PATTERN MATCHING

Ir. Rismayani, S.Kom., M.T

Algoritma Pattern Matching

- Pattern matching adalah sebuah proses pencarian pola dalam sekumpulan token yang berurutan.
- Pada umumnya, pattern matching digunakan untuk mencari pola karakter alfabet dalam sebuah kalimat (string).
- Persoalan pattern matching dirumuskan sebagai berikut: Diberikan sebuah teks (text), yakni string yang memiliki panjang n karakter; dan sebuah pola (pattern), yakni string dengan panjang m karakter ($m < n$) yang akan dicari di dalam text.

Ilustrasi

- Text: With gr**eat** power, comes great bills.
- Pattern: eat

- Pattern matching akan mencari kemunculan pertama (dari kiri) pattern pada text. Dalam kasus ini, ditemukan pattern ‘eat’ pada karakter ke-8 text (spasi termasuk sebuah karakter/token).

Pendekatan

- Brute force,
- Boyer-Moore
- Karp-Rabin
- Knuth-MorrisPratt (KMP) Algorithm

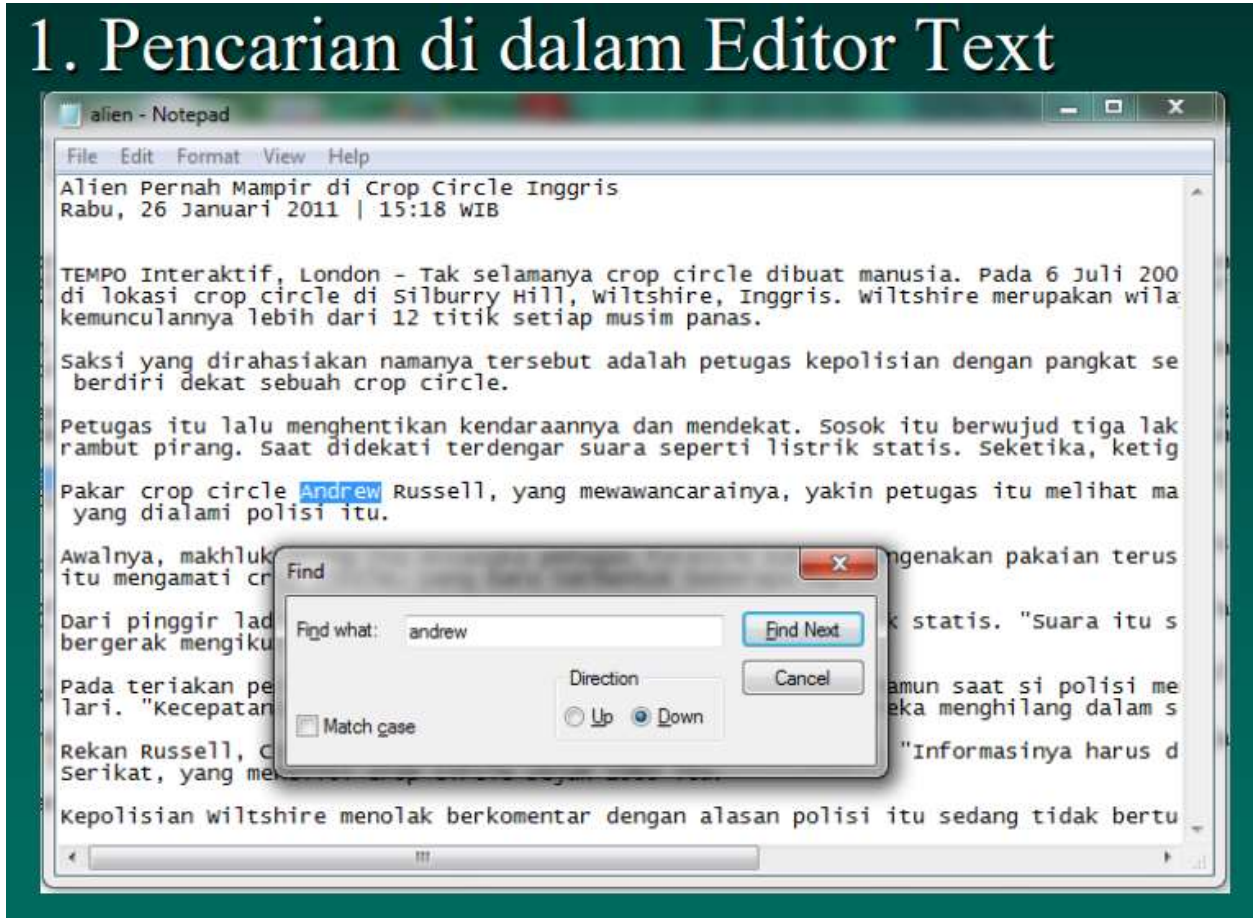
LANJUTAN

1. *What is Pattern Matching?*

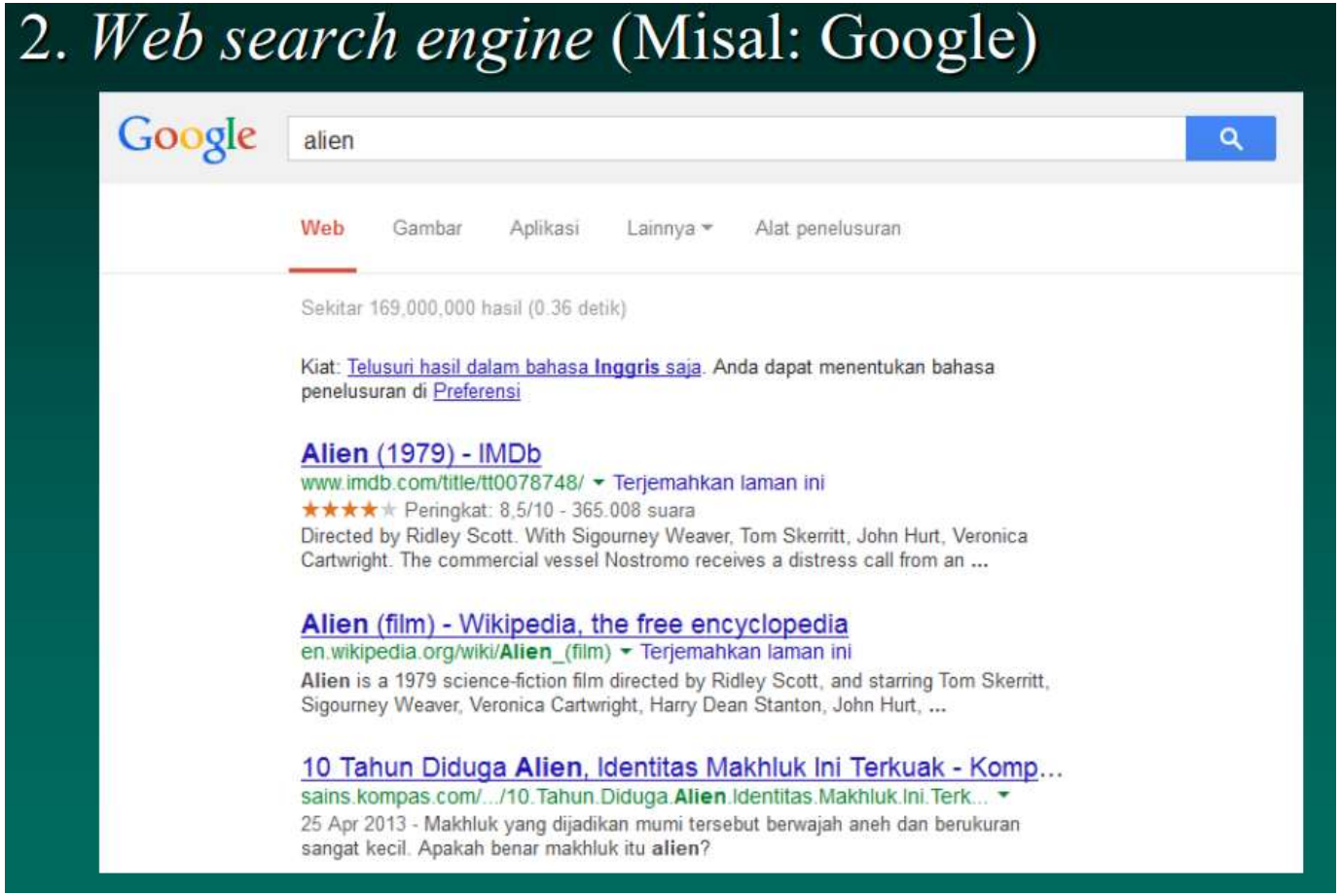
- Definisi: Diberikan:
  1. *T*: teks (*text*), yaitu (*long*) *string* yang panjangnya *n* karakter
  2. *P: pattern*, yaitu *string* dengan panjang *m* karakter (asumsi $m <<< n$) yang akan dicari di dalam teks.

  Carilah (*find* atau *locate*) lokasi pertama di dalam teks yang bersesuaian dengan *pattern*.
- Contoh:
  - T: "the rain in spain stays mainly on the plain"
  - P: "main"

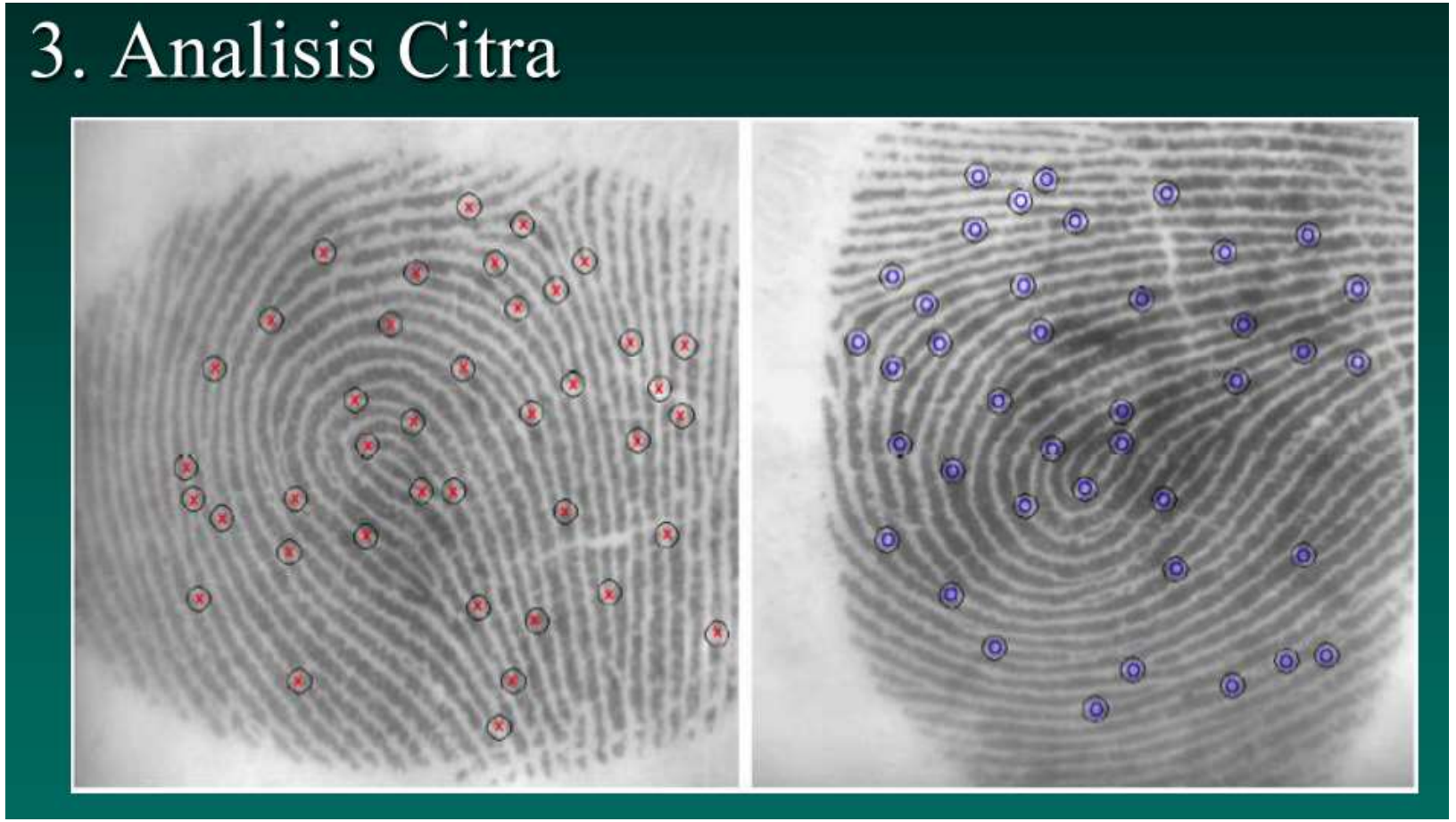
Aplikasi:


1. Pencarian di dalam Editor Text
alien - Notepad
File Edit Format View Help
Alien Pernah Mampir di Crop Circle Inggris
Rabu, 26 Januari 2011 | 15:18 WIB
TEMPO Interaktif, London - Tak selamanya crop circle dibuat manusia. Pada 6 Juli 200
di lokasi crop circle di Silburry Hill, Wiltshire, Inggris. Wiltshire merupakan wila
kemunculannya lebih dari 12 titik setiap musim panas.
Saksi yang dirahasiakan namanya tersebut adalah petugas kepolisian dengan pangkat se
berdiri dekat sebuah crop circle.
Petugas itu lalu menghentikan kendaraannya dan mendekat. Sosok itu berwujud tiga lak
rambut pirang. Saat didekati terdengar suara seperti listrik statis. Seketika, ketig
Pakar crop circle Andrew Russell, yang mewawancarainya, yakin petugas itu melihat ma
yang dialami polisi itu.
Awalnya, makhluk
itu mengamati cr
ngenakan pakaian terus
Dari pinggir lad
bergerak mengiku
k statis. "Suara itu s
Pada teriakan pe
lari. "Kecepatan
amun saat si polisi me
eka menghilang dalam s
Rekan Russell, C
Serikat, yang me
"Informasinya harus d
Kepolisian Wiltshire menolak berkomentar dengan alasan polisi itu sedang tidak bertu
Find
Find what:
andrew
Find Next
Cancel
Direction
Up
Down
Match case

LANJUTAN


2. Web search engine (Misal: Google)
Google
alien
Web
Gambar
Aplikasi
Lainnya
Alat penelusuran
Sekitar 169,000,000 hasil (0.36 detik)
Kiat: Telusuri hasil dalam bahasa Inggris saja. Anda dapat menentukan bahasa penelusuran di Preferensi
Alien (1979) - IMDb
www.imdb.com/title/tt0078748/ Terjemahkan laman ini
Peringkat: 8,5/10 - 365.008 suara
Directed by Ridley Scott. With Sigourney Weaver, Tom Skerritt, John Hurt, Veronica Cartwright. The commercial vessel Nostromo receives a distress call from an ...
Alien (film) - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/Alien_(film) Terjemahkan laman ini
Alien is a 1979 science-fiction film directed by Ridley Scott, and starring Tom Skerritt, Sigourney Weaver, Veronica Cartwright, Harry Dean Stanton, John Hurt, ...
10 Tahun Diduga Alien, Identitas Makhluk Ini Terkuak - Komp...
sains.kompas.com/.../10.Tahun.Diduga.Alien.Identitas.Makhluk.Ini.Terk...
25 Apr 2013 - Makhluk yang dijadikan mumi tersebut berwajah aneh dan berukuran sangat kecil. Apakah benar makhluk itu alien?

