# Tìm kiếm mẫu từ trái qua phải

## Thuật toán Brute Force

* **Đặc điểm**
  + Không có giai đoạn tiền xử lý
  + Bộ nhớ cần dùng cố định
  + Luôn luôn dịch 1 bước sang phải
  + Việc so sánh có thể phải dùng trong các trường hợp
  + So sánh khoảng 2n ký tự
* **Trình bày thuật toán**
  + Thuật toán Brute Force kiểm tra ở tất cả các vị trí trong đoạn văn bản giữa 0 và n-m, không cần quan tâm liệu mẫu này có tồn tại ở vị trí đó hay không. Sau đó, sau mỗi lần kiểm tra mẫu sẽ dịch sang phải một vị trí.
* Thuật toán Brute Force không cần giai đoạn tiền xử lý cũng như các mảng phụ cho quá trình tìm kiếm.
* **Độ phức tạp**
  + Độ phức tạp tính toán của thuật toán này là O(m.n).
* **Code**

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

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

for(int j=0 ; j<m && x[j]==y[j+i] ; j++){ // Kiểm tra tại j trong X có = i+j trong Y

if(j==m-1){

printf("FOUND AT %i \n",i);

}

}

}

}

* **Kiểm nghiệm thuật toán**
  + Xâu X=”AB”
  + Xâu Y=”ABDAAB”

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | **Y** | A | B | D | A | A | B |
| **X** | A (1) | B (2) |  |  |  |  |
| 2 | **Y** | A | B | D | A | A | B |
| **X** |  | A | B |  |  |  |
| 3 | **Y** | A | B | D | A | A | B |
| **X** |  |  | A |  |  |  |
| 4 | **Y** | A | B | D | A | A | B |
| **X** |  |  |  | A (1) | B |  |
| 5 | **Y** | A | B | D | A | A | B |
| **X** |  |  |  |  | A (1) | B (2) |

1. **Thuật toán Knuth-Morris-Pratt**

* **Đặc điểm**
  + Thực hiện từ trái qua phải
  + Pha tiền xử lý PreKMP có độ phức tạp không gian và thời gian là O(m)
* **Trình bày thuật toán**
  + Thuật toán là bản đơn giản và xử lý tương tự như thuật toán Morris-Pratt khi cố gắng dịch chuyển một đoạn dài nhất sao cho một tiền tố (prefix) *v* của x trùng với hậu tố (suffix) của u
  + Điểm khác nhau là KMP sẽ thực hiện thêm so sánh *c* và *b*, có nghĩa KMP sẽ thực hiện một pha dòm trước ký tự bên phải đoạn đang so khớp. Do đó mỗi bước KMP sẽ dịch chuyển thêm một bước sang phải so với MP nếu *c != b*
* **Độ phức tạp:**
  + Thời gian: O(n + m), trong đó n là độ dài của văn bản và m là độ dài của mẫu.
  + Không gian: O(m), với m là độ dài của mẫu.
* **Code**

PreMP(X,m,kmpNext){

i=1;

kmpNext[0]=0;

len=0;

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++;

}

}

}

KMP(X,m,Y,n){

i = 0; j = 0;

while (i < n) {

if ( X[j] == Y[i] ) { i++; j ++; }

if ( j == m ) {

< Tìm thấy mẫu ở vị trí i-j>;

j = kmpNext[j-1];

}

else if (i <n && X[j] != Y[i] ) {

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

else i = i +1;

}

}

}

* **Kiểm nghiệm thuật toán**
* Input:
  + - xâu mẫu X=”ABABCABAB” độ dài m=9
    - Xâu văn bản Y=”ABADABABCABAB” độ dài n=13
* B1: PreKMP(X,m,kmpNext[])

|  |  |  |  |
| --- | --- | --- | --- |
| **i** | **len** | **X[i]=X[len]** | **kmpNext[]** |
|  | 0 |  | 0 |
| 1 | 0 | B!=A | 0,0 |
| 2 | 0 | A=A | 0,0,1 |
| 3 | 1 | B=B | 0,0,1,2 |
| 4 | 2 | C!=A | 0,0,1,2 |
| 4 | 0 | C!=A | 0,0,1,2,0 |
| 5 | 0 | A=A | 0,0,1,2,0,1 |
| 6 | 1 | B=B | 0,0,1,2,0,1,2 |
| 7 | 2 | A=A | 0,0,1,2,0,1,2,3 |
| 8 | 3 | B=B | 0,0,1,2,0,1,2,3,4 |

