HỌC VIỆN KỸ THUẬT MẬT MÃ
ACADEMY OF CRYPTOGRAPHY TECHNIQUES

NỘI DUNG

HỌC VIỆN KỸ THUẬT MẬT MÃ
ACADEMY OF CRYPTOGRAPHY TECHNIQUES

CHƯƠNG 03

ĐỐI SÁNH MẪU TRÊN CHUỖI

Bài 01 – Mục tiêu

- Nắm được bài toán đối sánh mẫu
- Hiểu được các thuật toán đối sánh mẫu: Vét cạn, Boyer-Moore và Knuth-Morris-Pratt
- Hiểu và lập trình được các thuật toán đối sánh mẫu Vét cạn, Boyer-Moore và Knuth-Morris-Pratt

ĐỐI SÁNH MẪU TRÊN CHUỖI

- Bài toán đối sánh mẫu
- Thuật toán đối sánh vét cạn
- Thuật toán Boyer-Moore
- Thuật toán Knuth-Morris-Pratt
- Thuật toán Wu-Manber
- Thuật toán Aho-Corasick

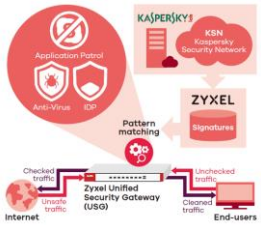
Đối sánh mẫu là gì?

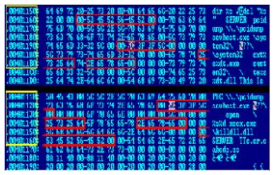
- **Định nghĩa:** Cho một văn bản T và một mẫu chuỗi ký tự cần tìm P, hãy tìm kiếm sự xuất hiện của mẫu P trong văn bản T
- $T= T_0T_1T_2...T_{n-1}$, $|T|=n$ (độ dài của văn bản)
- $P= P_0P_1P_2...P_{m-1}$, $|P|=m$ (độ dài của mẫu đối sánh)
- Σ – bảng chữ cái với $|\Sigma| = c$

Ví dụ

- **Ví dụ:**
  - T: “the rain in Viet Nam stays mainly on the plain”
  - P: “n th”
- **Ứng dụng:**
  - Các trình text editors, dịch vụ tìm kiếm (e.g. Google),…

Vai trò của đối sánh mẫu trong ATTT





String và Substring

- S là một chuỗi ký tự có kích thước là n.
- Một *chuỗi con* (*substring* S[i .. j]) của S là một phần của chuỗi ký tự S bắt đầu từ chỉ số i và kết thúc tại j, $0 \le i \le j \le n-1$
- với $i \in [0, n-1]$ ta có:
  - *Tiền tố (prefix)* của S là chuỗi con S[0 .. i]
  - *Hậu tố (suffix)* của S là chuỗi con S[i .. n-1]
- Ví dụ:

S

a	n	t	o	a	n
0					5

String và Substring (Ví dụ)

S

a	n	t	o	a	n
0					5

- Substring S[1..3] == "nto"
- Tất cả các tiền tố có thể có của S:
  - "antoan", "antoa", "anto", "ant", "an", "a"
- Tất cả các hậu tố có thể có của S:
  - "antoan", "ntoan", "toan", "oan", "an", "n"

Các thuật toán đối sánh mẫu

Đơn mẫu:

- Vét cạn
- Knuth-Morris-Pratt (KMP)
- Karp–Rabin
- Boyer–Moore
- Horspool
- Shift-OR, Shift-AND
- Factor searches

Đa mẫu:

- Wu – Manber
- Commentz - Walter
- Aho–Corasick

Indexing:

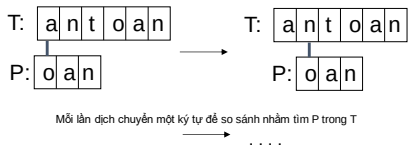
- Trie (và suffix trie)
- Suffix tree

ĐỐI SÁNH MẪU TRÊN CHUỖI

- Bài toán đối sánh mẫu
- Thuật toán đối sánh vét cạn
- Thuật toán Boyer-Moore
- Thuật toán Knuth-Morris-Pratt
- Thuật toán Wu-Manber
- Thuật toán Aho-Corasick

Thuật toán vét cạn

- Kiểm tra mỗi vị trí của đoạn văn bản T để tìm sự xuất hiện đầu tiên của mẫu P tại vị trí đó



Phân tích độ phức tạp tính toán

- Độ phức tạp thời gian O(mn) trong trường hợp tệ nhất.
- Tuy nhiên trong trường hợp trung bình độ phức tạp: O(m+n).
- Hiệu quả trong trường hợp bảng chữ cái lớn:
  - Ví dụ. A..Z, a..z, 1..9, ...
- Không hiệu quả trong trường hợp bảng chữ cái nhỏ
  - Ví dụ. 0, 1 (chẳng hạn file nhị phân, file ảnh...)

