IN244 – Strategi Algoritmik

11 – String Matching

Overview

1. Definisi String Matching/ Pattern Matching ?
2. Algoritma Brute Force
3. Algoritma Boyer-Moore
4. Algoritma Knuth-Morris-Pratt

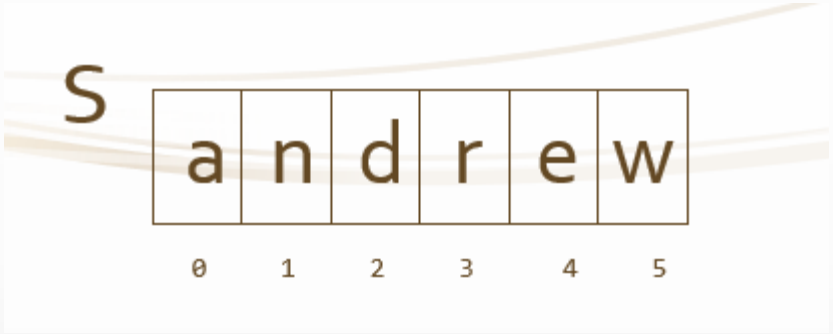
Diadaptasi dari Pattern Matching
Dr. Andrew Davison, WiG Lab (teachers room), CoE, ad@fivedots.coe.psu.ac.th

String Matching?

- Definisi :
  - diberikan string teks T dengan panjang n karakter
  - string pola P dengan panjang m karakter,
  - Temukan lokasi pertama di dalam teks T, dimana pola P ditemukan dalam teks T
  - Contoh:
    - **T: “the rain in spain stays mainly on the plain”**
    - **P: “main”**
- Aplikasi :
  - Pencarian dalam text editors, Web search engines (mis. Google), image analysis

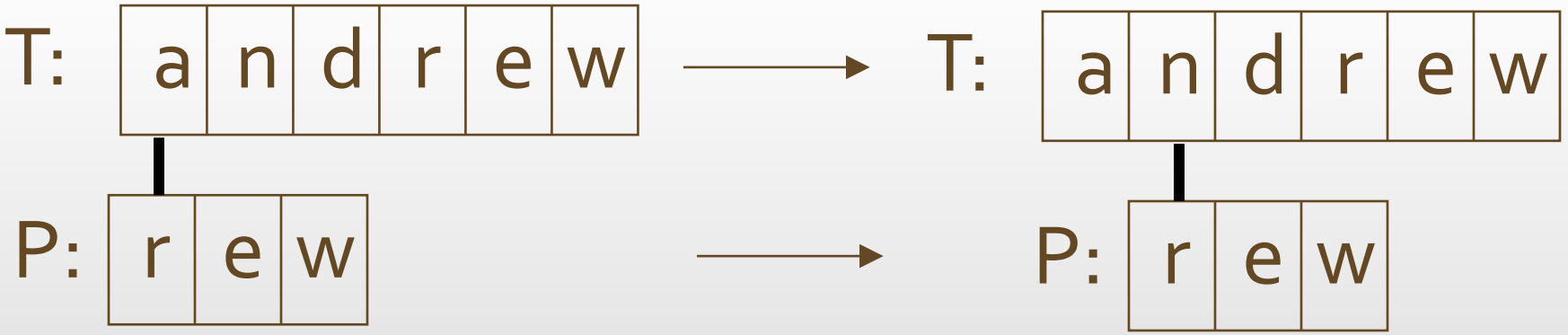
Konsep String

- S : string dengan panjang m.
  - *substring* S[i .. j] : substring di antara index i dan j.
  - *prefix* dari S : substring S[0 .. i]
  - *suffix* dari S : substring S[i .. m-1]
- Contoh string S
  - Substring S[1..3] == "ndr"
  - prefix dari S:
    - "andrew", "andre", "andr", "and", "an", "a"
  - suffix dari S:
    - "andrew", "ndrew", "drew", "rew", "ew", "w"



Algoritma Brute Force

- Periksa setiap posisi dalam text T, mulai dari awal string untuk memastikan pola P ada dalam T dimulai pada posisi ke j



Pemeriksaan maju 1 char untuk mencari P dalam T

Contoh pencocokan dengan Brute Force

- *Pattern*: NOT
- Teks: NOBODY NOTICED HIM
- Proses Pencocokan :

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

- *Pattern*: 001011
- Teks: 10010101**001011**110101010001
- Proses Pencocokan :

```
  10010101001011110101010001
1 001011
2  001011
3   001011
4    001011
5     001011
6      001011
7       001011
8        001011
9         001011
```

Fungsi Search secara Brute Force

```
#Kamus data
#T : text
#P : pola yang akan dicari dalam text T
#jika P ada dalam T, nilai posisi awal P dikirimkan
#jika P tidak ada, fungsi mengirimkan -1
def searchBruteF(T,P):
    n = len(T)
    m = len(P)
    for i in range (0,n-m+1,1):
        j = 0
        while (j < m) and (P[j] == T [i + j]):
            j = j + 1;
         if (j == m):
              return (i) #P ditemukan pada posisi ke i
    return (-1) #P tidak ditemukan
```

