CHUYÊN ĐỀ CNPM

CÁC THUẬT TOÁN ĐỐI SÁNH MẪU

(PATTERN MATCHING ALGORITHMS)

1. Giới thiệu vấn đề

Đối sánh xâu (String matching) là một chủ đề quan trọng trong lĩnh vực xử lý văn bản. Các thuật toán đối sánh xâu được xem là những thành phần cơ sở được cài đặt cho các hệ thống thực tế đang tồn tại trong hầu hết các hệ điều hành. Hơn thế nữa, các thuật toán đối sánh xâu cung cấp các mô hình cho nhiều lĩnh vực khác nhau của khoa học máy tính: xử lý ảnh, xử lý ngôn ngữ tự nhiên, tin sinh học và thiết kế phần mềm.

String-matching được hiểu là việc tìm một hoặc nhiều xâu mẫu (pattern) xuất hiện trong một văn bản (có thể là rất dài). Ký hiệu xâu mẫu hay xâu cần tìm là $X =(x_0, x_1,..,x_{m-1})$ có độ dài m. Văn bản $Y =(y_0, y_1,..,y_{n-1})$ có độ dài n. Cả hai xâu được xây dựng từ một tập hữu hạn các ký tự Alphabet ký hiệu là Σ với kích cỡ là σ. Như vậy một xâu nhị phân có độ dài n ứng dụng trong mật mã học cũng được xem là một mẫu. Một chuỗi các ký tự ABD độ dài m biểu diễn các chuỗi AND cũng là một mẫu.

Input:

- Xâu mẫu X =(x0, x1,.., xm), độ dài m.
- Văn bản Y =(y0, x1,.., yn), độ dài n.

Output:

- Tất cả vị trí xuất hiện của X trong Y.

Phân loại các thuật toán đối sánh mẫu

Thuật toán đối sánh mẫu đầu tiên được đề xuất là Brute-Force. Thuật toán xác định vị trí xuất hiện của X trong Y với thời gian O(m.n). Nhiều cải tiến khác nhau của thuật toán Brute-Force đã được đề xuất nhằm cải thiện tốc độ tìm kiếm mẫu. Ta có thể phân loại các thuật toán tìm kiếm mẫu thành các lớp:

- **Tìm kiếm mẫu từ bên trái qua bên phải**: Harrison Algorithm, Karp-Rabin Algorithm, Morris-Pratt Algorithm, Knuth- Morris-Pratt Algorithm, Forward Dawg Matching algorithm , Apostolico-Crochemore algorithm, Naive algorithm.

- **Tìm kiếm mẫu từ bên phải qua bên trái**: Boyer-Moore Algorithm , Turbo BM Algorithm, Colussi Algorithm, Sunday Algorithm, Reverse Factorand Algorithm, Turbo Reverse Factor, Zhu and Takaoka and Berry-Ravindran Algorithms.

- **Tìm kiếm mẫu từ một vị trí cụ thể**: Two Way Algorithm, Colussi Algorithm , Galil-Giancarlo Algorithm, Sunday's Optimal Mismatch Algorithm, Maximal Shift Algorithm, Skip Search, KMP Skip Search and Alpha Skip Search Algorithms.

- **Tìm kiếm mẫu từ bất kỳ**: Horspool Algorithm, Boyer-Moore Algorithm, Smith Algorithm , Raita Algorithm.

Một số khái niệm và định nghĩa cơ bản về tìm kiếm mẫu:

Giả sử Alphabet là tập hợp (hoặc tập con) các mã ASSCI. Một từ w =(w_0, w_1,..,w_l) có độ dài *l*, w_l =null giống như biểu diễn của ngôn ngữ C. Khi đó ta định nghĩa một số thuật ngữ sau:

- *Prefix* (*tiền tố*). Từ *u* được gọi là tiền tố của từ *w* nếu tồn tại một từ *v* để *w* =*uv* (*v* có thể là rỗng). Ví dụ: *u* =“**AB**” là tiền tố của *w* =“AB**CDEF**” và *u* =“**com**” là tiền tố của *w* =“com**munication**”.