Phân tích độ phức tạp tính toán

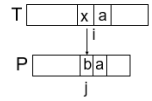
- Ví dụ về trường hợp tồi nhất:
  - T: "aaaaaaaaaaaaaaaaaaaaaaaaaah"
  - P: "aaah"
- Ví dụ về trường hợp trung bình:
  - T: "a string searching example is standard"
  - P: “example"

ĐỐI SÁNH MẪU TRÊN CHUỖI

- Bài toán đối sánh mẫu
- Thuật toán đối sánh vét cạn
- Thuật toán Boyer-Moore
- Thuật toán Knuth-Morris-Pratt
- Thuật toán Wu-Manber
- Thuật toán Aho-Corasick

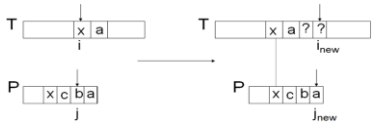
Thuật toán Boyer - Moore

- Thuật toán đối sánh mẫu Boyer – Moore dựa trên hai kỹ thuật chính:
  1. Kỹ thuật *looking-glass (đối sánh cuối trước)*
  - Tìm mẫu P trong T bằng cách tìm từ cuối về đầu của mẫu P
  - i, j là chỉ số của các kí tự đang được so sánh T[i] với P[j]
    - Nếu $T[i] = P[j]$ thì $i = i - 1, j = j - 1$
    - Ngược lại thực hiện nhảy cách
  2. Kỹ thuật *character-jump (nhảy cách)*
  - Khi ký tự đối sánh không khớp tại vị trí T[i] == x
  - Tức là $T[i] \neq P[j]$



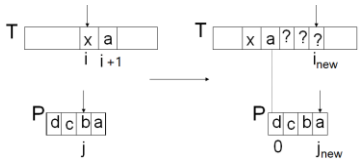
Thuật toán Boyer – Moore (Nhảy cách-Case 1)

- $T[i] = x \neq P[j]$
  - Nếu P chứa x ở một vị trí t nào đó, t < j, thì dịch chuyển *P* sang phải sao vị trí t trong P thẳng với vị trí i hiện tại của T. Sau đó i chuyển sang vị trí i_{new}, j chuyển về cuối của P (j_{new})



Thuật toán Boyer – Moore (Nhảy cách-Case 2)

- $T[i] = x \neq P[j]$
  - Ngược lại, dịch P sao cho P[0] thẳng với T[i+1], sau đó i chuyển sang vị trí i_{new} và j chuyển về cuối của P (j_{new})



Thuật toán Boyer – Moore (Ví dụ 1)

- Bài tập áp dụng: với T = a pattern matching algorithm;

  P = rithm

Tiền xử lý (hàm last occurrence)

- Thuật toán Boyer-Moore tiền xử lý mẫu P và bảng chữ cái Σ để xây dựng một hàm last occurrence L()
  - L() ánh xạ tất cả các ký tự trong bảng chữ cái Σ thành số nguyên
- L(x) được định nghĩa là:
  - Chỉ số i lớn nhất sao cho P[i] == x, hoặc
  - -1 nếu không tồn tại chỉ số i đó
- L() được tính khi mẫu P được đưa vào.
- Giá trị của L() được lưu trữ dưới dạng một mảng

Tiền xử lý (hàm last occurrence) – Ví dụ

P

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

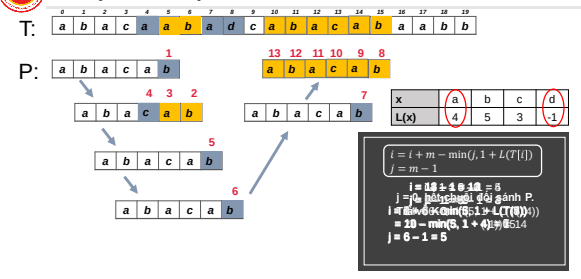
- A = {a, b, c, d}
- P: "abacab"

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

Tiền xử lý (hàm last occurrence)

