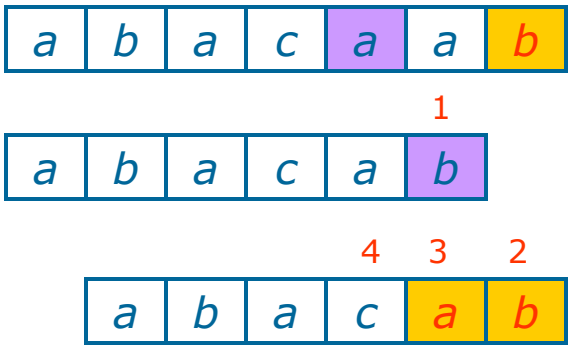
C Programming Basic – week 13

Nội dung

- Các giải thuật so khớp chuỗi
  - Giải thuật ngây thơ
  - Giải thuật Knuth-Morris-Pratt
  - Giải thuật Boyer-Moore
- Bài tập



Bài toán so khớp chuỗi

- Có P là chuỗi có kích thước m
  - Chuỗi con P[i .. j] của P bao gồm các kí tự từ vị trí i tới vị trí j
  - Tiền tố của P là chuỗi con có dạng P[0 .. i]
  - Hậu tố của P là chuỗi con có dạng P[i ..m – 1]
- Cho các chuỗi T (text) và P (pattern), bài toán so khớp chuỗi yêu cầu tìm chuỗi con của T bằng với P
- Các ứng dụng:
  - Soạn thảo văn bản, máy tìm kiếm, nghiên cứu sinh học

Brute Force Matching

- Giải thuật brute-force matching so sánh P với T với mỗi khả năng của T cho tới khi
  - tìm thấy một chuỗi con hoặc
  - đã thử tất cả các khả năng
- Độ phức tạp: O(nm)
- Trường hợp tệ nhất:
  - T = aaa … ah
  - P = aaah
  - có thể gặp trong ảnh hoặc chuỗi DNA
  - ít gặp trong văn bản tiếng Anh

Giải thuật

Algorithm BruteForceMatch(T, P)

```
// Input text T of size n and pattern P of size m
// Output starting index of a substring of T equal to P or
   -1
if no such substring exists
   for  i ← 0 to n − m {
      test shift i of the pattern
   }
   j ← 0
   while j < m ∧ T[i + j] = P[j]
      j ← j + 1
   if  j = m
      return  i {match at i}
   else
   break while loop {mismatch}
return  -1 {no match anywhere}
```

