**2.5 Ứng dụng của sắp xếp**

Các thuật toán sắp xếp và hàng đợi ưu tiên được sử dụng rộng rãi trong nhiều ứng dụng khác nhau. Mục đích của phần này là điểm qua một số ứng dụng đó.

**Sắp xếp các loại dữ liệu khác nhau**

Các cài đặt của Chúng ta sắp xếp các mảng đối tượng thuộc loại Comparable. Quy ước này trong Java cho phép Chúng ta sử dụng cơ chế gọi lại của Java để sắp xếp các mảng đối tượng thuộc bất kỳ loại nào mà triển khai giao diện Comparable.

**Ví dụ về giao dịch**

Chương trình Transaction.java triển khai giao diện Comparable cho một kiểu dữ liệu giao dịch, dựa trên thời điểm giao dịch xảy ra.

**Sắp xếp bằng con trỏ**

Cách tiếp cận mà Chúng ta đang sử dụng được biết đến trong tài liệu cổ điển là sắp xếp bằng con trỏ. Gọi như vậy vì Chúng ta xử lý các tham chiếu đến khóa và không di chuyển dữ liệu thực tế.

**Khóa là bất biến**

Có lý do để tin rằng một mảng có thể không còn được sắp xếp nếu người dùng được phép thay đổi giá trị của các khóa sau khi sắp xếp. Trong Java, nên đảm bảo rằng các giá trị khóa không thay đổi bằng cách sử dụng các khóa bất biến.

**Hoán đổi là rẻ**

Một lợi thế khác của việc sử dụng tham chiếu là Chúng ta tránh được chi phí di chuyển toàn bộ các mục. Cách tiếp cận sử dụng tham chiếu làm cho chi phí của một lần hoán đổi gần bằng với chi phí của một lần so sánh trong các tình huống chung.

**Trật tự thay thế**

Có nhiều ứng dụng yêu cầu chúng ta sử dụng hai thứ tự khác nhau cho các đối tượng mà chúng ta sắp xếp, tùy thuộc vào tình huống. Giao diện Comparator trong Java có một phương thức công khai duy nhất là compare() để so sánh hai đối tượng. Nếu chúng ta có một kiểu dữ liệu triển khai giao diện này, chúng ta có thể truyền một Comparator vào hàm sort() (nó sẽ truyền vào hàm less()) như trong Insertion.java.

**Các mục có nhiều khóa**

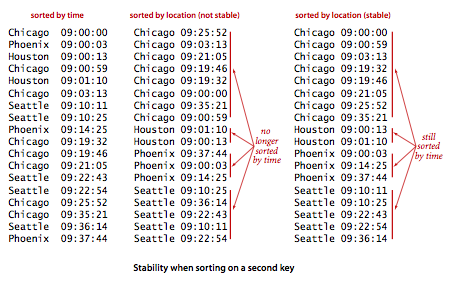
Trong các ứng dụng điển hình, các mục có nhiều biến thể hiện có thể cần được sử dụng làm khóa sắp xếp. Trong ví dụ giao dịch của Chúng ta, một khách hàng có thể cần sắp xếp danh sách giao dịch theo số tài khoản; một khách hàng khác có thể cần sắp xếp danh sách theo vị trí; và các khách hàng khác có thể cần sử dụng các trường khác làm khóa sắp xếp. Chúng ta có thể định nghĩa nhiều bộ so sánh khác nhau, như trong Transaction.java.

**Hàng đợi ưu tiên với comparator**

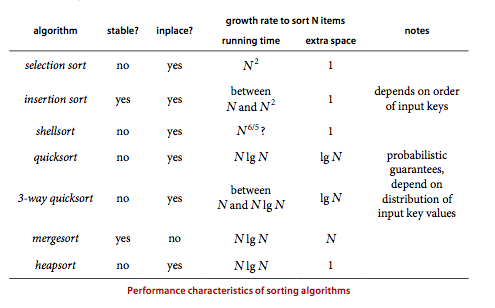
Tính linh hoạt tương tự của việc sử dụng các bộ so sánh cũng hữu ích cho hàng đợi ưu tiên. MaxPQ.java và MinPQ.java bao gồm một hàm khởi tạo nhận một Comparator làm tham số.

