|  |  |
| --- | --- |
| **HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**  **KHOA QUỐC TẾ VÀ SAU ĐẠI HỌC**  **---------------------------------------** | |
|  |  |
|  | |
|  | |
|  | |
|  | |
| **THUẬT TOÁN NÂNG CAO** | |
|  | |
|  | |
|  | |
| **Giảng viên: TS Nguyễn Duy Phương** | |
| **Học viên: B25CHHT056 - Vũ Minh Toàn**  **Lớp: M25DCHT01-B** | |
|  | |
|  | |
| HÀ NỘI – 2025 | |

**MỤC LỤC**

[1. THUẬT TOÁN BRUTE FORCE 6](#_Toc429383406)

[1.1. Phát biểu thuật toán 6](#_Toc429383407)

[1.2. Mô tả thuật toán 6](#_Toc429383408)

[1.3. Mã hóa thuật toán 6](#_Toc429383409)

[1.4. Kiểm nghiệm thuật toán 7](#_Toc429383410)

[2. THUẬT TOÁN TÌM KIẾM VỚI MỘT AUTOMAT 9](#_Toc429383411)

[2.1. Đặc điểm chính 9](#_Toc429383412)

[2.2. Mô tả thuật toán 9](#_Toc429383413)

[2.3. Mã hóa thuật toán 9](#_Toc429383414)

[2.4. Kiểm nghiệm thuật toán 10](#_Toc429383415)

[3. THUẬT TOÁN KARP-RABIN 14](#_Toc429383416)

[3.1. Đặc điểm 14](#_Toc429383417)

[3.2. Hàm băm cơ bản 14](#_Toc429383418)

[3.3. Mô tả thuật toán 14](#_Toc429383419)

[3.4. Mã hóa thuật toán 15](#_Toc429383420)

[3.5. Kiểm nghiệmthuật toán 16](#_Toc429383421)

[4. THUẬT TOÁN SHITF-OR 17](#_Toc429383422)

[4.1. Đặc điểm 17](#_Toc429383423)

[4.2. Mô tả thuật toán 17](#_Toc429383424)

[4.3. Mã hóa thuật toán 17](#_Toc429383425)

[4.4. Kiểm nghiệm thuật toán 18](#_Toc429383426)

[5. THUẬT TOÁN MORRIS – PRATT 19](#_Toc429383427)

[5.1. Đặc điểm chính 19](#_Toc429383428)

[5.2. Mô tả thuật toán 19](#_Toc429383429)

[5.3. Mã hóa thuật toán 20](#_Toc429383430)

[5.4. Kiểm nghiệm thuật toán 20](#_Toc429383431)

[6. THUẬT TOÁN KNUTH-MORRIS-PRATT 23](#_Toc429383432)

[6.1. Trình bày thuật toán 23](#_Toc429383433)

[6.2. Đặc điểm 23](#_Toc429383434)

[6.3. Mô tả thuật toán 23](#_Toc429383435)

[6.4. Mã hóa thuật toán 24](#_Toc429383436)

[6.5. Kiểm nghiệm thuật toán 24](#_Toc429383437)

[7. THUẬT TOÁN SIMON 26](#_Toc429383438)

[7.1. Các đặc điểm chính 26](#_Toc429383439)

[7.2. Mô tả thuật toán 26](#_Toc429383440)

[7.3. Mã hóa thuật toán 27](#_Toc429383441)

[7.4. Kiểm nghiệm thuật toán 29](#_Toc429383442)

[8. THUẬT TOÁN COLUSSI 31](#_Toc429383443)

[8.1. Đặc điểm 31](#_Toc429383444)

[8.2. Mô tả thuật toán 31](#_Toc429383445)

[8.3. Mã hóa thuật toán 33](#_Toc429383446)

[8.4. Kiểm nghiệm thuật toán 36](#_Toc429383447)

[9. THUẬT TOÁN GALIL-GIANCARLO 39](#_Toc429383448)

[9.1. Các đặc điểm chính 39](#_Toc429383449)

[9.2. Mô tả thuật toán 39](#_Toc429383450)

[9.3. Mã hóa thuật toán 40](#_Toc429383451)

[9.4. Kiểm nghiệm thuật toán 42](#_Toc429383452)

[10. THUẬT TOÁN APOSTOLICO - CROCHEMORE 45](#_Toc429383453)

[10.1. Đặc điểm: 45](#_Toc429383454)

[10.2. Mô tả thuật toán 45](#_Toc429383455)

[10.3. Mã hóa thuật toán 46](#_Toc429383456)

[10.4. Kiểm nghiệm thuật toán 47](#_Toc429383457)

[11. THUẬT TOÁN NOT SO NAIVE 50](#_Toc429383458)

[11.1. Đặc điểm 50](#_Toc429383459)

[11.2. Mô tả thuật toán 50](#_Toc429383460)

[11.3. Mã hóa thuật toán 51](#_Toc429383461)

[11.4. Kiểm nghiệm thuật toán 51](#_Toc429383462)

[12. THUẬT TOÁN BOYER-MOORE 56](#_Toc429383463)

[12.1. Đặc điểm chính 56](#_Toc429383464)

[12.2. Mô tả thuật toán 56](#_Toc429383465)

[12.3. Mã hóa thuật toán 58](#_Toc429383466)

[12.4. Kiểm nghiệm thuật toán 59](#_Toc429383467)

[13. THUẬT TOÁN TURBO BOYER-MOORE 62](#_Toc429383468)

[13.1. Đặc điểm chính 62](#_Toc429383469)

[13.2. Mô tả thuật toán 62](#_Toc429383470)

[13.3. Mã hóa thuật toán 63](#_Toc429383471)

[13.4. Kiểm nghiệm thuật toán 64](#_Toc429383472)

[14. THUẬT TOÁN FORWARD DAWG MATCHING 66](#_Toc429383473)

[14.1. Các tính năng chính 66](#_Toc429383474)

[14.2. Mô tả thuật toán 66](#_Toc429383475)

[14.3. Mã hóa thuật toán 66](#_Toc429383476)

[14.4. Kiểm nghiệm thuật toán 68](#_Toc429383477)

[15. THUÂT TOÁN HORSPOOL 71](#_Toc429383478)

[15.1. Phát biểu thuật toán 71](#_Toc429383479)

[15.2. Mã hóa thuật toán 71](#_Toc429383480)

[15.3. Kiểm nghiệmthuật toán 72](#_Toc429383481)

[16. THUẬT TOÁN QUICK SEARCH 74](#_Toc429383482)

[16.1. Đặc điểm chính 74](#_Toc429383483)

[16.2. Mô tả thuật toán 74](#_Toc429383484)

[16.3. Mã hóa thuật toán 74](#_Toc429383485)

[16.4. Kiểm nghiệm thuật toán 75](#_Toc429383486)

[17. THUẬT TOÁN TUNED BOYER-MOORE 77](#_Toc429383487)

[17.1. Đặc điểm chính 77](#_Toc429383488)

[17.2. Mô tả thuật toán 77](#_Toc429383489)

[17.3. Mã hóa thuật toán 78](#_Toc429383490)

[17.4. Kiểm nghiệm thuật toán 78](#_Toc429383491)

[18. THUẬT TOÁN ZHU – TAKAOKA 81](#_Toc429383492)

[18.1. Đặc điểm chính 81](#_Toc429383493)

[18.2. Mô tả thuật toán 81](#_Toc429383494)

[18.3. Mã hóa thuật toán 82](#_Toc429383495)

[18.4. Kiểm nghiệm thuật toán 82](#_Toc429383496)

[19. THUẬT TOÁN BERRY – RAVINDRAN 84](#_Toc429383497)

[19.1. Phát biểu thuật toán 84](#_Toc429383498)

[19.2. Mã hóa thuật toán 84](#_Toc429383499)

[19.3. Kiểm nghiệmthuật toán 85](#_Toc429383500)

[20. THUẬT TOÁN RAITA 87](#_Toc429383501)

[20.1. Các tính năng chính 87](#_Toc429383502)

[20.2. Mô tả thuật toán 87](#_Toc429383503)

[20.3. Mã hóa thuật toán 87](#_Toc429383504)

[20.4. Kiểm nghiệm thuật toán 88](#_Toc429383505)

[21. THUẬT TOÁN TURBO REVERSE FACTOR 90](#_Toc429383506)

[21.1. Đặc điểm 90](#_Toc429383507)

[21.2. Mô tả thuật toán 90](#_Toc429383508)

[21.3. Mã hóa thuật toán 91](#_Toc429383509)

[21.4. Kiểm nghiệm thuật toán 93](#_Toc429383510)

[22. THUẬT TOÁN TWO WAY 95](#_Toc429383511)

[22.1. Các đặc điểm chính: 95](#_Toc429383512)

[22.2. Mô tả thuật toán 95](#_Toc429383513)

[22.3. Mã hóa thuật toán: 95](#_Toc429383514)

[22.4. Nguyên tắc cơ bản của thuật toán Two-Way với tối đa ngắn 98](#_Toc429383515)

[22.5. Kiểm nghiệm thuật toán 98](#_Toc429383516)

1. Tổng quan

# THUẬT TOÁNBRUTE FORCE

## Phát biểu thuật toán

Thuật toán Brute-Force để tìm sự xuất hiện của một chuỗi (được gọi là mẫu) trong một văn bản bằng cách kiểm tra từng vị trí trong văn bản ở đó mẫu có thể khớp được, cho đến khi chúng khớp nhau thực sự.

Thuật toán Brute Force không cần giai đoạn tiền xử lý cũng như các mảng phụ cho quá trình tìm kiếm. Độ phức tạp tính toán của thuật toán này là O(N\*M).

## Mô tả thuật toán

Thuật toán Brute-Force:

***Input :***

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

• Văn bản 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.

## Mã hóa thuật toán

void BF(char \*x, int m, char \*y, int n) {

int i, j;

/\* Searching \*/

for (j = 0; j <= n - m; ++j) {

for (i = 0; i < m && x[i] == y[i + j]; ++i);

if (i >= m)

OUTPUT(j);

}

}

## Kiểm nghiệm thuật toán

X = 10100111 (m = 8)

Y = 1001110100101000101001110 (n = 25)

|  |  |  |  |
| --- | --- | --- | --- |
| **j** | **i** | **X[i] == Y[i+j]?** | **OUTPUT (j)** |
| **0** | 0  1  2 | 1 == 1 (Y)  0 == 0 (Y)  1 == 0 (N) |  |
| **1** | 0 | 1 == 0 (N) |  |
| **2** | 0 | 1 == 0 (N) |  |
| **3** | 0  1 | 1 == 1 (Y)  0 == 1 (N) |  |
| **4** | 0  1 | 1 == 1 (Y)  0 == 1 (N) |  |
| **5** | 0  1  2  3  4  5  6 | 1 == 1 (Y)  0 == 0 (Y)  1 == 1 (Y)  0 == 0 (Y)  0 == 0 (Y)  1 == 1 (Y)  1 == 0 (N) |  |
| **6** | 0 | 1 == 0 (N) |  |
| **7** | 0  1  2 | 1 == 1 (Y)  0 == 0 (Y)  1 == 0 (N) |  |
| **8** | 0 | 1 == 0 (N) |  |
| **9** | 0 | 1 == 0 (N) |  |
| **10** | 0  1  2  3  4  5 | 1 == 1 (Y)  0 == 0 (Y)  1 == 1 (Y)  0 == 0 (Y)  0 == 0 (Y)  1 == 0 (N) |  |
| **11** | 0 | 1 == 0 (N) |  |
| **12** | 0 | 1 == 1 (Y)  0 == 0 (Y)  1 == 0 (N) |  |
| **13** | 0 | 1 == 0 (N) |  |
| **14** | 0 | 1 == 0 (N) |  |
| **15** | 0 | 1 == 0 (N) |  |
| **16** | 0  1  2  3  4  5  6  7 | 1 == 1 (Y)  0 == 0 (Y)  1 == 1 (Y)  0 == 0 (Y)  0 == 0 (Y)  1 == 1 (Y)  1 == 1 (Y)  1 == 1 (Y) | OUTPUT(16) |
| **17** | 0 | 1 == 0 (N) |  |

# THUẬT TOÁN TÌM KIẾM VỚI MỘT AUTOMAT

## Đặc điểm chính

* Xây dựng một Automat xác định tối thiểu có khả năng nhận diện ngôn ngữ



* Mở rộng không gian trong O(m) nếu automat được lưu trữ trực tiếp trong một bảng truy nhập trực tiếp;



* Pha tiền xử lý có độ phức tạp thời gian O(m);



* Pha tìm kiếm có độ phức tạp thời gian O(n) nếu automat được lưu trữ trực tiếp trong một bảng truy nhập trực tiếp và O(nlog()) trong trường hợp khác.



## Mô tả thuật toán

Tìm kiếm một từ x bằng automat bắt đầu băng việc xây dựng một Automat hữu hạn xác định (DFA) A(x) có khả năng nhận diện ngôn ngữ ;



DFA A(x) = (Q, q0, T, E) có khả năng nhận diện ngôn ngữ được định nghĩa như sau:



* Q là tập tất cả các tiền tố của x: Q = {€, x[0], x[0…1], x[0…2], …x[0…m-2], x};
* q0 = € là trạng thái ban đầu;
* T = {x};
* Với mỗi q trong Q (q là một tiền tố của x) và a trong ∑, (q, a, qa) là một bộ trong E nếu và chỉ nếu qa cũng là một tiền tố của x

## Mã hóa thuật toán

void preAut(char \*x, int m, Graph aut) {

int i, state, target, oldTarget;

for (state = getInitial(aut), i = 0; i < m; ++i) {

oldTarget = getTarget(aut, state, x[i]);

target = newVertex(aut);

setTarget(aut, state, x[i], target);

copyVertex(aut, target, oldTarget);

state = target;

}

setTerminal(aut, state);

}

void AUT(char \*x, int m, char \*y, int n) {

int j, state;

Graph aut;

/\* Preprocessing \*/

aut = newAutomaton(m + 1, (m + 1)\*ASIZE);

preAut(x, m, aut);

/\* Searching \*/

for (state = getInitial(aut), j = 0; j < n; ++j) {

state = getTarget(aut, state, y[j]);

if (isTerminal(aut, state))

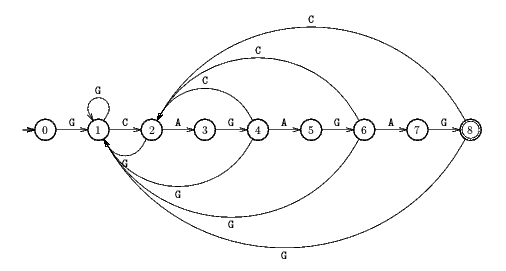
OUTPUT(j - m + 1);

}

}

## Kiểm nghiệm thuật toán

Pha tiền xử lý: Xây dựng một Automat hữu hạn



Pha tìm kiếm:Trạng thái hiện tại là: 0

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
| 1 |  | | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | 2 |  | | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | 3 |  | | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | 0 |  | | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | 0 |  | | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | 1 |  | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | 2 |  | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | 3 |  | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | 4 |  | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | 5 |  | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | 6 |  | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | 7 |  | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | 8 |  | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | 0 |  | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | 0 |  | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | | 0 |  | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | | | 0 |  | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | | | | 0 |  | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | | | | | 0 |  | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | | | | | | 1 |  | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | | | | | | | 0 |  | | |
|  | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | | | | | | | | 0 |  | |
|  | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | | | | | | | | | 0 |  |
|  | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | | | | | | | | | | 1 |

Thuật toán tìm kiếm với một Automat thực hiện kiểm tra 24 lượt ký tự trong ví dụ trên.

# THUẬT TOÁN KARP-RABIN

## Đặc điểm

* Sử dụng 1 hàm băm để tìm chuỗi con
* Độ phức tạp thời gian và không gian tiền xử lý *O(m)*
* Độ phức tạp thời gian và không gian xử lý tìm kiếm *O(m+n)*

## Hàm băm cơ bản

Hàm băm là giải thuật nhằm sinh ra các giá trị băm tương ứng với mỗi khối dữ liệu, một chuỗi kí tự, một đối tượng trong lập trình hướng đối tượng,.. Giá trị băm đóng vai gần như một khóa để phân biệt các khối dữ liệu, tuy nhiên, người ta chấp hiện tượng trùng khóa hay còn gọi là đụng độ và cố gắng cải thiện giải thuật để giảm thiểu sự đụng độ đó. Hàm băm thường được dùng trong bảng băm nhằm giảm chi phí tính toán khi tìm một khối dữ liệu trong một tập hợp, nhờ việc so sánh các giá trị băm nhanh hơn việc so sánh những khối dữ liệu có kích thước lớn.

