MỤC LỤC

[1. Tổng quan: 3](#_Toc486690274)

[1.1. Các thuật toán tìm kiếm mẫu từ trái sang phải: 3](#_Toc486690275)

[1.2. Các thuật toán tìm kiếm mẫu từ phải sang trái: 3](#_Toc486690276)

[1.3. Các thuật toán tìm kiếm mẫu từ vị trí xác định: 3](#_Toc486690277)

[1.4. Các thuật toán tìm kiếm mẫu từ vị trí bất kỳ: 3](#_Toc486690278)

[2. Tìm kiếm mẫu từ trái sang phải: 3](#_Toc486690279)

[2.1. Thuật toán Knuth-Morris-Partt: 3](#_Toc486690280)

[2.2. Thuật toán Karp-Rabin: 7](#_Toc486690281)

[2.3. Thuật toán Shift or: 9](#_Toc486690282)

[2.4. Thuật toán Morris-Pratt: 11](#_Toc486690283)

[2.5. Thuật toán Not So Naive: 15](#_Toc486690284)

[2.6. Thuật toán Apostolico-Crochemore: 16](#_Toc486690285)

[3. Tìm kiếm mẫu từ phải sang trái: 19](#_Toc486690286)

[3.1. Thuật toán Boyer-Moore: 19](#_Toc486690287)

[3.2. Thuật toán Turbo-BM: 22](#_Toc486690288)

[3.3. Thuật toán Zhu-Takaoka: 24](#_Toc486690289)

[3.4. Thuật toán Berry-Ravindran: 27](#_Toc486690290)

[3.5. Thuật toán: 28](#_Toc486690291)

[4. Tìm kiếm mẫu từ vị trí xác định: 28](#_Toc486690292)

[4.1. Thuật toán Two way: 28](#_Toc486690293)

[4.2. Thuật toán Colussi: 31](#_Toc486690294)

[4.3. Thuật toán Skip Search: 34](#_Toc486690295)

[4.4. Thuật toán KMP Skip Search: 36](#_Toc486690296)

[4.5. Thuật toán Alpha Skip Search: 38](#_Toc486690297)

[4.6. Thuật toán Optimal Mismatch: 41](#_Toc486690298)

[4.7. Thuật toán Maximal Shift: 44](#_Toc486690299)

[5. Tìm kiếm mẫu từ vị trí bất kỳ: 46](#_Toc486690300)

[5.1. Thuật toán Horspool: 46](#_Toc486690301)

[5.2. Thuật toán Quick Search: 48](#_Toc486690302)

[5.3. Thuật toán Smith: 49](#_Toc486690303)

[5.4. Thuật toán Raita: 50](#_Toc486690304)

[5.5. Thuật toán Tuned Boyer-Moore: 51](#_Toc486690305)

# Tổng quan:

## Các thuật toán tìm kiếm mẫu từ trái sang phải:

* Knuth-Morris-Pratt
* Karp-Rabin
* Shift-Or
* Morris-Pratt
* Not-so-naive
* Apostolico-Crochemore

## Các thuật toán tìm kiếm mẫu từ phải sang trái:

* Boyer - Moore Algorithm
* Turbo BM Algorithm
* Colussi Algorithm
* Sunday Algorithm
* Reverse Factorand Algorithm
* Turbo Reverse Factor
* Zhu and Takaoka and Berry - Ravindran Algorithms

## Các thuật toán tìm kiếm mẫu từ vị trí xác định:

* Two Way Algorithm
* Colussi Algorithm
* Galil-Giancarlo Algorithm
* Sunday's Optimal Mismatch  Algorithm
* Maximal Shift Algorithm
* Skip Search
* KMP Skip Search and Alpha Skip Search Algorithms.

## Các thuật toán tìm kiếm mẫu từ vị trí bất kỳ:

* Horspool Algorithm
* Tuned Boyer-Moore Algorithm
* Smith Algorithm
* Raita Algorithm.
* Quick search

# Tìm kiếm mẫu từ trái sang phải:

## Thuật toán Knuth-Morris-Partt:

### Đặc điểm:

* Thực hiện từ trái sang phải
* Pha tiền xử lí độ phức tạp O(m)
* Độ phức tạp thuật toán là O(n + m)
* Số lần so sánh tối đa cho 1 kí tự của xâu là với

### Thuật toán:

* Input:
  + Mẫu X = (x0, x1,…,xm-1), độ dài m
  + Văn bản Y = (y0, y1,…,yn-1), độ dài n
* Output: Tất cả các vị trí xuất hiện của X trong Y

|  |
| --- |
| 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:

* X[] = “GCAGAGAG”, m = 8
* Y[] = “GCATCGCAGAGAGTATACAGTACG”, n = 24

Bước 1: Tiền xử lý: preKmp(X, 8, kmpNext)

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

Bước 2: Tìm kiếm: KMP(X, 8, Y, 24)

* **j = 0, i = 0:**

|  |  |  |
| --- | --- | --- |
| i | X[i] != Y[j]? | mpNext[i] |
| 0 | NO |  |

i = i + 1 = 1; j = j + 1 = 1

* **j = 1, i = 1:**

|  |  |  |
| --- | --- | --- |
| i | X[i] != Y[j]? | mpNext[i] |
| 1 | NO |  |

i = i + 1 = 2; j = j + 1 = 2

* **j = 2, i = 2:**

|  |  |  |
| --- | --- | --- |
| i | X[i] != Y[j]? | mpNext[i] |
| 2 | NO |  |

i = i + 1 = 3; j = j + 1 = 3

* **j = 3, i = 3:**

|  |  |  |
| --- | --- | --- |
| i | X[i] != Y[j]? | mpNext[i] |
| 3 | YES | -1 |
| -1 |  |  |

i = i + 1 = 0; j = j + 1 = 4

* **j = 4, i = 0:**

|  |  |  |
| --- | --- | --- |
| i | X[i] != Y[j]? | mpNext[i] |
| 0 | YES | -1 |
| -1 |  |  |

i = i + 1 = 0; j = j + 1 = 5

* **j = 5, i = 0:**

|  |  |  |
| --- | --- | --- |
| i | X[i] != Y[j]? | mpNext[i] |
| 0 | NO |  |

i = i + 1 = 1; j = j + 1 = 1

* **j = 6, i = 1:**

|  |  |  |
| --- | --- | --- |
| i | X[i] != Y[j]? | mpNext[i] |
| 1 | NO |  |

i = i + 1 = 2; j = j + 1 = 7

* **j = 7, i = 2:**

|  |  |  |
| --- | --- | --- |
| i | X[i] != Y[j]? | mpNext[i] |
| 2 | NO |  |

i = i + 1 = 3; j = j + 1 = 8

* **j = 8, i = 3:**

|  |  |  |
| --- | --- | --- |
| i | X[i] != Y[j]? | mpNext[i] |
| 3 | NO |  |

i = i + 1 = 4; j = j + 1 = 9

* **j = 9, i = 4:**

|  |  |  |
| --- | --- | --- |
| i | X[i] != Y[j]? | mpNext[i] |
| 4 | NO |  |

i = i + 1 = 5; j = j + 1 = 10

* **j = 10, i = 5:**

|  |  |  |
| --- | --- | --- |
| i | X[i] != Y[j]? | mpNext[i] |
| 5 | NO |  |

i = i + 1 = 6; j = j + 1 = 11

* **j = 11, i = 6:**

|  |  |  |
| --- | --- | --- |
| i | X[i] != Y[j]? | mpNext[i] |
| 6 | NO |  |

i = i + 1 = 7; j = j + 1 = 12

* **j = 12, i = 7:**

|  |  |  |
| --- | --- | --- |
| i | X[i] != Y[j]? | mpNext[i] |
| 7 | NO |  |

i = i + 1 = 8; j = j + 1 = 13

* OUTPUT(13 – 8); i = kmpNext[i] = 1
* **j = 13, i = 1:**

|  |  |  |
| --- | --- | --- |
| i | X[i] != Y[j]? | mpNext[i] |
| 1 | YES | 0 |
| 1 | YES | -1 |
| -1 |  |  |

