BÁO CÁO KHẢO SÁT STRING SEARCH

Nhóm team3

Thành viên: Võ Huy Thành

Thành viên Đinh Trọng Tùng Sơn

**MỘT SỐ THUẬT TOÁN TÌM KIẾM CHUỖI NỔI TIẾNG**

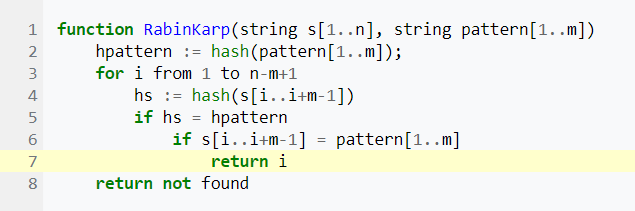
**Thuật toán vét cạn – ngây thơ**

Đây là thuật toán đơn giản nhất. Ý tưởng là ta sẽ quét qua hết tất cả trường hợp. Chuỗi cần tìm có m phần tử, văn bản để tìm kiếm có n phần tử. Ta sẽ duyệt lần lượt các phần tử trong văn bản(n) và các phần tử có trong chuỗi con rồi so sánh (m). Và độ phức tạp là O(m\*n)

**Thuật toán Rabin – Karp**

Thuật toán Rabin – Karp hay thuật toán Karp – Rabin là một thuật toán tìm kiếm chuỗi được tạo ra bởi Richard M. Karp và Michael O. Rabin (1987) sử dụng phép băm để tìm một chuỗi con khớp chính xác với một chuỗi input trong văn bản.

Ý tưởng chính là cũng giống như so sánh kiểu vét cạn. Đầu tiên ta sẽ tính giá trị băm của chuỗi truyền vào. Ta sẽ duyệt hết chuỗi trong văn bản(chuỗi con này sẽ có độ dài bằng chuỗi truyền vào), mỗi lần sẽ dịch chuyển 1 vị trí kí tự. Mỗi lần so sánh thì ta so sánh giá trị băm của 2 chuỗi nhỏ này, nếu chuỗi nhỏ này có giá trị băm như nhau thì tiếp tục so sánh từng phần tử. Nếu không sẽ dịch chuyển đến chuỗi tiếp theo.



Vậy điều quan trọng là phải implement hàm hash sao cho với các chuỗi khác nhau sẽ trả về các giá trị khác nhau, để không cần phải so sánh chi tiết từng phần tử.

Độ phức tạp trung bình. Nếu hàm hash mà tốt thì ta chỉ cần hash chuỗi truyền vào 1 lần (O(m)) , duyệt qua từng chuỗi con trong văn bản chứ không cần duyệt qua chuỗi truyền vào(O(n)). Nên độ phức tạp là O(n+m).

Độ phức tạp trong trường hợp tệ nhất: là trường hợp tất cả hàm hash tính toán đều trả về cùng giá trị. Lúc này ta cần duyệt qua từng phần tử trong chuỗi con (m lần) và duyệt qua tất cả các chuỗi con có thể trong văn bản (n-m lần). Thế nên độ phức tạp lúc này là O((n-m)\*m).

