**TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**Viện Công nghệ Thông tin và Truyền thông**

****

**Báo cáo Project 1**

**Đề tài:**

***Bài toán Tìm kiếm xâu mẫu (String Matching Problem)***

Sinh viên thực hiện:

**Hồ Xuân Hùng - SHHV: 20161935**

Mã lớp: 676422

Giáo viên hướng dẫn:

**PGS Nguyễn Đức Nghĩa**

**HÀ NỘI – 2018**

**MỤC LỤC**

[**I.** **Các vấn đề lý thuyết** 3](#_Toc531173221)

[1. Đặt vấn đề 3](#_Toc531173222)

[2. Khái niệm. 3](#_Toc531173223)

[3. Phân loại. 3](#_Toc531173224)

[4. Tìm hiểu một số thuật toán tìm kiếm xâu mẫu cơ bản. 4](#_Toc531173225)

[***a.*** ***Brute Force Algorithm*** 4](#_Toc531173226)

[***b.*** ***Knuth- Morris-Pratt Algorithm*** 4](#_Toc531173227)

[***c.*** ***Boyer – Moore*** 7](#_Toc531173228)

[**II.** **Hệ thống chương trình giải bài toán** 11](#_Toc531173229)

[***1.*** ***Định nghĩa:*** 11](#_Toc531173230)

[***2.*** ***Các biến toàn cục:*** 11](#_Toc531173231)

[*3. Hàm* ***main()*** *Chương trình gồm các bước* 11](#_Toc531173232)

[***4.*** ***Các module*** 12](#_Toc531173233)

[**III.** **Kết quả tính toán** 14](#_Toc531173234)

[1. Bộ dữ liệu test thử chương trình: 14](#_Toc531173235)

[2. Thống kê thời gian tính của chương trình theo kích thước dữ liệu 16](#_Toc531173236)

[3. Nhậnxét đánh giá khả năng ứng dụng chương trình 16](#_Toc531173237)

[**IV.** **Hướng dẫn sử dụng chương trình** 17](#_Toc531173238)

[**V.** **Listing chương trình nguồn** 17](#_Toc531173239)

[**VI.** **Tài liệu tham khảo** 17](#_Toc531173240)

1. **Các vấn đề lý thuyết**
2. Đặt vấn đề

Hiện nay trên thế giới dữ liệu tồn tại ở nhiều dạng khác nhau, trong đó dữ liệu dạng text chiếm một khối lượng không nhỏ. Việc tìm kiếm trong số những tài liệu này trong một khoảng thời gian nhanh chóng là một công việc có tính quan trọng bậc nhất. Việc so khớp chuỗi chỉ ra những vị trí trong văn bản xuất hiện chuỗi đầu vào có sẵn.

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

Ứng dụng tìm kiếm này cho phép người dùng lọc thông tin tìm kiếm nhanh nhất thông qua việc tìm kiếm những từ khóa (keyword). Ngoài ra, việc so sánh chuỗi còn có ứng dụng trong ngành tin sinh học như so khớp gen, xác định vị trí của gen trên ADN, ARM, từ đó xác định được sự phân chia tính trạng khi phân ly. Điều này có ý nghĩa to lớn trong việc thiết lập bản đồ gen người, tiến tới điều trị được những căn bệnh nan y do biến đổi gen gây ra.

Do những ý nghĩa và vai trò to lớn của đối sánh mẫu, rất nhiều thuật toán được con người nghĩ ra, triển khai và áp dụng vào công việc so khớp chuỗi. Trong báo cáo là một số những thuật toán tìm kiếm xâu mẫu bao gồm: Brute Force, Knuth Morris Pratt, Boyer – Moore.

1. Khái niệm.

String Matching hay Pattern Matching là việc tìm một hoặc nhiều xâu mẫu (Pattern) xuất hiện trong một văn bản (có thể là rất dài).

* Ký hiệu xâu mẫu hay xâu cần tìm là x =(x0, x1,..,xm-1) có độ dài m.
* Văn bản y=(y0, y1,..,yn-1) có độ dài n.