- Khi thực hiện đối sánh gặp kí tự không khớp $T[i] \neq P[j]$ thì cần sử dụng L() để nhảy cách như sau:
  - $i_{new} = i + m - \min(j, 1 + L(T[i]))$
  - $j_{new} = m - 1$
  - Trong đó m là độ dài của mẫu P

Ví dụ minh họa



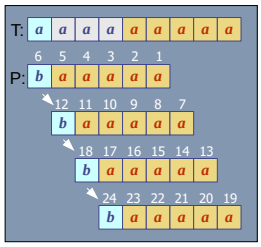
Độ phức tạp

- Độ phức tạp thời gian của thuật toán Boyer-Moore trong trường hợp tồi nhất là O(nm + Σ)
- Boyer-Moore thực hiện tốt nếu bảng chữ Σ lớn, chậm khi bảng chữ Σ nhỏ.
  - Tốt cho các văn bản ngôn ngữ thông thường, kém cho tệp nhị phân
- Boyer-Moore *nhanh hơn nhiều lần* so với vét cạn.

Độ phức tạp (trường hợp tệ nhất)

- T: "aaaaaaaaa"
- P: "baaaaa"

Mất bao nhiêu bước thực hiện đối sánh?



ĐỐI SÁNH MẪU TRÊN CHUỖI

- Bài toán đối sánh mẫu
- Thuật toán đối sánh vét cạn
- Thuật toán Boyer-Moore
- Thuật toán Knuth-Morris-Pratt
- Thuật toán Wu-Manber
- Thuật toán Aho-Corasick

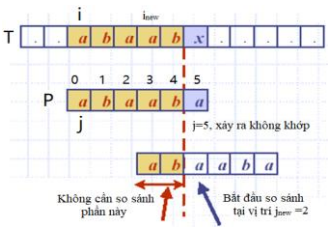
Knuth-Morris-Pratt (KMP)

- Thuật toán Knuth-Morris-Pratt tìm kiếm mẫu P trong chuỗi T theo thứ tự từ *trái sang phải* (tương tự vét cạn).
- Dịch chuyển mẫu thông minh hơn so với vét cạn
- Nếu xuất hiện sự không trùng khớp giữa mẫu P tại P[j], bước dịch chuyển *dài nhất* có thể thực hiện là bao nhiêu?
  - *Trả lời*: Độ dài của tiền tố dài nhất của P[0 .. j-1] sao cho nó là hậu tố của P[1 .. j-1]

Knuth-Morris-Pratt (KMP)

- Khi thực hiện so sánh, luôn cần sử dụng tới 2 giá trị sau:
  - i: vị trí bắt đầu của chuỗi con trong T đang so sánh với mẫu P, tức là T[i] thẳng với P[0]
  - j: vị trí của kí tự trong P đang được so sánh (so sánh T[i+j] với P[j])
- Gặp kí tự khớp thì tăng j lên 1
- Ngược lại, gặp kí tự không khớp, cần xác định i_{new} và j_{new}.

Ví dụ



Ví dụ

- Giả sử xuất hiện kí tự không giống nhau tại j, tức là $T[i+j] \neq P[j]$, khi đó:
  - Tìm tiền tố dài nhất của P[0..j-1] = "a b a a b" sao cho tiền tố đó cũng là hậu tố của P[1 .. j-1] = "b a a b"
    - => Đó là: "a b"
    - Không cần phải so sánh lại chuỗi con "a b" nữa nên đặt j_{new} = 2

Hàm failure function

- Là hàm tiền xử lý mẫu của KMP
  - Để tìm kiếm sự trùng khớp của các tiền tố trong P[0..j-1] với các hậu tố trong P[1..j-1], với j là vị trí tại đó $T[i+j] \neq P[j]$
  - Được sử dụng để xác định i_{new} và j_{new}

Hàm failure function

- Hàm *failure function* F(j) được định nghĩa là kích thước của tiền tố dài nhất *của* P[0..j-1] sao cho nó cũng là hậu tố của P[1..j-1].
  - Mặc định:
    - F(0)=-1
    - Độ dài của xâu rỗng là 0

Ví dụ: Tính hàm failure function

- P: "abacab“, j = [0..5]
  - j=0 => F(0)=-1
  - j=1
    - Các tiền tố của P[0..j-1]=“a” là {a}
    - Các hậu tố của P[1..j-1]= Ø là {Ø}
    - => F(1)=0

Ví dụ: Tính hàm failure function

- P: " abacab“, j = [0..5]
  - j=2
    - Các tiền tố của P[0..j-1]=“ab” là {a,ab}
    - Các hậu tố của P[1..j-1]=“b” là {b}
    - => F(2)=0

