C Programming Basic – week 13

So khớp xâu

Lecturer :

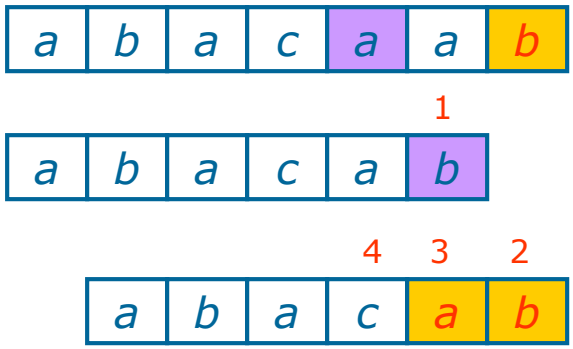
Do Quoc Huy

Dept of Computer Science

Hanoi University of Technology

Topics of this week

- Các thuật toán so khớp xâu
  - Naive algorithm
  - Knuth-Morris-Pratt algorithm
  - Boyer-Moore algorithm
- Exercises



Bài toán so khớp xâu

- P là xâu kí tự kích thước m
  - 1 xâu con P[i .. j] của P là chuỗi các kí tự với nằm giữa **i** và **j**
  - 1 tiến tố của P là: 1 xâu con có kiểu P[0 .. i]
  - 1 hậu tố của P là 1 xâu con có kiểu P[i ..m – 1]

- Cho các xâu T (text) và P (pattern), bài toán so khớp mẫu làviệc tìm một xâu con của T trùng với P

- Ứng dụng:
  - Soạn thảo văn bản, Search engines, nghiên cứu sinh học

Đối sánh mẫu vét cạn

- Thuật toán đối sánh mẫu vét cạn (brute-force) so sánh mẫu P với văn bản T với từng vị trí dịch chuyển của P và đối sánh với T, cho đến khi tìm thấy hoặc tất cả các vị trí của mẫu đều đã được thử
- Đối sánh mẫu Brute-force chạy trong O(nm)
- Ví dụ trường hợp tệ nhất:
  - T = aaa … ah
  - P = aaah
  - Có thể xảy ra trong hình ảnh (images) hay chuỗi DNA
  - Hoặc trong văn bản tiếng Anh

Thuật toán

Algorithm BruteForceMatch(T, P)

```
// Đầu vào: văn bản T có k/thước n và mẫu P có k/thước m
// Đầu ra: chỉ số bắt đầu của 1 xâu con của T trùng với P
// hoặc trả về -1
if không tồn tại xâu con như vậy
   for  i ← 0 to n – m {
        //kiểm tra dịch chuyển i của mẫu

        j ← 0
        while j < m ∧ T[i + j] = P[j]
              j ← j + 1
        if  j = m
              return  i {khớp tại vị trí i}
        else
              break while loop {không khớp}
   }
return  -1 {không khớp ở đâu cả}
```

Exercise 13.1

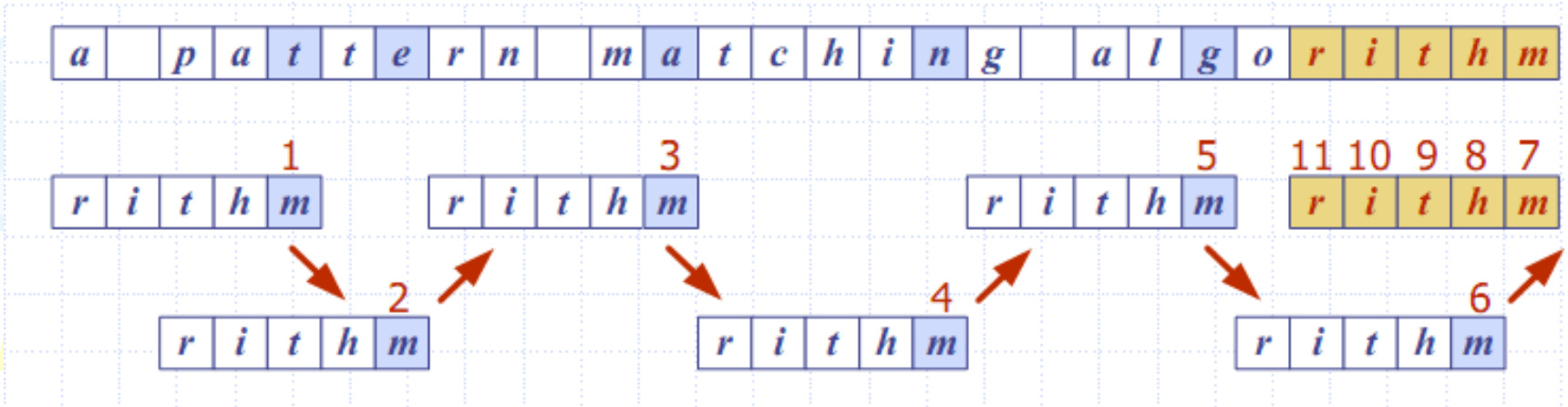
- Tạo một xâu ngẫu nhiên gồm 2000 kí tự.
- Ví dụ:
  - Tập các kí tự: abcdef
  - xâu: abcadacaeeeffaadbfacddedcedfbeccae…
- Viết chương trình tìm kiếm mẫu, ví dụ “aadbf”, từ xâu.
- Chú ý: Sử dụng giải thuật tìm kiếm giản đơn

