1. **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 |

1. **Tìm kiếm mẫu từ phải qua trái**
2. **Thuật toán Boyer-Moore(91)**

* **Đặc điểm**
* Thực hiện so sánh từ phải sang trái
  + Có 1 bước tiền xử lý preBM để xác định khoảng cách từ 1 kí tự trong xâu mẫu đến kí tự cuối cùng
  + Độ phức tạp thuật toán: O(m)
* **Trình bày thuật toán**
* Thuật toán Boyer-Moore được coi là thuật toán hiệu quả nhất trong vấn đề tìm kiếm chuỗi trong các ứng dụng thường gặp. Các biến thể của nó được dùng trong các bộ soạn thảo cho các lệnh như <<search> và <<subtitle>>.
* Thuật toán sẽ quét các ký tự của mẫu(pattern) từ phải sang trái bắt đầu ở phần tử cuối cùng.
* **Code**

#include <stdio.h>

#include <string.h>

int ASize=26;

void PreBM(char \*x,int m,int preBM[]){

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

{

preBM[i]=m;

}

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

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

}

}

void BMSearching(char \*x,int m,char \*y,int n,int preBM[]){

int j=m-1;

while(j<n){

bool check=false;

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

if(x[i]!=y[j-(m-i-1)]){

check=true;

break;

}

}

if(!check){

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

}

j+=preBM[y[j]-65];

}

}

main(){

int \*preBM =new int[ASize];

char x[]="ABCDAB";

PreBM(x,6,preBM);

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

printf("%i ",preBM[i]);

}

char y[]="AABCDABBDEFAABCDABCDAB";

printf("\n\nSTART SEARCHING\n");

BMSearching(x,6,y,22,preBM);

}

* **Kiểm nghiệm thuật toán**

Input:

X=” ABCDAB”, m=6

Y=” AABCDABBDEFAABCDABCDAB”, n=22

Tiền xử lý

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| X[i] | A | B | C | D | \* |
| preBM[i] | 1 | 4 | 3 | 2 | 6 |

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 |
| 5 | **Y** | A | A | B | C | D | **A** | B | B | D | E | F | A | A | B | C | D | A | B | C | D | A | B |
| **X** | A | B | C | D | A | **B** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Shift by 4 | | | | | | | | | | | | | | | | | | | | | |  |  |
| 1 | **Y** | A | A | B | C | D | A | ***B*** | B | D | E | F | A | A | B | C | D | A | B | C | D | A | B |
|  | **X** |  | A | B | C | D | A | ***B*** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Shift by 4, OUTPUT 1 | | | | | | | | | | | | | | | | | | | | | |  |  |
| 3 | **Y** | A | A | B | C | D | A | B | B | D | E | F | A | A | B | C | **D** | A | B | C | D | A | B |
|  | **X** |  |  |  |  |  |  |  |  |  |  | A | B | C | D | A | **B** |  |  |  |  |  |  |
| Shift by 2 | | | | | | | | | | | | | | | | | | | | | |  |  |
| 4 | **Y** | A | A | B | C | D | A | B | B | D | E | F | A | A | B | C | D | A | ***B*** | C | D | A | B |
|  | **X** |  |  |  |  |  |  |  |  |  |  |  |  | A | B | C | D | A | ***B*** |  |  |  |  |
| Shift by 4 OUTPUT 12 | | | | | | | | | | | | | | | | | | | | | |  |  |
| 6 | **Y** | A | A | B | C | D | A | B | B | D | E | F | A | A | B | C | D | A | B | C | D | A | ***B*** |
|  | **X** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | A | B | C | D | A | ***B*** |
| Shift by 4, OUTPUT 16, END | | | | | | | | | | | | | | | | | | | | | |  |  |

OUTPUT: 1,12,16

1. **Thuật toán Berry-Ravindran(133)**

* **Đặc điểm**
* Đây là thuật toán kết hợp giữa thuật toán Quic search và Zhu and Takaoka.
* Pha chuẩn bị có độ phức tạp O(m +∂2 )
* Pha tìm kiếm có độ phức tạp O(mn)
* **Trình bày thuật toán**
* Berry và Ravindran đã thiết kế ra thuật toán thực hiện bước dịch dựa vào tư tưởng bad-character. Nhưng ở đây lấy 2 kí tự liên tiếp ngoài cùng bên phải của sổ để xác định bước dịch.
* Quá trình chuẩn bị của thuật toán bao gồm việc xác định với mỗi cặp (a,b) vị trị ngoài cũng bên phải gần nhất bắt đầu xuất hiện ab trong x. Sau mỗi lần thử mẫu khi cửa sổ đang ở vị trí tương ứng y[j .. j+m-1] bước dịch tiếp theo sẽ là brBc[y[j+m],y[j+m+1]]
* **Code**