Pemanggilan Fungsi String Matching

```
#text : string sumber
#pola : pola yang akan dicari dalam text T
#pos : posisi awal pola dalam text
def main():
    text = str(input("text:"))
    pola = str(input("pola:"))
    pos = searchBruteF(text, pola)
    if (pos >= 0):
        print("Pola ", pola," ditemukan dalam text
        pada posisi ",pos)
    else:
        print("Pola ", pola," tidak ditemukan dalam
        text")
if __name__ == '__main__':
    main()
```

Analisis Brute Force

- Brute force untuk string matching mempunyai O(mn) untuk kasus terburuk
- Pada umumnya search dalam ordinary text mempunyai O(m+n) -> sangat cepat.
- Algoritma brute force berjalan cepat bila alphabet dari text berukuran besar
  - Mis. A..Z, a..z, 1..9, dll
- Algoritma brute force melambat bila alphabet berukuran kecil
  - Mis. 0, 1 (dalam binary files, image files, dll.)

continued

Analisis Brute Force

- Contoh best case:
  - T: “string ini berakhir dengan zzz"
  - P: “zzz”
- Contoh worst case:
  - T: "aaaaaaaaaaaaaaaaaaaaaaaaaah"
  - P: "aaah“
- Contoh average case:
  - T: "a string searching example is standard"
  - P: "store"

Algoritma Boyer-Moore

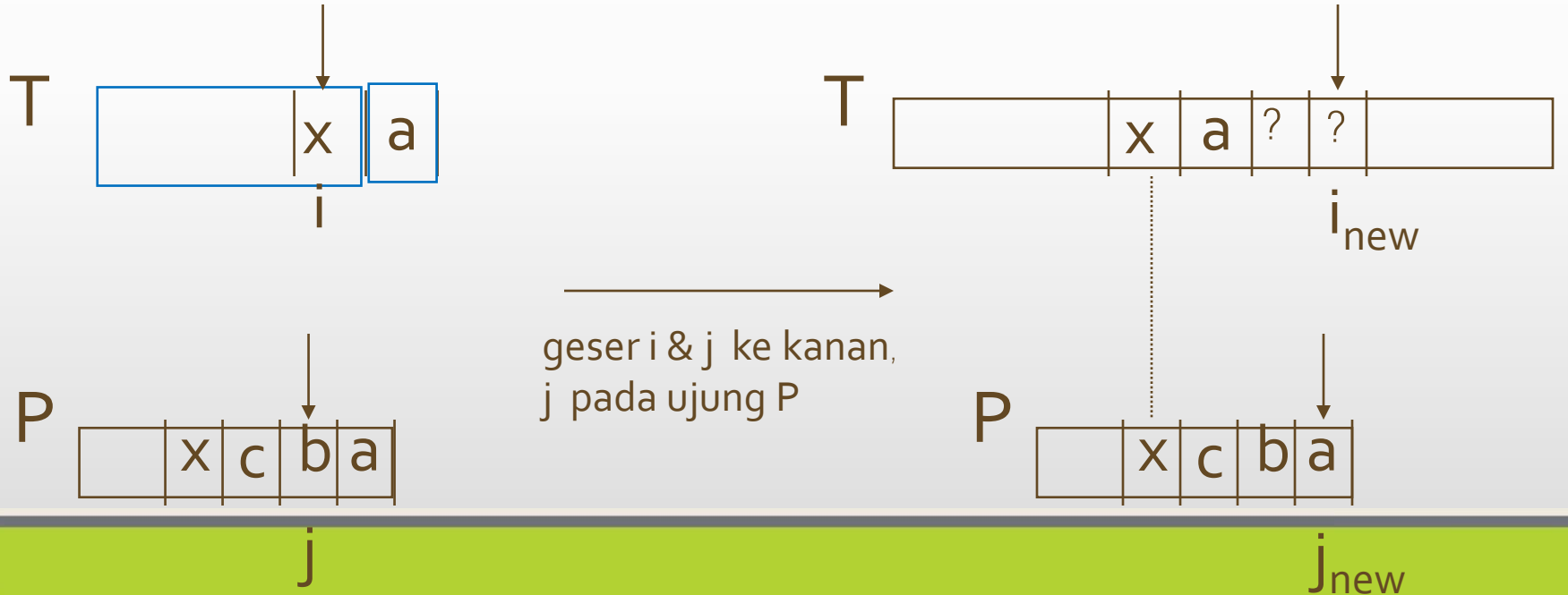
- Didasarkan pada 2 teknik :

1. The *looking-glass* technique
   - Cari P dalam T dengan bergerak mundur dalam P, mulai dari karakter terakhir dalam P
2. The *character-jump* technique
   - Bila ada ketidakcocokan pada T[i] == P[j], dilakukan character jump