i = i + 1 = 0; j = j + 1 = 14

* **j = 14, i = 0:**

|  |  |  |
| --- | --- | --- |
| i | X[i] != Y[j]? | mpNext[i] |
| 0 | YES | -1 |
| -1 |  |  |

i = i + 1 = 0; j = j + 1 = 15

* **j = 15, i = 0:**

|  |  |  |
| --- | --- | --- |
| i | X[i] != Y[j]? | mpNext[i] |
| 0 | YES | -1 |
| -1 |  |  |

i = i + 1 = 0; j = j + 1 = 16

* **j = 16, i = 0:**

|  |  |  |
| --- | --- | --- |
| i | X[i] != Y[j]? | mpNext[i] |
| 0 | YES | -1 |
| -1 |  |  |

i = i + 1 = 0; j = j + 1 = 17

## Thuật toán Karp-Rabin:

### Đặc điểm:

* Sử dụng hàm băm
* Pha tiền xử lý có độ phức tạp về thời gian và bộ nhớ là O(m)
* Pha tìm kiếm có độ phức tạp về thời gian là O(m \* n)
* Thời gian chạy mong muốn là O(m + n)

### Thuật toán:

* Input:
  + Mẫu X = (x0, x1,…,xm-1), độ dài m
  + Văn bản Y = (y0, y1,…,yn-1), độ dài n
* Output: Tất cả các vị trí xuất hiện của X trong Y

|  |
| --- |
| #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 nghiệm:

* X[] = “GCAGAGAG”, m = 8
* Y[] = “GCATCGCAGAGAGTATACAGTACG”, n = 24

**Bước 1**: Tiền xử lý:

* + Tính d = 2m-1= 128
  + Tính hx = hash(X[0 .. m-1]) và hy = hash(Y[0 .. m-1]):

|  |  |  |
| --- | --- | --- |
| i | hx | hy |
| 0 | 71 | 71 |
| 1 | 209 | 209 |
| 2 | 483 | 483 |
| 3 | 1037 | 1050 |
| 4 | 2139 | 2139 |
| 5 | 4349 | 4405 |
| 6 | 8763 | 8877 |
| 7 | 17597 | 17819 |

* hx = 17597, hy = 17819

**Bước 2:** Tìm kiếm:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| j | hy | hx == hy? | X == Y[j .. j + m – 1] ? | Output |
| 0 | 17819 | NO |  |  |
| 1 | 17533 | NO |  |  |
| 2 | 17979 | NO |  |  |
| 3 | 19389 | NO |  |  |
| 4 | 17339 | NO |  |  |
| 5 | 17597 | YES | YES | 5 |
| 6 | 17102 | NO |  |  |
| 8 | 17678 | NO |  |  |
| 9 | 17245 | NO |  |  |
| 10 | 17917 | NO |  |  |
| 11 | 17723 | NO |  |  |
| 12 | 18877 | NO |  |  |
| 13 | 19662 | NO |  |  |
| 14 | 17885 | NO |  |  |
| 15 | 19197 | NO |  |  |
| 16 | 16961 | NO |  |  |

## Thuật toán Shift or:

### Đặc điểm:

* Sử dụng các thao tác bit.
* Hiệu quả nếu chiều dài mẫu không lớn hơn độ dài từ máy của máy tính.
* Pha tiền xử lý có độ phức tạp O(m)
* Pha tìm kiếm có độ phức tạp O(n)
* Có thể thay đổi dễ dàng để tìm chuỗi gần đúng

### Thuật toán:

* Input:
  + Mẫu X = (x0, x1,…,xm-1), độ dài m
  + Văn bản Y = (y0, y1,…,yn-1), độ dài n
* Output: Tất cả các vị trí xuất hiện của X trong Y

|  |
| --- |
| 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:

* X[] = “GCAGAGAG”, m = 8
* Y[] = “GCATCGCAGAGAGTATACAGTACG”, n = 24

**Bước 1**: Tiền xử lý:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | SA | SC | SG | ST |
| G | 1 | 1 | 0 | 1 |
| C | 1 | 0 | 1 | 1 |
| A | 0 | 1 | 1 | 1 |
| G | 1 | 1 | 0 | 1 |
| A | 0 | 1 | 1 | 1 |
| G | 1 | 1 | 0 | 1 |
| A | 0 | 1 | 1 | 1 |
| G | 1 | 1 | 0 | 1 |

**Bước 2:** Tìm kiếm:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| G | C | A | G | A | G | A | G |
| 0 | G | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | C | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | A | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 3 | T | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 4 | C | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 5 | G | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 6 | C | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 7 | A | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 8 | G | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 9 | A | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 10 | G | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| 11 | A | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| 12 | G | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 13 | T | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 14 | A | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 15 | T | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 16 | A | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 17 | C | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 18 | A | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 19 | G | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 20 | T | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 21 | A | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 22 | C | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 23 | G | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

## Thuật toán Morris-Pratt:

### Đặc điểm:

* Thực hiện từ trái sang phải
* Có pha tiền xử lý độ phức tạp O(m)
* Pha tiềm kiếm độ phức tạp O(n + m)
* Số lần so sánh tối đa cho 1 kí tự của xâu là m

### Thuật toán:

* Input:
  + Mẫu X = (x0, x1,…,xm-1), độ dài m
  + Văn bản Y = (y0, y1,…,yn-1), độ dài n
* Output: Tất cả các vị trí xuất hiện của X trong Y

|  |
| --- |
| 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:

* X[] = “GCAGAGAG”, m = 8
* Y[] = “GCATCGCAGAGAGTATACAGTACG”, n = 24

Bước 1: Tiền xử lý: preMp(X, 8, mpNext)

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

Bước 2: Tìm kiếm: MP(X, 8, Y, 24)

* **j = 0, i = 0:**

|  |  |  |
| --- | --- | --- |
| i | X[i] != Y[j]? | mpNext[i] |
| 0 | NO |  |

i = i + 1 = 1; j = j + 1 = 1

* **j = 1, i = 1:**

|  |  |  |
| --- | --- | --- |
| i | X[i] != Y[j]? | mpNext[i] |
| 1 | NO |  |

i = i + 1 = 2; j = j + 1 = 2

* **j = 2, i = 2:**

|  |  |  |
| --- | --- | --- |
| i | X[i] != Y[j]? | mpNext[i] |
| 2 | NO |  |

i = i + 1 = 3; j = j + 1 = 3

* **j = 3, i = 3:**

|  |  |  |
| --- | --- | --- |
| i | X[i] != Y[j]? | mpNext[i] |
| 3 | YES | 0 |
| 0 | YES | -1 |
| -1 |  |  |

i = i + 1 = 0; j = j + 1 = 4

* **j = 4, i = 0:**

|  |  |  |
| --- | --- | --- |
| i | X[i] != Y[j]? | mpNext[i] |
| 0 | YES | -1 |
| -1 |  |  |