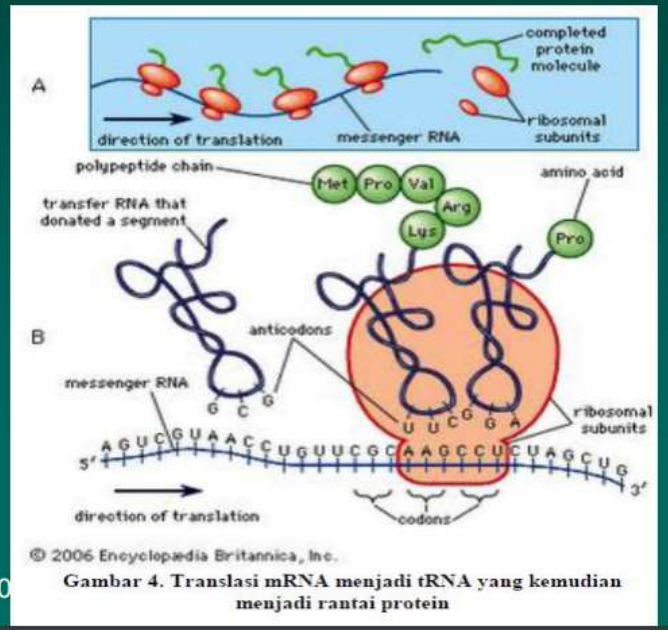
LANJUTAN



LANJUTAN

4. *Bionformatics*

- Pencocokan Rantai Asam Amino pada Rantai DNA



Gambar 4. Translasi mRNA menjadi tRNA yang kemudian menjadi rantai protein

Konsep String

- Assume S is a string of size m.

$$S = x_0 x_1 \ldots x_{m-1}$$

- A *prefix* of S is a substring S[0 .. k]
- A *suffix* of S is a substring S[k .. $m - 1$]
  - k is any index between 0 and $m - 1$

LANJUTAN

Examples

S

a	n	d	r	e	w
0					5

- All possible prefixes of S:
  - “a", "an", "and", "andr”, "andre“, "andrew“
- All possible suffixes of S:
  - “w", “ew", “rew", “drew", “ndrew” , "andrew“

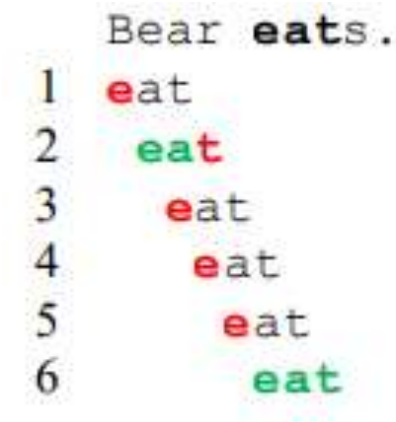
Algoritma Brute Force

- Pada pendekatan ini, setiap karakter pada pattern dicocokkan dengan karakter pada text secara straightforward (lempang).
- Jika diketahui text T[1..n] dan pattern P[1..m] adalah array of character dengan panjang masing-masing n dan m karakter, maka langkah penyelesaian dengan algoritma brute force adalah:
- 1. Sejajarkan pattern dan text pada awal karakter (P[i] dan T[j]) untuk dibandingkan, dimana i = j = 1.
- 2. Bergerak dari kiri ke kanan, bandingkan setiap karakter pada pattern dengan karakter pada text. Perbandingan dilakukan dengan membandingkan P[i] dan T[j], kemudian P[i+1] dan T[j+1], dst. sampai ditemukan:

LANJUTAN

- a. semua karakter pada pattern ditemukan (pencarian selesai), atau
- b. terdapat mismatch (ketidakcocokan) karakter

- 3. Jika terdapat mismatch dan text belum habis, ulangi langkah 2 dengan j = j+1 (menggeser pattern sebanyak 1 karakter)
- Berikut ilustrasi algoritma brute force:
- Text: Bear eats.
- Pattern: eat
- Bear eats.



LANJUTAN

- Pattern ‘eat’ ditemukan pada karakter ke-6 text dengan 10 perbandingan karakter.
- Jika menghitung jumlah operasi perbandingan yang dilakukan, kompleksitas waktu terbaik algoritma brute force adalah O(n).
- Pada kasus terburuk, dilakukan sebanyak m(nm+1) perbandingan karakter.

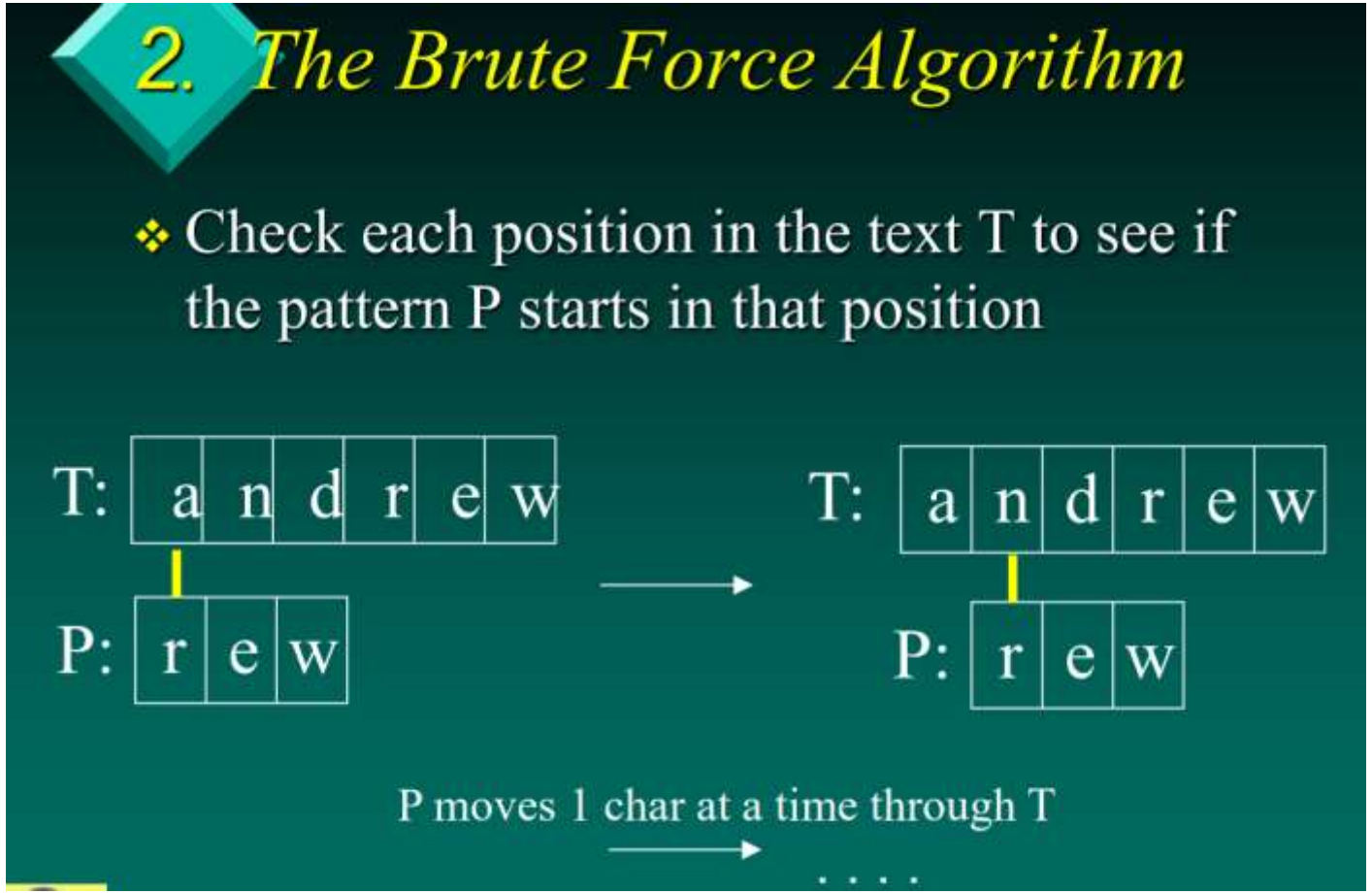
LANJUTAN

- Brute force : pendekatan yang lempang (straightforward) untuk memecahkan suatu masalah. Biasanya didasarkan pada:
  - pernyataan masalah (problem statement)
  - definisi konsep yang dilibatkan.
- Algoritma brute force memecahkan masalah dengan sangat sederhana, langsung, jelas (obvious way).
- Algoritma brute force merupakan algoritma pencocokan string yang ditulis tanpa memikirkan peningkatan performa. Algoritma ini sangat jarang dipakai dalam praktik, namun berguna dalam studi pembanding dan studi-studi lainnya.

CARA KERJA

1. Secara sistematis, langkah-langkah yang dilakukan algoritma brute force pada saat mencocokkan string adalah:
   Algoritma brute force mulai mencocokkan pattern pada awal teks.
2. Dari kiri ke kanan, algoritma ini akan mencocokkan karakter per karakter pattern dengan karakter di teks yang bersesuaian, sampai salah satu kondisi berikut dipenuhi:
   1. Karakter di pattern dan di teks yang dibandingkan tidak cocok (mismatch).
   2. Semua karakter di pattern cocok. Kemudian algoritma akan memberitahukan penemuan di posisi ini.
3. Algoritma kemudian terus menggeser pattern sebesar satu ke kanan, dan mengulangi langkah ke-2 sampai pattern berada di ujung teks.

LANJUTAN


2. The Brute Force Algorithm
Check each position in the text T to see if the pattern P starts in that position
T: a n d r e w
P: r e w
T: a n d r e w
P: r e w
P moves 1 char at a time through T
. . . .

LANJUTAN

Pattern: NOT

Teks: NOBODY NOTICED HIM

```
  NOBODY NOTICED HIM
1 NOT
2  NOT
3   NOT
4    NOT
5     NOT
6      NOT
7       NOT
8        NOT
```

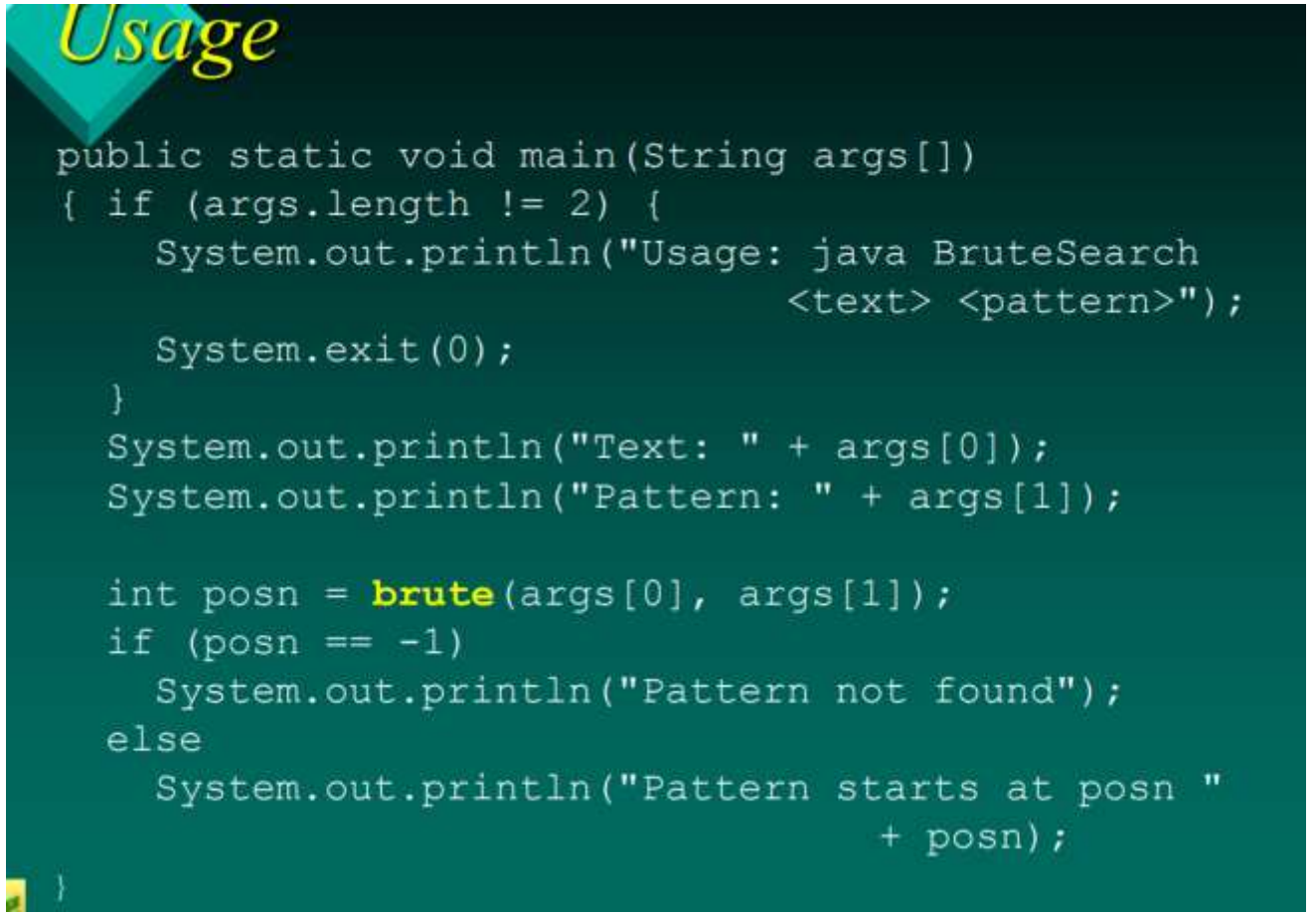
LANJUTAN

Brute Force in Java

Return index where pattern starts, or -1

```
public static int brute(String text,String pattern)
{ int n = text.length();     // n is length of text
  int m = pattern.length(); // m is length of pattern
  int j;
  for(int i=0; i <= (n-m); i++) {
    j = 0;
    while ((j < m) && (text.charAt(i+j)== pattern.charAt(j))
) {
        j++;
     }
    if (j == m)
        return i;    // match at i
  }
  return -1;    // no match
} // end of brute()
```

LANJUTAN


Usage
public static void main(String args[])
{ if (args.length != 2) {
System.out.println("Usage: java BruteSearch
<text> <pattern>");
System.exit(0);
}
System.out.println("Text: " + args[0]);
System.out.println("Pattern: " + args[1]);
int posn = brute(args[0], args[1]);
if (posn == -1)
System.out.println("Pattern not found");
else
System.out.println("Pattern starts at posn "
+ posn);
}

LANJUTAN

Analysis

Worst Case.

- Jumlah perbandingan: $m(n - m + 1) = O(mn)$
- Contoh:
  - T: "aaaaaaaaaaaaaaaaaaaaaaaaaaaah"
  - P: "aaah"

LANJUTAN

Best case

- Kompleksitas kasus terbaik adalah $O(n)$.
- Terjadi bila karakter pertama *pattern P* tidak pernah sama dengan karakter teks *T* yang dicocokkan
- Jumlah perbandingan maksimal *n* kali:
- Contoh:

```
T:  String ini berakhir dengan zzz
P:  zzz
```

LANJUTAN

Average Case

- But most searches of ordinary text take O(m+n), which is very quick.

- Example of a more average case:
  - T: "a string searching example is standard"
  - P: "store"

LANJUTAN

- The brute force algorithm is fast when the alphabet of the text is large
  - e.g. A..Z, a..z, 1..9, etc.
- It is slower when the alphabet is small
  - e.g. 0, 1 (as in binary files, image files, etc.)

Knuth-Morris-Pratt (KMP) Algorithm

- Pada algoritma brute force, jika terjadi mismatch, maka dilakukan pergeseran pattern sebanyak 1 karakter.
- Sedangkan pada algoritma Knuth-Morris-Pratt (KMP), setiap terjadi mismatch, pergeseran dilakukan berdasarkan informasi pattern (longest-prefix-suffix atau border function) yang telah diproses sebelum pencarian dimulai.
- Dengan informasi ini, jumlah pergeseran karakter dilakukan sesuai dengan informasi pattern yang dimiliki.

Ilustrasi

- Text: dead deadpool.
- Pattern: deadpool
- dead deadpool.

- Pattern ‘deadpool’ ditemukan pada karakter ke-6 text dengan 15 perbandingan karakter.
- Jika menggunakan brute force, diperlukan sebanyak 17 perbandingan karakter.
- Kompleksitas waktu untuk memproses border function adalah O(m), sedangkan pencarian pada text membutuhkan waktu O(n).
- Maka, kompleksitas waktu algoritma KMP adalah O(m+n).

LANJUTAN

- The Knuth-Morris-Pratt (KMP) algorithm looks for the pattern in the text in a *left-to-right* order (like the brute force algorithm).
- But it shifts the pattern more intelligently than the brute force algorithm.

