# Cấu trúc dữ liệu và giải thuật

# ĐỐI SÁNH CHUỖI

Văn Chí Nam – Nguyễn Thị Hồng Nhung – Đặng Nguyễn Đức Tiến – Vũ Thanh Hưng

# Nội dung trình bày



Giới thiệu

Thuật toán Brute-Force

Thuật toán Morris-Pratt

Cải tiến với Knuth-Morris-Pratt

# Giới thiệu



- Đối sánh chuỗi
  - Từ khóa: String matching, String searching, Pattern searching, Text Searching
  - Một trong những thuật toán quan trọng và có ứng dụng rộng rãi.

# Giới thiệu



- Ứng dụng của đối sánh chuỗi:
  - Máy tìm kiếm
  - Trình soạn thảo văn bản
  - Trình duyệt web
  - Sinh học phân tử (Tìm mẫu trong dãy DNA).
  - ..

# Giới thiệu



- Mục tiêu:
  - Kiểm tra sự tồn tại của một chuỗi ký tự (mẫu, pattern) trong một chuỗi ký tự có kích thước lớn hơn nhiều (văn bản, text).
  - Nếu tồn tại, trả về một (hoặc nhiều) vị trí xuất hiện.

- Quy ước:
  - Mẫu cần tìm: P (chiều dài *m*).
  - Văn bản: T (chiều dài *n*).
  - P và T có cùng tập hữu hạn ký tự ∑. (∑ = {0, 1}; ∑={A,..,Z},…)
  - $m \leq n$

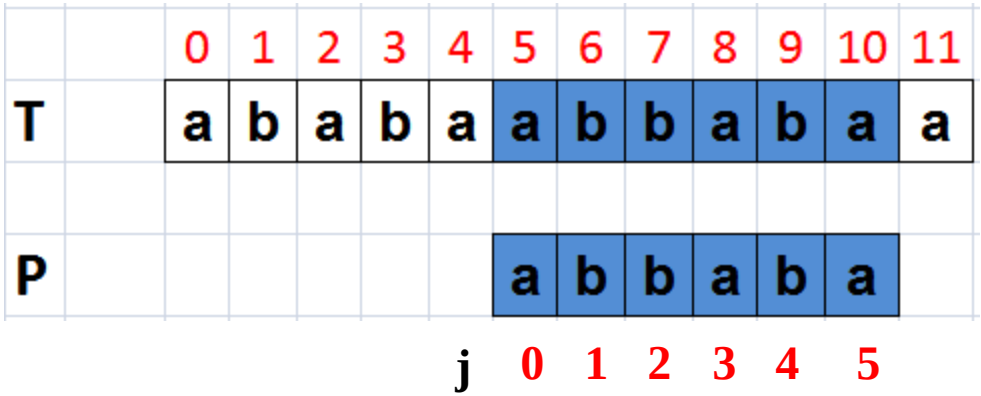
# Giới thiệu



- Đối sánh chuỗi:
  - Bằng cách lần lượt dịch chuyển (cửa sổ) P trên T.
  - P tồn tại trên T tại vị trí bắt đầu là $i$ ($0 \le i \le n - m$) nếu
    - T[$i$ + $j$] = P[$j$] với mọi $0 \le j \le m$ - 1.

- Ví dụ:
  - P = **abbaba**
  - T = **ababaabbabaa**

  => $i$ = 5



# Giới thiệu



- Các thuật toán tiêu biểu:
  - Brute Force
  - Rabin-Karp
  - Morris-Pratt
  - Knuth-Morris-Pratt
  - Boyer-Moore
  - …

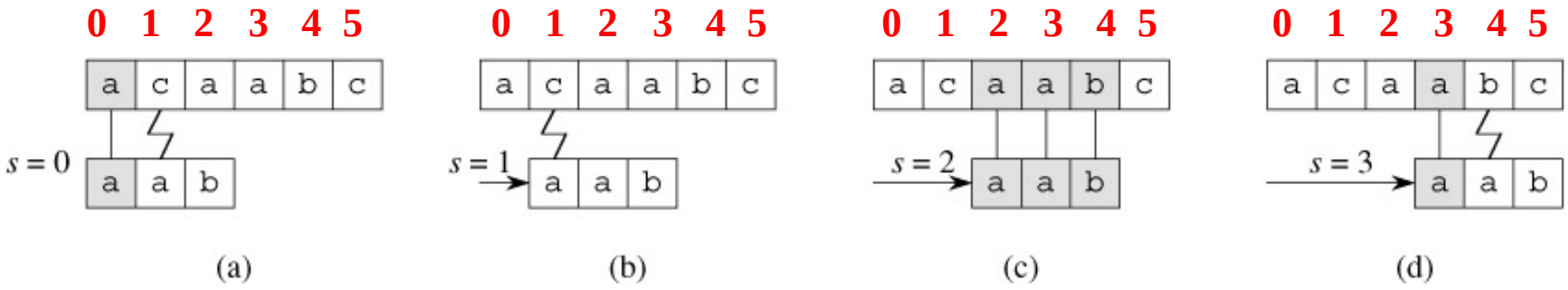
# Thuật toán Brute-Force

# Ý tưởng



- Lần lượt kiểm tra điều kiện P[0…*m*-1] = T[*i*…*i*+*m*-1] tại mọi vị trí có thể của *i*.

- Ví dụ
  - Tìm kiếm P = **aab** trong T = **acaabc**



# Cài đặt



```
bruteForceMatcher(T, P)
    n ← length[T]
    m ← length[P]
    for i ← 0 to n - m
        if P[0..m-1] = T[i…i+m-1]
            return i
```

# Đánh giá



- Trường hợp tốt nhất – không tìm thấy: O(n).
- Trường hợp xấu nhất – không tìm thấy: O(n*m).
- Trường hợp trung bình: O(n+m).