i = i + 1 = 1; j = j + 1 = 5

* **j = 5, i = 0:**

|  |  |  |
| --- | --- | --- |
| i | X[i] != Y[j]? | mpNext[i] |
| 0 | NO |  |

i = i + 1 = 1; j = j + 1 = 6

* **j = 6, i = 1:**

|  |  |  |
| --- | --- | --- |
| i | X[i] != Y[j]? | mpNext[i] |
| 1 | NO |  |

i = i + 1 = 2; j = j + 1 = 7

* **j = 7, i = 2:**

|  |  |  |
| --- | --- | --- |
| i | X[i] != Y[j]? | mpNext[i] |
| 2 | NO |  |

i = i + 1 = 3; j = j + 1 = 8

* **j = 8, i = 3:**

|  |  |  |
| --- | --- | --- |
| i | X[i] != Y[j]? | mpNext[i] |
| 3 | NO |  |

i = i + 1 = 4; j = j + 1 = 9

* **j = 9, i = 4:**

|  |  |  |
| --- | --- | --- |
| i | X[i] != Y[j]? | mpNext[i] |
| 4 | NO |  |

i = i + 1 = 5; j = j + 1 = 10

* **j = 10, i = 5:**

|  |  |  |
| --- | --- | --- |
| i | X[i] != Y[j]? | mpNext[i] |
| 5 | NO |  |

i = i + 1 = 6; j = j + 1 = 11

* **j = 11, i = 6:**

|  |  |  |
| --- | --- | --- |
| i | X[i] != Y[j]? | mpNext[i] |
| 6 | NO |  |

i = i + 1 = 7; j = j + 1 = 12

* **j = 12, i = 7:**

|  |  |  |
| --- | --- | --- |
| i | X[i] != Y[j]? | mpNext[i] |
| 7 | NO |  |

i = i + 1 = 8; j = j + 1 = 13

* OUTPUT(13 – 8); i = mpNext[i] = 1
* **j = 13, i = 1:**

|  |  |  |
| --- | --- | --- |
| i | X[i] != Y[j]? | mpNext[i] |
| 1 | YES | 0 |
| 0 | YES | -1 |
| -1 |  |  |

i = i + 1 = 0; j = j + 1 = 14

* **j = 14, i = 0:**

|  |  |  |
| --- | --- | --- |
| i | X[i] != Y[j]? | mpNext[i] |
| 0 | YES | -1 |
| -1 |  |  |

i = i + 1 = 0; j = j + 1 = 15

* **j = 15, i = 0:**

|  |  |  |
| --- | --- | --- |
| i | X[i] != Y[j]? | mpNext[i] |
| 0 | YES | -1 |
| -1 |  |  |

i = i + 1 = 0; j = j + 1 = 16

* **j = 16, i = 0:**

|  |  |  |
| --- | --- | --- |
| i | X[i] != Y[j]? | mpNext[i] |
| 0 | YES | -1 |
| -1 |  |  |

i = i + 1 = 0; j = j + 1 = 17

## Thuật toán Not So Naive:

### Đặc điểm:

* Pha tiền xử lý có độ phức tạp là hằng số
* Độ phức tạp không gian nhớ hằng số
* Pha tìm kiếm độ phức tạp O(m \* n)
* Gần như tuyến tính trong trường hợp trung bình

### Thuật toán:

* Input:
  + Mẫu X = (x0, x1,…,xm-1), độ dài m
  + Văn bản Y = (y0, y1,…,yn-1), độ dài n
* Output: Tất cả các vị trí xuất hiện của X trong Y

|  |
| --- |
| 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:

* X[] = “GCAGAGAG”, m = 8
* Y[] = “GCATCGCAGAGAGTATACAGTACG”, n = 24

**Bước 1**: Tiền xử lý: X[0] != X[1] => k = 1, ell = 2

**Bước 2**: Tìm kiếm:

|  |  |  |  |
| --- | --- | --- | --- |
| j | X[1] == Y[j+1]? | X[2..m-1] == Y[j+2..j+m-1] && X[0] == Y[j]? | Output |
| 0 | YES | NO |  |
| 2 | NO |  |  |
| 3 | YES | NO |  |
| 5 | YES | YES | 5 |
| 7 | NO |  |  |
| 8 | NO |  |  |
| 9 | NO |  |  |
| 10 | NO |  |  |
| 11 | NO |  |  |
| 12 | NO |  |  |
| 13 | NO |  |  |
| 14 | NO |  |  |
| 15 | NO |  |  |
| 16 | NO |  |  |

## Thuật toán Apostolico-Crochemore:

### Đặc điểm:

* Pha tiền xử lý có độ phức tạp thời gian và bộ nhớ O(m)
* Pha tìm kiếm có độ phức tạp O(n)
* Thực hiện 3n/2 phép so sánh trong trường hợp xấu nhất

### Thuật toán:

|  |
| --- |
| 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:

* X[] = “GCAGAGAG”, m = 8
* Y[] = “GCATCGCAGAGAGTATACAGTACG”, n = 24

**Bước 1**: Tiền xử lý:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 |

l = 1

**Bước 2:** Tìm kiếm:

* **i = 1, j = 0, k = 0:**

|  |  |
| --- | --- |
| i | X[i] == Y[i+j]? |
| 1 | YES |
| 2 | YES |
| 3 | NO |

j += (i – kmpNext[i]) = 4

k = max{0, kmpNext[i]} = 0

i = l = 1

* **i = 1, j = 4, k = 0:**

|  |  |
| --- | --- |
| i | X[i] == Y[i+j]? |
| 1 | NO |

j += (i – kmpNext[i]) = 5

k = max{0, k – 1} = 0

* **i = 1, j = 5, k = 0:**

|  |  |
| --- | --- |
| i | X[i] == Y[i+j]? |
| 1 | YES |
| 2 | YES |
| 3 | YES |
| 4 | YES |
| 5 | YES |
| 6 | YES |
| 7 | YES |
| 8 |  |

|  |  |
| --- | --- |
| k | X[k]==Y[j+k] |
| 0 | YES |
| 1 |  |

OUTPUT(5)

j += (i – kmpNext[i]) = 12

k = max{0, kmpNext[i]} = 1

i = l = 1

* **i = 1, j = 12, k = 1:**

|  |  |
| --- | --- |
| i | X[i] == Y[i+j]? |
| 1 | NO |

j += (i – kmpNext[i]) = 13

k = max{0, k – 1} = 0

* **i = 1, j = 13, k = 0:**

|  |  |
| --- | --- |
| i | X[i] == Y[i+j]? |
| 1 | NO |

j += (i – kmpNext[i]) = 14

k = max{0, k – 1} = 0

* **i = 1, j = 14, k = 0:**

|  |  |
| --- | --- |
| i | X[i] == Y[i+j]? |
| 1 | NO |

j += (i – kmpNext[i]) = 15

k = max{0, k – 1} = 0

* **i = 1, j = 15, k = 0:**

|  |  |
| --- | --- |
| i | X[i] == Y[i+j]? |
| 1 | NO |

j += (i – kmpNext[i]) = 16

k = max{0, k – 1} = 0

* **i = 1, j = 16, k = 0:**

|  |  |
| --- | --- |
| i | X[i] == Y[i+j]? |
| 1 | YES |
| 2 | YES |
| 3 | YES |
| 4 | NO |

j += (i – kmpNext[i]) = 19

k = max{0, kmpNext[i]} = 1

i = l = 1

# Tìm kiếm mẫu từ phải sang trái:

## Thuật toán Boyer-Moore:

### Đặc điểm:

* Thực hiện từ phải sang trái
* Pha tiền xử lý có độ phức tạp O(m + ) (với độ lớn của tập hợp kí tự)
* Pha tìm kiếm có độ phức tạp O(m \* n)
* Mất 3n lần so sánh trong trường hợp xấu nhất khi so tìm kiếm 1 mẫu không có chu kỳ
* Trường hợp tốt nhất O(n / m)