LANJUTAN

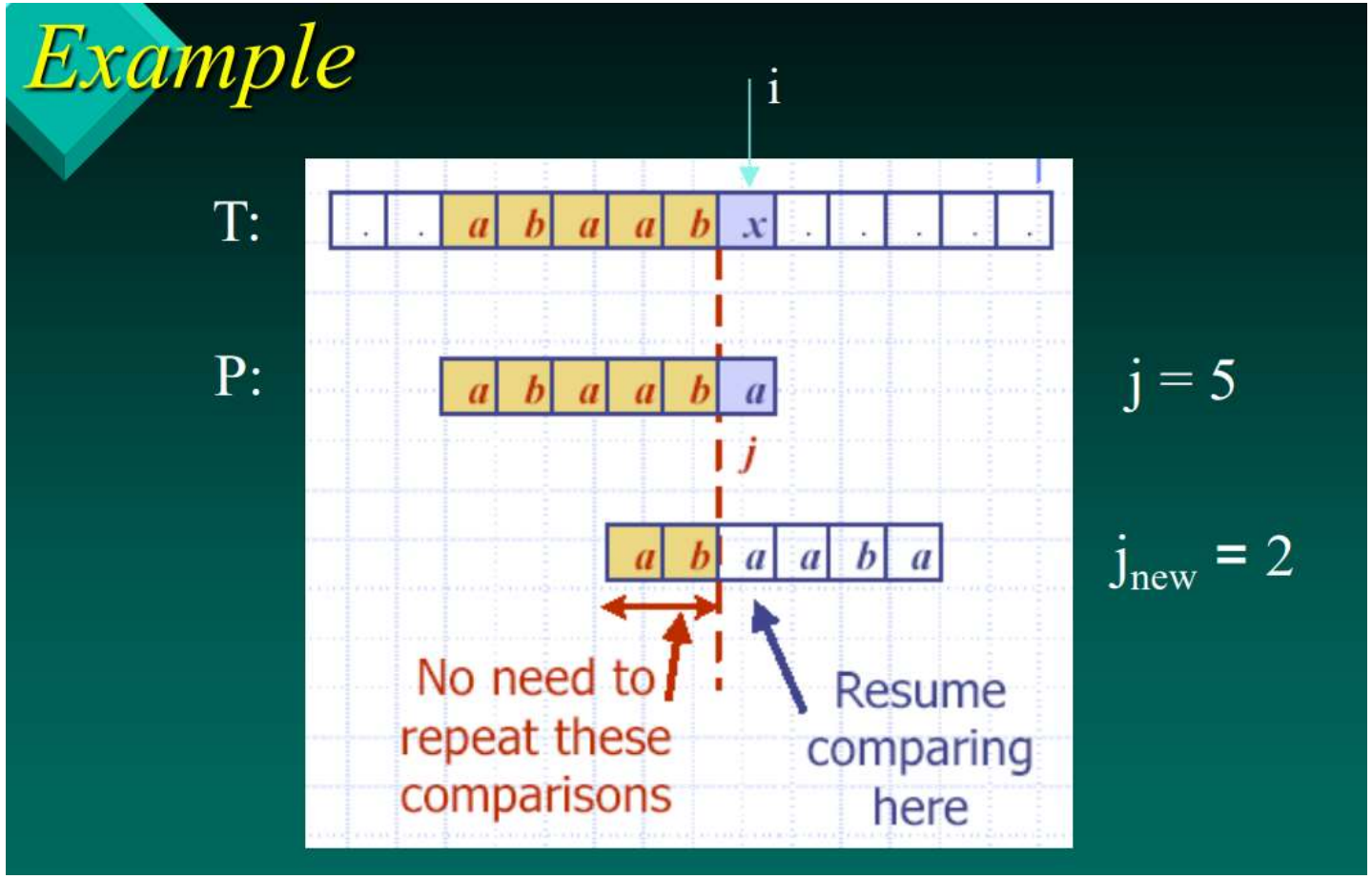
Donald Ervin Knuth (born January 10, 1938) is a computer scientist and Professor Emeritus at Stanford University. He is the author of the seminal multi-volume work *The Art of Computer Programming*.[3] Knuth has been called the "father" of the analysis of algorithms. He contributed to the development of the rigorous analysis of the computational complexity of algorithms and systematized formal mathematical techniques for it. In the process he also popularized the asymptotic notation.

LANJUTAN

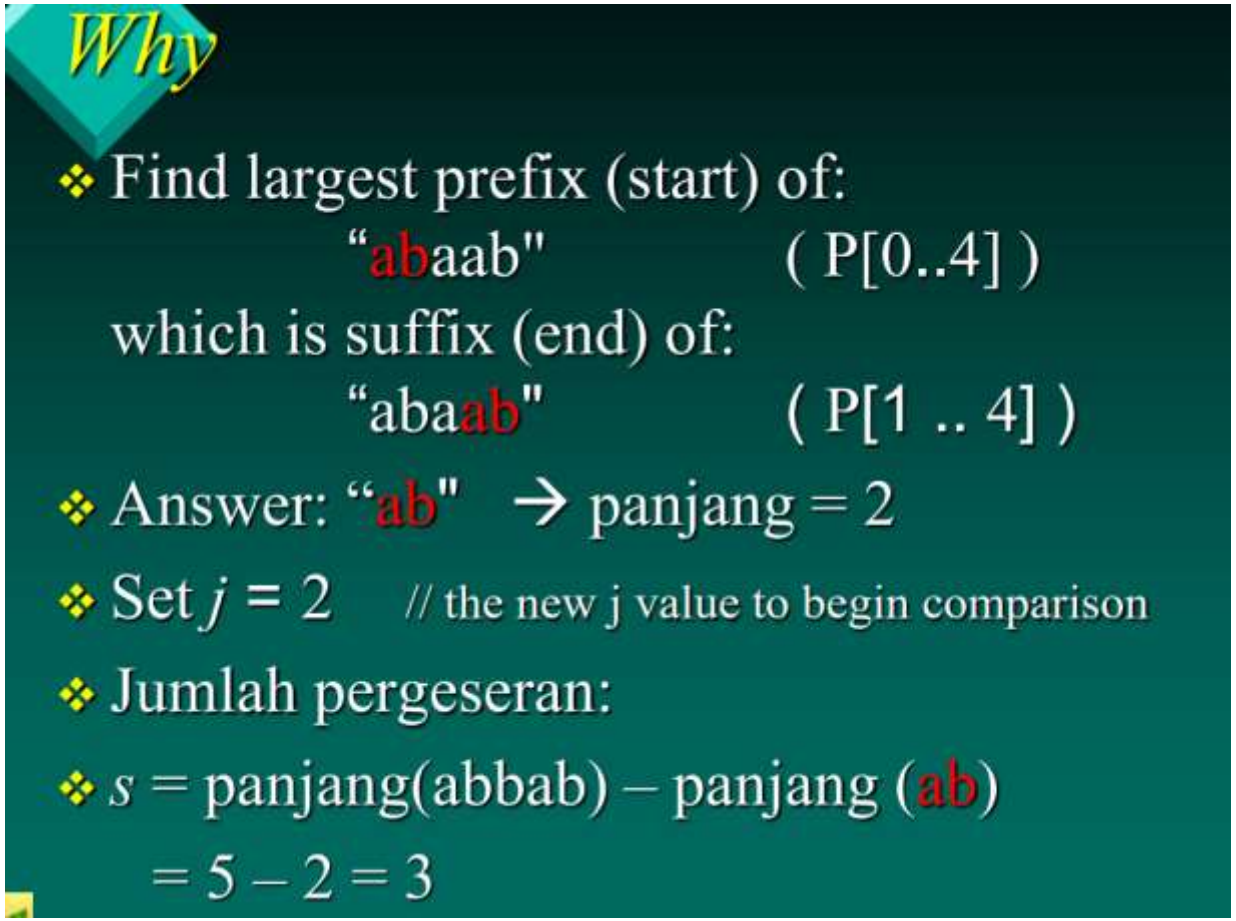
- If a mismatch occurs between the text and pattern P at P[j], i.e T[i] ≠ P[j], what is the *most* we can shift the pattern to avoid *wasteful comparisons*?

- *Answer*: the largest prefix of P[0 .. j-1] that is a suffix of P[1 .. j-1]

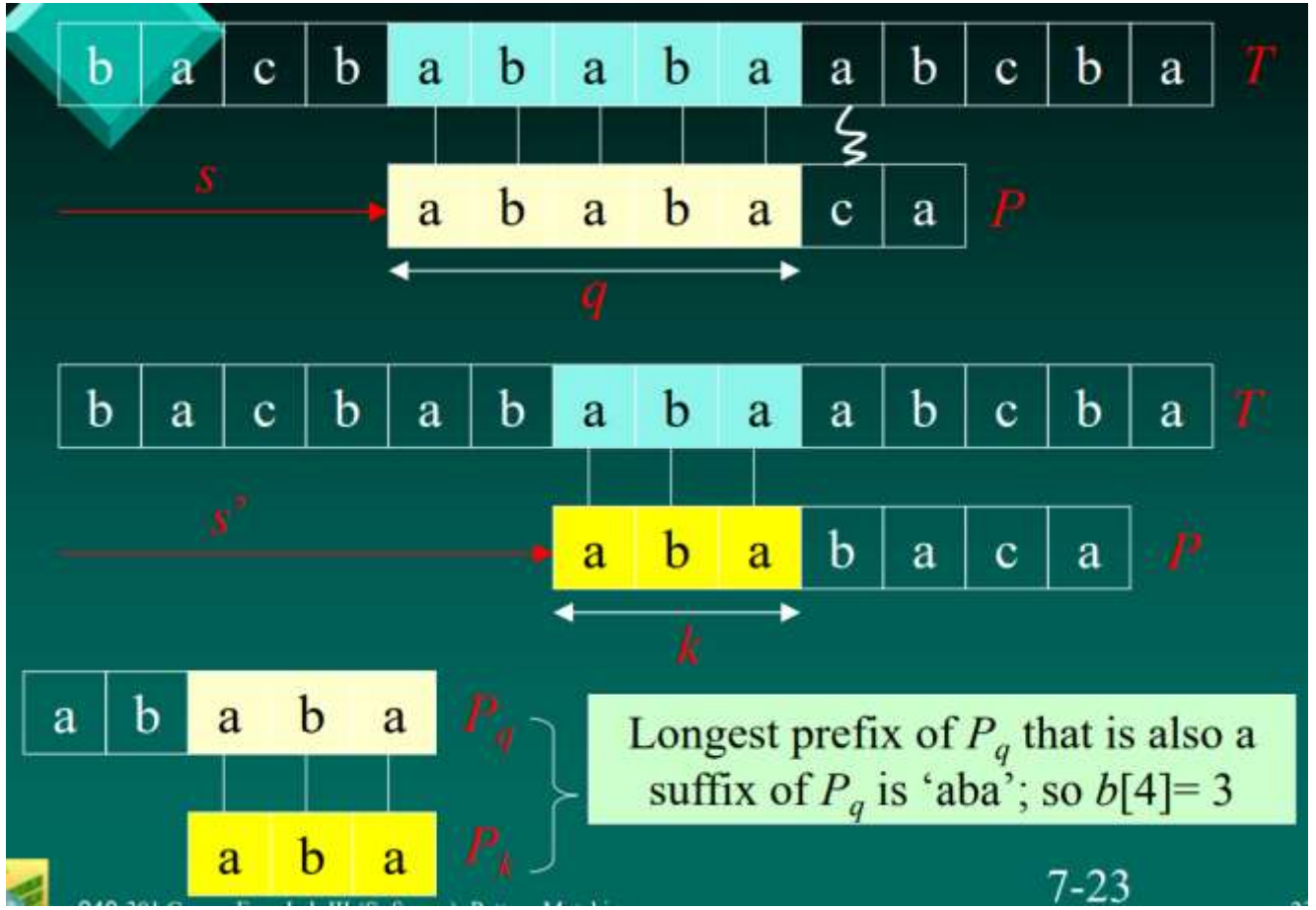
LANJUTAN


Example
i
T:
P:
j
j = 5
j_{new} = 2
No need to repeat these comparisons
Resume comparing here

LANJUTAN


Why
Find largest prefix (start) of:
"abaab" (P[0..4])
which is suffix (end) of:
"abaab" (P[1 .. 4])
Answer: "ab" → panjang = 2
Set j = 2 // the new j value to begin comparison
Jumlah pergeseran:
s = panjang(abbab) – panjang (ab)
= 5 – 2 = 3

LANJUTAN


b a c b a b a b a a b c b a T
s
a b a b a c a P
q
b a c b a b a b a a b c b a T
s'
a b a b a c a P
k
a b a b a P_q
a b a P_k
Longest prefix of P_q that is also a suffix of P_q is 'aba'; so $b[4]= 3$
7-23

LANJUTAN

Fungsi Pinggiran KMP (KMP Border Function)

- KMP preprocesses the pattern to find matches of prefixes of the pattern with the pattern itself.
- j = mismatch position in P[]
- k = position before the mismatch (k = j-1).
- The *border function* b(k) is defined as the *size* of the largest prefix of P[0..k] that is also a suffix of P[1..k].
- The other name: *failure function* (disingkat: *fail*)

LANJUTAN

Border Function Example

(k = j-1)

- P: abaaba

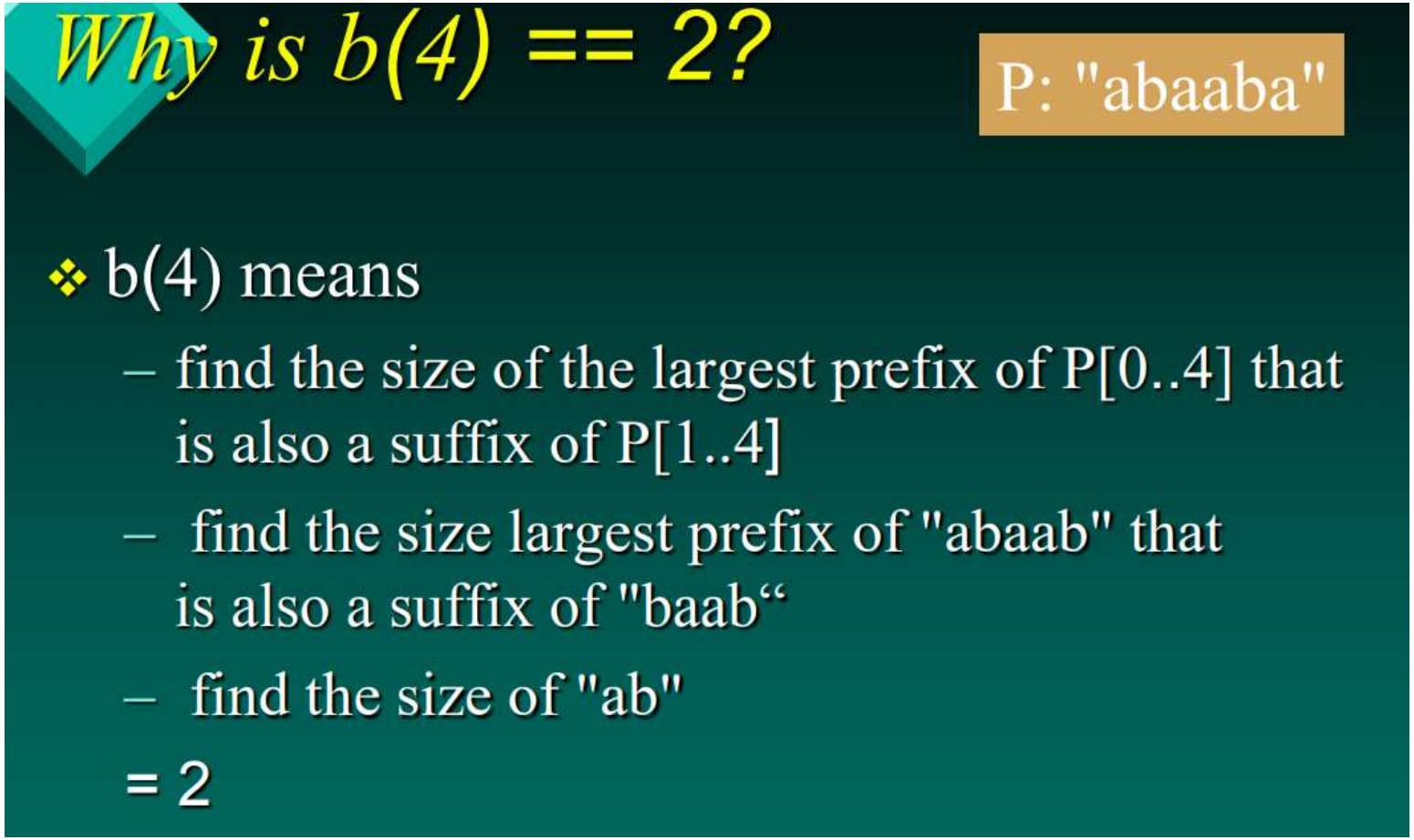
  j: 012345

j	0	1	2	3	4	5
P[j]	*a*	*b*	*a*	*a*	*b*	*a*
k	-	0	1	2	3	4
b(k)	-	0	0	1	1	2

b(k) is the size of the largest border.

- In code, b() is represented by an array, like the table.