Boyer-Moore Heuristics

- Thuật toán đối sánh mẫu Boyer-Moore's được dựa trên 2 hàm heuristic
- Looking-glass heuristic: So sánh P với một tập con của T
- Trở ngược về trước
- Character-jump heuristic: Khi có sự không khớp xảy ra ở vị trí T[i] = c
  - Nếu P chứa c, dịch P để căn lề với lần xuất hiện cuối cùng của c trong P với T[i]
  - Nếu không, dịch chuyển P để sắp xếp P[0] với T[i + 1]

Example

Khi có sự không khớp xảy ra ở vị trí T[i] = c
- Nếu P chứa c, dịch P để căn lề với lần xuất hiện cuối cùng của c trong P với T[i]
- Nếu không, dịch chuyển P để sắp xếp P[0] với T[i + 1]



Hàm Last-Occurrence

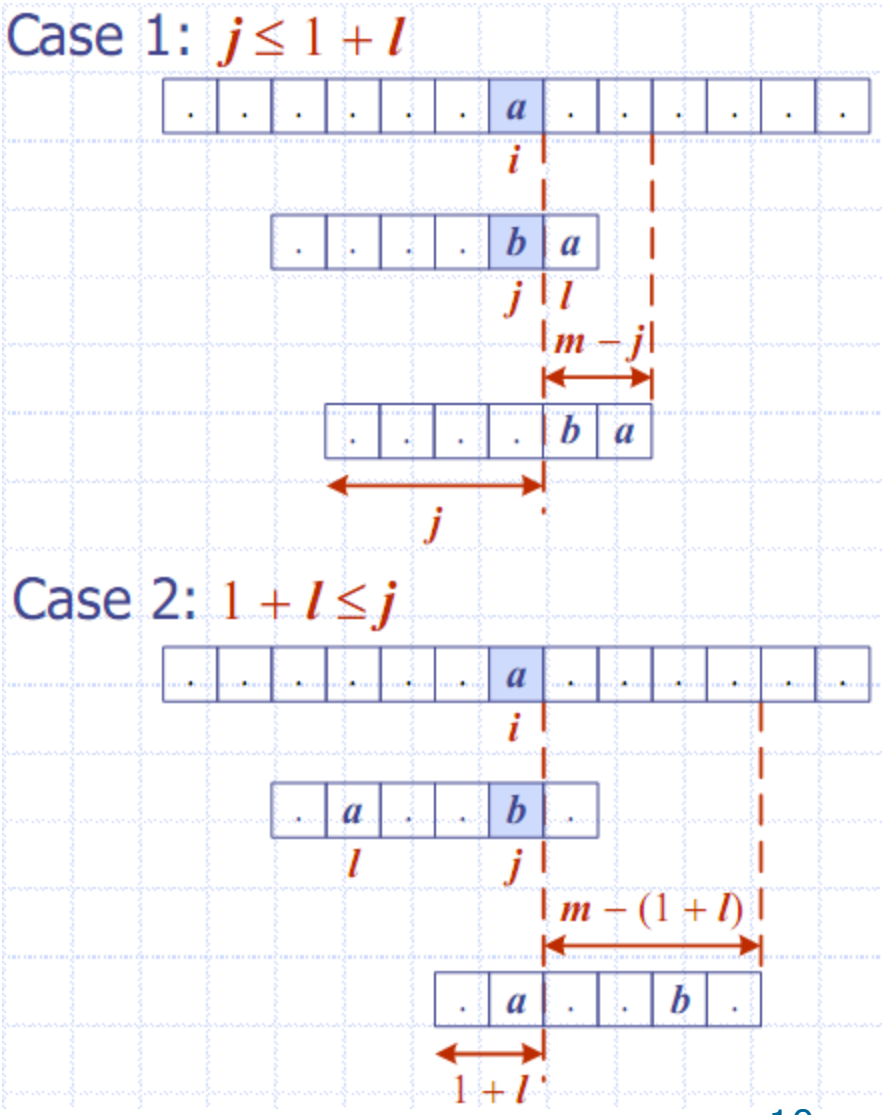
- Thuật toán Boyer-Moore's tiền xử lý mẫu P và tập ký tự alphabet Σ để xây dựng hàm xuất-hiện-sau-cùng L ánh xạ Σ thành số nguyên, với L(c) được định nghĩa như sau
  - Chỉ số lớn nhất i để *P[i] = c* hoặc
  - −1 nếu không có chỉ số nào thoả mãn điều trên
- Ví dụ:
  - Σ = {a, b, c, d}
  - P = abacab