### Thuật toán:

* Input:
  + Mẫu X = (x0, x1,…,xm-1), độ dài m
  + Văn bản Y = (y0, y1,…,yn-1), độ dài n
* Output: Tất cả các vị trí xuất hiện của X trong 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 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:

* X[] = “GCAGAGAG”, m = 8
* Y[] = “GCATCGCAGAGAGTATACAGTACG”, n = 24

**Bước 1**: Tiền xử lý:

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

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| X[i] | G | C | A | G | A | G | A | G |
| suff[i] | 1 | 0 | 0 | 2 | 0 | 4 | 0 | 8 |
| bmGs[i] | 7 | 7 | 7 | 2 | 7 | 4 | 7 | 1 |

**Bước 2**: Tìm kiếm:

* **j = 0**:

|  |  |
| --- | --- |
| i | X[i] == Y[i + j ] |
| 7 | NO |

* j += max(7, 1 – 7 + 7) = 1
* **j = 1:**

|  |  |
| --- | --- |
| i | X[i] == Y[i + j ] |
| 7 | YES |
| 6 | YES |
| 5 | NO |

* j += max(4, 6 – 7 + 1) = 5
* **j = 5:**

|  |  |
| --- | --- |
| i | X[i] == Y[i + j ] |
| 7 | YES |
| 6 | YES |
| 5 | YES |
| 4 | YES |
| 3 | YES |
| 2 | YES |
| 1 | YES |
| 0 | YES |
| -1 |  |

* Output(7); j += 7 = 12
* **j = 12:**

|  |  |
| --- | --- |
| i | X[i] == Y[i + j ] |
| 7 | YES |
| 6 | YES |
| 5 | NO |

* j += max(4, 6 – 7 + 1) = 16
* j = 16

|  |  |
| --- | --- |
| i | X[i] == Y[i + j ] |
| 7 | YES |
| 6 | NO |

* j += max(7, 6 – 7 + 1) = 23

## Thuật toán Turbo-BM:

### Đặc điểm:

* Là biến thể của thuật toán Boyer-Moore
* Không đòi hỏi thêm tiền xử lý nào
* Cần thêm không gian nhớ
* Pha tiền xử lý độ phức tạp O(m + )
* Pha tìm kiếm độ phức tạp O(n)
* Tốn 2n phép so sánh trong trường hợp xấu nhất

### Thuật toán:

* Input:
  + Mẫu X = (x0, x1,…,xm-1), độ dài m
  + Văn bản Y = (y0, y1,…,yn-1), độ dài n
* Output: Tất cả các vị trí xuất hiện của X trong Y

|  |
| --- |
| 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:

* X[] = “GCAGAGAG”, m = 8
* Y[] = “GCATCGCAGAGAGTATACAGTACG”, n = 24

**Bước 1**: Tiền xử lý:

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

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| X[i] | G | C | A | G | A | G | A | G |
| suff[i] | 1 | 0 | 0 | 2 | 0 | 4 | 0 | 8 |
| bmGs[i] | 7 | 7 | 7 | 2 | 7 | 4 | 7 | 1 |

**Bước 2**: Tìm kiếm:

* **j = 0; u = 0; shift = 8:**

|  |  |  |
| --- | --- | --- |
| i | X[i] == Y[i + j]? | i >= 0 && i == m – 1 - shitf |
| 7 | NO |  |

v = m – 1 – i = 0

turboShift = u – v = 0

bcShift = bmBc[Y[i + j]] – m + 1+ i = 1

shift = max{turboShift, bcShift, bmGs[i]} = bmGs[i] = 1

u = min{m – shift, v} = 0

j += shift = 1

* **j = 1; u = 0; shift = 1:**

|  |  |  |
| --- | --- | --- |
| i | X[i] == Y[i + j]? | i >= 0 && i == m – 1 - shitf |
| 7 | YES | NO |
| 6 | YES | YES |
| 5 | NO |  |

v = m – 1 – i = 2

turboShift = u – v = -2

bcShift = bmBc[Y[i + j]] – m + 1+ i = 4

shift = max{turboShift, bcShift, bmGs[i]} = bmGs[i] = 4

u = min{m – shift, v} = 3

j += shift = 5

* **j = 5; u = 2; shift = 4:**

|  |  |  |
| --- | --- | --- |
| i | X[i] == Y[i + j]? | i >= 0 && i == m – 1 - shitf |
| 7 | YES | NO |
| 6 | YES | NO |
| 5 | YES | NO |
| 4 | YES | NO |
| 3 | YES | YES |
| 1 | YES | NO |
| 0 | YES | NO |
| -1 |  |  |

OUTPUT(5)

shift = bmGs[0] = 7

u = m – shift = 1

j += shift = 12

* **j = 12; u = 1; shift = 7:**

|  |  |  |
| --- | --- | --- |
| i | X[i] == Y[i + j]? | i >= 0 && i == m – 1 - shitf |
| 7 | YES | NO |
| 6 | YES | NO |
| 5 | NO |  |

v = m – 1 – i = 2

turboShift = u – v = -2

bcShift = bmBc[Y[i + j]] – m + 1+ i = 4

shift = max{turboShift, bcShift, bmGs[i]} = bmGs[i] = 4

u = min{m – shift, v} = 3

j += shift = 16

* **j = 16; u = 3; shift = 4:**

|  |  |  |
| --- | --- | --- |
| i | X[i] == Y[i + j]? | i >= 0 && i == m – 1 - shitf |
| 7 | YES | NO |
| 6 | NO |  |

v = m – 1 – i = 1

turboShift = u – v = 2

bcShift = bmBc[Y[i + j]] – m + 1+ i = 5

shift = max{turboShift, bcShift, bmGs[i]} = bmGs[i] = 7

u = min{m – shift, v} = 1

j += shift = 23

## Thuật toán Zhu-Takaoka:

### Đặc điểm:

* Là biến thể của thuật toán Boyer-Moore
* Dùng 2 kí tự liên tiếp để tính bad-character shift
* Pha tiền xử lí độ phức tạp O(m + )
* Pha tìm kiếm độ phức tạp (m \* n)

### Thuật toán:

* Input:
  + Mẫu X = (x0, x1,…,xm-1), độ dài m
  + Văn bản Y = (y0, y1,…,yn-1), độ dài n
* Output: Tất cả các vị trí xuất hiện của X trong Y

|  |
| --- |
| 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:

* X[] = “GCAGAGAG”, m = 8
* Y[] = “GCATCGCAGAGAGTATACAGTACG”, n = 24

**Bước 1**: Tiền xử lý:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ztBC | A | C | G | T |
| A | 8 | 8 | 2 | 8 |
| C | 5 | 8 | 7 | 8 |
| G | 1 | 6 | 7 | 8 |
| T | 8 | 8 | 7 | 8 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| X[i] | G | C | A | G | A | G | A | G |
| suff[i] | 1 | 0 | 0 | 2 | 0 | 4 | 0 | 8 |
| bmGs[i] | 7 | 7 | 7 | 2 | 7 | 4 | 7 | 1 |