- Terdapat 3 kemungkinan kasus

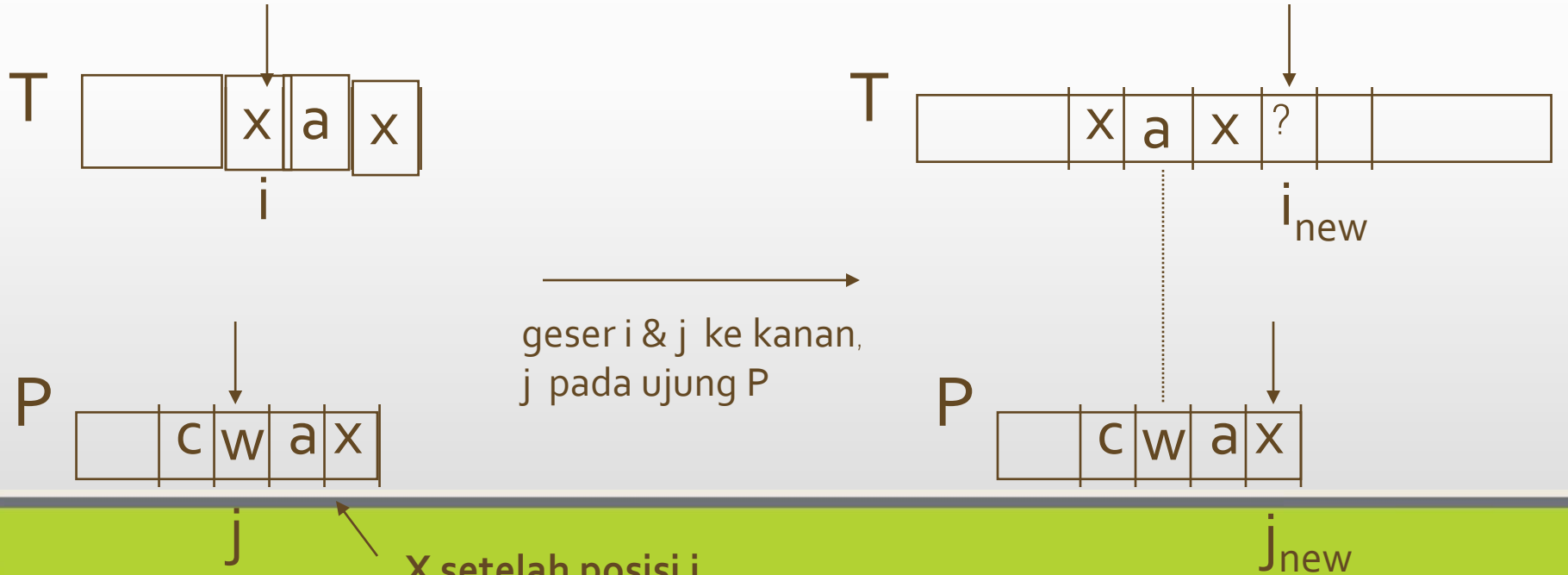
Kasus 1

- Jika P mengandung x , maka geser P ke kanan untuk menempatkan kemunculan x dalam P dengan T[i].



