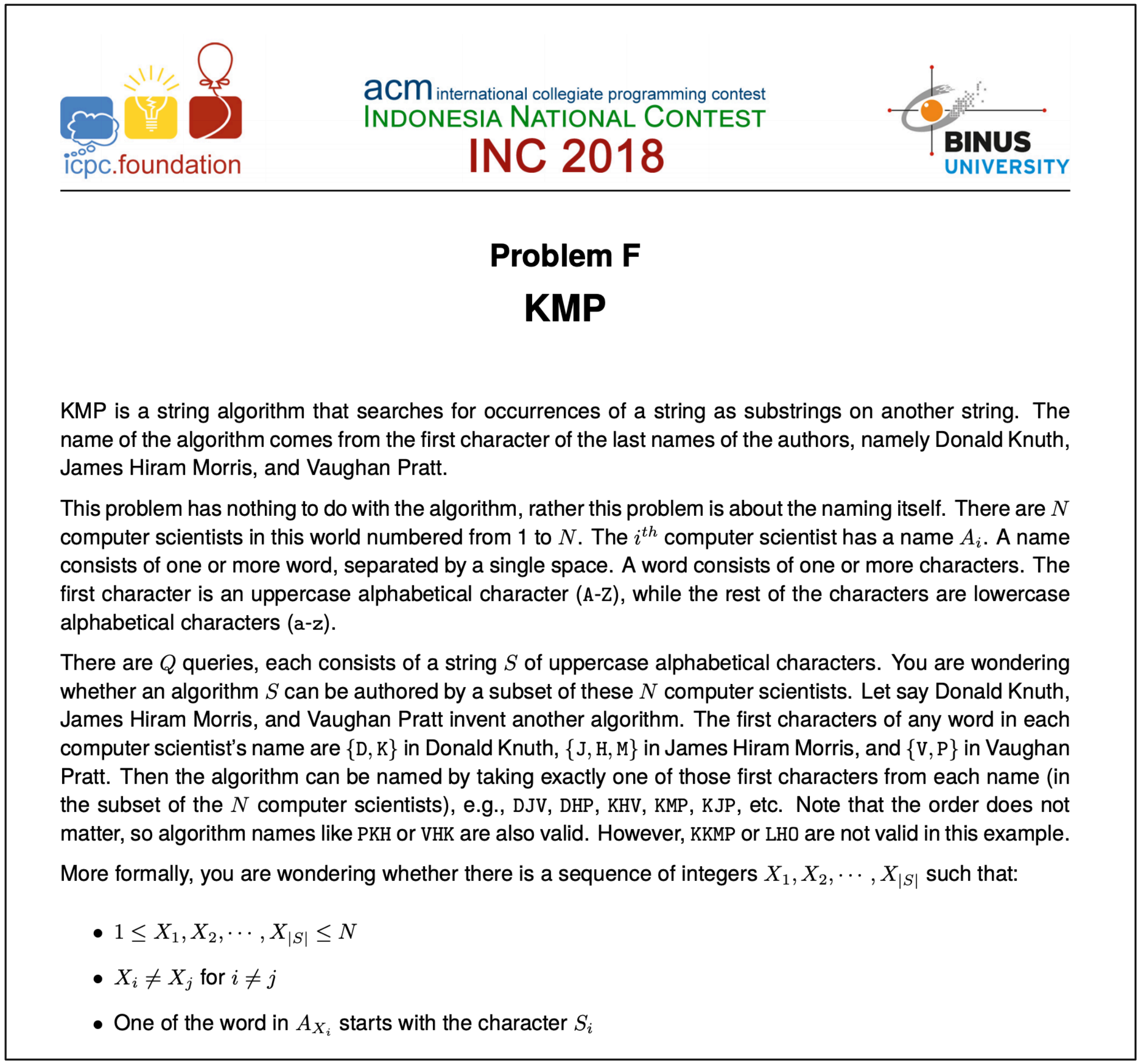
```
function brute_force(text[], pattern[])
{
    // n = size of text
    // m = size of pattern
    for(i = 0; i < n; i++)
    {
        // loop until mismatch found
        for(j = 0; j < m && i + j < n; j++)
            if(text[i + j] != pattern[j]) break;
        if(j == m)
            // match found at index i
  }
}
```

Komplesitasnya? Kapan best case? Kapan worst case?

String Matching

- Kompleksitas:
    - O(*mn*)
- Worst case:
    - T: AAAAAAAAAAAA
    - P: AA
- Best case:
    - T: AAAAAAAAAAAA
    - P: BAAAAA

Knuth-Morris-Prath (KMP)



acm international collegiate programming contest
INDONESIA NATIONAL CONTEST
INC 2018

icpc.foundation

BINUS UNIVERSITY

Problem F

KMP

KMP is a string algorithm that searches for occurrences of a string as substrings on another string. The name of the algorithm comes from the first character of the last names of the authors, namely Donald Knuth, James Hiram Morris, and Vaughan Pratt.

