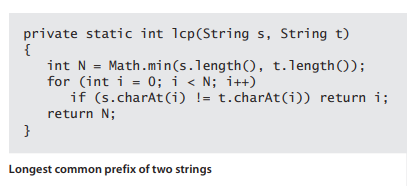
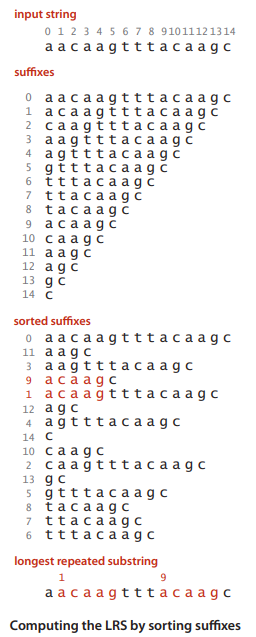
**Mảng Suffix** Các thuật toán hiệu quả để xử lý chuỗi đóng vai trò quan trọng trong các ứng dụng thương mại và trong khoa học máy tính. Từ vô số chuỗi xác định các trang web được tìm kiếm bởi hàng tỷ người dùng đến cơ sở dữ liệu genomic rộng lớn

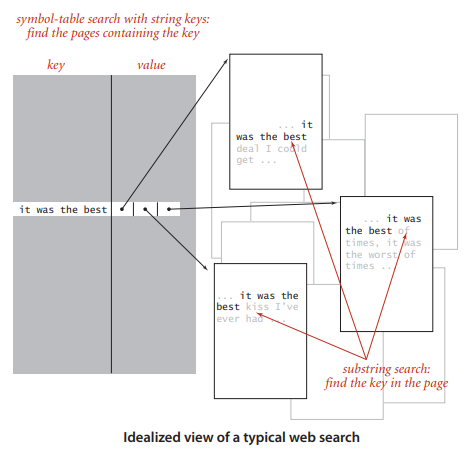
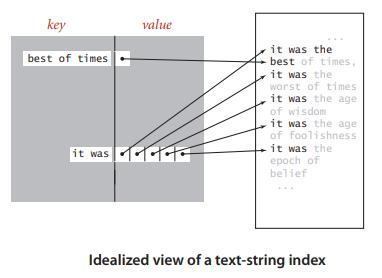
các nhà khoa học đang nghiên cứu để mở khóa bí mật của cuộc sống, các ứng dụng máy tính của thế kỷ 21 ngày càng phát triển dựa trên chuỗi. Thông thường, một số thuật toán cổ điển có hiệu quả, nhưng các thuật toán mới đáng chú ý đang được phát triển. Tiếp theo, chúng tôi mô tả một cấu trúc dữ liệu và một API hỗ trợ một số thuật toán này. Chúng tôi bắt đầu bằng cách mô tả một điển hình (và một vấn đề cổ điển) xử lý chuỗi.