LANJUTAN


Why is b(4) == 2?
P: "abaaba"
b(4) means
– find the size of the largest prefix of P[0..4] that is also a suffix of P[1..4]
– find the size largest prefix of "abaab" that is also a suffix of "baab“
– find the size of "ab"
= 2

LANJUTAN

Contoh lain: P = abababababca

j = 0123456789

(k = j-1)

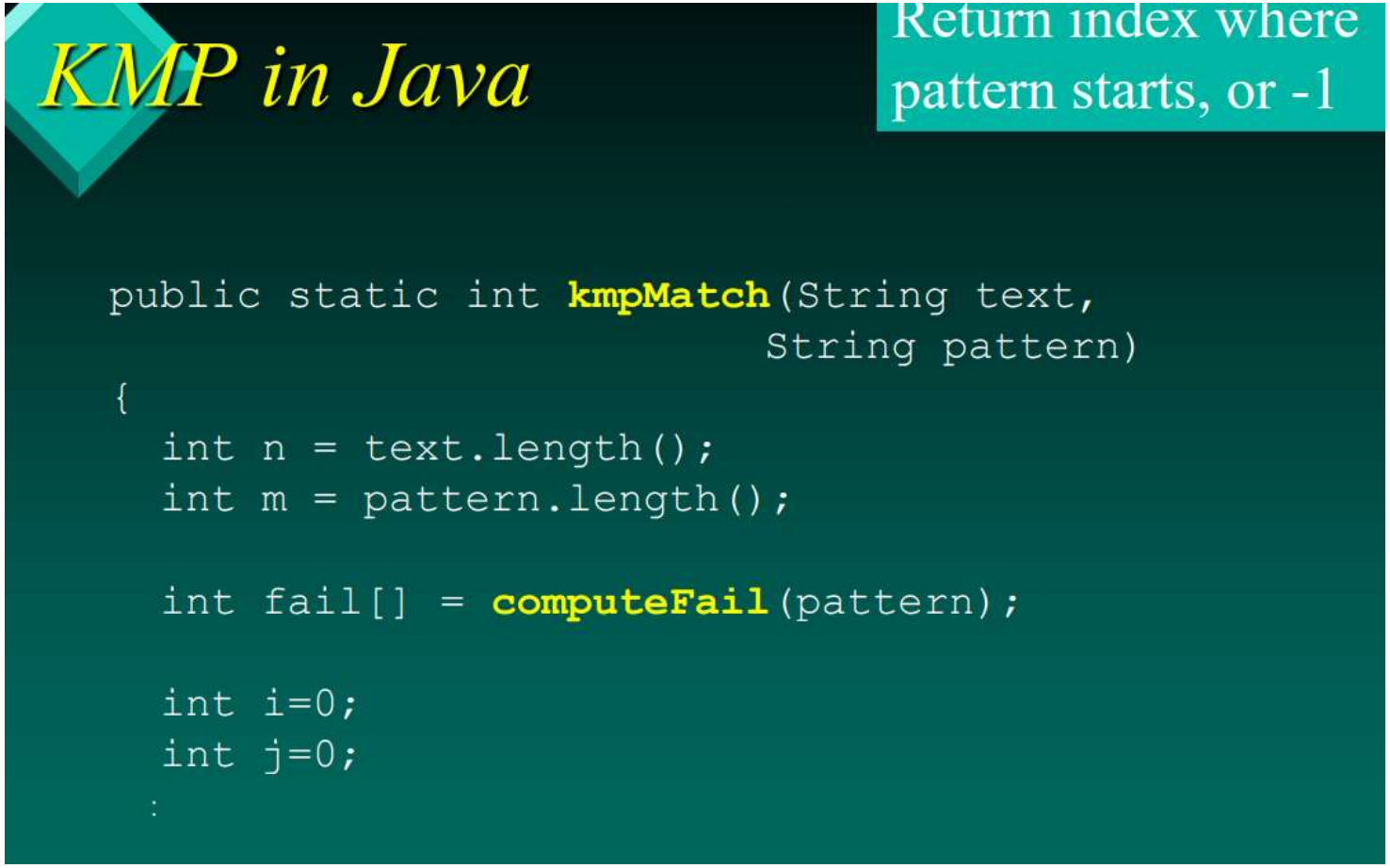
j	0	1	2	3	4	5	6	7	8	9
P[*j*]	a	b	a	b	a	b	a	b	c	a
k	-	0	1	2	3	4	5	6	7	8
b[*k*]	-	0	0	1	2	3	4	5	6	0

LANJUTAN

Using the Border Function

- Knuth-Morris-Pratt's algorithm modifies the brute-force algorithm.
  - if a mismatch occurs at P[j] (i.e. P[j] != T[i]), then
    k = j-1;
    j = b(k); // obtain the new j

LANJUTAN


KMP in Java
Return index where pattern starts, or -1
public static int kmpMatch(String text,
String pattern)
{
int n = text.length();
int m = pattern.length();
int fail[] = computeFail(pattern);
int i=0;
int j=0;

LANJUTAN

```
  while (i < n) {
    if (pattern.charAt(j) == text.charAt(i)) {
      if (j == m - 1)
        return i - m + 1; // match
      i++;
      j++;
    }
    else if (j > 0)
      j = fail[j-1];
    else
      i++;
  }
  return -1; // no match
} // end of kmpMatch()
```

LANJUTAN

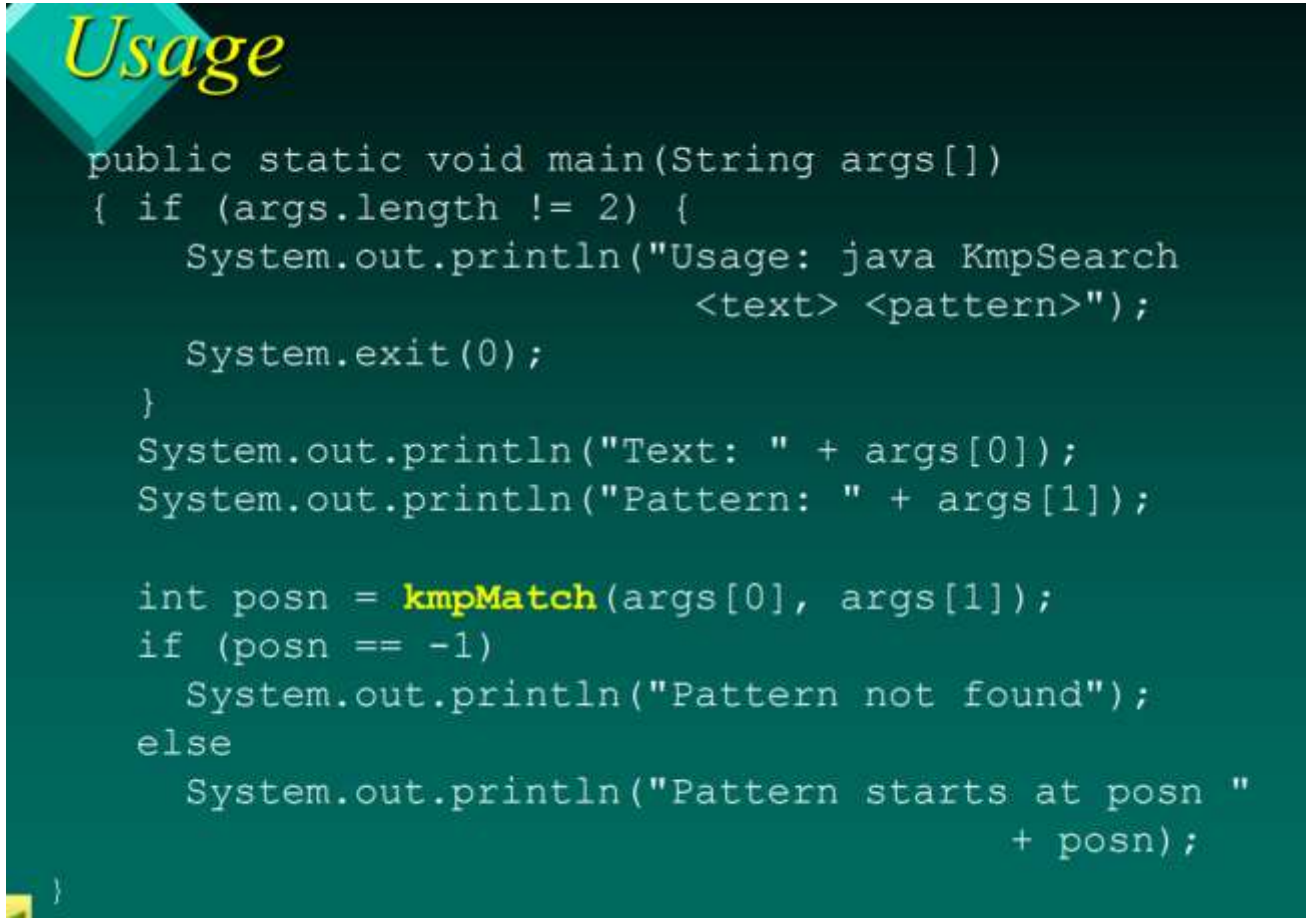
```
public static int[] computeFail(
                                String pattern)
{
   int fail[] = new int[pattern.length()];
   fail[0] = 0;

   int m = pattern.length();
   int j = 0;
   int i = 1;
```

LANJUTAN

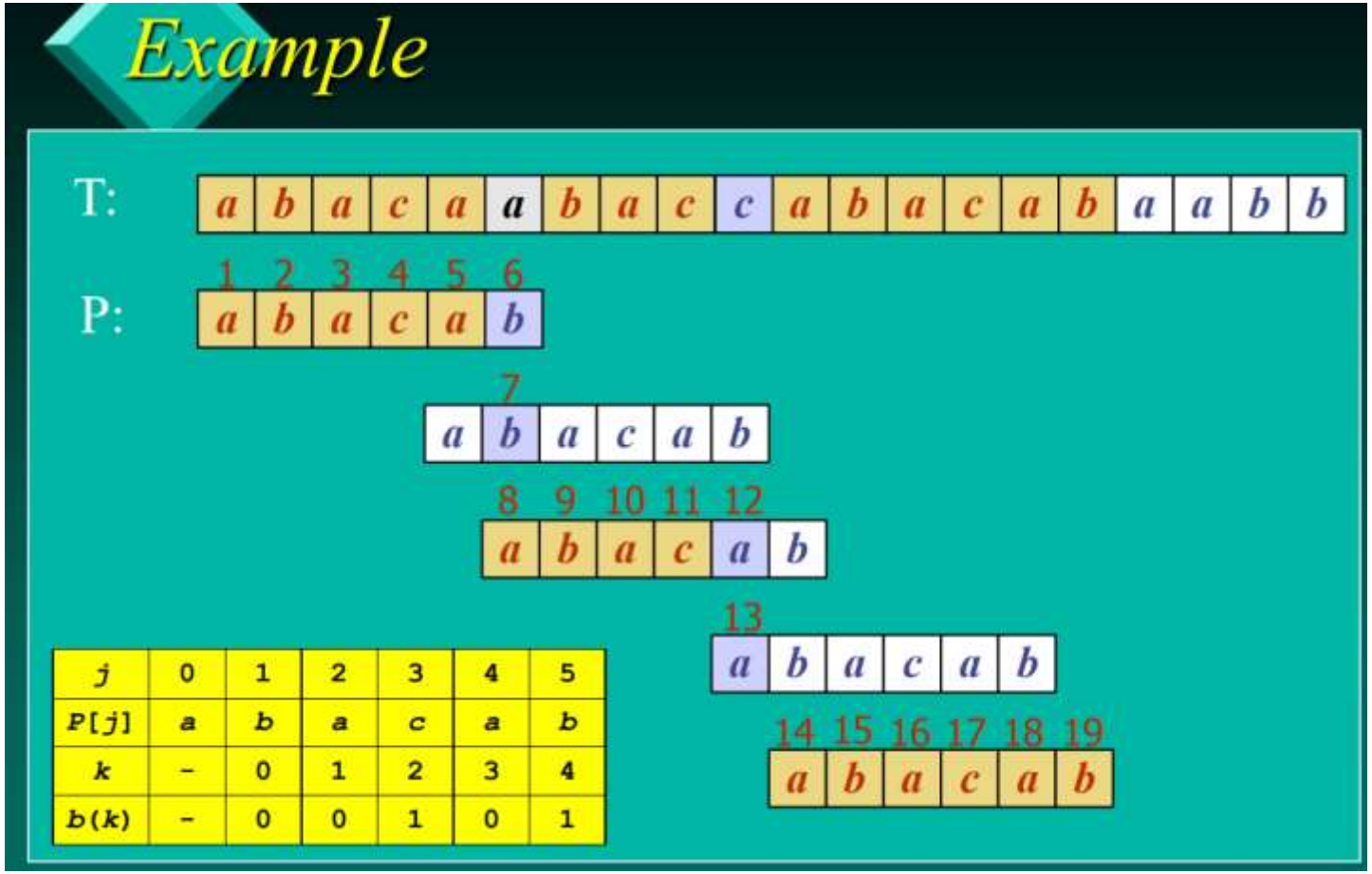
```
while (i < m) {
  if  (pattern.charAt(j) ==
       pattern.charAt(i)) {   //j+1 chars match
    fail[i] = j + 1;
    i++;
    j++;
  }
  else if (j > 0) // j follows matching prefix
    j = fail[j-1];
  else {     // no match
    fail[i] = 0;
    i++;
  }
}
return fail;
} // end of computeFail()
```

Similar code to kmpMatch()

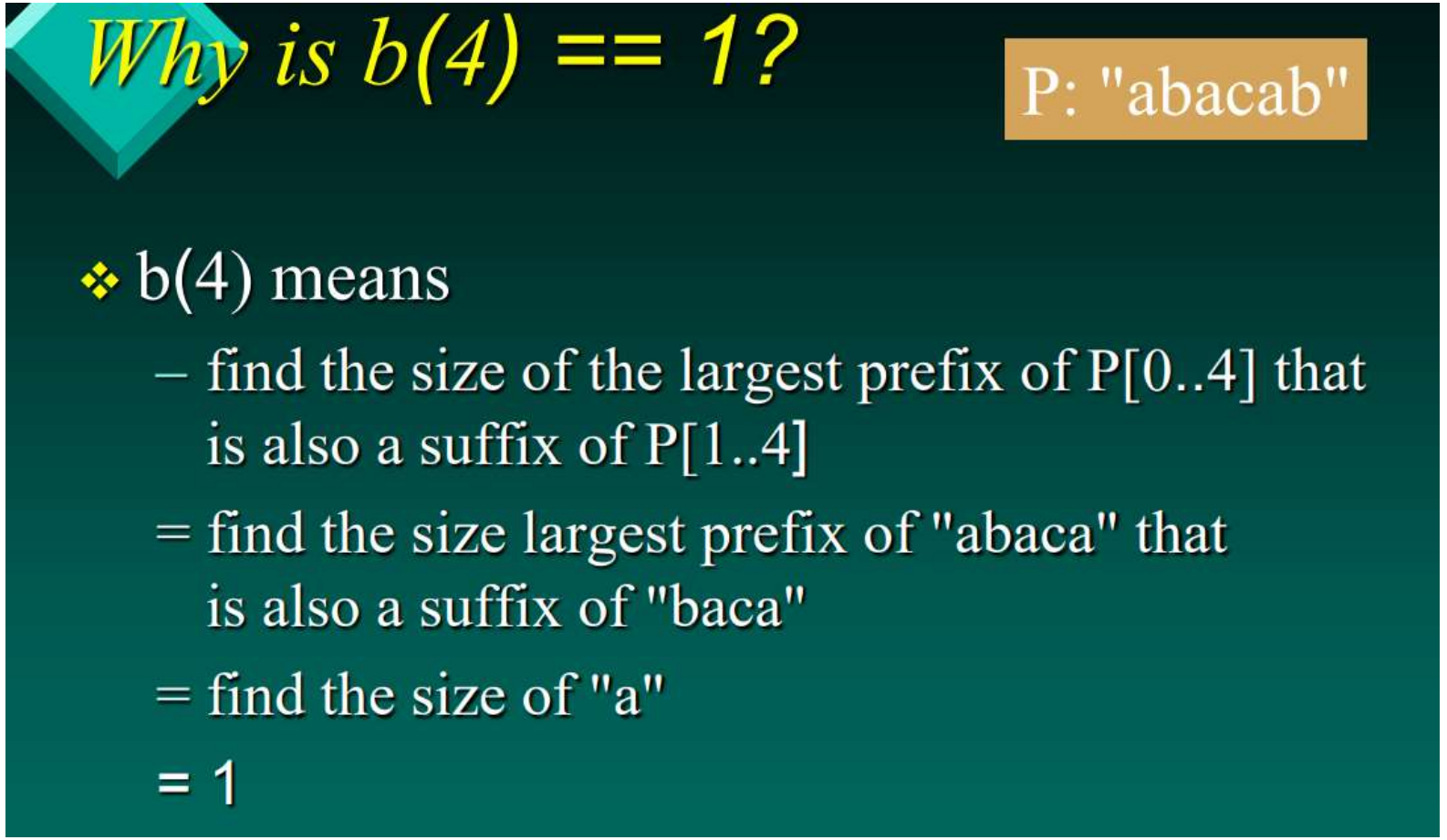
LANJUTAN


Usage
public static void main(String args[])
{ if (args.length != 2) {
System.out.println("Usage: java KmpSearch
<text> <pattern>");
System.exit(0);
}
System.out.println("Text: " + args[0]);
System.out.println("Pattern: " + args[1]);
int posn = kmpMatch(args[0], args[1]);
if (posn == -1)
System.out.println("Pattern not found");
else
System.out.println("Pattern starts at posn "
+ posn);
}