This problem has nothing to do with the algorithm, rather this problem is about the naming itself. There are N computer scientists in this world numbered from 1 to N. The i^{th} computer scientist has a name A_i. A name consists of one or more word, separated by a single space. A word consists of one or more characters. The first character is an uppercase alphabetical character (`A-Z`), while the rest of the characters are lowercase alphabetical characters (`a-z`).

There are Q queries, each consists of a string S of uppercase alphabetical characters. You are wondering whether an algorithm S can be authored by a subset of these N computer scientists. Let say Donald Knuth, James Hiram Morris, and Vaughan Pratt invent another algorithm. The first characters of any word in each computer scientist's name are $\{\texttt{D}, \texttt{K}\}$ in Donald Knuth, $\{\texttt{J}, \texttt{H}, \texttt{M}\}$ in James Hiram Morris, and $\{\texttt{V}, \texttt{P}\}$ in Vaughan Pratt. Then the algorithm can be named by taking exactly one of those first characters from each name (in the subset of the N computer scientists), e.g., `DJV`, `DHP`, `KHV`, `KMP`, `KJP`, etc. Note that the order does not matter, so algorithm names like `PKH` or `VHK` are also valid. However, `KKMP` or `LHO` are not valid in this example.

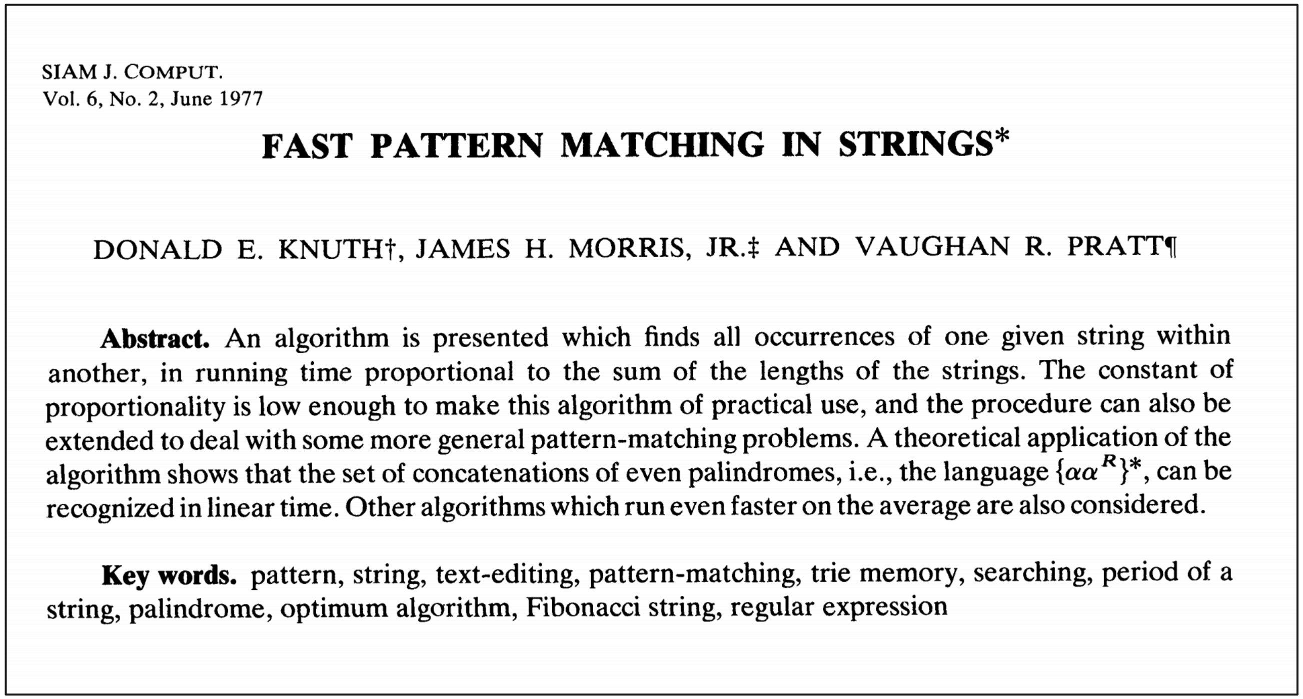
More formally, you are wondering whether there is a sequence of integers $X_1, X_2, \cdots, X_{|S|}$ such that:

- $1 \leq X_1, X_2, \cdots, X_{|S|} \leq N$
- $X_i \neq X_j$ for $i \neq j$
- One of the word in A_{X_i} starts with the character S_i

Spoiler:
Ini bukan soal string matching.

Knuth-Morris-Prath (KMP)

- Dibuat oleh Donald E Knuth dan Vaughan Pratt serta James Morris di tahun 1977.