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

- Hàm last-occurrence có thể được biểu diễn bởi một mảng, các phần tử trong mảng được đánh chỉ số bằng mã của các ký tự
- Hàm last-occurrence có thể được tính trong khoảng thời gian O(m + s), với m là **k/thước** của P và s là k/thước của Σ

Algorithm 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 }
```



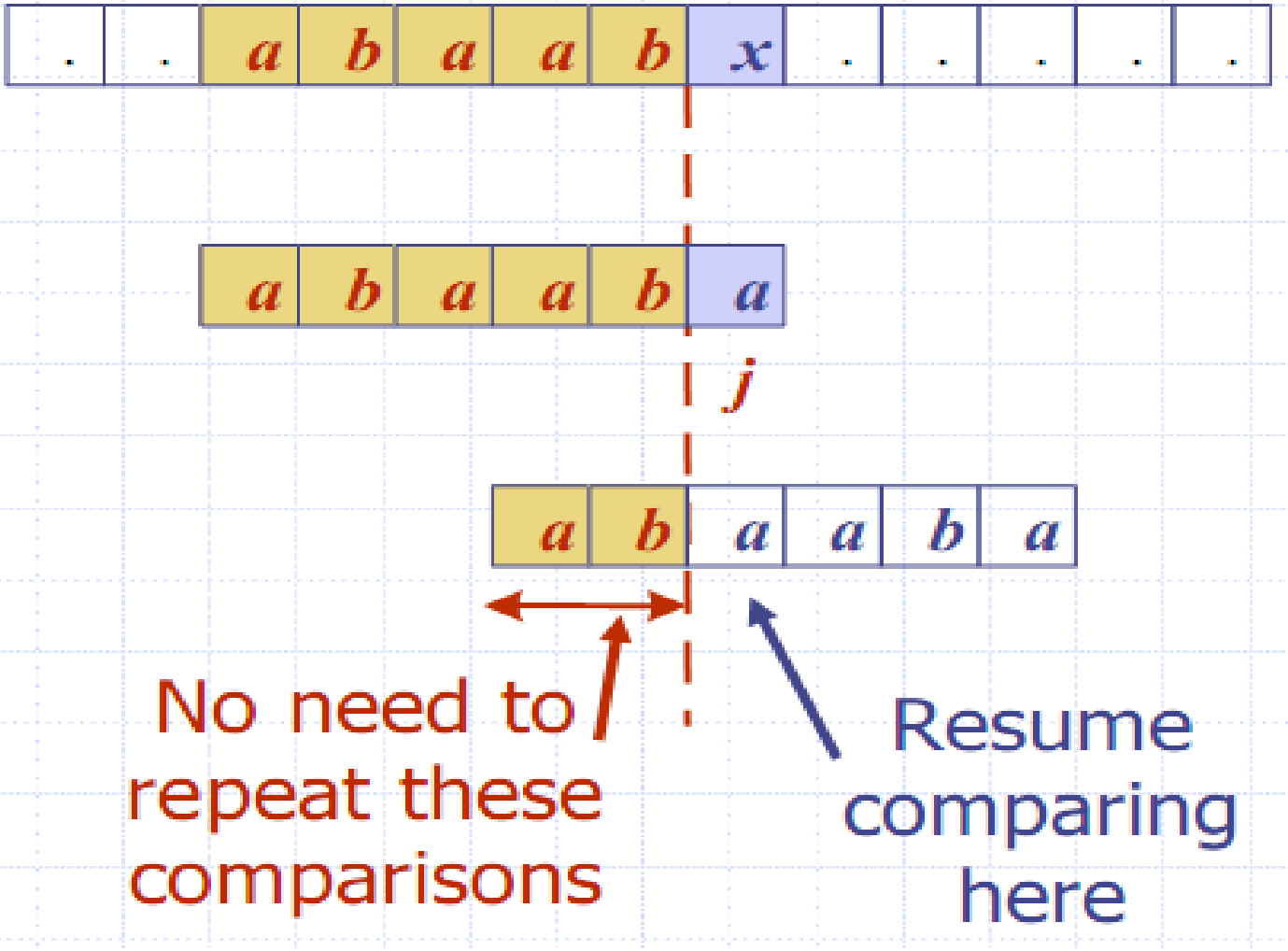
Exercise 13.2: Searching string by Boyer-Moore

- Giống bài 13.1
- Note: dùng thuật toán Boyer-Moore

KMP string matching

- Thuật toán Knuth-Morris-Pratt's so sánh mẫu với văn bản theo trật tự trái-sang-phải, nhưng dịch chuyển mẫu thông minh hơn thuật toán brute-force.