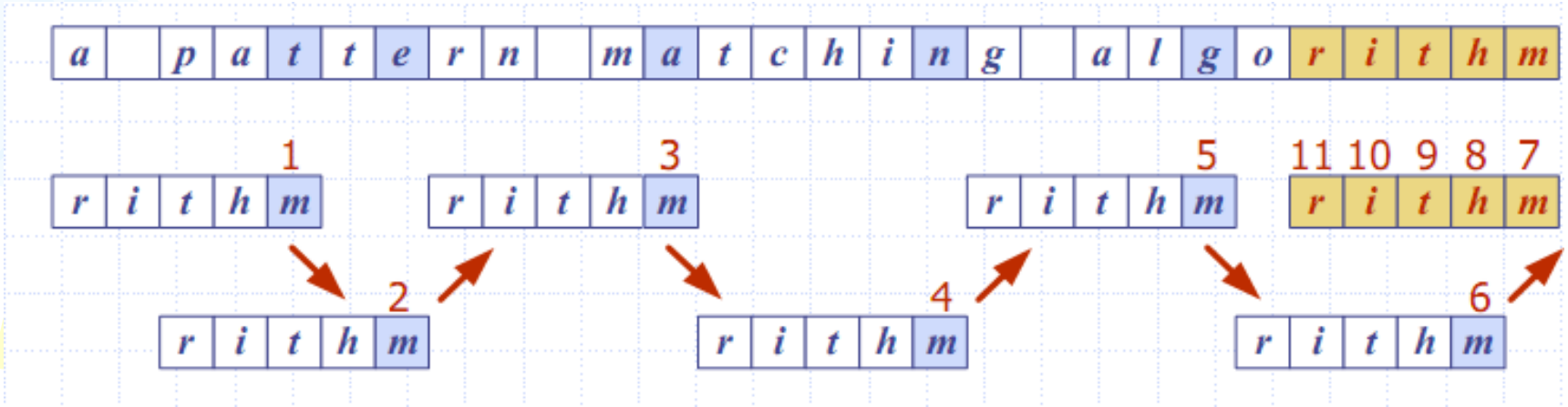
Exercise 13.1

- Tạo ra một chuỗi chứa 2000 kí tự ngẫu nhiên nằm trong một tập kí tự
- VD:
  - tập kí tự: abcdef
  - chuỗi: abcadacaeeeffaadbfacddedcedfbeccae…
- Viết chương trình tìm mẫu, vd "aadbf", từ chuỗi.
- Chú ý: sử dụng giải thuật so khớp đơn giản

Giải thuật Boyer-Moore

- Giải thuật Boyer-Moore bao gồm hai bước
- Looking-glass: So sánh P với một chuỗi con của T
- Quay trở lại
- Character-jump: Nếu không khớp tại vị trí T[i] = c
  - If P chứa c, dịch P để gióng hàng vị trí cuối cùng của c trong P với T[i]