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

B2:KMP(X,m,Y,n,kmpNext[])

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| STT | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| X | A | B | A | D | A | B | A | B | C | A | B | A | B | C | A | B | A | B |
| Y | A | B | A | **B** | C | A | B | A | B |  |  |  |  |  |  |  |  |  |
| I |  | J=1  I=3 | | | | | | | | | | | | | | | |  |
| j |  |  |
| X | A | B | A | **D** | A | B | A | B | C | A | B | A | B | C | A | B | A | B |
| Y |  |  | A | **B** | A | **B** | C | A | B | A | B |  |  |  |  |  |  |  |
| I |  |  | | | | | | | | | | | | | | | |  |
| j |  |  |
| X | A | B | A | **D** | A | B | A | B | C | A | B | A | B | C | A | B | A | B |
| Y |  |  |  | A | **B** | A | **B** | C | A | B | A | B |  |  |  |  |  |  |
| I |  |  | | | | | | | | | | | | | | | |  |
| j |  |  |
| X | A | B | A | D | **A** | **B** | **A** | **B** | **C** | **A** | **B** | **A** | **B** | C | A | B | A | B |
| Y |  |  |  |  | **A** | **B** | **A** | **B** | **C** | **A** | **B** | **A** | **B** |  |  |  |  |  |
| I |  |  | | | | | | | | | | | | | | | |  |
| j |  |  |
| X | A | B | A | **D** | A | B | A | B | C | A | B | A | B | C | A | B | A | B |
| Y |  |  |  |  |  |  |  |  |  | **A** | **B** | **A** | **B** | **C** | **A** | **B** | **A** | **B** |
| I |  |  | | | | | | | | | | | | | | | |  |
| j |  |  |

1. **Thuật toán Karp- Rabin**

* **Đặc điểm**
  + Biểu diễn xâu kí tự bằng số nguyên
  + Sử dụng hàm băm
* **Trình bày thuật toán**
  + Hàm băm cung cấp phương thức đơn giản để tránh những con số phức tạp trong việc so sánh những kí tự trong hầu hết các trường hợp thực tế.
  + Thay cho việc kiểm tra từng vị trí trong văn bản nếu như có mẫu xuất hiện, nó chỉ phải kiểm tra những đoạn “gần giống” xâu mẫu.
  + Để kiểm tra sự giống nhau giữa 2 từ sử dụng hàm băm.
  + Giúp cho việc đối chiếu xâu, hàm băm hash:
    - Có khả năng tính toán được
    - Đánh giá xâu mức cao.
    - Hash(y[j+1…j+m]) được tính toán dễ hơn dựa trên hash(y[j…j+m-1]) và hash(y[j+m]):
      * hash(y[j+1 .. j+m])= rehash(y[j], y[j+m], hash(y[j .. j+m-1]).

Với từ w có độ dài m có hash(w) là:

* + - * hash(w[0 .. m-1])=(w[0]\*2m-1+ w[1]\*2m-2+···+ w[m-1]\*20) mod q

Với q là một số lớn.

* + - * Sau đó rehash(a,b,h)= ((h-a\*2m-1)\*2+b) mod q
  + Pha chuẩn bị của Karp- Rabin có hàm hash(x) có thể tính toán được. nó được dùng lại không gian nhớ và có độ phức tạp O(m)
  + Trong quá trình thực thi nó so sánh hash(x) với hash([j..j+m-1]) với 0<= j<=n-m. nếu so sánh đúng, nó phải kiểm tra lại xem các kí tự trong x và y có đúng bằng nhau hay không x=y[j…j+m-1]
* **Độ phức tạp**
  + Độ phức tạp thuật toán O((n-m+1)\*m)
* **Code**