Ví dụ: Tính hàm failure function

- P: " abacab“, j = [0..5]
  - j=3
    - Các tiền tố của P[0..j-1]=“aba” là {a,ab, aba}
    - Các hậu tố của P[1..j-1]=“ba” là {ba, a}
    - => F(3)=1

Ví dụ: Tính hàm failure function

- P: " abacab“, j = [0..5]
  - j=4
    - Các tiền tố của P[0..j-1]=“abac” là {a,ab,aba, abac}
    - Các hậu tố của P[1..j-1]=“bac” là {bac, ac, c}
    - => F(4)=0

Ví dụ: Tính hàm failure function

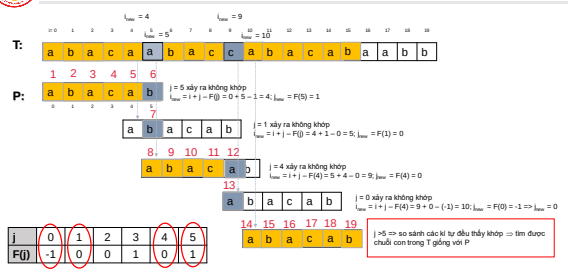
- P: " abacab“, j = [0..5]
  - j=5
    - Các tiền tố của P[0..j-1]=“abaca” là {a,ab,aba, abac, abaca}
    - Các hậu tố của P[1..j-1]=“baca” là {baca, aca, ca, a}
    - => F(5)=1

j	0	1	2	3	4	5
F(j)	-1	0	0	1	0	1

Knuth-Morris-Pratt

- Giả sử:
  - i là vị trí bắt đầu của chuỗi con trong T mà đang so sánh với mẫu P, tức là T[i] thẳng với P[0]
  - j: vị trí của kí tự trong P đang được so sánh và $T[i+j] \neq P[j]$
- => Xác định i_{new} và j_{new} như sau:
  - $i_{new} = i + j - F[j]$
  - $j_{new} = F[j]$, nếu F$[j]$ =-1 thì $j_{new} = 0$

Ví dụ



Ưu nhược điểm của KMP

- KMP có độ phức tạp tối ưu: O(m+n) → rất nhanh
- Thuật toán không cần di chuyển ngược lại trong văn bản T
  - Điều này làm cho nó hiệu quả khi xử lý trên các tệp tin rất lớn được đọc vào từ thiết bị ngoại vi hoặc trong luồng mạng
- KMP không làm việc tốt trên bộ ký tự đối sánh có kích thước lớn
  - Xác suất không khớp lớn hơn
  - Việc không khớp xuất hiện sớm trong mẫu, nhưng KMP lại nhanh hơn khi việc không khớp xảy ra ở phía sau của mẫu

Bài 02 – Mục tiêu

- Nắm được thuật toán đối sánh mẫu Wu-Manber và Aho-Corasick
- Hiểu và lập trình được hai thuật toán đối sánh mẫu Wu-Manber và Aho-Corasick

ĐỐI SÁNH MẪU TRÊN CHUỖI

- Bài toán đối sánh mẫu
- Thuật toán đối sánh vét cạn
- Thuật toán Boyer-Moore
- Thuật toán Knuth-Morris-Pratt
- Thuật toán Wu-Manber
- Thuật toán Aho-Corasick

Bài toán đối sánh mẫu

Văn bản N K A N M A N K K B A A
Mẫu K B A A

- Bài toán đối sánh đơn mẫu: Cho trước một chuỗi ký tự gọi là văn bản và một chuỗi ký tự gọi là mẫu, xác định xem văn bản có chứa mẫu hay không
- Bài toán đối sánh đa mẫu: Cho trước một chuỗi ký tự gọi là văn bản và một tập hợp các mẫu (cũng là các chuỗi ký tự), xác định xem văn bản có chứa mẫu nào hay không
- Hai thuật toán phổ biến nhất để giải quyết bài toán đối sánh mẫu
  - Boyer-Moore
  - Aho-Corasick

Thuật toán Boyer-Moore (BM)

Văn bản (T) N K A N M A N K K B A A
Mẫu (P) K B A A

- Giải quyết bài toán đối sánh đơn mẫu
- Chuỗi ký tự cần tìm kiếm gọi là mẫu, ký hiệu P và có độ dài *m*
- Chuỗi ký tự được tìm kiếm trong đó gọi là văn bản, ký hiệu T và có độ dài *l*
- **Bài toán:** Xác định mẫu có xuất hiện trong văn bản không và nếu có xuất hiện bao nhiêu lần và ở vị trí nào
- Thuật giải duyệt toàn bộ
  - Đặt P và T sao cho ký tự đầu tiên của 2 chuỗi thẳng hàng
  - Dịch chuyển P sang bên phải từng ký tự một, so sánh P với các ký tự thẳng hàng tương ứng của T xem có trùng khớp không
  - Quá trình kết thúc khi ký tự cuối cùng của P thẳng hàng với ký tự cuối cùng của T