Một hàm băm đơn giản nhất đó là tính toán giá trị băm dựa trên mã ASCII hoặc UNICODE của từng ký tự. Ví dụ với chuỗi nguồn là “abcdefgh” và chuỗi cần tìm có chiều dài là 4 thì giá trị băm đầu tiên như sau:

h1 = a + b + c + d

= 97 + 98 + 99 + 100

= 394

Giá trị băm tiếp theo cần tính là:

h2 = b + c + d + e

= h1 – a + e

= 394 – 97 + 101

## Mô tả thuật toán

Thuật toán Rabin-Karp sử dụng hàm băm để so sánh giá trị băm của các chuỗi trước khi thực sự so sánh chuỗi. Phương pháp này giúp tiết kiệm được thời gian so sánh, đặc biệt với các chuỗi tìm kiếm dài.

***Input:***

* T[0 .. n-1] : là văn bản có n ký tự
* P[0 .. m -1]: là pattern có m ký tự với m ≤ n
* ts : là giá trị băm của chuỗi con tuần tự T[s .. s+m-1] trong T với độ dịch chuyển là s, trong đó 0 ≤ s ≤n-m
* p: là giá trị băm của P.

***Output***

Khi này thuật toán so sánh lần lượt giá trị ts với p với s chạy từ 0 đến n-m, bước tiếp theo của thuật toán sẽ xảy ra với hai trường hợp như sau:

* TH1: ts = p, thực hiện phép đối sánh chuỗi giữa T[s .. s+m-1] và P[0.. m-1]
* TH2: ts ≠ p, nếu s ≤ m tính gán s = s+1 và tính tiếp giá trị băm ts .

## Mã hóa thuật toán

#define REHASH(a, b, h) ((((h) - (a)\*d) << 1) + (b))

void KR(char \*x, int m, char \*y, int n) {

int d, hx, hy, i, j;

/\* Preprocessing \*/

/\* computes d = 2^(m-1) with

the left-shift operator \*/

for (d = i = 1; i < m; ++i)

d = (d<<1);

for (hy = hx = i = 0; i < m; ++i) {

hx = ((hx<<1) + x[i]);

hy = ((hy<<1) + y[i]);

}

/\* Searching \*/

j = 0;

while (j <= n-m) {

if (hx == hy && memcmp(x, y + j, m) == 0)

OUTPUT(j);

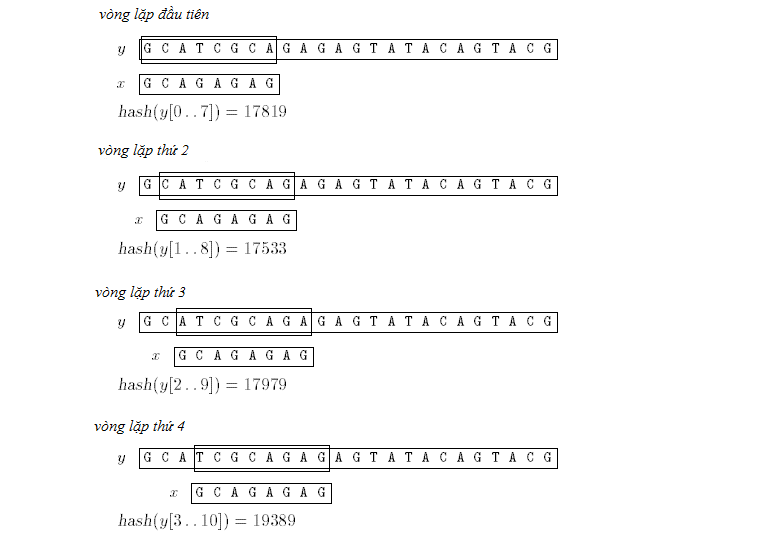
hy = REHASH(y[j], y[j + m], hy);

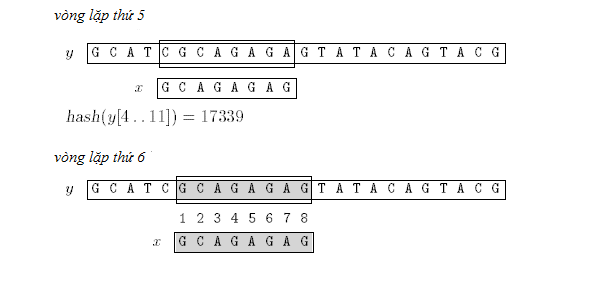
++j;

}

}

## Kiểm nghiệmthuật toán

******

******

Thuật toán tìm được chuỗi khớp với chuối so sánh, thì lưu chuỗi rồi tiếp tục thực hiện như thế cho đến hết.

Kết quả là qua 1 vòng lặp so sánh bộ 8 kí tự và tìm được 1 bộ kí tự ở vòng lặp thứ trùng khớp.

# THUẬT TOÁN SHITF-OR

## Đặc điểm

* Sử dụng các toán tử thao tác bít (Bitwise).
* Hiệu quả trong trường hợp độ dài mẫu nhỏ hơn một từ máy.
* Thực hiện pha tiền xử lý với thời gian O( m +);
* Pha tìm kiếm có độ phức tạp tính toán O(n).

## Mô tả thuật toán

***Input:***

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

***Output:***

* Đưa ra mọi vị trí xuất hiện của X trong Y.

***Formats: k = Shift-Or(X, m, Y, n);***

***Actions:***

Bước 1 (Tiền xử lý): PreSo(X, m, S); //chuyển X thành tập các số.

Bước 2 (Tìm kiếm): SO(X, n, Y, m); //Tìm kiếm mẫu X trong Y

***EndActions***

## Mã hóa thuật toán

int preSo(char \*x, int m, unsigned int S[]) {

unsigned int j, lim;

int i;

for (i = 0; i < ASIZE; ++i)

S[i] = ~0;

for (lim = i = 0, j = 1; i < m; ++i, j <<= 1) {

S[x[i]] &= ~j;

lim |= j;

}

lim = ~(lim>>1);

return(lim);

}

void SO(char \*x, int m, char \*y, int n) {

unsigned int lim, state;

unsigned int S[ASIZE];

int j;

if (m > WORD)

error("SO: Use pattern size <= word size");

/\* Preprocessing \*/

lim = preSo(x, m, S);

/\* Searching \*/

for (state = ~0, j = 0; j < n; ++j) {

state = (state<<1) | S[y[j]];

if (state < lim)

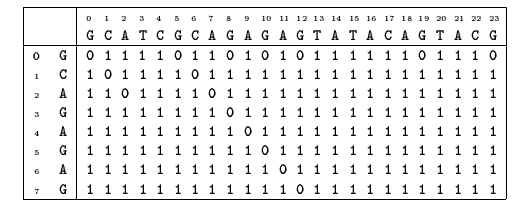
OUTPUT(j - m + 1);

}

}

## Kiểm nghiệm thuật toán

Pha tìm kiếm:

  
Khi***R***12[7]=0 điều đó có nghĩa một sự xuất hiện của *x*đã được tìm thấy tại vị trí 12-8+1=5.

# THUẬT TOÁN MORRIS – PRATT

## Đặc điểm chính

* Thực hiện việc so sánh từ trái qua phải;
* Pha tiền xử lý có độ phức tạp thời gian và không gian O(m);
* Pha tìm kiềm có độ phức tạp thời gian là O(m+n);
* Thực hiện tối đa 2n-1 thu thập thông tin trong quá trình dò quét văn bản;
* Độ trễ giới hạn là m.

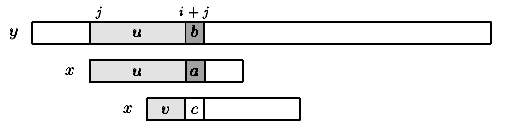
## Mô tả thuật toán

Thiết kế của thuật toán Morris-Pratt tuân theo một phân tích một cách chặt chẽ thuật toán Brute-Force.

Có thể cải tiến chiều dài đoạn dịch chuyển và ghi nhớ những thông tin liên quan tới các đoạn ký tự phù hợp với mẫu. Việc đó giúp tiết kiệm việc so sánh các ký tự của mẫu và ký tự trong đoạn văn bản nên tăng được tốc độ tìm kiếm;

Hãy xem xét một lần thử tại một vị trí j bên trái của y, khi đó cửa sổ được đặt trên đoạn văn bản y [j .. j + m-1]. Giả sử rằng không phù hợp đầu tiên xảy ra giữa x [i] và y [i + j] với 0 <i <m. Do đó, x [0..i-1] = y [j .. i + j-1] = u và a = x [i] khác với y [i + j] = b

Khi dịch chuyển, hi vọng răng tồn tại một tiền tố v phù hợp với đoạn hậu tố u của đoạn văn bản. Chiều dài lớn nhất của v được gọi là biên của u. Sử dụng ký hiệu mpNext[i] là chiều dài của biên dài nhất của x[0…i-1] với 0<i<=m. Do vậy, sau một lần dịch chuyển, việc so sánh có thể bắt đầu lại giữa ký tự c=x[mpNext[i]] và y[i+j] = b. Giá trị mpNext[0] = -1



## Mã hóa thuật toán

void preMp(char \*x, int m, int mpNext[]) {

int i, j;

i = 0;

j = mpNext[0] = -1;

while (i < m) {

while (j > -1 && x[i] != x[j])

j = mpNext[j];

mpNext[++i] = ++j;

}

}

void MP(char \*x, int m, char \*y, int n) {

int i, j, mpNext[XSIZE];

/\* Preprocessing \*/

preMp(x, m, mpNext);

/\* Searching \*/

i = j = 0;

while (j < n) {

while (i > -1 && x[i] != y[j])

i = mpNext[i];

i++;

j++;

if (i >= m) {

OUTPUT(j - i);

i = mpNext[i];

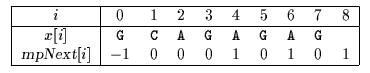
}

}

}

## Kiểm nghiệm thuật toán

Pha tiền xử lý:



*Bảng mpNext*

Pha tìm kiếm:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| First attempt | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
| 1 | 2 | 3 | 4 |  | | | | | | | | | | | | | | | | | | | |
| G | C | A | G | A | G | A | G |  | | | | | | | | | | | | | | | |

Shift by: 3 (*i*-*mpNext*[*i*]=3-0)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Second attempt | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | 1 |  | | | | | | | | | | | | | | | | | | | |
|  | | | G | C | A | G | A | G | A | G |  | | | | | | | | | | | | |

Shift by: 1 (*i*-*mpNext*[*i*]=0- -1)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Third attempt | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | 1 |  | | | | | | | | | | | | | | | | | | |
|  | | | | G | C | A | G | A | G | A | G |  | | | | | | | | | | | |

Shift by: 1 (*i*-*mpNext*[*i*]=0- -1)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Fourth attempt | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |  | | | | | | | | | | |
|  | | | | | G | C | A | G | A | G | A | G |  | | | | | | | | | | |

Shift by: 7 (*i*-*mpNext*[*i*]=8-1)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Fifth attempt | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | 1 |  | | | | | | | | | |
|  | | | | | | | | | | | | G | C | A | G | A | G | A | G |  | | | |

Shift by: 1 (*i*-*mpNext*[*i*]=1-0)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Sixth attempt | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | 1 |  | | | | | | | | | |
|  | | | | | | | | | | | | | G | C | A | G | A | G | A | G |  | | |

Shift by: 1 (*i*-*mpNext*[*i*]=0- -1)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Seventh attempt | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | 1 |  | | | | | | | | |
|  | | | | | | | | | | | | | | G | C | A | G | A | G | A | G |  | |

Shift by: 1 (*i*-*mpNext*[*i*]=0- -1)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Eighth attempt | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | | 1 |  | | | | | | | |
|  | | | | | | | | | | | | | | | G | C | A | G | A | G | A | G |  |

Shift by: 1 (*i*-*mpNext*[*i*]=0- -1)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Ninth attempt | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | | | 1 |  | | | | | | |
|  | | | | | | | | | | | | | | | | G | C | A | G | A | G | A | G |

Shift by: 1 (*i*-*mpNext*[*i*]=0- -1)

Thuật toán Morris-Pratt thực hiện 19 so sánh ký tự trong ví dụ nêu trên.

# THUẬT TOÁN KNUTH-MORRIS-PRATT

## Trình bày thuật toán

Ý tưởng chính của phương pháp này như sau : trong quá trình tìm kiếm vị trí của mẫu P trong xâu gốc T, nếu tìm thấy một vị trí sai ta chuyển sang vị trí tìm kiếm tiếp theo và quá trình tìm kiếm sau này sẽ được tận dụng thông tin từ quá trình tìm kiếm trước để không phải xét các trường hợp không cần thiết.

## Đặc điểm

* Thực hiện so sánh từ trái qua phải
* Độ phức tạp thời gian và không gian tiền xử lý *O(m)*
* Độ phức tạp thời gian và không gian xử lý tìm kiếm *O(m+n)*
* Xử lý gần 2n-1 kí tự chữ trong quá trình tìm kiếm

## Mô tả thuật toán

***Input:***

Xâu mẫu X = (x0, x1,…, xm) có độ dài m

***Output:***

Mảng giá trị kmpNext[]

*Bước tiền xử lý*

Khởi tạo: PreKMP(X, m, kmpNext);

i = 1; kmpNext[0] = 0; len = 0;

Thực hiện: while(i<m){

if(X[i] == X[len]) { len++; kmpNext[i] = len; i++;}

else {

if(len!=0) len = kmpNext[len-1];

else kmpNext[i] = 0;

i++; }

}

*Bước xử lý*

***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.

## Mã hóa thuật toán

void preKmp(char \*x, int m, int kmpNext[]) {

int i, j;

i = 0;

j = kmpNext[0] = -1;

while (i < m) {

while (j > -1 && x[i] != x[j])

j = kmpNext[j];

i++;

j++;

if (x[i] == x[j])

kmpNext[i] = kmpNext[j];

else

kmpNext[i] = j;

}

}

void KMP(char \*x, int m, char \*y, int n) {

int i, j, kmpNext[XSIZE];

/\* Preprocessing \*/

preKmp(x, m, kmpNext);

/\* Searching \*/

i = j = 0;

while (j < n) {

while (i > -1 && x[i] != y[j])

i = kmpNext[i];

i++;

j++;

if (i >= m) {

OUTPUT(j - i);

i = kmpNext[i];

}

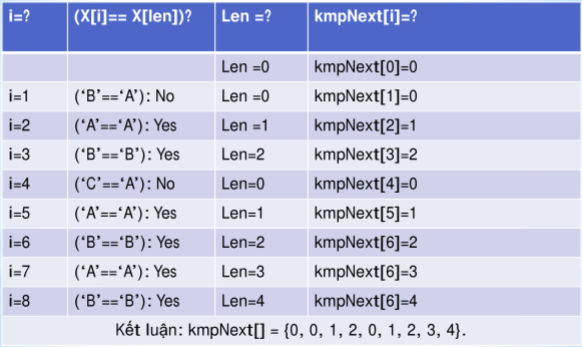
}

}

## Kiểm nghiệm thuật toán

*Bước tiền xử lý*

* Với X[] = “ABABCABAB”, m = 9



*Bước xử lý*

Knuth-Moriss-Patt (X, m, Y, n)

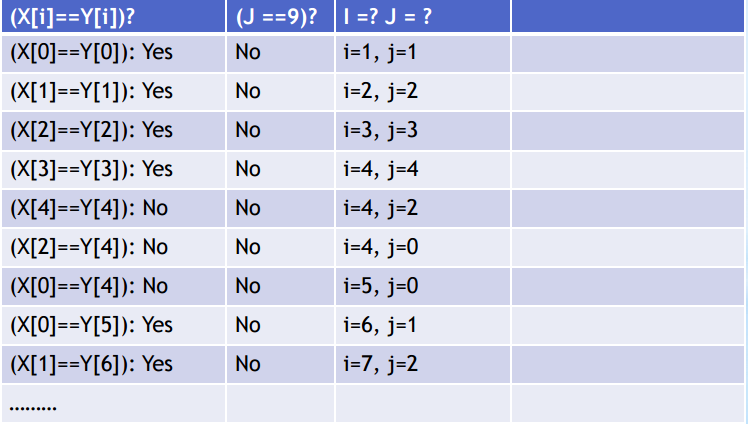
• X[] = “ABABCABAB”, m = 9.