Trong Project1 này cả hai xâu được xây dựng từ một tập hữu hạn các ký tự Alphabet hoặc số (Do việc sinh ngẫu nhiên ra văn bản sử dụng tất cả các ký tự ASCII sẽ gây ra văn bản vô nghĩa) ký hiệu là S với kích cỡ là . Như vậy một xâu nhị phân có độ dài n ứng dụng trong mật mã học cũng được xem là một mẫu. Một chuỗi các ký tự ABD độ dài m biểu diễn các chuỗi AND cũng là một mẫu.

***Input:*** Xâu mẫu x =(x0, x1,..,xm-1) độ dài m. Văn bản bản y=(y0, y1,..,yn-1) độ dài n.

***Output:*** Tất cả vị trí xuất hiện của x trong y.

1. Phân loại.

Ta có thể phân loại các thuật toán tìm kiếm mẫu thành các lớp:

* ***Tìm kiếm mẫu từ bên trái qua bên phải:*** Harrison Algorithm, Karp-Rabin Algorithm, Morris-Pratt Algorithm, Knuth- Morris-Pratt Algorithm, Forward Dawg Matching algorithm , Apostolico-Crochemore algorithm, Naive algorithm.
* ***Tìm kiếm mẫu từ bên phải qua bên trái:*** Boyer-Moore Algorithm , Turbo BM Algorithm, Colussi Algorithm, Sunday Algorithm, Reverse Factorand Algorithm, Turbo Reverse Factor, Zhu and Takaoka and Berry-Ravindran Algorithms.
* ***Tìm*** ***kiếm mẫu từ một vị trí cụ thể:*** Two Way Algorithm, Colussi Algorithm , Galil-Giancarlo Algorithm, Sunday's Optimal Mismatch Algorithm, Maximal Shift Algorithm, Skip Search, KMP Skip Search and Alpha Skip Search Algorithms.
* ***Tìm kiếm mẫu từ bất kỳ:*** Horspool Algorithm, Boyer-Moore Algorithm, Smith Algorithm , Raita Algorithm.

1. Tìm hiểu một số thuật toán tìm kiếm xâu mẫu cơ bản.
2. ***Brute Force Algorithm***

**Đặc điểm:**

* Là thuật toán tìm kiếm mẫu từ trái qua phải
* Không có pha chuẩn bị (tiền xử lí)
* Bộ nhớ cần dùng cố định
* Luôn luôn dịch 1 bước sang phải
* Việc so sánh có thể phải dùng trong các trường hợp
* Độ phức tạp pha thực thi là *O(mn)*

**Thuật toán:**

Thuật toán Brute Force bao gồm kiểm tra 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. Tức là lần lượt xét từng vị trí i trong xâu ký tự gốc từ 0 đến n-m, so sánh y[i…(i+m-1)] với x[0…m-1] bằng cách xét từng cặp ký tự một và đưa ra kết quả tìm kiếm.

**Source code:**

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

**int** i, j;

*/\* Searching \*/*

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

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

**if** (i >= m)

OUTPUT(j);

}

}

1. ***Knuth- Morris-Pratt Algorithm***

Thuật toán được phát minh năm 1977 bởi hai giáo sư của ĐH Stanford, Hoa Kỳ (một trong số ít các trường đại học xếp hàng số một về khoa học máy tính trên thế giới, cùng với trường MIT, CMU cũng của Hoa Kỳ và Cambrige của Anh) là Donal Knuth và Vaughan Ronald Pratt. Giáo sư Knuth (giải Turing năm 1971) còn rất nổi tiếng với cuốn sách “Nghệ thuật lập trình” (The Art of Computer Programming), hiện nay đã có đến tập 6. Thuật toán này còn có tên là KMP, tức là lấy tên viết của ba người đồng phát minh ra nó, chữ “M” là chỉ giáo sư J.H.Morris, cũng là một giáo sư rất nổi tiếng trong ngành khoa học máy tính.