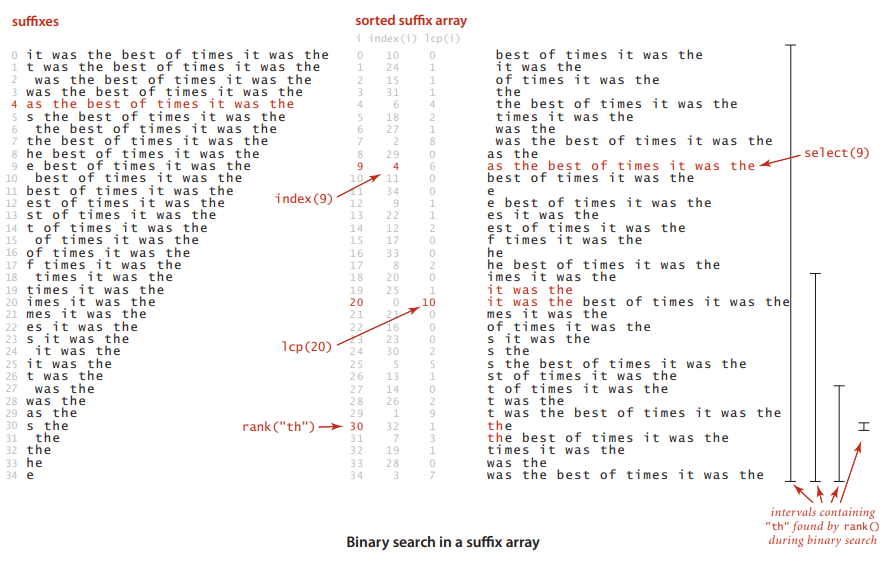
***Chuỗi con lặp lại dài nhất.***Chuỗi con dài nhất xuất hiện ít nhất hai lần trong một chuỗi nhất định là gi? Ví dụ: chuỗi con lặp lại dài nhất trong chuỗi "to be or not to be" là chuỗi "to be" . Hãy suy nghĩ ngắn gọn về cách bạn có thể giải quyết nó. Bạn có thể tìm thấy chuỗi con lặp lại dài nhất trong một chuỗi có hàng triệu ký tự? Vấn đề này là đơn giản để nêu và có nhiều ứng dụng quan trọng, bao gồm nén dữ liệu, ghi mật mã và phân tích âm nhạc hỗ trợ máy tính. Ví dụ, một kỹ thuật tiêu chuẩn được sử dụng trong việc phát triển các hệ thống phần mềm lớn là *tái cấu trúc* *code* . Các lập trình viên thường kết hợp các chương trình mới bằng cách cắt và dán code từ chương trình cũ. Trong một chương trình lớn được xây dựng trong một khoảng thời gian dài, thay thế trùng lặp code theo hàm gọi đến một bản sao của code có thể làm cho chương trình dễ dàng để hiểu và duy trì. Sự cải thiện này có thể trọn vẹn bằng cách tìm kiếm các chuỗi con lặp lại dài trong chương trình. Một ứng dụng khác được tìm thấy trong tính toán sinh học. Những mảnh giống hệt nhau được tìm thấy trong một bộ gen nhất định? Lần nữa, vấn đề tính toán cơ bản trong câu hỏi này là tìm ra chuỗi con lặp lại dài nhất trong một chuỗi. Các nhà khoa học thường quan tâm đến các câu hỏi chi tiết hơn (thực sự, bản chất của các chuỗi con lặp đi lặp lại chính xác là những gì các nhà khoa học tìm hiểu), nhưng những câu hỏi như vậy chắc chắn không dễ trả lời hơn câu hỏi cơ bản về tìm chuỗi con lặp lại dài nhất.

***Giải pháp Brute-force.***Khi khởi động, hãy xem xét nhiệm vụ đơn giản sau: đưa ra hai chuỗi, tìm *tiền tố* chung dài nhất của chúng (chuỗi con dài nhất là tiền tố của cả hai chuỗi). Ví dụ: tiền tố phổ biến dài nhất của acctgttaac và accgttaa là acc . Code bên phải là một điểm khởi đầu hữu ích cho địa chỉ nhiều thao tác phức tạp hơn: phải mất thời gian tỷ lệ thuận với chiều dài tương ứng. Bây giờ, làm thế nào để chúng ta tìm thấy chuỗi con lặp lại dài nhất trong một chuỗi nhất định? Với lcp () , giải pháp brute-force ngay lập tức gợi ý: chúng tôi so sánh chuỗi con bắt đầu từ mỗi vị trí chuỗi i với chuỗi con bắt đầu tại vị trí bắt đầu j khác , theo dõi từ dài nhất được tìm thấy. Mã code này không phù hợp với các chuỗi dài, bởi vì thời gian chạy của nó ít nhất là *bậc hai* theo chiều dài của chuỗi: như thường lệ, số lượng các cặp i và j khác nhau là *N* ( *N* - 1)/ 2, vì vậy số lượng gọi hàm lcp () cho phương pháp này sẽ là ~ *N2* /2. Sử dụng giải pháp này cho chuỗi bộ gen với hàng triệu ký tự sẽ yêu cầu hàng nghìn tỷ cuộc gọi hàm lcp () , nó là không khả thi.

