CHƯƠNG 1 - Vai trò của thuật toán trong lập trình

Thuật toán là gì? Tại sao nên nghiên cứu về thuật toán? Thuật toán liên hệ tới những công nghệ gì khác trong máy tính? Trong chương này, ta sẽ trả lời những câu hỏi đó.

1.1 Thuật toán

Theo cách hiểu không chính thức, thuật toán là bất kỳ một phương thức tính toán được định nghĩa rõ ràng bởi vài giá trị hoặc một tập hợp giá trị, như là input (đầu vào) và trả về một hoặc tập hợp giá trị, như là output (đầu ra). Thuật toán là một chuỗi các bước chuyển đổi giá trị input thành output.

Chúng ta cũng có thể xem một thuật toán như là công cụ để giải quyết một vấn đề máy tính được xác định rõ ràng. Vấn đề yêu cầu chỉ rõ mối quan hệ giữa input và output. Thuật toán miêu tả một thủ tục máy tính đặc trưng để đạt được mối quan hệ đó.

Lấy ví dụ, chúng ta cần sắp xếp các số theo trình tự tăng dần. Vấn đề này phát sinh thường xuyên trong thực tế và cung cấp ví dụ hữu ích cho nhiều kỹ thuật thiết kế tiêu chuẩn và các công cụ phân tích. Bây giờ chúng ta sẽ chỉ ra vấn đề sắp xếp là như thế nào:

Input: chuỗi n số $<a_1,a_2,...,a_n>$.

Output: hoán vị (sắp xếp lại) $<a'_1,a'_2,...,a'_n>$ của chuỗi input với $a'_1 \leq a'_2 \leq ... \leq a'_n$.

Ví dụ, cho chuỗi input <31,41,59,26,41,58>, thuật toán sắp xếp trả về chuỗi output<26,31,41,58,59>. chuỗi input này được gọi là một yêu cầu của bài toán sắp xếp. Thông thường, yêu cầu của bài toán bao gồm input (thỏa mãn bất cứ các vấn đề được đặt vào bài toán) các tính toán cần thiết để giải quyết bài toán.

Vì có nhiều chương trình sử dụng nó như một bước trung gian cho bài toán, măc khác sắp xếp là một quá trình căn bản của khoa học máy tính. Và vì thế chúng ta có rất nhiếu các thuật toán sắp xếp hiệu quả cho chúng ta dử dụng. Đâu là thuật toán hay nhất để đưa vào ứng dụng phụ thuộc-trong nhiều vấn đề cần xét- số lượng đối tượng cần sắp xếp, mức độ sắp xếp của các đối tượng, mức giới hạn của các giá trị trên đối tượng, kiến trúc của máy tính, và loại các thiết bị lưu trữ được sử dụng: main, bộ nhớ, đĩa thậm chí là băng.

Một thuật toán được coi là đúng nếu như với mỗi trường hợp input, nó sẽ dừng lại với một output đúng. Chúng ta nói rằng một thuật toán đúng giải quyết thành công bài toán được giao. Một thuật toán sai có thể sẽ không dừng lại ở tất cả trường hợp input, hoặc nó sẽ dừng lại với kết quả sai. Ngược lại với những gì bạn nghĩ, thuật toán sai có thể đôi lúc có ích, nếu chúng ta có thể kiểm soát tỷ lệ sai của nó. Chúng ta sẽ thấy ví dụ của thuật toán với tỷ lệ sai được kiểm soát ở Chương 31 khi ta nghiên cứu thuật toán để tìm các số nguyên tố lớn. Thông thường, chúng ta chỉ quan tâm đến các thuật toán đúng.

Một thuật toán có thể được quy định bằng tiếng Anh, như là chương trình máy tính, hoặc thậm chí là một phần cứng đã được thiết kế. Yêu cầu duy nhất là phải cung cấp chính xác quy trình tính toán để người khác có thể tuân thủ theo.

Thuật toán gải quyết những vấn đề gì?

Sắp xếp không chỉ là vấn đề máy tính duy nhất để phát triển thuật toán. (Bạn có thể thấy được điều đó khi xem qua kích thước của cuốn sách này). Ứng dụng thực tế của thuật toán là phổ biến và bao gồm những ví dụ sau:

- Dự án về bộ gen của con người có được bước tiến lớn với mục đích là xác định được toàn bộ 100,000 gen trong DNA của con người, xác định chuỗi với 3 triệu cặp hóa chất cơ sở tạo nên DNA của con người, lưu giữ thông tin này vào các cơ sở dữ liệu và hệ thống những công cụ được phát triển cho phân tích dữ liệu. Mỗi bước đó cần những thuật toán tinh vi. Mặc dù giải pháp cho các vấn đề đó vượt xa phạm vi của cuốn sách này, có thể một vài phương pháp để giải quyết các vấn đề sinh học được lấy ý tưởng từ vài chương trong cuốn sách này, bằng cách ấy cho phép các nhà khoa học hoàn thành nhiệm vụ với các nguồn lực một cách hiệu quả. Tiết kiệm được thời gian, công sức của máy móc và con người, tiền của, nhiều thông tin hơn được lấy từ phòng thí nghiệm.
- Internet cho phép mọi người trên toàn thế giới truy cập và khôi phục lượng lớn thông tin một cách nhanh chóng. Với sự hỗ trợ của các thuật toán thông minh, các trang trên internet có thể vận hành và sử dụng lượng lớn dữ liệu như thế. Ví dụ cho các vấn đề sử dụng các thuật toán cần thiết để tìm đường truyền phù hợp cho dữ liệu (các kỹ thuật để giải quyết các vấn đề xuất hiện trong Chương 24), và sử dụng một hệ thống tìm kiếm nhanh chóng các trang với các thông tin cụ thể (các kỹ thuật liên quan ở Chương 11 và 32).
- Thương mại điện tử cho phép các mặt hàng và dịch vụ được giao dịch và trao đổi điện tử, và nó phụ thuộc vào các thông tin cá nhân như mà số thẻ tín dụng, tài khoản, bản sao kê của ngân hàng. Cốt lõi của các cọng nghệ được dùng trong thương mại điện tử bao gồm mã khóa công khai và chữ ký số (được nói đến trong Chương 31), được dựa trên thuật toán số và lý thuyết số.
- Các doanh nghiệp sản xuất và thương mại đôi lúc cũng cần phân bố các tài nguyên quý hiếm một cách có lợi nhất. Một công ty dầu sẽ mong sao biết được nơi nào đặt thuận lợi để tối đa lợi nhuận. Một ứng cử viên chính trị có thể muốn xác định nơi nào nên đầu tư tiền vào để quảng bá tên tuổi nhằm mục đích khả năng chiến thắng cuộc tranh cử là cao nhất. Hãng máy bay mong muốn được ký hợp đồng với phi hành đoàn sao cho chi phí thấp nhất có thể, mà vẫn đảm bảo mỗi chuyến bay được chu dáo và tuân thủ các quy định của chính phủ về lịch trình của phi hành đoàn. Một nhà cung cấp dịch vụ internet mong muốn xác định được đâu là nơi để tăng cường nguồn tài nguyên để có thể phục vụ khách hàng hiệu quả hơn. Tất cả các ví dụ trên có thể giải quyết bằng lập trình tuyến tính, mà chúng ta sẽ nghiên cứu ở Chương 29.

Mặc dù chi tiết thực hiện của các ví dụ trên vượt quá tầm của cuốn sách này, chúng tôi sẽ đưa ra các kỹ thuật cơ bản được áp dụng cho những vấn đề đó và những gì liên quan. Chúng tôi cũng sẽ chỉ ra cách giải quyết các vấn đề riêng biệt, bao gồm các phần sau:

- Chúng ta được đưa tấm bản đồ với các phần giao nhau được đánh dấu, và chúng tôi mong sao sẽ xác định được lộ trình ngắn nhất với mỗi phần. Số các lộ trình có thể sẽ rất lớn, kể cả khi chúng tôi không cho phép các lộ trình vượt qua nhau. Vậy thì làm thế nào chúng ta có thể chọn ra phần ngắn nhất? Đây, chúng ta thiết kế bản đồ đi (cũng chính là bản đồ đường đi thực tế) như là biểu đồ (sẽ gặp ở Phần VI và Phụ lục B), và mong sao có thể tìm ra con đường ngắn nhất từ đỉnh của đồ thị. Chúng ta sẽ xem xét làm thế nào để giải quyết vấn đề một cách hiệu quả ở Chương 24.
- Chúng ta được đưa 2 chuỗi ký hiệu đã sắp xếp. $X=\langle x_1,x_2,\ldots,x_m\rangle$ và $Y=\langle y_1,y_2,\ldots,y_n\rangle$, và ta muốn tìm dãy chung dài nhất của *X* và *Y*. Một dãy của *X* chỉ là *X* với một vài (có thể là toàn bộ hoặc rỗng) phần tử của nó bị loại bỏ. Lấy ví dụ, một chuỗi của <*A,B,C,D,E,F,G*> có thể là <*B,C,E,G*>. Độ dài cảu chuỗi chung dài nhất cảu X và Y đưa ra vấn đề là hai chuỗi giống nhau ra sao. Ví dụ, nếu 2 chưỡi là sợi các

cặp DNA, sau đó chúng ta xem chúng giống nhau nếu chúng có chuỗi chung dài. Nếu *X* có *m* ký hiệu và *Y* có *n* ký hiệu, *X* và *Y* có 2^m và 2^n chuỗi có thể có tương ứng. Chọn ra tất cả các chuỗi có thể của *X* và *Y* sau đó so sánh chúng lại với nhau có thể mất nhiều thời gian trừ khi m và n rất nhỏ. Chúng ta sẽ thấy ở Chương 15 là làm cách nào sử dụng một lỹ thuật thông dụng gọi là quy hoạch động để giải quyết vấn đề hiệu quả hơn.