Phương pháp duyệt toàn bộ

① Văn bản N K A N M A N K K B A A / Mẫu K B A A
② Văn bản N K A N M A N K K B A A / Mẫu K B A A
③ Văn bản N K A N M A N K K B A A / Mẫu K B A A K
④ Văn bản N K A N M A N K K B A A / Mẫu K B A A
⑤ Văn bản N K A N M A N K K B A A / Mẫu K B A A
⑥ Văn bản N K A N M A N K K B A A / Mẫu K B A A
⑦ Văn bản N K A N M A N K K B A A / Mẫu K B A A
⑧ Văn bản N K A N M A N K K B A A / Mẫu K B A A
⑨ Văn bản N K A N M A N K K B A A / Mẫu K B A A

Quy tắc dịch chuyển của BM (1)

- Với mỗi bước so sánh P với T, nếu ký tự cuối cùng của mẫu không trùng với ký tự tương ứng của văn bản giả sử là *σ*, xét 2 trường hợp sau
  1. *σ* không xuất hiện trong P. Dịch chuyển P sao cho toàn bộ P nằm ngay bên phải *σ*
  2. Nếu *σ* có xuất hiện trong P, xác định *N* nằm ngoài cùng bên phải của P. Dịch chuyển P sao cho *σ* của T thẳng hàng với *σ* của P

1 2 3 4 5 6 7 8 9 10 11 12
N K A N M A N K K B A A
K B A A
Bỏ qua: K B A A / K B A A / K B A A
1) *σ* = N không xuất hiện trong mẫu
N P A N M A N K K B A A
K B A A
Bỏ qua: K B A A / K B A A / K B A A
2) *σ* = K xuất hiện trong mẫu.

Quy tắc dịch chuyển của BM (2)

- Nếu ký tự cuối cùng của mẫu trùng với ký tự tương ứng của văn bản
- So sánh từng ký tự của mẫu với ký tự tương ứng của văn bản. Có 2 trường hợp sau:
  1. Nếu đến một ký tự nào đó không có sự trùng khớp, dịch mẫu sang phải 1 ký tự để thực hiện bước tiếp theo
  2. Nếu toàn bộ ký tự của mẫu trùng với các ký tự tương ứng của văn bản, thông báo tìm thấy mẫu trong văn bản

1 2 3 4 5 6 7 8 9 10 11 12
N K A N M A N P K B A A
K B A A
1) B của T không trùng A của P
1 2 3 4 5 6 7 8 9 10 11 12
N K A N M A N P K B A A
K B A A
2)Tìm thấy mẫu trong văn bản

Xây dựng bảng delta

- Xây dựng bảng delta[σ] cho mỗi σ∈∑. delta[σ] cho biết số ký tự sẽ được dịch chuyển khi gặp σ trong văn bản
- Nếu σ không xuất hiện trong mẫu P, delta[σ] = độ dài mẫu
- Nếu σ xuất hiện trong mẫu P, delta[σ] = độ dài mẫu – j, với j là vị trí ngoài cùng bên phải σ xuất hiện trong P

- ∑ = {A, B, K, M, N}
- P =KBAA
  - delta[A] = 0
  - delta[B] = 2
  - delta[K] = 3
  - delta[M] = 4
  - delta[N] = 4

	Delta[]
A	0
B	2
K	3
M	4
N	4

Ví dụ thuật toán Boyer-Moore

① 1 2 3 4 5 6 7 8 9 10 11 12
N K A N M A N K K B A A
K B A A

∑ = {A, B, K, M, N}

delta[N] = 4, dịch chuyển 4 ký tự

② 1 2 3 4 5 6 7 8 9 10 11 12
N K A N M A N K K B A A
K B A A

delta[K] = 3, dịch chuyển 3 ký tự

③ 1 2 3 4 5 6 7 8 9 10 11 12
N K A N M A N K K B A A
K B A A

delta[A] = 0 kiểm tra các ký tự tiếp theo

④ 1 2 3 4 5 6 7 8 9 10 11 12
N K A N M A N K K B A A
K B A A

delta[A] = 0 kiểm tra các ký tự tiếp theo

A	0
B	2
K	3
M	4
N	4

Bảng Delta

Thuật toán Wu-Manber

Thuật toán Boyer-Moore được mở rộng bởi Wu và Manber cho phép đối sánh nhiều mẫu đồng thời

