Tạp chí Tin học và Điều khiển học, T. 18, S. 3 (2002), 201–210

TIẾP CẬN MỜ TRONG MỘT SỐ THUẬT TOÁN SO MẪU

NGUYỄN THỊ THANH HUYỀN, PHAN TRUNG HUY

Abstract. In this paper, a fuzzy approach using fuzzy automata in exact string matching algorithms is introduced. Comparing with traditional algorithms, it is shown that there are some remarkable improvements for speed of these. This approach can be extended easily for single- or multi- approximate string matching algorithms, in both cases of one occurrences or all occurences of patterns.

Tóm tắt. Trong bài này, một cách tiếp cận mờ có sử dụng ôtômát mờ trong một số thuật toán so mẫu xâu chính xác được đưa vào giới thiệu. Khi so sánh với những thuật toán truyền thống, các thuật toán này đã cho thấy sự cải thiện đáng kể về tốc độ. Cách tiếp cận này có thể dễ dàng được mở rộng để so mẫu xâu xấp xỉ trong trường hợp đơn mẫu, đa mẫu có tính đến nhiều lần lặp.

1. MỞ ĐẦU

Trong xu thế phát triển của khoa học công nghệ, tìm kiếm mẫu là một vấn đề thời sự bởi ứng dụng của nó trong rất nhiều bài toán mà hiện nay đang được quan tâm nghiên cứu. Chẳng hạn bài toán tìm kiếm tương tự trong các CSDL gen và các vùng bảo tồn gen; tìm kiếm mẫu lặp trong nén dữ liệu; so mẫu kết hợp với logic mờ để nhận dạng tiếng nói, dùng cho các hệ thống điều khiển bằng tiếng nói, nhận dạng từ, dùng cho việc phát hiện và chỉnh lỗi cú pháp chương trình dịch, nhận dạng ảnh, dùng trong viễn thám, xử lý ảnh; so mẫu để phát hiện virút ...

Hiện nay đã có các thuật toán tìm kiếm mẫu kinh điển, được đánh giá tốt, thời gian thực hiện nhanh, như thuật toán Knuth - Morris - Pratt (KMP), Boyer - Moore (BM), Rabin - Kapp,... ([1, 2, 3]) và được sử dụng một cách hiệu quả trong các trình soạn thảo, xử lý văn bản, trình xử lý ảnh. Tuy nhiên, trong các thuật toán này, mức độ trùng khớp với mẫu (khi nhìn theo quan điểm của hệ mờ ta gọi là độ mờ) chưa được phản ánh tức thời. Hơn nữa, tìm kiếm mẫu trong những bài toán thực tế đòi hỏi không chỉ nhanh mà còn phải có sự mềm dẻo, kết quả trả lời không chỉ là những đoạn giống hoàn toàn với mẫu mà cả những đoạn gần giống mẫu. Trong những tình huống như vậy, cách tìm kiếm theo tiếp cận mờ và xấp xỉ là cần thiết. Trong bài này chúng tôi trình bày một cách tiếp cận mờ có sử dụng Ôtômát mờ để tìm kiếm chính xác xâu với nhiều lần lặp. Một số so sánh phân tích với những thuật toán kinh điển như thuật toán KMP, BM cũng được đưa vào xem xét cho thấy sự tiếp cận mờ là mềm dẻo và cải thiện đáng kể tốc độ tìm mẫu trong những dòng dữ liệu lớn. Việc thử nghiệm bằng một chương trình được cài đặt trên các thuật toán của chúng tôi đưa ra những số liệu cụ thể cho thấy thời gian tìm kiếm có thể giảm đi 20% trong nhiều trường hợp với những tệp dữ liệu lớn từ 1000 KB đến trên 100.000KB.

Đồng thời, để tiện cho việc so sánh sau này, chúng tôi cũng trình bày lại một số thuật toán kinh điển. Các thuật toán này giải quyết bài toán cơ bản sau: Cho trước một chuỗi văn bản (kí tự hoặc nhị phân) độ dài N và một mẫu độ dài M, tìm sự xuất hiện của mẫu trong chuỗi.

2. THUẬT TOÁN TÌM NHIỀU LẦN LẶP MẪU THEO TIẾP CẬN MỜ

2.1. Phát biểu bài toán

Có một dòng s các kí tự trên bảng chữ A (có thể là tệp hoặc dòng dữ liệu trên mạng) mà ta cần phải tìm kiếm số lần và vị trí xuất hiện của mẫu p với độ dài M. Bản chất của vấn đề tìm kiếm là duyệt từ trái sang phải và xét từng kí tự xuất hiện trên s.

Yêu cầu khi mỗi lần phát hiện ra sự xuất hiện của mẫu p sau khi đọc xong một kí tự trên s thì thuật toán phải xác định vị trí xuất hiện của mẫu và tăng số lần xuất hiện mẫu lên 1.