- Chúng ta được đưa cho một thết kế cơ khí với một bảng các bộ phận, nơi mà mỗi phần có thể bao gồm các bộ phận khác, và chúng ta cần liệt kê các bộ phận để mỗi bộ phận ấy xuất hiện trước khi các phần sử dụng bộ phận đó. Nếu bản thiết kế bao gồm n bộ phận, và ta có n! cách sắp xếp có thể có, nơi mà n! biểu thị hàm giai thừa. Vì hàm giai thừa tăng nhanh hơn hàm mũ, ta không thể tạo ra từng thứ tự có thể rồi kiểm chứng nó được, bên trong thứ tự đó, mỗi bộ phận xuất hiện trước khi các bộ phận sử dụng nó (trừ khi chúng ta chỉ có một vài bộ phận). Vấn đề này là một trường hợp của sắp xếp tô-pô, và ta sẽ được thấy ở Chương 22 làm thế nào để giải quyết vấn đề này hiệu quả.
- Chúng ta có *n* điểm trên mặt phẳng, và ta muốn tìm bao lồi của những điểm này. Bao lồi là một đa giác lồi nhỏ nhất chứa các điểm này. Trực quan ta thấy, có thể tưởng tượng mỗi điểm được biểu diễn như là một cái đinh trên một tấm bảng. Bao lồi được biểu diễn là cọng dây thun bao quanh các cây đinh. Mỗi cây đinh sung quanh cọng dây thun làm thành một vòng và là đỉnh của bao lồi. (Xem ví dụ hình 33.6 ở trang 1029). Bất kỳ một đỉnh nào trong tập hợp 2^n điểm có thể là đỉnh của bao lồi. Chương 33 đưa ra 2 phương pháp tốt để tìm bao lồi.

Các danh dách này vượt quá giới hạn (bạn có thể đoán được điều này qua trọng lượng của cuốn sách), có hai đặc điểm chung của nhiều vấn đề thuật toán được quan tâm:

1. Có nhiều giải pháp, đa số trong đó là không thể giải quyết bằng tay. Để tìm một cái có thể, hoặc "tốt nhất", khá là thách thức.
2. Có nhiều ứng dụng thực tiễn. Tất cả các vấn đề trên, tìm con đưởng ngắn nhất cho ta ví dụ dễ nhất. Một công ty vận tải, có thể là công ty đường bộ hoặc đường ray, quan tâm đến việc tìm con đường ngắn nhất từ hệ thống đường bộ hoặc đường ray vì đường đi ngắn nhất tốn ít nhân công và nhiên liệu. Hoặc một thiết bị định tuyến Internet có thể tìm đường ngắn nhất từ hệ thống mạng để truyền tin nhắn đi nhanh hơn. Hoặc một người mong muốn được lái xe từ New York tới Boston có thể cần các chỉ dẩn lái từ một trang web phù hợp, hoặc cô ấy muốn dùng GPS trong khi lái xe.

Không phải tất cả vấn đề được giải quyết bằng thuật toán có những giải pháp dễ xác định. Ví dụ, giả sử chúng ta có một bộ các giá trị số tượng trưng cho một tín hiệu, và chúng ta muốn tính được các biến đổi Fourier rời rạc cho các mẫu. Biến đổi Fourier rời rạc chuyển đổi miền thời gian thành miền tần số, tạo ra một bộ các hệ số, do đó chúng ta có thể xác định cường độ của các tần số khác nhau trong các mẫu tín hiệu. Ngoài ra để đặt trung tâm xử lý tín hiệu, biến đổi Fourier rời rạc có cac ứng dụng trong dữ liệu nén và nhân đa thức với số nguyên. Chương 30 cho ta thuật toán hiệu quả, tốc độ của biến đổi Fourier (thường được gọi là FFT), cho vấn đề này, và chương đó cũng phác thảo thiết kế một phần cứng để tính toán FFT.

Cấu trúc dữ liệu

Cuốn sách này cũng chứa các cấu trúc về dữ liệu. Một **cấu trúc dữ liệu** là con đường để lưu giữ và tổ chức dữ liệu tạo điều kiện tiếp cận và sửa đổi. Không một cấu trúc dữ liệu đơn lẻ nào hoạt động tốt với tất cả mục đích, và do đó quan trọng để biết điểm mạnh và gới hạn của một vài loại trong số đó.

Kỹ thuật

Mặc dù bạn có thể sử dụng cuốn sách này như là một cuốn sách "nấu ăn" cho thuật toán, nhưng có thể một ngày mào đó bạn sẽ gặp phải vấn đề cho việc bạn không thể tìm thấy một thuật toán có sẵn một cách dễ dàng (như nhiều bài tập và bài toán trong cuốn sách, làm ví dụ). Cuốn sách sẽ dạy bạn các kỹ thuật về thiết kế và phận tích thuật toán từ đó bạn có thể một mình phát triển thuật toán, cho thấy rằng nó cho kết quả đúng, và hiểu được hiệu quả của nó. Các chương khác nhau tiếp cận khía cạnh khác nhau, như là tìm trung vị và thống kẹ ở Chuong 9, cây bao trùm nhỏ nhất ở Chương 23, xác định lưu lượng tối đa trong mang5o73 Chuong 26. Các chương khác tiếp cận kỹ thuật, như là chia để trị ở Chương 4, quy hoạch động ở Chuong 15, và độphức tạp thuật toán ở Chương 17.

Các bài toán khó

Cuốn sách này phần lớn giới thiệu về các thuật toán hiệu quả. Thước đo về hiệu quả ở đây là tốc độ, thuật toán mất thời gian bao lâu để cho ra kết quả. Có thể có vài vấn đề xảy ra, tuy nhiên, không có thuật toán hiệu quả hoàn toàn. Chương 34 nghiên cứu đến một vài vấn đề thú vị giống thế, được biết như là NP-đầy đủ.