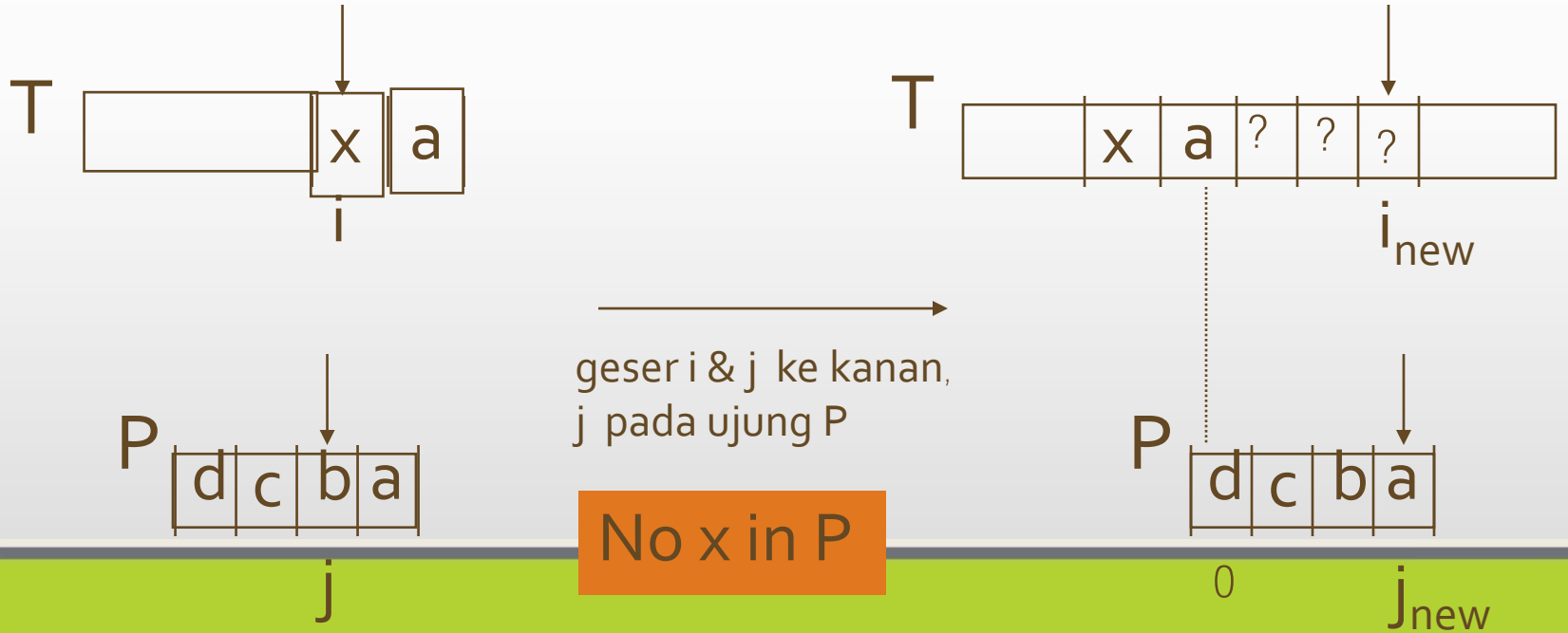
Kasus 2

- Jika P mengandung x, tapi langkah pada kasus 1 tidak mungkin, maka geser P ke kanan 1 karakter pada T[i+1].

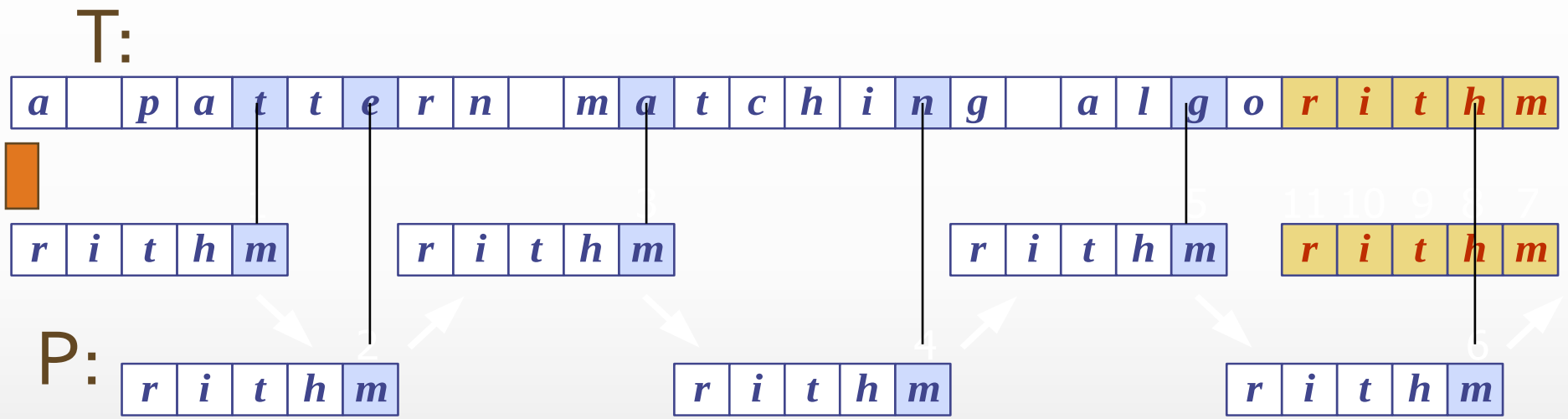


Kasus 3

- Jika bukan kasus 1 dan 2 (x tidak ada dalam P), maka geser P untuk menempatkan **P[0]** dengan **T[i+1].**



Contoh Boyer-Moore



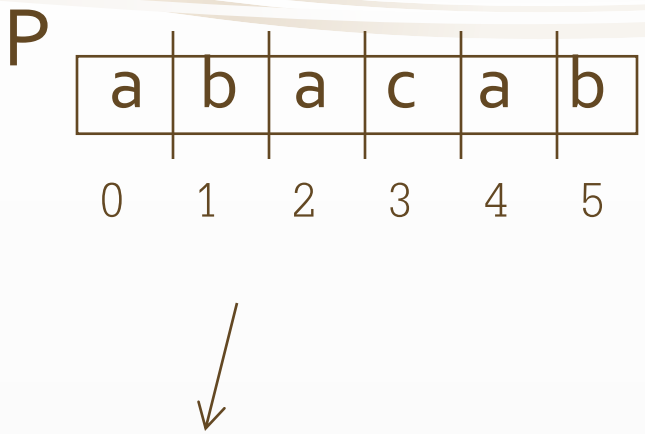
1. Kasus-1
2. Kasus-3
3. Kasus-3
4. Kasus-3
5. Kasus-3
6. Kasus-1

Fungsi Last Occurrence

- Algoritma Boyer-Moore melakukan preprocessing terhadap P dan alfabet A untuk membangun fungsi last occurrence L()
  - L() memetakan semua huruf dalam A menjadi integer
- L(x) didefinisikan sbg:
  - index terbesar i sehingga P[i] == x, atau
  - -1 jika x tidak muncul dalam P
  - x : huruf dalam alfabet A