# Đặc điểm chính



- Không cần thao tác tiền xử lý trên P.
- Luôn luôn dịch chuyển mẫu (cửa sổ) sang phải một vị trí.
- Thao tác so sánh có thể thực hiện theo bất kỳ chiều nào.
- Trường hợp xấu nhất: O((n-m+1)*m).

- Viết hàm để tìm kiếm vị trí đâu tiên của P trong T.

```
Int bruteForceMatcher(char* P,char* T)
    n ← length[T]
    m ← length[P]
    1. for i ← 0 to n – m
       cờ = 1
       1.2 Duyệt qua các vị trí  j của P
            1.2.1 Kiểm tra P[j] != T[i+j]
                      flag= 0;
             1.2.2 thoát vòng for
       1.3 Nếu cờ == 1
              Trả về vị trí thích hợp
```

# Thuật toán Morris-Pratt

# Đặt vấn đề



- Điểm hạn chế của thuật toán Brute-Force:
  - Không ghi nhớ được thông tin đã trùng khớp (trước) khi xảy ra tình trạng không so khớp.
  - Phải so sánh lại từ đầu (trên P) trong tất cả trường hợp

# Đặt vấn đề



- Ví dụ:

  i i+j

  - T: **10101**011101001;
  - P: **10100**

    j

  - Tìm đoạn trùng gần nhất để nhảy tới?
    - i $i_1$
      - T : 10**101**011101001
      - P: **101**00

        $j_1$

- i = 0, j = 4, T[i+j] != P[j] => $i_1$ = 2, $j_1$ = 2

# Ví dụ:



i = 0

- T: **10110**1011101001;
- P: **101100**

j = 5

- T: **10110**1011101001;
- P: **10**1100

$i_1 = 3$

- T: **10110**1011101001;
- P: **10**1100

$j_1 = 2$

- Dịch chuyển thành: $i_1 = 3$ và $j_1 = 2$

# Ví dụ:



i = 0

- T: **acbXyacbDefffea**;
- P: **acbXyacbmyio**

j = 8

- T: **acbXyacbDefffea**;
- P: **acbXyacbmyio**

***Vị trí i mới sẽ được tính như thế nào?***

$i_1 = 5$

- T: **acbXyacbDefffea**;
- P: **acbXyacbmyio**

$j_1 = 3$

- Dịch chuyển thành: i= 5 và $j_1 = 3$

# Ví dụ: *Vị trí i mới mà không cần biết T không?*



- T: ??????????????
- P: **acbXyacbmyio**

  j = 8

- T: **acbXyacb??????**;
- P: **acbXyacbmyio**

- P: **acbXyacbmyio**;

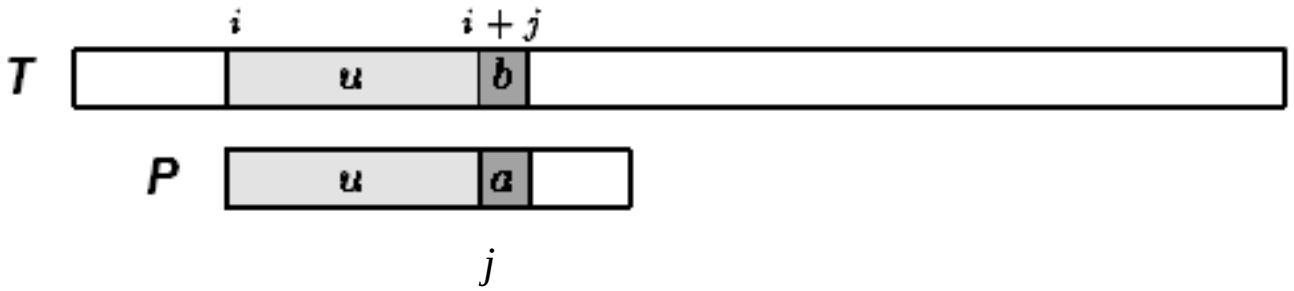
  $i_1 = 5$
- P: **acbXyacbmyio**;
- P: **acbXyacbmyio**

  $j_1 = 3$

# Đề xuất của thuật toán



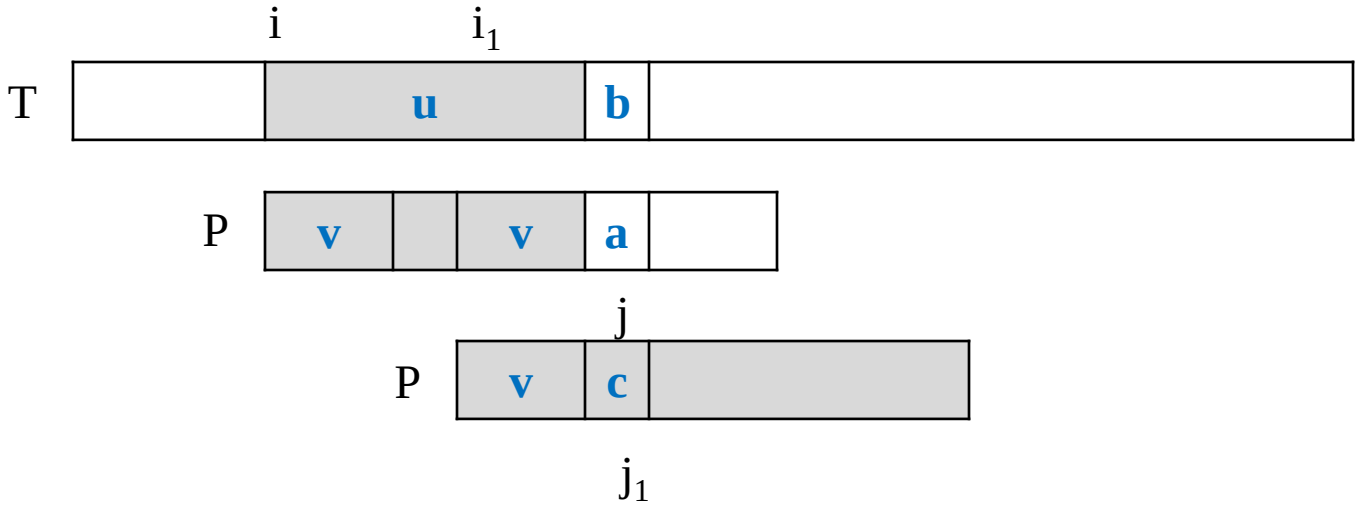
- Giả sử:
  - *i* là vị trí bắt đầu sự đối sánh (trên T).
  - *j* là vị trí đang so sánh (trên P). (Ký tự tương ứng trên T tại vị trí *i*+*j*).
  - T[i+j] != P[j] => không so khớp



