

**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT**

**THÀNH PHỐ HỒ CHÍ MINH**

**KHOA: CÔNG NGHỆ THÔNG TIN**

**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT**

**THÀNH PHỐ HỒ CHÍ MINH**

**KHOA: CHÍNH TRỊ VÀ LUẬT**

Logo, icon

Description automatically generated

**Sinh viên thực hiện**

|  |  |  |
| --- | --- | --- |
| STT | Họ và tên | MSSV |
| 1 | Nguyễn Thái Văn | 21110939 |
| 2 | Huỳnh Nhật Nam | 21110903 |
| 3 | Nguyễn Trọng Phúc | 21110915 |

**FAST PATTERN MATCHING IN STRINGS**

Môn học: Cấu trúc dữ liệu và giải thuật

**TIỂU LUẬN cuối kỳ**

**môn học: TƯ TƯỞNG HỒ CHÍ MINH**

*Tp. Hồ Chí Minh, tháng 12 năm 2022*

*Tp. Hồ Chí Minh, tháng 11 năm 2022*

*GIỚI THIỆU:*

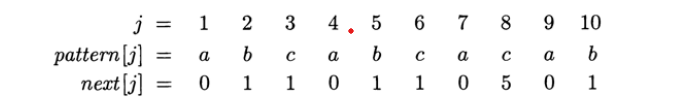
Trong lập trình, chắc hẳn ít nhiều chúng ta cũng nghe về vấn đề xét sự so khớp chuỗi, cụ thể là chúng ta xem thử chuỗi này có đang nằm trong chuỗi kia hay không. Và khi gặp những vấn đề như thế này, những bạn sinh viên CNTT ban đầu đều suy nghĩ ra một phương pháp đó chính là chúng ta sẽ trượt chuỗi cần xét trên chuỗi kia, và mỗi lần như thế chúng ta sẽ so sánh từng kí tự của hai chuỗi với nhau, nếu như chuỗi mẫu hoàn toàn nằm trong chuỗi kia thì chúng ta return true và kết thúc chương trình, còn nếu tại một vị trí mà kí tự của 2 chuỗi khác nhau thì chúng ta sẽ ngừng vòng lặp và tiến hành tịnh tiến chuỗi mẫu lên thêm 1 lần nữa cho đến khi nó khớp hoặc đến cuối vị trí của chuỗi kia. Tuy nhiên cách làm này thực sự không hiệu quả với những chuỗi có kích thước lớn, vì độ phức tạp của cách làm này là O(n \* m) trong đó n là độ dài chuỗi mẫu và m là độ dài chuỗi còn lại. Vì vậy trong chapter này chúng ta sẽ cùng nghiên cứu phương pháp để giảm thiểu độ phức tạp của thuật toán so khớp chuỗi này cụ thể độ phức tạp của thuật toán trong Chapter này sẽ là O(n + m).

*THUẬT TOÁN:*

Trước tiên, ta gọi+ pattern là chuỗi cần đi so khớp , gọi j là index của các kí tự chuỗi pattern

+ text là chuỗi mà để chuỗi pattern đi so khớp, gọi k là index của các kí tự trong chuỗi text

Trước khi chúng ta tiến hành so khớp chuỗi, chúng ta sẽ đi tạo ra mảng next trước, mảng next là mảng mà tại vị trí j của chuỗi next[j] chính là vị trí cần so sánh tiếp theo nếu như tại vị trí j này next[j] != text[k].

Bản chất của mảng next[j] này đó chính là tại một vị trí j mà pattern[j] != text[k] thì chúng ta không cần phải dịch pattern lên thêm 1 đơn vị để so sánh lại mà chúng ta có thể dịch pattern lên j – next[j] đơn vị, nhờ vậy mà độ phức tạp của thuật toán được giảm đi đáng kể. Đây chính là một ví dụ mẫu về mảng next[j].

Và cách để tính toán mảng next này chỉ tốn O(m) bước với m là độ dài của chuỗi pattern và trước khi đi tính mảng next chúng ta sẽ cùng nhau tạo ra mảng f, mảng f ở đây chính là tiền đề để tạo ra mảng next.