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

int hashX=0;

int hashY=0;

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

hashX+= x[i]\*(pow(prime,i));

hashY+= y[i]\*(pow(prime,i));

}

int i=0;

while(i<n){

if(hashY==hashX){

printf("FOUND AT %i\n",i);

}

if(i<n){

hashY=(hashY -y[i])/prime + y[i+m]\*prime\*prime;

}

i++;

}

}

* **Kiểm nghiệm thuật toán**
* Input:
* X=”ABC” m=3;
* Y=”EABABCACD” n=9
* Bảng định nghĩa các kí tự:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | C | D | E |
| 65 | 66 | 67 | 68 | 69 |

* + Prime=3

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Y | E | A | B | A | B | C | A | C | D |

+ Tiền xử lý:

Hash(ABC)= 65\*prime^0 + 66\*prime^1 + 67\*prime^2= 866

Hash(EAB) = 69 + 65\*prime + 66\*prime^2=858

|  |  |  |  |
| --- | --- | --- | --- |
| i | Substring | Hash(y) | == hash(ABC)? |
| 0 | **EAB**ABCACD | Hash(EAB)=858 | No |
| 1 | E**ABA**BCACD | Hash(ABA)= (858-E)/prime + A\*prime^2=848 | No |
| 2 | EA**BAB**CACD | Hash(BAB)= (858-A)/prime + B\*prime^2=855 | No |
| 3 | EAB**ABC**ACD | Hash(ABC)= (858-B)/prime + C\*prime^2=866 | YES, OUT(3) |
| 4 | EABA**BCA**CD | Hash(BCA)= (858-A)/prime + A\*prime^2=852 | NO |
| 5 | EABAB**CAC**D | Hash(CAC)= (858-B)/prime + B\*prime^2=865 | NO |
| 6 | EABABC**ACD** | Hash(ACD)= (858-C)/prime + D\*prime^2=878 | NO |

1. **Thuật toán Morris-Pratt**

* **Đặc điểm**
  + Thực hiện việc so sanh từ trái qua phải
  + Pha tiền xử lý có độ phức tạp không gian và thời gian là O(m)
  + Pha tiền xử lý có độ phức tạp thời gian là O(m+n)
  + Thực thi 2n-1 thông tin thu thập được trong quá trình quét văn bản
  + Độ trễ m (số lượng tối đa các lần so sánh ký tự đơn)
* **Trình bày thuật toán**
  + Thuật toán MP cải tiến thuật toán Brute Force, thay vì dịch chuyển từng bước một, phí công các ký tự đã so sánh trước đó, ta tìm cách dịch x đi một đoạn xa hơn.
  + Giả sử tại bước so sánh bất kỳ, ta có một pattern “*u*” trùng nhau giữa x và y, tại x[i] != y[j+i] ( *a* != *b*), thay vì dịch chuyển 1 bước sang phải, ta cố gắng dịch chuyển dài hơn sao cho một tiền tố (prefix) *v* của x trùng với hậu tố (suffix) của u.
  + Ta có mảng mpNext[] để tính trước độ dài trùng nhau lớn nhất giữa tiền tố và hậu tố trong x, khi so sánh với y tại vị trí thứ i, x sẽ trượt một khoảng = i – mpNext[i].
  + Việc tính toán mảng mpNext[] có độ phức tạp thời gian và không gian là O(n). Giai đoạn tìm kiếm sau đó có độ phức tạp thời gian là O(m+n).
* **Độ phức tạp:**
  + Pha tiền xử lý có độ phức tạp không gian và thời gian là O(m)
  + Pha tiền xử lý có độ phức tạp thời gian là O(m+n)
  + Thực thi 2n-1 thông tin thu thập được trong quá trình quét văn bản
* **Code**

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

#include <string.h>

#include <iostream>

using namespace std;

#define MAX 12

int mpNext[MAX];

void Init()

{

for(int i = 0; i < MAX; i++)

mpNext[i] = 999;