- *Suffix* (*hậu tố*). Từ *v* được gọi là hậu tố của từ *w* nếu tồn tại một từ *u* để *w* =*uv* (*u* có thể là rỗng). Ví dụ: *v* =“**EF**” là hậu tố của *w* =“AB**CDEF**” và v =“**tion**” là hậu tố của w =“com**munication**”.

- *Factor* (substring, subword). Một từ *z* được gọi là một xâu con, từ con hay nhân tố của từ *w* nếu tồn tại hai từ *u*, *v* (u, v có thể rỗng) sao cho *w* = *uzv*. Ví dụ từ z =“CD” là factor của từ *w* =“AB**CD**EF” và z =“**muni**” là factor của w =“com**muni**cation”.

- *Period* (đoạn). Một số tự nhiên *p* được gọi là đoạn của từ *w* nếu với mọi i (0<=i< m-1) thì w[i] = w[i+p]. Giá trị đoạn nhỏ nhất của w được gọi là đoạn của w ký hiệu là pre(w). Ví dụ w =“ABABCDEF”, khi đó tồn tại p=2.

- *Periodic* (tuần hoàn). Từ w được gọi là tuần hoàn nếu đoạn của từ nhỏ hơn hoặc bằng l/2. Trường hợp ngược lại được gọi là không tuần hoàn. Ví dụ từ w=“ABAB” là từ tuần hoàn. Từ w =“ABABCDEF” là không tuần hoàn.

Một số khái niệm và định nghĩa cơ bản về tìm kiếm mẫu:

•*Basic word* (từ cơ sở). Từ *w* được gọi là từ cơ sở nếu nó không thể viết như lũy thừa của một từ khác. Không tồn tại z và k để z^k = w.

• Boder word (từ biên). Từ z được gọi là boder của w nếu tồn tại hai từ u, v sao cho w = uz =zv. Khi đó z vừa là tiền tố vừa là hậu tố của w. Trong tình huống này |u| = |v| là một đoạn của w.

• *Reverse word* (*Từ đảo).* Từ đảo của từ w có độ dài l ký hiệu là $w^R=(w_{r-1},w_{r-2},\ldots, w_1,w_0)$.

• Deterministic Finite Automata (DFA). Một automat hữu hạn A là bộ bốn (Q, q0, T, E) trong đó:

- Q là tập hữu hạn các trạng thái.
- q0 là trạng thái khởi đầu.
- T là tập con của Q là tập trạng thái dừng.
- E là tập con của (Q,Σ,T) tập các chuyển dịch.

Ngôn ngữ L(A) đoán nhận bởi A được định nghĩa:

$$\{w \in \Sigma^* : \exists q_0, \ldots, q_n, n = |w|, q_n \in T, \forall 0 \leq i < n, (q_i, w[i], q_{i+1}) \in \delta\}$$

Ví dụ . Thuật toán Brute-Force

Đặc điểm:

- Không có pha tiền xử lý.
- Sử dụng không gian nhớ phụ hằng số.
- Quá trình so sánh thực hiện theo bất kỳ thứ tự nào.
- Độ phức tạp thuật toán là O(n.m);

Thuật toán Brute-Force:

Input :

- Xâu mẫu X =(x0, x1,..,xm), độ dài m.
- Văn bàn nguồn Y=(y1, y2,..,yn) độ dài n.

Output:

- Mọi vị trí xuất hiện của X trong Y.

Formats: Brute-Force(X, m, Y, n);

Actions:

```
for ( j = 0; j <= (n-m); j++) { //duyệt từ trái qua phải xâu X
        for (i =0; i<m && X[i] == Y[i+j]; i++) ; //Kiểm tra mẫu
        if (i>=m) OUTPUT (j);
}
```

EndActions.

Ví dụ. Thuật toán Knuth-Morris-Pratt

Đặc điểm:

- Thực hiện từ trái sang phải.
- Có pha tiền xử lý với độ phức tạp O(m).
- Độ phức tạp thuật toán là O(n + m);

Thuật toán PreKmp: //thực hiện bước tiền xử lý

Input :

- Xâu mẫu X =(x0, x1,..,xm), độ dài m.

Output: Mảng giá trị kmpNext[].