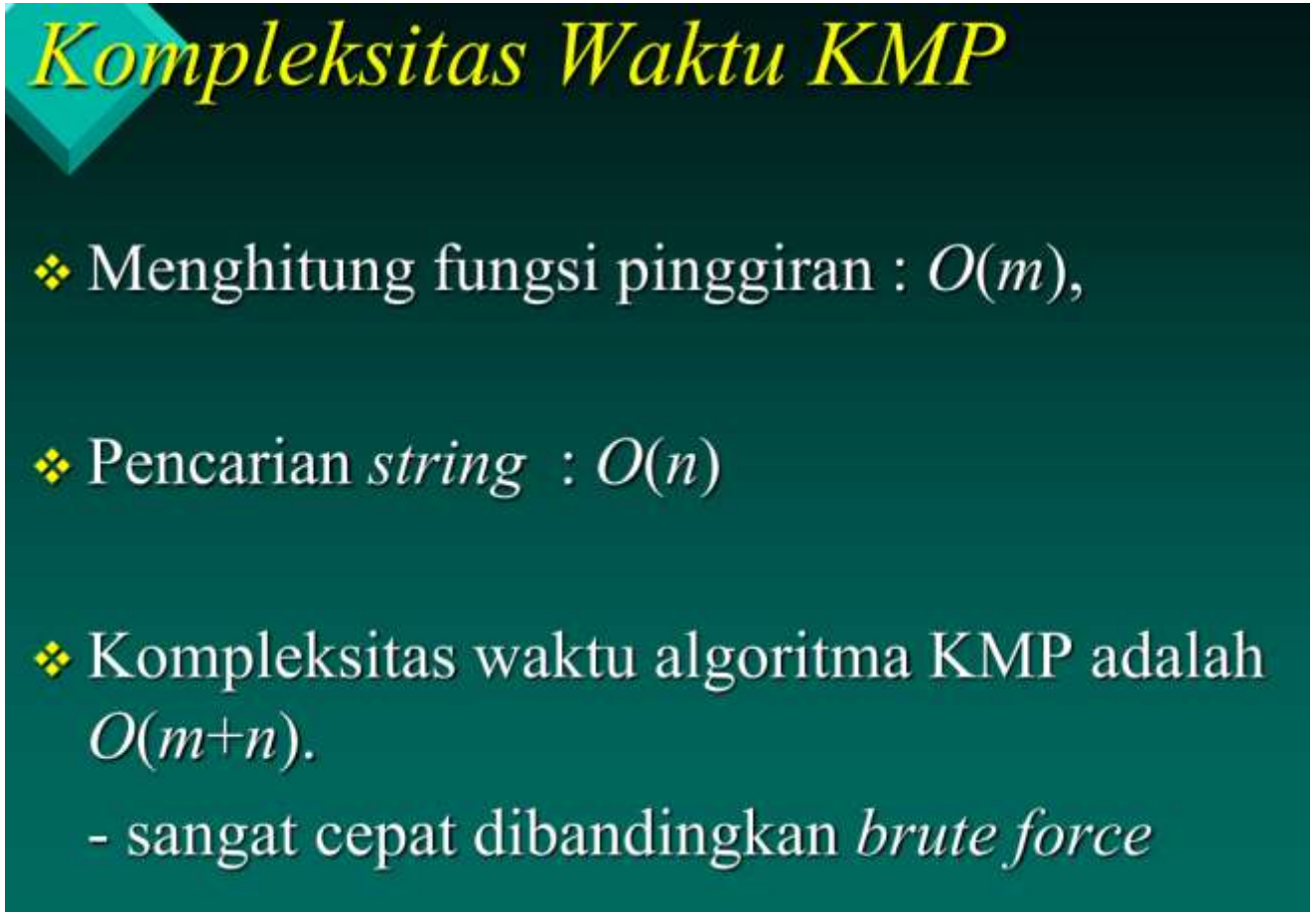
LANJUTAN


Example
T:
a b a c a a b a c c a b a c a b a a b b
1 2 3 4 5 6
P:
a b a c a b
7
a b a c a b
8 9 10 11 12
a b a c a b
13
a b a c a b
14 15 16 17 18 19
a b a c a b
j 0 1 2 3 4 5
P[j] a b a c a b
k - 0 1 2 3 4
b(k) - 0 0 1 0 1

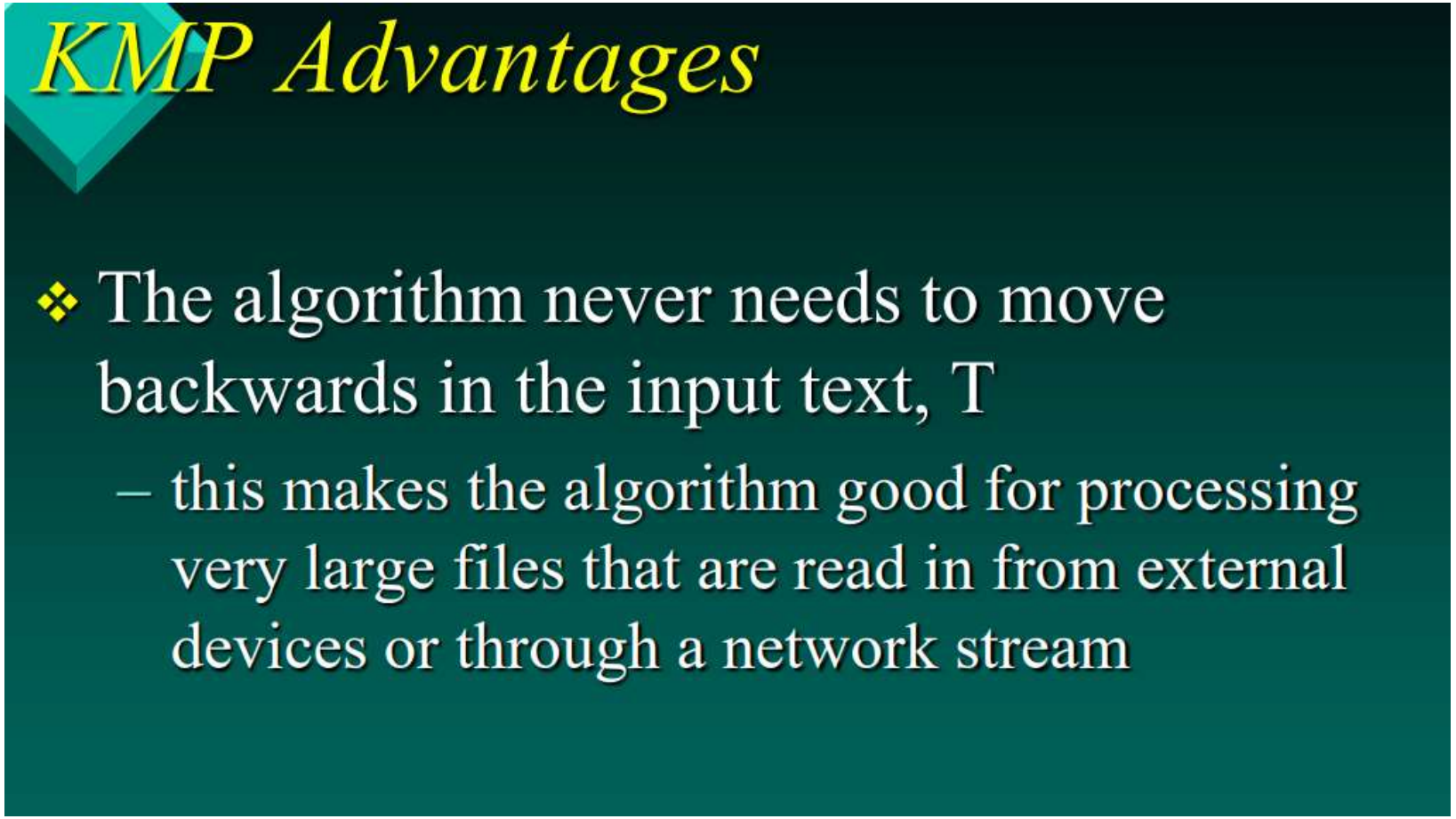
LANJUTAN


Why is b(4) == 1?
P: "abacab"
b(4) means
– find the size of the largest prefix of P[0..4] that is also a suffix of P[1..4]
= find the size largest prefix of "abaca" that is also a suffix of "baca"
= find the size of "a"
= 1

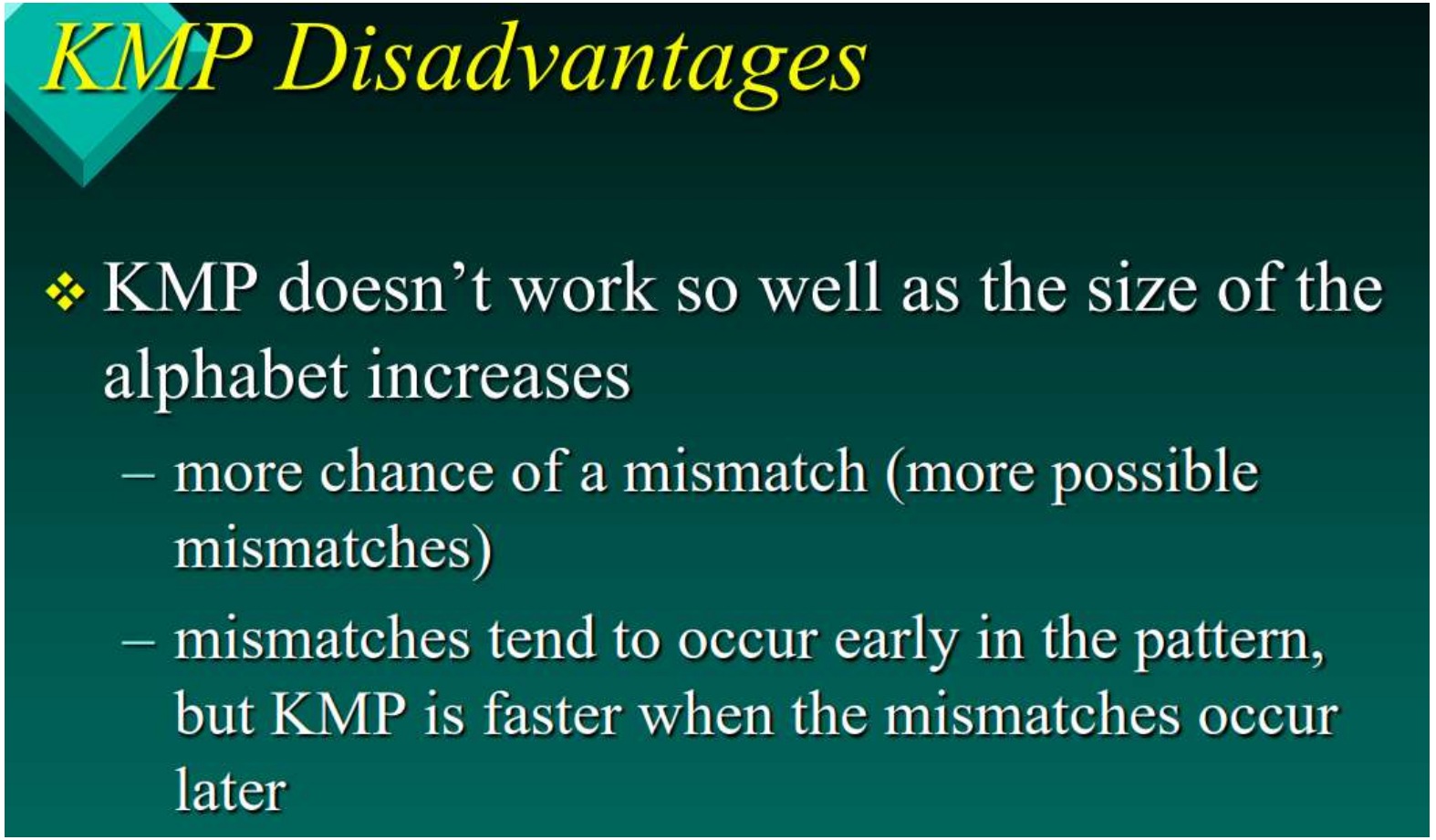
LANJUTAN


Kompleksitas Waktu KMP
Menghitung fungsi pinggiran : O(m),
Pencarian string : O(n)
Kompleksitas waktu algoritma KMP adalah O(m+n).
- sangat cepat dibandingkan brute force

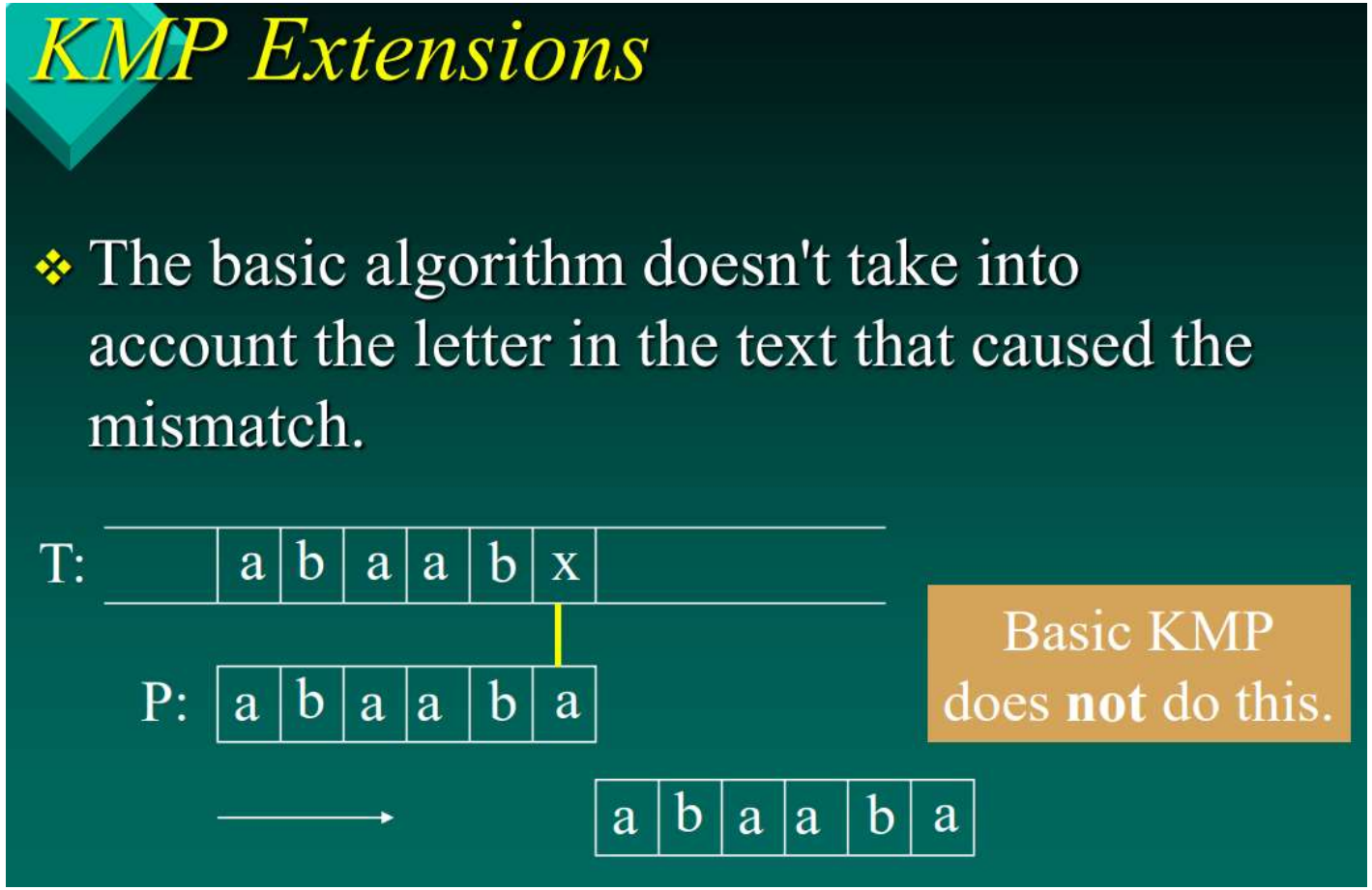
LANJUTAN


KMP Advantages
The algorithm never needs to move backwards in the input text, T
– this makes the algorithm good for processing very large files that are read in from external devices or through a network stream

LANJUTAN


KMP Disadvantages
KMP doesn't work so well as the size of the alphabet increases
– more chance of a mismatch (more possible mismatches)
– mismatches tend to occur early in the pattern, but KMP is faster when the mismatches occur later

LANJUTAN

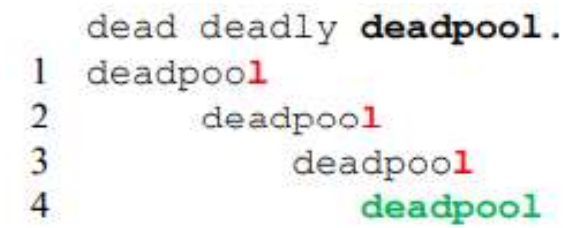

KMP Extensions
The basic algorithm doesn't take into account the letter in the text that caused the mismatch.
T:
a b a a b x
P:
a b a a b a
a b a a b a
Basic KMP does not do this.