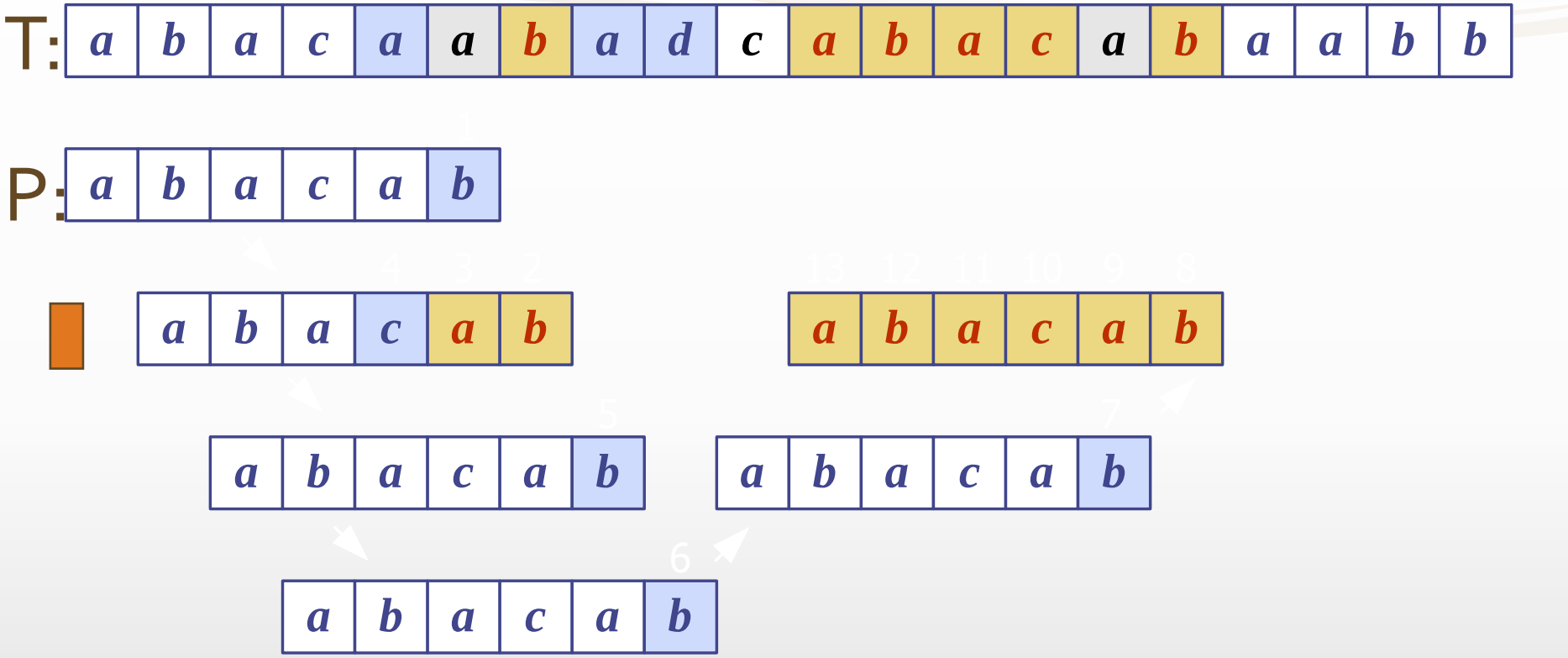
Contoh L()

- A = {a, b, c, d}
- P: "abacab"



x	***a***	***b***	***c***	***d***
***L*(*x*)**	4	5	3	-1

Contoh Boyer-Moore



1. Kasus-1
2. Kasus-1
3. Kasus-1
4. Kasus-3
5. Kasus-1

x	a	b	c	d
L(x)	4	5	3	−1

Boyer-Moore (Python)

```
def searchBM(T,P):
    last = lastO(P)
    n = len(T)
    m = len(P)
    i = m-1
    if (i > n-1):
     #pola > text
        hasil = -1
    else:
        j = m-1
        Found = False
        # while ...
```

```
while (i < n) and (not Found):
  if (P[j] == T[i]):
        if (j == 0):
            Found = True
        else:
           j = j - 1;
           i = i - 1;
  else: #character jump
     lo = last[ord(T[i])]
     i = i + m - min(j,1+lo)
     j = m-1
if (Found):
    hasil = i
else:
    hasil = -1
return (hasil)
```