# Đề xuất của thuật toán



- Tìm:
  - Vị trí mới $i_1$ (trên T) và $j_1$ (trên P) sao cho
    - i+j = $i_1$+$j_1$ (vị trí đang xem xét)
    - $v$ =T[$i_1$ … $i_1$+$j_1$–1] là đoạn so khớp mới giữa P và T.
- Khi đó:
  - Đoạn dịch chuyển cửa sổ: $j - j_1$.(do $j_1 < j$)
  - Có thể tìm $i_1$ dựa trên $j_1$.



# Tính $i_1$ dựa trên $j_1$



- Do $i + j = i_1 + j_1$ nên $i_1 = i + j - j_1$

# Đề xuất của thuật toán

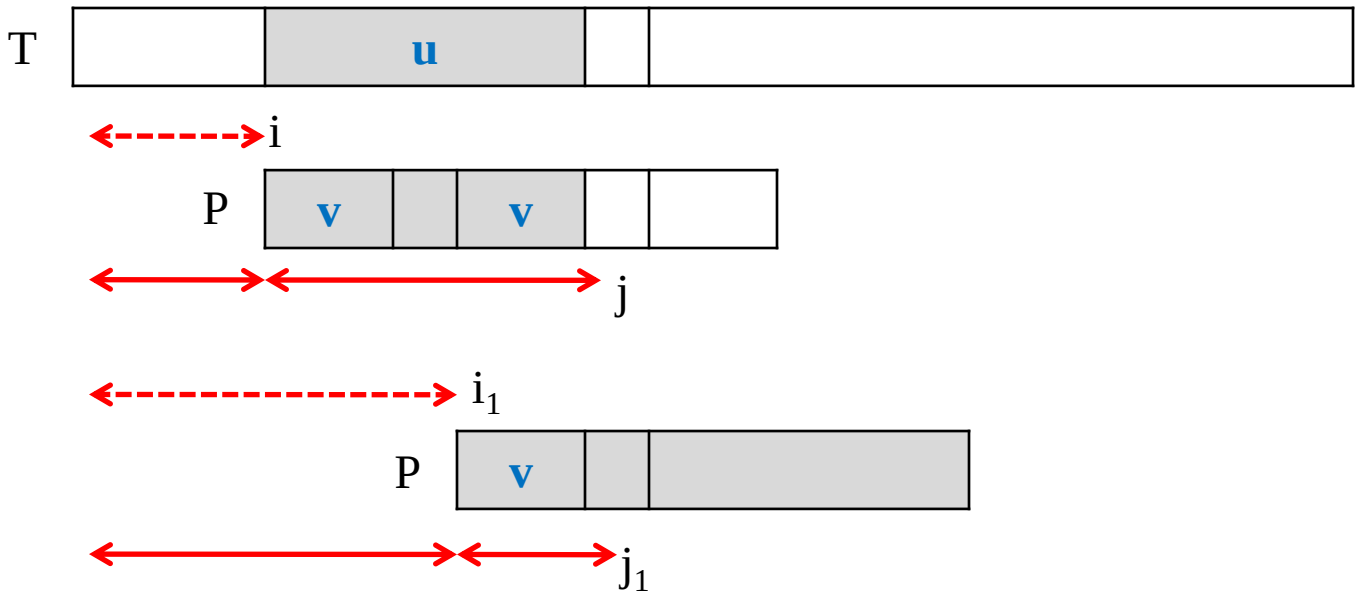


- Vấn đề:
  - Tìm giá trị $j_1$ dựa trên $j$.
- Cách thức:
  - Tính sẵn các giá trị của $j_1$ ứng với mỗi giá trị $j$ (dựa trên P).
- Câu hỏi:
  - Có thể làm được không? Tại sao?

# Tính $i_1$ dựa trên $j_1$



- Do $i + j = i_1 + j_1$ nên $i_1 = i + j - j_1$




$$i + j = i_1 + j_1$$

# Xây dựng bảng NEXT



- Bảng NEXT:
  - Bảng chứa các giá trị $j_1$ ứng với các giá trị $j$.

- Ví dụ:

| j | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| $j_1$ | -1 | 0 | 1 | 1 | 0 | 3 | 2 |

# Xây dựng bảng NEXT



- Hoàn toàn dựa trên P.