**Đặc điểm:**

* Là thuật toán tìm kiếm mẫu từ trái qua phải
* Pha chuẩn bị có độ phức tạp về không gian và thời gian là *O(m)*
* Pha tìm kiếm có độ phức tạp về thời gian *O(m+n)*
* Knuth Morris Pratt là một phân tích chặt chẽ của thuật toán Morris pratt.
* Bảng kmpNext[] giúp loại bỏ việc so sánh lại những kí tự đã so sánh trước đó trong phương pháp thông thường.
* Thực hiện: Dò từ trái sang phải cho tới khi gặp vị trí x và y không

giống nhau. Giả sử: x[i] != [j], xem giá trị kmpNext[i] tương ứng.

* Nếu kmpNext[i] = -1 thì dịch chuỗi x lên một bước.
* Nếu kmpNext[i] >= 0 thì ta di chuyển x[m] trùng với i.

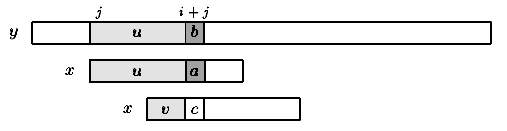
**Thuật toán:**

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

Thuật toán Knuth-Morris-Pratt là thuật toán có độ phức tạp tuyến tính đầu tiên được phát hiện ra, nó dựa trên thuật toán Brute force với ý tưởng lợi dụng lại những thông tin của lần thử trước cho lần sau. Trong thuật toán Brute force vì chỉ dịch cửa sổ đi một ký tự nên có đến m-1 ký tự của cửa sổ mới là những ký tự của cửa sổ vừa xét. Trong đó có thể có rất nhiều ký tự đã được so sánh giống với mẫu và bây giờ lại nằm trên cửa sổ mới nhưng được dịch đi về vị trí so sánh với mẫu. Việc xử lý những ký tự này có thể được tính toán trước rồi lưu lại kết quả. Nhờ đó lần thử sau có thể dịch đi được nhiều hơn một ký tự, và giảm số ký tự phải so sánh lại.

Xét lần thử tại vị trí j, khi đó cửa sổ đang xét bao gồm các ký tự y[j...j+m-1], giả sử sự khác biệt đầu tiên xảy ra giữa hai ký tự x[i] và y[j+i-1].

Khi đó x[1...i] = y[j...i+j-1] = u và a = x[i] ≠ y[i+j] = b. Với trường hợp này, dịch cửa sổ phải thỏa mãn v là phần đầu của xâu x khớp với phần đuôi của xâu u trên văn bản. Hơn nữa ký tự c ở ngay sau v trên mẫu phải khác với ký tự a. Trong những đoạn như v thoả mãn các tính chất trên ta chỉ quan tâm đến đoạn có độ dài lớn nhất.



Thuật toán Knuth-Morris-Prath sử dụng mảng kmpNext để lưu trữ độ dài lớn nhất của xâu v trong trường hợp xâu u=x[1...i-1]. Mảng này có thể tính trước với chi phí về thời gian là O(m).

Thuật toán này có chi phí về thời gian là O(m+n) với nhiều nhất là 2n-1 lần số lần so sánh kí tự trong quá trình tìm kiếm.

**Source code:**

**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];

}

}

}

1. ***Boyer – Moore***

Boyer-Moore là thuật toán tìm kiếp xâu mẫu với hiệu quả cao được công bố năm 1977 bởi Robert S. Boyer và J Strother Moore và hiện nay nó có nhiều biến thể.

**Đặc điểm:**

Thuật toán Boyer Moore là thuật toán tìm kiếm chuỗi rất có hiệu quả trong thực tiễn, các dạng khác nhau của thuật toán này thường được cài đặt trong các chương trình soạn thảo văn bản.