Vì sao NP-đầy đủ thú vị? Thứ nhất, mặc dù chưa có thuật toán hiệu quả cho bài toán NP-đầy đủ được tìm thấy, không một ai từng chứng minh được rằng một thuật toán hiệu quả là không tồn tại. Nói cách khác, không ai biết được rằng có hay không tồn tại một thuật toán hiệu quả cho NP-đầy đủ. Thứ hai, các tập bài toán NP-đầy đủ có đặc tính đáng chú ý do đó nếu có một thuật toán hiệu quả tồn tại, thì sẽ có các thuật toán hiệu quả tồn tại cho từng bài toán. Mối quan hệ này trong các bài toán NP-đầy đủ làm cho việc thiếu các giải pháp hiệu quả giống như trêu ngươi mình. Thứ ba, một vài NP-đầy đủ khá là tương tự nhau, nhưng không giống nhau, đến các bài toán mà ta biết về thuật toán hiệu quả. Các nhà khoa học máy tính thích thú với việc một thay đổi nhỏ trong đề bài toán có thể gây ra sự thay đổi lớn tính hiệu quả của thuật toán phổ biến.

Bạn nên biết về bài toán NP-đầy đủ vì một vài trong số đó thường xuất hiện đột ngột trong các ứng dụng thực tế. Nếu bạn bị gọi để tạo ra một thuật toán hiệu quả cho bài toán NP-đầy đủ, có thể bạn sẽ dành nhiều thời gian cho một nghiên cứu không có kết quả. Nếu bạn có thể chỉ ra bìa toán này là NP-đầy đủ, thì bạn có thể dành thời gian để phát triển một thuật toán đưa ra kết quả tốt thay vì một kết quả hoàn hảo nhất.

Như một ví dụ cụ thể, xem xét một công ty phân phối với một kho trung tâm. Mỗi ngày, người ta chất hàng phân phối vào trong kho và gửi đi các hàng hóa tới các địa chỉ cần nhận. Tới cuối ngày, mỗi xe tải phải trở lại kho để có thể chuẩn bị sẵn sàng cho ngày tiếp theo. Để giảm gái cả, công ty muốn chọn thứ tự cho các điểm dừng mà mang lại quãng đường đi ngắn nhất cho mỗi xe. Vấn đề này được biết như là "bài toán người bán hàng", và nó là một NP-đầy đủ. Không có thuật toán hoàn hảo cho vấn đề này. Dưới các giả thuyết nhất định, tuy nhiên, ta biết rằng thuật toán hiệu quả cho kết quả đường đi không quá xa so với cái nhỏ nhất. Chương 35 thảo luận về "thuật toán xấp xỉ".

Tính song song

Qua từng ấy năm, chúng ta có thể đếm trên tốc độ của bộ xử lý đồng hồ tăng theo một tỷ lệ không đổi. Các giới hạn vật lý thể hiện rào cản trong việc tăng tốc độ đồng hồ, tuy nhên: vì mật độ công suất tăng nhanh theo tốc độ đồng hồ, các con chip sẽ có nguy cơ bị cahy3 một khi tốc độ đồng hồ của chúng nhanh đến mức nào đó. Để biểu diễn nhiều hơn các phép tính trên giây, vì vì thế, các con chip được thiết kế để chứa không chỉ một mà nhiều "lõi" xử lý. Chúng ta có thể so sánh các máy tính đa lõi với một số máy tính tuần tự sử dụng chip đơn; nói cách khác, chúng là một dạng của "máy tính song song". Để khám phá ra hiệu suất tốt nhất cho máy tính đa lõi, chúng ta cần thiết kế những thuật toán song song trong đầu. Chương 27 giới thiệu một mẫu của thuật toán "đa luồng", với thế mạnh của đa lõi. Mẫu này có lợi thế từ một quan điểm lý thuyết, và nó tạo dựng nền cơ bản của một vài chương trình máy tính thành công, bao gồm cả chương trình vô địch về cờ vua.

1.2 Thuật toán như là một công nghệ

Giả sử máy tính xử lý cực nhanh và bộ nhớ là vô hạn. Liệu bạn có lý do nào để học về thuật toán không? Câu trả lời là có, không có bất kỳ lý do nào khác hơn việc bạn có thể biểu diễn giải pháp của mình một cách triệt để và hơn thế nữa nếu nó ra kết quả đúng.

Nếu máy tính xử lý cực nhanh, bất kỳ phương pháp đúng nào cũng giải quyết được bài toán. Bạn có thể muốn hệ thống xử lý được trong giới hạn thực hành các kỹ năng kỹ thuật phần mềm (lấy ví dụ, hệ thống xử lý của bạn nên được thiết kế tốt và dẫn chứng), nhưng bạn sẽ dùng bất cứ phương pháp nào là dễ nhất để hoàn thành nó.

Tất nhiên, máy tính có thể rất nhanh, nhưng chúng không xử lý cực nhanh được. Và bộ nhớ có thể không đắt, nhưng nó không miễn phí. Do đó việc tính toán bị giới hạn bơi tài nguyên, và cả không gian cho bộ nhớ. Bạn nên sử dụng tài nguyên một cách thông minh, và thuật toán là phương tiện hiệu quả cho thời gian và không gian sẽ giúp cho bạn làm điều đó.