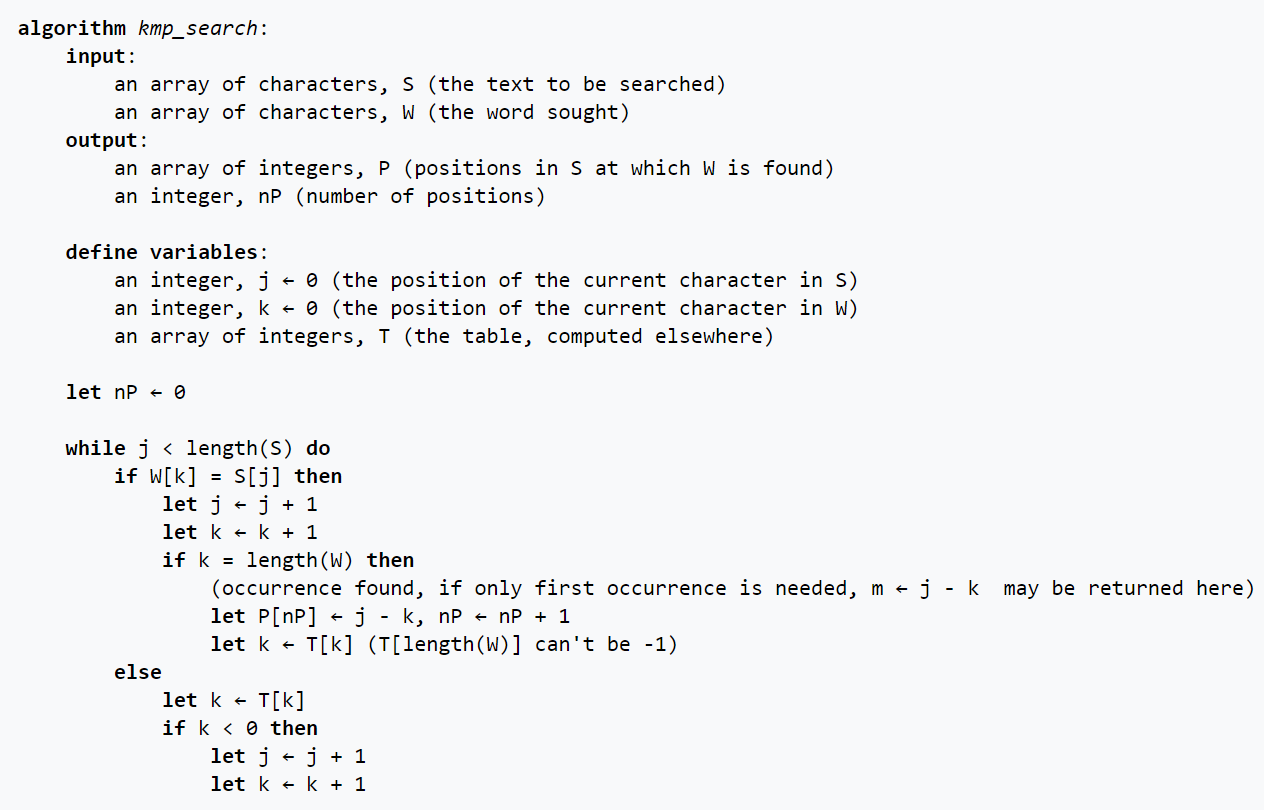
**Thuật toán  Knuth–Morris–Pratt**

Thuật toán được hình thành bởi James H. Morris và được Donald Knuth phát hiện một cách độc lập sau vài tuần từ lý thuyết automata. Morris và Vaughan Pratt cũng đã xuất bản một báo cáo kỹ thuật vào năm 1970. Sau đó cả ba cùng công bố bài báo vào năm 1977 và từ đó nó được gọi là Thuật toán Knuth-Morris-Pratt hay còn gọi là KMP Algorithm.

Ý tưởng của thuật toán là chỉ duyệt qua văn bản (n) và mỗi lần so sánh kí tự của chuỗi con và văn bản mà không trùng thì ta sẽ không lặp lại việc so chuỗi tiếp theo (sau khi dịch phải 1 index) mà sẽ tiếp tục so sánh tại vị trí không trùng.