}

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

int i, j; i = 0; //mang mpNext the hien do dai trung nhau lon

j = mpNext[0] = -1; //nhat giua tien to va hau to

while (i < m) {

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

{

j = mpNext[j]; //chay nguoc xet xem do dai lon nhat cua

//vi tri giong voi x[i]

}

i++;

j++;

mpNext[i] = j;

int a = 2;

}

}

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

int i, j;// mpNext[m];

//int mpNext[8];

/\* Preprocessing \*/

Init();

preMp(x, m);

for(int k =0;k<m;k++){

cout<<x[k]<<" "<<mpNext[k]<<endl;

}

/\* Searching \*/

i = j = 0;

while (j < n) {

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

i = mpNext[i];

i++;

j++;

if (i >= m) {

cout<<j - i;

i = mpNext[i];

}

}

}

void main()

{

char \*x = "GCAGAGAG";

int m = strlen(x);

char \*y = "GCATCGCAGAGAGTATACAGTACG";

int n = strlen(y);

MP(x, m, y, n);

}

* **Kiểm nghiệm thuật toán**

**Kiểm nghiệm pha tiền xử lý( thuật toán preMp)**

x[] = GCAGAGAG

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **x[j]** | **x[i]** | **i** | **j** | **mpNext[i]** | **Ghi chú** |
|  |  | 0 | -1 |  |  |
|  | G | 0 | -1 | -1 | =>mpNext[1] =0 |
| G | C | 1 | 0  -1 | 0 |  |
| G | A | 2 | 0  -1 | 0 |  |
| G | G | 3 | 0 | 0 |  |
| C  G | A  A | 4 | 1  0  -1 | 1 |  |
| G | G | 5 | 0 | 0 |  |
| C  G | A  A | 6 | 1  0  -1 | 1 |  |
| G | G | 7 | 0 | 0 |  |
|  |  | 8 | 1 | 1 |  |

Ta được bảng mpNext[]

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| x[i] | G | C | A | G | A | G | A | G |  |
| mpNext[i] | -1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |

# IV. Tìm kiếm mấu từ vị trí bất kỳ

1. **Thuật toán Horspool Algorithm**

* **Đặc điểm**
  + Phiên bản đơn giản của Boyer-Moore
  + Chỉ sử dụng Bad-Character shift
  + Dễ thực thi
  + Có 1 bước tiền xử lý PreQS
  + Rất nhanh trong thực thi với mẫu ngắn và bảng chữ cái lớn
* **Trình bày thuật toán**
  + Tạo bảng dịch chuyển:
    - Tạo một bảng dịch chuyển cho tất cả các ký tự trongbảng chữ cái.
    - Đặt giá trị dịch chuyển cho mỗi ký tự là độ dài của mẫu.
    - Sau đó, cập nhật lại giá trị dịch chuyển cho các ký tự trong mẫu, ngoại trừ ký tự cuối cùng, thành độ dài của mẫu trừ đi vị trí của ký tự đó.
  + Tìm kiếm mẫu trong văn bản:
    - Bắt đầu từ đầu văn bản và so sánh từ ký tự cuối cùng của mẫu.
    - Nếu ký tự cuối cùng của mẫu khớp với ký tự hiện tại của văn bản, tiến hành so sánh các ký tự còn lại của mẫu từ cuối về đầu.
    - Nếu mẫu khớp hoàn toàn với văn bản, ghi nhận vị trí khớp.
    - Nếu không, sử dụng bảng dịch chuyển để di chuyển con trỏ trong văn bản.
* **Độ phức tạp**
  + Thời gian:
    - Trường hợp trung bình: O(n), trong đó n là độ dài của văn bản. Do thuật toán thường di chuyển nhiều ký tự một lúc.
    - Trường hợp xấu nhất: O(nm), trong đó n là độ dài của văn bản và m là độ dài của mẫu.
  + Không gian: O(m + σ), trong đó m là độ dài của mẫu và σ là số **lượng ký tự trong bảng chữ cái.**
* **Code**

int \* PreQS(char \*x,int m){

int \*preQS=new int[26];

for(int i=0;i<26;i++){

preQS[i]=m-1;

}

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

preQS[x[i]-65]=m-i;