• 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):



# THUẬT TOÁN SIMON

## Các đặc điểm chính

* Thực hiện một cách tiết kiệm A (x) tối thiểu Deterministic Finite Automaton công nhận ;



* Giai đoạn tiền xử lý trong thời gian O(m) và không gian phức tạp
* Giai đoạn tìm kiếm độ phức tạp thời gian là O (m + n) độc lập với kích thước bảng chữ cái
* Nhiều nhất là 2n -1 văn bản so sánh các phần tử trong giai đoạn tìm kiếm
* Thời gian trễ giới hạn Min{1 + , }



## Mô tả thuật toán

Thuật toán hạn chế việc tìm kiếm với tối thiểu A (x) là kích thước của automaton: O (m x ). Simon nhận thấy rằng chỉ có một vài góc cạnh quan trọng có ý nghĩa của A(x), đó là:



* Cạnh phía trước sẽ từ tiền tố của x chiều dài k đến tiền tố chiều dài k + 1 với 0 k < m (m là số cạnh xác định)



* Quay trở lại cạnh đi từ các tiền tố của x chiều dài k đến một tiền tố có chiều dài nhỏ hơn khác không. Số lượng cạnh nhỏ hơn hoặc bằng m.

Các cạnh khác dẫn đến trạng thái ban đầu và sau đó có thể được suy ra. Do đó, số các cạnh quan trọng được giới hạn bởi 2m. sau đó, đối với mỗi trạng thái automaton chỉ cần thiết để lưu trữ các danh sách của các cạnh đi ra quan trọng của nó

Mỗi trạng thái được dại diện bởi chiều dài của tiền tố liên quan của nó trừ đi 1 để mỗi cạnh dẫn đến phần tử i với -1 i m-1 gọi là x[i] do đó nó không phải là cần thiết để lưu trữ các tên của các cạnh. Các cạnh phía trước có thể dễ dàng được suy ra từ mô hình, do đó chúng không được lưu trữ. Nó chỉ còn lại để lưu trữ các cạnh có ý nghĩa.



Sử dụng một bảng L, kích thước m-1, danh sách liên kết. các phần tử L[i] cho danh sách các mục tiêu của các cạnh bắt đầu từ trạng thái i. để tránh lưu trữ các danh sách trạng thái m-1 trong việc tính toán của bảng L, các số nguyên l được tính toán như vậy mà l + 1 là chiều dài của đường biên giới dài nhất của x.

* Giai đoạn 1: tiền xử lý thuật toán Simon bao gồm bảng L và số nguyên l. nó có thể được thực hiện trong không gian O (m) và phức tạp thời gian.
* Giai đoạn 2: Tìm kiến thực hiện tương tự như trong những tìm kiếm với một automaton. Khi sự xuất hiện của mô hình được tìm thấy, trạng thái hiện tại được cập nhật bởi trạng thái l. Giai đoạn này có thể được thực hiện trong thời gian O (m + n). Các thuật toán Simon thực hiện nhiều nhất là 2n -1 so sánh phần tử văn bản trong giai đoạn tìm kiếm, sự chậm trễ được giới hạn Min{1 + , }.



## Mã hóa thuật toán

int getTransition(char \*x, int m, int p, List L[],

char c) {

List cell;

if (p < m - 1 && x[p + 1] == c)

return(p + 1);

else if (p > -1) {

cell = L[p];

while (cell != NULL)

if (x[cell->element] == c)

return(cell->element);

else

cell = cell->next;

return(-1);

}

else

return(-1);

}

void setTransition(int p, int q, List L[]) {

List cell;

cell = (List)malloc(sizeof(struct \_cell));

if (cell == NULL)

error("SIMON/setTransition");

cell->element = q;

cell->next = L[p];

L[p] = cell;

}

int preSimon(char \*x, int m, List L[]) {

int i, k, ell;

List cell;

memset(L, NULL, (m - 2)\*sizeof(List));

ell = -1;

for (i = 1; i < m; ++i) {

k = ell;

cell = (ell == -1 ? NULL : L[k]);

ell = -1;

if (x[i] == x[k + 1])

ell = k + 1;

else

setTransition(i - 1, k + 1, L);

while (cell != NULL) {

k = cell->element;

if (x[i] == x[k])

ell = k;

else

setTransition(i - 1, k, L);

cell = cell->next;

}

}

return(ell);

}

void SIMON(char \*x, int m, char \*y, int n) {

int j, ell, state;

List L[XSIZE];

/\* Preprocessing \*/

ell = preSimon(x, m, L);

/\* Searching \*/

for (state = -1, j = 0; j < n; ++j) {

state = getTransition(x, m, state, L, y[j]);

if (state >= m - 1) {

OUTPUT(j - m + 1);

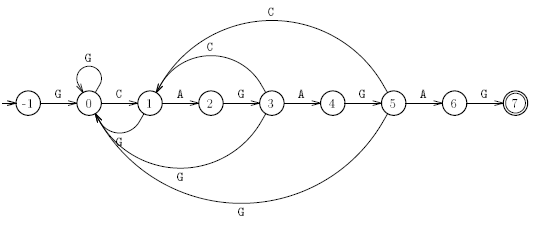
state = ell;

}

}

}

## Kiểm nghiệm thuật toán



***Giai đoạn 1: Tiền xử lý***

***Giai đoạn 2: Tìm kiếm***

Trạng thái ban đầu: -1

G C A T C G C A G A G A G T A T A C A G T A C G

0

G C A T C G C A G A G A G T A T A C A G T A C G

1

G C A T C G C A G A G A G T A T A C A G T A C G

2

G C A T C G C A G A G A G T A T A C A G T A C G

-1

G C A T C G C A G A G A G T A T A C A G T A C G

-1

G C A T C G C A G A G A G T A T A C A G T A C G

0

G C A T C G C A G A G A G T A T A C A G T A C G

1

G C A T C G C A G A G A G T A T A C A G T A C G

2

G C A T C G C A G A G A G T A T A C A G T A C G

3

G C A T C G C A G A G A G T A T A C A G T A C G

4

G C A T C G C A G A G A G T A T A C A G T A C G

5

G C A T C G C A G A G A G T A T A C A G T A C G

6

G C A T C G C A G A G A G T A T A C A G T A C G

7

G C A T C G C A G A G A G T A T A C A G T A C G

-1

G C A T C G C A G A G A G T A T A C A G T A C G

-1

G C A T C G C A G A G A G T A T A C A G T A C G

-1

G C A T C G C A G A G A G T A T A C A G T A C G

-1

G C A T C G C A G A G A G T A T A C A G T A C G

-1

G C A T C G C A G A G A G T A T A C A G T A C G

-1

G C A T C G C A G A G A G T A T A C A G T A C G

0

G C A T C G C A G A G A G T A T A C A G T A C G

-1

G C A T C G C A G A G A G T A T A C A G T A C G

-1

G C A T C G C A G A G A G T A T A C A G T A C G

-1

G C A T C G C A G A G A G T A T A C A G T A C G

0

Độ dài xâu: 24

Độ dài mẫu: 8

Ký tự kiểm tra: 24

# THUẬT TOÁN COLUSSI

## Đặc điểm

* Sàng lọc lại thuật toán Knutt-Morris-Pratt;
* Phân vùng tập các vị trí mẫu thành 2 tập con rời nhau; các vị trí trong tập đầu tiên được từ trái qua phải và khi không có sự phù hợp xảy ra các vị trí trong tập con thứ 2 sẽ được quét từ phải qua trái;
* Pha tiền xử lý có độ phức tạp không gian và thời gian là O(m);
* Pha tìm kiếm có độ phức tạp thời gian là O(n);
* Trong trường hợp xấu nhất phải thực hiện so sánh ký tự văn bản



## Mô tả thuật toán

Việc thiết kế thuật toán Colussi tuân theo một phân tích có tính chặt chẽ của thuật toán Knutt-Morris-Pratt

Tập các vị trí mẫu được phân chia thành 2 tập con rời nhau. Sau đó, mỗi mẫu thử bao gồm 2 pha:

* Trong pha đầu tiên, các so sánh được thực hiện từ trái qua phải với các ký tự văn bản phù hợp với vị trí mẫu mà giá trị của hàm kmpNext hoàn toàn lớn hơn -1. Những vị trí đó được gọi là ***noholes;***
* Pha thứ 2 bao gồm việc so sánh các vị trí còn lại (được gọi là ***holes***) từ phải qua trái.

Chiến lược này có 2 ưu điểm:

* Khi một không phù hợp xảy ra trong pha đầu tiên, sau khi dịch chuyển thích hợp không cần thiết phải so sánh ký tự văn bản phù hợp với noholes được so sánh trong suốt mẫu thử trước;
* Khi một không phù hợp xảy ra trong pha thứ 2 điều đó có nghĩa là một hậu tố của mẫu thử phù hợp với một nhân tố của văn bản, sau khi dịch chuyển tương ứng một tiền tố của mẫu thử cũng sẽ vẫn phù hợp với một nhân tố của văn bản, do đó không cần thiết phải so sánh lại với nhân tố đó nữa.

**Định nghĩa thuật toán:**

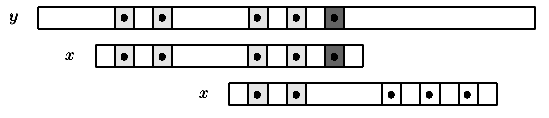
* For 0 leq *i* leq *m*-1:



* Khi *kmin* neq 0 một chu kỳ kết thúc tại vị trí *i* trong *x*.
* For 0 < *i* < *m*:

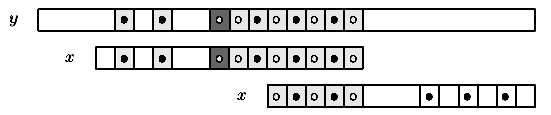


* Lấy *nd*+1 là số lượng của noholes trong *x*.
* Bảng *h* chứa *nd+1* noholes đầu tiên theo thứ tự tăng dần và tiếp đó là *m-nd-1* holes theo thứ tự giảm dần:
  + for *0 leq i leq nd*, *h[i]* là một nohole và *h[i] < h[i+1]* với *0 leq i<nd*;
  + for *nd < i < m*, *h[i]* là một hole và *h[i] > h[i+1]* với *nd < i < m-1*
* Nếu *i* là một hole thì *rmin[i]* là chu kỳ nhỏ nhất của *x* lớn hơn *i*
* Giá trị của *first[u]* là số nguyên nhỏ nhất *v* mà *u* leq *h*[*v*]
* Tiếp theo, giả sử rằng *x* phù hợp với *y[j .. j+m-1]*. Nếu *x[h[k]]=y[j+h[k]]* với *0 leq k < r < nd* và *x[h[r]] neq y[j+h[r]]*. Lấy *j’ = j+kmin[h[r]]*. Tiếp đó không có sự xuất hiện của *x* bắt đầu trong *y[j .. j’]* và *x* có thể được dịch *kmin*[h[r]] vị trí sang phải .
* Ngoài ra *x[h[k]]=y[j’+h[k]]* với *0 leq k < first[h[r]-kmin[h[r]]]* có nghĩa rằng việc so sánh có thể được tiếp tục với *x[h[first[h[r]-kmin[h[r]]]]]* và *y[j’+h[first[h[r]-kmin[h[r]]]]]*.



*Hình 1: Không phù hợp với một nohole. Noholes là các vòng tròn màu đen và được so sánh từ trái qua phải*

* Nếu *x[h[k]]=y[j+h[k]]* với *0 leq k < r* và *x[h[r]] neq y[j+h[r]]* với *nd leq r < m*. Lấy *j’=j+rmin[h[r]]*. Tiếp đó, không có sự xuất hiện nào của *x* bắt đầu trong *y[j .. j’]* và *x* có thể được dịch *kmin[h[r]]* vị trí sang phải.
* Ngoài ra *x[0 .. m-1-rmin[h[r]]]=y[j’ .. j+m-1]* có nghĩa rằng việc so sánh có thể được tiếp tục với *x[h[first[m-1-rmin[h[r]]]]]* và *y[j’+h[first[m-1-rmin[h[r]]]]]*.



*Hình 2: Không phù hợp với một hole. Noholes là các vòng tròn màu đen và được so sánh từ trái qua phải, trong khi holes là các vòng tròn màu trắng và được so sánh từ phải qua trái*

* Để tính toán giá trị *kmin*, một bảng *hmax* được sử dụng và được định nghĩa như sau: *hmax[k]* thỏa mãn *x[k .. hmax[k]-1]=x[0 .. hmax[k]-k-1]* và *x[hmax[k]] neq x[hmax[k]-k]*.
* Giá trị của *nhd0[i]* là số lượng các noholes chắc chắn nhỏ hơn *i*
* Chúng ta định nghĩa 2 hàm *shift* và *next* như sau:
  + *shift[i]=kmin[h[i]] và next[i]=nhd0[h[i]-kmin[h[i]]] với i < nd;*
  + *shift[i]=rmin[h[i]] và next[i]=nhd0[m-rmin[h[i]]] với nd leq i < m;*
  + *shift[m]=rmin[0] và next[m]=nhd0[m-rmin[h[m-1]]].*
* Do đó, trong suốt một lần thử, khi cửa sổ được đặt ở vị trí trên nhân tố văn bản *y[j .. j+m-1]*, khi một không phù hợp xuất hiện giữa *x[h[r]] và y[j+h[r]]* cửa số phải được dịch đi *shift[r]* và những so sánh có thể được tiếp tục với mẫu ởi vị trí *h[next[r]]*.

Pha tiền xử lý có thể hoàn thành trong một O(m) không gian và thời gian. Pha tìm kiếm có thể hoàn thành trong O(n) độ phức tạp thời gian và hơn nữa tối đa 3/2n lần việc so sánh ký tự văn bản được thực hiện trong suốt pha tìm kiếm.

## Mã hóa thuật toán

int preColussi(char \*x, int m, int h[], int next[],

int shift[]) {

int i, k, nd, q, r, s;

int hmax[XSIZE], kmin[XSIZE], nhd0[XSIZE], rmin[XSIZE];

/\* Computation of hmax \*/

i = k = 1;

do {

while (x[i] == x[i - k])

i++;

hmax[k] = i;

q = k + 1;

while (hmax[q - k] + k < i) {

hmax[q] = hmax[q - k] + k;

q++;

}

k = q;

if (k == i + 1)

i = k;

} while (k <= m);

/\* Computation of kmin \*/

memset(kmin, 0, m\*sizeof(int));

for (i = m; i >= 1; --i)

if (hmax[i] < m)

kmin[hmax[i]] = i;

/\* Computation of rmin \*/

for (i = m - 1; i >= 0; --i) {

if (hmax[i + 1] == m)

r = i + 1;

if (kmin[i] == 0)

rmin[i] = r;

else

rmin[i] = 0;

}

/\* Computation of h \*/

s = -1;

r = m;

for (i = 0; i < m; ++i)

if (kmin[i] == 0)

h[--r] = i;

else

h[++s] = i;

nd = s;

/\* Computation of shift \*/

for (i = 0; i <= nd; ++i)

shift[i] = kmin[h[i]];

for (i = nd + 1; i < m; ++i)

shift[i] = rmin[h[i]];

shift[m] = rmin[0];

/\* Computation of nhd0 \*/

s = 0;

for (i = 0; i < m; ++i) {

nhd0[i] = s;

if (kmin[i] > 0)

++s;

}

/\* Computation of next \*/

for (i = 0; i <= nd; ++i)

next[i] = nhd0[h[i] - kmin[h[i]]];

for (i = nd + 1; i < m; ++i)

next[i] = nhd0[m - rmin[h[i]]];

next[m] = nhd0[m - rmin[h[m - 1]]];

return(nd);

}

void COLUSSI(char \*x, int m, char \*y, int n) {

int i, j, last, nd,

h[XSIZE], next[XSIZE], shift[XSIZE];

/\* Processing \*/

nd = preColussi(x, m, h, next, shift);

/\* Searching \*/

i = j = 0;

last = -1;

while (j <= n - m) {

while (i < m && last < j + h[i] &&

x[h[i]] == y[j + h[i]])

i++;

if (i >= m || last >= j + h[i]) {

OUTPUT(j);

i = m;

}

if (i > nd)

last = j + m - 1;

j += shift[i];

i = next[i];

}

}

## Kiểm nghiệm thuật toán

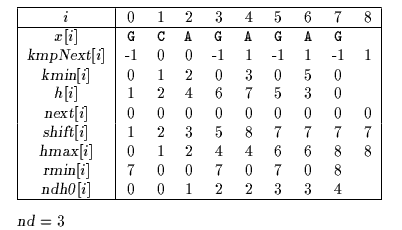
**Văn bản:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |

**Mẫu tìm kiếm**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| G | C | A | G | A | G | A | G |

**Bước tiền xử lý:**



**Bước tìm kiếm:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Lần thử đầu tiên | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | 1 | 2 |  | 3 |  | | | | | | | | | | | | | | | | | | |
| G | C | A | G | A | G | A | G |  | | | | | | | | | | | | | | | |

Shift by: 3 (*shift*[2])

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Lần thử thứ 2 | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | 1 | 2 |  | | | | | | | | | | | | | | | | | |
|  | | | G | C | A | G | A | G | A | G |  | | | | | | | | | | | | |

Shift by: 2 (*shift*[1])

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Lần thử thứ ba | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | 8 | 1 | 2 | 7 | 3 | 6 | 4 | 5 |  | | | | | | | | | | |
|  | | | | | G | C | A | G | A | G | A | G |  | | | | | | | | | | |

Shift by: 7 (*shift*[8])

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Lần thử thứ tư | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | 1 |  | | | | | | | | | |
|  | | | | | | | | | | | | G | C | A | G | A | G | A | G |  | | | |

Shift by: 1 (*shift*[0])

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Lần thử thứ năm | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | 1 |  | | | | | | | | |
|  | | | | | | | | | | | | | G | C | A | G | A | G | A | G |  | | |

Shift by: 1 (*shift*[0])

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Lần thử thứ sáu | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | | 1 |  | | | | | | | |
|  | | | | | | | | | | | | | | G | C | A | G | A | G | A | G |  | |

Shift by: 1 (*shift*[0])

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Lần thử thứ bảy | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | | | 1 |  | | | | | | |
|  | | | | | | | | | | | | | | | G | C | A | G | A | G | A | G |  |

Shift by: 1 (*shift*[0])

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Lần thử thứ tám | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | | | | 1 | 2 |  | 3 |  | | |
|  | | | | | | | | | | | | | | | | G | C | A | G | A | G | A | G |

Shift by: 3 (*shift*[2])

Thuật toán Colussi thực hiện so sánh 20 ký tự trong ví dụ nêu trên.