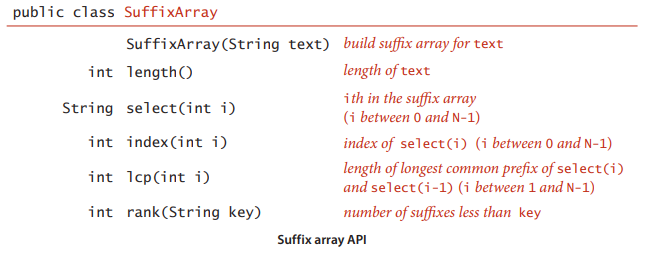
***Giải pháp sắp xếp Suffix.***Cách tiếp cận thông minh sau đây tận dụng lợi thế của việc sắp xếp một cách bất ngờ, là một cách hiệu quả để tìm chuỗi con lặp lại dài nhất, ngay cả trong một chuỗi lớn: chúng tôi sử dụng phương pháp Java’s substring() để tạo một chuỗi các chuỗi bao gồm các hậu tố của s (các chuỗi con bắt đầu từ mỗi vị trí và đi đến cuối cùng ), và sau đó chúng tôi sắp xếp mảng này. Chìa khóa của thuật toán là các chuỗi con xuất hiện ở đâu đó như một tiền tố của một trong các hậu tố trong mảng. Sau khi sắp xếp, chuỗi lặp lại dài nhất sẽ xuất hiện ở vị trí liên kề trong mảng. Như vậy, chúng ta có thể làm một lần đi qua mảng đã được sắp xếp, xét các tiền tố phù hợp dài nhất giữa các chuỗi liền kề. Cách tiếp cận này là hiệu quả đáng kể hơn phương pháp brute-force, nhưng trước khi thực hiện và phân tích nó, chúng tôi xét ứng dụng khác của sắp xếp hậu tố.