**Bước 2**: Tìm kiếm:

* **j = 0**:

|  |  |
| --- | --- |
| i | X[i] == Y[i + j]? |
| 7 | NO |

max{bmGs[i], ztBc[Y[j+m-2]][Y[j+m-1]]} = 5

j += 5 = 5

* **j = 5:**

|  |  |
| --- | --- |
| i | X[i] == Y[i + j]? |
| 7 | YES |
| 6 | YES |
| 5 | YES |
| 4 | YES |
| 3 | YES |
| 2 | YES |
| 1 | YES |
| 0 | YES |
| -1 |  |

OUTPUT(j)

j += bmGs[0] = 12

* **j = 12:**

|  |  |
| --- | --- |
| i | X[i] == Y[i + j]? |
| 7 | YES |
| 6 | YES |
| 5 | NO |

max{bmGs[i], ztBc[Y[j+m-2]][Y[j+m-1]]} = 4

j += 5 = 16

* **j = 16:**

|  |  |
| --- | --- |
| i | X[i] == Y[i + j]? |
| 7 | YES |
| 6 | NO |

max{bmGs[i], ztBc[Y[j+m-2]][Y[j+m-1]]} = 7

j += 7 = 23

## Thuật toán Berry-Ravindran:

### Đặc điểm:

* Kết hợp giữa thuật toán Quick Search và thuật toán Zhu-Takaoka
* Pha tiền xử lí có độ phức tạp thời gian và bộ nhớ là O(m+)
* Pha tìm kiếm độ phức tạp O(m\*n)

### 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 nghiệm:

* X[] = “GCAGAGAG”, m = 8
* Y[] = “GCATCGCAGAGAGTATACAGTACG”, n = 24

**Bước 1**: 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 |

**Bước 2:** Tìm kiếm:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| j | X == Y[j..j+m-1]? | Y[j+m] | Y[j+m+1] | Output |
| 0 | NO | G | A |  |
| 1 | NO | A | G |  |
| 3 | NO | A | G |  |
| 5 | YES | T | A | 5 |
| 15 | NO | G | \0 |  |
| 16 | NO | \0 | \0 |  |
| 26 |  |  |  |  |

## Thuật toán:

### Đặc điểm:

### Thuật toán:

### Kiểm nghiệm:

# Tìm kiếm mẫu từ vị trí xác định:

## Thuật toán Two way:

### Đặc điểm:

* Pha tiền xử lí độ phức tạp O(m)
* Pha tìm kiếm độ phức tạp n
* Thực hiện 2n-m phép so sánh trong trường hợp xấu nhất

### 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);  }  }  } |

### Kiểm nghiệm:

## Thuật toán Colussi:

### Đặc điểm:

* Cải tiến từ thuật toán Knuth-Morris Pratt
* Chia tập hợp các vị trí của mẫu thành 2 tập con: các vị trí trong tập con thứ nhất được duyệt từ trái sang phải và khi không có sự sai khác nào xảy ra thì các vị trí của tập con thứ 2 được duyệt từ phải sang trái.
* Pha tiền xử lý độ phức tạp O(m)
* Pha tìm kiếm độ phức tạp O(n)
* Thực hiện 3n/2 phép so sánh trong trường hợp xấu nhất

### 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:

* X[] = “GCAGAGAG”, m = 8
* Y[] = “GCATCGCAGAGAGTATACAGTACG”, n = 24

**Bước 1**: Tiền xử lý:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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

**Bước 2**: Tìm kiếm:

* **Lặp lần 1**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| i | j | last | X[h[i]]==Y[j+h[i]]? | Output |
| 0 | 0 | -1 | YES |  |
| 1 | 0 | -1 | YES |  |
| 2 | 0 | -1 | NO |  |

j += shift[2] = 3

i = next[i] = 0

* **Lặp lần 2:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| i | j | last | X[h[i]]==Y[j+h[i]]? | Output |
| 0 | 3 | -1 | YES |  |
| 1 | 3 | -1 | NO |  |

j += shift[1] = 5

i = next[1] = 0

* **Lặp lần 3:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| i | j | last | X[h[i]]==Y[j+h[i]]? | Output |
| 0 | 5 | -1 | YES |  |
| 1 | 5 | -1 | YES |  |
| 2 | 5 | -1 | YES |  |
| 3 | 5 | -1 | YES |  |
| 4 | 5 | -1 | YES |  |
| 5 | 5 | -1 | YES |  |
| 6 | 5 | -1 | YES |  |
| 7 | 5 | -1 | YES |  |
| 8 |  |  |  |  |

OUTPUT(5)

last = j + m – 1 = 12

j += shift[8] = 12

i = next[8] = 0

* **Lặp lần 4:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| i | j | last | X[h[i]]==Y[j+h[i]]? | Output |
| 0 | 12 | 12 | NO |  |

j += shift[0] = 13

i = next[0] = 0

* **Lặp lần 5:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| i | j | last | X[h[i]]==Y[j+h[i]]? | Output |
| 0 | 13 | 12 | NO |  |

j += shift[0] = 14

i = next[0] = 0

* **Lặp lần 6:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| i | j | last | X[h[i]]==Y[j+h[i]]? | Output |
| 0 | 14 | 12 | NO |  |

j += shift[0] = 15

i = next[0] = 0

* **Lặp lần 7:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| i | j | last | X[h[i]]==Y[j+h[i]]? | Output |
| 0 | 15 | 12 | NO |  |

j += shift[0] = 16

i = next[0] = 0

* **Lặp lần 8:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| i | j | last | X[h[i]]==Y[j+h[i]]? | Output |
| 0 | 16 | 12 | YES |  |
| 1 | 16 | 12 | YES |  |
| 2 | 16 | 12 | NO |  |

j += shift[2] = 19

i = next[2] = 0

## Thuật toán Skip Search:

### Đặc điểm:

* Sử dụng “bucket” các vị trí cho mỗi kí tự trong tập kí tự
* Pha tiền xử lí độ phức tạp O(m+)
* Pha tìm kiếm độ phức tạp O(m\*n)
* Mong đợi O(n) phép so sánh kí tự

### Thuật toán:

|  |
| --- |
| void SKIP(char \*x, int m, char \*y, int n) {  int i, j;  List ptr, z[ASIZE];  /\* Preprocessing \*/  memset(z, NULL, ASIZE\*sizeof(List));  for (i = 0; i < m; ++i) {  ptr = (List)malloc(sizeof(struct \_cell));  if (ptr == NULL)  error("SKIP");  ptr->element = i;  ptr->next = z[x[i]];  z[x[i]] = ptr;  }    /\* Searching \*/  for (j = m - 1; j < n; j += m)  for (ptr = z[y[j]]; ptr != NULL; ptr = ptr->next)  if (memcmp(x, y + j - ptr->element, m) == 0)  OUTPUT(j - ptr->element);  } |

### Kiểm nghiệm:

* X[] = “GCAGAGAG”, m = 8
* Y[] = “GCATCGCAGAGAGTATACAGTACG”, n = 24

**Bước 1:** Tiền xử lý:

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

**Bước 2:** Tìm kiếm:

* **j = 7, Y[j] = ‘A’:**

|  |  |  |
| --- | --- | --- |
| z[Y[j]] | X == Y[j – z[Y[j]], j – z[Y[j]] + m]? | Output |
| 6 | NO |  |
| 4 | NO |  |
| 2 | YES | 5 |

* **j = 15, Y[j] = ‘T’:**
* **j = 23, Y[j] = ‘G’:**

|  |  |  |
| --- | --- | --- |
| z[Y[j]] | X == Y[j – z[Y[j]], j – z[Y[j]] + m]? | Output |
| 7 | NO |  |
| 5 | NO |  |
| 3 | NO |  |
| 0 | NO |  |