2.2. Mô hình của Ôtômát mờ

Việc nghiên cứu Ôtômat mờ và ứng dụng đã được nhiều tác giả quan tâm. Độc giả có thể xem [4].

Giả sử A là bảng chữ cái biểu diễn p và s; p là mẫu độ dài m; A_p là tập các kí tự xuất hiện trong p và tất cả các kí tự còn lại thuộc $A - A_p$, để đơn giản, ta kí hiệu #.

Vậy, không mất tính tổng quát, trong bài toán này ta giả thiết $A = A_p \cup \{\#\}$.

Ta xét Ôtômát mờ $\mathcal{A} = (Q, A, \delta, q_0, F)$, trong đó:

+ Q: tập hữu hạn các trạng thái mờ trên tập nền $X = [1, .., k]$. Mỗi trạng thái mờ là một tập mờ trên X với các giá trị mờ nguyên thuộc đoạn $[0, ..., m]$, ta biểu diễn $q = (v_1, .., v_k)$.
+ Hàm chuyển trạng thái được xác định thông qua hàm TFuzz sẽ được mô tả sau.

  Với $q = (v_1, .., v_k)$, ta có:

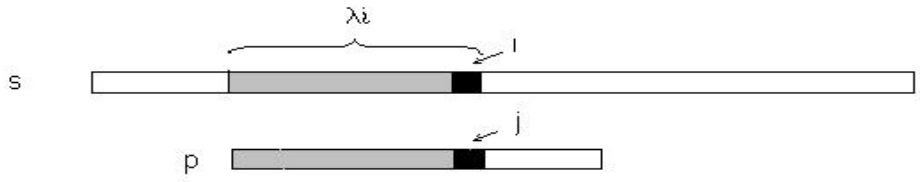
$$\delta(q, a) = q' = (v'_1, ..., v'_k) \text{ mà } \; v'_i = \text{TFuzz}(v_i, a).$$

+ $q_0 = (0, 0, ..., 0)$.
+ F: tập hữu hạn các trạng thái mờ $q_f = (v_1, ..., v_k)$ mà $\exists i : v_i =$ độ dài mẫu i. Trong bài này, trường hợp tìm một mẫu, $k = 1,\ q_0 = 0,\ q_f = m,\ Q = \{0, 1, .., m\}$.

2.3. Xây dựng TFuzz và bản chất thuật toán

2.3.1. Bản chất thuật toán

Khi chúng ta duyệt s từ trái sang phải, bắt đầu từ kí tự thứ $i = 1, 2, ...$ ta sẽ kí hiệu kí tự thứ i là s_i, ta gọi λ_i là độ mờ xuất hiện của mẫu cho tới điểm thứ i trên s. Ở đây, độ mờ được hiểu là độ dài của khúc đầu dài nhất trên p mà đã khớp cho tới điểm s_i (hình 1).



Hình 1

Vậy khi ta nhận được $\lambda_i = m$ thì chứng tỏ có thêm một lần xuất hiện mẫu tại vị trí $i - m + 1$.

2.3.2. Hoạt động của Ôtômát

Khởi đầu Ôtômát từ trạng thái q_0 mỗi khi nhận được một ký tự s_i và Ôtômát đang ở trạng thái q thì ta nhận được trạng thái mới $q' = \delta(q, s_i)$ và nếu $q' \in F$ thì báo có một lần xuất hiện mẫu. Do đó hàm TFuzz sẽ phải được xây dựng một cách thích hợp để phản ánh được bản chất của thuật toán trên.

Ta sẽ sử dụng một ví dụ minh hoạ để dễ dàng hiểu được cách xây dựng bảng TFuzz.

Ví dụ 1. p= aababaab, A= {a, b, #}, A_p= {a, b}.

TFuzz:

Q \ A	a	b	#
0	1	0	0
1	2	0	0
2	2	3	0
3	4	0	0
4	2	5	0
5	6	0	0
6	7	0	0
7	2	8	0
8	4	0	0

2.3.3. Bản chất của TFuzz

Tại trạng thái 0, khi đọc được $s_i = a$ thì nhận được $q_1 = 1$, có nghĩa $\lambda_i = 1, ...$

Chẳng hạn với $q = 4$, nếu đọc được $s_i = a$ thì $q_1 = 2$, điều này nghĩa là $\lambda_{i-1} = 4$ và khi đọc được $s_i = a$ thì $\lambda_i = 2$. Vậy, bản chất của q tại từng thời điểm là phản ánh độ mờ có mẫu trên s tại thời điểm tương ứng.

Vậy có thể hình thức hoá $\delta(\lambda_{i-1}, s_i) = \lambda_i$.

Trường hợp $k = 1$, ở đây ta có: $\lambda_i = \text{TFuzz}(\lambda_{i-1}, s_i)$.