Boyer-Moore Algorithm

- Algoritma Boyer-Moore didasarkan pada dua teknik, yakni:
- 1. The looking glass technique, yakni membandingkan karakter pada pattern dan text dimulai dari belakang-kedepan (dimulai dari karakter terakhir pattern)
- 2. The character-jump technique, yakni jika terjadi mismatch pada karakter x di text T[i] dengan karakter y di pattern P[j], maka ada 3 kasus yang dicoba secara berurutan, yakni:
  - a. Jika P berisi karakter x di kiri dari lokasi mismatch, maka sejajarkan x dengan kemunculan terakhir x pada pattern.
  - b. Jika P berisi karakter x hanya di kanan dari lokasi mismatch, maka geser pattern sebanyak 1 karakter ke kanan.
  - c. Jika P tidak berisi karakter x, maka sejajarkan P[1] dengan T[i+1].

Ilustrasi

- Text: dead deadly deadpool.
- Pattern: deadpool
- dead deadly deadpool.



- Pattern ‘deadpool’ ditemukan pada karakter ke-13 text dengan 11 perbandingan karakter.
- Kompleksitas waktu terburuk algoritma Boyer-Moore adalah O(mn+A), dimana A adalah waktu yang dibutuhkan untuk melakukan pre-processing alfabet pada pattern

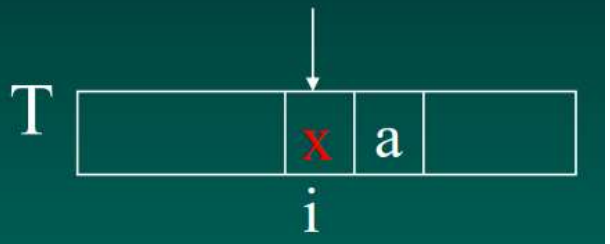
LANJUTAN

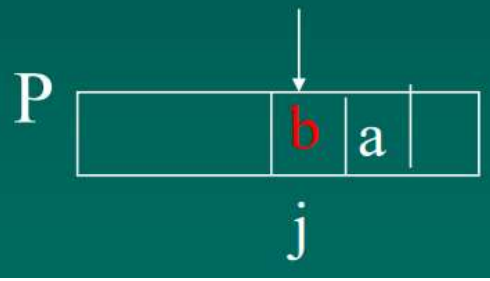
- The Boyer-Moore pattern matching algorithm is based on two techniques.
- 1. The *looking-glass* technique
  - find P in T by moving *backwards* through P, starting at its end

LANJUTAN

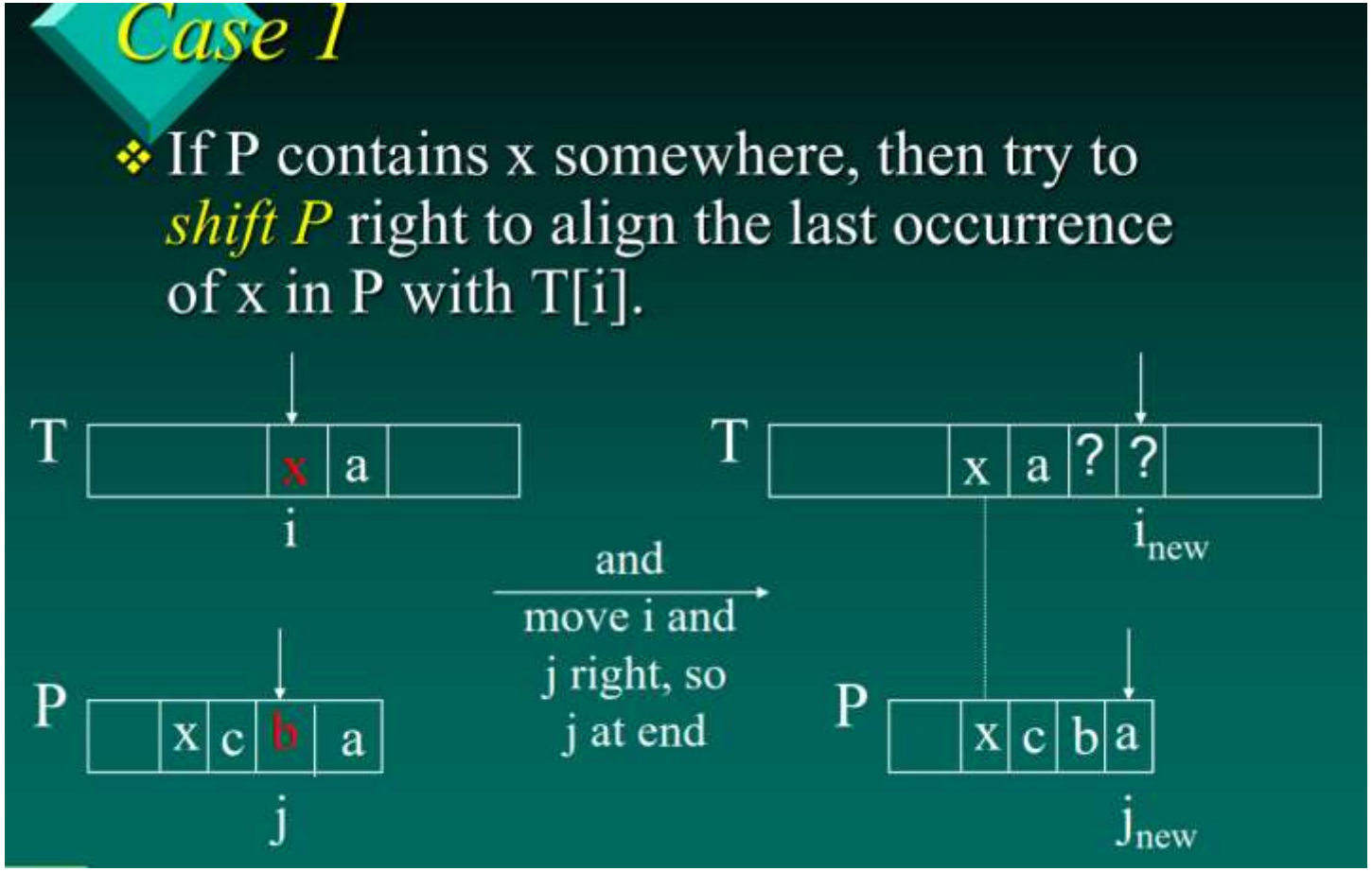
- 2. The *character-jump* technique
  - when a mismatch occurs at T[i] == x
  - the character in pattern P[j] is not the same as T[i]

- There are 3 possible cases, tried in order.

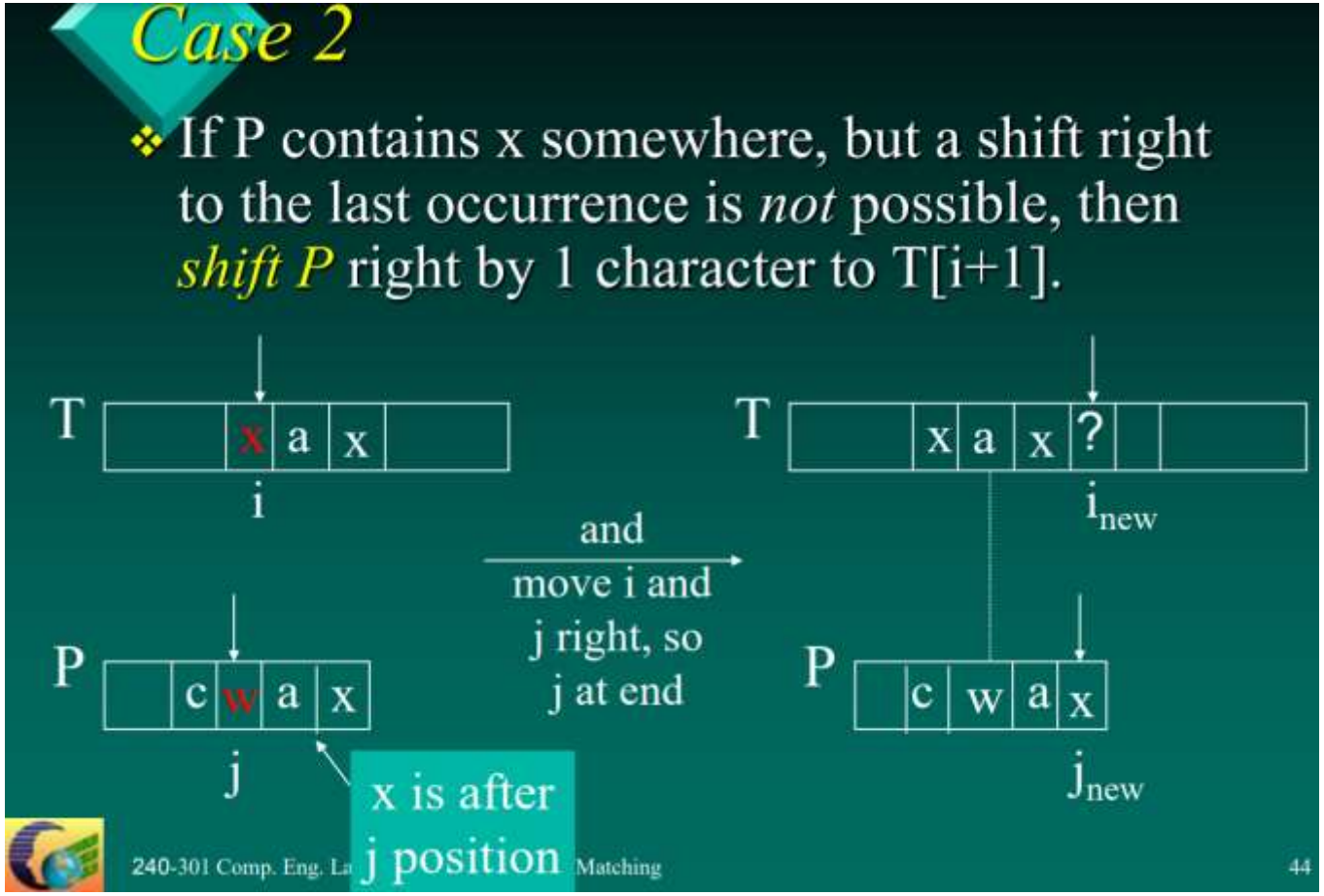




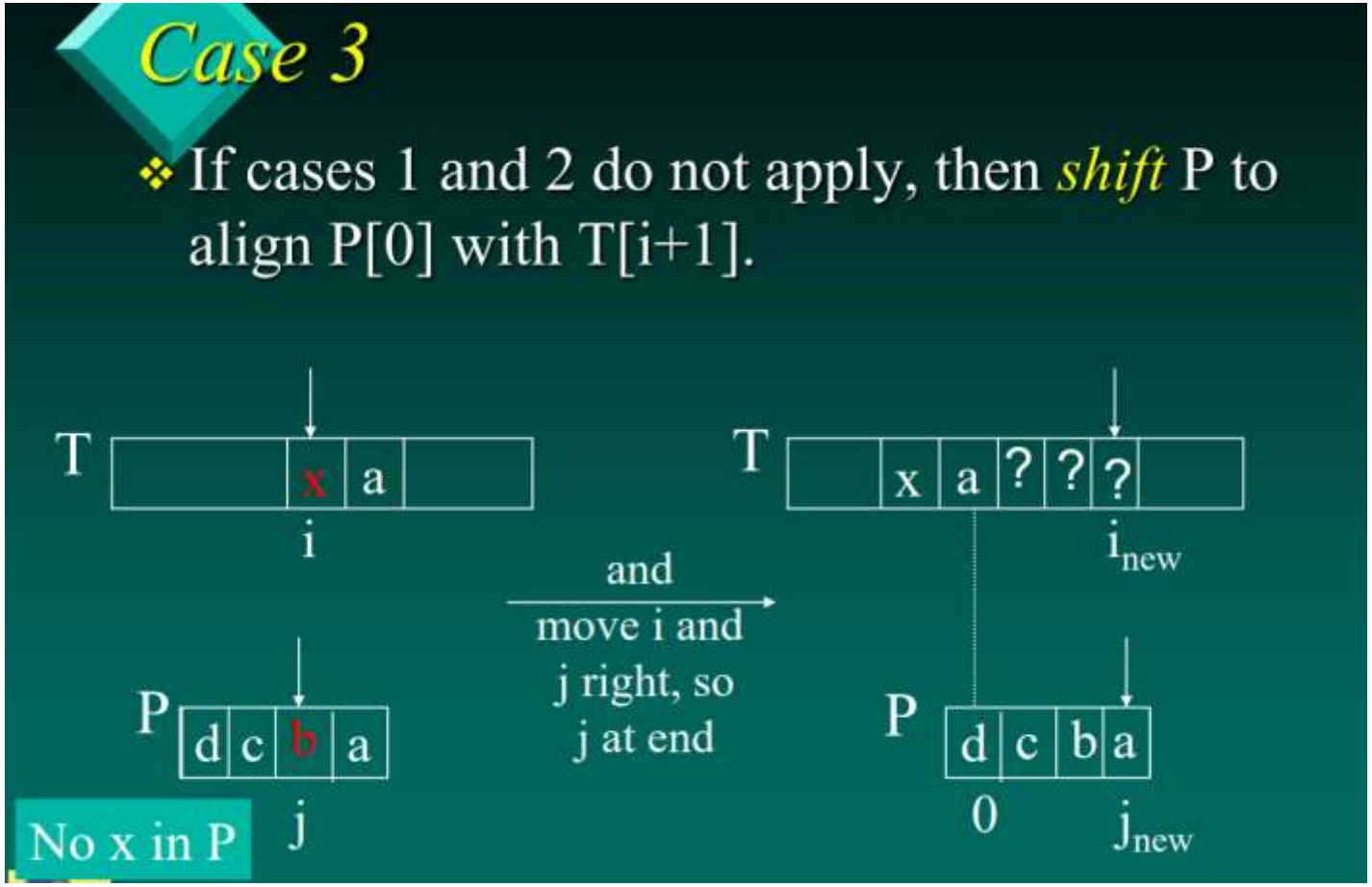
LANJUTAN


Case 1
If P contains x somewhere, then try to shift P right to align the last occurrence of x in P with T[i].
T
x a
i
P
x c b a
j
and
move i and
j right, so
j at end
T
x a ? ?
i new
P
x c b a
j new