- Cách thức:
  - NEXT[0] = -1
  - Với mỗi j > 0, giá trị của NEXT[$j$] ($j_1$) là số $j_1$ **lớn nhất** ($j_1 < j$) sao cho:
    - $j_1$ ký tự đầu tiên khớp với $j_1$ ký tự cuối trước vị trí $j$.
    - Nghĩa là P[0..$j_1$-1] = P[j-$j_1$ ..j-1]

# Xây dựng bảng NEXT



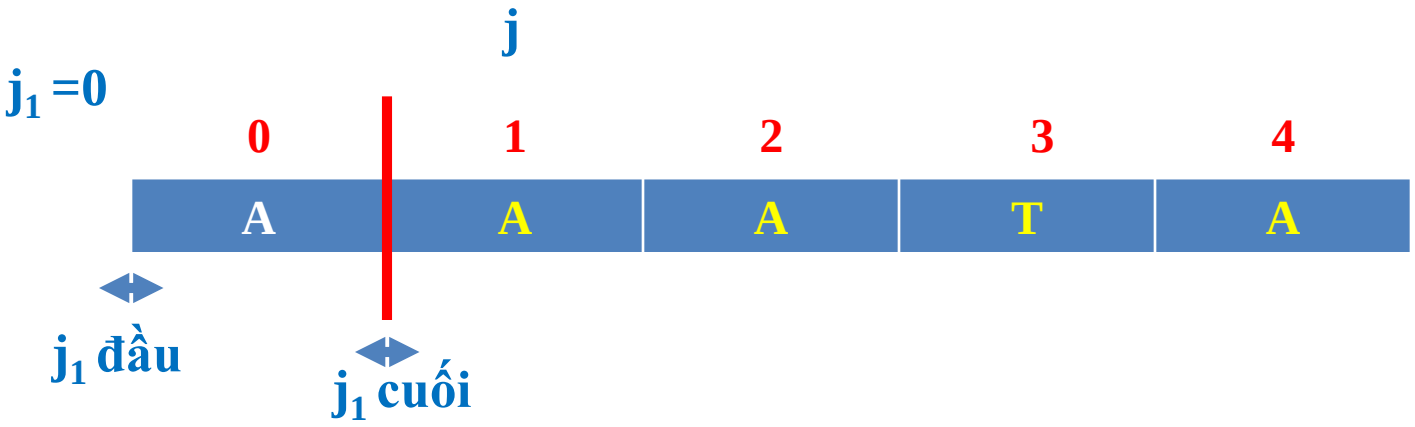
- Ví dụ:
  - P = AAATA
  - Bảng NEXT:
    - NEXT[0] = -1