Ví dụ 2. Áp dụng bảng trên, với

p = aababaab

s = aabaababaabababaabaabb

Cách tìm mẫu như sau:

$s =$		a	a	b	a	a	b	a	b	a	a	b	a	b	a	a	b	a	b	a	a	b	b
i		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
λ_i	0	1	2	3	4	2	3	4	5	6	7	8	4	5	6	7	8	4	5	6	7	8	0
												↓					↓					↓	
												ghi nhận					ghi nhận					ghi nhận	
												11-8+1 = 4					16-8+1= 9					21-8+1 =14	

2.4. So sánh với thuật toán KMP

Bản chất của thuật toán này theo khía cạnh tính toán λ_i rất gần với phương pháp tính toán của thuật toán rõ KMP. Do đó, ta sẽ so sánh để thấy sự gần nhau và giải thích lý do của sự cải thiện đáng kể tốc độ theo tiếp cận mờ so với thuật toán KMP.

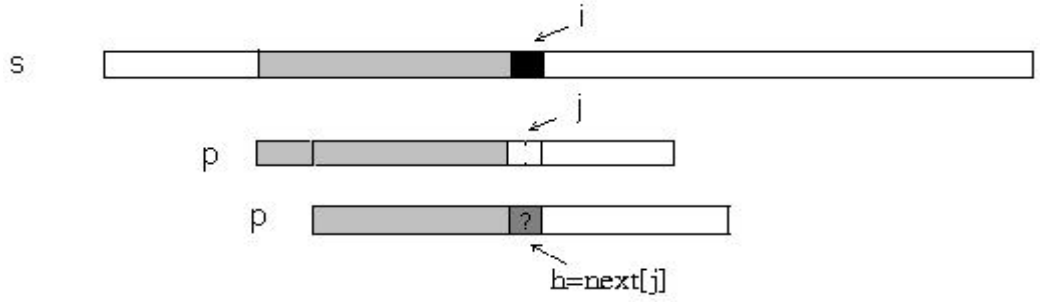
Trước hết, ta nhắc lại thuật toán KMP trong trường hợp tìm một mẫu và tìm nhiều lần lặp mẫu.

2.4.1. Thuật toán KMP tìm một lần lặp mẫu

2.4.1.a. Tư tưởng của phương pháp

Khi đoạn đầu của mẫu p đã khớp với một đoạn trên s_i, giả sử là $p_1p_2...p_{j-1} \equiv s_{i-j+1}s_{i-j+2}...s_{i-1}$, $p_j \neq s_i$ cần phải bắt đầu so mẫu lại từ vị trí $i - h + 1$ nào đó trên văn bản (trường hợp xấu nhất $h = j - 1$ trong thuật toán brute - force). Nếu $\exists h > 0$ sao cho $h - 1$ kí tự đầu của mẫu khớp với $h - 1$ kí tự cuối của đoạn $s_1s_2...s_{i-1}$ (kí hiệu là $s(i-1)$) hay có nghĩa là đã khớp với $h - 1$ kí tự cuối của đoạn $p(j-1)$ thì ta có thể bỏ qua $h - 1$ phép so sánh và tiếp tục so sánh 2 kí tự p_h và s_i (hình 2).

Nếu $s_i \neq p_h$ thì ta phải tiếp tục lùi con trỏ trên mẫu. Để khắc phục nhược điểm do tình huống này gây ra, ta cố gắng tìm h sao cho p_h có nhiều khả năng bằng s_i. Vì $s_i \neq p_j$ nên p_h phải khác p_j. Vì h phụ thuộc vào j, ta kí hiệu $h = \text{next}[j]$. Như vậy, với mỗi j, ta cần xác định next[j] là số h lớn nhất sao cho $h - 1$ kí tự đầu của mẫu khớp với $h - 1$ kí tự cuối của đoạn $p(j-1)$ và $p_j \neq p_{\text{next}[j]}$.



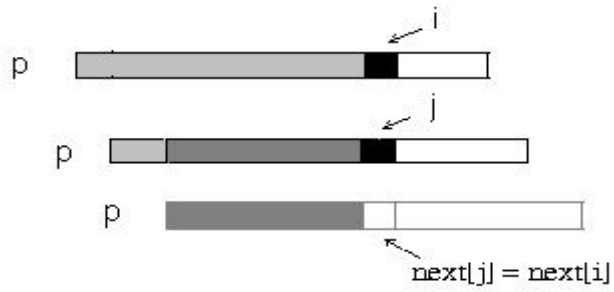
Hình 2

2.4.1.b. Xây dựng bảng next

1) next[1] = 0

2) Tại thời điểm có $p_1p_2...p_{j-1} \equiv p_{i-j+1}...p_{i-1}$ tính next[i]

a) Nếu $p_i \neq p_j$ thì next[i] :=j
b) Nếu $p_i = p_j$, dịch tiếp mẫu đi sang phải sao cho $h-1$ kí tự đầu của mẫu khớp với $h-1$ kí tự cuối của đoạn $p(i-1)$, với h là số lớn nhất, $h < j < i$ có tính chất này, h=next[i]. Nhưng $h-1$ kí tự cuối của đoạn $p(i-1)$ đã khớp với $h-1$ kí tự cuối của đoạn $p(j-1) \rightarrow h = \text{next}[j]$. Vậy next[$i$] = next[$j$] (hình3).