# THUẬT TOÁN GALIL-GIANCARLO

## Các ðặc ðiểm chính

* Cần hoàn chỉnh các thuật toán Colussi;
* Pha tiền xử lý có độ phức tạp không gian và thời gian là O(m);
* Pha tìm kiếm có độ phức tạp thời gian là O(n);
* Trong trường hợp xấu nhất phải thực hiện so sánh ký tự văn bản



## Mô tả thuật toán

Các thuật toán Galil-Giancarlo là một biến thể của thuật toán Colussi (xem chương 9). Sự thay đổi can thiệp vào pha tìm kiếm. Phương pháp này được áp dụng khi x không phải là một lũy thừa của một ký tự đơn. Như vậy x ≠ *cm* với c € Σ. Lấy l là chỉ số cuối cùng trong patern sao cho 0 ≤ i ≤ l , x [0] = [i] x và x [0] ≠ x [l + 1]. Giả sử trong lần thử trước đó tất cả các noholes đã được so khớp và một hậu tố của pattern đã được so khớp nghĩa là sau sự thay đổi tương ứng một tiền tố của pattern sẽ vẫn khớp với một phần của văn bản. Do đó các ô được đặt ở vị trí trên các nhân tố văn bản *y*[j ... j + m -1] và phân ra *y*[j ... cuối cùng] khớp với x [0 ... cuối cùng - j]. Sau đó lần thử tiếp theo thuật toán sẽ quét các ký tự văn bản bắt đầu với y [last + 1] cho đến khi hoặc kết thúc của văn bản là đạt hoặc một ký tự x[0] ≠ y [j + k] được tìm thấy. Trong trường hợp sau này hai subcases có thể phát sinh:

x [l + 1] ≠ y [j + k] hoặc quá ít x [0] đã được tìm thấy (k ≤ l) sau đó các ô được chuyển và định vị trên các nhân tố văn bản y [k + 1 ... k + m), quá trình quét của văn bản được khôi phục lại (như trong thuật toán Colussi) với nohole đầu tiên và tiền tố ghi nhớ của pattern là những từ rỗng.

x [l + 1] = y [j + k] và đầy đủ của x [0] đã được tìm thấy (k>l) sau đó các ô được chuyển và định vị trên các nhân tố văn bản y [k - l -1 ... k - l + m - 2], quá trình quét của văn bản được khôi phục lại (như trong thuật toán Colussi) với nohole thứ hai (x [l + 1] là đầu tiên) và tiền tố ghi nhớ của pattern là x [0 ... l + 1].

Pha tiền xử lý là chính xác giống như trong thuật toán Colussi (chương 9) và có thể được thực hiện trong O(m) không gian và thời gian. Pha tìm kiếm sau đó có thể thực hiện độ phức tạp về thời gian O(n) và hơn nữa tối đa là 4/3n so sánh ký tự văn bản được thực hiện trong giai đoạn tìm kiếm.

## Mã hóa thuật toán

int preColussi(char \*x, int m, int h[], int next[],

int shift[]) {

int i, k, nd, q, r, s;

int hmax[XSIZE], kmin[XSIZE], nhd0[XSIZE], rmin[XSIZE];

/\* Computation of hmax \*/

i = k = 1;

do {

while (x[i] == x[i - k])

i++;

hmax[k] = i;

q = k + 1;

while (hmax[q - k] + k < i) {

hmax[q] = hmax[q - k] + k;

q++;

}

k = q;

if (k == i + 1)

i = k;

} while (k <= m);

/\* Computation of kmin \*/

memset(kmin, 0, m\*sizeof(int));

for (i = m; i >= 1; --i)

if (hmax[i] < m)

kmin[hmax[i]] = i;

/\* Computation of rmin \*/

for (i = m - 1; i >= 0; --i) {

if (hmax[i + 1] == m)

r = i + 1;

if (kmin[i] == 0)

rmin[i] = r;

else

rmin[i] = 0;

}

/\* Computation of h \*/

s = -1;

r = m;

for (i = 0; i < m; ++i)

if (kmin[i] == 0)

h[--r] = i;

else

h[++s] = i;

nd = s;

/\* Computation of shift \*/

for (i = 0; i <= nd; ++i)

shift[i] = kmin[h[i]];

for (i = nd + 1; i < m; ++i)

shift[i] = rmin[h[i]];

shift[m] = rmin[0];

/\* Computation of nhd0 \*/

s = 0;

for (i = 0; i < m; ++i) {

nhd0[i] = s;

if (kmin[i] > 0)

++s;

}

/\* Computation of next \*/

for (i = 0; i <= nd; ++i)

next[i] = nhd0[h[i] - kmin[h[i]]];

for (i = nd + 1; i < m; ++i)

next[i] = nhd0[m - rmin[h[i]]];

next[m] = nhd0[m - rmin[h[m - 1]]];

return(nd);

}

void GG(char \*x, int m, char \*y, int n) {

int i, j, k, ell, last, nd;

int h[XSIZE], next[XSIZE], shift[XSIZE];

char heavy;

for (ell = 0; x[ell] == x[ell + 1]; ell++);

if (ell == m - 1)

/\* Searching for a power of a single character \*/

for (j = ell = 0; j < n; ++j)

if (x[0] == y[j]) {

++ell;

if (ell >= m)

OUTPUT(j - m + 1);

}

else

ell = 0;

else {

/\* Preprocessing \*/

nd = preCOLUSSI(x, m, h, next, shift);

/\* Searching \*/

i = j = heavy = 0;

last = -1;

while (j <= n - m) {

if (heavy && i == 0) {

k = last - j + 1;

while (x[0] == y[j + k])

k++;

if (k <= ell || x[ell + 1] != y[j + k]) {

i = 0;

j += (k + 1);

last = j - 1;

}

else {

i = 1;

last = j + k;

j = last - (ell + 1);

}

heavy = 0;

}

else {

while (i < m && last < j + h[i] &&

x[h[i]] == y[j + h[i]])

++i;

if (i >= m || last >= j + h[i]) {

OUTPUT(j);

i = m;

}

if (i > nd)

last = j + m - 1;

j += shift[i];

i = next[i];

}

heavy = (j > last ? 0 : 1);

}

}

}

## Kiểm nghiệm thuật toán

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **i** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| x[i] | G | C | A | G | A | G | A | G |  |
| kmpNext[i] | -1 | 0 | 0 | -1 | 1 | -1 | 1 | -1 | 1 |
| kmin[i] | 0 | 1 | 2 | 0 | 3 | 0 | 5 | 0 |  |
| h[i] | 1 | 2 | 4 | 6 | 7 | 5 | 3 | 0 |  |
| next[i] | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| shift[i] | 1 | 2 | 3 | 5 | 8 | 7 | 7 | 7 | 7 |
| hmax[i] | 0 | 1 | 2 | 4 | 4 | 6 | 6 | 8 | 8 |
| rmin[i] | 7 | 0 | 0 | 7 | 0 | 7 | 0 | 8 |  |
| ndh0[i] | 0 | 0 | 1 | 2 | 2 | 3 | 3 | 4 |  |

nd = 3 and l = 0

**Pha tìm kiếm**

***Lần thử đầu tiên:***

y

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
| x | 1 | 2 |  | 3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| G | C | A | G | A | G | A | G |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Shift by 3 (shift[2])

***Lần thử thứ 2:***

y

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  |  | x |  | 1 | 2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | G | C | A | G | A | G | A | G |  |  |  |  |  |  |  |  |  |  |  |  |  |

Shift by 2 (shift[1])

***Lần thử thứ 3:***

y

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  |  |  | x |  | 8 | 1 | 2 | 7 | 3 | 6 | 4 | 5 |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | G | C | A | G | A | G | A | G |  |  |  |  |  |  |  |  |  |  |  |

Shift by 7 (shift[8])

***Lần thử thứ 4:***

y

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  |  |  |  |  |  |  |  |  |  |  | x |  | 1 |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  | G |  |  |  |  |  |  |  |  |  |  |

Shift by 2

***Lần thử thứ 5:***

y

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  |  |  |  |  |  |  |  |  |  |  |  |  | x |  | 1 |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | G | C | A | G | A | G | A | G |  |  |

Shift by 1 (shift[0])

***Lần thử thứ 6:***

y

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | x |  | 1 |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | G | C | A | G | A | G | A | G |  |

Shift by 1 (shift[0])

***Lần thử thứ 7:***

y

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | x |  | 1 | 2 |  | 3 |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | G | C | A | G | A | G | A | G |

Shift by 3 (shift[2])

Các thuật toán Galil-Giancarlo thực hiện 19 so sánh ký tự văn bản trên ví dụ.

# THUẬT TOÁN APOSTOLICO - CROCHEMORE

## Đặc điểm:

* Thực hiện từ trái sang phải
* Pha tiền xử lý có độ phức tạp về không gian và thời gian là



* Pha tìm kiếm có độ phức tạp về thời gian là



* Trong trường hợp xấu nhất, thuật toán thực hiện so sánh ký tự



## Mô tả thuật toán

***Input:***

* Xâu mẫu X = (, ,…, ), độ dài m.



* Văn bản Y = (, ,…, ), độ dài n



***Output:***

* Đưa ra mọi vị trí xuất hiện của X trong Y.

***Formats: Apostolico-Crochemore(X, m, Y, n);***

**Actions:**

Bước 1 (Tiền xử lý):

* Xây dựng bảng dịch để tính toán bước dịch: preKMP(x, m, kmpNext). //sử dụng thuật toán xây dựng bảng dịch của phương pháp Knuth-Morris-Pratt.
* Tìm vị trí ký tự khác nhau đầu tiên trong xâu mẫu X, ký hiệu là *l*.
  + *l* = 0 nếu x chỉ chứa 1 loại ký tự, vd: aaaaaa.
  + *l* = vị trí của ký tự đầu tiền của X khác với ký tự x0.

Bước 2 (Tìm kiếm):

* Việc so sánh được tính toán trên mô hình các vị trí theo thứ tự sau:

*l*, *l* + 1, …, *m* – 2, *m* – 1, 0, 1, …, *l* – 1.

* Xét bộ ba *(i, j, k)* thỏa mãn:
  + Cửa sổ chạy được xác định bởi factor [, , …, ]



* + 0 *kl* và [, , …, ] = [, , …, ]



* + *l im* và [, , …, ] = [, , …, ]



* Cách tính toán bộ ba *(i, j, k)* tiêp theo:

If ( *i* = l ) {

If( = ){



Bộ 3 tiếp theo là (*i* + 1, *j*, *k*)

}else{

Bộ 3 tiếp theo là (*l*, *j+1*, max{0, *k – 1}*)

}

}else if( *l < i < m* ) {

If( = ){



Bộ 3 tiếp theo là (*i* + 1, *j*, *k*)

}else{

If( kmpNext[i]*l*){



Bộ 3 tiếp theo là (*l*, *i* + *j – kmpNext[i]*, max{0, *kmpNext[i]*, *}*))

}else{

Bộ 3 tiếp theo là (*kmpNext[i]*, *i* + *j – kmpNext[i]*, *l*)

}

}

}else if( i = m ){

If ( *kl*&& = ){



Bộ 3 tiếp theo là (*i*, *j*, *k +* 1)

}else{

If ( k = l ){

<tìm thấy mẫu ở vị trí j – i >

}

Tính bộ 3 theo trường hợp *l < i < m.*

}

}

## Mã hóa thuật toán

void preKmp(char \*x, int m, int kmpNext[]) {

int i, j;

i = 0;

j = kmpNext[0] = -1;

while (i < m) {

while (j > -1 && x[i] != x[j])

j = kmpNext[j];

i++;

j++;

if (x[i] == x[j])

kmpNext[i] = kmpNext[j];

else

kmpNext[i] = j;

}

}

void AXAMAC(char \*x, int m, char \*y, int n) {

int i, j, k, ell, kmpNext[XSIZE];

/\* Preprocessing \*/

preKmp(x, m, kmpNext);

for (ell = 1; x[ell - 1] == x[ell]; ell++);

if (ell == m)

ell = 0;

/\* Searching \*/

i = ell;

j = k = 0;

while (j <= n - m) {

while (i < m && x[i] == y[i + j])

++i;

if (i >= m) {

while (k < ell && x[k] == y[j + k])

++k;

if (k >= ell)

OUTPUT(j);

}

j += (i - kmpNext[i]);

if (i == ell)

k = MAX(0, k - 1);

else

if (kmpNext[i] <= ell) {

k = MAX(0, kmpNext[i]);

i = ell;

}

else {

k = ell;

i = kmpNext[i];

}

}

}

## Kiểm nghiệm thuật toán

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Vòng lặp thứ nhất | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | 1 | 2 | 3 |  | | | | | | | | | | | | | | | | | | | |
| G | C | A | G | A | G | A | G |  | | | | | | | | | | | | | | | |

Số bước dịch chuyển: 4 (*i*-*kmpNext*[*i*]=3- -1)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Vòng lặp thứ hai | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | 1 |  | | | | | | | | | | | | | | | | | |
|  | | | | G | C | A | G | A | G | A | G |  | | | | | | | | | | | |

Số bước dịch chuyển: 1 (*i*-*kmpNext*[*i*]=1-0)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Vòng lặp thứ ba | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | 8 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |  | | | | | | | | | | |
|  | | | | | G | C | A | G | A | G | A | G |  | | | | | | | | | | |

Số bước dịch chuyển: 7 (*i*-*kmpNext*[*i*]=8-1)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Vòng lặp thứ tư | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | 1 |  | | | | | | | | | |
|  | | | | | | | | | | | | G | C | A | G | A | G | A | G |  | | | |

Số bước dịch chuyển: 1 (*i*-*kmpNext*[*i*]=1-0)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Vòng lặp thứ năm | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | 1 |  | | | | | | | | |
|  | | | | | | | | | | | | | G | C | A | G | A | G | A | G |  | | |

Số bước dịch chuyển: 1 (*i*-*kmpNext*[*i*]=1-0)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Vòng lặp thứ sáu | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | | 1 |  | | | | | | | |
|  | | | | | | | | | | | | | | G | C | A | G | A | G | A | G |  | |

Số bước dịch chuyển: 1 (*i*-*kmpNext*[*i*]=1-0)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Vòng lặp thứ bảy | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | | | 1 |  | | | | | | |
|  | | | | | | | | | | | | | | | G | C | A | G | A | G | A | G |  |

Số bước dịch chuyển: 1 (*i*-*kmpNext*[*i*]=1-0)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Vòng lặp thứ bảy | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | | | | 1 | 2 | 3 | 4 |  | | |
|  | | | | | | | | | | | | | | | | G | C | A | G | A | G | A | G |

# THUẬT TOÁN NOT SO NAIVE

## Ðặc ðiểm

* Thực hiện từ phải sang trái.
* Pha tiền xử lý có độ phức tạp hằng số.
* Độ phức tạp về không gian là hằng số
* Pha tìm kiếm có độ phức tạp thuật toán là O(n.m);

## Mô tả thuật toán

**Input :**

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

**Output:**

* Đưa ra mọi vị trí xuất hiện của X trong Y.