Ban đầu ta sẽ lần lượt tìm kiếm từng phần tử của string đầu vào với từng phần tử của chuỗi con. Ta lần lượt so sánh từ trái sang phải và dịch chuyển index từ trái sáng phải. Nếu có vị trị không giống nhau thì ta sẽ dịch chuyển string đầu vào qua phải 1 đơn vị. Nếu từng kí tự trong chuỗi giống nhau thì ta ghi dấu lại. Nếu có 1 kí tự không trùng thì ngay lập tức ta xét xem phần tử gần sát cuối cùng có trùng với phần tử trong string đầu vào không. Nếu có thì dịch phần tử bên trái cùng của chuỗi con đến với phần tử trùng trong văn bản và tiếp tục so sánh. Nếu không thì ta di chuyển toàn bộ string đầu vào qua phải và điểm bắt đầu của string đầu vào trùng với vị trị không trùng trước đó. Trong quá trình so sánh nếu toàn bộ phần tử trong string đầu vào đều trùng với 1 chuỗi con nào đó đang xét thì return về vị trí index.

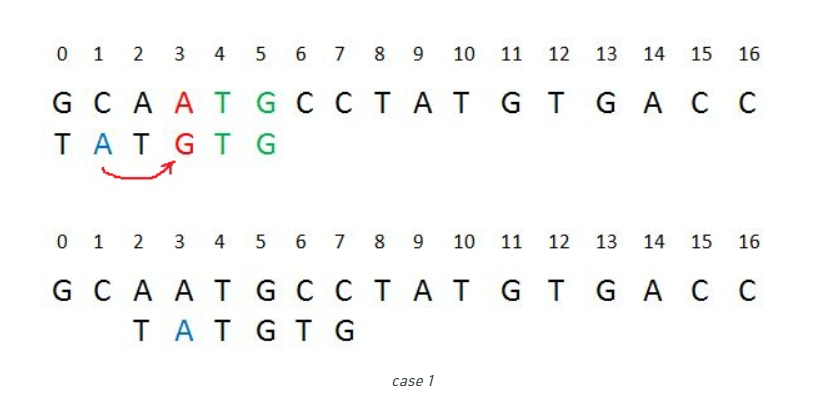
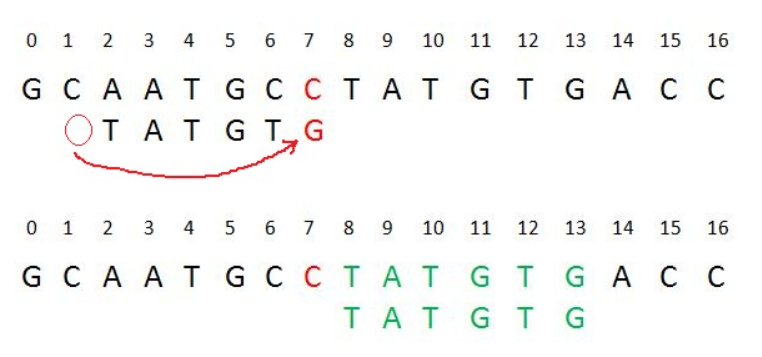
Độ phức tạp này là O(n) cho việc dịch chuyển qua tất cả các phần tử của văn bản.



**Thuật toán Boyer – Moore**

Thuật toán tìm kiếm chuỗi Boyer – Moore là một thuật toán tìm kiếm chuỗi hiệu quả, là tiêu chuẩn tiêu chuẩn cho tài liệu tìm kiếm chuỗi thực tế và được sử dụng trong nhiều phần mềm nổi tiếng vid dụ grep …..Nó được phát triển bởi Robert S. Boyer và J Strother Moore vào năm 1977.

Ý tưởng chính của bài toán là bỏ qua các lần dịch chuyển nhiều nhất có thể dựa vào quy luật dịch chuyển. String đầu vào và chuỗi con sẽ lần lượt so sánh từ phải sáng trái và dịch chuyển từ trái sang phải. Khác với các thuật toán trước đó đều so sánh và dịch chuyển từ trái sang phải. Kết hợp với bảng tìm kiếm là nguyên nhân làm cho thuật toán này trở nên tối ưu hơn.