**Tính ổn định**

Một phương pháp sắp xếp được gọi là ổn định nếu nó giữ nguyên thứ tự tương đối của các khóa bằng nhau trong mảng. Ví dụ, trong ứng dụng thương mại điện tử của chúng ta, giả sử rằng chúng ta nhập các giao dịch vào mảng khi chúng đến, vì vậy chúng sẽ theo thứ tự của trường thời gian trong mảng. Bây giờ giả sử rằng ứng dụng yêu cầu các giao dịch được tách ra theo địa điểm để xử lý thêm. Một cách dễ dàng để làm điều này là sắp xếp mảng theo địa điểm. Nếu sắp xếp không ổn định, các giao dịch cho mỗi thành phố có thể không còn theo thứ tự thời gian sau khi sắp xếp. Một số phương pháp sắp xếp mà chúng ta đã xem xét trong chương này là ổn định (sắp xếp chèn và sắp xếp trộn); nhiều phương pháp thì không (sắp xếp chọn, sắp xếp shell, sắp xếp nhanh, và sắp xếp heap).



Việc biết được thuật toán nào là tốt nhất phụ thuộc rất nhiều vào chi tiết của ứng dụng và cách cài đặt, nhưng chúng ta đã nghiên cứu một số phương pháp đa dụng có thể gần như hiệu quả nhất cho nhiều ứng dụng khác nhau. Bảng sau đây là một hướng dẫn chung tóm tắt các đặc điểm quan trọng của các thuật toán sắp xếp mà chúng ta đã nghiên cứu trong chương này.



**Thuộc tính:** Quicksort là thuật toán sắp xếp tổng quát nhanh nhất.

Trong hầu hết các tình huống thực tế, quicksort là phương pháp được lựa chọn. Nếu tính ổn định quan trọng và có đủ bộ nhớ, mergesort có thể là tốt nhất. Trong một số ứng dụng yêu cầu hiệu suất cao, mục tiêu có thể chỉ là sắp xếp các số, vì vậy sẽ hợp lý khi tránh chi phí sử dụng tham chiếu và thay vào đó sắp xếp các kiểu dữ liệu nguyên thủy.

**Sắp xếp các kiểu dữ liệu nguyên thủy**

Chúng ta có thể phát triển các phiên bản hiệu quả hơn của mã sắp xếp bằng cách thay thế Comparable bằng tên kiểu dữ liệu nguyên thủy, và thay thế các lời gọi hàm less() bằng các đoạn mã như `a[i] < a[j]`. Tuy nhiên, cần chú ý với các kiểu dữ liệu dấu phẩy động để xử lý các giá trị như -0.0 và NaN.

**Sắp xếp hệ thống trong Java**

Phương thức sắp xếp chính của hệ thống Java là Arrays.sort() trong thư viện java.util, bao gồm một tập hợp các phương thức quá tải:

- Một phương thức khác nhau cho mỗi kiểu dữ liệu nguyên thủy.

- Một phương thức cho các kiểu dữ liệu triển khai Comparable.

- Một phương thức sử dụng Comparator.

Các lập trình viên hệ thống của Java đã chọn sử dụng quicksort (với phân vùng 3 chiều) để triển khai các phương thức dành cho kiểu dữ liệu nguyên thủy, và mergesort cho các phương thức dành cho kiểu tham chiếu. Hệ quả thực tế chính của các lựa chọn này là đánh đổi giữa tốc độ và việc sử dụng bộ nhớ (đối với kiểu dữ liệu nguyên thủy) so với tính ổn định và hiệu suất được đảm bảo (đối với kiểu tham chiếu).

**Phép giảm thiểu**

Ý tưởng sử dụng các thuật toán sắp xếp để giải quyết các vấn đề khác là một ví dụ của kỹ thuật cơ bản trong thiết kế thuật toán được gọi là phép giảm thiểu. Một phép giảm thiểu là tình huống trong đó một thuật toán phát triển cho một vấn đề được sử dụng để giải quyết một vấn đề khác. Chúng ta sẽ bắt đầu với một số ví dụ cơ bản về việc sắp xếp.

**Các giá trị trùng lặp**

Có bất kỳ khóa nào trùng lặp trong một mảng đối tượng Comparable không? Có bao nhiêu khóa khác nhau trong mảng? Giá trị nào xuất hiện thường xuyên nhất? Với sắp xếp, bạn có thể trả lời các câu hỏi này trong thời gian log tuyến tính: trước tiên sắp xếp mảng, sau đó duyệt qua mảng đã sắp xếp, ghi chú các giá trị trùng lặp xuất hiện liên tiếp trong mảng đã được sắp xếp.

**Xếp hạng**

Một hoán vị (hoặc xếp hạng) là một mảng N số nguyên, trong đó mỗi số nguyên từ 0 đến N-1 xuất hiện chính xác một lần. Khoảng cách Kendall tau giữa hai xếp hạng là số cặp nằm ở thứ tự khác nhau trong hai xếp hạng. Ví dụ, khoảng cách Kendall tau giữa 0 3 1 6 2 5 4 và 1 0 3 6 4 2 5 là bốn, vì các cặp 0-1, 3-1, 2-4, 5-4 nằm ở thứ tự khác nhau trong hai xếp hạng, nhưng tất cả các cặp khác có thứ tự giống nhau.