**Đặc điểm:**

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

**Thuật toán:**

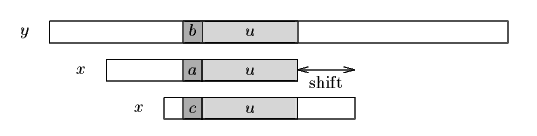
Trong cài đặt ta dùng mảng bmGs để lưu cách dịch 1, mảng bmBc để lưu phép dịch thứ 2 (ký tự không khớp).

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

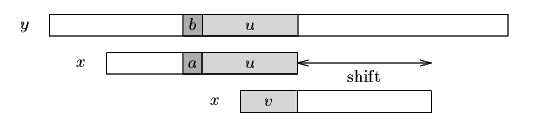
Trong trường hợp mis-match (hoặc là trường hợp đã tìm được 01 đoạn khớp với mẫu), nó sẽ dùng 2 hàm được tính toán trước để dịch cửa sổ sang bên phải. Hai hàm dịch chuyển này được gọi là **good-suffix shift** ( còn được biết với cái tên phép dịch chuyển khớp) và **bad-character shift** (hay phép dịch chuyển xuất hiện).

Đối với mẫu x[0...m-1], ta dùng 01 biến số chỉ số i chạy từ cuối về đầu, đối với chuỗi y[0...n-1], ta dùng 01 biến j để chốt ở phía đầu. Giả sử trong quá trình so sánh, ta gặp 1 mis-match tại vị trí x[i] = a của mẫu và y[i+j] = b trong khi đang thử khớp tại vị trí j. Khi đó, x[i+1...m-1] = y[j+i+1...j+m-1] = u và x[i] ≠ y[i+j]. Đối với từng trường hợp, 2 hàm trên sẽ thực hiện việc di chuyển như sau:

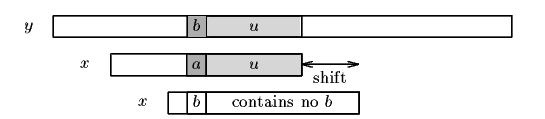
Phép dịch chuyển good-suffix shift sẽ dịch cửa sổ sang bên phải cho đến khi gặp 1 kí tự khác với x[i] trong trường hợp đoạn u lại xuất hiện trong x.



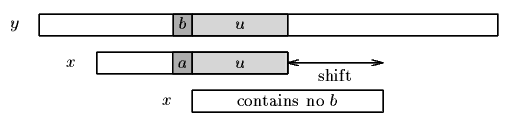
Nếu đoạn u không xuất hiện lại trong x, mà chỉ có 1 phần cuối (suffix) của u khớp với phần đầu (prefix) của x, thì ta sẽ dịch 1 đoạn sao cho phần suffix dài nhất v của y[j+i+1...j+m-1] khớp với prefix của x.



Phép dịch chuyển bad-character shift sẽ khớp kí tự y[i+j] với 1 kí tự (bên trái nhất) trong đoạn x[0...m-2].



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



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

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

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

Với 1 ≤ i < m, suff[i] = max{k: x[i-k+1...i] = x[m-k...m-1]}

Hàm bad-character shift được lưu trong bảng bmBc có kích thước *σ*. Cho c trong ∑: bmBc[c] = min{i: 1 ≤ i <m-1 và x[m-1-i] = c} nếu c xuất hiện trong x, m ngược lại. Bảng bmGs và bmBc được tính toán trong thời gian *O(m+σ)* trước khi thực hiện tìm kiếm và cần 1 không gian phụ là *O(m+σ)*. Giai đoạn tìm kiếm có độ phức tạp thời gian bậc 2 nhưng lại chỉ có 3n phép so sánh khi tìm kiếm 1 chuỗi không có chu kì. Đối với việc tìm kiếm trong một khối lượng lớn các chữ cái, thuật toán có thể thực hiện với một tốc độ rất nhanh. Khi tìm kiếm chuỗi am-1 trong bn chuỗi, thuật toán chỉ sử dụng O(m/n) phép so sánh, là chi phí thấp nhất của các thuật toán tìm kiếm hiện đại có thể đạt được [2]