Hình 3

Thuật toán

```
procedure initnext;
var i, j: integer;
begin i := 1; j := 0; next[1] := 0;
    while i < m do
        begin
            while (j > 0) and (p_i ≠ p_j) do j := next[j];
            i := i + 1; j := j + 1;
            if (p_i = p_j) then next[i]:=next[j]
            else next[i]:=j;
        end;
end;
```

Ví dụ 3. Với p = aababaab, có bảng next như sau:

i	1	2	3	4	5	6	7	8
next[i]	0	0	2	0	2	0	0	2

2.4.1.c. Tìm kiếm mẫu dựa trên bảng next

Gọi j là con trỏ trên mẫu p, i là con trỏ trên xâu s
1) Bắt đầu từ đầu xâu và đầu mẫu $i := 1; j := 1$
2) Tìm đoạn trên xâu s khớp với đoạn đầu của mẫu

Giả sử có $p_1p_2...p_j \equiv s_{i-j+1}...s_i$
a) Nếu $j = m$, vị trí xuất hiện mẫu là $i - m + 1$.
b) Nếu $j < m$. Kiểm tra sự khớp nhau ở kí tự tiếp theo $i := i + 1; j := j + 1$
+ Nếu $p_j = s_i$ thực hiện 2a)
+ Nếu $p_j \neq s_i$, tiếp tục lùi con trỏ mẫu để quay về bước 2

Thuật toán KMP

```
function kmp : integer;
var i, j : integer;
begin
    i := 1; j := 1; initnext();
    while (j <= m) and (i <= n) do
        begin
            while (j > 0) and (s_i ≠ p_j) do j:=next[j];
            i := i + 1; j := j + 1;
        end;
```

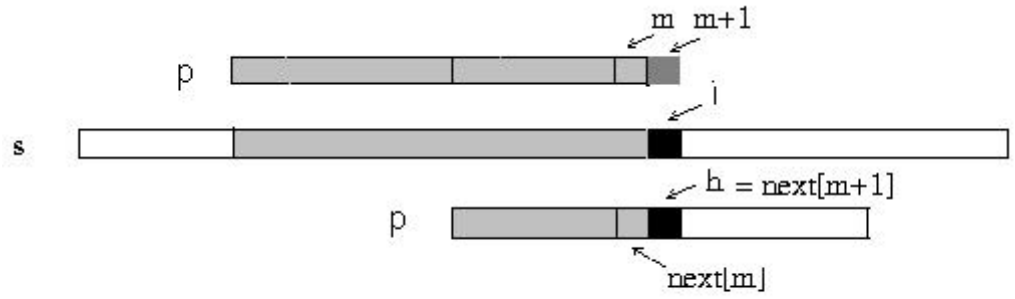
if $j > m$ then kmp:= $i - m$
else kmp:= 0;
end;

2.4.2. Thuật toán KMP tìm nhiều lần lặp mẫu

2.4.2.a. Tư tưởng của thuật toán

Khi sử dụng thuật toán KMP, nếu $j > m$, ta được 1 xuất hiện của mẫu từ vị trí $i - m$. Để tìm xuất hiện tiếp theo, nếu bắt đầu so mẫu từ s_i, ta có thể bỏ qua một số xuất hiện mẫu vì mẫu có thể lồng nhau. Vì vậy ta sẽ xét s_i, trượt mẫu đi một số vị trí sao cho $h - 1$ kí tự đầu của mẫu khớp với $h - 1$ kí tự của cuối của $s(i - 1)$ hay chính là $h - 1$ kí tự cuối của $p(m)$ (hình 4).

Như vậy, ta sẽ mở rộng bảng next trong KMP cho $j = m + 1$



Hình 4

2.4.2.b. Xây dựng bảng next

+ Với $i <= m$, bảng next được xây dựng giống như trong KMP.
+ Với $i = m + 1$, để đơn giản, ta giả sử đang xét một mẫu mở rộng p' gồm $m + 1$ ký tự từ mẫu p nhờ thêm một ký tự giả # không thuộc A_p vào đuôi của p về bên phải, nghĩa là p là khúc đầu của p'. Khi đó việc áp dụng cùng phương pháp KMP để xây dựng p' sẽ cho ta giá trị next$[m + 1]$ thoả mãn: khúc đầu độ dài next$[m + 1]$ của p có khúc đầu gồm next $[m + 1]$-1 ký tự là trùng với khúc đuôi của p, dễ thấy $p'_{m+1} = \#$ thoả mãn $\# = p'_{m+1} <> p_{\text{next}[m+1]}$. Vậy next $[m + 1]$ có thể dễ dàng tính được như sau: điều kiện dừng của vòng lặp While thay $i < m$ bởi $i <= m$ và trong vòng While cần có điều kiện **if** $(i <= m)$ **and** $(p_i = p_j)$ **then ...** thay cho chỉ xét **if** $p_i = p_j$ **then ...** như một lần lặp mẫu.