***Formats: k = NSN(X, m, Y, n);***

**Actions:**

Bước 1 (Tiền xử lý): Xác định số bước dịch chuyển SubY tren Y

- Gọi:

+ SubY: chuỗi con gồm m phần tử (y[j.. j+m-1]) của Y

+ k: số bước dịch khi x[1] != SubY[1]

+ l: số bước dịch khi x[1] = SubY[1]

- Xác định số dịch chuyển chuỗi con SubY khi chưa thỏa mãn:

+ Nếu (x[0] = x[1] && x[1] != y[j+1]) --> x[0] != y[j+1] --> SubY dịch chuyển 2 đơn vị j = j+2 --> k =2

+ Nếu (x[0] != x[1] && x[1] = y[j+1]) --> x[0] != y[j+1] --> SubY dịch chuyển 2 đơn vị j = j+2 --> l =2

+ Còn lại SubY dịch chuyển 1 đơn vị j = j+1

/\* Preprocessing \*/

if (x[0] == x[1]) {

k = 2; // dịch chuyển 2 nếu x[1] != suby[1]

l = 1; // ngược lại dịch chuyển 1

}

else {

k = 1; // dịch chuyển 1 nếu x[1] != suby[1]

l = 2; // ngược lại dịch chuyển 2

}

Bước 2 (Tìm kiếm):

• Ký tự của SubY được so sánh với chuỗi mẫu X theo thứ tự 1, 2, ... , m-2, m-1, 0

• Nếu SubY = chuỗi mẫu thì in ra j (vị trí bắt đầu của SubY trên Y)

• Ngược lại di chuyển SubY trên Y

**EndActions.**

## Mã hóa thuật toán

void NSN(char \*x, int m, char \*y, int n) {

int j, k, ell;

/\* Preprocessing \*/

if (x[0] == x[1]) {

k = 2;

ell = 1;

}

else {

k = 1;

ell = 2;

}

/\* Searching \*/

j = 0;

while (j <= n - m)

if (x[1] != y[j + 1])

j += k;

else {

if (memcmp(x + 2, y + j + 2, m - 2) == 0 &&

x[0] == y[j])

OUTPUT(j);

j += ell;

}

}

## Kiểm nghiệm thuật toán

* Pha tiền xử lý

X[0] = ‘G’

X[1] = ‘C’

X[0] != X[1] nên k = 1, l =2

* Pha tìm kiếm

J = 0, k =1, l =2

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Lần 1 | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | 1 | 2 | 3 |  | | | | | | | | | | | | | | | | | | | |
| G | C | A | G | A | G | A | G |  | | | | | | | | | | | | | | | |

Do X[1] = SubY[1], số bước dịch chuyển: 2 (*l*)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Lần 2 | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | 1 |  | | | | | | | | | | | | | | | | | | | |
|  | | G | C | A | G | A | G | A | G |  | | | | | | | | | | | | | |

Do X[1] != SubY[1], số bước dịch chuyển: 1 (*k*)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Lần 3 | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | 1 | 2 |  | | | | | | | | | | | | | | | | | |
|  | | | G | C | A | G | A | G | A | G |  | | | | | | | | | | | | |

Dịch chuyển: 2 (*l*)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Lần 4 | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | 8 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |  | | | | | | | | | | |
|  | | | | | G | C | A | G | A | G | A | G |  | | | | | | | | | | |

In ra vị trí chuỗi SubY thỏa mãn: 5 và dịch chuyển: 2 (*l*)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Lần 5 | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | 1 |  | | | | | | | | | | | | | | |
|  | | | | | | | G | C | A | G | A | G | A | G |  | | | | | | | | |

Dịch chuyển: 1 (*k*)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Lần 6 | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | 1 |  | | | | | | | | | | | | | |
|  | | | | | | | | G | C | A | G | A | G | A | G |  | | | | | | | |

Dịch chuyển: 1 (*k*)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Lần 7 | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | 1 |  | | | | | | | | | | | | |
|  | | | | | | | | | G | C | A | G | A | G | A | G |  | | | | | | |

Dịch chuyển: 1 (*k*)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Lần 8 | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | 1 |  | | | | | | | | | | | |
|  | | | | | | | | | | G | C | A | G | A | G | A | G |  | | | | | |

Dịch chuyển: 1 (*k*)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Lần 9 | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | 1 |  | | | | | | | | | | |
|  | | | | | | | | | | | G | C | A | G | A | G | A | G |  | | | | |

Dịch chuyển: 1 (*k*)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Lần 10 | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | 1 |  | | | | | | | | | |
|  | | | | | | | | | | | | G | C | A | G | A | G | A | G |  | | | |

Dịch chuyển: 1 (*k*)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Lần 11 | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | 1 |  | | | | | | | | |
|  | | | | | | | | | | | | | G | C | A | G | A | G | A | G |  | | |

Dịch chuyển: 1 (*k*)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Lần 12 | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | | 1 |  | | | | | | | |
|  | | | | | | | | | | | | | | G | C | A | G | A | G | A | G |  | |

Dịch chuyển: 1 (*k*)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Lần 13 | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | | | 1 |  | | | | | | |
|  | | | | | | | | | | | | | | | G | C | A | G | A | G | A | G |  |

Dịch chuyển: 1 (*k*)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Lần 14 | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | | | | 1 | 2 | 3 | 4 |  | | |
|  | | | | | | | | | | | | | | | | G | C | A | G | A | G | A | G |

Dịch chuyển: 2 (*l*)

Thuật toán dừng. Kết quả in ra vị trí bắt đầu của chuỗi con: 5.

# THUẬT TOÁN BOYER-MOORE

## Đặc điểm chính

* Thực hiện việc so sánh từ phải sang trái.
* Giai đoạn tiền xử lý (preprocessing) có độ phức tạp thời gian và không gian là O(m+σ).
* Giai đoạn tìm kiếm có độ phức tạp O(m\*n).
* So sánh tối đa 3n ký tự trong trường hợp xấu nhất đối với mẫu không có chu kỳ (non periodic pattern).
* Độ phức tạp O(n/m) trong trường hợp tốt nhất.

## Mô tả thuật toán

Thuật toán Boyer-Moore được coi là thuật toán hiệu quả nhất trong vấn đề tìm kiếm chuỗi (string-matching) trong các ứng dụng thường gặp. Các biến thể của nó thường được dùng trong các bộ soạn thảo cho các lệnh như <<search>> và <<subtitute>>.

Thuật toán sẽ quét các ký tự của mẫu (pattern) từ phải sang trái bắt đầu ở phần tử cuối cùng.

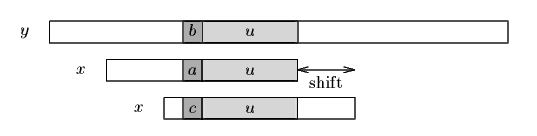
Trong trường hợp mis-match (hoặc là trường hợp đã tìm được 1 đoạn khớp với mẫu), nó sẽ dùng 2 hàm được tính toán trước để dịch cửa sổ sang bên phải. Hai hàm dịch chuyển này được gọi là good-suffix shift (còn được biết với cái tên phép dịch chuyển khớp) và bad-character shift (còn được biết với cái tên phép dịch chuyển xuất hiện).

Đối với mẫu x[0..m-1] ta dùng 1 biến chỉ số i chạy từ cuối về đầu, đối với chuỗi y[0..n-1] ta dùng 1 biến j để chốt ở phía đầu.

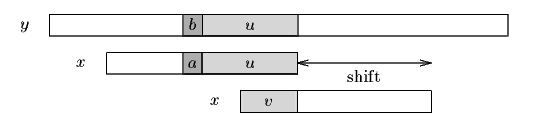
G/s rằng trong quá trình so sánh ta gặp 1 mis-match tai vị trí x[i]=a của mẫu và y[i+j]=b trong khi đang thử khớp tại vị trí j.

Khi đó, x[i+1..m-1]=y[j+i+1..j+m-1]=u và x[i]≠y[i+j] . Bây giờ ta đi xét xem đối với từng trường hợp, 2 hàm trên sẽ thực hiện việc dịch chuyển như thế nào:

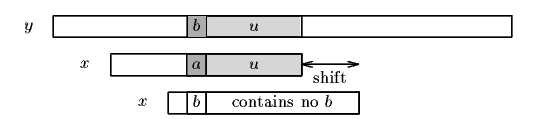
Phép dịch chuyển good-suffix shift sẽ dịch cửa sổ sang bên phải cho đến khi gặp 1 ký tự khác với x[i] trong trường hợp đoạn u lại xuất hiện trong x.



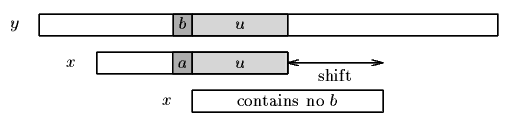
Nếu đoạn u không xuất hiện lại trong x, mà chỉ có 1 phần cuối (suffix) của u khớp với phần đầu (prefix) của x, thì ta sẽ dịch 1 đoạn sao cho phần suffix dài nhất v của y[j+i+1..j+m-1] khớp với prefix của x.



Phép dịch chuyển bad-character shift sẽ khớp kí tự y[i+j] với 1 ký tự (bên phải nhất) trong đoạn x[0..m-2] (các bạn thử nghĩ xem tại sao không phải là m-1)



Nếu y[i+j] không xuất hiện trong x, ta thấy ngay rằng không có xuất hiện nào của x trong y mà lại chứa chấp y[i+j], do đó ta có thể đặt cửa sổ ngay sau y[i+j], tức là y[j+i+1] .



Thuật toán Boyer-Moore sẽ chọn đoạn dịch chuyển dài nhất trong 2 hàm dịch chuyển good-suffix shift và bad-character shift. Hai hàm này được định nghĩa như sau:  
Hàm good-suffix shift được lưu trong bảng bmGs có kích thước m+1.  
Ta định nghĩa 2 điều kiện sau:

* Cs(i, s): với mỗi k mà i < k < m, s ≥ k hoặc x[k-s]=x[k] và
* Co(i, s): nếu s <i thì x[i-s] ≠ x[i]

Khi đó, với 0≤ i <m: bmGs[i+1]=min{s>0 : Cs(i, s) and Co(i, s) hold}  
và chúng ta định nghĩa bmGs[0] là độ dài chu kỳ của x. Việc tính toán bảng bmGs sử dụng 1 bảng suff định nghĩa như sau: với 1 ≤ i < m, suff[i]=max{k : x[i-k+1 .. i]=x[m-k .. m-1]}

Hàm bad-character shift được lưu trong bảng bmBc có kích thước σ. Cho c trong Σ : bmBc[c] = min{i : 1≤ i <m-1 và x[m-1-i]=c} nếu c xuất hiện trong x, m ngược lại.

Bảng bmGs và bmBc được tính toán trong thời gian O(m+σ) trước khi thực hiện tìm kiếm và cần 1 không gian phụ là O(m+σ). Giai đoạn tìm kiếm có độ phức tạp thời gian bậc hai nhưng lại chỉ có 3n phép so sánh khi tìm kiếm 1 chuỗi không có chu kì. Đối với việc tìm kiếm trong 1 khối lượng lớn các chữ cái thuật toán thực hiện với tốc độ nhanh “khủng khiếp”. Khi tìm kiếm chuỗi am-1b trong bn chuỗi thuật toán chỉ sử dụng O(n/m) phép so sánh, đây được coi là “cảnh giới” cho bất cứ một thuật toán tìm kiếm chuỗi nào mà mẫu đã được xử lý trước.

## Mã hóa thuật toán

void preBmBc(char \*x, int m, int bmBc[]) {

int i;

for (i = 0; i < ASIZE; ++i)

bmBc[i] = m;

for (i = 0; i < m - 1; ++i)

bmBc[x[i]] = m - i - 1;

}

void suffixes(char \*x, int m, int \*suff) {

int f, g, i;

suff[m - 1] = m;

g = m - 1;

for (i = m - 2; i >= 0; --i) {

if (i > g && suff[i + m - 1 - f] < i - g)

suff[i] = suff[i + m - 1 - f];

else {

if (i < g)

g = i;

f = i;

while (g >= 0 && x[g] == x[g + m - 1 - f])

--g;

suff[i] = f - g;

}

}

}

void preBmGs(char \*x, int m, int bmGs[]) {

int i, j, suff[XSIZE];

suffixes(x, m, suff);

for (i = 0; i < m; ++i)

bmGs[i] = m;

j = 0;

for (i = m - 1; i >= 0; --i)

if (suff[i] == i + 1)

for (; j < m - 1 - i; ++j)

if (bmGs[j] == m)

bmGs[j] = m - 1 - i;

for (i = 0; i <= m - 2; ++i)

bmGs[m - 1 - suff[i]] = m - 1 - i;

}

void BM(char \*x, int m, char \*y, int n) {

int i, j, bmGs[XSIZE], bmBc[ASIZE];

/\* Preprocessing \*/

preBmGs(x, m, bmGs);

preBmBc(x, m, bmBc);

/\* Searching \*/

j = 0;

while (j <= n - m) {

for (i = m - 1; i >= 0 && x[i] == y[i + j]; --i);

if (i < 0) {

OUTPUT(j);

j += bmGs[0];

}

else

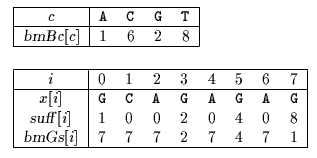
j += MAX(bmGs[i], bmBc[y[i + j]] - m + 1 + i);

}

}

## Kiểm nghiệm thuật toán

Pha tiền xử lý:



Bảng bmBc và bmBs sử dụng trong pha tìm kiếm

Pha tìm kiếm:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| First attempt | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | 1 |  | | | | | | | | | | | | | | | |
| G | C | A | G | A | G | A | G |  | | | | | | | | | | | | | | | |

Shift by: 1 (*bmGs*[7]=*bmBc*[A]-8+8)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Second attempt | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | 3 | 2 | 1 |  | | | | | | | | | | | | | | |
|  | G | C | A | G | A | G | A | G |  | | | | | | | | | | | | | | |

Shift by: 4 (*bmGs*[5]=*bmBc*[C]-8+6

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Third attempt | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |  | | | | | | | | | | |
|  | | | | | G | C | A | G | A | G | A | G |  | | | | | | | | | | |

Shift by: 7 (*bmGs*[0])

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Fourth attempt | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | | | | 3 | 2 | 1 |  | | | |
|  | | | | | | | | | | | | G | C | A | G | A | G | A | G |  | | | |

Shift by: 4 (*bmGs*[5]=*bmBc*[C]-8+6)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Fifth attempt | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | | | | | | | | | 2 | 1 |
|  | | | | | | | | | | | | | | | | G | C | A | G | A | G | A | G |

Shift by: 7 (*bmGs*[6])

Thuật toán Boyer-Moore thực hiện 17 lượt so sánh ký tự trong ví dụ trên.

# THUẬT TOÁN TURBO BOYER-MOORE

## Đặc điểm chính

* Đây là 1 biến thể của thuật toán Boyer-Moore
* Không yêu cầu pha tiền xử lý như thuật toán Boyer-Moore
* Cần không gian nhớ phụ như với thuật toán Boyer-Moore;
* Pha tiền xử lý có độ phức tạp O(m+σ);
* Pha tìm kiếm có độ phức tạp thuật toán là O(n);

## Mô tả thuật toán

Thuật toán Turbo-Bm là 1 thuật toán được cải thiện từ thuật toán Boyer-Moore. Thuật toán này không cần pha tiền xử lý mà chỉ cần cung cấp không gian nhớ phụ . Nó bao gồm việc ghi nhớ các thành phần của các kí tự khớp với hậu tố của các mẫu trong lần thử cuối cùng ( và chỉ thực hiện nếu có sự dịch hậu tố tốt được thực hiện).

Kỹ thuật này có 2 lợi ích:

- Có thể nhảy qua thành phần

- Có thể thực hiện dịch chuyển nhanh

Một dịch chuyển nhanh có thể thực hiện ra nếu trong quá trình xử lý hiện tại của các hậu tố của mẫu phù hợp với các kí tự ngắn hơn các xử lý trước đó. Trong trường hợp này chúng ta hãy gọi u là yếu tố nhớ và v là các hậu tố xuất hiện trong các xử lý hiện tại như vậy uzv là một hậu tố của x. Hãy để a và b là các phần tử không phù hợp trong các xử lý hiện tại trong các mẫu và các kĩ tự tương ứng. Sau đó, av là một hậu tố của x. Hai kí tự a,b xảy ra tại khoảng cách p trong chuỗi kí tự, và các hậu tố của x có chiều dài |uzv| có một đọ dài thời gian là p = |ZV| kể từ u là một biên giới của uzv, do đó nó không thể chồng lên nhau cả hai lần xuất hiện của hai nhân vật a và b khác nhau, tại khoảng cách p trong chuỗi kí tự. Sự dịch chuyển nhỏ nhất có thể có chiều dài |u| - |v|, mà chúng ta gọi là dịch chuyển nhanh (turbo-shift)

Tuy nhiên trong trường hợp nơi | v | <| u | nếu chiều dài của sự thay đổi phần tử-tồi lớn hơn độ dài của sự dịch chuyển của hậu tố - tốt và độ dài của dịch chuyển nhanh sau đó chiều dài của sự chuyển đổi thực tế phải lớn hơn hoặc bằng |u| + 1.