SIAM J. COMPUT.
Vol. 6, No. 2, June 1977

FAST PATTERN MATCHING IN STRINGS*

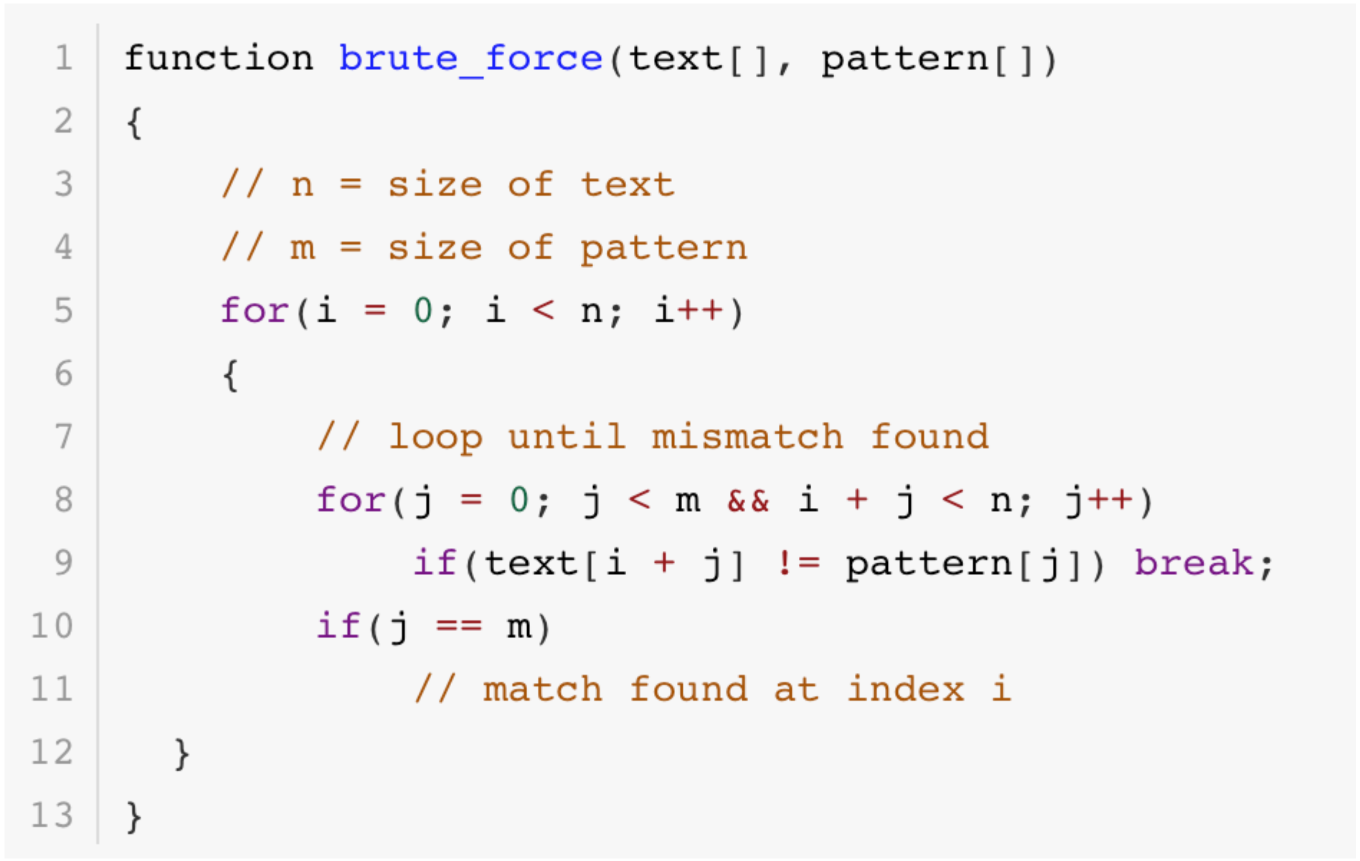
DONALD E. KNUTH†, JAMES H. MORRIS, JR.‡ AND VAUGHAN R. PRATT¶

Abstract. An algorithm is presented which finds all occurrences of one given string within another, in running time proportional to the sum of the lengths of the strings. The constant of proportionality is low enough to make this algorithm of practical use, and the procedure can also be extended to deal with some more general pattern-matching problems. A theoretical application of the algorithm shows that the set of concatenations of even palindromes, i.e., the language $\{\alpha\alpha^R\}^*$, can be recognized in linear time. Other algorithms which run even faster on the average are also considered.