**Phép giảm thiểu hàng đợi ưu tiên**

Chúng ta đã xem xét hai ví dụ về các vấn đề có thể được giảm thiểu thành một chuỗi các thao tác trên hàng đợi ưu tiên. TopM.java tìm M phần tử có khóa lớn nhất trong một luồng đầu vào. Multiway.java gộp M luồng đầu vào đã được sắp xếp để tạo ra một luồng đầu ra đã sắp xếp. Cả hai vấn đề này đều được giải quyết dễ dàng bằng một hàng đợi ưu tiên kích thước M.

**Số trung vị và thống kê thứ tự**

Phân vùng để tìm số trung vị là một ứng dụng quan trọng liên quan đến sắp xếp. Tìm số trung vị của một tập hợp các khóa (giá trị mà một nửa số khóa nhỏ hơn hoặc bằng, và nửa còn lại lớn hơn hoặc bằng) là một tính toán phổ biến trong thống kê và trong các ứng dụng xử lý dữ liệu khác nhau. Tìm số trung vị là một trường hợp đặc biệt của việc chọn lựa: tìm phần tử nhỏ thứ k trong một tập hợp số. Bạn có thể dễ dàng giải quyết vấn đề này trong thời gian log tuyến tính bằng cách sắp xếp.

Chúng ta mô tả một cách tiếp cận giải quyết vấn đề trong thời gian tuyến tính: duy trì các biến lo và hi để giới hạn phạm vi con chứa chỉ số k của phần tử cần chọn và sử dụng phân vùng quicksort để thu nhỏ kích thước của phạm vi con, như sau:

- Nếu k bằng j, thì chúng ta đã xong.

- Nếu k < j, thì chúng ta tiếp tục xử lý phạm vi con bên trái (bằng cách thay đổi giá trị hi thành j-1).

- Nếu k > j, thì chúng ta tiếp tục xử lý phạm vi con bên phải (bằng cách thay đổi lo thành j+1).

Phạm vi thu nhỏ dần cho đến khi chỉ còn lại k. Khi kết thúc, a[k] chứa phần tử nhỏ thứ (k+1), a[0] đến a[k-1] đều nhỏ hơn (hoặc bằng) a[k], và a[k+1] đến cuối mảng đều lớn hơn (hoặc bằng) a[k].

Phương thức select() trong Quick.java triển khai cách tiếp cận này, nhưng nó yêu cầu phải có một phép ép kiểu trong chương trình khách. Phương thức select() trong QuickPedantic.java là một đoạn mã cẩn thận hơn, không yêu cầu phải ép kiểu.

Khảo sát ngắn về các ứng dụng của sắp xếp.

**Tính toán thương mại**

Các tổ chức chính phủ, tổ chức tài chính và các doanh nghiệp thương mại tổ chức nhiều thông tin bằng cách sắp xếp nó. Dù là tài khoản được sắp xếp theo tên hoặc số, giao dịch được sắp xếp theo thời gian hoặc địa điểm, thư được sắp xếp theo mã bưu điện hoặc địa chỉ, tệp tin được sắp xếp theo tên hoặc ngày tháng, hay bất cứ thông tin nào khác, việc xử lý dữ liệu này chắc chắn sẽ cần đến một thuật toán sắp xếp.

**Tìm kiếm thông tin**

Giữ dữ liệu theo thứ tự sắp xếp giúp có thể tìm kiếm nó một cách hiệu quả bằng cách sử dụng thuật toán tìm kiếm nhị phân cổ điển.

**Nghiên cứu vận hành**

Giả sử chúng ta có N công việc cần hoàn thành, trong đó công việc thứ j cần t\_j giây để xử lý. Chúng ta cần hoàn thành tất cả các công việc, nhưng muốn tối đa hóa sự hài lòng của khách hàng bằng cách giảm thiểu thời gian hoàn thành trung bình của các công việc. Quy tắc ưu tiên công việc có thời gian xử lý ngắn nhất trước tiên, nơi chúng ta lập lịch các công việc theo thứ tự tăng dần của thời gian xử lý, được biết đến là phương pháp tốt nhất để đạt được mục tiêu này. Một ví dụ khác là bài toán cân bằng tải, trong đó chúng ta có M bộ xử lý giống nhau và N công việc cần hoàn thành, mục tiêu là lập lịch tất cả các công việc lên các bộ xử lý sao cho thời gian hoàn thành công việc cuối cùng là sớm nhất có thể. Bài toán này thuộc loại NP-khó nên chúng ta không mong đợi tìm được cách tối ưu một cách thực tế. Một phương pháp được biết đến là quy tắc ưu tiên công việc có thời gian xử lý dài nhất trước tiên, trong đó chúng ta xem xét các công việc theo thứ tự giảm dần của thời gian xử lý, gán mỗi công việc cho bộ xử lý nào có sẵn đầu tiên.