$P = \{p_1, p_2, ..., p_k\}$ là một tập hợp các mẫu

Mỗi mẫu là một chuỗi ký tự với các ký tự lấy từ bảng chữ cái ∑

$T = t_1 t_2 ... t_N$ là chuỗi ký tự để tìm kiếm các mẫu trong đó

m là độ dài nhỏ nhất của các mẫu

Thay vì chỉ xét một ký tự như trong thuật toán Boyer-Moore, một block B ký tự sẽ được so sánh cùng một lúc

Quy tắc dịch chuyển

Văn bản	B	M	A	N	P	A	N	A	M	A	N	A	P
Mẫu 1	N	A	M	A	N	A							
Mẫu 2	A	M	A	N	A	B							
Mẫu 3	A	B	M	A	N	A							

N A M A N A
A M A N A B
A B M A N A

1) NPA không xuất hiện trong bất cứ mẫu nào. Cả 3 mẫu cùng dịch 4 ký tự

B M A N P A N A M A N A P
N A M A N A
A M A N A B
A B M A N A

2) AMA xuất hiện trong 3 mẫu. Mẫu 1 được phép dịch 2 ký tự, mẫu 2 được phép dịch 3 ký tự, mẫu 3 được phép dịch 4 ký tự, cả 3 mẫu cùng dịch 2 ký tự

Xây dựng bảng SHIFT

- Cho trước một chuỗi ký tự bất kỳ có độ dài B, bảng SHIFT cho biết số ký tự sẽ được dịch chuyển khi gặp chuỗi ký tự đó trong văn bản T
- Giá trị ban đầu $SHIFT[X_1...X_B] = m-B+1$
- Xét từng mẫu $p_i = a_1a_2...a_m$. Xét các chuỗi ký tự con có độ dài B của $p_{ij} = a_{j-B+1}...a_j$ với $B \le j \le m$. Ta có $SHIFT[p_{ij}] = min (SHIFT[p_{ij}], m-j)$

Xét 3 mẫu

Mẫu 1 NAMANA

Mẫu 2 AMANAB

Mẫu 3 ABMANA

Tính SHIFT[AMA]

Ban đầu SHIFT[AMA] = 6-3+1 = 4

Xét NAMANA→SHIFT[AMA] = 2

Xét AMANAB →SHIFT[AMA] = 2

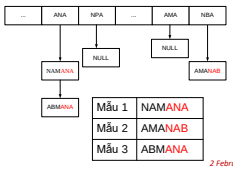
Xét ABMANA→SHIFT[AMA] = 2

SHIFT[AMA] = 2

Xây dựng bảng HASH

- Khi $SHIFT[X_1...X_B] = 0$ tức là tồn tại các mẫu có B ký tự cuối cùng trùng với B ký tự đang xét của T
- Bảng HASH cho biết các mẫu đó là mẫu nào
- $HASH[X_1...X_B]$ chứa con trỏ trỏ đến danh sách các mẫu với B ký tự cuối cùng là $X_1...X_B$.
- Xét từng mẫu $p_i = a_1a_2...a_m$.
- Đưa đưa p_i vào danh sách liên kết mà $HASH[a_{m-B+1}...a_m]$ trỏ tới

B M A N P A N A M A N A P
N A M A N A
A M A N A B
A B M A N A



Mẫu 1	NAMANA
Mẫu 2	AMANAB
Mẫu 3	ABMANA

Ví dụ thuật toán Wu-Manber

NPA	4
AMA	2
ANA	0
NAP	4
NAB	0

Bảng SHIFT

NAB	→AMANAB
ANA	→NAMANA→ABMANA

Bảng HASH

① B M A N P A N A M A N A P
N A M A N A
A M A N A B
A B M A N A

② B M A N P A N A M A N A P
N A M A N A
A M A N A B
A B M A N A

③ B M A N P A N A M A N A P
N A M A N A
A M A N A B
A B M A N A

④ B M A N P A N A M A N A P
N A M A N A
A M A N A B
A B M A N A

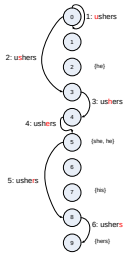
ĐỐI SÁNH MẪU TRÊN CHUỖI

- Bài toán đối sánh mẫu
- Thuật toán đối sánh vét cạn
- Thuật toán Boyer-Moore
- Thuật toán Knuth-Morris-Pratt
- Thuật toán Wu-Manber
- Thuật toán Aho-Corasick