# Xây dựng bảng NEXT



- P = AAATA
- j = 1 → $j_1$ = {0}
- NEXT[1] = $j_1$ = 0

**Thỏa: $j_1$ đầu (rỗng) = $j_1$ cuối (rỗng)**

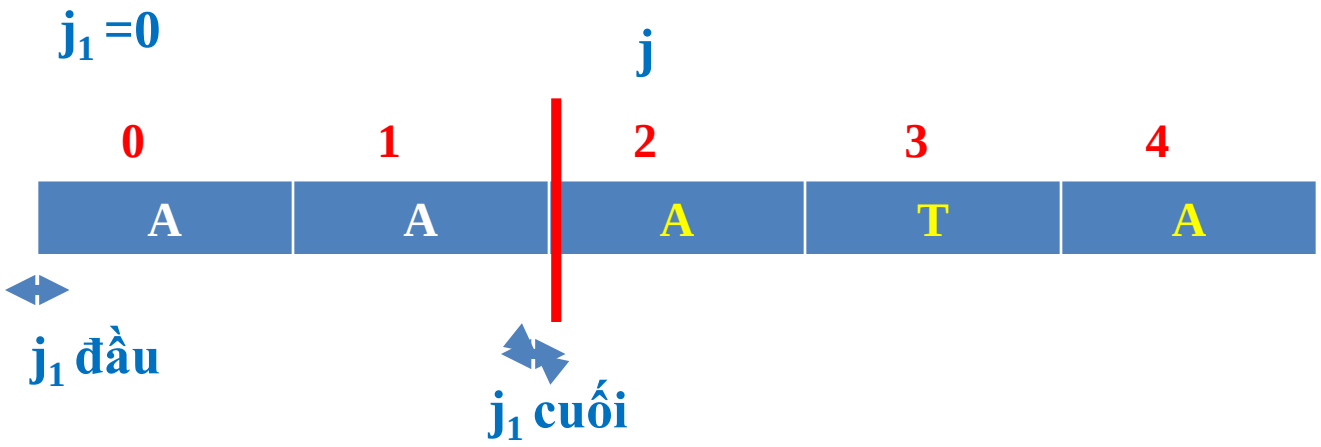


# Xây dựng bảng NEXT



- P = AAATA
- j = 2 → $j_1$ = {0, 1}

**Thỏa: $j_1$ đầu (rỗng) = $j_1$ cuối (rỗng)**

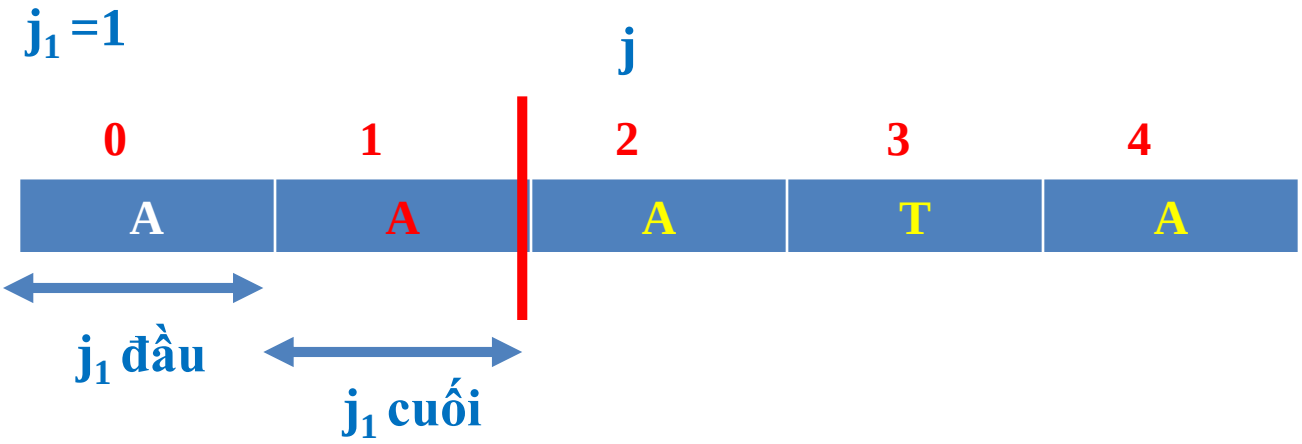


# Xây dựng bảng NEXT



- P = AAATA
- j = 2 → $j_1$ = {0, 1}

**Thỏa: $j_1$ đầu (A) = $j_1$ cuối (A)**



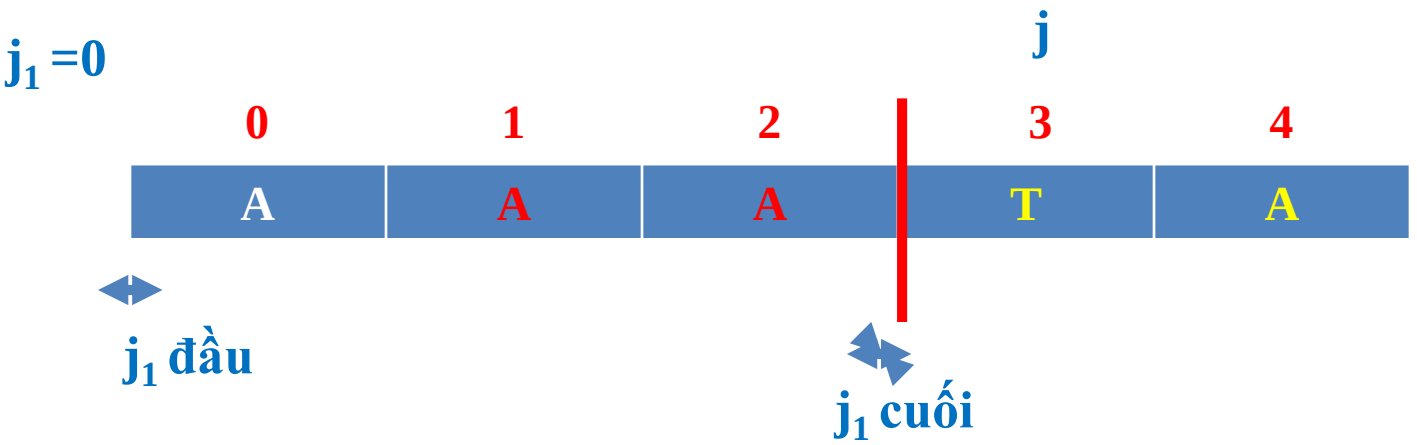

➔NEXT[2] = $j_1$ = 1

# Xây dựng bảng NEXT



- P = AAATA
- j = 3 → $j_1$ = {0, 1, 2}

**Thỏa: $j_1$ đầu (rỗng) = $j_1$ cuối (rỗng)**

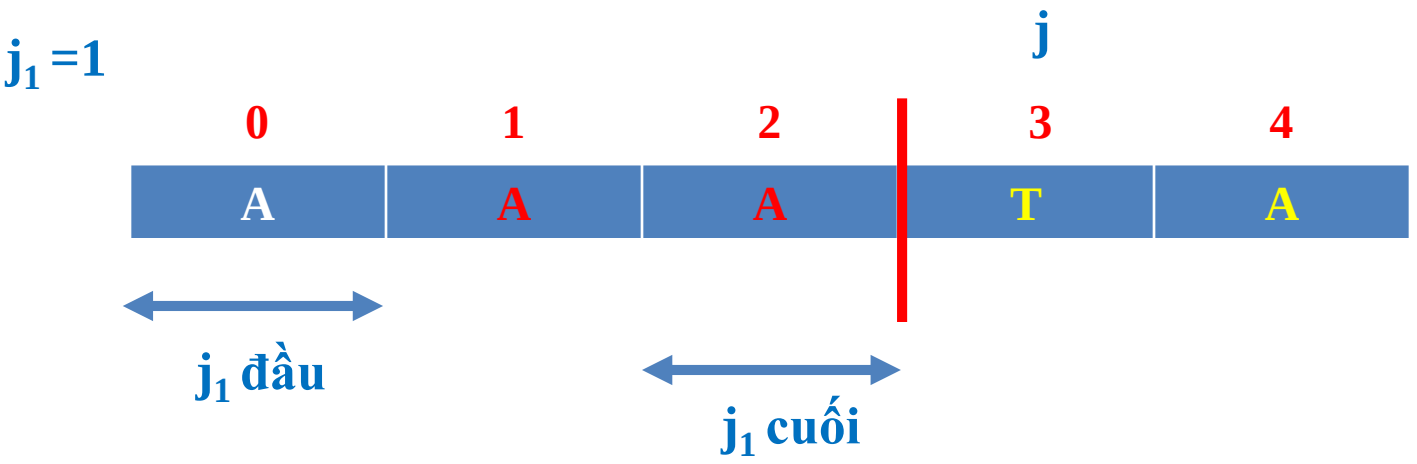


# Xây dựng bảng NEXT



- P = AAATA
- j = 3 → $j_1$ = {0, 1, 2}

**Thỏa: $j_1$ đầu (A) = $j_1$ cuối (A)**

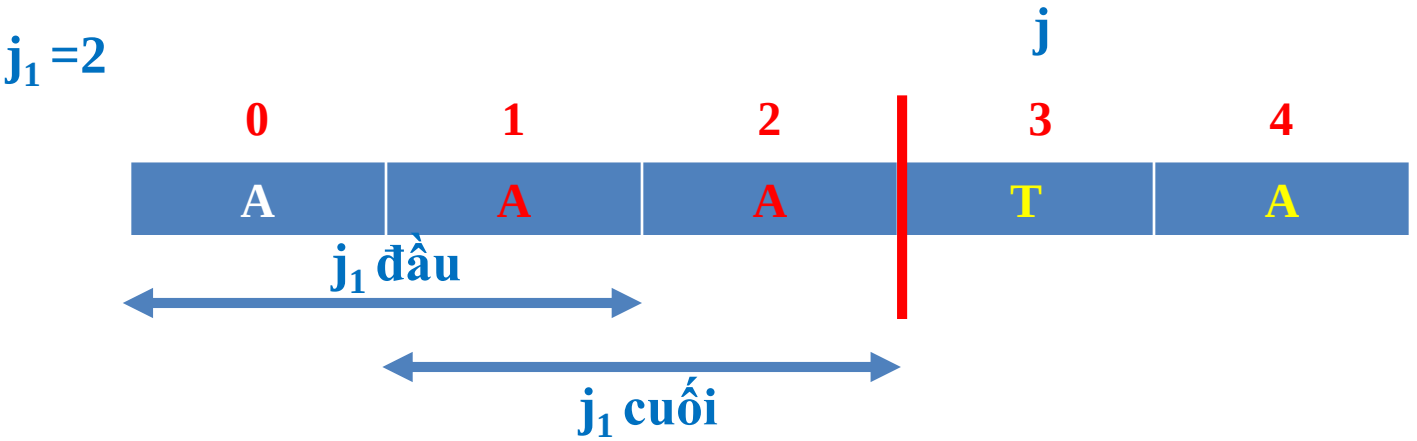


# Xây dựng bảng NEXT



- P = AAATA
- j = 3 → $j_1$ = {0, 1, 2}

**Thỏa: $j_1$ đầu (AA) = $j_1$ cuối (AA)**




➔NEXT[3] = $j_1$ = 2

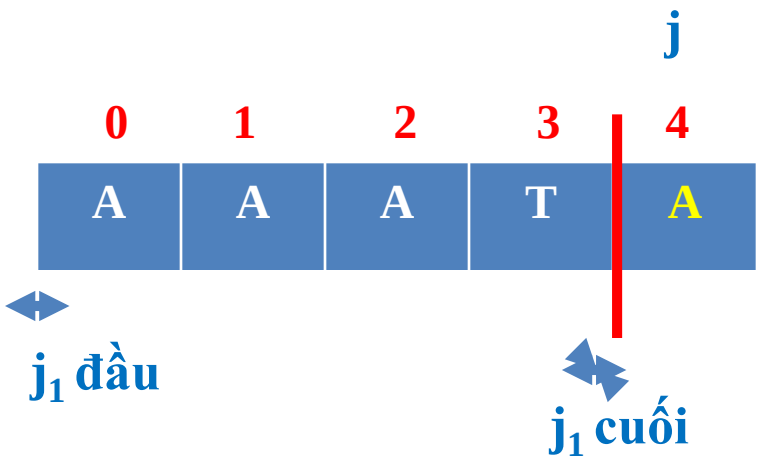