  - Else, dịch P để gióng hàng P[0] với T[i + 1]

VD


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
1
r i t h m
2
r i t h m
3
r i t h m
4
r i t h m
5
r i t h m
6
r i t h m
11 10 9 8 7
r i t h m

Hàm last-occurrence

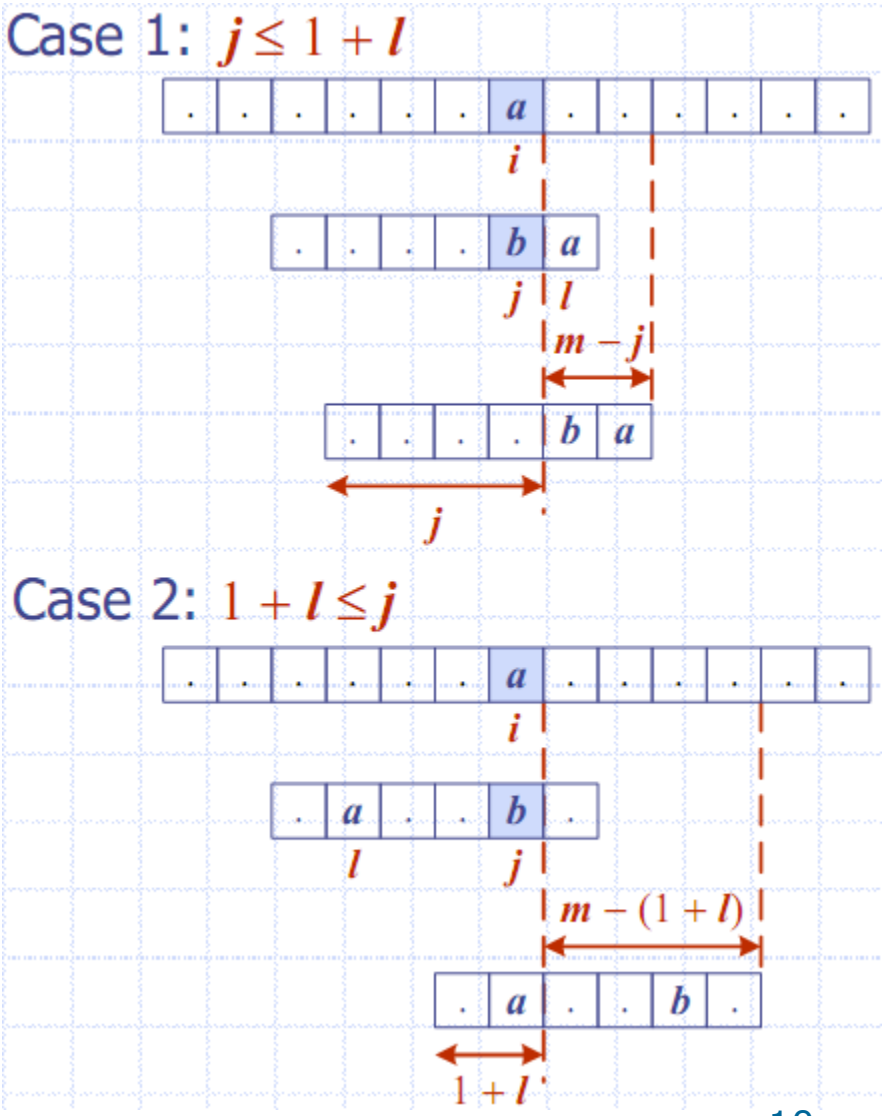
- Giải thuật Boyer-Moore tiền xử lý mẫu P và tự vựng Σ để xây dựng hàm last-occurrence L tham chiếu Σ tới các số nguyên, với L(c) được định nghĩa như sau
  - chỉ số i lớn nhất sao cho P[i] = c hoặc
  - −1 nếu không tồn tại chỉ số như vậy
- VD:
  - Σ = {a, b, c, d}
  - P = abacab