**Mô phỏng sự kiện**

Nhiều ứng dụng khoa học liên quan đến mô phỏng, nơi mục đích của tính toán là mô phỏng một khía cạnh nào đó của thế giới thực để hiểu rõ hơn về nó. Thực hiện các mô phỏng này một cách hiệu quả có thể đòi hỏi các thuật toán và cấu trúc dữ liệu thích hợp. Chúng ta sẽ xem xét một ví dụ về mô phỏng va chạm hạt trong Mục 6.1 để minh họa điểm này.

**Tính toán số học**

Tính toán khoa học thường quan tâm đến độ chính xác (chúng ta có gần đúng với kết quả thực sự không?). Độ chính xác rất quan trọng khi chúng ta thực hiện hàng triệu phép tính với các giá trị ước tính, chẳng hạn như biểu diễn dấu phẩy động của các số thực mà chúng ta thường dùng trên máy tính. Một số thuật toán số học sử dụng hàng đợi ưu tiên và sắp xếp để kiểm soát độ chính xác trong các tính toán.

**Tìm kiếm tổ hợp**

Một mô hình cổ điển trong trí tuệ nhân tạo là định nghĩa một tập hợp các cấu hình với các bước di chuyển được xác định rõ ràng từ một cấu hình sang cấu hình khác và mỗi bước di chuyển có một mức ưu tiên. Cũng có một cấu hình khởi đầu và một cấu hình mục tiêu (tương ứng với việc giải quyết xong vấn đề). Thuật toán A\ là một quá trình giải quyết vấn đề, trong đó chúng ta đưa cấu hình khởi đầu vào hàng đợi ưu tiên, sau đó thực hiện các bước sau cho đến khi đạt được mục tiêu: loại bỏ cấu hình có ưu tiên cao nhất và thêm vào hàng đợi tất cả các cấu hình có thể đạt được từ đó chỉ với một bước di chuyển (ngoại trừ cấu hình vừa bị loại bỏ).

Thuật toán Prim và Dijkstra là những thuật toán cổ điển để xử lý đồ thị. Hàng đợi ưu tiên đóng vai trò cơ bản trong việc tổ chức tìm kiếm đồ thị, cho phép các thuật toán hiệu quả.

Thuật toán Kruskal là một thuật toán cổ điển khác dành cho đồ thị có cạnh mang trọng số, trong đó việc xử lý các cạnh theo thứ tự trọng số là quan trọng. Thời gian chạy của nó chủ yếu phụ thuộc vào chi phí của việc sắp xếp.

Nén Huffman là một thuật toán nén dữ liệu cổ điển, dựa trên việc xử lý một tập hợp các phần tử có trọng số nguyên bằng cách kết hợp hai phần tử nhỏ nhất để tạo ra một phần tử mới có trọng số bằng tổng của hai phần tử gốc. Việc thực hiện thao tác này rất đơn giản, sử dụng hàng đợi ưu tiên.

**Thuật toán xử lý chuỗi thường dựa trên sắp xếp**

Ví dụ, chúng ta sẽ thảo luận các thuật toán tìm tiền tố chung dài nhất giữa một tập hợp các chuỗi và chuỗi lặp lại dài nhất trong một chuỗi cho trước, dựa trên việc sắp xếp các hậu tố của chuỗi đó trước tiên.

**Kết luận**

Thuật toán sắp xếp đóng vai trò cốt lõi trong nhiều lĩnh vực ứng dụng, từ thương mại, nghiên cứu, đến khoa học và kỹ thuật. Khả năng tổ chức và xử lý dữ liệu hiệu quả thông qua sắp xếp không chỉ giúp tối ưu hóa hiệu suất hệ thống mà còn là cơ sở để giải quyết nhiều bài toán phức tạp như nén dữ liệu, tìm kiếm, cân bằng tải, và mô phỏng. Các thuật toán sắp xếp khác nhau được áp dụng tùy theo yêu cầu cụ thể về thời gian, bộ nhớ và tính ổn định, mang lại sự linh hoạt và hiệu quả trong việc xử lý dữ liệu quy mô lớn. Việc lựa chọn đúng thuật toán sắp xếp cho từng ứng dụng cụ thể có thể mang lại lợi ích lớn về mặt hiệu quả và hiệu suất.