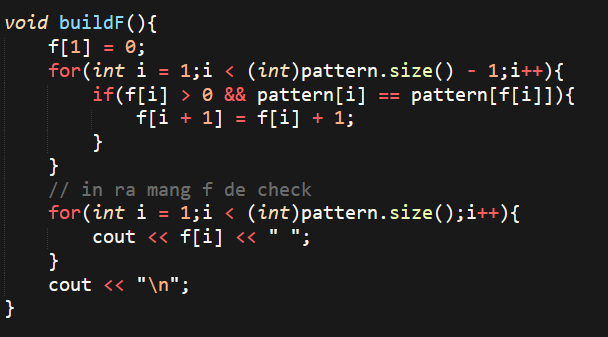
Bây giờ, mục đích chính của mảng next đó chính là lưu lại vị trí i lớn nhất mà chuỗi từ pattern[1] …pattern[i – 1] đã khớp với text để mà khi đó chúng ta sẽ so sánh text[k] với pattern[next[j]] thôi và hiệu quả là chúng ta đã bỏ qua i – 1 vị trí. Để dễ dàng hơn trong việc tạo ra mảng next, chúng ta sẽ đi tạo ra mảng f, mảng f ở đây chính là tại một vị trí j bất kì f[j] = i trong đó i là lớn nhất và nhỏ hơn j sao cho



Và chúng ta thấy điều kiện này hoàn toàn đúng với i bằng 1, vì vậy chúng ta sẽ có f[j] >= 1 với mọi j > 1. Theo quy ước, chúng ta sẽ để f[1] = 0. Và giờ chúng ta sẽ đi tạo ra mảng f



*CODE C++*

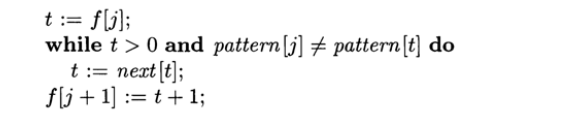


Đây chính là một vấn đề đã từng xuất hiện ở trước đây cụ thể đó chính là ta tính j lớn nhất nhỏ hơn k sao cho

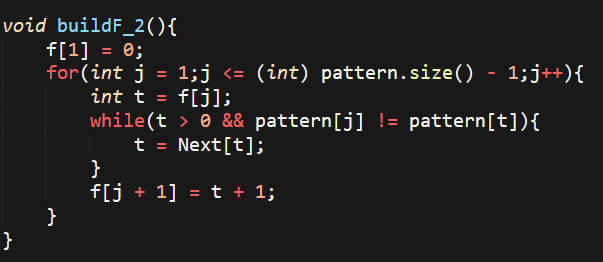
pattern[1] … pattern[j – 1] = text[k – j + 1] … text[k – 1]

vì vậy chúng ta sẽ sử dụng nó cho vấn đề hiện tại. Và khi chúng ta thực hiện thuật toán trên, chúng ta đã có thể tính toán được mảng f rồi.

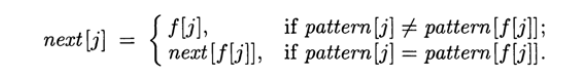
Và em bổ sung thêm một chương trình để tính mảng f, chương trình này sẽ tính f[j + 1] và giả sử rằng f[j] và next[1] …next[j – 1] đã được tính. Thuật toán:

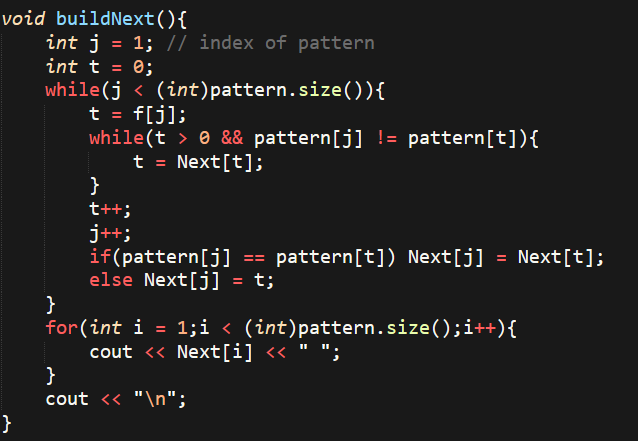


*CODE C++*



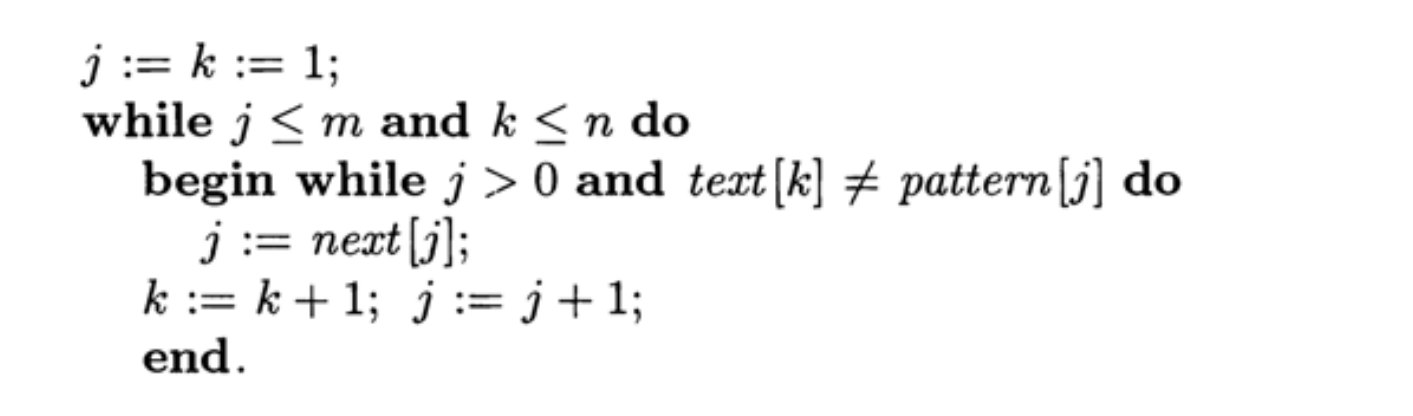
Và sau khi chúng ta đã tính toán được mảng f, chúng ta sẽ cùng nhau đi tạo mảng next, chúng ta có công thức sau:



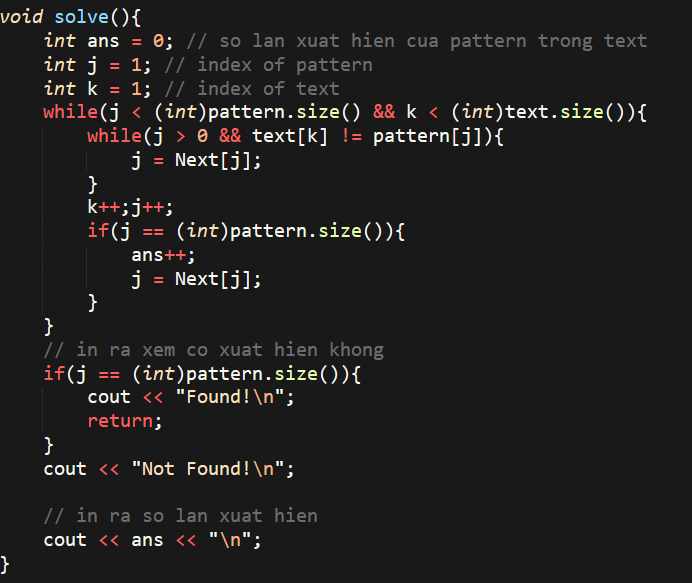
Thuật toán xây dựng mảng next: *CODE C++*

Độ phức tạp thuật toán: O(m) với m là độ dài của chuỗi pattern

Và sau khi chúng ta đã có mảng next, việc duyệt so khớp sẽ diễn ra một cách dễ dàng:



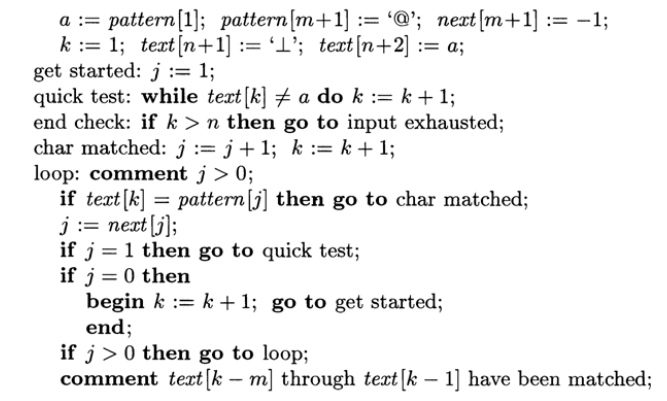
Đây là thuật toán trong bài báo, và thuật toán này chỉ để xét xem chuỗi pattern có xuất hiện trong chuỗi text không hay thôi, và em sẽ mở rộng ra để xét thêm chuỗi pattern xuất hiện trong chuỗi text bao nhiêu lần. Thuật toán:

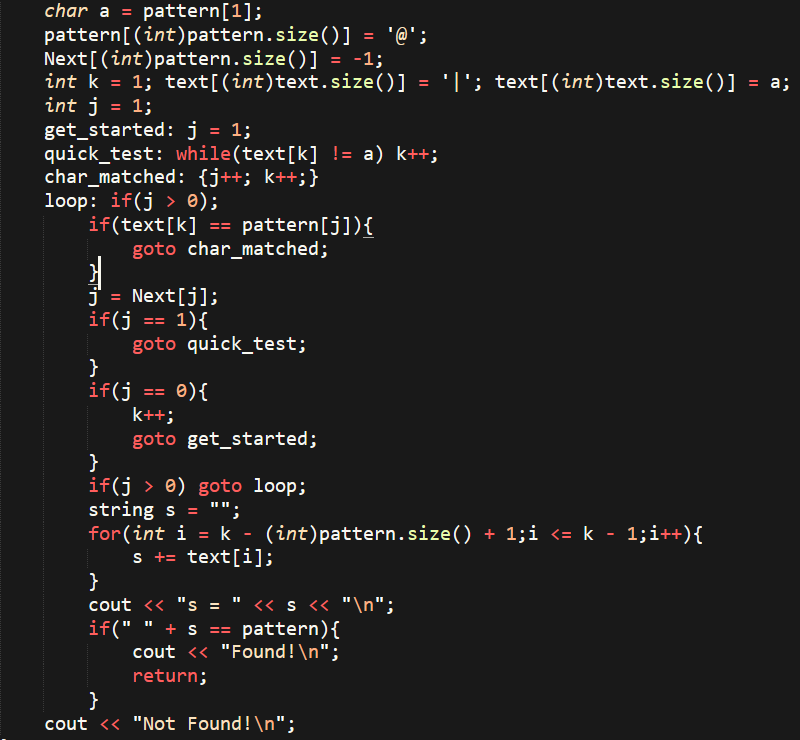
*CODE C++*

Độ phức tạp của thuật toán: O(m + n) với m là độ dài của chuỗi pattern và n là độ dài của chuỗi text.

*Cải tiến độ hiệu quả:*

Chúng em vừa mới trình bày thuật toán so khớp chuỗi ở trên, nhưng hình thức này dường như không hiệu quả lắm. Và bây giờ chúng ta sẽ đi tìm hiểu một thuật toán mới được triển khai tốt hơn, và đây chính là thuật toán:



*CODE C++*

*ỨNG DỤNG CỦA THUẬT TOÁN*

1. *Tìm kiếm văn bản:* Thuật toán Knuth-Morris-Pratt (KMP) có thể được sử dụng để tìm kiếm một mẫu trong chuỗi văn bản một cách hiệu quả bằng cách tránh phải quay lại và khớp lại các ký tự đã khớp trước đó.

2*. Nén dữ liệu:* Thuật toán KMP có thể được sử dụng trong các thuật toán nén dữ liệu như thuật toán LZ77 và LZ78 để tìm các mẫu lặp lại trong một chuỗi dữ liệu một cách hiệu quả.

3. *Truyền dữ liệu:* Thuật toán KMP có thể được sử dụng trong các giao thức sửa lỗi để truyền dữ liệu nhằm xác định và sửa lỗi hiệu quả trong dữ liệu được truyền.