**Source code:**

**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);

}

}

1. **Hệ thống chương trình giải bài toán**
2. ***Định nghĩa:***

RANDCHAR: Số ký tự tối đa của văn bản y khi sinh ngẫu nhiên

XSIZE: Số ký tự tối đa của xâu mẫu x

YSIZE: Số ký tự tối đa của văn bản y

ASIZE: Giá trị tối đa của ký tự xuất hiện trong x và y tương ứng trong bảng mã ASCII

1. ***Các biến toàn cục:***

**int** kq[YSIZE]: *Mảng lưu kết quả.*

**int** k=0: *Biến đếm chỉ số của mảng kết quả*

**float** timeBF,timeKMP,timeBM: *Thời gian thực hiện giải thuật*

**int** showkqtg = 1: *Lưu tuỳ chọn Xem / Không xem kết quả trung gian.*

1. Hàm **main()** Chương trình gồm các bước:

* Cấp phát động bộ nhớ cho 2 mảng có địa chỉ trỏ bởi 2 con trỏ **char** x, y
* Hiện menu tương ứng các tuỳ chọn:

1. Nhập dữ liệu từ bàn phím
2. Nhập dữ liệu từ file (Nhập tên file)
3. Tạo dữ liệu ngẫu nhiên
4. Xem dữ liệu hiển thị lên màn hình
5. Ghi dữ liệu ra file (Nhập tên file)
6. Tìm kiếm bằng giải thuật Brute Force
7. Tìm kiếm bằng giải thuật Knuth – Morris – Pratt
8. Tìm kiếm bằng giải thuật Boyer – Moore
9. Xem kết quả trên màn hình
10. Ghi kết quả ra file (Nhập tên file)
11. Tuỳ chọn Xem / Không xem kết quả trung gian (Chạy lại mục 6,7,8 để xem các kết quả trung gian)
12. Thoát

* Rẽ nhánh thực hiện chức năng mà người dùng chọn qua hàm switch(ch)

**Nhập vào ký tự *ch* để chọn các tuỳ chọn trên menu:**

Nếu ch = 1 thì gọi hàm NhapTuBanPhim(y,x);

Nếu ch = 2 thì gọi hàm NhapTuFile(y,x);

Nếu ch = 3 thì gọi hàm TaoDuLieuNgauNhien(y,x);

Nếu ch = 4 thì gọi hàm HienKetQua(y,x);

Nếu ch = 5 thì gọi hàm LuuDuLieuDenFile(y,x);

Nếu ch = 6 thì gọi hàm BF(y,x);

Nếu ch = 7 thì gọi hàm KMP(y,x);

Nếu ch = 8 thì gọi hàm BM(y,x);

Nếu ch = 9 thì gọi hàm XemKetQua(kq);

Nếu ch = 10 thì gọi hàm LuuKetQuaRaFile(kq);

Nếu ch = 11 thì thay đổi giá trị của biến showkqtg thành 0 hoặc 1

Nếu ch = 0 thì in ra: “Tam biet”

1. ***Các module***
2. Module NhapTuBanPhim()

**INPUT: Địa chỉ 2 chuỗi x, y trong bộ nhớ**

**OUTPUT: Không**

**Mô tả:** HàmNhapTuBanPhim()cho phép nhập từ bàn phím giá trị của 2 chuỗi có địa chỉ nhận từ đầu vào.

1. Module NhapTuFile()

**INPUT:** Địa chỉ 2 chuỗi x, y trong bộ nhớ

**OUTPUT: Không**

**Mô tả:** HàmNhapTuFile()cho phép nhập từ bàn phím tên của file. Sau đó nội dung file sẽ được ghi vào giá trị của chuỗi y có địa chỉ nhận từ đầu vào. Giá trị của chuỗi x được nhập từ bàn phím.