Fungsi Last Occurence

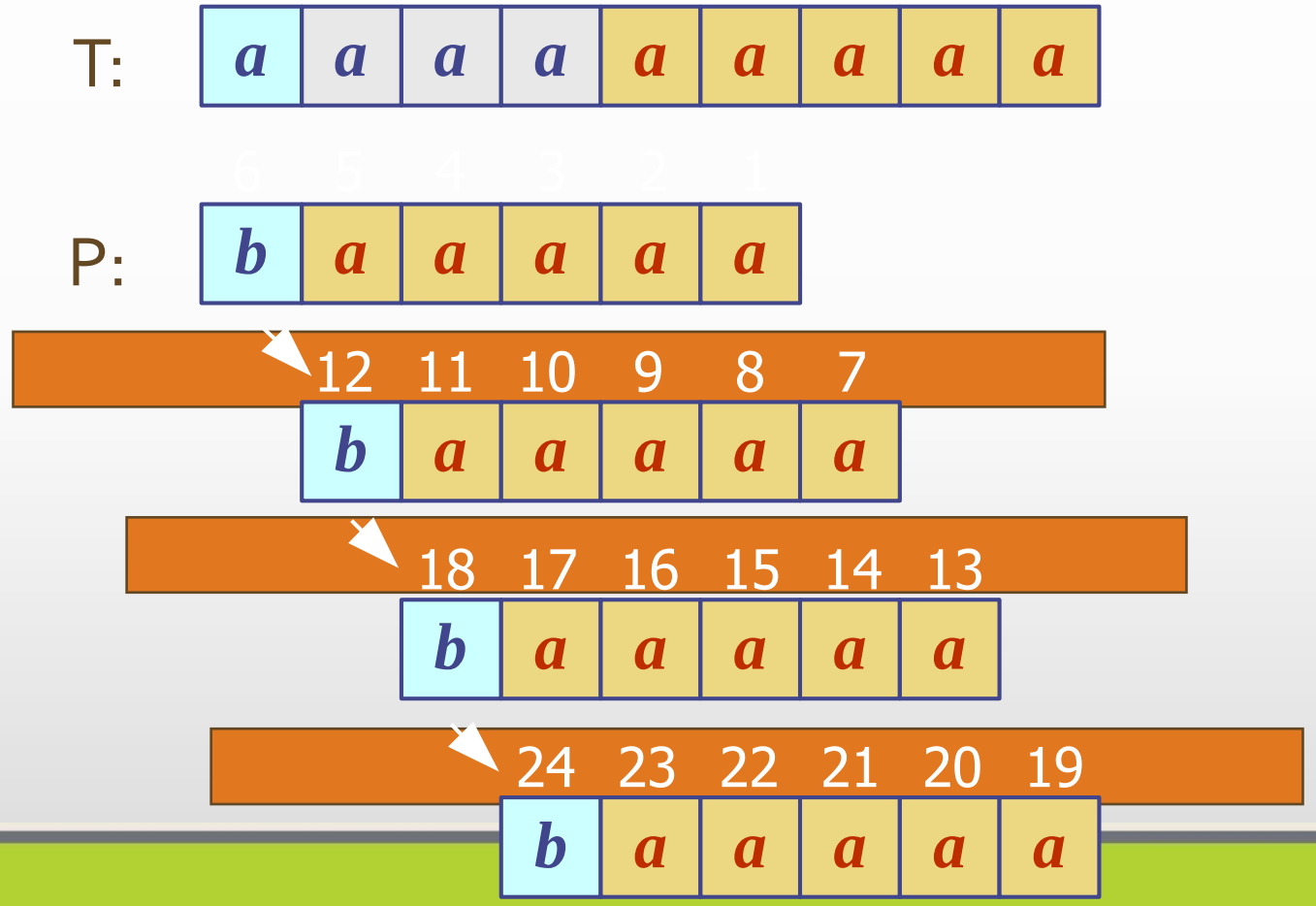
```
# array last menyimpan kemunculan terakhir
# darisetiap karakter dalam pola
# jika karakter tidak muncul diberi nilai -1

def lastO(pola):
    last = [None]*128
    for i in range(0,128,1):
        last[i] = -1
    for i in range(0, len(pola), 1):
        last[ord(pola[i])] = i;
    return last;
```

Analisis Boyer Moore

- Boyer-Moore worst case running time : O(nm + A)
- Boyer-Moore :
  - berjalan dengan cepat jika alfabet (A) berukuran besar,
  - Berjalan lambat jika alfabet (A) berukuran kecil,
  - Mis. Baik untuk English text, kurang baik untuk binary
- Boyer-Moore *lebih cepat dari brute force* untuk searching English text.

Contoh Worst Case untuk Boyer Moore

- T: "aaaaa...a"
- P: "baaaaa"