Ta sẽ duyệ từ cuối chuỗi đến đầu chuỗi đầu vào. Nếu phần tử đang xé không trùng khớp nhau thì ta sẽ xét xem phần tử đang xét trong văn bản đó(phần tử bị lệch) có nằm trong chuỗi đầu vào không(sử dung bảng tìm kiếm O(1)). Nếu có tồn tại thì ta dịch chuyển String đầu vào sao cho phần tử đó kí tự mà nằm bên trái cùng sẽ nằm cùng vị trí với phần tử bị lệch. Với cách này thì ta sẽ bỏ qua được một số lần so sánh lãng phí. Và vì ta chọn phần tử bên trái cùng mà khớp với phần tử bị lệch nên chắc chắn sẽ không để lọt trường hợp chuỗi. Còn nếu phần tử bị lệch không nằm trong chuỗi input thì chỉ cần đơn giản dịch chuyển chuỗi vào qua ngay bên phải của phần tử bị lệch.(hình bên phải).

Độ phức tạp thuật toán: Với trường hợp tốt nhất thì tất cả các phần tử là khác nhau, nghĩ là ta chỉ cần dịch chuyển và thực hiện so sánh n/m lần. Vì vậy O(n/m)

Với trường hợp xuất nhất thì các phần tử trong văn bản là giống nhau và các phần tử trong chuỗi đầu vào cũng giống nhau thì mỗi lần phải dịch chuyển 1 lần. Nên độ phức tạp phải là O(n\*m).

**TÌM HIỂU VỀ PHẦN MỀM GREP**

**Lịch sử**

Grep là một phần mềm tìm kiếm một hoặc nhiều chuỗi con hoặc các chuỗi có đặc trưng cho trước trong 1 file văn bản thuần. Grep ban đầu được phát triển cho hệ điều hành Unix nhưng gần đây thì có các phiên bản trên hệ điều hành gần của Unix. Tác giả của grep là Ken Thompson và được lập trình bởi AT&T Bell Laboratories. Ban đầu thì có 2 phiên bản là fgrep và egrep. 2 phiên bản này dùng để tìm kiếm các string được fix cứng, có nghĩa là tìm kiếm chính xác string đó và tìm kiếm 1 string thõa mãn một syntax cho trước. Và sau này thì cả 2 được gộp lại thành grep. Và mặc định thì grep sẽ xuất ra tất cả các dòng chứa các substring ta muốn.

**Thuật toán phía sau phần mềm**

Thuật toán: Boyer-Moore algorithm, hạn chế so sánh và tìm kiếm các phần tử dư thừa.Tìm kiếm từ kí tự cuối và sử dụng bảng tìm kiếm.

GNU grep cũng giải phóng vòng lặp bên trong của Boyer-Moore và thiết lập

các mục nhập bảng delta Boyer-Moore theo cách mà nó không cần thực hiện kiểm tra thoát khỏi vòng lặp ở mỗi bước chưa được thực hiện. Kết quả điều này giúp cho GNU grep chạy trung bình ít hơn 3 x86 các instructor thực thi cho mỗi byte đầu vào mà nó thực sự nhìn vào (và nó hoàn toàn bỏ qua nhiều byte).

GNU grep sử dụng lệnh gọi hệ thống đầu vào Unix thô và tránh sao chép dữ liệusau khi đọc nó.

Và Unix hạn chế tách văn bản thành nhiều dòng bởi vì khi tìm kiếm 1 dòng mới nó sẽ phải đi tìm kiếm.

**Thao tác sử dụng grep**



Trong đó options là các lựa chọn để thực thi với các yêu cầu khác nhau và files là tên file để tìm kiếm.