Key words. pattern, string, text-editing, pattern-matching, trie memory, searching, period of a string, palindrome, optimum algorithm, Fibonacci string, regular expression

http://www.cin.br/~paguso/courses/if767/bib/KMP_1977.pdf

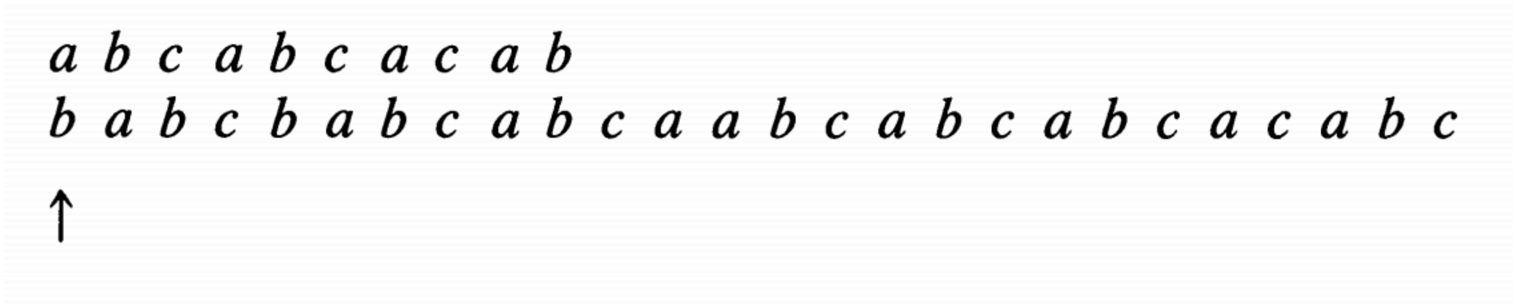
Mengapa Cara Naïve Lambat?



```
function brute_force(text[], pattern[])
{
    // n = size of text
    // m = size of pattern
    for(i = 0; i < n; i++)
    {
        // loop until mismatch found
        for(j = 0; j < m && i + j < n; j++)
            if(text[i + j] != pattern[j]) break;
        if(j == m)
            // match found at index i
  }
}
```

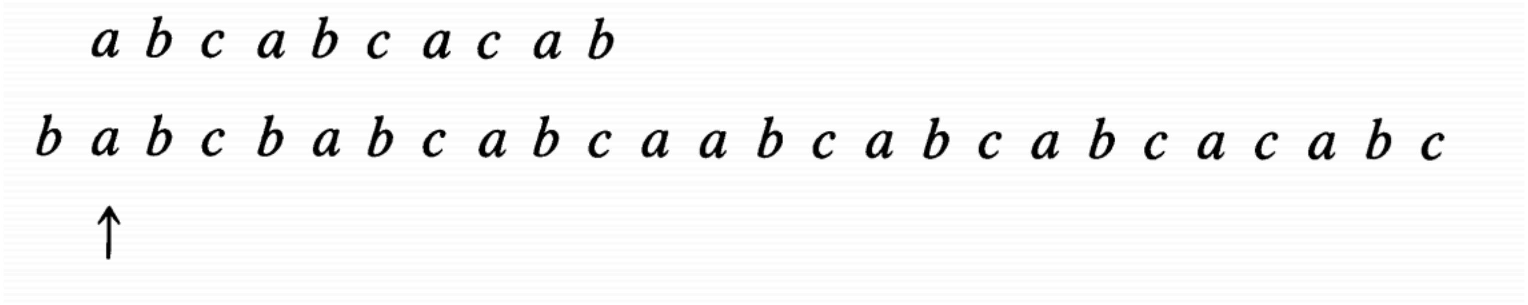
Mengapa Cara Naïve Lambat?

- Proses ‘sliding’ selalu dilakukan 1 indeks.
- Ketika ditemukan *mismatch*:



```
a b c a b c a c a b
b a b c b a b c a b c a a b c a b c a b c a c a b c
↑
```

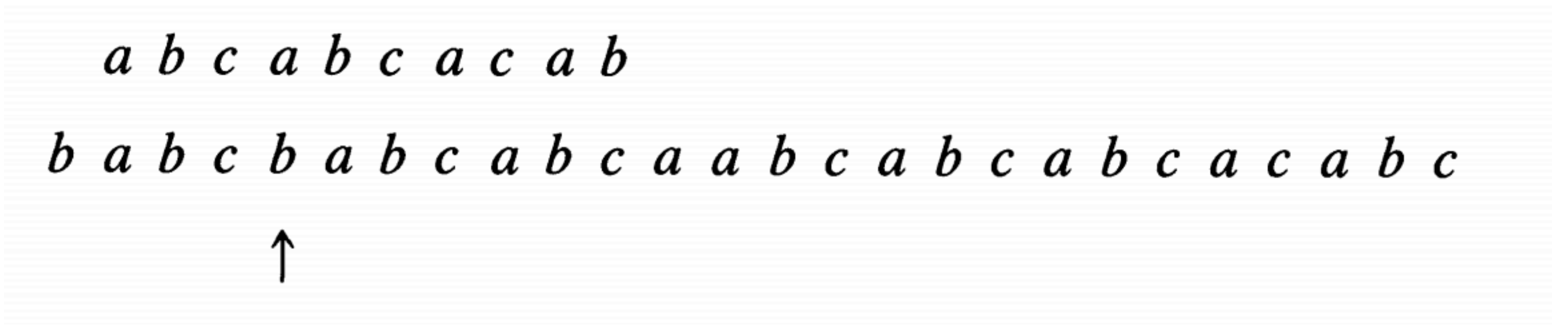
- Kita ‘geser’ 1 ke kanan:



```
  a b c a b c a c a b
b a b c b a b c a b c a a b c a b c a b c a c a b c
  ↑
```

Bagaimana Kalau Terjadi *Partial Match*?