}

return preQS;

}

void QuickSearch(char \*x, int m, char \*y, int n,int preQS []) {

int j=0;

while(j<=n-m){

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

{

printf("FOUND AT %i\n",j);

}

j+=preQS[y[j+m]-65];

}

}

main(){

char x []="AABA";

char y []="AABABBAABACDCDAABAABAA";

int \*preQS =PreQS(x,4);

QuickSearch(x,4,y,22,preQS);

}

* **Kiểm nghiệm thuật toán**

Tiền xử lý:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| X[i] | A | C | G | T | \* |
| preBM[i] | 2 | 7 | 1 | 9 | 9 |

j=m-1=5:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| J |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| 0 | **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** | G | C | A | **G** | A | G | A | G |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Shift by preQS[G]=1 | | | | | | | | | | | | | | | | | | | | | |  |  |  |  |
| 1 | **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** |  | **G** | C | A | G | A | G | A | G |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Shift by preQS[A]=2 | | | | | | | | | | | | | | | | | | | | | |  |  |  |  |
| 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** |  |  |  | **G** | C | A | G | A | G | A | G |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Shift by 2 | | | | | | | | | | | | | | | | | | | | | |  |  |  |  |
| 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** |  |  |  |  |  | G | C | A | G | A | G | A | G |  |  |  |  |  |  |  |  |  |  |  |
| Shift by 9 OUTPUT 5 | | | | | | | | | | | | | | | | | | | | | |  |  |  |  |
| 14 | **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** |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **G** | C | A | G | A | G | A | G |  |  |
| Shift by 7, END | | | | | | | | | | | | | | | | | | | | | |  |  |  |  |

1. **Thuật toán Smith**

* **Đặc điểm**

Smith-Waterman Algorithm là một thuật toán nổi tiếng trong sinh học tin học, dùng để tìm kiếm các chuỗi con tương tự giữa hai chuỗi. Thuật toán này dựa trên kỹ thuật lập trình động để tìm kiếm sự tương đồng tối đa của các chuỗi con bằng cách tính toán một ma trận điểm số.

* **Trình bày thuật toán**
  + Smith nhận thấy rằng có thể tính toán được rằng số bước dịch với kí tự tiếp theo kí tự đầu mút bên phải của cửa sổ dịch chuyển đôi khi cho số bước dịch ngắn hơn việc sử dụng đúng kí tự đầu mút bên phải cửa sổ ấy. Ông ấy khuyên nên chọn max giữa 2 giá trị.
  + Pha tiền xử lí của thuật toán Smith bao gồm tính toán của hàm dịch chuyển các bad- character: preBmBc(X,m) , và hàm dịch chuyển các bad- character trong Quick Search: preQsBc(X,m)
* **Độ phức tạp**
  + Thời gian: O(m\*n), với m là độ dài của chuỗi mẫu và n là độ dài của chuỗi văn bản.
  + Không gian: O(m\*n), để lưu trữ ma trận điểm số.
* **Code**

int SIZE=26;

int\* preBmBc(char \*x,int m){

int \*bmbc=new int[SIZE];

for(int i=0;i<SIZE;i++){

bmbc[i]=m;

}

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

bmbc[x[i]-65]=m-1-i;

}

return bmbc;

}

int\* preQsBc(char \*x,int m){

int \*bmbc=new int[SIZE];

for(int i=0;i<SIZE;i++){

bmbc[i]=m+1;

}

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

bmbc[x[i]-65]=m-i;

}

return bmbc;

}

int max (int x,int y){

return x>y?x:y;

}

void search(char \*x, int m,char \*y,int n,int \*preBmBc,int \*preQsBc){

int i=0;

while(i<=n-m){

if(memcmp(x,y+i,m)==0){

printf("FOUND AT: %i\n",i);

}

i+=max(preBmBc[y[i+m-1]-65],preQsBc[y[i+m]-65]);