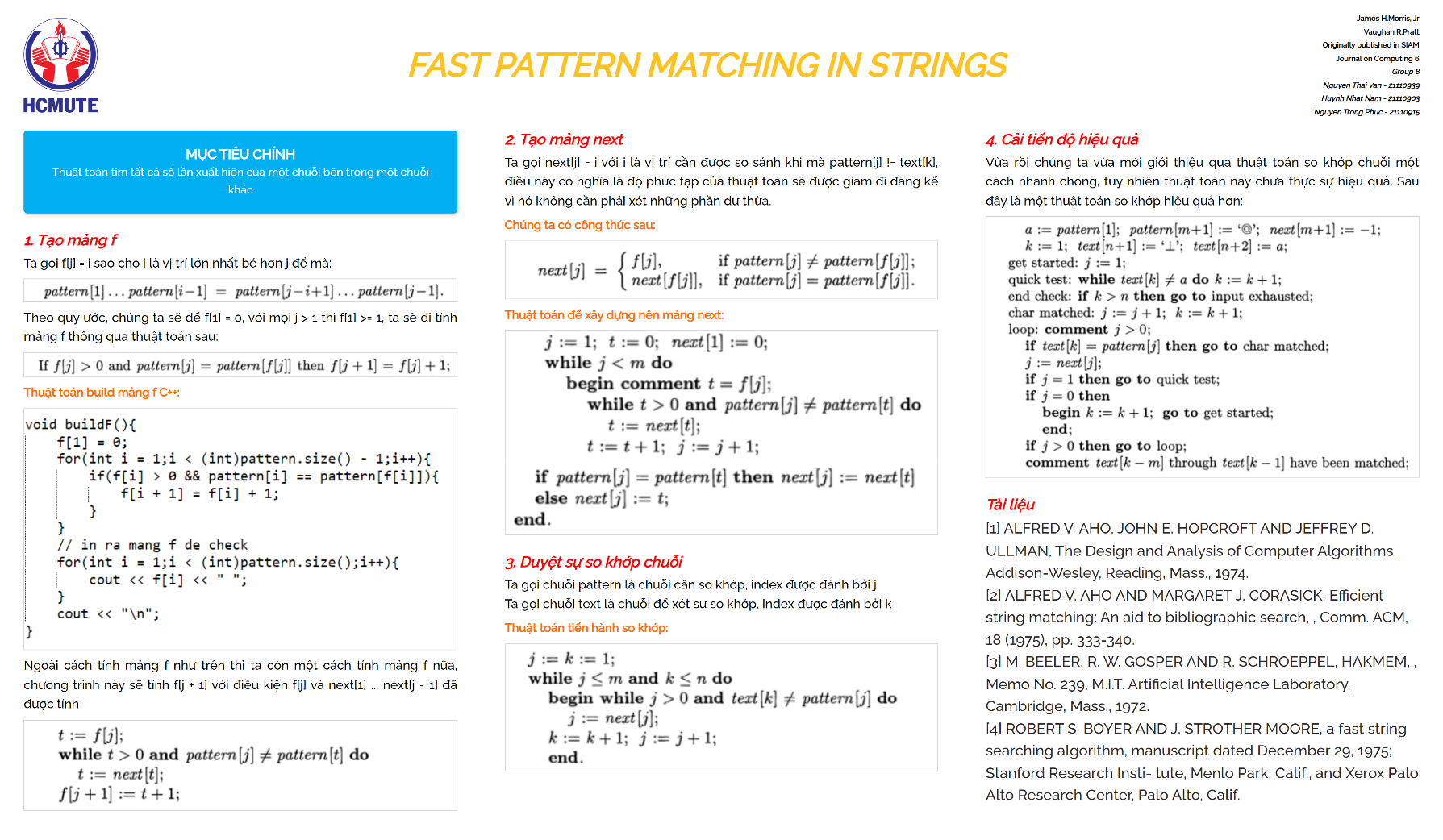
4. *Phân tích bộ gen:* Thuật toán KMP có thể được sử dụng trong tin sinh học để tìm kiếm hiệu quả các mẫu trong chuỗi DNA.

5. *Định tuyến mạng:* Thuật toán KMP có thể được sử dụng trong các thuật toán định tuyến cho mạng máy tính để tìm đường đi ngắn nhất giữa hai nút trong mạng một cách hiệu quả.

*HƯỚNG PHÁT TRIỂN CỦA THUẬT TOÁN*

Trong bài báo này, chúng tôi đã phân tích, thử nghiệm và đánh giá hiệu quả của các thuật toán so khớp chuỗi. Việc hiểu rõ các thuật toán này sẽ giúp xây dựng các hệ thống an ninh mạng đạt hiệu quả cao trong thực tế. Tuy nhiên với sự bùng nổ thông tin trên internet hiện nay sẽ kéo theo sự phát triển nhanh và đa dạng của các mẫu virus và khi tập các mẫu này ngày càng lớn thì các thuật toán với độ phức tạp thời gian lớn sẽ không còn khả thi. Do đó, chúng ta cần xây dựng và cải tiến các các thuật toán có khả năng thực thi trên một tập lớn các mẫu với độ phức tạp thời gian là tuyến tính hoặc logarit. Trong các hướng tiếp cận hiện nay, cách tiếp cận dựa dựa trên hàm băm là phù hợp nhất. Việc xây dựng cấu trúc bảng băm đa lựa chọn cho phép thực hiện linh hoạt các thao tác xử lý. Thêm nữa, kết hợp sử dụng nhiều hàm băm sẽ làm tăng khả năng tìm kiếm và tốc độ so sánh giữa tập mẫu với nội dung gói tin cần kiểm soát. Việc xây dựng mô hình ánh xạ địa chỉ bộ nhớ trong bảng băm dựa trên các thông tin tóm tắt được tích hợp trên thiết bị phần cứng hay tiếp cận song song hóa các thuật toán cũng là những hướng đầy hứa hẹn và đây là hướng tiếp cận của chúng tôi trong tương lai.

*POSTER*

**