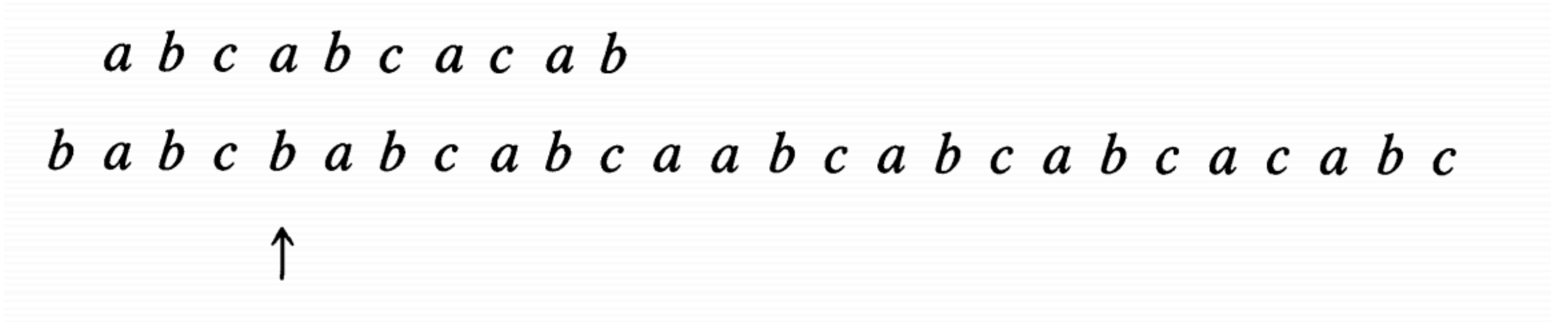
- Misal, kita sudah mencocokkan *T*[2…4] dengan *P*[1…3]. *Mismatch* di *T*[5] dan *P*[4].



- Kalau pada cara *naïve*, kita tetap geser 1 ke kanan. Dan pengecekan kembali ke posisi P[0] dengan T posisi berikutnya.
- Adakah hal yang bisa kita manfaatkan dari *partial match* ini agar lebih efisien?

Ilustrasi

- *Mismatch* di *T*[5] dan *P*[4].

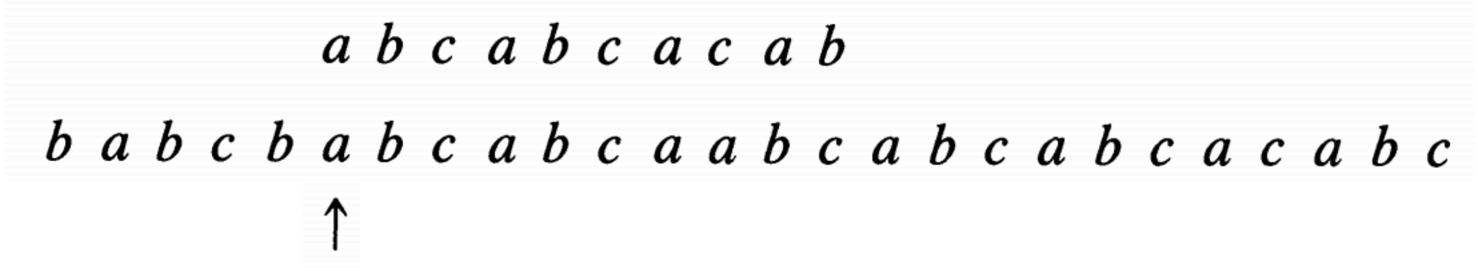


```
  a b c a b c a c a b
b a b c b a b c a b c a a b c a b c a b c a c a b c
        ↑
```

- Geser berapa kita ke kanan?

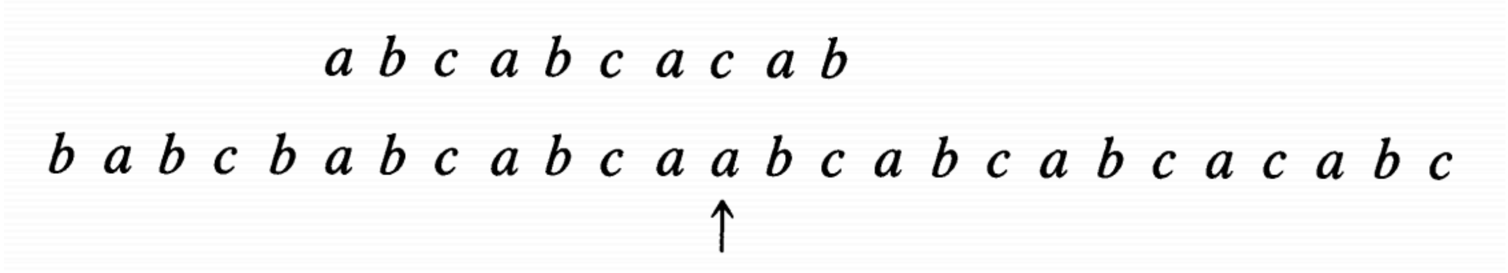
Ilustrasi

- Kita bisa geser 4 ke kanan:



```
          a b c a b c a c a b
b a b c b a b c a b c a a b c a b c a b c a c a b c
          ↑
```