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

- Hàm last-occurrence có thể được biểu diễn bởi một mảng đánh chỉ số bởi mã số học của các kí tự
- Độ phức tạp: O(m + s), trong đó m là kích thước của P và s là kích thước của Σ

Giải thuật Boyer Moore

```
Algorithm BoyerMooreMatch(T, P, Σ)
    L ← lastOccurenceFunction(P, Σ )
    i ← m − 1
    j ← m − 1
    repeat
    if T[i] = P[j]
          if j = 0
          return i { match at i }
          else
          i ← i − 1
          j ← j − 1
    else
    { character-jump }
          l ← L[T[i]]
          i ← i + m – min(j, 1 + l)
          j ← m − 1
    until i > n − 1
    return −1 { no match }
```



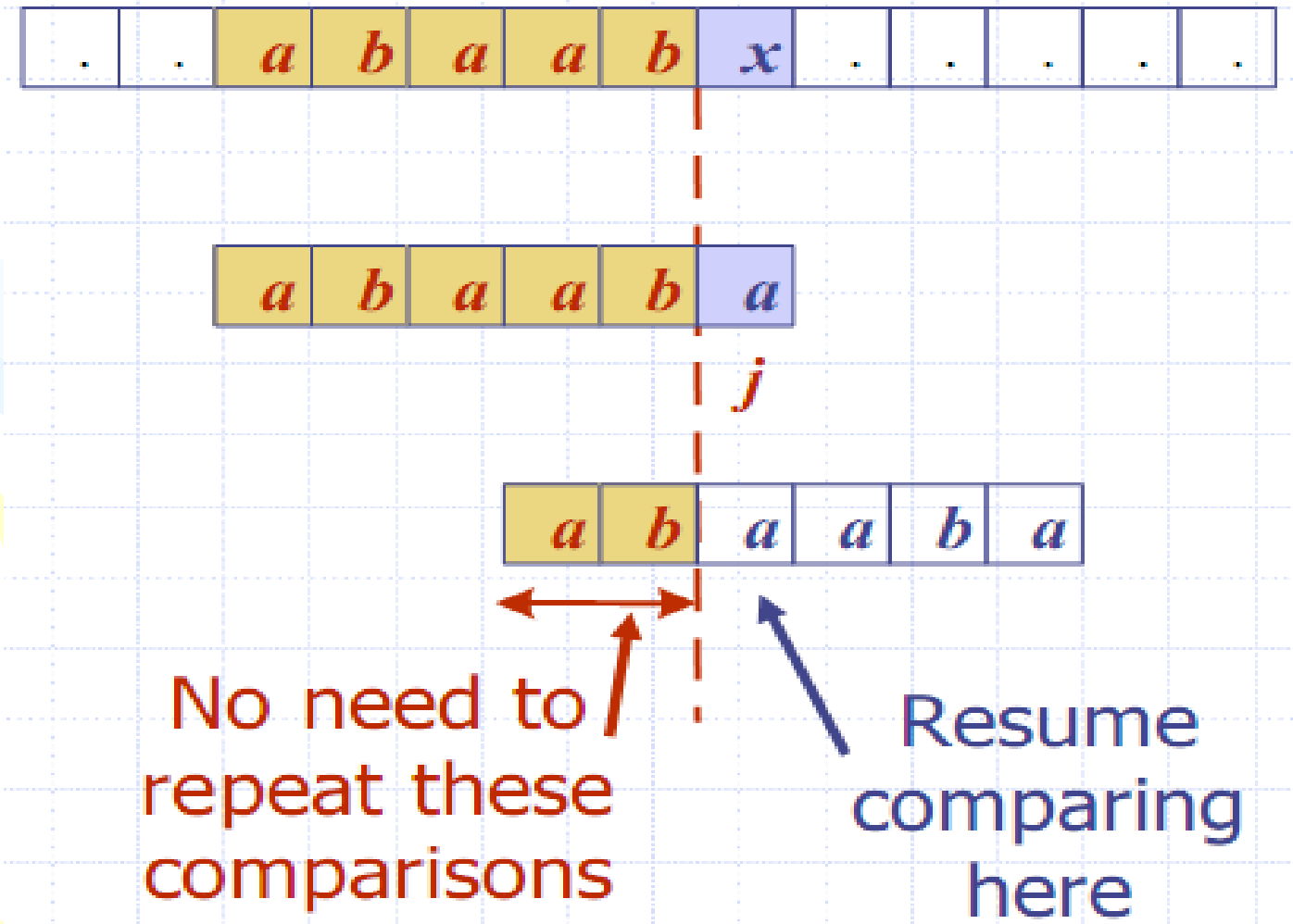
Exercise 13.2

- Tạo ra chuỗi có 2000 kí tự được tạo ngẫu nhiên từ một tập các kí tự
- VD với tập kí tự: abcdef
- Chuỗi: abcadacaeeeffaadbfacddedcedfbeccae…
- Viết chương trình tìm một mẫu, vd “aadbf”, trong chuỗi.
- Chú ý: Sử dụng giải thuật Boyer-Moore

Giải thuật KMP

- Giải thuật Knuth-Morris-Pratt so sánh mẫu với chuỗi từ trái-qua-phải, nhưng dịch chuyển mẫu thông minh hơn brute-force matching.
- Khi xuất hiện không khớp, có thể dịch mẫu tối đa bao nhiêu để tránh các so sánh trùng lặp?
- Trả lời: tiền tố dài nhất P[0..j] mà là hậu tố của P[1..j]

VD


. . a b a a b x . . . . .
a b a a b a
j
a b a a b a
No need to repeat these comparisons
Resume comparing here