$P = \{p_1, p_2, ..., p_k\}$ là một tập hợp các mẫu, mỗi mẫu là một chuỗi ký tự
$T = t_1 t_2 ... t_m$ là chuỗi ký tự để tìm kiếm các mẫu trong đó và được gọi là văn bản
Các ký tự của mẫu và văn bản được lấy ra từ bảng chữ cái ∑
Bài toán: Xác định các mẫu trên có xuất hiện trong văn bản và nếu có thì xuất hiện bao nhiêu lần và ở vị trí nào
Thuật toán Aho-Corasick (AC) đưa ra lời giải cho bài toán trên với độ phức tạp O(n+m+z) với z là số lần xuất hiện của các mẫu và n là tổng độ dài của các mẫu

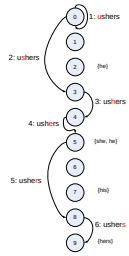
Automaton của Aho-Corasick

- Thuật toán Aho-Corasick hoạt động thông qua việc xây dựng và vận hành một automaton hữu hạn
- Automaton của thuật toán Aho-Corasick là một máy hữu hạn trạng thái (finite-state machine) được xây dựng từ tập mẫu P.
  - Mỗi trạng thái được đại diện bằng một con số.
  - Trạng thái ban đầu thường được đại diện bởi số 0.
  - Hàm chuyển trạng thái $t = \delta(s, \sigma)$ cho biết chuyển đến trạng thái nào từ trạng thái hiện thời và ký tự đầu vào
  - Mỗi trạng thái được liên kết với một tập con của mẫu (có thể là rỗng) cho biết các mẫu của tập con này đã được tìm thấy
- Ví dụ: automaton cho tập mẫu P = {he, she, his, hers}; Văn bản T = ushers



Hoạt động của automaton Aho-Corasick

- Hoạt động của automaton dựa trên các chu trình
  - Chu trình đầu tiên: Bắt đầu từ trạng thái 0, đọc ký tự đầu tiên của văn bản và chuyển sang trạng thái tiếp theo căn cứ trên ký tự này và trạng thái hiện thời
  - Chu trình thứ hai: Đọc ký tự thứ hai của văn bản và chuyển sang trạng thái tiếp theo dựa trên ký tự này và trạng thái hiện thời
  - Lần lượt làm như trên cho đến chu trình cuối cùng: đọc ký tự cuối cùng của văn bản, chuyển đến trạng thái cuối cùng và ngừng hoạt động
  - Khi đến trạng thái liên kết với một tập con khác rỗng của P, in các mẫu của tập con đó ra. Đó là các mẫu xuất hiện trong văn bản
- **Ví dụ:** automaton cho tập mẫu P = {he, she, his, hers}; Văn bản T = ushers

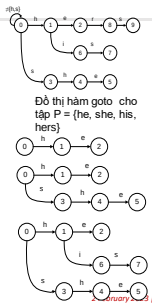


Hàm chuyển trạng thái δ

Trạng thái hiện thời	Ký tự đầu vào	Trạng thái tiếp theo	Trạng thái hiện thời	Ký tự đầu vào	Trạng thái tiếp theo
0	h	1	9, 7, 3	h	4
	s	3		s	3
	.	0		.	0
1	e	2	5, 2	r	8
	i	6		h	1
	h	1		s	3
	s	3		.	0
	.	0	6	s	7
4	e	5		h	1
	i	6		.	0
	h	1	8	s	9
	s	3		h	1
	.	0		.	0

Hàm goto

- Hàm chuyển trạng thái được xây dựng dựa trên hàm goto và hàm failure
- Hàm goto g
  - *g(trạng thái, ký tự đầu vào) = trạng thái*
  - hoặc *g(trạng thái, ký tự đầu vào) = fail.*
- Ví dụ: đồ thị hàm goto cho tập mẫu P = {he, she, his, hers}
  - $g(1, e) = 2$; $g(1, i) = 6$;
  - $g(1, \sigma) = fail$ với $\sigma \neq \{e, i\}$ do đó $g(1,h) = fail$; $g(1, r) = fail$...
  - Riêng $g(0, \sigma) \neq fail$ với mọi σ
- Xây dựng đồ thị hàm goto:
  - Các node tương ứng với các trạng thái, các cạnh tương ứng với các ký tự
  - Bắt đầu với trạng thái 0
  - Lần lượt thêm các mẫu vào đồ thị
  - Mỗi mẫu được thêm bằng cách thêm một đường đi có hướng từ trạng thái đầu tiên sao cho các cạnh của đường đi đó tương ứng với mẫu đó.