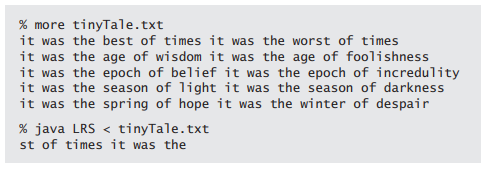
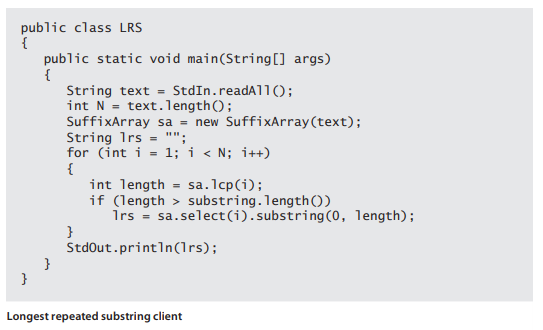
***Lập chỉ mục một chuỗi.***Khi bạn đang cố gắng tìm một chuỗi con cụ thể trong một phạm vi lớn. Ví dụ , trong khi làm việc trong trình soạn thảo văn bản hoặc trong một trang bạn đang xem với một trình duyệt mà bạn đang thực hiện *tìm kiếm chuỗi con* , vấn đề chúng tôi đã xem xét. Đối với vấn đề đó, chúng tôi giả sử văn bản tương đối lớn và tập trung vào xử lý *chuỗi con* trước, với mục tiêu có thể tìm thấy chuỗi con đó một cách hiệu quả trong bất kỳ văn bản nào. Khi bạn nhập khóa tìm kiếm vào trình duyệt web, bạn đang thực hiện *tìm kiếm* *với các khóa chuỗi*. Công cụ tìm kiếm của bạn phải tính toán trước một chỉ mục, vì nó không đủ khả năng để quét tất cả các trang trong web để tìm khóa của bạn. Khi chúng ta thảo luận ở phần trước, điều này sẽ lý tưởng là một chỉ mục đảo liên kết mỗi chuỗi có thể tìm kiếm với tất cả các trang web bao gồm nó một bảng biểu tượng trong đó mỗi bảng mục nhập là một khóa chuỗi và mỗi giá trị là một tập hợp các con trỏ (mỗi con trỏ nhận thông tin cần thiết để định vị vị trí xuất hiện của khóa trên web có thể là một URL có tên một trang web và một số nguyên di chuyển trong trang đó). Trong thực tế, một bảng biểu tượng sẽ là quá lớn, vì vậy công cụ tìm kiếm của bạn sử dụng nhiều các thuật toán phức tạp để giảm kích cỡ của nó. Một cách tiếp cận là xếp hạng trang web quan trọng (có lẽ sử dụng một thuật toán như thuật toán PageRank mà chúng ta đã thảo luận trên) và chỉ làm việc với trang có thứ hạng cao, không phải tất cả các trang. Một cách tiếp cận khác để giảm kích thước của bảng biểu tượng để hỗ trợ tìm kiếm bằng các khóa chuỗi là liên kết URL với các *từ* (các chuỗi được phân tách bằng khoảng trắng) làm các khóa trong chỉ mục được tính toán trước. Sau đó, khi bạn tìm kiếm một từ, công cụ tìm kiếm có thể sử dụng chỉ mục để tìm từ (quan trọng) các trang chứa khóa tìm kiếm của bạn (từ) và sau đó sử dụng tìm kiếm chuỗi con trong mỗi trang để tìm thấy chúng. Nhưng với phương pháp này, nếu văn bản chứa "everything" và bạn đã tìm kiếm "thing" , bạn sẽ không tìm thấy nó. Đối với một số ứng dụng, nó là đáng giá để xây dựng một chỉ số để giúp đỡ tìm *bất kỳ chuỗi con* trong một văn bản nhất định. Việc này có thể là hợp lý cho một nghiên cứu ngôn ngữ của một tác phẩm văn học quan trọng, cho một dãy gen nó có thể là một đối tượng nghiên cứu cho nhiều nhà khoa học, hoặc chỉ cho một cách rộng rãi truy cập trang web. Một lần nữa, ý tưởng là chỉ số sẽ liên kết tất cả các chuỗi con có thể của chuỗi văn bản với từng vị trí nơi nó xuất hiện trong chuỗi văn bản, như được mô tả ở bên phải. Bài toán cổ điển với ý tưởng này là số của các chuỗi con là quá lớn để có một mục biểu tượng cho từng chuỗi (một *N* - ký tự có *N*( *N* - 1) /2 chuỗi con).Bảng ví dụ bên phải sẽ cần các mục cho b , be , bes, best, best o, best of, e, es, est, est o, est of, s, st, st o, st of, t, t o, t of, o, of và nhiều, nhiều chuỗi con khác. Một lần nữa, chúng ta có thể sử dụng một loại hậu tố để giải quyết vấn đề này theo cách tương tự như việc thực hiện bảng ký hiệu đầu tiên của chúng tôi sử dụng tìm kiếm nhị phân. Chúng tôi coi mỗi hậu tố *N* là khóa, tạo một mảng được sắp xếp của các khóa (hậu tố) và sử dụng tìm kiếm nhị phân để tìm kiếm trong mảng đó,so sánh khóa tìm kiếm với mỗi hậu tố.

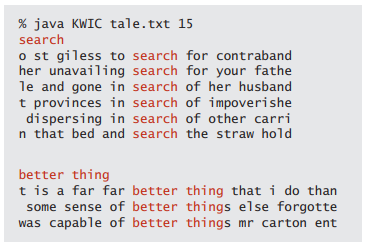
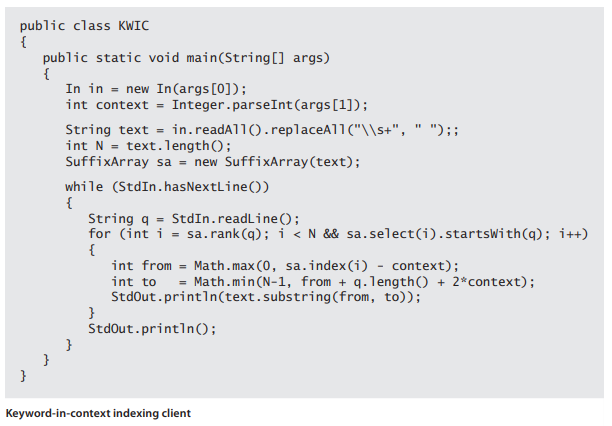