1. Module Random()

**INPUT:** 2 số nguyên a, b

**OUTPUT: Số nguyên ngẫu nhiên có giá trị nằm giữa a và b**

**Mô tả:** HàmRandom() sinh ngẫu nhiên số nguyên có giá trị nằm giữa 2 số đầu vào

1. Module Randd()

**INPUT:** Không

**OUTPUT:** 1 ký tự số hoặc chữ hoa / thường bất kỳ

**Mô tả:** HàmRandd() sinh ngẫu nhiên một ký tự hoa thường bất kỳ trong bảng mã ASCII để phù hợp với mục đích sinh chuỗi ngẫu nhiên chỉ chứa các ký tự số hoặc chữ tránh chuỗi “kỳ dị”

1. Module LuuDuLieuDenFile()

**INPUT:** Không

**OUTPUT:** 1 ký tự số hoặc chữ hoa / thường bất kỳ

**Mô tả:** HàmLuuDuLieuDenFile()cho phép nhập từ bàn phím tên của file. Sau đó nội dung chuỗi y có địa chỉ nhận từ đầu vào sẽ được ghi vào file.

1. Module BF()

**INPUT:** Con trỏ trỏ đến 2 chuỗi y, x

**OUTPUT:** Mảng kq[] chứa vị trị xuất hiện của x trong y và số thực timeBF là thời gian thực hiện tính toán

**Mô tả:** HàmBF()sử dụng thuật toán Brute Force tìm vị trí xuất hiện của chuỗi x trong đoạn y.

1. Module preKmp()

**INPUT:** Con trỏ trỏ chuỗi x, số nguyên m tương ứng là độ dài chuỗi x, mảng số nguyên kmpNext[]

**OUTPUT:** Mảng kmpNext[]

**Mô tả:** hàm preKmp() trả về mảng kmpNext[] có giá trị với kmpNext[i] là độ dài tagged border của x[0..i-1] đứng trước ký tự c khác với xi hoặc *-1* nếu không có tagged border nào tồn tại

1. Module KMP()

**INPUT:** Con trỏ trỏ tới 2 chuỗi y, x

**OUTPUT:** Mảng kq[] chứa vị trị xuất hiện của x trong y và số thực timeKMP là thời gian thực hiện tính toán

**Mô tả:** HàmKMP()sử dụng thuật toán KMP sử dụng mảng kmpNext[] tìm vị trí xuất hiện của chuỗi x trong đoạn y.

1. Module preBmBc()

**INPUT:** Con trỏ trỏ chuỗi x, số nguyên m tương ứng là độ dài chuỗi x, mảng số nguyên bmBc[]

**OUTPUT:** Mảng bmBc[]

**Mô tả:** hàm preBmBc() trả về mảng bmBc[] lưu giá trị của các chữ cái khác nhau tính từ phải sang trái trong x

1. Module suffixes()

**INPUT:** Con trỏ trỏ chuỗi x, số nguyên m tương ứng là độ dài chuỗi x, mảng số nguyên bmBc[]

**OUTPUT:** Mảng suff[]

**Mô tả:** hàm suffixes() trả về mảng suff[] với suff[i] là độ dài của chuỗi con (bắt đầu từ vị trí i) lặp lại từ hậu số của chuỗi x khi duyệt từ phải sang trái chuỗi x

1. Module suffixes()

**INPUT:** Con trỏ trỏ chuỗi x, số nguyên m tương ứng là độ dài chuỗi x, mảng số nguyên bmBc[]

**OUTPUT:** Mảng suff[]

**Mô tả:** hàm suffixes() trả về mảng suff[] với suff[i] là độ dài của chuỗi con (bắt đầu từ vị trí i

1. **Kết quả tính toán**
2. Bộ dữ liệu test thử chương trình:

x = "GCAGAGAG";

y = "GCATCGCAGAGAGTATACAGTACG";