Thuật toán

```
procedure initnext ();
var i, j : integer;
begin
    i := 1; j := 0; next[1]:= 0;
    while (i <= m) do
        begin
            while (j > 0) and (p_i ≠ p_j) do j:=next[j];
            i := i + 1; j := j + 1;
            if (i <= m) and (p_i = p_j) then next[i] = next[j]
            else next[i]:=j;
        end ;
end;
```

Ví dụ 4. p= aababaab, có bảng next như sau

i	1	2	3	4	5	6	7	8	9
next [i]	0	0	2	0	2	0	0	2	4

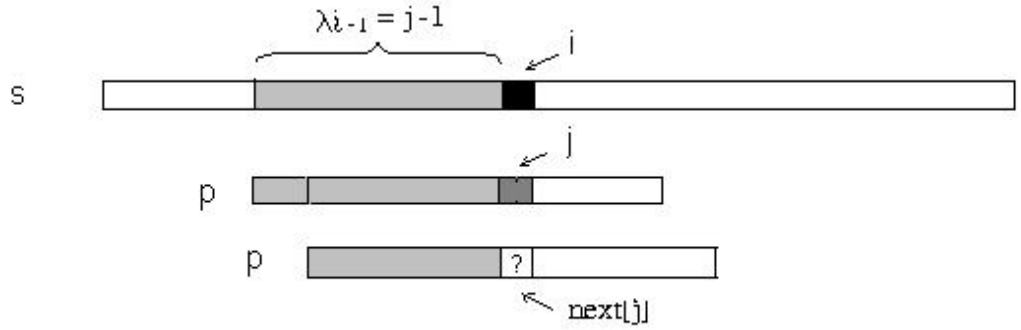
2.4.2.c. Tìm kiếm mẫu dựa trên bảng next

Thuật toán

```
procedure kmp();
var  i, j : integer;
     flag: integer;
begin
     i := 1; j := 1; initnext(); flag:=0;
     repeat
          while (j <= m) and (i <= n) do
               begin
                    while (j > 0) and (s_i ≠ p_j) do j:=next[j] ;
                    i := i + 1; j := j + 1;
               end;
          if (j > m) then
               begin
                    ghi nhận vị trí xuất hiện mẫu i − m;
                    flag:= flag + 1;
               end;
          j:=next[m + 1];
     until i > n;
end;
```

2.4.3. So sánh thuật toán theo tiếp cận mờ và thuật toán KMP

Qua sự trình bày về thuật toán KMP ở trên, ta có thể nhận thấy (hình 5): $j - 1 = \lambda_{i-1}$ nhưng có thể $\lambda_i \neq j$



Hình 5

+ Trường hợp 1: $\lambda_i = j$
  Tăng i, j lên 1. Với trường hợp này tốc độ thao tác của thuật toán như trong cách tiếp cận mờ.
+ Trường hợp 2: $\lambda_i \neq j$
  Ta giữ nguyên con trỏ i trên s và dịch chuyển trên mẫu, nghĩa là dùng lệnh j:=next[j]. Như vậy phải mất thời gian dịch chuyển theo bảng next, thậm chí có thể nhiều lần.

Ví dụ 5. p=aababaab, sử dụng bảng next trong ví dụ 4 để tìm sự xuất hiện của mẫu trong xâu s như sau:

```
                  s=a a c.....                  i = 3
                  p=a a b a b a a b             j = 3
Dịch lần thứ1:        a a b a b a a b           j:=next[j]=2
Dịch lần thứ 2:           a a b a b a a b       j:=next[j]=0
```

Ta thấy có 2 lần dùng j:=next[j] và 2 lần so sánh s_i và p_j. Nói chung, có thể xảy ra nhiều lần dùng next và so sánh s_i với p_j (j khác nhau). Điều này làm chậm đáng kể so với tiếp cận mờ: mỗi lần nhận 1 kí tự s_i là một lần điều chỉnh giá trị mờ theo ôtômat là:

$$\lambda_i = \text{TFuzz}(\lambda_{i-1}, s_i)$$

Lệnh này thực hiện rất nhanh nếu TFuzz được biểu diễn dưới dạng một mảng và được tính toán cẩn thận trước theo thông tin của mẫu p.