***API và mã máy khách.***Để hỗ trợ mã máy khách để giải quyết hai vấn đề này, chúng tôi nói rõ API hiển thị bên dưới. Nó bao gồm một cấu trúc; một phương thức length () ; phương thức select () và index () , cung cấp chuỗi và chỉ mục của hậu tố của một thứ hạng nhất định trong danh sách sắp xếp các hậu tố; một phương thức lcp () cho độ dài của tiền tố phổ biến dài nhất của mỗi hậu tố và một tiền tố trước nó trong danh sách được sắp xếp; và một phương thức rank() cung cấp số lượng hậu tố ít hơn khóa đã cho (giống như chúng ta đã sử dụng kể từ lần đầu tiên chúng tôi kiểm tra tìm kiếm nhị phân). Chúng ta sử dụng *mảng hậu tố* để mô tả sự trừu tượng của một danh sách các chuỗi hậu tố được sắp xếp mà không nhất thiết phải cam kết để sử dụng một chuỗi các chuỗi làm cấu trúc dữ liệu cơ bản.



Trong ví dụ trên trang trước, chọn (9) là "as the best of times ..." , chỉ mục (9) là 4 , lcp (20) là 10 vì "it was the best of times ..." và "it was the" có tiền tố phổ biến "it was the" có độ dài 10 và rank ("th") là 30 . Chú ý ngoài ra, select (rank (key)) là hậu tố có thể đầu tiên trong danh sách hậu tố được sắp xếp có khóa là tiền tố và tất cả các lần xuất hiện khác của khóa trong văn bản ngay lập tức theo sau (xem hình trên trang trước). Với API này, mã máy khách trên hai trang tiếp theo là ngay lập tức LRS tìm thấy chuỗi con lặp lại dài nhất trong văn bản đầu vào bằng cách xây dựng một mảng hậu tố và sau đó quét qua các hậu tố được sắp xếp để tìm giá trị lcp () tối đa . KWIC xây dựng một mảng hậu tố cho văn bản có tên làm đối số dòng lệnh, lấy các truy vấn từ đầu vào tiêu chuẩn và in tất cả sự xuất hiện của mỗi truy vấn trong văn bản (bao gồm một số ký tự được chỉ định trước và sau để đưa ra văn cảnh). Tên KWIC là viết tắt của keyword-in-context search , một thuật ngữ có ít nhất từ những năm 1960. Sự đơn giản và hiệu quả của mã khách hàng này cho các ứng dụng xử lý chuỗi điển hình là đáng chú ý và minh chứng cho tầm quan trọng của thiết kế API cẩn thận (và sức mạnh của một ý tưởng đơn giản nhưng khéo léo).





***Thực hiện.***Câu lệnh trên trang trước là một triển khai đơn giản của các SuffixArray API. Các biến thể hiện của nó là một mảng các chuỗi và một biến N chứa độ dài của mảng (độ dài của chuỗi và số lượng hậu tố). Hàm tạo xây dựng mảng hậu tố và sắp xếp nó, vì vậy select (i) chỉ trả về suffixes [i]. Việc thực hiện index() cũng là một-liner, nhưng nó là hơi khó, dựa trên quan sát rằng *độ dài của chuỗi hậu tố xác định duy nhất* *là* *điểm bắt đầu của nó* . Hậu tố có độ dài N bắt đầu ở vị trí 0 , hậu tố có độ dài N-1 bắt đầu ở vị trí 1 , hậu tố có độ dài N-2 bắt đầu từ vị trí 2 , v.v., vì vậy index(i) chỉ trả về N - suffixes [i] .length () . Việc thực hiện lcp () là ngay lập tức, đưa ra phương thức tĩnh lcp () và rank() gần giống như phương thức tìm kiếm nhị phân cho các bảng ký hiệu. Một lần nữa, sự đơn giản và tao nhã của việc thực hiện này không nên che dấu sự thật rằng nó là một thuật toán phức tạp cho phép giải quyết các vấn đề quan trọng như chuỗi con lặp lại dài nhất đó dường như là không thể.