# Xây dựng bảng NEXT



- P = AAATA
- j = 4 → $j_1$ = {0, 1, 2, 3}
- NEXT[4] = $j_1$ = 0

**Thỏa: $j_1$ đầu (rỗng) = $j_1$ cuối (rỗng)**

$j_1$ =0



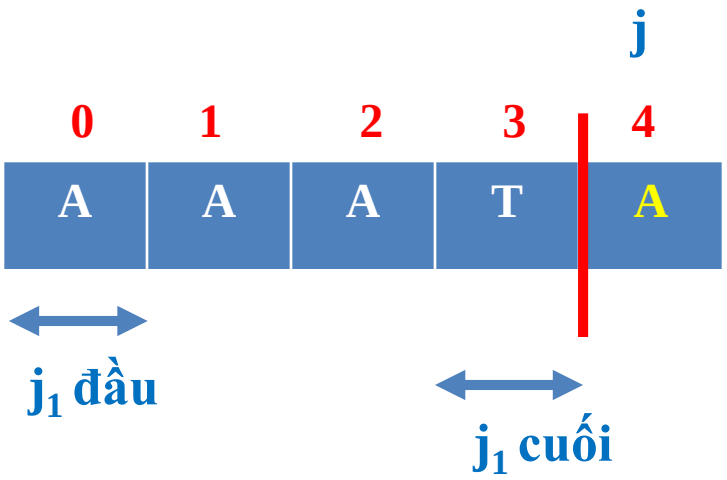
# Xây dựng bảng NEXT



- P = AAATA
- j = 4 → $j_1$ = {0, 1, 2, 3}
- NEXT[4] = $j_1$ = 0

**Không thỏa: $j_1$ đầu (A) <> $j_1$ cuối (T)**

$j_1$ =1



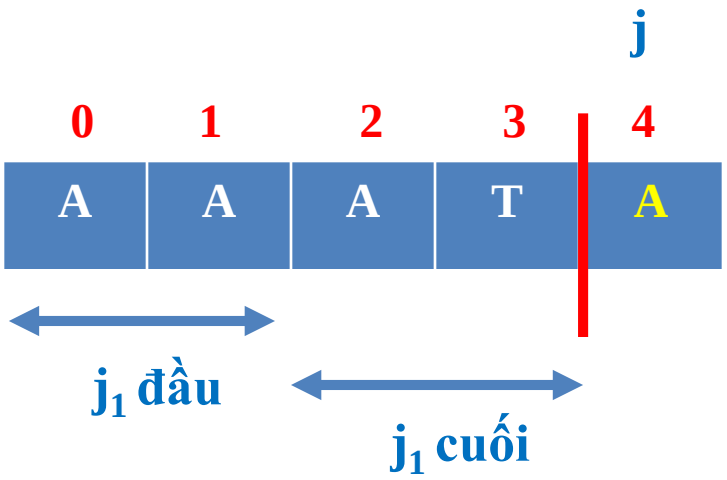
# Xây dựng bảng NEXT



- P = AAATA
- j = 4 → $j_1$ = {0, 1, 2, 3}
- NEXT[4] = $j_1$ = 0

**Không thỏa: $j_1$ đầu (AA) <> $j_1$ cuối (AT)**

$j_1$ =2



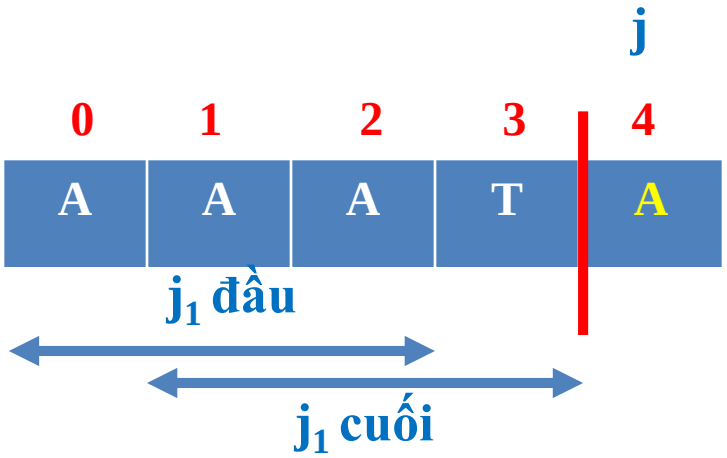
# Xây dựng bảng NEXT



- P = AAATA
- j = 4 → $j_1$ = {0, 1, 2, 3}

**Không thỏa: $j_1$ đầu (AAA) <> $j_1$ cuối (AAT)**

$j_1$ =3




➔ NEXT[4] = $j_1$ = 0

# Xây dựng bảng NEXT



- P = AAATA
- Bảng NEXT
  - NEXT[0] = -1
  - NEXT[1] = 0
  - NEXT[2] = 1
  - NEXT[3] = 2
  - NEXT[4] = 0

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| **NEXT** | **-1** | **0** | **1** | **2** | **0** |