Bảng số liệu sau so sánh tốc độ thực hiện việc tìm sự xuất hiện của mẫu p trong tệp dữ liệu lớn s theo 2 thuật toán KMP và tiếp cận mờ trên máy PC IBM tốc độ 233MHz.

	mẫu p	kích thước tệp S	T_{KMP}	T_{Fuzzy}
1)	aababcab	1400 KB	17% s	11%s
2)	MDSVF6V	140.000 KB	35 s	30 s
3)	bacabccaa	1200 KB	16% s	10% s
4)	S068FAB50	140.000 KB	37 s	30 s

2.5. Cài đặt thuật toán tìm nhiều lần lặp mẫu theo tiếp cận mờ

2.5.1. Xây dựng bảng TFuzz

Ta có:

+ $\text{TFuzz}(0, x) = \begin{cases} 0, \forall x \neq p_1 \\ 1, \quad x = p_1 \end{cases}$

+ $\text{TFuzz}(i, \#) = 0, \quad \forall i = 0, 1, ..., m$

+ $\text{TFuzz}(i, x) = \begin{cases} i+1, \quad \text{nếu} x = p_{i+1}, \\ \text{next}[i+1] \text{ sao cho } p_{\text{next}[i+1]} = x, \text{với } x \neq p_{i+1}, \# \end{cases}$
$i = 0, 1, ..., m,$

(bảng next sử dụng ở đây được xây dựng như trong thuật toán KMP tìm mọi xuất hiện của mẫu).

Gọi $A[0...k]$ là mảng lưu giữ bảng chữ cái A của ôtômát, $A = A_P \cup \{\#\}$. Mảng được sắp theo chiều tăng của kí tự và $A[k] = \#$. Để thuận tiện khi truy nhập đến các chữ cái trong bảng chữ cái, ta sử dụng mảng index xác định vị trí của chữ trong bảng:

$$\text{index}[c] = \begin{cases} i, \text{nếu } c = A[i] \\ k, \text{nếu} c \notin \{A[0], A[1], ..., A[k-1]\} \end{cases}$$

TFuzz là mảng $[0...m, 0...k]$, với TFuzz $[i, j]$=độ mờ mớii khi độ mờ i gặp kí tự x có index[x]=j.

Thuật toán xây dựng bảng TFuzz

```
procedure initTFuzz();
var i, j, t: integer;
begin
    for i := 0 to m do
        TFuzz[i, m]:=0;
    for j:=0 to k do TFuzz[0, j]:=0;
    TFuzz[0, index[p1]] = 1;
    for i:=1 to m do
        for t:=0 to k-1 do
            begin
                if i = m then j:=next[i + 1]
                else j := i + 1;
                while (j > 0) and (pj ≠ A[t]) do j:=next[j];
                TFuzz[i, index[A[t]]:=j;
            end;
end;
```

2.5.2. Thuật toán tìm kiếm mẫu dựa vào bảng TFuzz

```
procedure FPM();
var i, flag: integer; fuz: array[1..50] of integer; {* độ mờ*}
begin
        i:= 1; flag := 0; fuz[0] := 0;
        while (còn đọc được s_i) do
            begin
                fuz[i] = TFuzz[fuz[i-1], index[s_i]];
    {* độ mờ mới = TFuzz[độ mờ cũ, kí tự s_i đọc được] *}
        if fuz[i] = m then
            begin      flag := flag + 1;
                ghi nhận vị trí i − m + 1
            end;
        end; {while}
        if flag = 0 then ghi nhận 0;
end;
```

3. TIẾP CẬN TỔNG QUÁT

Hiện nay có rất nhiều tác giả đã xem xét các phương pháp so mẫu chính xác theo một số tiếp cận chính: ngoài thuật toán KMP theo hướng duyệt mẫu từ trái sang phải còn có cách tiếp cận theo Boyer Moore (BM) theo hướng duyệt từ phải sang trái (Turn Boyer-Moore Algorithms) hoặc từ những điểm nào đó giữa mẫu. Tương ứng như vậy, trong tiếp cận mờ cũng có nhiều cách xem xét. Các cách xem xét này đều có thể được xét bởi một cách nhìn thống nhất theo sơ đồ áp dụng ôtômát mờ sau đây. Do mục tiêu của bài báo, chúng tôi chỉ xét trường hợp tìm kiếm chính xác tất cả các lần lặp với đơn mẫu, tuy nhiên cách tiếp cận cho đa mẫu có thể dễ dàng phát triển không khó khăn, và cũng có thể phát triển thêm cho các phương pháp tìm mẫu xấp xỉ đơn và đa mẫu.

Cho mẫu P là xâu mẫu độ dài m trên bảng chữ A, A_p là bảng chữ các ký tự xuất hiện trong P.