#include<iostream>

#include<algorithm>

#include<iomanip>

#include<cstdio>

#include<cstring>

#define For(i,a,b) for(long i = a;i<=b;i++)

using namespace std;

char x[100001],y[100001];

int m, n,ASIZE = 256;

void nhap(){

printf("Nhap x: ");

gets(x);

m = strlen(x);

printf("Nhap y: ");

gets(y);

n = strlen(y);

}

void preBrBc(char \*x, int m, int brBc[256][256]) {

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[256][256];

/\* Preprocessing \*/

preBrBc(x, m, brBc);

/\* Searching \*/

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

j = 0;

while (j <= n - m) {

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

printf("position is %d\n",j);

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

}

}

main(){

nhap();

BR(x,m,y,n);

}

* **Kiểm nghiệm thuật toán**

Input:

X = “GCAGAGAG”

Y =”GCATCGCAGAGAGTATACAGTACG”

Tiền xử lý:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| brBc | A | C | G | T | \* |
| A | 10 | 10 | 2 | 10 | 10 |
| C | 7 | 10 | 9 | 10 | 10 |
| G | 1 | 1 | 1 | 1 | 1 |
| T | 10 | 10 | 9 | 10 | 10 |
| \* | 10 | 10 | 9 | 10 | 10 |

Pha tìm kiếm:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 brBC[G][A] =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 brBC[A][G] =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 brBC[A][G] =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 brBC[T][A] =10, OUTPUT(5) | | | | | | | | | | | | | | | | | | | | | |  |  |  |  |
| 15 | 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 brBC[G][0] =1 | | | | | | | | | | | | | | | | | | | | | |  |  |  |  |
| 16 | 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 |
| END | | | | | | | | | | | | | | | | | | | | | |  |  |  |  |

1. **Thuật toán Turbo BM**(99)

* **Đặc điểm**
  + Đây là thuật toán đơn giản hóa từ thuật toán Boyer-moore.
  + Dễ cài đặt
* **Trình bày thuật toán**
* Tuned Boyer-moore là cài đặt đơn giản của thuật toán Boyer-Moore. Chi phí cho thuật toán string-matching thường phần nhiều là việc kiểm tra
* Để tránh việc phải so sánh nhiều lần. Chúng ta có thể thực hiện nhiều bước dịch hơn trước khi thực sự so sánh xâu. Thuật toán này sẽ sử dụng hàm bad-character xác định bước dịch. Và tìm x[m-1] trong y cho tới khi nào tìm được. Yêu cầu lưu giá trị bmBc[x[m-1]] vào biến shift và đặt lại giá trị bmBc[x[m-1]] = 0 . Khi ta tìm được vị trí x[m-1] trong y, thì bước dịch tiếp theo sẽ là shift.
* **Code**

#include<iostream>

#include<algorithm>

#include<iomanip>

#include<cstdio>

#include<cstring>

#include<map>

#include<set>

#include<stack>

#include<queue>

#include<vector>

#define For(i,a,b) for(long i = a;i<=b;i++)

using namespace std;

char x[100001],y[100001];

int m,n, ASIZE = 256;

string s = "";

void nhap(){

printf("Nhap x: ");

gets(x); m = strlen(x);

printf("Nhap y: ");

gets(y); n = strlen(y);

}

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 - m) {

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 - m)

printf("position is %d\n",j);

j += shift;

/\* shift \*/

}

}

main(){

nhap();

TUNEDBM(x,m,y,n);

}

* **Kiểm nghiệm thuật toán**

Input:

X = “GCAGAGAG”

Y =”GCATCGCAGAGAGTATACAGTACG”

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| X[i] | A | C | G | T | \* |
| preBmBc | 1 | 6 | 0 | 8 | 8 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| X |  | G | C | A | G | A | G | A | G |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| X |  | G | C | A | G | A | G | A | G |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| SHIFT BY 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 |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 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 |  |  |  |  |  |  |  |  |  |  |  |
|  | X |  |  |  |  |  | G | C | A | G | A | G | A | G |  |  |  |  |  |  |  |  |  |  |  |
| SHIFT BY 2, OUT PPUT 5 | | | | | | | | | | | | | | | | | | | | | |  |  |  |  |
| 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 |  |  |  |  |  |  |  | G | C | A | G | A | G | A | G |  |  |  |  |  |  |  |  |  |
|  | X |  |  |  |  |  |  |  |  | G | C | A | G | A | G | A | G |  |  |  |  |  |  |  |  |
|  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | G | C | A | G | A | G | A | G |
|  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | G | C | A | G | A | G | A | G |
| END | | | | | | | | | | | | | | | | | | | | | |  |  |  |  |