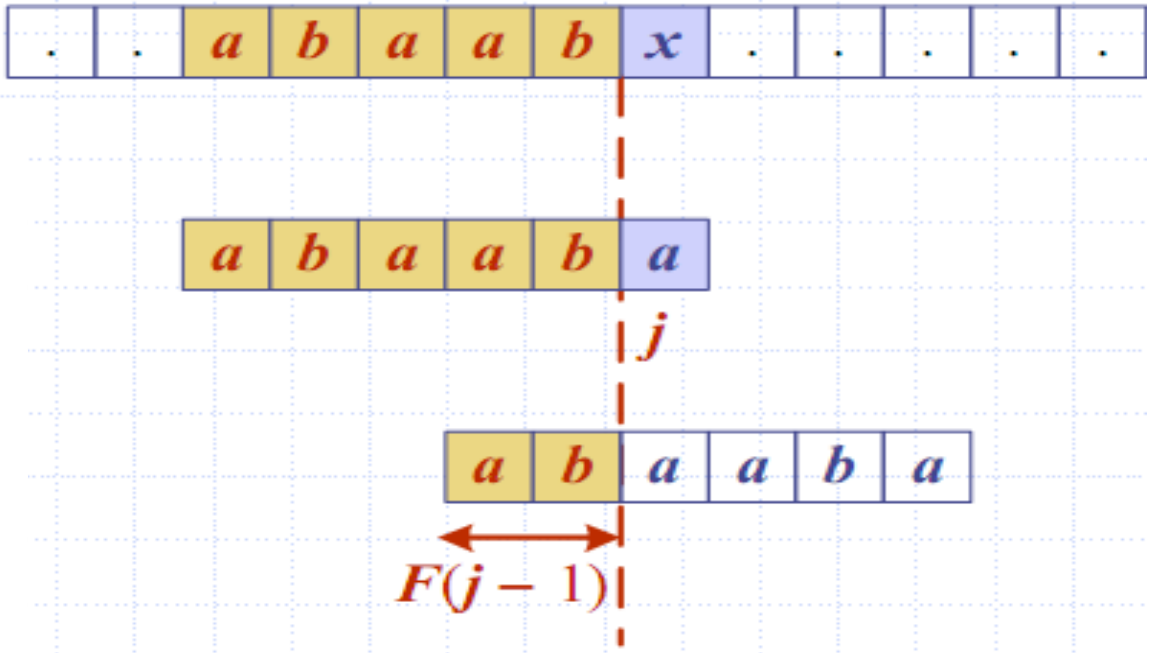
Hàm lỗi KMP

- Thuật toán Knuth-Morris-Pratt tiền xử lý mẫu để tìm so khớp của tiền tố của mẫu với chính nó
- Hàm lỗi F(j) được định nghĩa là kích thước của tiền tố dài nhất P[0..j] đồng thời là hậu tố của P[1..j]
- Thuật toán Knuth-Morris-Pratt cải thiện brute-force matching sao cho nếu một không khớp xuất hiện tại P[j] ≠ T[i] ta thiết lập

$j \leftarrow F(j - 1)$

VD

j	0	1	2	3	4	5
$P[j]$	a	b	a	a	b	a
$F(j)$	0	0	1	1	2	3



```
Algorithm failureFunction(P)
    F[0] ← 0
    i ← 1
    j ← 0
    while i < m
        if P[i] = P[j]
        {we have matched j + 1 chars}
            F[i] ← j + 1
            i ← i + 1
            j ← j + 1
        else if  j > 0 then
        {use failure function to shift P}
            j ← F[j − 1]
        else
            F[i] ← 0 { no match }
            i ← i + 1
```