## Thuật toán KMP Skip Search:

### Đặc điểm:

* Cải tiến từ thuật toán Skip Search
* Sử dụng “bucket” các vị trí cho mỗi kí tự trong tập kí tự
* Pha tiền xử lí độ phức tạp O(m+)
* Pha tìm kiếm độ phức tạp O(n)

### Thuật toán:

|  |
| --- |
| int attempt(char \*y, char \*x, int m, int start, int wall) {  int k;  k = wall - start;  while (k < m && x[k] == y[k + start])  ++k;  return(k);  }  void KMPSKIP(char \*x, int m, char \*y, int n) {  int i, j, k, kmpStart, per, start, wall;  int kmpNext[XSIZE], list[XSIZE], mpNext[XSIZE],  z[ASIZE];  /\* Preprocessing \*/  preMp(x, m, mpNext);  preKmp(x, m, kmpNext);  memset(z, -1, ASIZE\*sizeof(int));  memset(list, -1, m\*sizeof(int));  z[x[0]] = 0;  for (i = 1; i < m; ++i) {  list[i] = z[x[i]];  z[x[i]] = i;  }  /\* Searching \*/  wall = 0;  per = m - kmpNext[m];  i = j = -1;  do {  j += m;  } while (j < n && z[y[j]] < 0);  if (j >= n)  return;  i = z[y[j]];  start = j - i;  while (start <= n - m) {  if (start > wall)  wall = start;  k = attempt(y, x, m, start, wall);  wall = start + k;  if (k == m) {  OUTPUT(start);  i -= per;  }  else  i = list[i];  if (i < 0) {  do {  j += m;  } while (j < n && z[y[j]] < 0);  if (j >= n)  return;  i = z[y[j]];  }  kmpStart = start + k - kmpNext[k];  k = kmpNext[k];  start = j - i;  while (start < kmpStart ||  (kmpStart < start && start < wall)) {  if (start < kmpStart) {  i = list[i];  if (i < 0) {  do {  j += m;  } while (j < n && z[y[j]] < 0);  if (j >= n)  return;  i = z[y[j]];  }  start = j - i;  }  else {  kmpStart += (k - mpNext[k]);  k = mpNext[k];  }  }  }  } |

### Kiểm nghiệm:

* X[] = “GCAGAGAG”, m = 8
* Y[] = “GCATCGCAGAGAGTATACAGTACG”, n = 24

**Bước 1:** Tiền xử lý:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| c | A | C | G | T |
| z[c] | 6 | 1 | 7 | -1 |

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

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 |
| kmpNext[i] | -1 | 0 | 0 | -1 | 1 | -1 | 1 | -1 | 1 |

**Bước 2:** Tìm kiếm:

**Khởi tạo:**

wall = 0

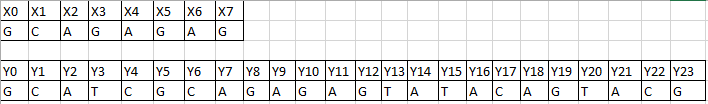
per = m – kmpNext[m] = 7

j = 7

i = z[Y[j]] = 6

start = j – i = 1

**Lặp:**



## Thuật toán Alpha Skip Search:

### Đặc điểm:

* Cải tiến từ thuật toán Skip Search
* Dùng “bucket” chứa các vị trí cho mỗi đoạn chiều dài của mẫu
* Pha tiền xử lí có độ phức tạp O(m)
* Pha tìm kiếm có độ phức tạp O(m\*n)
* Mong đợi có O( \* (n/(m-))) phép so sánh

### Thuật toán:

|  |
| --- |
| List \*z;    #define getZ(i) z[(i)]    void setZ(int node, int i) {  List cell;    cell = (List)malloc(sizeof(struct \_cell));  if (cell == NULL)  error("ALPHASKIP/setZ");  cell->element = i;  cell->next = z[node];  z[node] = cell;  }      /\* Create the transition labelled by the  character c from node node.  Maintain the suffix links accordingly. \*/  int addNode(Graph trie, int art, int node, char c) {  int childNode, suffixNode, suffixChildNode;    childNode = newVertex(trie);  setTarget(trie, node, c, childNode);  suffixNode = getSuffixLink(trie, node);  if (suffixNode == art)  setSuffixLink(trie, childNode, node);  else {  suffixChildNode = getTarget(trie, suffixNode, c);  if (suffixChildNode == UNDEFINED)  suffixChildNode = addNode(trie, art,  suffixNode, c);  setSuffixLink(trie, childNode, suffixChildNode);  }  return(childNode);  }  void ALPHASKIP(char \*x, int m, char \*y, int n, int a) {  int b, i, j, k, logM, temp, shift, size, pos;  int art, childNode, node, root, lastNode;  List current;  Graph trie;    logM = 0;  temp = m;  while (temp > a) {  ++logM;  temp /= a;  }  if (logM == 0) logM = 1;      /\* Preprocessing \*/  size = 2 + (2\*m - logM + 1)\*logM;  trie = newTrie(size, size\*ASIZE);  z = (List \*)calloc(size, sizeof(List));  if (z == NULL)  error("ALPHASKIP");    root = getInitial(trie);  art = newVertex(trie);  setSuffixLink(trie, root, art);  node = newVertex(trie);  setTarget(trie, root, x[0], node);  setSuffixLink(trie, node, root);  for (i = 1; i < logM; ++i)  node = addNode(trie, art, node, x[i]);  pos = 0;  setZ(node, pos);  pos++;  for (i = logM; i < m - 1; ++i) {  node = getSuffixLink(trie, node);  childNode = getTarget(trie, node, x[i]);  if (childNode == UNDEFINED)  node = addNode(trie, art, node, x[i]);  else  node = childNode;  setZ(node, pos);  pos++;  }  node = getSuffixLink(trie, node);  childNode = getTarget(trie, node, x[i]);  if (childNode == UNDEFINED) {  lastNode = newVertex(trie);  setTarget(trie, node, x[m - 1], lastNode);  node = lastNode;  }  else  node = childNode;  setZ(node, pos);    /\* Searching \*/  shift = m - logM + 1;  for (j = m + 1 - logM; j < n - logM; j += shift) {  node = root;  for (k = 0; node != UNDEFINED && k < logM; ++k)  node = getTarget(trie, node, y[j + k]);  if (node != UNDEFINED)  for (current = getZ(node);  current != NULL;  current = current->next) {  b = j - current->element;  if (x[0] == y[b] &&  memcmp(x + 1, y + b + 1, m - 1) == 0)  OUTPUT(b);  }  }  free(z);  } |

### Kiểm nghiệm:

* X[] = “GCAGAGAG”, m = 8
* Y[] = “GCATCGCAGAGAGTATACAGTACG”, n = 24

**Bước 1:** Tiền xử lý:

|  |  |
| --- | --- |
| u | z[u] |
| AGA | (4, 2) |
| CAG | (1) |
| GAG | (5, 3) |
| GCA | (0) |

**Bước 2:** Tìm kiếm:

**Khởi tạo:** shift = m – 3 + 1 = 6

**Lặp:**

## Thuật toán Optimal Mismatch:

### Đặc điểm:

* Là biến thể của thuật toán Quick Search
* Yêu cầu số lần xuất hiện của mỗi kí tự
* Pha tiền xử lí có độ phức tạp thời gian O() và độ phức tạp không gian nhớ O(m + )
* Pha tìm kiếm độ phức tạp O(m\*n)