Suy ra m = 8; n=24

1. **Brute Force**

|  |  |  |  |
| --- | --- | --- | --- |
| **Vị trí so sánh y** | **Số ký tự trùng** | **Kết quả** | **Bước dịch chuyển y** |
| **0** | **3** | **mismatch tại x[3]** | **1** |
| **1** | **0** | **mismatch tại x[0]** | **1** |
| **2** | **0** | **mismatch tại x[0]** | **1** |
| **3** | **0** | **mismatch tại x[0]** | **1** |
| **4** | **0** | **mismatch tại x[0]** | **1** |
| **5** | **8** | **Match** | **1** |
| **6** | **0** | **mismatch tại x[0]** | **1** |
| **7** | **0** | **mismatch tại x[0]** | **1** |
| **8** | **1** | **mismatch tại x[1]** | **1** |
| **9** | **0** | **mismatch tại x[0]** | **1** |
| **10** | **1** | **mismatch tại x[1]** | **1** |
| **11** | **0** | **mismatch tại x[0]** | **1** |
| **12** | **1** | **mismatch tại x[1]** | **1** |
| **13** | **0** | **mismatch tại x[0]** | **1** |
| **14** | **0** | **mismatch tại x[0]** | **1** |
| **15** | **0** | **mismatch tại x[0]** | **1** |
| **16** | **0** | **mismatch tại x[0]** | **1** |

1. **Knuth Morris Pratt**

* ***Bảng kmpNext[]:***

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| **x** | **G** | **C** | **A** | **G** | **A** | **G** | **A** | **G** |
| **kmpNext[]** | **-1** | **0** | **0** | **-1** | **1** | **-1** | **1** | **-1** |

* ***Các bước so sánh:***

|  |  |  |  |
| --- | --- | --- | --- |
| **Vị trí so sánh y** | **Số ký tự trùng** | **Kết quả** | **Bước dịch chuyển** |
| **0** | **3** | **mismatch tại x[3]** | **4** |
| **4** | **0** | **mismatch tại x[0]** | **1** |
| **5** | **8** | **Match** | **7** |
| **12** | **0** | **mismatch tại x[0]** | **1** |
| **13** | **0** | **mismatch tại x[0]** | **1** |
| **14** | **0** | **mismatch tại x[0]** | **1** |
| **15** | **0** | **mismatch tại x[0]** | **1** |

1. **Boyer – Moore**

* ***Bảng bmBc[]:***

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| **x** | **G** | **C** | **A** | **G** | **A** | **G** | **A** | **G** |
| **bmBc[]** | **2** | **6** | **1** | **2** | **1** | **2** | **1** | **2** |
| **suff[]** | **1** | **0** | **0** | **2** | **0** | **4** | **0** | **8** |
| **bmGs[]** | **7** | **7** | **7** | **2** | **7** | **4** | **7** | **1** |

* ***Các bước so sánh:***

|  |  |  |  |
| --- | --- | --- | --- |
| **Vị trí so sánh y** | **Số ký tự trùng** | **Kết quả** | **Bước dịch chuyển** |
| **0** | **0** | **mismatch tại x[7]** | **1** |
| **1** | **2** | **mismatch tại x[5]** | **4** |
| **5** | **8** | **Match** | **7** |
| **12** | **2** | **mismatch tại x[5]** | **4** |
| **16** | **0** | **mismatch tại x[6]** | **7** |

1. Thống kê thời gian tính của chương trình theo kích thước dữ liệu

***Máy tính thực hiện:***

CPU: Intel® Xeon® CPU E3-1270 V2 @ 3.50 GHz

RAM: 8GB

Ổ cứng: 128 GB SSD

***Lưu ý:*** RANDCHAR tối đa máy tính có thể sinh là 10000000

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **m** | **N** | **Brute Force** | **Knuth Morris Pratt** | **Boyer – Moore** |
| 8 | 24 | 0 | 0 | 0 |
| 9 | 118 | 0 | 0 | 0 |
| 3 | 50000 | 0.001 | 0.001 | 0.000 |
| 3 | 10 | 0.001 | 0.001 | 0.001 |
| 3 | 1000000 | 0.0070000000000000001 | 0.0050000000000000001 | 0.0040000000000000001 |
| 3 | 10000000 | 0.070000000000000007 | 0.041000000000000002 | 0.034000000000000002 |