Tính hiệu quả

Các thuật toán khác nhau được tạo ra để giải quyết cùng một vấn đề thường khác nhau rất nhiều trong tính hiệu quả của chúng. Sự khác nhau đó có thể có nhiều ý nghĩa hoặc khác nhau hơn tùy thuộc vào phần cứng và phần mềm.

Như là ví dụ, ở Chương 2, chúng ta sẽ tìm hiểu 2 thuật toán sắp xếp. Thứ nhất, được biết như ***insertion sort*** (sắp xếp chèn), mất khoảng thời gian bằng c_1n^2 để sắp xếp n phần tử, với c_1 là hằng số không phụ thuộc vào n. Do đó, nó mất thời gian tỉ lệ với n^2. Thứ hai, ***merge sort*** (sắp xếp hợp), mất thời gian bằng với c_2n lg n, với lg n là $\log_2$ n và c_2 là một hằng số khác mà nó không phụ thuộc vào n. Insertion sort thường có phần hằng số nhỏ hơn merge sort, vì thế $c_1 < c_2$. chúng ta sẽ thấy các hằng số ít ảnh hưởng đến thời gian chạy hơn là kích thước của n. Hãy bắt đầu viết thời gian chạy của insertion sort là $c_1n.n$ và merge sort là c_2n.lg n. Chúng ta sẽ thấy rằng insertion sort có n trong thời gian chạy của nó, merge sort thì có lg n, và nhỏ hơn rất nhiều. (ví dụ, nếu n=1000, lg n xấp xỉ là 10, và khi n bằng một triệu, lg n chỉ xấp xỉ có 20). Mặc dù insertion sort thường chạy nhanh hơn so với merge sort đối với input có kích thước nhỏ, một khi nhập kích thước của n đủ lớn, lợi thế của merge sort giữa lg n và n sẽ hơn phần bù lại của sự khác nhau về hằng số. Bất kể c_1 nhỏ hơn c_2 bao nhiêu, luôn luôn có giao điểm mà merge sort sẽ nhanh hơn.

Ví dụ cụ thể, chúng ta hãy cho máy tính chạy nhanh hơn (máy A) chạy insertion sort so với máy chạy chậm hơn (máy B) chạy merge sort. Mỗi máy sắp xếp một mảng 10 triệu số. (mặc dù 10 triệu số có thể là

rất nhiều, nếu các số là một số nguyên tám byte, do đó input sẽ nhập khoảng 80 megabytes, sẽ vừa với bộ nhớ của cả một máy laptop không đắt tiền). Giả sử rằng máy tính A thực hiện 10 tỷ lệnh mỗi giây (nhanh hơn nhiều so với các máy tính xư lý tuần tự ở thời điểm cuốn sách này được viết) và máy tính B thực hiện 10 triệu lệnh mỗi giây, vì thế máy tính A chạy nhanh hơn 1000 lần so với máy B về năng lực của máy. Để làm cho sự khác nhau thêm kịch tính, giả sử rằng lập trình viên lành nghề nhất thế giới lập trình insertion sort bằng ngôn ngữ máy cho máy A, và kết quả cần $2n^2$ lệnh để sắp xếp n số. Giả sử xa hơn rằng chỉ cho lập trình viên trung bình hoàn thiện merge sort, sử dụng một ngôn ngữ cấp cao với trình biên dịch không hiệu quả, với kết quả tốn $50n \lg n$ lệnh. Để sắp xếp 10 triệu số, máy A cần:

$$\frac{2.(10^7)^2 \text{lệnh}}{10^{10} \text{lệnh/giây}} = 20{,}000 \text{ giây (hơn 5.5 giờ).}$$

trong khi máy B cần:

$$\frac{50.10^7 \text{ lệnh}}{10^7 \text{ lệnh/giây}} \approx 1163 \text{ giây (ít hơn 20 phút).}$$

Bằng việc sử dụng một thuật toán mà có thời gian chạy tăng ít hơn, mà ngay cả khi một trình biên dịch tồi, máy B chạy nhanh hơn 17 lần so với máy A! Lợi thế của merge sort thậm chí càng rõ hơn khi sắp xếp 100 triệu số: khi mà insertion sort mất hơn 23 ngày, merge sort mất dưới bốn giờ. Tóm lại, nếu như kích thước của bài toán tăng lên, đồng nghỉa với lợi thế của merge sort tăng lên.

Thuật toán và các công nghệ khác

Ví dụ trên chỉ ra rằng chúng ta nên cân nhắc đến việc sử dụng thuật toán, như là phần cứng của máy tính, đó là công nghệ. Toàn bộ hiệu suất hệ thống phụ thuộc vào việc chọn ra thuật toán hiệu quả cũng như việc chọn phần cứng tốt. Như việc lợi thế về tốc độ đang dần tạo ra một công nghệ máy tính mới, tất hiên nó cũng được làm từ thuật toán nữa.

Bạn có thể băn khoăn liệu thuật toán có thật sự quan trọng đối với các máy tính hiện nay hay là các kỹ thuật tiên tiến khác, như là

- Kiến trúc máy tính tiên tiến và công nghệ chế tạo,
- Để sử dụng, trực quan, giao diện đồ họa cho người dùng (GUIs),
- Hệ thống hướng đối tượng,
- Công nghệ wed tích hợp, và
- Mang lưới mạng tốc độ cao, có dây và không dây.