}

}

main(){

char \*x="GCAGAGAG";

char \*y="GCATCGCAGAGAGTATACAGTACGGCAGAGAGGCAGAGAGGCAGAGA";

int \* bmbc=preBmBc(x,strlen(x));

int \* qsbc=preQsBc(x,strlen(x));

search(x,strlen(x),y,strlen(y),bmbc,qsbc);

}

* **Kiểm nghiệm thuật toán**

Input:

x=”GCAGAGAG” m=8;

y=”GCAGAGAGGCGCAGAGAGGGGG” n=22;

Tiền xử lý

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| X | A | C | G | \* |
| preBmBc[i] | 1 | 6 | 2 | 8 |
| preQsBc[i] | 2 | 7 | 1 | 9 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| J |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 |
| 0 | Y | **G** | **C** | **A** | **G** | **A** | **G** | **A** | **G** | G | C | G | C | A | G | A | G | A | G | G | G | G | G |
| X | **G** | **C** | **A** | **G** | **A** | **G** | **A** | **G** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Shift by 2 ( preBmBc[G]=2,preQsBc[G]=2), OUTPUT(0) | | | | | | | | | | | | | | | | | | | | | |  |  |
| 2 | Y | G | C | **A** | G | A | G | A | G | G | C | G | C | A | G | A | G | A | G | G | G | G | G |
|  | X |  |  | **G** | C | A | G | A | G | A | G |  |  |  |  |  |  |  |  |  |  |  |  |
| Shift by 6 ( preBmBc[C]=6,preQsBc[G]=2) | | | | | | | | | | | | | | | | | | | | | |  |  |
| 8 | Y | G | C | A | G | A | G | A | G | **G** | **C** | G | C | A | G | A | G | A | G | G | G | G | G |
|  | X |  |  |  |  |  |  |  |  | **G** | **C** | A | G | A | G | A | G |  |  |  |  |  |  |
| Shift by 2 ( preBmBc[G]=2,preQsBc[A]=1) | | | | | | | | | | | | | | | | | | | | | |  |  |
| 10 | Y | G | C | A | G | A | G | A | G | G | C | **G** | **C** | **A** | **G** | **A** | **G** | **A** | **G** | G | G | G | G |
|  | X |  |  |  |  |  |  |  |  |  |  | **G** | **C** | **A** | **G** | **A** | **G** | **A** | **G** |  |  |  |  |
| Shift by 2 ( preBmBc[G]=2,preQsBc[G]=2) , OUTPUT(10) | | | | | | | | | | | | | | | | | | | | | |  |  |
| 12 | Y | G | C | A | G | A | G | A | G | G | C | G | C | **A** | G | A | G | A | G | G | G | G | G |
|  | X |  |  |  |  |  |  |  |  |  |  |  |  | **G** | C | A | G | A | G | A | G |  |  |
| Shift by 2 ( preBmBc[G]=2,preQsBc[G]=2) | | | | | | | | | | | | | | | | | | | | | |  |  |
| 14 | Y | G | C | A | G | A | G | A | G | G | C | G | C | A | G | **A** | G | A | G | G | G | G | G |
|  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **G** | C | A | G | A | G | A | G |
| END | | | | | | | | | | | | | | | | | | | | | |  |  |

1. **Thuật toán Raita**

* **Đặc điểm:**
  + Đầu tiên so sánh kí tự cuối cùng của xâu mẫu, sau đó là kí tự đầu tiên và kí tự giữa trước khi thực thi lệnh so sánh khác.
  + Pha cài đặt có độ phức tạp thuật toán là O(m+σ) và độ phức tạp bộ nhớ là O(σ)
  + Pha thực thi có độ phức tạp là O(mxn)
* **Trình bày thuật toán**

Raita thiết kế một thuật toán mà với mỗi lần duyệt, đầu tiên đem so sánh kí tự cuối cùng của mẫu với kí tự đầu mút bên phải của cửa sổ, sau đó nếu chúng trùng nhau, so sánh kí tự đầu tiên của mẫu với kí tự đầu mút bên trái của cửa sổ. Nếu chúng trùng nhau, tiếp tục so sánh kí tự giữa của mẫu với kí tự giữa của xâu văn bản đang trong cửa sổ duyệt. Nếu chúng vẫn trùng nhau, sẽ đem so sánh các kí tự khác trong xâu mẫu bắt đầu vị trí thứ 2 tới vị trí cuối cùng(so sánh lại kí tự ở giữa một lần nữa).