# Tóm tắt Morris-Pratt



- Xây dựng bảng Next m cột, ở đây m là chiều dài P
  - Với mỗi giá trị j (<m)
    - Lần lượt thử với các giá trị của $j_1$ ($j_1$ < j) sao cho sự trùng lặp P[0..$j_1$-1] = P[j-$j_1$ ..j-1] nhiều nhất
  - Ghi giá trị $j_1$ này vào cột j của bảng
- So khớp:
  - Tại mỗi giá trị i và j
    - Tính giá trị $j_1$ từ j dùng bảng NEXT
    - Tính $i_1$ tư công thức i + j = $i_1$ + $j_1$
    - Nhảy tới vị trí mới $i_1$, j1

# Xây dựng bảng NEXT



- Xây dựng bảng NEXT cho P = 10100
- Xây dựng bảng NEXT cho P = ABACAB
- Xây dựng bảng NEXT cho P = GCAGAGAG
- Xây dựng bảng NEXT cho P = AABAABA

# Xây dựng bảng NEXT



- P = 10100

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| NEXT | **-1** | **0** | **0** | **1** | **2** |

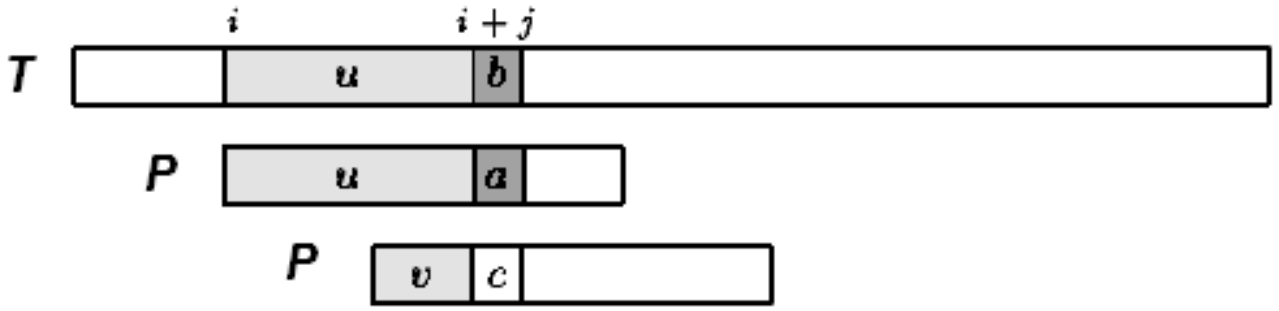
- P = ABACAB

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| NEXT | **-1** | **0** | **0** | **1** | **0** | **1** |