- Khi gặp sự không trùng khớp(mismatch), chúng ta có thể dịch chuyển mẫu để tránh so sánh giản lược?
- Trả lời: tiền tố lớn nhất của P[0..j] là hậu tố của P[1..j]

Example


. . 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

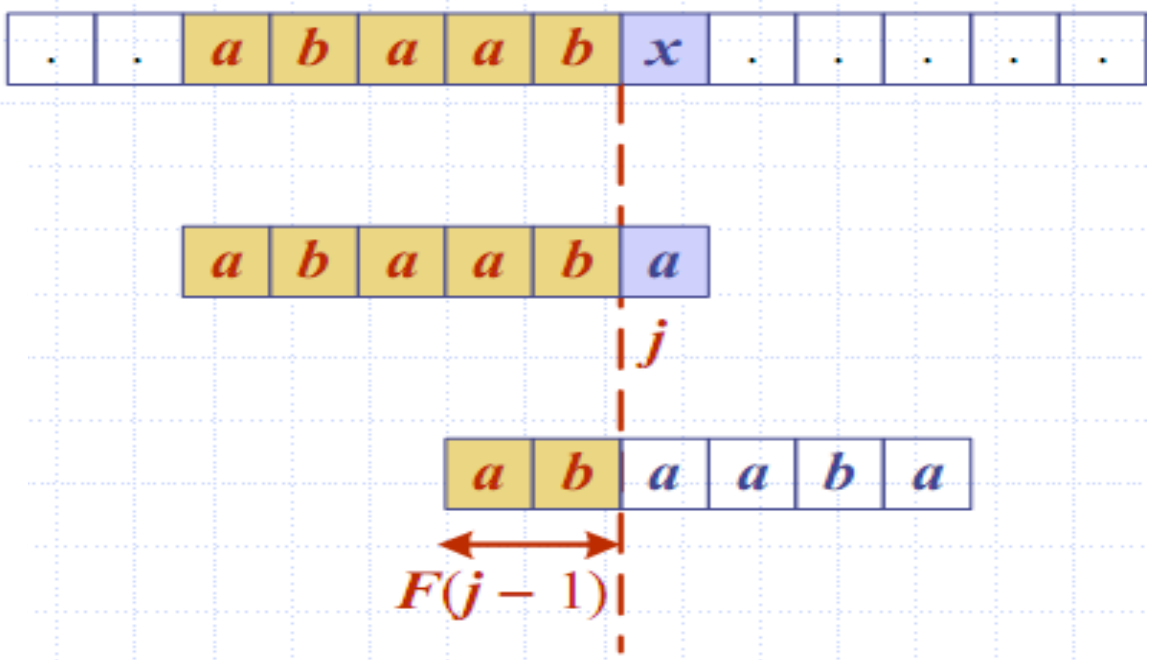
KMP Failure Function

- Thuật toán Knuth-Morris-Pratt's tiền xử lý mẫu để tìm các trường hợp tiền tố khớp với bản thân mẫu
- Hàm failure F(j) được định nghĩa như là k/thước của tiền tố lớn nhất của P[0..j] mà cũng là hậu tố của P[1..j]
- Knuth-Morris-Pratt's sửa đổi thuật toán vét cạn: nếu một sự sai khớp(mismatch) xảy ra ở vị trí P[j] ≠ T[i] thì ta gán

$j \leftarrow F(j-1)$