- Lanjutkan pengecekan, dan bertemu *mismatch* lagi:

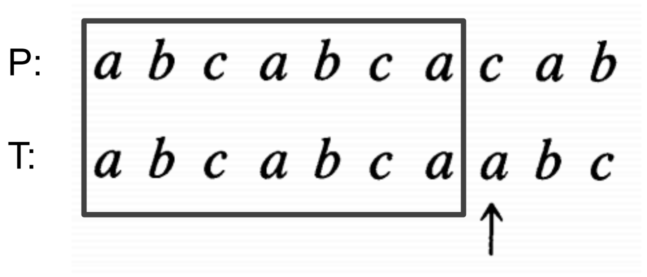


```
          a b c a b c a c a b
b a b c b a b c a b c a a b c a b c a b c a c a b c
                        ↑
```

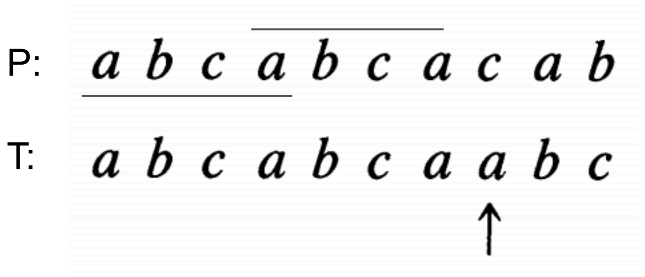
- Sekarang geser berapa ke kanan?

Observasi

- T dan P yang dicek hingga *mismatch* terjadi pasti sama:



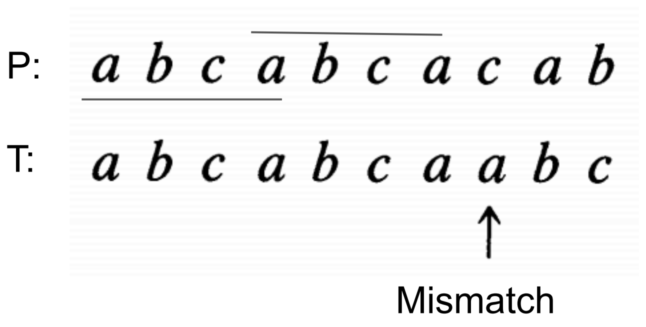
- Ada substring yang berulang:

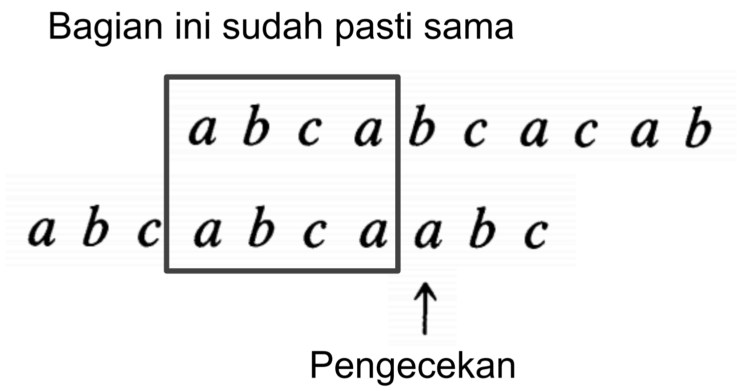


- Jadi kita geser berapa ke kanan?

Speedup di KMP

- Kita geser 3.





- Dan kita lanjutkan **pengecekan mulai dari lokasi tempat *mismatch* terjadi**.
- Pada KMP, terdapat langkah untuk membuat tabel pergeseran yang dilakukan jika terjadi mismatch di P_i.

KMP

- KMP terbagi jadi dua langkah:
  - Prefix Function (O(m))
  - String Matching (O(n))
- Kompleksitasnya O(m + n), linear!

Prefix

```
COMPUTE-PREFIX-FUNCTION$(P)$
$m = P.length$
let $\pi[1..m]$ be a new array
$\pi[1] = 0$
$k = 0$
for $q = 2$ to $m$
    while $k > 0$ and $P[k+1] \neq P[q]$
        $k = \pi[k]$
    if $P[k+1] == P[q]$
        $k = k + 1$
    $\pi[q] = k$
return $\pi$
```

- Apa hasilnya jika diberikan P = abcaby?

Jawaban

a	b	c	a	b	y
0	0	0	1	2	0

String Matching

- Pada saat matching, pergeseran disesuaikan dengan tabel prefix yang sudah dihitung sebelumnya.