### Thuật toán:

|  |
| --- |
| typedef struct patternScanOrder {  int loc;  char c;  } pattern;  int freq[ASIZE];  /\* Construct an ordered pattern from a string. \*/  void orderPattern(char \*x, int m, int (\*pcmp)(),  pattern \*pat) {  int i;  for (i = 0; i <= m; ++i) {  pat[i].loc = i;  pat[i].c = x[i];  }  qsort(pat, m, sizeof(pattern), pcmp);  }  /\* Optimal Mismatch pattern comparison function. \*/  int optimalPcmp(pattern \*pat1, pattern \*pat2) {  float fx;  fx = freq[pat1->c] - freq[pat2->c];  return(fx ? (fx > 0 ? 1 : -1) :  (pat2->loc - pat1->loc));  }  /\* Find the next leftward matching shift for  the first ploc pattern elements after a  current shift or lshift. \*/  int matchShift(char \*x, int m, int ploc,  int lshift, pattern \*pat) {  int i, j;  for (; lshift < m; ++lshift) {  i = ploc;  while (--i >= 0) {  if ((j = (pat[i].loc - lshift)) < 0)  continue;  if (pat[i].c != x[j])  break;  }  if (i < 0)  break;  }  return(lshift);  }  /\* Constructs the good-suffix shift table  from an ordered string. \*/  void preAdaptedGs(char \*x, int m, int adaptedGs[],  pattern \*pat) {  int lshift, i, ploc;  adaptedGs[0] = lshift = 1;  for (ploc = 1; ploc <= m; ++ploc) {  lshift = matchShift(x, m, ploc, lshift, pat);  adaptedGs[ploc] = lshift;  }  for (ploc = 0; ploc <= m; ++ploc) {  lshift = adaptedGs[ploc];  while (lshift < m) {  i = pat[ploc].loc - lshift;  if (i < 0 || pat[ploc].c != x[i])  break;  ++lshift;  lshift = matchShift(x, m, ploc, lshift, pat);  }  adaptedGs[ploc] = lshift;  }  }  /\* Optimal Mismatch string matching algorithm. \*/  void OM(char \*x, int m, char \*y, int n) {  int i, j, adaptedGs[XSIZE], qsBc[ASIZE];  pattern pat[XSIZE];  /\* Preprocessing \*/  orderPattern(x, m, optimalPcmp, pat);  preQsBc(x, m, qsBc);  preAdaptedGs(x, m, adaptedGs, pat);  /\* Searching \*/  j = 0;  while (j <= n - m) {  i = 0;  while (i < m && pat[i].c == y[j + pat[i].loc])  ++i;  if (i >= m)  OUTPUT(j);  j += MAX(adaptedGs[i],qsBc[y[j + m]]);  }  } |

### Kiểm nghiệm:

* X[] = “GCAGAGAG”, m = 8
* Y[] = “GCATCGCAGAGAGTATACAGTACG”, n = 24

**Bước 1:** Tiền xử lý:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 |

**Bước 2:** Tìm kiếm:

* **j = 0:**

|  |  |
| --- | --- |
| i | pat[i].c == Y[j + pat[i].loc]? |
| 0 | YES |
| 1 | NO |

j += max{adaptedGs[i], qsBc[Y[j+m]] = 3

* **j = 3:**

|  |  |
| --- | --- |
| i | pat[i].c == Y[j + pat[i].loc]? |
| 0 | YES |
| 1 | YES |
| 2 | YES |
| 3 | NO |

j += max{adaptedGs[i], qsBc[Y[j+m]] = 5

* **j = 5:**

|  |  |
| --- | --- |
| i | pat[i].c == Y[j + pat[i].loc]? |
| 0 | YES |
| 1 | YES |
| 2 | YES |
| 3 | YES |
| 4 | YES |
| 5 | YES |
| 6 | YES |
| 7 | YES |
| 8 |  |

OUTPUT(5)

j += max{adaptedGs[i], qsBc[Y[j+m]] = 14

* **j = 14:**

|  |  |
| --- | --- |
| i | pat[i].c == Y[j + pat[i].loc]? |
| 0 | NO |

j += max{adaptedGs[i], qsBc[Y[j+m]] = 21

## Thuật toán Maximal Shift:

### Đặc điểm:

* Là 1 biến thể của thuật toán Quick Search
* Độ phức tạp bậc 2 trong trường hợp xấu nhất
* Pha tiền xử lí độ phức tạp thời gian O(), độ phức tạp không gian nhớ O(m + )
* Pha tìm kiếm độ phức tạp O(m\*n)

### Thuật toán:

|  |
| --- |
| typedef struct patternScanOrder {  int loc;  char c;  } pattern;  int minShift[XSIZE];  /\* Computation of the MinShift table values. \*/  void computeMinShift(char \*x, int m) {  int i, j;  for (i = 0; i < m; ++i) {  for (j = i - 1; j >= 0; --j)  if (x[i] == x[j])  break;  minShift[i] = i - j;  }  }  /\* Maximal Shift pattern comparison function. \*/  int maxShiftPcmp(pattern \*pat1, pattern \*pat2) {  int dsh;  dsh = minShift[pat2->loc] - minShift[pat1->loc];  return(dsh ? dsh : (pat2->loc - pat1->loc));  }  /\* Maximal Shift string matching algorithm. \*/  void MS(char \*x, int m, char \*y, int n) {  int i, j, qsBc[ASIZE], adaptedGs[XSIZE];  pattern pat[XSIZE];  /\* Preprocessing \*/  computeMinShift(x ,m);  orderPattern(x, m, maxShiftPcmp, pat);  preQsBc(x, m, qsBc);  preAdaptedGs(x, m, adaptedGs, pat);  /\* Searching \*/  j = 0;  while (j <= n - m) {  i = 0;  while (i < m && pat[i].c == y[j + pat[i].loc])  ++i;  if (i >= m)  OUTPUT(j);  j += MAX(adaptedGs[i], qsBc[y[j + m]]);  }  } |

### Kiểm nghiệm:

* X[] = “GCAGAGAG”, m = 8
* Y[] = “GCATCGCAGAGAGTATACAGTACG”, n = 24

**Bước 1:** Tiền xử lý:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| c | A | C | G | T |
| 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 |
| minShift[i] | 1 | 2 | 3 | 3 | 2 | 2 | 2 | 2 |
| pat[i].loc | 3 | 2 | 7 | 6 | 5 | 4 | 1 | 0 |
| pat[i].c | G | A | G | A | G | A | C | G |

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

**Bước 2:** Tìm kiếm:

* **j = 0:**

|  |  |
| --- | --- |
| i | pat[i].c == Y[j + pat[i].loc]? |
| 0 | NO |

j += max{adaptedGs[i], qsBc[Y[j+m]] = 1

* **j = 1:**

|  |  |
| --- | --- |
| i | pat[i].c == Y[j + pat[i].loc]? |
| 0 | NO |

j += max{adaptedGs[i], qsBc[Y[j+m]] = 2

* **j = 3:**

|  |  |
| --- | --- |
| i | pat[i].c == Y[j + pat[i].loc]? |
| 0 | NO |

j += max{adaptedGs[i], qsBc[Y[j+m]] = 5

* **j = 5:**

|  |  |
| --- | --- |
| i | pat[i].c == Y[j + pat[i].loc]? |
| 0 | YES |
| 1 | YES |
| 2 | YES |
| 3 | YES |
| 4 | YES |
| 5 | YES |
| 6 | YES |
| 7 | YES |
| 8 |  |