1. Nhậnxét đánh giá khả năng ứng dụng chương trình

Chương trình đã triển khai thành công các thuật toán trong đè tài nghiên cứu và có thể tìm kiếm các xâu mẫu hoạt động tốt đạt được mục đích đề ra và cho ra kết quả tìm kiếm thấy vị trí chuỗi x trong văn bản y với khoảng thời gian nhỏ. Tốc độ tìm kiếm tăng dần theo thứ tự Brute Force – KMP – Boyer-Moore.

1. **Hướng dẫn sử dụng chương trình**

* Hệ điều hành: Windows 7 - 8 - 10
* Khi chạy chương trình sẽ hiện menu cho 12 tuỳ chọn dược đánh số 1..11 và 0.
* Để nhập dữ liệu từ bàn phím nhập giá trị 1
* Để nhập dữ liệu từ file nhập giá trị 2
* Để tạo dữ liệu ngẫu nhiên nhập giá trị 3. Để thay đổi độ dài văn bản y được tạo ngẫu nhiên thay đổi giá trị RANDCHAR ở bộ tiền xử lí
* Để hiển thị giá trị x, y ra màn hình nhập giá trị 4
* Để ghi dữ liệu x, y ra file nhập giá trị 5
* Sau khi đã có x, y để thực hiện tìm kiếm vị trí x trong y sử dụng giải thuật Brute Force nhập giá trị 6
* Sau khi đã có x, y để thực hiện tìm kiếm vị trí x trong y sử dụng giải thuật Knuth Morris Pratt nhập giá trị 7
* Sau khi đã có x, y để thực hiện tìm kiếm vị trí x trong y sử dụng giải thuật Boyer – Moore nhập giá trị 8
* Để hiển thị kết quả tìm kiếm nhập giá trị 9
* Để ghi kết quả ra file nhập giá trị 10
* Để thay đổi tuỳ chọn Xem / Không xem kết quả trung gian trong quá trình tìm kiếm nhập giá trị 11. Sau đó chọn 6-7-8 để thực hiện các thuật toán có hiển thị kết quả trung gian.
* Để kết thúc chương trình chọn 0.

1. **Listing chương trình nguồn**

* Tài liệu tham khảo GVHD Nguyễn Đức Nghĩa.
* Brute Force: *http://www-igm.univ-mlv.fr/~lecroq/string/node3.html#SECTION0030*
* Knuth Morris Pratt*:* [*http://www-igm.univ-mlv.fr/~lecroq/string/node8.html#SECTION0080*](http://www-igm.univ-mlv.fr/~lecroq/string/node8.html#SECTION0080)
* Boyer – Moore*:* [*http://www-igm.univ-mlv.fr/~lecroq/string/node14.html#SECTION00140*](http://www-igm.univ-mlv.fr/~lecroq/string/node14.html#SECTION00140)

1. **Tài liệu tham khảo**

* Ngân Hoàng Mỹ Linh, *“Bài toán đối sánh mẫu sử dụng giải thuật di truyền”,* Luận văn thạc sĩ khoa học máy tính, Đại học CNTT&TT Đại học Thái Nguyên
* Website, [*https://www.geeksforgeeks.org/kmp-algorithm-for-pattern-searching/*](https://www.geeksforgeeks.org/kmp-algorithm-for-pattern-searching/) , Knuth Morris Pratt, Boyer – Moore.
* Website*,* [*http://www-igm.univ-mlv.fr/~lecroq/string/node1.html*](http://www-igm.univ-mlv.fr/~lecroq/string/node1.html), Brute Force, Knuth Morris Pratt, Boyer – Moore.
* Website*,* [*https://dzone.com/articles/algorithm-week-boyer-moore*](https://dzone.com/articles/algorithm-week-boyer-moore), Boyer – Moore.