1. **Thuật toán Colussi**(61)

* **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
* **Trình bày 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.
* **Code**

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];

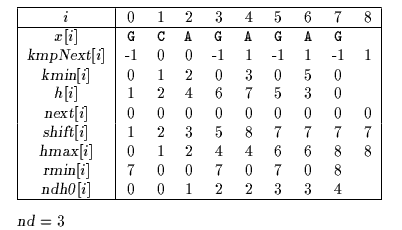
i = next[i];

}

}

* **Kiểm nghiệm thuật toán**

Tiền xử lý

****

1. **Tìm kiếm mẫu từ vị trí cụ thể**
2. **Thuật toán KMP Skip Search**(199)

* **Đặc điểm**
  + Sử dụng thùng chứa (bucket) các vị trí xuất hiện của kí tự trong xâu mẫu.
  + Pha tìm kiếm có độ phức tạp thời gian O(mn).
* **Trình bày thuật toán**
  + Với mỗi kí tự trong bảng chữ cái, một thùng chứa (bucket) sẽ chứa tất cả các vị trí xuất hiện của kí tự đó trong xâu mẫu x. khi một kí tự xuất hiện k lần trong mẫu. bucket sẽ lưu k vị trí của kí tự đó. Khi mà xâu y chứa it kí tự hơn trong bản chữ cái thì sẽ có nhiều bucket rỗng.
  + Quá trình xử lý của thuật toán Skip Search bao gồm việc tính các buckets cho tất cả các kí tự trong bảng chữ cái. for c in z[c] = {i: 0 i m-1 and x[i] = c}
  + Thuật toán Skip Search có độ phức tạp bình phương trong trường hợp tồi nhất. nhưng cũng có trường hợp là O(n).
* **Code**

|  |
| --- |
| #include<iostream>  #include<algorithm>  #include<iomanip>  #include<cstdio>  #include<cstring>  #include<list>  using namespace std;  char x[100001],y[100001];  int ASIZE = 256, m, n;  void nhap(){  printf("Nhap x: ");  gets(x);  m = strlen(x);  printf("Nhap y: ");  gets(y);  n = strlen(y);  }  void SKIP(char \*x, int m, char \*y, int n){  int i,j;  list<int> z[ASIZE];  list<int>::iterator it;  /\*preprocessing \*/  for(i = m-1 ; i >=0 ;i--){  z[x[i]].push\_back(i); }  /\* search \*/  for(j = m-1; j < n ; j +=m){  for(it = z[y[j]].begin() ;  it!= z[y[j]].end();  it++)  {  if(memcmp(x,y+j-(\*it),m) == 0)  printf("position is %d\n",j- (\*it));  }  }  }  main(){  nhap();  SKIP(x,m,y,n);  } |

* **Kiểm nghiệm thuật toán**

Input:

**X = “GCAGAGAG”**

**Y = ”GCATCGCAGAGAGTATACAGTACG”**

Pha tiền xử lý xác định:

|  |  |
| --- | --- |
| C | Z[c] |
| A | 6,4,2 |
| C | 1 |
| G | 7,5,3,0 |
| T |  |

Pha tìm kiếm

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 |
| 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 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 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 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 15 | 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 |  |  |  |  |  |  |  |  |  |  |  |
| 16 | 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 |  |  |  |
| 16 | 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** |

1. **Thuật toán Two Way**(171)