Example

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

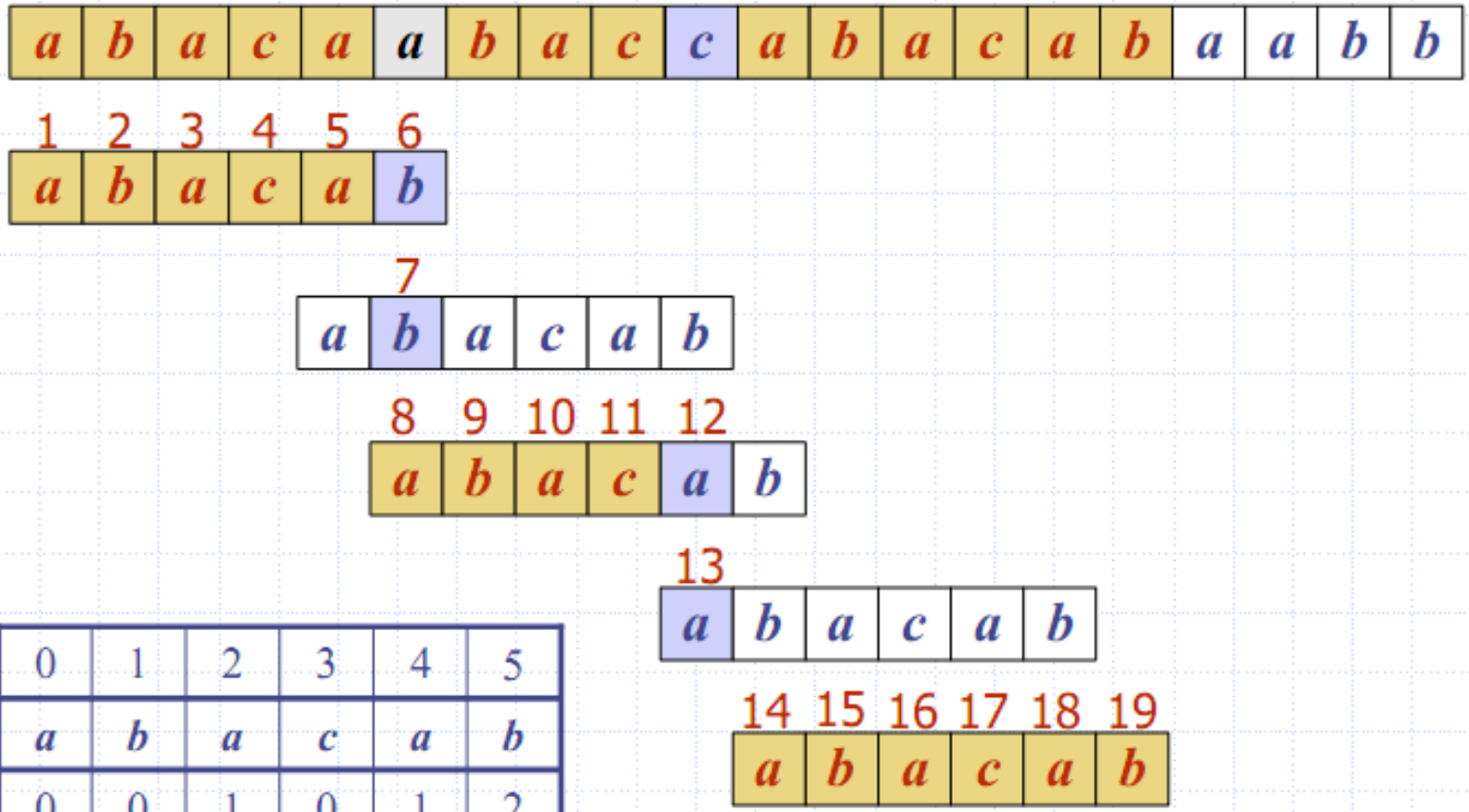
- Làm bài 13.2 sử dụng thuật toán KMP.
- Tính số lượng phép so sánh.

The KMP algorithm

- Hàm failure có thể được biểu diễn bằng một mảng và có thể được tính trong khoảng thời gian O(m)
- Tại mỗi vòng lặp của while-loop,
  - i sẽ tăng lên 1, hoặc
  - Dịch chuyển khoảng i – j tăng lên ít nhất 1 (dễ nhận thấy F(j – 1) < j)
- Do đó, sẽ có không nhiều hơn 2n vòng lặp của while-loop
- Vì vậy, thuật toán KMP's chạy trong khoảng 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 }
```

Example



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