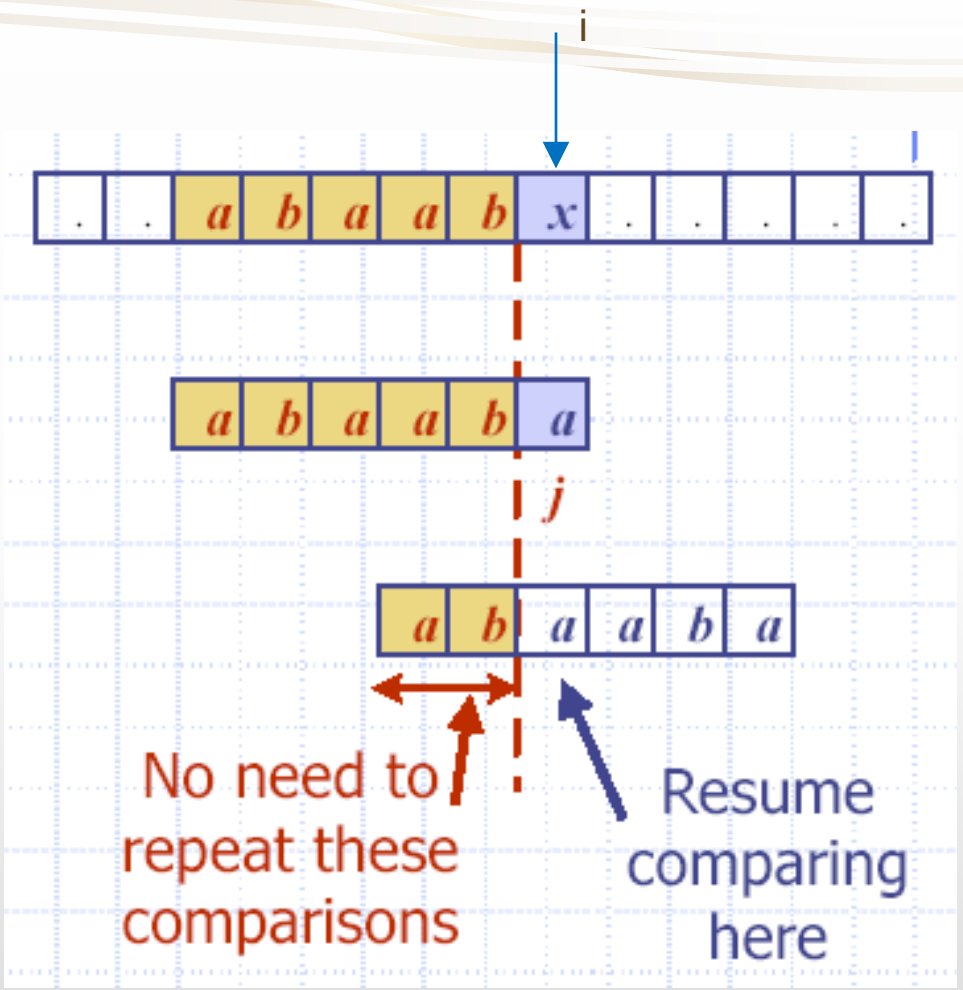


Algoritma KMP

- Algoritma Knuth-Morris-Pratt (KMP) mencari pola dalam teks secara *left-to-right* order
- KMP menggeser pola lebih cerdas dibandingkan algoritma brute force
- Jika ada ketidakcocokan antara text dan pola P pada posisi P[j], bagaimana pergeseran pola utk menghindari pencocokan yang tidak diperlukan?
- *Jawab*:
  - `the largest prefix of P[0 .. j-1] that is a suffix of P[1 .. j-1]`

continued

Contoh



j = 5

j_{new} = 2

Teknik KMP

- Temukan largest prefix (start) of:
  "a b a a b" (**P[0..j-1])**

  yang merupakan suffix (end) of:
  "b a a b" (**p[1 .. j-1])**

j == 5

- Jawaban : "a b"
- Set j = 2 // the new j value
- Pergeseran = len("abaab") – len("ab") = 5 – 2 = 3

Fungsi KMP Failure

- KMP melakukan preprocesing pola untuk mencocokkan prefix dari pola dengan polanya sendiri.
- j = posisi mismatch dalam P[]
- k = posisi sebelum mismatch (k = j-1).
- *failure function* F(k) didefinisikan sbg
  - Ukuran dari largest prefix of P[0..k] yang merupakan suffix of P[1..k].

Contoh Failure Function

POLA

a	b	a	a	b	a
0	1	2	3	4	5

j	0	1	2	3	4
F(j)	0	0	1	1	2

Failure Function

F(j) : ukuran dari the largest prefix.

J=4

a	b	a	a	b
0	1	2	3	4

J=3

a	b	a	a
0	1	2	3

J=2

a	b	a
0	1	2

Mengapa F(4) = 2?

P: "abaaba"

- F(4) berarti
  - Temukan ukuran dari largest prefix P[0..4] yang merupakan suffix P[1..4]

  = temukan ukuran largest prefix “abaab” yang juga suffix dari "baab"

  = ukuran dari "ab"

  = **2**

Pemakaian Failure Function

- Modifikasi algoritma Knuth-Morris-Pratt's :
  - if a mismatch occurs at P[j] (P[j] != T[i]),
  - then

    k = j-1;

    j = F(k);   // obtain the new j

Fungsi Search KMP

```
def searchKMP(T,P):
    fail = cfail(P)
    n = len(T)
    m = len(P)
    i = 0
    j = 0

    Found = False
    #while ……
```