* **Trình bày thuật toán**
* Thuật toán:
* Thuật toán Two Way là một phương pháp tìm kiếm chuỗi con P trong chuỗi lớn T, đồng thời từ hai hướng trái và phải.
* Nguyên lý xử lý:
* Tìm kiếm từ hai hướng: Thực hiện tìm kiếm chuỗi con P trong chuỗi T đồng thời từ hai hướng trái và phải.
* So sánh song song: So sánh các kí tự của chuỗi con P với các kí tự trong chuỗi T từ hai phía, giúp tìm ra tất cả các vị trí xuất hiện của chuỗi con P.
* Tìm kiếm:
* Bắt đầu tìm kiếm từ hai đầu của chuỗi T đồng thời.
* Tại mỗi bước, so sánh kí tự tương ứng của chuỗi con P và chuỗi T từ hai hướng.
* Nếu các kí tự khớp nhau, tiếp tục so sánh cho đến khi gặp kí tự không khớp.
* Tiếp tục tìm kiếm từ hai hướng cho đến khi đạt được tất cả các vị trí xuất hiện của chuỗi con P trong chuỗi T.
* **Đánh giá độ phức tạp**
* Tìm kiếm từ hai hướng: Độ phức tạp của tìm kiếm là O(n+m), với n là độ dài của chuỗi T và m là độ dài của chuỗi con P, do cả hai đầu của chuỗi T đều được duyệt một lần.
* Tổng thể: Tổng thể, độ phức tạp của thuật toán Two Way là O(n+m), làm cho nó trở thành một phương pháp hiệu quả cho việc tìm kiếm chuỗi con P trong chuỗi T từ cả hai phía.
* **Kiểm nghiệm thuật toán**

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

* **Code:**

#include <iostream>

#include <string>

#include <vector>

#include <algorithm>

using namespace std;

// Hàm tính toán suffix tối đa

int maxSuf(const vector<char>& x, int m, vector<int>& p) {

int ms = -1, j = 0, k = p[0] = 1;

while (j + k < m) {

char a = x[j + k], b = x[ms + k];

if (a < b) {

j += k;

k = 1;

p[0] = j - ms;

} else if (a == b) {

if (k != p[0]) k++;

else {

j += p[0];

k = 1;

}

} else {

ms = j;

j = ms + 1;

k = p[0] = 1;

}

}

return ms;

}

// Hàm tính toán suffix tối đa ngược lại

int maxSufTilde(const vector<char>& x, int m, vector<int>& p) {

int ms = -1, j = 0, k = p[0] = 1;

while (j + k < m) {

char a = x[j + k], b = x[ms + k];

if (a > b) {

j += k;

k = 1;

p[0] = j - ms;

} else if (a == b) {

if (k != p[0]) k++;

else {

j += p[0];

k = 1;

}

} else {

ms = j;

j = ms + 1;

k = p[0] = 1;

}

}

return ms;

}

// Hàm tìm kiếm Two-way

void search(const vector<char>& x, int m, const vector<char>& y, int n) {

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

vector<int> mp(1);

// Tiền xử lý

ms = maxSuf(x, m, mp);

p = mp[0];

ell = max(ms, maxSufTilde(x, m, mp));

// Giai đoạn tìm kiếm

if (ms > maxSufTilde(x, m, mp)) {

per = p;

} else {

per = mp[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) {

cout << "Mẫu xuất hiện tại vị trí dịch = " << j << endl;

j += per;

memory = m - per - 1;

} else {

j += i - ell;

memory = -1;

}

} else {

j += i - ell;

memory = -1;

}

}

}

int main() {

string text = "ABABDABACDABABCABAB";

string pattern = "ABABCABAB";

vector<char> y(text.begin(), text.end());

vector<char> x(pattern.begin(), pattern.end());

int n = y.size();

int m = x.size();

search(x, m, y, n);

return 0;

}

1. **Thuật toán Sunday’s Optimal Mismath**(185)

* **Trình bày thuật toán**
* Thuật toán: Sunday’s Optimal Mismatch là một phương pháp tìm kiếm chuỗi con P trong chuỗi lớn T, với mục tiêu là tối ưu hóa số lần so sánh thông qua việc xác định sự không khớp tối ưu của kí tự trong chuỗi T.
* Nguyên lý xử lý:
* Tìm kiếm với sự không khớp tối ưu: Xác định sự không khớp tối ưu của kí tự trong chuỗi T để quyết định bước nhảy khi tìm kiếm chuỗi con P.
* Sử dụng bảng dịch chuỗi con: Xây dựng bảng dịch để quyết định sự di chuyển của chuỗi con P khi tìm kiếm.
* Tìm kiếm:
* Bắt đầu tìm kiếm từ đầu chuỗi T.
* So sánh các kí tự của chuỗi con P với các kí tự trong chuỗi T.
* Nếu có sự không khớp xảy ra ở vị trí i trong chuỗi T:
* Nếu kí tự không khớp T[i+m] không có trong chuỗi con P, thì di chuyển chuỗi con P về phía trước m bước.
* Nếu kí tự không khớp T[i+m] có trong chuỗi con P, thì di chuyển chuỗi con P để khớp với kí tự không khớp trong chuỗi T.
* Lặp lại quá trình này cho đến khi tìm thấy tất cả các vị trí xuất hiện của chuỗi con P trong chuỗi T.
* **Đánh giá độ phức tạp**
* Tìm kiếm với sự không khớp tối ưu: Độ phức tạp của tìm kiếm với sự không khớp tối ưu là O(n), với n là độ dài của chuỗi T.
* Tổng thể: Tổng thể, độ phức tạp của thuật toán Sunday’s Optimal Mismatch là O(n) trong trường hợp tốt nhất, khi mọi kí tự trong chuỗi T đều là các kí tự không khớp, và O(nm) trong trường hợp xấu nhất, khi tìm kiếm mỗi lần di chuyển m bước. Đây là một phương pháp tìm kiếm hiệu quả đặc biệt khi chuỗi con P thường xuyên xuất hiện không kỳ vọng trong chuỗi T.
* **Kiểm nghiệm thuật toán**
* Pha tiền xử lý xác định