Formats:

- PreKmp(X, m, kmpNext);

Actions:

```
i = 1; kmpNext[0] = 0; len = 0; //kmpNex[0] luôn là 0
while (i < m) {
        if (X[i] == X[len] ) { //Nếu X[i] = X[len]
                len++;  kmpNext[i] = len; i++;
        }
        else  { // Nếu X[i] != X[len]
                if ( len != 0 ) {  len = kmpNext[len-1]; }
                else {  kmpNext[i] = 0; i++; }
        }
}
```

EndActions.

Kiểm nghiệm PreKmp (X, m, kmpNext):

- X[] = “ABABCABAB”, m = 9.

i=?	(X[i]== X[len])?	Len =?	kmpNext[i]=?
		Len =0	kmpNext[0]=0
i=1	(‘B’==‘A’): No	Len =0	kmpNext[1]=0
i=2	(‘A’==‘A’): Yes	Len =1	kmpNext[2]=1
i=3	(‘B’==‘B’): Yes	Len=2	kmpNext[3]=2
i=4	(‘C’==‘A’): No	Len=0	kmpNext[4]=0
i=5	(‘A’==‘A’): Yes	Len=1	kmpNext[5]=1
i=6	(‘B’==‘B’): Yes	Len=2	kmpNext[6]=2
i=7	(‘A’==‘A’): Yes	Len=3	kmpNext[6]=3
i=8	(‘B’==‘B’): Yes	Len=4	kmpNext[6]=4
Kết luận: kmpNext[] = {0, 0, 1, 2, 0, 1, 2, 3, 4}.			

Kiểm nghiệm PreKmp (X, m, kmpNext) với X[] = “AABAACAABAA”, m = 11.

i=?	(X[i]== X[len])?	Len =?	kmpNext[i]=?
		Len =0	kmpNext[0]=0
i=1	(‘A’==‘A’): Yes	Len =1	kmpNext[1]=1
i=2	(‘B’==‘A’): No	Len =0	kmpNext[2]=chưa xác định
i=2	(‘B’==‘A’): No	Len=0	kmpNext[2]=0
i=3	(‘A’==‘A’): Yes	Len=1	kmpNext[3]=1
i=4	(‘A’==‘A’): Yes	Len=2	kmpNext[4]=2
i=5	(‘C’==‘B’): No	Len=1	kmpNext[5]=chưa xác định
i=5	(‘C’==‘A’): No	Len=0	kmpNext[5]=chưa xác định
i=5	(‘C’==‘A’): No	Len=0	kmpNext[5]=0
i=6	(‘A’==‘A’): Yes	Len =1	kmpNext[6]=1
i=7	(‘A’==‘A’): Yes	Len =2	kmpNext[7]=2
i=8	(‘B’=‘B’):Yes	Len=3	kmpNext[8] = 3
i=9	(‘A’=‘A’):Yes	Len=4	kmpNext[9] = 4
i=10	(‘A’=‘A’):Yes	Len=5	kmpNext[10] = 5
Kết luận: kmpNext = {0, 1, 0, 1, 2, 0, 1, 2, 3, 4, 5}			

Thuật toán Knuth-Morris-Partt:

Input :

- Xâu mẫu X =(x0, x1,..,xm), độ dài m.
- Văn bản Y =(y0, y1,..,xn), độ dài n.

Output:

- Tất cả vị trí xuất hiện X trong Y.

Formats: Knuth-Morris-Partt(X, m, Y, n);

Actions:

Bước 1(Tiền xử lý):

preKmp(x, m, kmpNext); //Tiền xử lý với độ phức tạp O(m)

Bước 2 (Lặp):

```
i = 0; j = 0;
while (i < n) {
        if ( X[j] == Y[i] ) { i++; j ++; }
        if ( i == m ) {
                < Tìm thấy mẫu ở vị trí i-j>;
                j = kmpNext[j-1];
        }
        else if (i <n && X[j] != Y[i] ) {
                if (j !=0) j = kmpNext[ j-1];
                else i = i +1;
        }
}
```

EndActions.