Câu trả lời là có. Mặc dù một vài ứng dụng không rõ ràng cần thuật toán ở mức ứng dụng (như là một vài cái đơn giản, ứng dụng web), thường như thế. Lấy ví dụ, cân nhắc một dịch vụ web mà xác định làm cách nào để di chuyển từ điểm này đến điểm kia. Việc hoàn thiện có thể tùy thuộc vào tốc độ của phần cứng, một giao diện đồ họa cho người dùng, mạng lưới mạng bao phủ rộng, và cũng có thể là hướng đối tượng. Tuy nhiên, nó cũng cần thuật toán cho một vài hoạt động nhất định, như là tìm kiếm tuyến đường (có thể dùng thuật toán đường đi ngắn nhất), vẽ lại bản đồ, và thêm vào các địa chỉ.

Hơn thế nữa, kể cả một ứng dụng không cần nội dung thuật toán cũng dựa vào thuật toán ở mức độ ứng dụng. Liệu một ứng dụng có dựa vào phần cứng nhanh không? Phần cứng sử dụng thuật toán. Liệu ứng dụng có cần tới giao diện đồ họa không? Bất cứ một thiết kế GUI nào cũng dựa vào thuật toán. Liệu ứng dụng có dựa vào mạng lưới mang không? Định tuyến một mạng nào cần nhiều tới thuật toán. Liệu ứng dụng có nên viết bằng các ngôn ngữ khác thay vì ngôn ngữ máy ay không? Nó cần trình biên dịch để xử lý, trình phiên dịch, hoặc trình lắp ráp, và tất cả các cái gì sử dụng các kiến thức phổ thông của thuật toán. Thuật toán là cốt lõi của công nghệ được dùng cho các máy tính hiện hành.

Hơn thế nữa, với năng lực ngày càng tăng cảu máy tính, chúng ta có thể sử dụng chúng để giải quyết các bài toán lớn hơn bao giờ hết. Như chúng ta đã thấy ở trên so sánh giữa insertion sort và merge sort, nếu kích thước bài toán càng lớn thì sự khác biệt về hiệu suất giữa các thuật toán đặc biệt nổi bật.

Có một nền tảng cững chắc về kiến thức và kỹ thuật của thuật toán là một trong những đặc trưng tách biệt lập trình viên có tay nghề với người mới. Với kỹ thuật máy tính hiện đại, bạn có thể, bạn có thể hoàn thành một vài việc dù không biết nhiều về thuật toán, nhưng với một lý lịch tốt về thuật toán, bạn có thể làm được nhiều hơn, nhiều hơn nữa.

CHAPTER 2 – Bắt đầu

Chương này sẽ giúp bạn làm quen với một khuôn khổ đó là thông qua cuốn sách để suy nghĩ về thiết kế và phân tích thuật toán. Nó trong thật huyền bí nhưng nó có nhiều tài liệu tham khảo mà chúng tôi giới thiệu trong chương 3 và chương 4. (Nó cũng bảo gồm một số kết quả mà ở phụ lục A đã tìm ra cách giải quyết)

Chúng ta bắt đầu bằng cách kiểm tra thuật toán "sắp xếp chèn" (insertion sort) để giải quyết vấn đề sắp xếp, phân loại được giới thiệu ở chương 1. Chúng tôi định nghĩa một loại mã giả bạn đã quen thuộc nếu đã thực hành trên chương trình máy tính và chúng tôi sử dụng nó để trình bày cụ thể các thuật toán của bạn. Sau khi chỉ định thuật toán sắp xếp chèn, chúng tôi cho rằng đây là cách sắp xếp chính xác, chúng tôi bắt đầu phân tích thời gian chạy của nó. Cuộc phân tích đưa ra một chú thích: tập trung vào thời gian tăng lên như thế nào với số lượng phần tử được sắp xếp. Sau cuộc thảo luận về thuật toán sắp xếp chèn, chúng tôi giới thiệu phương pháp chia và chiếm để thiết kế thuật toán và sử dụng nó để phát triển thuật toán mới "hợp nhất sắp xếp" (merge sort).

2.1 Sắp xếp chèn

Như đã thông tin, sắp xếp chèn giải quyết các vấn đề về sắp xếp được giới thiệu ở chương 1.