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| c | A | C | G | T |
| Freq[c] | 8 | 5 | 7 | 4 |
| QsBc[c] | 2 | 7 | 1 | 9 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| x[i] | G | C | A | G | A | G | A | G |
| Pat[i].loc | 1 | 7 | 5 | 3 | 0 | 6 | 4 | 2 |
| Pat[i].c | C | G | G | G | G | A | A | A |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| AdaptedGs[i] | 1 | 3 | 4 | 2 | 7 | 7 | 7 | 7 | 7 |

* Pha tìm kiếm:

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

* **Code:**

#include <iostream>

#include <string>

#include <vector>

#include <algorithm>

#include <cstring>

using namespace std;

const int ASIZE = 256; // Kích thước bảng ASCII

struct Pattern {

char c;

int loc;

};

// Hàm sắp xếp và xây dựng bảng mẫu từ chuỗi

void orderPattern(const vector<char>& x, int m, vector<Pattern>& pat) {

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

pat[i].loc = i;

pat[i].c = x[i];

}

sort(pat.begin(), pat.end(), [](const Pattern& p1, const Pattern& p2) {

return p1.c < p2.c;

});

}

// Hàm tìm kiếm dịch chuyển tiếp theo

int matchShift(const vector<char>& x, int m, int ploc, int shift, const vector<Pattern>& pat) {

int j = shift;

while (j <= m) {

if (j == m) return m;

int i = ploc;

while (i < m && (j < m && pat[i].loc - shift >= 0) && pat[i].c == x[pat[i].loc - shift]) {

i++;

}

if (i == m) return j;

j++;

}

return m;

}

// Hàm xây dựng bảng good-suffix

void preAdaptedGs(const vector<char>& x, int m, vector<int>& adaptedGs, const vector<Pattern>& pat) {

int shift = matchShift(x, m, 0, 1, pat);

adaptedGs[0] = shift;

for (int ploc = 1; ploc <= m; ploc++) {

shift = matchShift(x, m, ploc, shift, pat);

adaptedGs[ploc] = shift;

}

}

// Hàm tiền xử lý bảng bad character

void preBadChar(const vector<char>& x, int m, vector<int>& badChar) {

fill(badChar.begin(), badChar.end(), m);

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

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

}

}

// Hàm tìm kiếm Sunday's Optimal Mismatch

void search(const vector<char>& x, int m, const vector<char>& y, int n) {

vector<int> adaptedGs(m + 1);

vector<int> badChar(ASIZE);

vector<Pattern> pat(m);

// Tiền xử lý

orderPattern(x, m, pat);

preBadChar(x, m, badChar);

preAdaptedGs(x, m, adaptedGs, pat);

// Giai đoạn tìm kiếm

int j = 0;

while (j <= n - m) {

int i = 0;

while (i < m && pat[i].c == y[j + pat[i].loc]) {

i++;

}

if (i >= m) {

cout << "Mẫu xuất hiện tại vị trí dịch = " << j << endl;

}

if (j + m < n) {

j += max(adaptedGs[i], badChar[y[j + m]]);

} else {

j++;

}

}

}

int main() {

string text = "ABABDABACDABABCABAB";

string pattern = "ABABCABAB";

vector<char> y(text.begin(), text.end());

vector<char> x(pattern.begin(), pattern.end());

int n = y.size();

int m = x.size();

search(x, m, y, n);

return 0;

}

1. **Tìm kiếm mẫu từ vị trí bất kì**
2. **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 | | | | | | | | | | | | | | | | | | | | | |  |  |