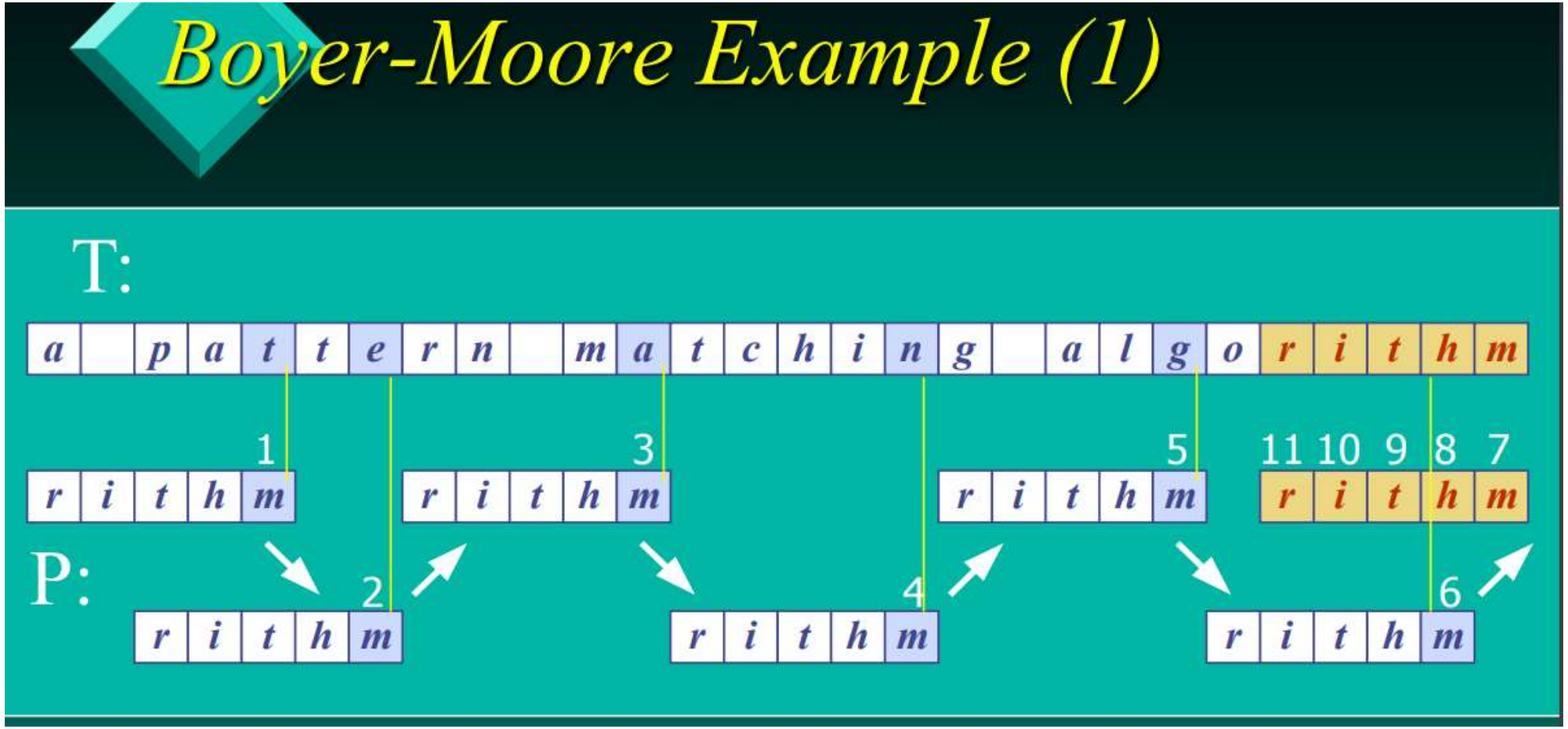
LANJUTAN


Case 2
If P contains x somewhere, but a shift right to the last occurrence is *not* possible, then *shift P* right by 1 character to T[i+1].
T
x a x
i
and
move i and
j right, so
j at end
T
x a x ?
i_{new}
P
c w a x
j
x is after
j position
P
c w a x
j_{new}
240-301 Comp. Eng. La
Matching
44

LANJUTAN


Case 3
If cases 1 and 2 do not apply, then shift P to align P[0] with T[i+1].
T
x a
i
P
d c b a
j
No x in P
and
move i and
j right, so
j at end
T
x a ? ? ?
i new
P
d c b a
0
j new

LANJUTAN


Boyer-Moore Example (1)
T:
a p a t t e r n m a t c h i n g a l g o r i t h m
P:
r i t h m
1
2
3
4
5
6
7
8
9
10
11

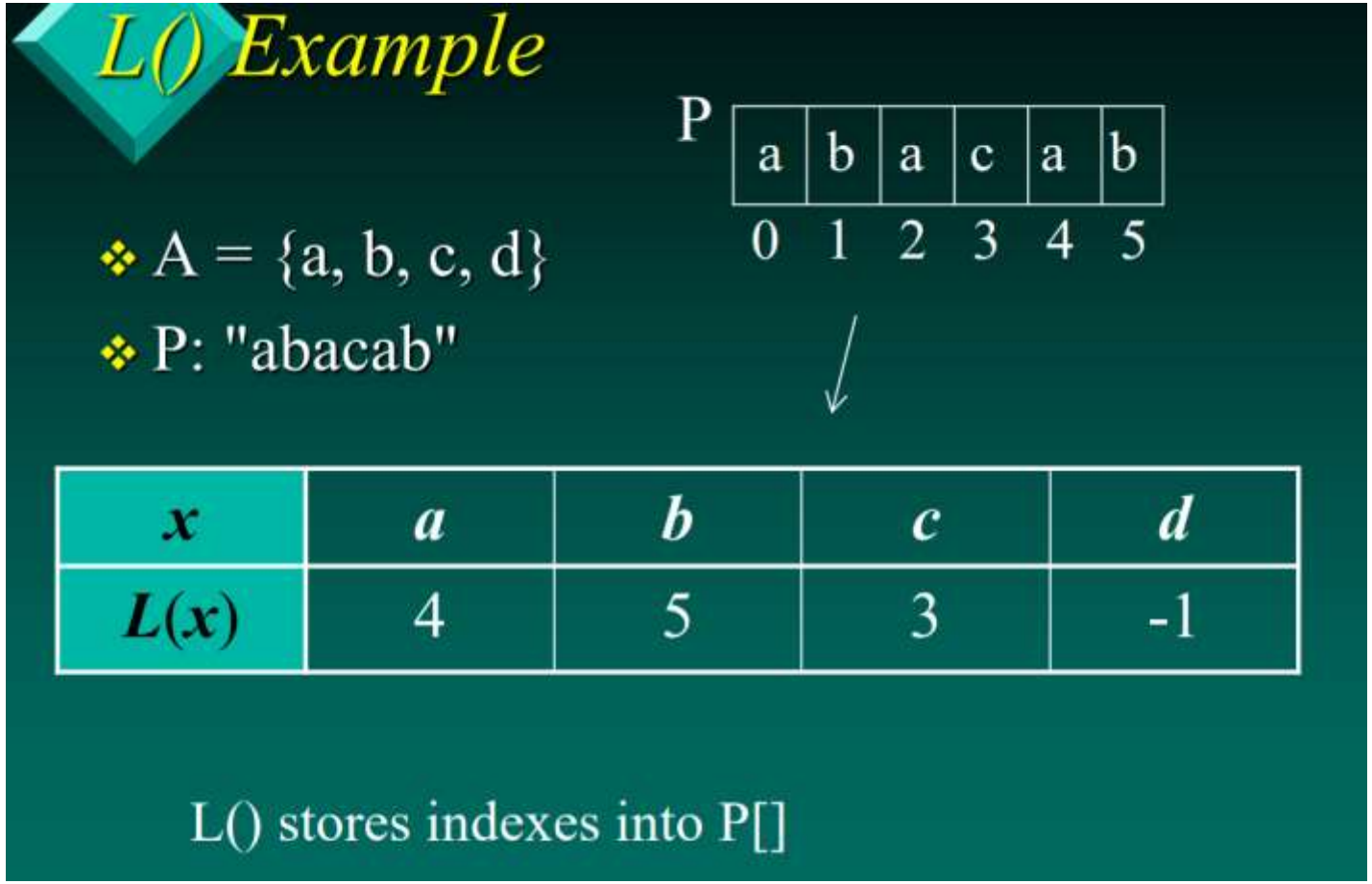
LANJUTAN

Last Occurrence Function

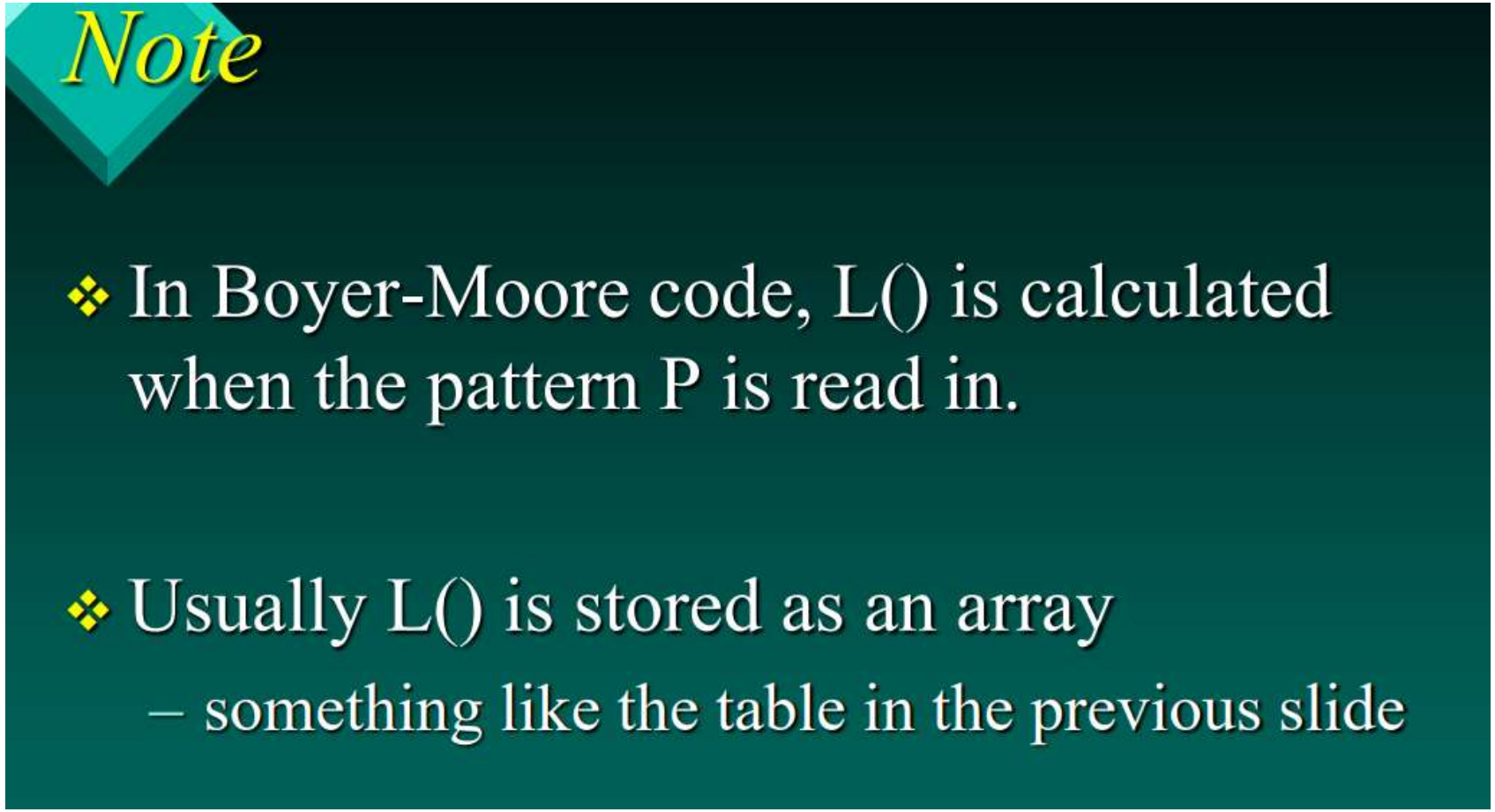
- Boyer-Moore's algorithm preprocesses the pattern P and the alphabet A to build a last occurrence function L()
  - L() maps all the letters in A to integers

- L(x) is defined as: // x is a letter in A
  - the largest index i such that P[i] == x, or
  - -1 if no such index exists

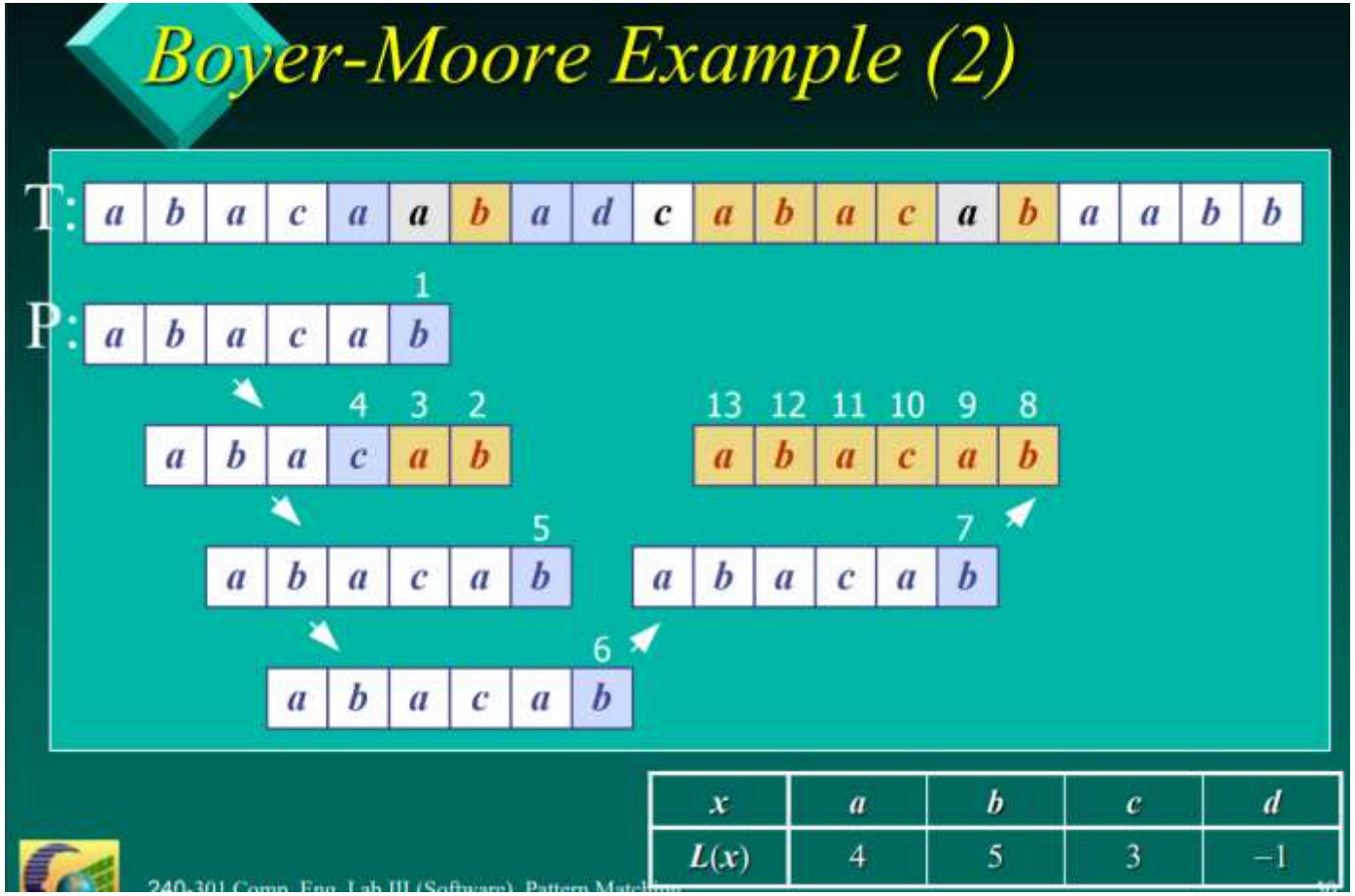
LANJUTAN


L() Example
P
a b a c a b
0 1 2 3 4 5
A = {a, b, c, d}
P: "abacab"
x | a | b | c | d
L(x) | 4 | 5 | 3 | -1
L() stores indexes into P[]