# Sử dụng NEXT trong thuật toán



- Mục tiêu :
  - Xác định vị trí mới $i_1$ (trên T) và $j_1$ (trên P) sao cho
    - i+j = $i_1$+$j_1$ (vị trí đang xem xét)
    - $v$ =T[$i_1$ … $i_1$+$j_1$–1] là đoạn so khớp mới giữa P và T.
- Đã có $j_1$ = NEXT[$j$]
- Vậy, $i_1 = i + j$ – NEXT[$j$]



# Sử dụng NEXT trong thuật toán`1



- Ví dụ:
  - T = AATAAAATA
  - P = AAATA

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| **NEXT** | **-1** | **0** | **1** | **2** | **0** |

  - *i* = 0 **A**ATAAAATA
  - *j* = 0 **A**AATA

  - *i* = 0 **AA**TAAAATA
  - *j* = 1 **AA**ATA

# Sử dụng NEXT trong thuật toán



- Ví dụ:
  - T = AATAAAATA
  - P = AAATA

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| **NEXT** | **-1** | **0** | **1** | **2** | **0** |

  - *i* = 0 **AA**TAAAATA ($i_1 = 0 + 2 - 1 = 1$)
  - *j* = 2 **AA**ATA ($j_1 = 1$)

  - *i* = 1 AATAAAATA ($i_1 = 1 + 1 - 0 = 2$)
  - *j* = 1 AAATA ($j_1 = 0$)

# Sử dụng NEXT trong thuật toán



- Ví dụ:
  - T = AATAAAATA
  - P = AAATA

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| **NEXT** | **-1** | **0** | **1** | **2** | **0** |

  - *i* = 2 AATAAAATA ($i_1 = 2 + 0 - (-1) = 3$)
  - *j* = 0 AAATA ($j_1 = -1$)

  - *i* = 3 AATAAAATA
  - *j* = 0 AAATA

# Sử dụng NEXT trong thuật toán



- Ví dụ:
  - T = AATAAAATA
  - P = AAATA

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| **NEXT** | **-1** | **0** | **1** | **2** | **0** |

  - *i* = 3 AATAAAATA ($i_1 = 3 + 3 - 2 = 4$)
  - *j* = 3 AAATA ($j_1 = 2$)

  - *i* = 4 AATAAAATA
  - *j* = 2 AAATA

# Sử dụng NEXT trong thuật toán



- Ví dụ:
  - T = AATAAAATA
  - P = AAATA

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| **NEXT** | **-1** | **0** | **1** | **2** | **0** |

  - *i* = 4 AATA**AAATA**
  - *j* = 4 **AAATA**

(Hoàn toàn so khớp, vị trí xuất hiện của P trong T tại *i*=4)

# Độ phức tạp



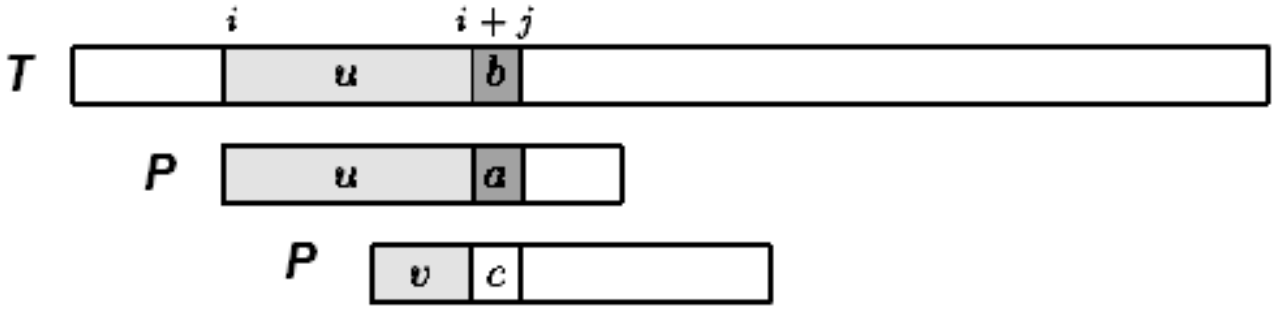
- Tính NEXT: O(m)
- Tìm kiếm: O(n)
- Tổng: O(n+m)

# Thuật toán Knuth-Morris-Pratt

# Ý tưởng



- Thuật toán Knuth-Morris-Pratt cải tiến Morris-Pratt bằng cách
  - bổ sung thêm điều kiện a ≠ c (vì nếu a =c sẽ không so khớp ngay sau khi dịch chuyển).



# Ý tưởng



- Thuật toán Knuth-Morris-Pratt cải tiến Morris-Pratt bằng cách
  - Giai đoạn tiền xử lý có một cải tiến nhỏ:
    - Tính độ dịch chuyển tốt hơn → tránh so sánh cùng một ký tự trong T hai lần.
  - Giai đoạn tìm kiếm: hoàn toàn giống thuật toán Morris-Pratt

# Ý tưởng



- Thay đổi cách tính bảng NEXT:
  - Nếu **p[j]** $\neq$ **p[$j_1$]** thì **NEXT[j] = $j_1$**
  - Ngược lại **NEXT[j] = NEXT[$j_1$]**

- Thao tác tìm kiếm vẫn không thay đổi

# Xây dựng bảng NEXT



- P = 10100

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| MP | -1 | 0 | 0 | 1 | 2 |
| KMP | -1 | 0 | -1 | 0 | 2 |

- P = ABACAB

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| MP | -1 | 0 | 0 | 1 | 0 | 1 |
| KMP | -1 | 0 | -1 | 1 | -1 | 0 |

# Độ phức tạp



- Tính NEXT: O(m)
- Tìm kiếm: O(n)
- Tổng: O(n+m)

# Hỏi và Đáp