***Hiệu suất.***Hiệu quả của việc sắp xếp hậu tố phụ thuộc vào thực tế là chuỗi con Java

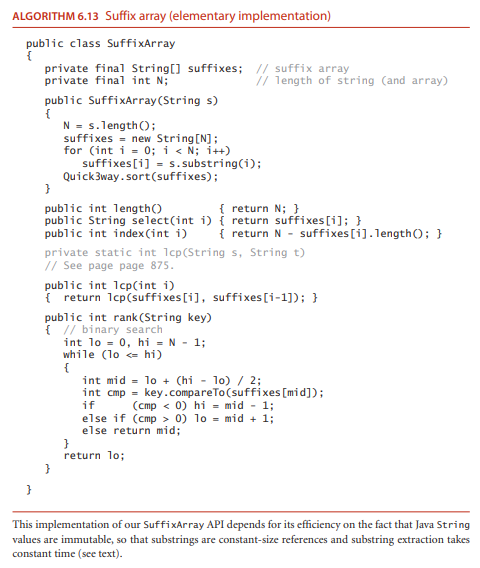
trích xuất sử dụng một lượng không gian không đổi, mỗi chuỗi con được cấu thành từ đối tượng chuẩn, một con trỏ vào bản gốc và chiều dài. Do đó, kích thước của chỉ số

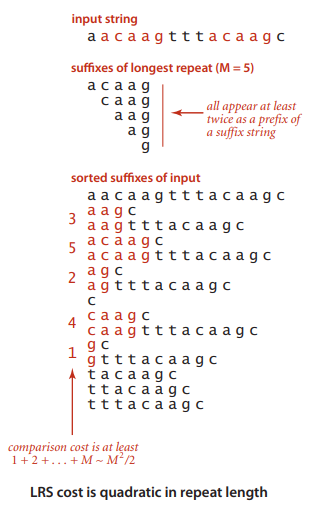
là tuyến tính trong kích thước của chuỗi. Điểm này hơi phản trực giác vì tổng số ký tự trong các hậu tố là ~ *N2*/2, một hàm bậc hai của kích thước của chuỗi. Hơn nữa, yếu tố bậc hai đó tạo ra một khoảng dừng khi xem xét chi phí của sắp xếp mảng hậu tố. Điều này rất quan trọng là phải nhớ rằng phương pháp này là hiệu quả cho chuỗi dài vì biểu diễn Java cho chuỗi: khi chúng ta trao đổi hai chuỗi, chúng ta chỉ trao đổi các tham chiếu, không phải toàn bộ chuỗi. Bây giờ, chi phí của việc so sánh hai chuỗi có thể tỷ lệ thuận với độ dài của chuỗi trong trường hợp khi tiền tố chung của chúng rất dài, nhưng hầu hết các so sánh trong các ứng dụng điển hình đều liên quan đến chỉ một vài ký tự. Nếu vậy, thời gian chạy của sắp xếp hậu tố là tuyến tính. Ví dụ, trong nhiều ứng dụng, thật hợp lý để sử dụng mô hình chuỗi ngẫu nhiên:

**Mệnh đề C.**Sử dụng quicksort chuỗi 3 chiều, chúng ta có thể xây dựng một mảng hậu tố từ một chuỗi ngẫu nhiên có độ dài *N* với không gian tỷ lệ với *N* và *~* 2 *N* ln *N* ký tự so sánh, trên trung bình.

**Thảo luận:**Không gian bị ràng buộc là ngay lập tức, nhưng thời gian bị ràng buộc là từ một kết quả nghiên cứu chi tiết và khó khăn của P. Jacquet và W. Szpankowski, trong đó ngụ ý rằng chi phí sắp xếp các hậu tố là không có triệu chứng giống như chi phí của

sắp xếp *N* chuỗi ngẫu nhiên.