Thật vậy (xem hình 15.2), trong trường hợp này hai phần tử c và d là khác nhau vì chúng ta giả định rằng sự thay đổi trước đó là một sự dịch chuyển hậu tố tốt. Sau đó, một sự thay đổi lớn hơn dịch chuyển nhanh nhưng nhỏ hơn |u| + 1 sẽ sắp xếp c và d, với một kí tự tương tự trong v. Vì vậy, nếu trường hợp này chiều dài của sự thay đổi thực tế phải có ít nhất bằng |u| + 1.

Giai đoạn tiền xử lý có thể được thực hiện trong O (m+ σ) thời gian. Giai đoạn tìm kiếm là O (n). Các số so sánh phần tử trong đoạn mã được thực hiện bởi các thuật toán Turbo-BM được bao bọc bởi 2n.

## Mã hóa thuật toán

void TBM(char \*x, int m, char \*y, int n) {

int bcShift, i, j, shift, u, v, turboShift,

bmGs[XSIZE], bmBc[ASIZE];

/\* Preprocessing \*/

preBmGs(x, m, bmGs);

preBmBc(x, m, bmBc);

/\* Searching \*/

j = u = 0;

shift = m;

while (j <= n - m) {

i = m - 1;

while (i >= 0 && x[i] == y[i + j]) {

--i;

if (u != 0 && i == m - 1 - shift)

i -= u;

}

if (i < 0) {

OUTPUT(j);

shift = bmGs[0];

u = m - shift;

}

else {

v = m - 1 - i;

turboShift = u - v;

bcShift = bmBc[y[i + j]] - m + 1 + i;

shift = MAX(turboShift, bcShift);

shift = MAX(shift, bmGs[i]);

if (shift == bmGs[i])

u = MIN(m - shift, v);

else {

if (turboShift < bcShift)

shift = MAX(shift, u + 1);

u = 0;

}

}

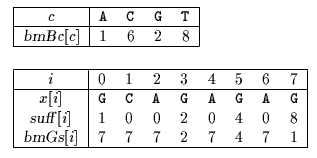
j += shift;

}

}

## Kiểm nghiệm thuật toán

Pha tiền xử lý:



Bảng bmBc và bmGs dùng trong pha tìm kiếm

Pha tìm kiếm:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| First attempt | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | 1 |  | | | | | | | | | | | | | | | |
| G | C | A | G | A | G | A | G |  | | | | | | | | | | | | | | | |

Shift by: 1 (*bmGs*[7]=*bmBc*[A]-8+8)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Second attempt | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | 3 | 2 | 1 |  | | | | | | | | | | | | | | |
|  | G | C | A | G | A | G | A | G |  | | | | | | | | | | | | | | |

Shift by: 4 (*bmGs*[5]=*bmBc*[C]-8+6)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Third attempt | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | 6 | 5 | - | - | 4 | 3 | 2 | 1 |  | | | | | | | | | | |
|  | | | | | G | C | A | G | A | G | A | G |  | | | | | | | | | | |

Shift by: 7 (*bmGs*[0])

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Fourth attempt | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | | | | 3 | 2 | 1 |  | | | |
|  | | | | | | | | | | | | G | C | A | G | A | G | A | G |  | | | |

Shift by: 4 (*bmGs*[5]=*bmBc*[C]-8+6)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Fifth attempt | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | | | | | | | | | 2 | 1 |
|  | | | | | | | | | | | | | | | | G | C | A | G | A | G | A | G |

Shift by: 7 (*bmGs*[6])

Thuật toán Turbo Boyer-Moore thực hiện 15 so sánh ký tự trong ví dụ nêu trên.

# THUẬT TOÁN FORWARD DAWG MATCHING

## Các tính nãng chính

* Sử dụng automaton hậu tố của x.
* Độ phức tạp thời gian trong trường hợp xấu nhất O (n).
* Thực hiện chính xác kiểm tra nkí tự văn bản.

## Mô tả thuật toán

Các thuật toán Forward Dawg Matching tính toán các yếu tố dài nhất của mô hình kết thúc tại mỗi vị trí trong văn bản. Điều này là bằng cách có thể sử dụng automaton hậu tố nhỏ nhất (cũng gọi là Dawg cho Directed Acyclic Word Graph) của mô hình. Các hậu tố tự động nhỏ nhất một từ w là một automaton hữu hạn xác định S (w) = (Q, q 0, T, E). Ngôn ngữ được chấp nhận bởi S (w) là L (S (w)) = { u trong http://www.htmlpublish.com/newTestDocStorage/DocStorage/3332e9acde324f5fa14e33fc4f201673/%5B123doc.vn%5D%20-%20cac-thuat-toan-tim-kiem-va-so-khop-chuoi_images/%5B123doc.vn%5D%20-%20cac-thuat-toan-tim-kiem-va-so-khop-chuoi38xi2.jpg\*: Tồn tại v http://www.htmlpublish.com/newTestDocStorage/DocStorage/3332e9acde324f5fa14e33fc4f201673/%5B123doc.vn%5D%20-%20cac-thuat-toan-tim-kiem-va-so-khop-chuoi_images/%5B123doc.vn%5D%20-%20cac-thuat-toan-tim-kiem-va-so-khop-chuoi38xi3.jpg\* W = vu}.

Giai đoạn tiền xử lý của các thuật toán phù hợp Forward Dawg bao gồm trong việc tính toán các hậu tố tự động nhỏ nhất cho mẫu x. Nó là tuyến tính trong thời gian và không gian trong chiều dài của mô hình.

Trong giai đoạn tìm kiếm các thuật toán Forward Dawg Matching phân tích các kí tự của văn bản từ trái sang phải với S automaton (x) bắt đầu với trạng thái q0. Đối với mỗi trạng thái q trong S (x) được ký hiệu là con đường dài nhất từ q 0 để p theo chiều dài (q). Cấu trúc này sử dụng rộng rãi các khái niệm liên kết hậu tố. Đối với mỗi p, các liên kết hậu tố của p được kýhiệu là S [p ]. Đối với một trạng p, hãy để đường dẫn (p) = (0 p, p, 1 ..., p http://www.htmlpublish.com/newTestDocStorage/DocStorage/3332e9acde324f5fa14e33fc4f201673/%5B123doc.vn%5D%20-%20cac-thuat-toan-tim-kiem-va-so-khop-chuoi_images/%5B123doc.vn%5D%20-%20cac-thuat-toan-tim-kiem-va-so-khop-chuoi38xi4.jpg) Được con đường hậu tố của p p 0 = p, cho 1 http://www.htmlpublish.com/newTestDocStorage/DocStorage/3332e9acde324f5fa14e33fc4f201673/%5B123doc.vn%5D%20-%20cac-thuat-toan-tim-kiem-va-so-khop-chuoi_images/%5B123doc.vn%5D%20-%20cac-thuat-toan-tim-kiem-va-so-khop-chuoi38xi5.jpgi http://www.htmlpublish.com/newTestDocStorage/DocStorage/3332e9acde324f5fa14e33fc4f201673/%5B123doc.vn%5D%20-%20cac-thuat-toan-tim-kiem-va-so-khop-chuoi_images/%5B123doc.vn%5D%20-%20cac-thuat-toan-tim-kiem-va-so-khop-chuoi38xi6.jpg, P i = S [p i -1] và p http://www.htmlpublish.com/newTestDocStorage/DocStorage/3332e9acde324f5fa14e33fc4f201673/%5B123doc.vn%5D%20-%20cac-thuat-toan-tim-kiem-va-so-khop-chuoi_images/%5B123doc.vn%5D%20-%20cac-thuat-toan-tim-kiem-va-so-khop-chuoi38xi7.jpg= Q 0. Đối với mỗi ký tự văn bản y [j] tuần tự,thay p là trạng thái hiện tại, sau đó các thuật toán Forward Dawg Matching có một sự chuyển đổi quy định cho y [j] cho các tiểu bang đầu tiên của Con Đường (p) cho quá trình chuyển đổi như vậy được định nghĩa. Trạng thái P hiện tại được cập nhật với các mục tiêu trạng thái của quá trình chuyển đổi này hoặc với tình trạng ban *đầu*q0 nếu quá trình chuyển đổi không có dán nhãn *với*y *[j]*từ một trạng thái *của*Con Đường *(p)*.Một sự xuất hiện của *x*được tìm thấy khi *chiều dài (p)*= *m.*Các thuật toán Forward Dawg phù hợp thực hiện kiểm tra văn bản ký tự *chính*xác n.

## Mã hóa thuật toán

void buildSuffixAutomaton(char \*x, int m, Graph aut) {

int i, art, init, last, p, q, r;

char c;

init = getInitial(aut);

art = newVertex(aut);

setSuffixLink(aut, init, art);

last = init;

for (i = 0; i < m; ++i) {

c = x[i];

p = last;

q = newVertex(aut);

setLength(aut, q, getLength(aut, p) + 1);

setPosition(aut, q, getPosition(aut, p) + 1);

while (p != init &&

getTarget(aut, p, c) == UNDEFINED) {

setTarget(aut, p, c, q);

setShift(aut, p, c, getPosition(aut, q) -

getPosition(aut, p) - 1);

p = getSuffixLink(aut, p);

}

if (getTarget(aut, p, c) == UNDEFINED) {

setTarget(aut, init, c, q);

setShift(aut, init, c,

getPosition(aut, q) -

getPosition(aut, init) - 1);

setSuffixLink(aut, q, init);

}

else

if (getLength(aut, p) + 1 ==

getLength(aut, getTarget(aut, p, c)))

setSuffixLink(aut, q, getTarget(aut, p, c));

else {

r = newVertex(aut);

copyVertex(aut, r, getTarget(aut, p, c));

setLength(aut, r, getLength(aut, p) + 1);

setSuffixLink(aut, getTarget(aut, p, c), r);

setSuffixLink(aut, q, r);

while (p != art &&

getLength(aut, getTarget(aut, p, c)) >=

getLength(aut, r)) {

setShift(aut, p, c,

getPosition(aut,

getTarget(aut, p, c)) -

getPosition(aut, p) - 1);

setTarget(aut, p, c, r);

p = getSuffixLink(aut, p);

}

}

last = q;

}

setTerminal(aut, last);

while (last != init) {

last = getSuffixLink(aut, last);

setTerminal(aut, last);

}

}

int FDM(char \*x, int m, char \*y, int n) {

int j, init, ell, state;

Graph aut;

/\* Preprocessing \*/

aut = newSuffixAutomaton(2\*(m + 2), 2\*(m + 2)\*ASIZE);

buildSuffixAutomaton(x, m, aut);

init = getInitial(aut);

/\* Searching \*/

ell = 0;

state = init;

for (j = 0; j < n; ++j) {

if (getTarget(aut, state, y[j]) != UNDEFINED) {

++ell;

state = getTarget(aut, state, y[j]);

}

else {

while (state != init &&

getTarget(aut, state, y[j]) == UNDEFINED)

state = getSuffixLink(aut, state);

if (getTarget(aut, state, y[j]) != UNDEFINED) {

ell = getLength(aut, state) + 1;

state = getTarget(aut, state, y[j]);

}

else {

ell = 0;

state = init;

}

}

if (ell == m)

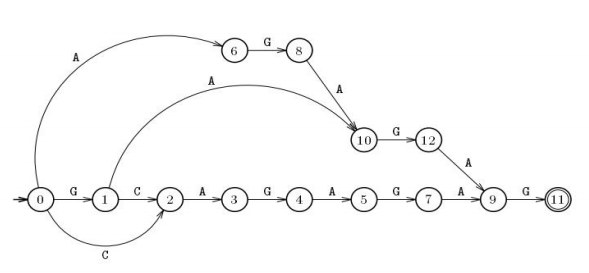
OUTPUT(j - m + 1);

}

}

## Kiểm nghiệm thuật toán

***Giai đoạn tiền xử lý***



Hậu tố tự động được sử dụng bởi Forward Dawg thuật toán tìm kiếm Matching.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Biến trạng thái | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Hậu tố liên kết | 0 | 0 | 0 | 6 | 8 | 10 | 0 | 12 | 1 | 10 | 6 | 12 | 8 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Chiều dài | 0 | 1 | 2 | 3 | 4 | 5 | 1 | 6 | 2 | 7 | 3 | 8 | 4 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |

<http://translate.googleusercontent.com/translate_c?hl=vi&prev=/search%3Fq%3Dt%25C3%25A0i%2Bli%25E1%25BB%2587u%2Bv%25E1%25BB%2581%2BForward%2BDawg%2BMatching%2Balgorithm%26hl%3Dvi%26biw%3D1280%26bih%3D800%26prmd%3Dimvns&rurl=translate.google.com.vn&sl=en&u=http://igm.univ-mlv.fr/%7Elecroq/string/examples/exp36.html&usg=ALkJrhgiXWc5XAa7q4M-puNHbDnkOdyQuQ>***Giai đoạn tìm kiếm***

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| First attempt | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
| 1 | 2 | 3 | 0 |  | | | | | | | | | | | | | | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Second attempt | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | 2 | 0 |  | | | | | | | | | | | | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Third attempt | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | 1 | 2 | 3 | 4 | 5 | 7 | 9 | 1 | 0 |  | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | - | - | - | - | - | - | - | 1 |  | | | | | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Fourth attempt | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | 6 | 0 |  | | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Fifth attempt | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | | | 6 | 2 | 3 | 4 | 0 |  | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Sixth attempt | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | | | | | | | | 6 | 2 | 1 |

Thuật toán Forward Dawg Matching thực hiện 24 so sánh ký tự trong ví dụ này.

# THUÂT TOÁN HORSPOOL

## Phát biểu thuật toán

* Là dạng đơn giản của thuật toán Boyer – Moore
* Chỉ dùng để thay đổi các ký tự xấu
* Dễ thực hiện
* Giai đoạn tiền xử lý độ phức tạp *O(m+)* thời gian và *O()* không gian



* Giai đoạn tìm kiếm phức tạp *O(m x n )* thời gian
* Chỉ số trung bình của sự so sánh cho 1 ký tự text là giữa *1/* và *2/( +1)*



## Mã hóa thuật toán

void preBmBc(char \*x, int m, int bmBc[]) {

int i;

for (i = 0; i < ASIZE; ++i)

bmBc[i] = m;

for (i = 0; i < m - 1; ++i)

bmBc[x[i]] = m - i - 1;

}

void HORSPOOL(char \*x, int m, char \*y, int n) {

int j, bmBc[ASIZE];

char c;

/\* Preprocessing \*/

preBmBc(x, m, bmBc);

/\* Searching \*/

j = 0;

while (j <= n - m) {

c = y[j + m - 1];

if (x[m - 1] == c && memcmp(x, y + j, m - 1) == 0)

OUTPUT(j);

j += bmBc[c];

}

}

## Kiểm nghiệmthuật toán

Horspool algorithm bmBc table

Bảng ký tự tồi

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bước 1: | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | 1 |  | | | | | | | | | | | | | | | |
| G | C | A | G | A | G | A | G |  | | | | | | | | | | | | | | | |

Dịch : 1 (*bmBc*[A])

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bước 2: | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | 2 |  |  |  |  |  |  | 1 |  | | | | | | | | | | | | | | |
|  | G | C | A | G | A | G | A | G |  | | | | | | | | | | | | | | |

Dịch: 2 (*bmBc*[G])

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bước 3: | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | 2 |  |  |  |  |  |  | 1 |  | | | | | | | | | | | | |
|  | | | G | C | A | G | A | G | A | G |  | | | | | | | | | | | | |

Dịch: 2 (*bmBc*[G])

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bước 4: | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 1 |  | | | | | | | | | | |
|  | | | | | G | C | A | G | A | G | A | G |  | | | | | | | | | | |

Dịch: 2 (*bmBc*[G])

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bước 5: | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | 1 |  | | | | | | | | |
|  | | | | | | | G | C | A | G | A | G | A | G |  | | | | | | | | |

Dịch: 1 (*bmBc*[A])

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bước 6: | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | | 1 |  | | | | | | | |
|  | | | | | | | | G | C | A | G | A | G | A | G |  | | | | | | | |

Dịch: 8 (*bmBc*[T])

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bước 7: | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | | | 2 |  |  |  |  |  |  | 1 |
|  | | | | | | | | | | | | | | | | G | C | A | G | A | G | A | G |

Dịch: 2 (*bmBc*[G])

Kết thúc thuật toán.