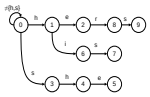


Đồ thị hàm goto cho tập P = {he, she, his, hers}

Hàm failure

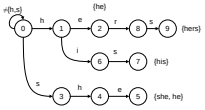
- Hàm failure *f*: f(trạng thái) = trạng thái
- Hàm failure được xây dựng dựa trên hàm goto
- f(s) = 0 cho tất cả các trạng thái kề với trạng thái 0. Ví dụ: f(1) = 0, f(3) = 0.
- Bắt đầu từ node gốc 0, các node được thăm theo phương pháp tìm kiếm theo chiều rộng: các node ở gần gốc hơn được thăm trước
- Giả sử ta muốn tìm giá trị hàm failure cho trạng thái s. Tìm trạng thái r và ký tự a sao cho g(r,a) = s:
  a) Đặt u_1 = f(r), u_2 = f(u_1), u_3 = f(u_2) ... cho đến khi g(u_i, a) ≠ *fail*
  b) Đặt f(s) = g(u_i, a)
- Tính f(2): ta có g(1, e) = 2 nên đặt u_1 = f(1) = 0; vì g(0,e) = 0 ≠ *fail* nên f(2) = 0
- Tính f(5): ta có g(4, e) = 5 nên đặt u_1 = f(4) = 1; vì g(1, e) = 2 ≠ *fail* nên f(5) = 2

i	1	2	3	4	5	6	7	8	9
f(i)	0	0	0	1	2	0	3	0	3



Hàm output

- Hàm output liên kết một trạng thái với một tập con của tập mẫu P (có thể là rỗng) cho biết đã tìm thấy tập con đó trong văn bản
- Hàm output được xây dựng đồng thời với hàm goto. Mẫu kết thúc tại trạng thái nào thì liên kết với trạng thái đó
- Hàm output còn được cập nhật bởi hàm failure. Ví dụ, f(5) = 2 mà output(2) = he nên output(5) = output(5) U output(2) = {she, he}

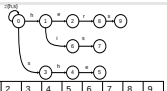


i	output(i)
0	ø
1	ø
2	{he}
3	ø
4	ø
5	{she, he}
6	ø
7	{his}
8	ø
9	{hers}

Xây dựng hàm chuyển trạng thái

- Xây dựng hàm chuyển trạng thái δ từ hàm goto và hàm failure
- Bắt đầu từ node gốc 0, các node được thăm theo phương pháp tìm kiếm theo chiều rộng
- Nếu g(s, a) ≠ *fail* δ(s, a) = g(s, a)
- Nếu không δ(s, a) = δ(f(s), a)
- δ(4, e) = g(4, e) = 5
- δ(4, i) = δ(f(4), i) = δ(1, i) = 6
- δ(4, h) = δ(f(4), h) = δ(1, h) = 1
- δ(4, s) = δ(f(4), s) = δ(1, s) = 3
- δ(4, .) = δ(f(4), .) = δ(1, .) = 0

i	1	2	3	4	5	6	7	8	9
f(i)	0	0	0	1	2	0	3	0	3



Trạng thái hiện thời	Ký tự đầu vào	Trạng thái tiếp theo	Trạng thái hiện thời	Ký tự đầu vào	Trạng thái tiếp theo
0	h	1	3	h	4
	s	3		s	3
	.	0		.	0
1	e	2	4	e	5
	i	6		i	6
	h	1		h	1
	s	3		s	3
	.	0		.	0

So sánh Wu-Manber và Aho-Corasick

	Wu-Manber	Aho-Corasick
Cấu trúc dữ liệu của chữ ký mã độc	Đơn giản, chủ yếu là danh sách liên kết	Phức tạp, chủ yếu là cây tiền tố (trie)
Quá trình tiền xử lý	Đơn giản	Phức tạp
Khả năng quét nhiều mẫu đồng thời	Không hiệu quả bằng Aho-Corasick	Tối ưu cho việc quét nhiều mẫu đồng thời
Format chữ ký mã độc	Thích hợp với các chữ ký dài và cố định	Thích hợp với các chữ ký sử dụng ký tự thay thế (wildcard, có dạng aabbcc*bbaacc)

Bảng so sánh một số tính chất của Wu-Manber và Aho-Corasick