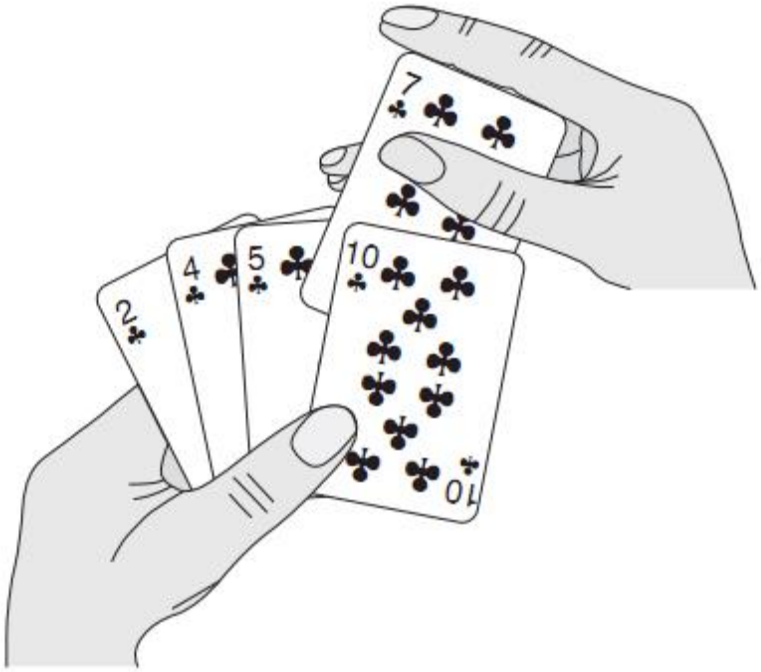
Đầu vào: Một loạt N số cho trước <a1, a2, ..., aN>

Đầu ra: Một dãy hoán vị <a1', a2', ..., aN'> từ các thông số đầu vào (với a1' <= a2' <= ... <= aN')

Các con số mà chúng ta muốn sắp xếp cũng được biết đến như là các khóa. Mặc dù theo khái niệm chúng ta đang sắp xếp một dãy nhưng đầu vào đến với chúng ta dưới dạng một mảng với các phần tử.

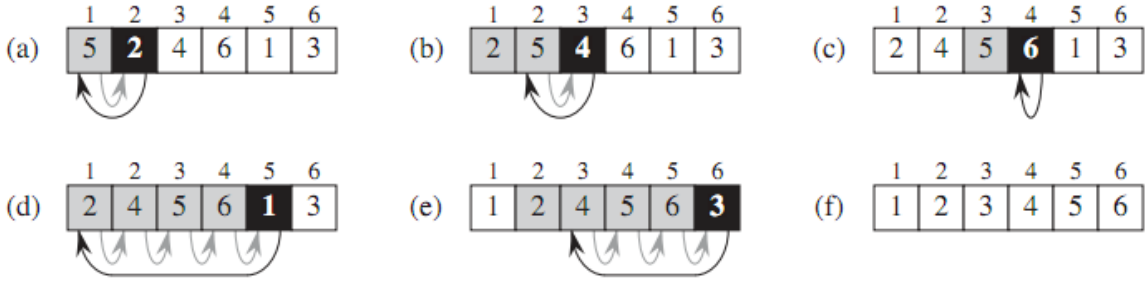
Trong cuốn sách này, chúng ta sẽ mô tả các thuật toán như các chương trình được viết bằng mã giả cái mà đã khá quen thuộc với C, C++, Java, Python hay Pascal. Nếu bạn đã được giới thiệu với bất kỳ ngôn ngữ nào trong số những ngôn ngữ này, bạn sẽ tương đối ổn trong việc đọc thuật toán. Chúng tôi sẽ cố gắn để biểu hiện rõ ràng và súc tích thuật toán. Thường thì tiếng Anh là cách mà thể hiện rõ ràng nhất, thế nên đừng ngạc nhiên khi gặp một từ hoặc cụm từ hoặc câu tiếng Anh được nhúng trong mã thực. Sự khác biệt về mã thực và mã giả là: mã giả thường không liên quan đến các vấn đề về kỹ thuật phần mềm. Các vấn đề về trừu tượng dữ liệu, mô đun và xử lý lỗi thường bị bỏ qua để truyền đạt bản chất của thuật toán một cách ngắn gọn hơn.

Chúng ta bắt đầu với thuật toán trên, đó là một thuật toán hiệu quả để phân loại một số lượng nhỏ các phần tử. Sắp xếp chèn hoạt đồng tựa như cách người ta sắp xếp quân bài trên tay. Chúng ta bắt đầu với một bàn tay trống đồng thời các lá bài được nằm sấp trên bàn. Chúng ta loại 1 lá bài trên bàn và giữ trên tay. Để tìm vị trí chính xác cho một thẻ, chúng tôi so sánh nó với từng thẻ đã có trong tay, từ phải sang trái, như minh họa trong Hình 2.1. Lúc nào cũng vậy, các thẻ được giữ trong tay trái được sắp xếp, và các thẻ này ban đầu là các thẻ trên cùng của xấp bài trên bàn



Hình 2.1 Sắp xếp bài trên tay bằng sắp xếp chèn

Chúng tôi trình bày mã giả cho sắp xếp chèn là một thủ tục gọi là INSERTION SORT, lấy tham số trong mảng A[1..n] chứa chuỗi số với độ dài n cần được sắp xếp. (trong đoạn code, số n được chỉ như là độ dài của A). Thuật toán sắp xếp các số input ngay tại chỗ: tái sắp xếp các số trong mảng A, với tối đa số không đổi được lưu bên ngoài mảng bất cứ lúc nào. input của mảng chứa output là chuỗi đã được sắp xếp xem như thủ tục sắp xếp đã hoàn tất.