3.1 Ôtômát mờ tìm mẫu P

Là bộ $\mathcal{A} = (Q, A^k, f, q_0, F = \{q_f\})$, trong đó:

i) A^k là bảng ký tự vào, mỗi ký tự được xem là một xâu k ký tự trên $A, k = \varphi(m)$ là hàm theo đối m.

ii) Q là tập hữu hạn các trạng thái , mỗi trạng thái có dạng một cặp $(n1, n2)$ các số tự nhiên, $0 \le n1, n2 \le k+1$, $n1$ gọi là *độ mờ của thời điểm đã xét*, A_2 gọi là bước nhảy tiếp theo thời điểm đã xét. $q_0 = (0, 1), q_f = (m, x)$ với x là số tự nhiên tuỳ ý, $0 \le x \le k+1$

iii) Giả sử ta có một hàm xác định *độ mờ* của mẫu khi so sánh *ký tự* u $(u \in A^k)$ với mẫu P là $\text{Fuz}(q, u, P)$ thể hiện *độ mờ có mẫu* P khi xét tới tác động gây ra bởi u. Giả sử $\text{Jump}(q, u, P)$ là một hàm tính toán bước nhảy tiếp theo (trên xâu đích S nào đó chứa u từ trái qua phải kể từ điểm đầu xâu u, sau khi sánh mẫu P với u để có thể nhận u mới trên S sao cho sự xuất hiện của P chỉ có thể xảy ra với u mới và bước nhảy không gây ra khả năng bỏ sót một sự xuất hiện nào của P). Khi đó ta có thể hiện của hàm chuyển trạng thái f:
$f(q, u) = q'$, ở đó q'=(Fuz(q, u, P), Jump(q, u, P)) được tính từ $q = (n1, n2), u$ và P.

3.2. Hoạt động của Ôtômát

Giả sử ta có xâu đích S trên bảng chữ A và xâu mẫu P. Ta cần sử dụng ôtômát $\mathcal{A}$ để đoán nhận số lần có mẫu P trong S. Bắt đầu khởi đầu từ q_0 và duyệt từ đầu trái xâu đích S sang phải, xét u là khúc đầu độ dài $k = \varphi(m)$ xem như một ký tự của A^k, khi sánh với P, kết quả tính toán $f(q_0, u) = q' = (n1', n2')$ cho ta biết độ mờ $n1'$ của mẫu P khi xét tới u. Nếu $n1' = m$ nghĩa là $q' \in F$ và cho thấy đã gặp mẫu. Giá trị $n2'$ cho biết sau khi sánh u và P, cần thực hiện bước nhảy $n2'$ kể từ vị trí đầu của u trên xâu đích S để tới vị trí đầu xâu con mới u' của S. Quá trình lại sánh

u' và P thực hiện bởi $f(q', u') = q''$, và quá trình cứ thế tiếp diễn... cho tới khi không thể lấy được các xâu con u trên S độ dài đủ k thì dừng.

3.3. Một số ví dụ

Ví dụ 6. Trường hợp so mẫu mờ theo tiếp cận xét trong mục 2 ở trên, dễ dàng kiểm tra là trường hợp riêng của phương pháp xét ở đây với $k = m$ và bước nhảy jump luôn là 1. Hàm Fuz(q, u, P) = TFuzz($n1$,index[$u1$]) trong đó $q = (n1, n2), u1$ là chữ đầu của u, là một chữ trên bảng chữ A và nằm trên xâu đích S.

Ví dụ 7. Trường hợp thuật toán kiểu xét ngược Boyer-Moore, $k = m$ và Fuz(q, u, P) được xét như khúc đuôi dài nhất trùng nhau của u và P không phụ thuộc q, còn bước nhảy Jump(q, u, P) cũng không phụ thuộc q và chỉ cần thoả mãn yêu cầu không bỏ sót sự xuất hiện của mẫu P. Về chi tiết đề nghị độc giả có thể suy luận và tính toán như cách tiếp cận mờ ở mục 2 kết hợp tinh thần thuật toán Boyer-Moore, chẳng hạn theo [AV,CL]. Cài đặt thử nghiệm đã cho thấy trên file lớn cỡ 140MB một mẫu dài 8 ký tự thời gian tìm kiếm là cỡ 26s trong khi đó thuật toán tiếp cận mờ kiểu 1 nêu trong mục 2.5 mất 30s, thuật toán KMP mất 35s.

Ví dụ 8. Ví dụ sau đây đưa vào một thuật toán mới rất nhanh dựa trên sơ đồ ôtômát ở trên có kế thừa cách tiếp cận mờ ở mục 2. Số liệu chạy thử trên tệp cỡ140 000 KB mất thời gian khoảng 26s với mẫu có độ dài từ 8 đến 14.