Exercise 13.3

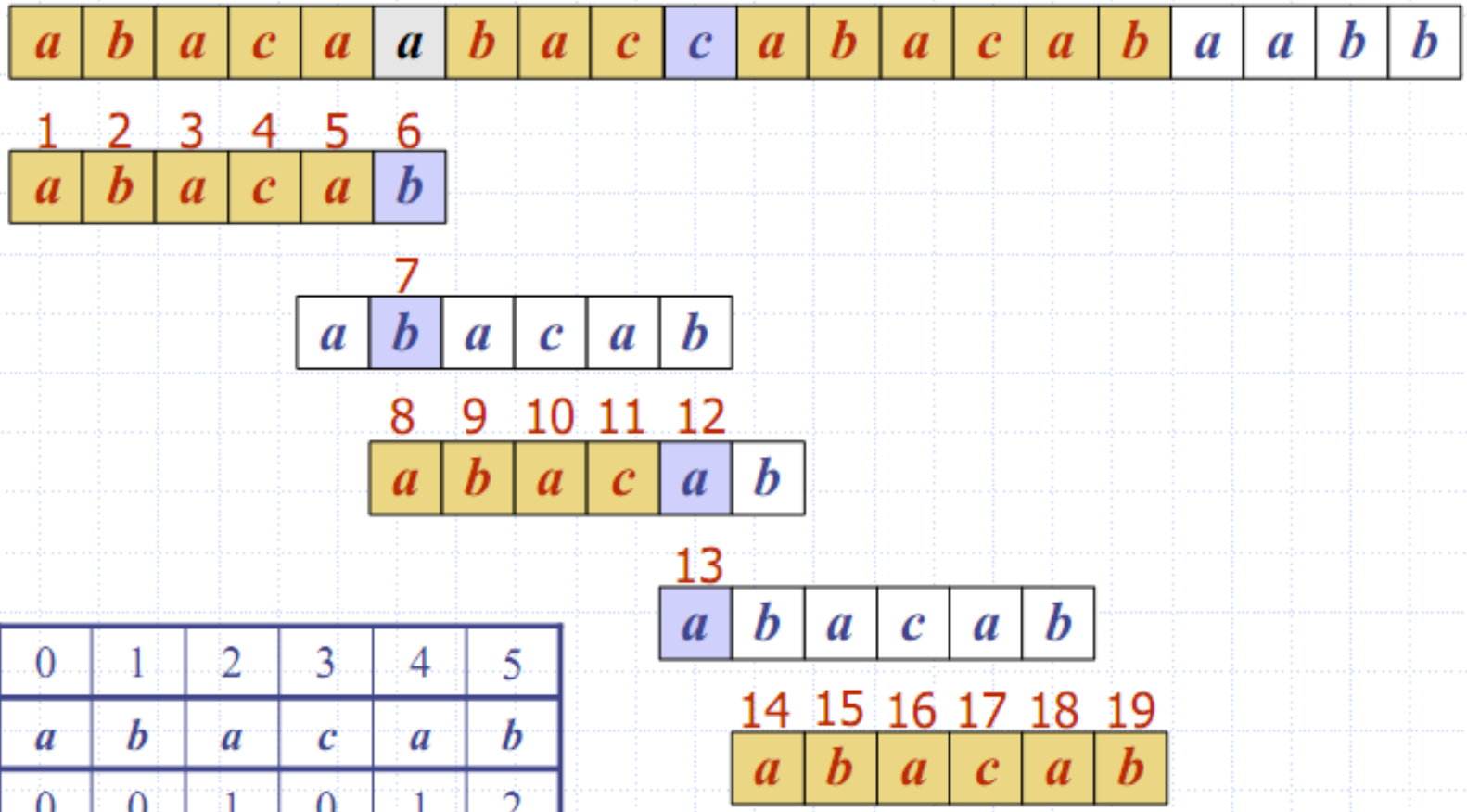
- Thực hiện Exercise 13.2 với giải thuật KMP
- Tính số phép so sánh

Giải thuật KMP

- Hàm lỗi KMP có thể được biểu diễn bởi một mảng và có độ phức tạp O(m)
- Tại mỗi bước lặp
  - tăng i lên 1, hoặc
  - bước dịch chuyển i – j tăng lên ít nhất 1 (quan sát thấy F(j – 1) < j)
- Vì vậy, có tối đa 2n bước lặp
- Vì vậy, thuật toán KMP thực thi trong thời gian tối ưu O(m + n)

```
Algorithm KMPMatch(T, P)
    F ← failureFunction(P)
    i ← 0
    j ← 0
    while i < n
        if T[i] = P[j]
            if  j = m − 1
                return  i − j { match }
            else
                i ← i + 1
                j ← j + 1
        else
            if  j > 0
                j ← F[j − 1]
            else
                i ← i + 1
    return  −1 { no match }
```

VD



j	0	1	2	3	4	5
$P[j]$	a	b	a	c	a	b
$F(j)$	0	0	1	0	1	2