# THUẬT TOÁN QUICK SEARCH

## Đặc điểm chính

* Thuật toán là sự đơn giản hóa của thuật toán Boyer-Moore
* Sử dụng duy nhất bad-character shift
* Dễ dàng thực hiện
* Pha tiền xử lý có độ phức tạp về thời gian là O(m+σ) và không gian O(σ);
* Pha tìm kiếm có độ phức tạp thuật toán là O(mn);

## Mô tả thuật toán

**Input:**

* Xâu mẫu X = (, ,…, ), độ dài m.



* Văn bản Y = (, ,…, ), độ dài n



**Output:**

* Đưa ra mọi vị trí xuất hiện của X trong Y.

**Formats: Quick Search** (x,m,qsBc);

Thuật toán Quick Search sử dụng duy nhất bad-character shift. Sau khi dịch chuyển cửa sổ trượt của xâu con trên xâu gốc y[j .. j+m-1] thì độ dài của dịch chuyển vị trí tổi thiếu là 1.

Bad-character shift của thuật toán có sự thay đổi nhỏ để đưa vào giá trị cuối cùng của X như sau : for c in Sigma, *qsBc*[c]=*min*{i : 0  < i leq m and x[m-i]=c} if c occurs in x, m+1 otherwise

Trong suốt quá trình tìm kiếm xâu con với sự so sánh với xâu gốc, quá trình tìm kiếm sẽ không phân biệt theo thứ tự.

## Mã hóa thuật toán

void preQsBc(char \*x, int m, int qsBc[]) {

int i;

for (i = 0; i < ASIZE; ++i)

qsBc[i] = m + 1;

for (i = 0; i < m; ++i)

qsBc[x[i]] = m - i;

}

void QS(char \*x, int m, char \*y, int n) {

int j, qsBc[ASIZE];

/\* Preprocessing \*/

preQsBc(x, m, qsBc);

/\* Searching \*/

j = 0;

while (j <= n - m) {

if (memcmp(x, y + j, m) == 0)

OUTPUT(j);

j += qsBc[y[j + m]]; /\* shift \*/

}

}

## Kiểm nghiệm thuật toán

Pha tiền xử lý:

Quick Search algorithm qsBc table

Bảng qsBc dùng trong pha tìm kiếm

Pha tìm kiếm:

Vòng lặp thứ 1

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
| 1 | 2 | 3 | 4 |  | | | | | | | | | | | | | | | | | | | |
| G | C | A | G | A | G | A | G |  | | | | | | | | | | | | | | | |

Số bước dịch chuyển: 1 (*qsBc*[G])

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Vòng lặp thứ 2 | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | 1 |  | | | | | | | | | | | | | | | | | | | | | |
|  | G | C | A | G | A | G | A | G |  | | | | | | | | | | | | | | |

Số bước dịch chuyển: 2 (*qsBc*[A])

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Vòng lặp thứ 3 | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | 1 |  | | | | | | | | | | | | | | | | | | | |
|  | | | G | C | A | G | A | G | A | G |  | | | | | | | | | | | | |

Số bước dịch chuyển: 2 (*qsBc*[A])

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Vòng lặp thứ 4 | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |  | | | | | | | | | | |
|  | | | | | G | C | A | G | A | G | A | G |  | | | | | | | | | | |

Số bước dịch chuyển : 9 (*qsBc*[T])

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Vòng lặp thứ 5 | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | 1 |  | | | | | | | | |
|  | | | | | | | | | | | | | | G | C | A | G | A | G | A | G |  | |

Số bước dịch chuyển : 7 (*qsBc*[C])

# THUẬT TOÁN TUNED BOYER-MOORE

## Đặc điểm chính

* Là sự đơn giản hóa thuật toán Boyer-Moore;
* Dễ thực hiện;
* Thực hiện nhanh.

## Mô tả thuật toán

***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****:****TUNEDBM (char \*x, int m, char \*y, int n)***

**Actions**:

void TUNEDBM(char \*x, int m, char \*y, int n) {

int j, k, shift, bmBc[ASIZE];

/\* Preprocessing \*/

preBmBc(x, m, bmBc);

shift = bmBc[x[m - 1]];

bmBc[x[m - 1]] = 0;

memset(y + n, x[m - 1], m);

/\* Searching \*/

j = 0;

while (j < n) {

k = bmBc[y[j + m -1]];

while (k != 0) {

j += k; k = bmBc[y[j + m -1]];

j += k; k = bmBc[y[j + m -1]];

j += k; k = bmBc[y[j + m -1]];

}

if (memcmp(x, y + j, m - 1) == 0 && j < n)

OUTPUT(j);

j += shift; /\* shift \*/

}

}

**EndActions**.

## Mã hóa thuật toán

void preBmBc(char \*x, int m, int bmBc[]) {

int i;

for (i = 0; i < ASIZE; ++i)

bmBc[i] = m;

for (i = 0; i < m - 1; ++i)

bmBc[x[i]] = m - i - 1;

}

void TUNEDBM(char \*x, int m, char \*y, int n) {

int j, k, shift, bmBc[ASIZE];

/\* Preprocessing \*/

preBmBc(x, m, bmBc);

shift = bmBc[x[m - 1]];

bmBc[x[m - 1]] = 0;

memset(y + n, x[m - 1], m);

/\* Searching \*/

j = 0;

while (j < n) {

k = bmBc[y[j + m -1]];

while (k != 0) {

j += k; k = bmBc[y[j + m -1]];

j += k; k = bmBc[y[j + m -1]];

j += k; k = bmBc[y[j + m -1]];

}

if (memcmp(x, y + j, m - 1) == 0 && j < n)

OUTPUT(j);

j += shift; /\* shift \*/

}

}

## Kiểm nghiệm thuật toán

Y[]="ABABDABACDABAACABABE" n=20

X[]="ABAACABAB" m=9

BmBc[] các giá trị khác = m=9

|  |  |  |  |
| --- | --- | --- | --- |
|  | **A** | **B** | **C** |
| **bmbc** | **1** | **2** | **4** |

**Giai đoạn đầu**

shift=BmBc[ x[m - 1]]=BmBc[x[8]]= BmBc[B];

BmBc[ x[m - 1]]=BmBc[x[8]]= BmBc[B]=0;

Có lại mảng BmBc[]

|  |  |  |  |
| --- | --- | --- | --- |
|  | **A** | **B** | **C** |
| **bmbc** | **1** | **0** | **4** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| J+=shift<n=20? | k = bmBc[y[j + m -1]] | Nếu k!=0 lặp | J+=k; | k = bmBc[y[j + m -1]] | So sánh mảng x và y từ vị trí j với m-1phần tử và j>n? |
| 0(yes) | k=bmBc[Y[8]]= bmBc[C]=4 | Lặp |  |  |  |
|  |  |  | J=0+4=4 | k = bmBc[y[12]]= bmBc[A]=1 |  |
|  |  |  | J=4+1=5 | k = bmBc[y[13]]= bmBc[B]=0 |  |
|  |  |  | J=5+0=5 | k = bmBc[y[13]]= bmBc[B]=0 |  |
|  |  | Thoát lặp |  |  | “ABAACABA” vs “ABACDABA” (no)  J=5<n=20 (yes) |
| 7(yes) | k=bmBc[Y[15]]= bmBc[A]=1 | Lặp | J=7+1=8 | k = bmBc[y[16]]= bmBc[B]=0 |  |
|  |  |  | J=8+0=8 | k = bmBc[y[16]]= bmBc[B]=0 |  |
|  |  |  | J=8+0=8 | k = bmBc[y[16]]= bmBc[B]=0 |  |
|  |  | Thoát lặp |  |  | “ABAACABA” vs “CDABAACA” (no)  J=8<n=20 (yes) |
| 10(yes) | k=bmBc[Y[18]]= bmBc[B]=0 | Không lặp |  |  | **“ABAACABA” vs “ABAACABA” (yes)**  **J=10<n=20 (yes)** |
| 12(yes) | k=bmBc[Y[20]]= bmBc[B]=9 | Lặp | J=12+9=21 | k = bmBc[y[29]]= bmBc[B]=0 |  |
|  |  |  | J=21+0=21 | k = bmBc[y[29]]= bmBc[B]=0 |  |
|  |  |  | J=21+0=21 | k = bmBc[y[29]]= bmBc[B]=0 |  |
|  |  | Thoắt lặp |  |  | J=21>n=20 (yes) |
| 23(no) |  |  |  |  |  |

# THUẬT TOÁN ZHU – TAKAOKA

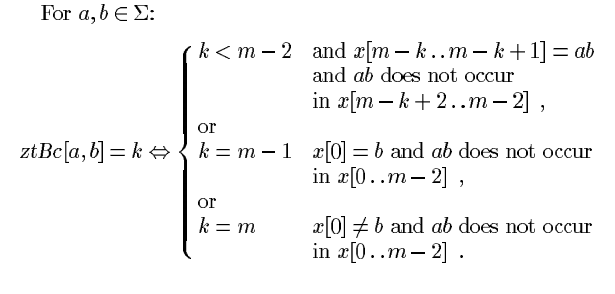
## Đặc điểm chính

* Biến thể của thuật toán Boyer - moore
* Sử dụng 2 ký tự văn bản liên tiếp để tính toán sự thay đổi ký tự xấu
* Giai đoạn tiền xử lý trong 0(m + a2) thời gian và phức tạp không gian
* Tìm kiếm trong giai đoạn 0 (m x n) thời gian phức tạp

## Mô tả thuật toán

Zhu và Takaoka thiết kế một thuật toán thực hiện sự thay đổi bằng cách xem xét các thay đổi xấu ký tự ( xem trang 14) cho 2 ký tự văn bản liên tiếp. Trong giai đoạn tìm kiếm so sánh được thực hiện từ phải sang trái và khi cửa sổ đặt trên nhân tố văn bản Y[j1] và một ghép đôi không xứng xảy ra giữa *x[m — k]* và *y[j + m — k]* khi đó *x[m — k + I . . m —* 1] = *y[j + m — k + I . . j + m —* 1] sự thay đổi là thực hiện với thay đổi ký tự xấu cho ký tự văn bản *y[j* + *m* — 2] và *y[j* + *m —* 1]. Bảng thay đổi good- sufEx cũng được dùng để tính toán sự thay đổi

Giai đoạn tiền xử lý của thuật toán bao gồm trong tính toán cho mỗi cặp ký tự (a,b) với a,b € ∑ xảy ra ở đầu bên phải của ab trong x[0..m-2]



Nó cũng bao gồm việc tính toán bang bmGs ( xem trang 14). Giai đoạn tiền xử lý là trong 0(m + 2) thời gian và phức tạp không gian

Giai đoạn tìm kiếm có một trường hợp xấu bậc 2

## Mã hóa thuật toán

void preZtBc(char \*x, int m, int ztBc[ASIZE][ASIZE]) {

int i, j;

for (i = 0; i < ASIZE; ++i)

for (j = 0; j < ASIZE; ++j)

ztBc[i][j] = m;

for (i = 0; i < ASIZE; ++i)

ztBc[i][x[0]] = m - 1;

for (i = 1; i < m - 1; ++i)

ztBc[x[i - 1]][x[i]] = m - 1 - i;

}

void ZT(char \*x, int m, char \*y, int n) {

int i, j, ztBc[ASIZE][ASIZE], bmGs[XSIZE];

/\* Preprocessing \*/

preZtBc(x, m, ztBc);

preBmGs(x, m, bmGs);

/\* Searching \*/

j = 0;

while (j <= n - m) {

i = m - 1;

while (i < m && x[i] == y[i + j])

--i;

if (i < 0) {

OUTPUT(j);

j += bmGs[0];

}

else

j += MAX(bmGs[i],

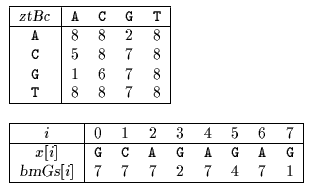
ztBc[y[j + m - 2]][y[j + m - 1]]);

}

}

## Kiểm nghiệm thuật toán

Pha tiền xử lý:



Bảng ztBc và bmGs được sử dụng trong pha tìm kiếm

Pha tìm kiếm:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| First attempt | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | 1 |  | | | | | | | | | | | | | | | |
| G | C | A | G | A | G | A | G |  | | | | | | | | | | | | | | | |

Shift by: 5 (*ztBc*[C][A])

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Second attempt | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |  | | | | | | | | | | |
|  | | | | | G | C | A | G | A | G | A | G |  | | | | | | | | | | |

Shift by: 7 (*bmGs*[0])

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Third attempt | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | | | | 3 | 2 | 1 |  | | | |
|  | | | | | | | | | | | | G | C | A | G | A | G | A | G |  | | | |

Shift by: 7 (*bmGs*[6])

Thuật toán Zhu-Takaoka thực hiện 12 soa sánh ký tự trong ví dụ trên.

# THUẬT TOÁN BERRY – RAVINDRAN

## Phát biểu thuật toán

* Sự pha trộn của thuật toán Quick Search và thuật toán Zhu-Takaoka
* Giai đoạn tiền xử lý độ phức tạp *O(m + 2)*



* Giai đoạn tìm kiếm độ phức tạp *O(m x n)*

## Mã hóa thuật toán

void preBrBc(char \*x, int m, int brBc[ASIZE][ASIZE]) {

int a, b, i;

for (a = 0; a < ASIZE; ++a)

for (b = 0; b < ASIZE; ++b)

brBc[a][b] = m + 2;

for (a = 0; a < ASIZE; ++a)

brBc[a][x[0]] = m + 1;

for (i = 0; i < m - 1; ++i)

brBc[x[i]][x[i + 1]] = m - i;

for (a = 0; a < ASIZE; ++a)

brBc[x[m - 1]][a] = 1;

}

void BR(char \*x, int m, char \*y, int n) {

int j, brBc[ASIZE][ASIZE];

/\* Preprocessing \*/

preBrBc(x, m, brBc);

/\* Searching \*/

y[n + 1] = '\0';

j = 0;

while (j <= n - m) {

if (memcmp(x, y + j, m) == 0)

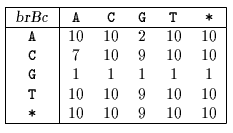
OUTPUT(j);

j += brBc[y[j + m]][y[j + m + 1]];

}

}

## Kiểm nghiệmthuật toán

  
Bảng ký tự tồi. Ký tự (\*) biểu thị cho các ký tự còn lại

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bước 1: | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
| 1 | 2 | 3 | 4 |  | | | | | | | | | | | | | | | | | | | |
| G | C | A | G | A | G | A | G |  | | | | | | | | | | | | | | | |

Dịch: 1 (*brBc*[G][A])

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bước 2: | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | 1 |  | | | | | | | | | | | | | | | | | | | | | |
|  | G | C | A | G | A | G | A | G |  | | | | | | | | | | | | | | |

Dịch: 2 (*brBc*[A][G])

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bước 3: | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | 1 |  | | | | | | | | | | | | | | | | | | | |
|  | | | G | C | A | G | A | G | A | G |  | | | | | | | | | | | | |

Dịch: 2 (*brBc*[A][G])

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bước 4: | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |  | | | | | | | | | | |
|  | | | | | G | C | A | G | A | G | A | G |  | | | | | | | | | | |

Dịch: 10 (*brBc*[T][A])

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bước 5: | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | | 1 |  | | | | | | | |
|  | | | | | | | | | | | | | | | G | C | A | G | A | G | A | G |  |

Dịch: 7 (*brBc*[G][0])

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bước 6: | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | | | 1 |  | | | | | | |
|  | | | | | | | | | | | | | | | | G | C | A | G | A | G | A | G |

Dịch: 10 (*brBc*[0][0])

Kết thúc thuật toán

# THUẬT TOÁN RAITA

## Các tính năng chính

* Trước tiên so sánh các ký tự của các mẫu cuối cùng, sau đó là mẫu đầu tiên và cuối cùng là một trong những so sánh khác nhau ở hiện tại.
* Thực hiện các thay đổi như ở thuật toán Horspool ( Trang 117 )
* Thời gian đưa vào từng bước xử lý trước là O(m + a) và độ phức tạp không gian O(a)
* Độ phức tạp của thời gian tìm kiếm là O(mxn).