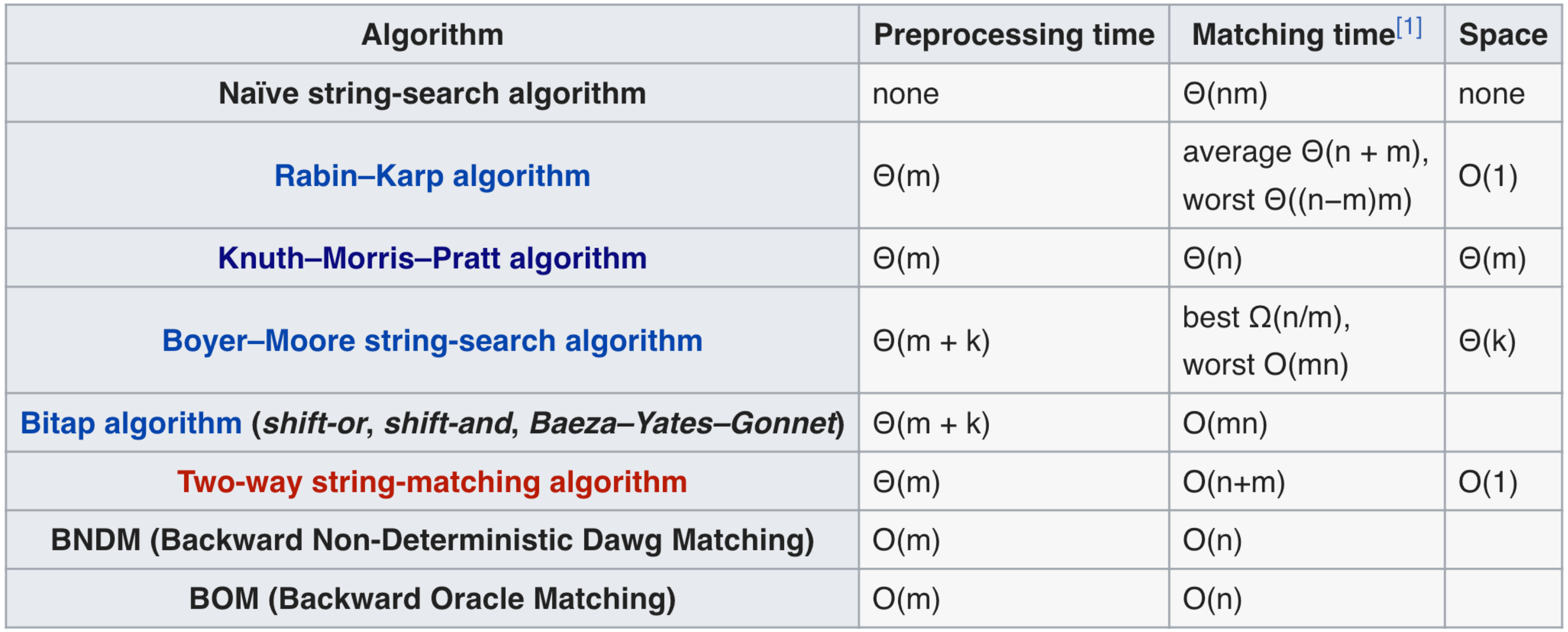
```
KMP-MATCHER($T$, $P$)
$n = T.length$
$m = P.length$
$\pi$ = COMPUTE-PREFIX-FUNCTION($P$)
$q = 0$                                   // number of characters matched
for $i = 1$ to $n$                        // scan the text from left to right
    while $q > 0$ and $P[q+1] \neq T[i]$
        $q = \pi[q]$                      // next character does not match
    if $P[q+1] == T[i]$
        $q = q + 1$                       // next character matches
    if $q == m$                           // is all of $P$ matched?
        print "Pattern occurs with shift" $i - m$
        $q = \pi[q]$                      // look for the next match
```

Contoh

- Lakukan *string matching* dengan:
- T: a b x a b c a b c a b y
- P: a b c a b y
- Prefix Table:

a	b	c	a	b	y
0	0	0	1	2	0

Algoritme String Matching Lainnya

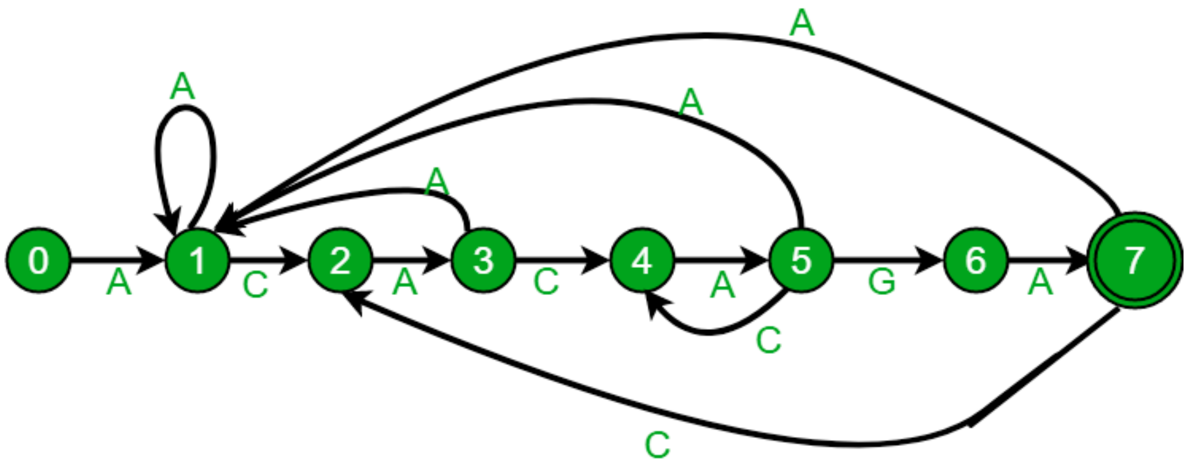


Algorithm	Preprocessing time	Matching time[1]	Space
Naïve string-search algorithm	none	Θ(nm)	none
Rabin–Karp algorithm	Θ(m)	average Θ(n + m), worst Θ((n−m)m)	O(1)
Knuth–Morris–Pratt algorithm	Θ(m)	Θ(n)	Θ(m)
Boyer–Moore string-search algorithm	Θ(m + k)	best Ω(n/m), worst O(mn)	Θ(k)
Bitap algorithm (*shift-or*, *shift-and*, *Baeza–Yates–Gonnet*)	Θ(m + k)	O(mn)	
Two-way string-matching algorithm	Θ(m)	O(n+m)	O(1)
BNDM (Backward Non-Deterministic Dawg Matching)	O(m)	O(n)	
BOM (Backward Oracle Matching)	O(m)	O(n)	

https://en.wikipedia.org/wiki/String-searching_algorithm

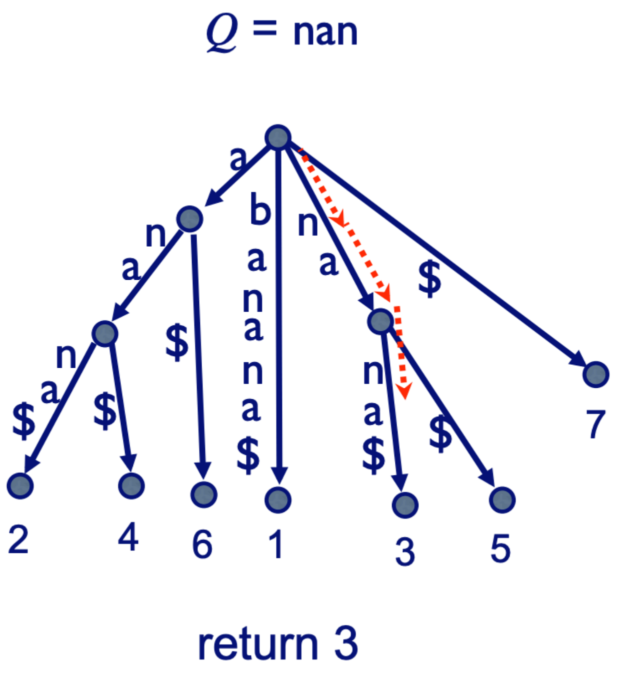
Lihat Juga:

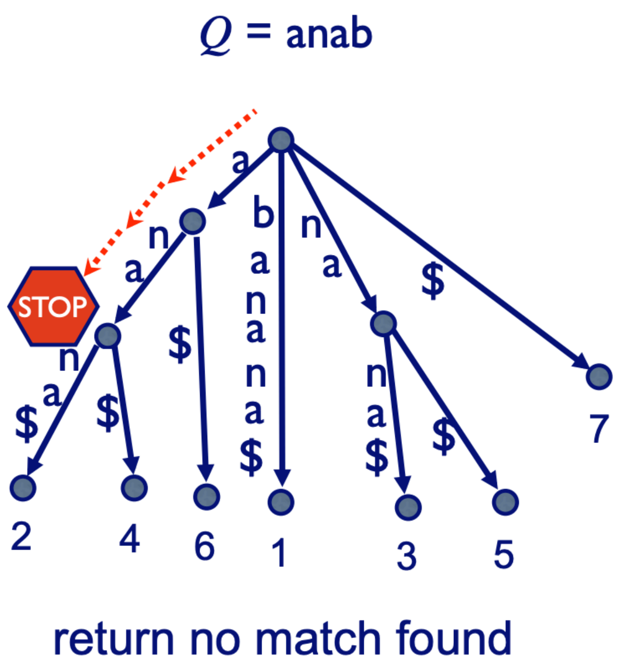
- *Finite automata*



Lihat Juga:

- *Suffix Tree*





https://www.biostat.wisc.edu/bmi776/lectures/suffix-trees.pdf

Lihat Juga

- Boyer-Moore algorithm