Các option ta có thể lựa chọn

**-c** : hiện ra tổng số hàng chứa yêu cầu, không phải số lần tìm thấy yêu cầu

**-h :** hiện thị hàng mà yêu cầu được tìm thấy

**-i :** bỏ qua chữ hoa chữ thường, hoa thường thì cũng như nhau

**-l :** hiện thị duy nhất tên folder

**-n :** hiển thị hàng mà tìm thấy yêu cầu và số thứ tự của hàng

**-v :** hiển thị tất cả các hàng mà không tìm thấy yêu cầu

**-f file :** lấy pattern từ 1 file

**-w :** khớp chính xác toàn bộ từ trong yêu cầu

**-o :** hiển thị tất cả các substring thõa yêu cầu trên các hàng khác nhau

**KHẢO SÁT PHẦN MỀM GREP TRÊN UBUNTU**

**1.Thông số máy tính dùng thực hiện:**

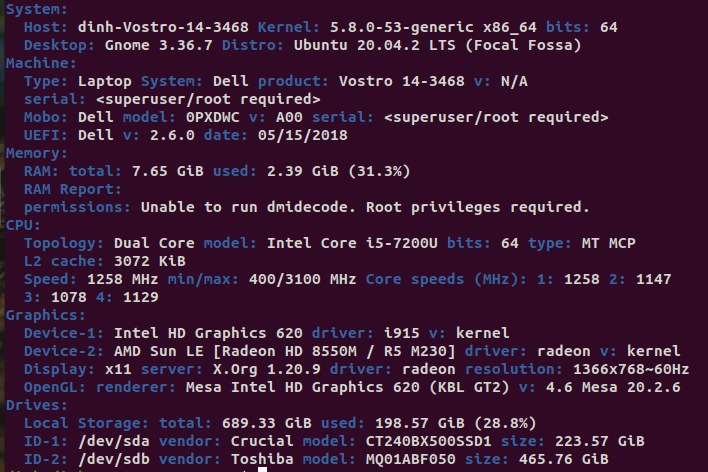
- OS: Ubuntu 20.04.2 LTS x86\_64

- CPU: Dual Core model: Intel Core i5-7200U bits: 64

-RAM: 8GB

-DISK: vendor:+Toshiba model: MQ01ABF050 size: 500 GB

+Tốc độ đọc: ~100 MB/s

 +Tốc độ ghi: ~100 MB/s

**2. Thực nghiệm:**

- Tạo file test: C++20 để tạo các file chứa chuỗi cần tìm và các file chứa chuỗi lớn, kết hợp với shell cho quá trình thực nghiệm.

- C++20 được sử dụng để tự động tạo file test và tự động tạo file csv.

- Shell được dùng để hỗ trợ cho việc chạy và đo đạc các thông số như: thời gian chạy, dung lượng ram, nhiệt độ, và dung lượng đĩa cứng, ngoài ra nó được dùng để tự động chỉnh sửa lại các file kết quả mà không lưu được các thông số cần thiết (dung lượng ram) bằng cách chạy lại chỉ một cái testcase lỗi đó.

- Để tránh việc gõ lệnh nhiều lần và giúp tiết kiệm thời gian nhóm em có sử dụng make để tự động hóa việc thử nghiệm.

- Lệnh để tự động hóa quá trình test: sudo make auto NUM\_FILE=50.

- NUM\_FILE=50 được hiểu là sẽ sử dụng 50 file khác nhau của tham số m cho lần lượt từng file của các tham số n và k, n và k mỗi tham số có số lượng file là 50.

TH1: File chứa ít chuỗi nhưng chuỗi rất dài.

Số lần test = 10\*10 = 1000 (lần test)

TH2: File chứa nhiều chuỗi, có độ dài xấp xỉ nhau.

Số lần test = 10\*10 + 10\*10 = 2000 (lần test)

**KẾT QUẢ VÀ KẾT LUẬN**

Nguồn bài viết:

[why GNU grep is fast (freebsd.org)](https://lists.freebsd.org/pipermail/freebsd-current/2010-August/019310.html)

[unix - How does grep run so fast? - Stack Overflow](https://stackoverflow.com/questions/12629749/how-does-grep-run-so-fast)

"Fast String Searching", by Andrew Hume and Daniel Sunday,

in the November 1991 issue of Software Practice & Experience