***Cải thiện triển khai.***Triển khai cơ bản của chúng tôi về SuffixArray có hiệu suất trường hợp xấu nhất. Ví dụ: nếu tất cả các ký tự đều bằng nhau, các loại kiểm tra mỗi ký tự trong mỗi chuỗi con và do đó mất thời gian *bậc hai* . Đối với chuỗi của loại chúng ta đã sử dụng làm ví dụ, chẳng hạn như trình tự bộ gen hoặc văn bản ngôn ngữ tự nhiên, điều này không có khả năng là vấn đề, nhưng thuật toán có thể chậm do các văn bản với thời gian dài của các ký tự giống hệt nhau. Một cách nhìn khác về vấn đề là quan sát rằng chi phí tìm kiếm chuỗi con lặp lại dài nhất là *bậc hai* *trong độ dài của chuỗi con* vì tất cả tiền tố lặp lại cần phải được kiểm tra (xem sơ đồ bên phải). Đây không phải là một vấn đề cho một văn bản như *A Tale of Two Cities* , nơi lặp lại lâu nhất

"s dropped because it would have been a bad thing for me in a worldly point of view i "

chỉ có 84 ký tự, nhưng nó là một vấn đề nghiêm trọng cho dữ liệu gen, trong đó lặp đi lặp lại chuỗi không phải là bất thường. Làm thế nào có thể hành vi bậc hai tìm kiếm lặp lại được tránh? Lại đáng chú ý, nghiên cứu của P. Weiner năm 1973 cho thấy

rằng *có thể giải quyết vấn đề chuỗi con lặp lại dài nhất trong thời gian tuyến tính được đảm bảo* . Thuật toán của Weiner dựa trên việc xây dựng cấu trúc dữ liệu cây hậu tố (về cơ bản là trie cho hậu tố). Với nhiều con trỏ cho mỗi ký tự, cây hậu tố tiêu thụ quá nhiều

không gian cho nhiều vấn đề thực tế, dẫn đến sự phát triển của mảng hậu tố. Những năm 1990, U. Manber và E. Myers đã trình bày một thuật toán tuyến tính để xây dựng mảng hậu tố trực tiếp và một phương thức xử lý cùng lúc với sắp xếp hậu tố để hỗ trợ lcp () . Một số thuật toán sắp xếp hậu tố thời gian tuyến tính đã được phát triển từ đó. Với nhiều công việc hơn, việc triển khai Manber-Myers cũng có thể hỗ trợ một lcp () hai đối số tìm thấy tiền tố chung dài nhất của hai hậu tố là không nhất thiết phải liền kề trong thời gian liên tục được bảo đảm, một lần nữa đáng chú ý cải thiện việc thực hiện đơn giản. Những kết quả này khá kỳ lạ, vì họ đạt được hiệu quả vượt xa những gì bạn có thể mong đợi.

**Mệnh đề D.**Với mảng hậu tố, chúng ta có thể giải quyết cả cách sắp xếp hậu tố và vấn đề chuỗi con lặp đi lặp lại dài nhất trong thời gian tuyến tính.

**Bằng chứng:**Các thuật toán đáng chú ý cho các tác vụ này nằm ngoài phạm vi của chúng ta, nhưng bạn có thể tìm thấy trên mã bookite thực hiện hàm tạo SuffixArray trong thời gian tuyến tính và truy vấn lcp () trong thời gian không đổi.

Việc triển khai SuffixArray dựa trên những ý tưởng hỗ trợ hiệu quả các giải của nhiều vấn đề xử lý chuỗi, với mã máy khách đơn giản, như trong LRS và KWIC đã ví dụ.

Mảng Suffix là đỉnh cao của nhiều thập kỷ nghiên cứu bắt đầu với sự phát triển của trie cho các chỉ số KWIC trong những năm 1960. Các thuật toán mà chúng tôi thảo luận đã được làm bởi nhiều nhà nghiên cứu trong nhiều thập kỷ trong bối cảnh giải quyết các vấn đề thực tế khác nhau, từ việc đưa *Từ điển tiếng Anh Oxford* trực tuyến đến phát triển các công cụ tìm kiếm web đầu tiên để giải trình tự bộ gen người. Câu chuyện này chắc chắn giúp việc phân tích và thiết kế thuật toán càng trở nên quan trọng.