Kiểm nghiệm Knuth-Moriss-Patt (X, m, Y, n):

- X[] = “ABABCABAB”, m = 9.
- Y[] = “ABABDABACDABABCABAB”, n = 19

Bước 1 (Tiền xử lý). Thực hiện Prekmp(X, m, kmpNext) ta nhận được:

kmpNext[] = { 0, 0, 1, 2, 0, 1, 2, 3, 4}

Bước 2 (Lặp):

(X[i]==Y[i])?	(J ==9)?	I =? J = ?	
(X[0]==Y[0]): Yes	No	i=1, j=1	
(X[1]==Y[1]): Yes	No	i=2, j=2	
(X[2]==Y[2]): Yes	No	i=3, j=3	
(X[3]==Y[3]): Yes	No	i=4, j=4	
(X[4]==Y[4]): No	No	i=4, j=2	
(X[2]==Y[4]): No	No	i=4, j=0	
(X[0]==Y[4]): No	No	i=5, j=0	
(X[0]==Y[5]): Yes	No	i=6, j=1	
(X[1]==Y[6]): Yes	No	i=7, j=2	
.....			

2. BÁO CÁO CHUYÊN ĐỀ: Biểu diễn, đánh giá và cài đặt những thuật toán tìm kiếm mẫu dưới đây.

1. Thuật toán Knuth-Morris-Partt
2. Thuật toán Karp-Rabin
3. Thuật toán Shift or
4. Thuật toán Morris-Partt
5. Thuật toán Automat hữu hạn
6. Thuật toán Simon
7. Thuật toán Colussi
8. Thuật toán Galil-Giancarlo
9. Apostolico-Crochemore algorithm
10. Not So Naive algorithm
11. Turbo BM algorithm
12. Apostolico-Giancarlo algorithm
13. Reverse Colussi algorithm
14. Reverse Colussi algorithm
15. Horspool algorithm
16. Quick Search algorithm
17. Tuned Boyer-Moore algorithm
18. Zhu-Takaoka algorithm
19. Berry-Ravindran algorithm
20. Smith algorithm
21. Raita algorithm
22. Reverse Factor algorithm
23. Turbo Reverse Factor algorithm
24. Forward Dawg Matching algorithm
25. Backward Nondeterministic Dawg 2.38. Matching algorithm
26. Backward Oracle Matching algorithm

27 Galil-Seiferas algorithm

28. Two Way algorithm
29. String Matching on Ordered
30. Alphabets algorithm
31. Optimal Mismatch algorithm
32. Maximal Shift algorithm
33. Skip Search algorithm
34. KMP Skip Search algorithm
35. Alpha Skip Search algorithm

2.NỘI DUNG BÁO CÁO CHUYÊN ĐỀ

1. **Tổng quan: Sắp xếp lại các thuật toán trên thành 4 loại**:
   - Tìm kiếm mẫu từ trái sang phải.
   - Tìm kiếm mẫu từ phải sang trái.
   - Tìm kiếm mẫu từ vị trí xác định.
   - Tìm kiếm mẫu từ vị trí bất kỳ.

2. Tìm kiếm mẫu từ trái sang phải

Với mỗi thuật toán ta cần trình bày:

- Trình bày thuật toán.
- Đánh giá độ phức tạp thuật toán.
- Kiểm nghiệm thuật toán.
- Lập trình theo thuật toán.

3. Tìm kiếm mẫu từ phải sang trái

Với mỗi thuật toán ta cần trình bày:

- Trình bày thuật toán.
- Đánh giá độ phức tạp thuật toán.
- Kiểm nghiệm thuật toán.
- Lập trình theo thuật toán.

4. Tìm kiếm mẫu từ vị trí xác định

Với mỗi thuật toán ta cần trình bày:

- Trình bày thuật toán.
- Đánh giá độ phức tạp thuật toán.
- Kiểm nghiệm thuật toán.
- Lập trình theo thuật toán.

5. Tìm kiếm mẫu từ vị trí xác định

Với mỗi thuật toán ta cần trình bày:

- Trình bày thuật toán.
- Đánh giá độ phức tạp thuật toán.
- Kiểm nghiệm thuật toán.
- Lập trình theo thuật toán.