## Mô tả thuật toán

Raita thiết kế một thuật toán mà tại mỗi nỗ lực đầu tiên so sánh các ký tự cuối cùng của mô hình với các ký tự bên phải của cửa sổ, sau đó nếu chúng phù hợp với các so sánh các ký tự đầu tiên của mô hình với các ký tự văn bản ngoài cùng bên trái của cửa sổ, rồi nếu chúng phù hợp với so sánh ký tự văn bản giữa các mô hình với giữa các ký tự của cửa sổ. Và cuối cùng của chúng phù hợp với các ký tự của mô hình khác từ thứ hai đến cuối, nhưng một, có thể so sánh một lần nữa giữa các ký tự này.

Raita quan sát thấy rằng thuật toán của nó đã có một tác động tốt trong thực tế khi tìm kiếm mô hình trong các văn bản tiếng Anh và các tác động đến sự tồn tại của các ký tự phụ thuộc. Smith thực hiện một số thí nghiệm và kết luận rằng hiện tượng này có thể thay thế bỡi các kết quả của trình biên dịch khác.

Giai đoạn tiền xử lý của thuật toán Raita bao gồm trong tính toán các chức năng chuyển đổi xấu ký tự (xem chương 14). Nó có thể được thực hiện trong thời gian O (m + a) và sự phức tạp không gian O (a)

Giai đoạn tìm kiếm của thuật toán Raita có một trường hợp xấu là nhất bậc hai về sự phức tạp thời gian.

## Mã hóa thuật toán

void preBmBc(char \*x, int m, int bmBc[]) {

int i;

for (i = 0; i < ASIZE; ++i)

bmBc[i] = m;

for (i = 0; i < m - 1; ++i)

bmBc[x[i]] = m - i - 1;

}

void RAITA(char \*x, int m, char \*y, int n) {

int j, bmBc[ASIZE];

char c, firstCh, \*secondCh, middleCh, lastCh;

/\* Preprocessing \*/

preBmBc(x, m, bmBc);

firstCh = x[0];

secondCh = x + 1;

middleCh = x[m/2];

lastCh = x[m - 1];

/\* Searching \*/

j = 0;

while (j <= n - m) {

c = y[j + m - 1];

if (lastCh == c && middleCh == y[j + m/2] &&

firstCh == y[j] &&

memcmp(secondCh, y + j + 1, m - 2) == 0)

OUTPUT(j);

j += bmBc[c];

}

}

## Kiểm nghiệm thuật toán

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a | A | C | G | T |
| bmBc[a] | 1 | 6 | 2 | 8 |

Searching phase

First attempt

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| y | G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |

1

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| x | G | C | A | G | A | G | A | G |

Shift by 1 (bmBc[a])

Second attempt

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| y | G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  |  | 2 |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | x | G | C | A | G | A | G | A | G |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Shift by 2 (bmBc[G])

Fourth attempt

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| y | G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  |  |  |  |  |  | 2 | 4 | 5 | 6 | 3 | 8 | 9 | 1 |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  | 7 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | x | G | C | A | G | A | G | A | G |  |  |  |  |  |  |  |  |  |  |  |

Shirt by 2 (bmBc[G])

Firth attempt

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| y | G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | x | G | C | A | G | A | G | A | G |  |  |  |  |  |  |  |  |  |

Sifth by 1 (bmBc[A])

Sixth attempt

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| y | G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | x | G | C | A | G | A | G | A | G |  |  |  |  |  |  |  |  |

Shift by 8 (bmBc[T])

Seventh attempt:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| y | G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 2 |  |  |  |  |  |  | 1 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | x | G | C | A | G | A | G | A | G |

Shift by 2 (bmBc[G])

Thuật toán Raita thực hiện 18 so sánh đặc tính văn bản trên ví dụ.

# THUẬT TOÁN TURBO REVERSE FACTOR

## Đặc điểm

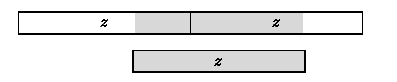
* Tìm kiếm mẫu từ bên phải sang bên trái;
* Giai đoạn tiềm xử lý trong O(m) thời gian và phức tạp không gian;
* Giai đoạn tìm kiếm trong O(n) phức tạp thời gian;
* Thực hiện 2n ký tự văn bản, kiểm tra trong các trường hợp xấu nhất;
* Tối ưu ở mức trung bình;

## Mô tả thuật toán

Nó có thể làm cho thuật toán Reverse Factor tuyến tính. Đó là, trong thực tế, đủ để nhớ các tiến tố *u* của *x* phù hợp trong làn thử cuối cùng. Sau đó, trong lần thử hiện tại khi đạt tới cuối bên phải của *u*, nó dễ dàng để cho thấy rằng nó đủ để đọc lại nhiều nhất của nửa bên phải nhất của *u*. Điều này được thực hiện bởi thuật toán Turbo Severse Factor.

Nếu một từ *z* từ một yếu tố của từ *w*, chúng ta định nghĩa disp (z,w) là sự dịch chuyển của z trong w nhỏ nhất là số nguyên d>0 sao cho [m - d - | z | -1 .. m - d] = z.

Trạng thái chung của thuật toán Turbo Severse Factor là khi một tiền tố u được tìm thấy trong các văn bản trong lần thử cuối cùng và cho các lần thử hiển tại thuật toán cố gắng để phù hợp với yếu tố v của chiều dài m - |u| trong văn bản hiện thời ở bên phải của u. Nếu v không phải là một yếu tố của x thì nó xem xét lại min{per(u), |u|/2} bên phải nhất của u. Nếu u là tuần hoàn, cho z là hậu tố của u, độ dài per(u). Theo định nghĩa của tuần hoàn z là một từ không tuần hoàn và khi đó một sự chồng chéo như hình 26.1 là không thể.



Như vậy z chỉ có thể xảy ra trong u ở khoảng cách nhiều của per (u) trong đó hàm ý rằng các hậu tố thích hợp nhỏ nhất của uv mà là một tiền tố của x có chiều dài bằng | uv | - shift (zv, x) = m -shift (zv, x). Như vậy chiều dài của sự thay đổi để thực hiện là shift (zv, x).

Nếu u không (per (u)> | u | / 2), nó là rõ ràng rằng x không thể tái xuất hiện ở phần bên trái của u có độ dài per (u). Đó là sau đó đủ để quét các phần bên phải của u dài | u | - per (u) <| u | / 2 để tìm một không xác định quá trình chuyển đổi trong automaton.

Các hàm shift được thực hiện trực tiếp tại automaton S (x) mà không thay đổi phức tạp của nó.

Giai đoạn tiền xử lý bao gồm việc xây dựng các automaton hậu tố của xR. Nó có thể được thực hiện trong O (m) phức tạp thời gian .Giai đoạn tìm kiếm là O (n) phức tạp thời gian. Các Turbo Reverse Factor thực hiện nhiều nhất là 2n kiểm tra các ký tự văn bản.

## Mã hóa thuật toán

void preMp(char \*x, int m, int mpNext[]) {

int i, j;

i = 0;

j = mpNext[0] = -1;

while (i < m) {

while (j > -1 && x[i] != x[j])

j = mpNext[j];

mpNext[++i] = ++j;

}

}

void TRF(char \*x, int m, char \*y, int n) {

int period, i, j, shift, u, periodOfU, disp, init,

state, mu, mpNext[XSIZE + 1];

char \*xR;

Graph aut;

/\* Preprocessing \*/

aut = newSuffixAutomaton(2\*(m + 2), 2\*(m + 2)\*ASIZE);

xR = reverse(x, m);

buildSuffixAutomaton(xR, m, aut);

init = getInitial(aut);

preMp(x, m, mpNext);

period = m - mpNext[m];

i = 0;

shift = m;

/\* Searching \*/

j = 0;

while (j <= n - m) {

i = m - 1;

state = init;

u = m - 1 - shift;

periodOfU = (shift != m ?

m - shift - mpNext[m - shift] : 0);

shift = m;

disp = 0;

while (i > u &&

getTarget(aut, state, y[i + j]) !=

UNDEFINED) {

disp += getShift(aut, state, y[i + j]);

state = getTarget(aut, state, y[i + j]);

if (isTerminal(aut, state))

shift = i;

--i;

}

if (i <= u)

if (disp == 0) {

OUTPUT(j);

shift = period;

}

else {

mu = (u + 1)/2;

if (periodOfU <= mu) {

u -= periodOfU;

while (i > u &&

getTarget(aut, state, y[i + j]) !=

UNDEFINED) {

disp += getShift(aut, state, y[i + j]);

state = getTarget(aut, state, y[i + j]);

if (isTerminal(aut, state))

shift = i;

--i;

}

if (i <= u)

shift = disp;

}

else {

u = u - mu - 1;

while (i > u &&

getTarget(aut, state, y[i + j]) !=

UNDEFINED) {

disp += getShift(aut, state, y[i + j]);

state = getTarget(aut, state, y[i + j]);

if (isTerminal(aut, state))

shift = i;

--i;

}

}

}

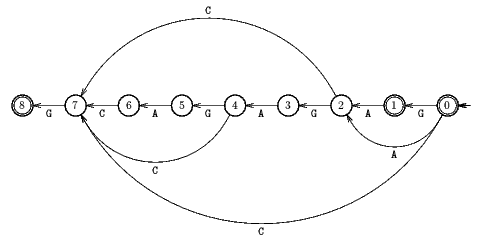
j += shift;

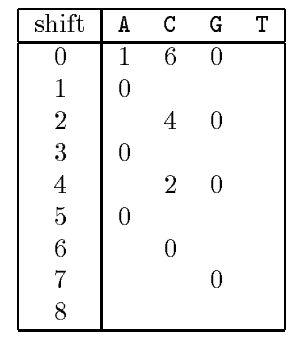
}

}

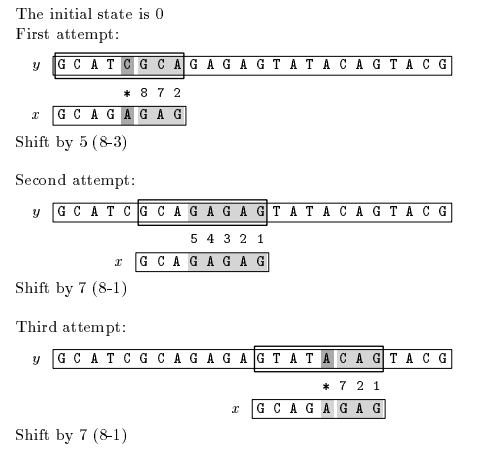
## Kiểm nghiệm thuật toán

Langage accepted





Giai đoạn tìm kiếm



Thuật toán Turbo Reverse Factor thực hiện kiểm tra 13 ký tự văn bản trong ví dụ trên.

# THUẬT TOÁN TWO WAY

## Các ðặc ðiểm chính:

* Tìm kiếm mẫu từ một vị trí cụ thể
* Yêu cầu có sẵn 1 bảng chữ cái alphabet
* Có pha tiền xử lý với độ phức tạp O(m)
* Tìm kiếm pha với thời gian O(n)
* Thực hiện so sánh 2n-m ký tự văn bản trong trường hợp xấu nhất

## Mô tả thuật toán

**Input:**

* Xâu mẫu X=(x0, x1,…,xm), độ dài m
* Văn bản Y=(y0, y1,…,yn ), độ dài n

**Output:**

* Đưa ra mọi vị trí xuất hiện của X trong Y

**Formats:** k=TW (X,m,Y,n)

**Actions:**

* Bước 1 ( tiền xử lý ):

i= maxSuf (x, m, &p);

j=maxSufTilde (x, m, &q);

if (i>j){

ell=i;

per=p;

}

Else{

ell=j;

per=q;

}

* Bước 2 (tìm kiếm):

## Mã hóa thuật toán:

/\* Computing of the maximal suffix for <= \*/

int maxSuf(char \*x, int m, int \*p) {

int ms, j, k;

char a, b;

ms = -1;

j = 0;

k = \*p = 1;

while (j + k < m) {

a = x[j + k];

b = x[ms + k];

if (a < b) {

j += k;

k = 1;

\*p = j - ms;

}

else

if (a == b)

if (k != \*p)

++k;

else {

j += \*p;

k = 1;

}

else { /\* a > b \*/

ms = j;

j = ms + 1;

k = \*p = 1;

}

}

return(ms);

}

/\* Computing of the maximal suffix for >= \*/

int maxSufTilde(char \*x, int m, int \*p) {

int ms, j, k;

char a, b;

ms = -1;

j = 0;

k = \*p = 1;

while (j + k < m) {

a = x[j + k];

b = x[ms + k];

if (a > b) {

j += k;

k = 1;

\*p = j - ms;

}

else

if (a == b)

if (k != \*p)

++k;

else {

j += \*p;

k = 1;

}

else { /\* a < b \*/

ms = j;

j = ms + 1;

k = \*p = 1;

}

}

return(ms);

}

/\* Two Way string matching algorithm. \*/

void TW(char \*x, int m, char \*y, int n) {

int i, j, ell, memory, p, per, q;

/\* Preprocessing \*/

i = maxSuf(x, m, &p);

j = maxSufTilde(x, m, &q);

if (i > j) {

ell = i;

per = p;

}

else {

ell = j;

per = q;

}

/\* Searching \*/

if (memcmp(x, x + per, ell + 1) == 0) {

j = 0;

memory = -1;

while (j <= n - m) {

i = MAX(ell, memory) + 1;

while (i < m && x[i] == y[i + j])

++i;

if (i >= m) {

i = ell;

while (i > memory && x[i] == y[i + j])

--i;

if (i <= memory)

OUTPUT(j);

j += per;

memory = m - per - 1;

}

else {

j += (i - ell);

memory = -1;

}

}

}

else {

per = MAX(ell + 1, m - ell - 1) + 1;

j = 0;

while (j <= n - m) {

i = ell + 1;

while (i < m && x[i] == y[i + j])

++i;

if (i >= m) {

i = ell;

while (i >= 0 && x[i] == y[i + j])

--i;

if (i < 0)

OUTPUT(j);

j += per;

}

else

j += (i - ell);

}

}

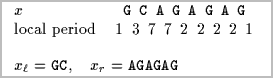
}

## Nguyên tắc cõ bản của thuật toán Two-Way với tối ða ngắn

* Hãy để phân hủy của P = uv, trong đó v là hậu tố tối đa của P và v cũng là một hậu tố tối đa ngắn.
* Sau đó chúng tôi tìm thấy nơi v xuất hiện trong T từ trái sang phải. Giả sử khi bắt đầu so sánh ở vị trí i. Khi một không phù hợp xảy ra ở v [j + 1], chúng tôi thay đổi v với j + 1 ký tự và bắt đầu so sánh tiếp theo tại P [1] với T [i + j + 1].
* Khi một phần của v đã được tìm thấy trong T, chúng tôi quét các phần của u từ phải sang trái. Nếu không phù hợp xảy ra khi quét u, chúng tôi thay đổi P với Period (P)
* Nếu chúng ta tìm thấy cả hai phần của v và u trong T, chúng tôi báo cáo một sự xuất hiện của P trong T. Sau đó chúng tôi chuyển v với Period (P)

## Kiểm nghiệm thuật toán

***Pha tiền xử lý***



***Pha tìm kiếm:***

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| First attempt | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | 1 | 2 |  | | | | | | | | | | | | | | | | | | | |
| G | C | A | G | A | G | A | G |  | | | | | | | | | | | | | | | |

Shift by: 2

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Second attempt | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | 1 |  | | | | | | | | | | | | | | | | | | |
|  | | G | C | A | G | A | G | A | G |  | | | | | | | | | | | | | |

Shift by: 1

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Third attempt | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | 1 |  | | | | | | | | | | | | | | | | | |
|  | | | G | C | A | G | A | G | A | G |  | | | | | | | | | | | | |

Shift by: 1

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Fourth attempt | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | 1 |  | | | | | | | | | | | | | | | | |
|  | | | | G | C | A | G | A | G | A | G |  | | | | | | | | | | | |

Shift by: 1

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Fifth attempt | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | 7 | 8 | 1 | 2 | 3 | 4 | 5 | 6 |  | | | | | | | | | | |
|  | | | | | G | C | A | G | A | G | A | G |  | | | | | | | | | | |

Shift by: 7

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Sixth attempt | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | 1 | 2 |  | | | | | | | |
|  | | | | | | | | | | | | G | C | A | G | A | G | A | G |  | | | |

Shift by: 2

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Seventh attempt | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | | | 1 | 2 |  | | | | | |
|  | | | | | | | | | | | | | | G | C | A | G | A | G | A | G |  | |

Shift by: 2

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Eighth attempt | | | | | | | | | | | | | | | | | | | | | | | |
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
|  | | | | | | | | | | | | | | | | | | 1 | 2 | 3 |  | | |
|  | | | | | | | | | | | | | | | | G | C | A | G | A | G | A | G |

Shift by: 3

Thuật toán Two Way thực hiện 20 so sánh ký tự trong ví dụ trên