OUTPUT(5)

j += max{adaptedGs[i], qsBc[Y[j+m]] = 14

* **j = 14:**

|  |  |
| --- | --- |
| i | pat[i].c == Y[j + pat[i].loc]? |
| 0 | NO |

j += max{adaptedGs[i], qsBc[Y[j+m]] = 21

# Tìm kiếm mẫu từ vị trí bất kỳ:

## Thuật toán Horspool:

### Đặc điểm:

* Là phiên bản đơn giản của thuật toán Boyer-Moore
* Chỉ dùng bad-character shift
* Dễ cài đặt
* Pha tiền xử lý có độ phức tạp thời gian O(m+) và độ phức tạp bộ nhớ O()
* Pha tìm kiếm có độ phức tạp O(m\*n)
* Số phép so sánh trung bình cho 1 kí tự từ 1/ đến 2/

( độ lớn của tập hợp kí tự)

### Thuật toán:

|  |
| --- |
| 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 nghiệm:

* X[] = “GCAGAGAG”, m = 8
* Y[] = “GCATCGCAGAGAGTATACAGTACG”, n = 24

**Bước 1**: Tiền xử lý:

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

**Bước 2:** Tìm kiếm:

|  |  |  |  |
| --- | --- | --- | --- |
| j | c = Y[j+m-1] | X[m-1] == c && X[0..m-2] == Y[j, j+m-2]? | Ouput |
| 0 | A | NO |  |
| 1 | G | NO |  |
| 3 | G | NO |  |
| 5 | G | YES | 5 |
| 7 | A | NO |  |
| 8 | T | NO |  |
| 16 | G | NO |  |
| 23 |  |  |  |

## Thuật toán Quick Search:

### Đặc điểm:

* Là phiên bản đơn giản của thuật toán Boyer-Moore
* Chỉ dùng bad-character shift
* Dễ cài đặt
* Pha tiền xử lý có độ phức tạp thời gian O(m+) và độ phức tạp bộ nhớ O()
* Pha tìm kiếm có độ phức tạp O(m\*n)
* Rất nhanh với trường hợp mẫu ngắn và tập hợp kí tự lớn

### 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:

* X[] = “GCAGAGAG”, m = 8
* Y[] = “GCATCGCAGAGAGTATACAGTACG”, n = 24

**Bước 1**: Tiền xử lý:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| c | A | C | G | T |
| qsBc[c] | 2 | 7 | 1 | 9 |

**Bước 2:** Tìm kiếm:

|  |  |  |
| --- | --- | --- |
| j | X == Y[j, j+m-1]? | Ouput |
| 0 | NO |  |
| 1 | NO |  |
| 3 | NO |  |
| 5 | YES | 5 |
| 14 | NO |  |
| 21 |  |  |

## Thuật toán Smith:

### Đặc điểm:

* Lấy max của hàm Horspool bad-character shift và hàm Quick Search bad-character shift
* Pha tiền xử lý có độ phức tạp thời gian O(m+) và độ phức tạp bộ nhớ O()
* Pha tìm kiếm có độ phức tạp O(m\*n)

### Thuật toán:

|  |
| --- |
| void SMITH(char \*x, int m, char \*y, int n) {  int j, bmBc[ASIZE], qsBc[ASIZE];  /\* Preprocessing \*/  preBmBc(x, m, bmBc);  preQsBc(x, m, qsBc);  /\* Searching \*/  j = 0;  while (j<= n - m) {  if (memcmp(x, y + j, m) == 0)  OUTPUT(j);  j += MAX(bmBc[y[j + m - 1]], qsBc[y[j + m]]);  }  } |

### Kiểm nghiệm:

* X[] = “GCAGAGAG”, m = 8
* Y[] = “GCATCGCAGAGAGTATACAGTACG”, n = 24

**Bước 1**: Tiền xử lý:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| c | A | C | G | T |
| bmBc[c] | 1 | 6 | 2 | 8 |
| qsBc[c] | 2 | 7 | 1 | 9 |

**Bước 2:** Tìm kiếm:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| j | X == Y[j, j+m-1]? | bmBc[Y[j+m-1]] | qsBc[Y[j+m]] | Ouput |
| 0 | NO | 1 | 1 |  |
| 1 | NO | 2 | 2 |  |
| 3 | NO | 2 | 2 |  |
| 5 | YES | 2 | 9 | 5 |
| 14 | NO | 1 | 7 |  |
| 21 |  |  |  |  |

## Thuật toán Raita:

### Đặc điểm:

* Thực hiện phép dịch như thuật toán Horspool
* Đầu tiên so sánh kí thự cuối của mẫu, sau đó là kí tự đầu tiên, rồi đến kí tự giữa trước khi so sánh các kí tự khác
* Pha tiền xử lý có độ phức tạp thời gian O(m+) và độ phức tạp bộ nhớ O()
* Pha tìm kiếm có độ phức tạp O(m\*n)

### Thuật toán:

|  |
| --- |
| 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:

* X[] = “GCAGAGAG”, m = 8
* Y[] = “GCATCGCAGAGAGTATACAGTACG”, n = 24

**Bước 1**: Tiền xử lý:

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

firstCh = ‘G’

middleCh = ‘A’

lastCh = ‘G’

**Bước 2:** Tìm kiếm:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| j | c = Y[j+m-1] == lastCh? | Y[j]==firstCh? | Y[j+m/2]==middleCh? | X[1..m-2] == Y[j+1..j+m=2]? | Ouput |
| 0 | NO |  |  |  |  |
| 1 | YES | NO |  |  |  |
| 3 | YES | NO |  |  |  |
| 5 | YES | YES | YES | YES | 5 |
| 7 | NO |  |  |  |  |
| 8 | NO |  |  |  |  |
| 16 | YES | NO |  |  |  |
| 18 |  |  |  |  |  |

## Thuật toán Tuned Boyer-Moore:

### Đặc điểm:

* Là phiên bản đơn giản của thuật toán Boyer-Moore
* Dễ cài đặt
* Rất nhanh trong thực tế
* Pha tiền xử lý có độ phức tạp O(m + ) (với độ lớn của tập hợp kí tự)
* Pha tìm kiếm có độ phức tạp O(m \* n)

### Thuật toán:

|  |
| --- |
| 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:

* X[] = “GCAGAGAG”, m = 8
* Y[] = “GCATCGCAGAGAGTATACAGTACG”, n = 24

**Bước 1**: Tiền xử lý:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| c | A | C | G | T |
| bmBc[c] | 1 | 6 | 0 | 8 |

shift = 2

**Bước 2:** Tìm kiếm:

* **j = 0:**

k = bmBc[Y[j+m-1]] = 1

|  |  |
| --- | --- |
| j | k = bmBc[Y[j+m-1]] |
| 1 | 0 |
| 1 | 0 |
| 1 | 0 |

X[0..m-2] != Y[j..j+m-2]

j += 2 = 3

* **j = 3:**

k = bmBc[Y[j+m-1]] = 0

X[0..m-2] != Y[j..j+m-2]

j += 2 = 5

* **j = 5:**

k = bmBc[Y[j+m-1]] = 0

X[0..m-2] == Y[j..j+m-2] => OUTPUT(5)

j += 2 = 7

* **j = 7:**

k = bmBc[Y[j+m-1]] = 1

|  |  |
| --- | --- |
| j | k = bmBc[Y[j+m-1]] |
| 8 | 8 |
| 16 | 0 |
| 16 | 0 |

X[0..m-2] != Y[j..j+m-2]

j += 2 = 17