* **Độ phức tạp**
  + Pha cài đặt có độ phức tạp thuật toán là O(m+σ) và độ phức tạp bộ nhớ là O(σ)
  + Pha thực thi có độ phức tạp là O(mxn)
* **Code**

int SIZE=26;

int \* preBmBc(char \*x,int m){

int \* bmbc=new int[SIZE];

for(int i=0;i<SIZE;i++){

bmbc[i]=m;

}

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

bmbc[x[i]-65]=m-1-i;

}

return bmbc;

}

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

char first=x[0],last=x[m-1],middle=x[m/2];

int i=0;

while(i<=n-m){

if(last==y[i+m-1] && first==y[i] && middle==y[i+m/2]){

if(memcmp(x+1,y+i+1,m-2)==0){

printf("FOUND AT: %i \n",i);

}

}

i+=bmbc[y[i+m-1]-65];

}

}

main(){

char \*x="GCAGAGAG";

char \*y="GCAGAGAGGCGCAGAGAGGGGG";

int \* bmbc=preBmBc(x,strlen(x));

raitaSearch(x,strlen(x),y,strlen(y),bmbc);

}

* **Kiểm nghiệm thuật toán**

Input:

x=”GCAGAGAG” m=8;

y=”GCAGAGAGGCGCAGAGAGGGGG” n=22;

Tiền xử lý

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| X | A | C | G | \* |
| preBmBc[i] | 1 | 6 | 2 | 8 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| J |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 |
| 0 | Y | **G** | **C** | **A** | **G** | **A** | **G** | **A** | **G** | G | C | G | C | A | G | A | G | A | G | G | G | G | G |
| X | **G** | **C** | **A** | **G** | **A** | **G** | **A** | **G** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Shift by 2 ( preBmBc[G]=2) OUTPUT(0) | | | | | | | | | | | | | | | | | | | | | |  |  |
| 2 | Y | G | C | A | G | A | G | A | G | G | **C** | G | C | A | G | A | G | A | G | G | G | G | G |
|  | X |  |  | G | C | A | G | A | G | A | **G** |  |  |  |  |  |  |  |  |  |  |  |  |
| Shift by 6 ( preBmBc[C]=6) | | | | | | | | | | | | | | | | | | | | | |  |  |
| 8 | Y | G | C | A | G | A | G | A | G | **G** | C | G | C | A | G | A | **G** | A | G | G | G | G | G |
|  | X |  |  |  |  |  |  |  |  | **G** | C | A | G | A | G | A | **G** |  |  |  |  |  |  |
| Shift by 2( preBmBc[G]=2) | | | | | | | | | | | | | | | | | | | | | |  |  |
| 10 | Y | G | C | A | G | A | G | A | G | G | C | **G** | ***C*** | ***A*** | ***G*** | ***A*** | ***G*** | ***A*** | **G** | G | G | G | G |
|  | X |  |  |  |  |  |  |  |  |  |  | **G** | ***C*** | ***A*** | ***G*** | ***A*** | ***G*** | ***A*** | **G** |  |  |  |  |
| Shift by 2 ( preBmBc[G]=2) , OUTPUT(10) | | | | | | | | | | | | | | | | | | | | | |  |  |
| 12 | Y | G | C | A | G | A | G | A | G | G | C | G | C | A | G | A | G | A | G | G | **G** | G | G |
|  | X |  |  |  |  |  |  |  |  |  |  |  |  | G | C | A | G | A | G | A | **G** |  |  |
| Shift by 2 ( preBmBc[G]=2) | | | | | | | | | | | | | | | | | | | | | |  |  |
| 14 | Y | G | C | A | G | A | G | A | G | G | C | G | C | A | G | A | G | A | G | G | G | G | **G** |
|  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  | G | C | A | G | A | G | A | **G** |
| END | | | | | | | | | | | | | | | | | | | | | |  |  |