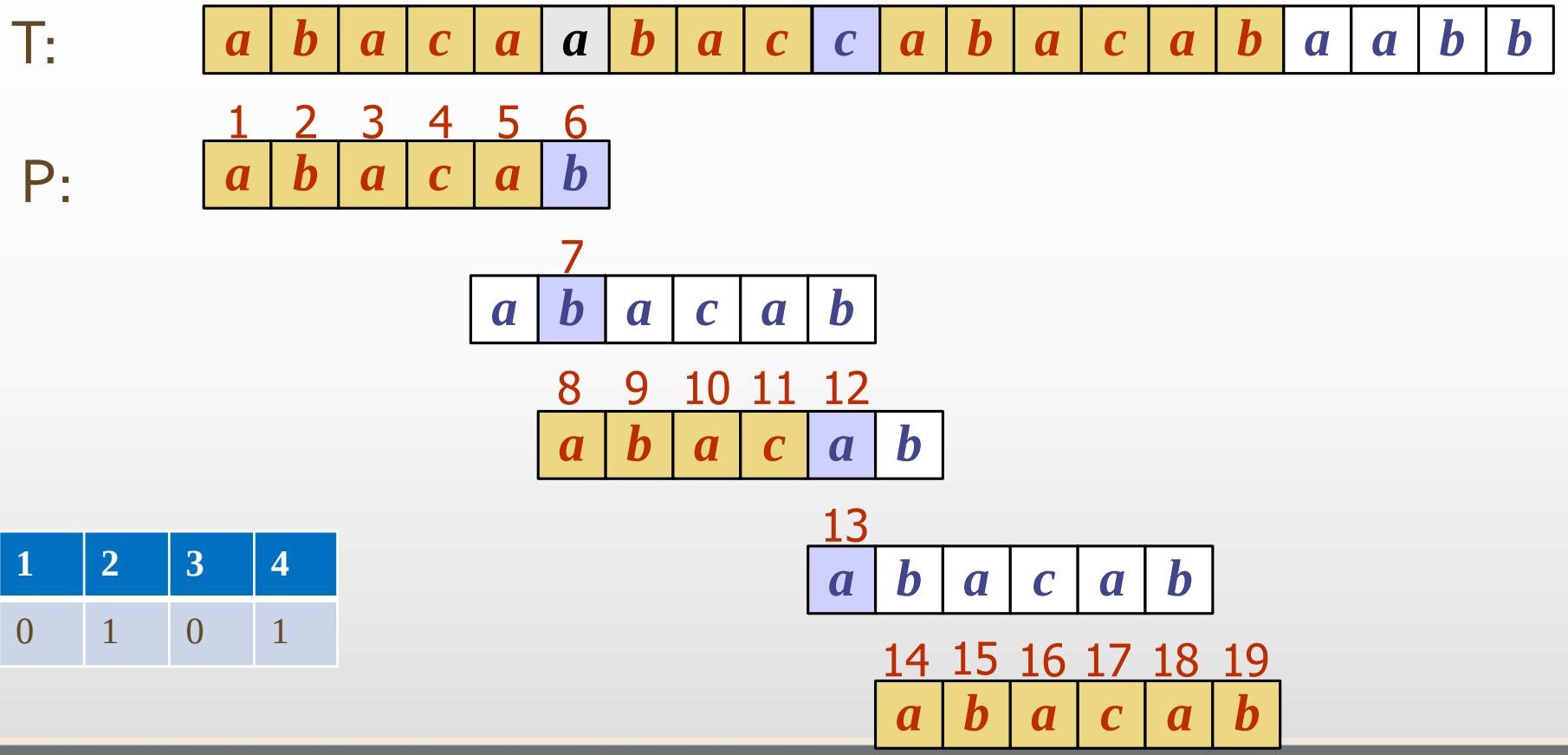
```
while (i < n) and (not Found):
    if (P[j] == T[i]):
        if (j == m-1):
            Found = True
            hasil = i-m+1
        else:
            j = j + 1;
            i = i + 1;
    elif (j > 0):
        j = fail[j-1]
    else:
        i = i + 1

if (not Found):
    hasil = -1
return (hasil)
```

Failure Function

```
def cfail(pola):

   fail = [None] * len(pola)

   fail[0]=0

   m = len(pola)

   j = 0

   i = 1

   #while . . .
```

```
while (i < m):
   if  (pola[j] == pola[i]) :
        #j+1 chars match
        fail[i] = j + 1;
        i= i+1;
        j= j+1;
   elif (j > 0):
   # j follows matching prefix
         j = fail[j-1];
   else:  # no match
         fail[i] = 0;
         i= i+1;
return fail
```

Contoh KMP



k	0	1	2	3	4
F(k)	0	0	1	0	1

Keuntungan KMP

- KMP mempunyai waktu optimal : O(m+n) => sangat cepat
- Algoritma tidak perlu mundur dalam memeriksa teks input T
  - Algoritma KMP baik untuk pemrosesan files yg sangat besar yg berasal dari external devices atau dikirim melalui network

Kerugian KMP

- KMP bekerja kurang baik jika ukuran alfabet meningkat
  - Ketidakcocokan akan meningkat
  - mismatch cenderung terjadi di awal pola, sedangkan KMP berjalan cepat jika mismatch terjadi di akhir pola

LATIHAN

- Diberikan P dan T sbb :
  - P = badbaaa
  - T = badbacabcbadbaaacb
- Tunjukkan pemakaian algoritma KMP dan algoritma BM untuk pencarian pola P dalam teks T di atas.

Referensi

- Anany Levitin, Introduction to The Design and Analysis of Algorithms, Pearson, 2012.
- S.Dasgupta, et al. Algorithms, 2006.
- Cormen, et al. Algorithms, MGH,2009.
- Strategi Algoritma – Rinaldi Munir (ITB) :
  - https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/stmik.htm
- Khan Academy, Computer Science: Algorithm, https://www.khanacademy.org/computing/computer-science/algorithms