Xây dựng ôtômát:

a) Hệ số $k = m + 1, q_0 = (0, 1)$

b) Dựa vào hàm TFuzz(n,index[a]) ta biểu diễn tác động $f(q, u) = (n1', n2')$ của ôtômát mới như sau: giả sử $q = (n1, n2)$. Khi đó

i) Tính $n1'$ = TFuzz($n1$,index[u_1]),

ii) Nếu $n1' = m$ hoặc $n1' = n1 + 1$ thì $n2' = 1$

iii) Còn trái lại {nghĩa là $n1' \leq n1$ và $n1' < m$} thì xét:

Nếu $u_{2+m-n1'} \in A_P$, thì $n2' = 1$,

Còn trái lại đặt $n2' = m + 2 - n1'$ và sau đó $n1' = 0$;

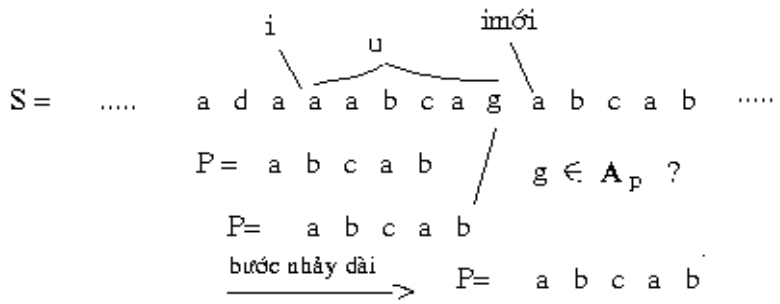
iv) Fuz(q, u, P)=$n1'$, Jump(q, u, P)=$n2'$

c) Hoạt động của otomat trên: Để tìm số lần và vị trí xuất hiện của P trong xâu đích S, khởi đầu biến trỏ i trên $S : i = 1$, khởi đầu $u = u_0 =$ khúc đầu $m + 1$ ký tự của S xem như *ký tự* khởi đầu thuộc A^k. Khởi đầu $q = q_0 = (0, 1)$. Để không mất thông tin về P, ta xem rằng đã bổ sung m ký tự # vào cuối S. Ta bắt đầu tính $f(q, u)$

Vòng lặp tổng quát: tính $q' = (n1', n2') = f(q, u)$, nếu $n1' = m$ kết luận tăng thêm một xuất hiện của P trên S và vị trí cuối của xuất hiện đó là ở vị trí i trên S, $n2'$ chỉ ra bước nhảy tiếp của i:

imới:=$i + n2'$; umới=khúc con $m + 1$ ký tự của S kể từ vị trí imới.

Tiếp tục xét f(qmới,umới) và cứ thế cho tới khi không thể lấy được umới đủ $m + 1$ ký tự thì dừng. Việc thêm m ký tự giả # vào cuối xâu đích S đảm bảo trường hợp chữ đầu của xâu umới nếu đủ $m + 1$ ký tự, luôn luôn là một ký tự của xâu S do đó phép tính $n1'$= TFuzz($n1$,index[u_1]) ở bước i) không bỏ qua việc xét mẫu P trên S lần cuối. Ta có thể kiểm tra rằng trong trường hợp S lớn, A so với A_P nhiều lần lớn hơn thì xác suất của khả năng $n1 = n1' = 0$ và $n2' = m + 1$ là lớn, nghĩa là xảy ra nhiều bước nhảy dài không cần tính, hơn nữa nếu m khá lớn thì độ dài bước nhảy càng lớn, *điều này cải thiện hơn nhiều so với thuật toán tiếp cận mờ ở mục 2, vì ở đó luôn luôn chỉ xét bước nhảy là 1.* Kiểu duyệt vẫn là hướng left-to-right. Đây là một ví dụ về ưu điểm của việc xét dạng tổng quát ở mục này. trên hình 6 minh hoạ một tình huống như vậy.



Hình 6

Qua một số ví dụ trên, bằng mô hình đã xét, ta còn có thể xây dựng nhiều dạng khác nhau cho những ứng dụng, cả tìm kiếm chính xác hay xấp xỉ, đơn hay đa mẫu.

TÀI LIỆU THAM KHẢO

[1] Alfred V. Aho, Algorithms for Finding Pattern in Strings, *Handbook of Theoretical Computer Science*, Elsevier, Amsterdam. J.V. Leeuwen Ed., 1990, 257-299.

[2] Chritsian Charras, Thierry Lecroq, *Hanbook of Exact String- matching Algorithms*, Book online 2002.

[3] Robert Sedgewick, *Algorithms*, 2nd Edition, Addition-Wesley Publishing Co. Bản dịch tiếng Việt: *Cẩm nang thuật toán*, Tập **1**: Các thuật toán thông dụng, NXB KHKT 1994.

[4] Mordeson John N., Malik Davender S., *Fuzzy Automata and Languages: Theory and Application*, ISBN 1-58488-225-5 Publication date: 3/25/2002.

Khoa Toán ứng dụng, Đại học Bách Khoa Hà Nội.

Nhận bài ngày 15-4-2002
Nhận lại sau khi sửa ngày 30-4-2002