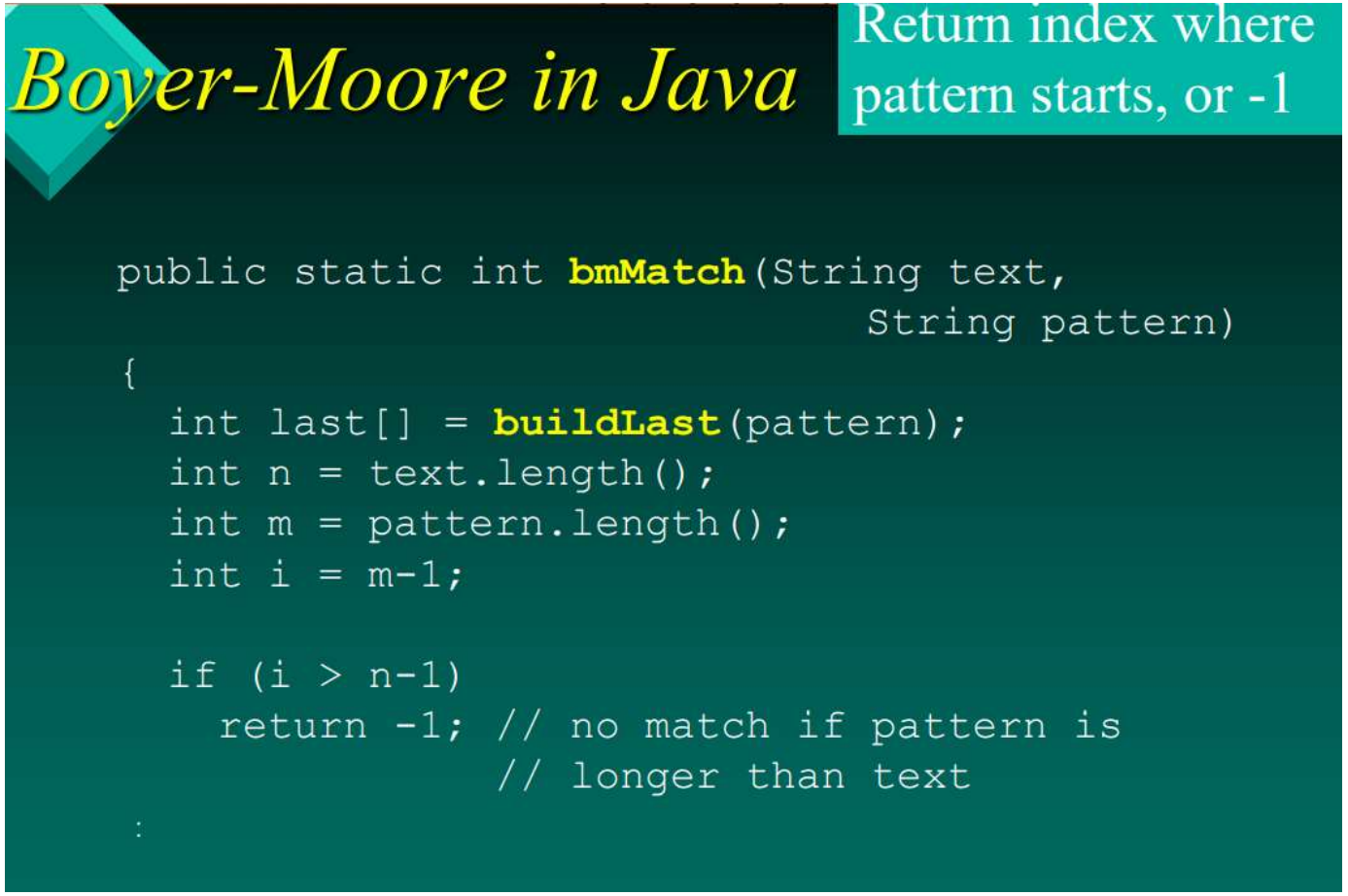
LANJUTAN


Note
In Boyer-Moore code, L() is calculated when the pattern P is read in.
Usually L() is stored as an array
– something like the table in the previous slide

LANJUTAN


Boyer-Moore Example (2)
T: a b a c a a b a d c a b a c a b a a b b
P: a b a c a b
1
4 3 2
a b a c a b
13 12 11 10 9 8
a b a c a b
5
a b a c a b
7
a b a c a b
6
a b a c a b
x | a | b | c | d
L(x) | 4 | 5 | 3 | −1

LANJUTAN


Boyer-Moore in Java
Return index where pattern starts, or -1
public static int bmMatch(String text,
String pattern)
{
int last[] = buildLast(pattern);
int n = text.length();
int m = pattern.length();
int i = m-1;
if (i > n-1)
return -1; // no match if pattern is
// longer than text

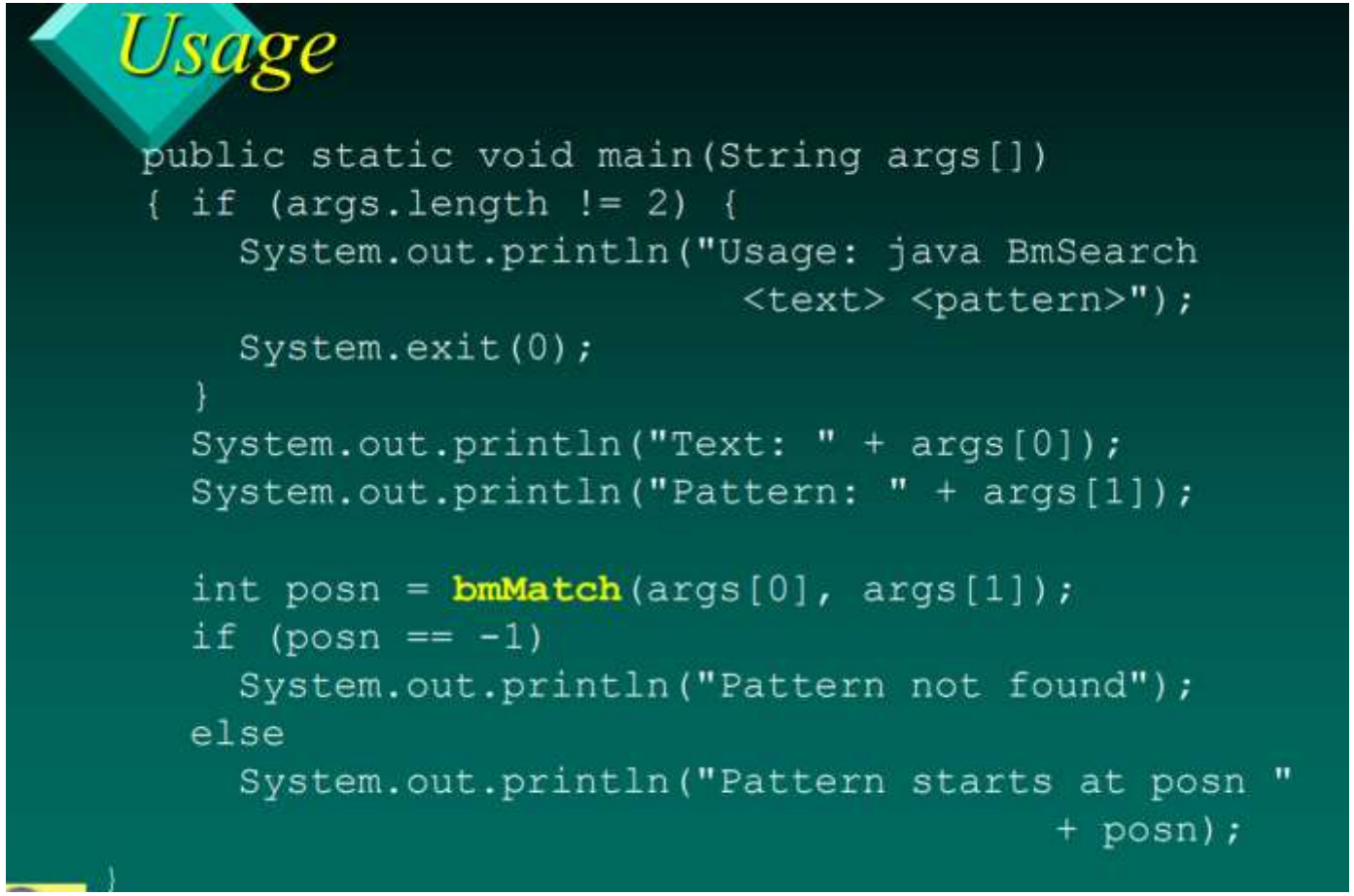
LANJUTAN

```
    int j = m-1;
    do {
      if (pattern.charAt(j) == text.charAt(i))
        if (j == 0)
          return i; // match
        else { // looking-glass technique
          i--;
          j--;
        }
      else { // character jump technique
         int lo = last[text.charAt(i)];  //last occ
         i = i + m - Math.min(j, 1+lo);
         j = m - 1;
      }
    } while (i <= n-1);

    return -1; // no match
  } // end of bmMatch()
```

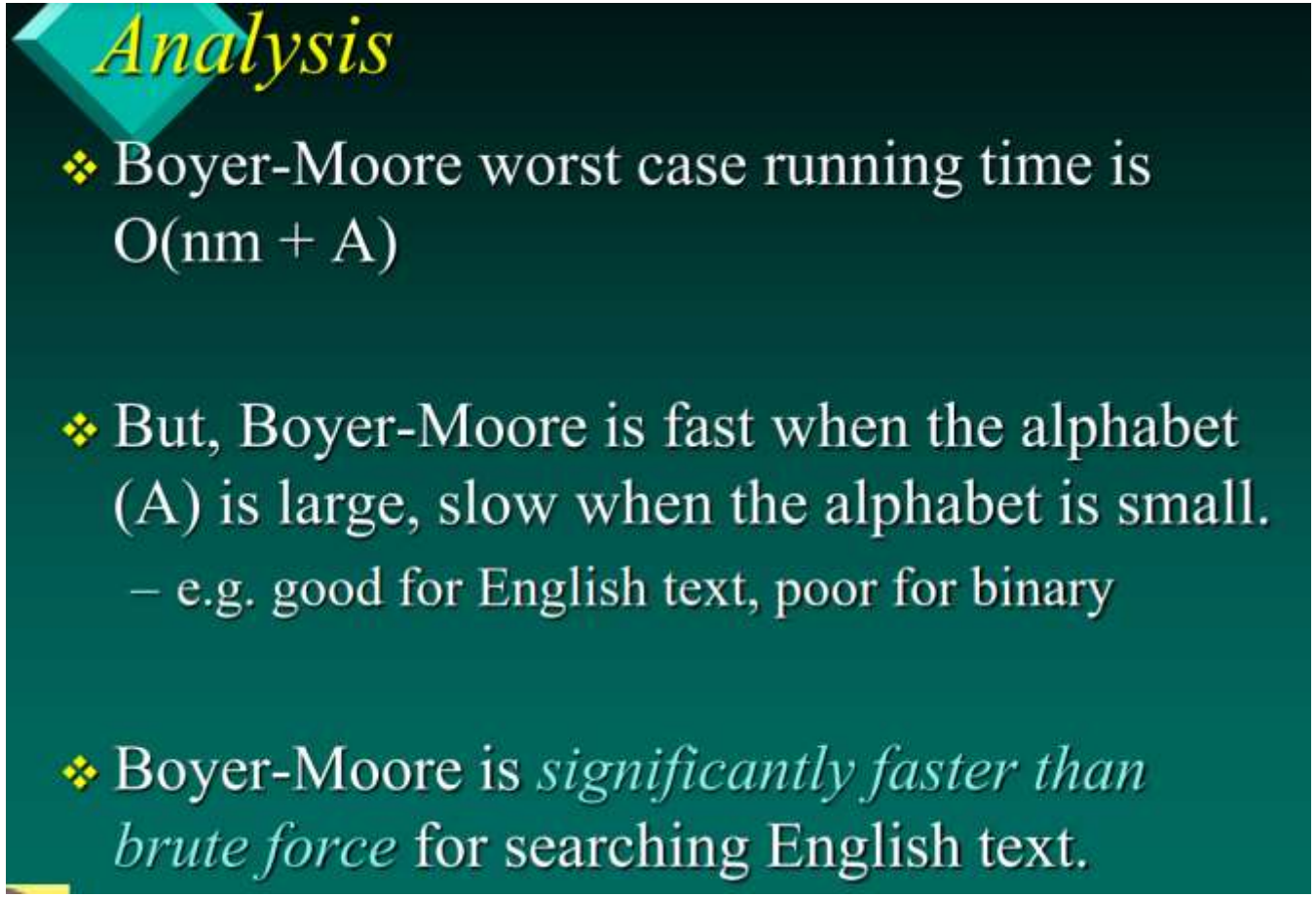
LANJUTAN

```
public static int[] buildLast(String pattern)
/* Return array storing index of last
   occurrence of each ASCII char in pattern. */
{
   int last[] = new int[128]; // ASCII char set

   for(int i=0; i < 128; i++)
      last[i] = -1; // initialize array

   for (int i = 0; i < pattern.length(); i++)
      last[pattern.charAt(i)] = i;

   return last;
} // end of buildLast()
```

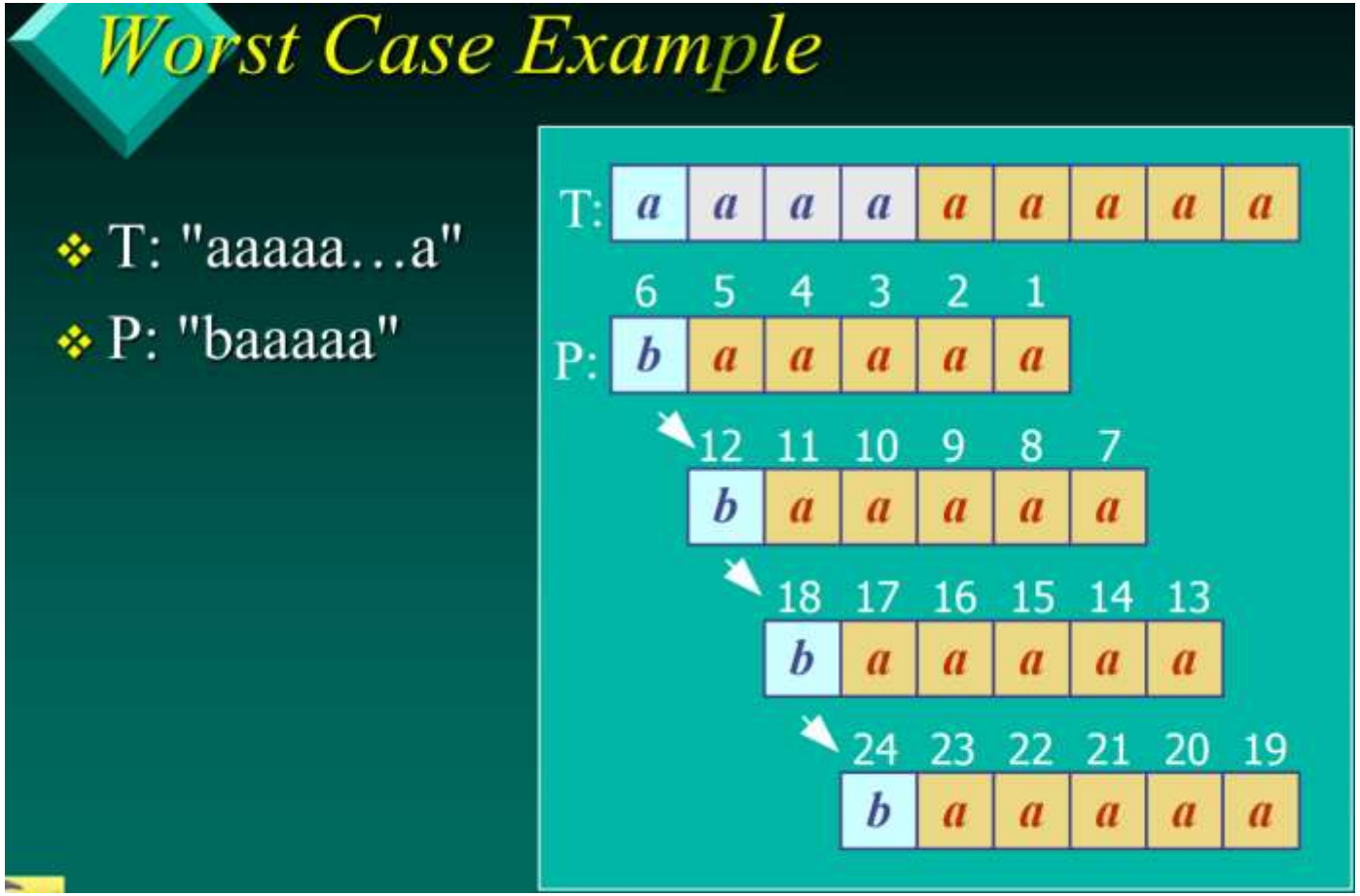
LANJUTAN


Usage
public static void main(String args[])
{ if (args.length != 2) {
System.out.println("Usage: java BmSearch
<text> <pattern>");
System.exit(0);
}
System.out.println("Text: " + args[0]);
System.out.println("Pattern: " + args[1]);
int posn = bmMatch(args[0], args[1]);
if (posn == -1)
System.out.println("Pattern not found");
else
System.out.println("Pattern starts at posn "
+ posn);

LANJUTAN


Analysis
Boyer-Moore worst case running time is O(nm + A)
But, Boyer-Moore is fast when the alphabet (A) is large, slow when the alphabet is small.
– e.g. good for English text, poor for binary
Boyer-Moore is *significantly faster than brute force* for searching English text.

LANJUTAN


Worst Case Example
T: "aaaaa…a"
P: "baaaaa"
T: a a a a a a a a a
6 5 4 3 2 1
P: b a a a a a
12 11 10 9 8 7
b a a a a a
18 17 16 15 14 13
b a a a a a
24 23 22 21 20 19
b a a a a a