Hình 2.2 Quá trình làm việc của INSERTION-SORT trên mảng A = <5, 2, 4, 6, 1, 3>. Các chỉ số phần tử được đánh dấu ở trên các ô thành phần của mảng, và các giá trị được lưu trữ trong các vị trí mảng xuất hiện trong các hình chữ nhật này. Từ (a) đến (e) là các lần lặp để sắp xếp mảng. Trong mỗi vòng lặp, hình chữ nhật màu đen giữ các đối tượng cần so sánh với các đối tượng trước nó đã được sắp xếp (ô bóng) trong phép thử ở dòng 5. Các mũi tên bóng mờ hiển thị giá trị mảng di chuyển một vị trí sang bên phải (trong dòng 6) và các mũi tên màu đen cho biết vị trí đối tượng đang xét di chuyển đến nới thích hợp (trong dòng 8). Sau cùng, mảng đã được sắp xếp.

```
INSERTION-SORT
1       for j = 2 to A.length
2               key = A[j]
3               // Chèn A[j] vào phần đã sắp xếp A[1 .. j-1]
4               I = j -1
5               while i>0 and A[i]>key
6                       A[i+1] = A[i]
7                       I=i-1
8               A[i+1] = key
```

Sự bất biến và chính xác của Insertion-sort

Hình 2.2 cho thấy thuật toán này hoạt động như thế nào đối với mảng A = <5, 2, 4, 6, 1, 3>. Chỉ số j cho biết vị trí hiện tại đang được thực hiện ở trên tay. Vào đầu mỗi lần lặp của vòng lặp for (của chỉ số j), có thể ngầm hiểu rằng đã có A[1..j-1] phần tử đã sắp xếp trên tay và A[j+1..N] lá bài thì đang nằm trên bàn. Chúng tôi muốn chính thức trình bày thuộc tính của mảng A[1..j-1] như là một vòng lặp bất biến: Vào đầu mỗi lần lặp của vòng lặp for ở các dòng 1-8, các phần tử trong mảng A[1..j-1] có các thành phần là các phần tử ban đầu trước khi sắp xếp, nhưng ở đây nó đã được sắp xếp.

Chúng tôi phải cho thấy 3 điều sau trong vòng lặp bất biến:

Khởi tạo: Phải đúng trước lần lặp đầu tiên của vòng lặp

Sự duy trì: Nếu nó đúng trước khi lặp lại của vòng lặp thì nó vẫn còn đúng trước lần lặp tiếp theo

Sự kết thúc: Khi vòng kết thúc, biến bất biến cho ta một thuộc tính hữu ích giúp cho thấy rằng thuật toán là chính xác

Chúng ta sẽ cũng nhìn xem những thuộc tính trên có ảnh hưởng gì đối với Insertion-sort

Khởi tạo: Chúng ta bắt đầu vòng lặp bất biến trước vòng lặp đầu tiên, j = 2. Ban đầu mảng A[1..j-1] sẽ lưu trữ phần tử A[1] “nguyên thủy”

Sự duy trì: Tiếp theo, chúng ta giải quyết thuộc tính thứ 2: cho thấy sự duy trì của vòng lặp bất biến. Thông thường, phần thân của vòng lặp for làm việc bằng cách thông qua các phần tử A[j-1], A[j-2], A[j-3] … và cứ tiếp tục đến khi A[j] tìm được vị trí thích hợp. Mảng sau cùng A[1..j] bao gồm các phần tử nguyên thủy cũ, đã được sắp xếp lại.

Sự kết thúc: Cuối cùng, chúng ta kiểm tra xem điều gì sẽ xảy ra khi vòng lặp kết thúc. Điều kiện dừng của vòng lặp for là j > A.length = n. Quan sát thấy rằng mảng A[1..n] đã được sắp xếp

Quy ước về mã giả

Ta sẽ tuân theo một số quy ước trong mã nguồn giả.

Cấu trúc lặp while, for, repeat – until và cấu trúc điều kiện if-else có một số điểm chung trong C, C++, Java, Python, Pascal. Trong cuốn sách này, biến đếm của vòng lặp được lưu lại giá trị sau khi vòng lặp xuất hiện không như một số trường hợp trong C++, Java và Pascal. Vì thế, ngay sau khi vòng lặp for chạy, giá trị của biến đếm tăng theo số vòng lặp. Chúng ta sử dụng tính chất này để thảo luận về tính đúng đắn ở phần sắp xếp chèn.

Các ký tự "//" là một ghi chú

Các biến (như I, j, và key) là biến cục bộ trong hàm. Chúng ta sẽ không sử dụng biến toàn cục mà không có một chỉ thị rõ ràng.

Chúng ta truy cập các phần tử của mảng bằng tên viết hoa cùng với các chỉ số trong dấu ngoặc vuông. Ví dụ A[i] biểu diễn cho phần tử thứ I của mảng A. Lưu ý dấu ".." biểu diễn nhiều giá trị liên tục của mảng. Vì thế A[1..j] biểu diễn mảng con của A bao gồm j phần tử A[1], A[2], …, A[j].

Chúng ta tổ chức các dữ liệu trong lớp bao gồm các thuộc tính. Ta truy cập những thuộc tính liên quan đang sử dụng cú pháp tìm kiếm trong nhiều ngôn ngữ lập trình hướng đối tượng: tên đối tượng kế đến là dấu chấm, theo sau là tên thuộc tính. Ví dụ, ta xét một mảng các đối tượng với thuộc tính là length biểu diễn số phần tử mà nó chứa. Để chỉ định số phần tử của mảng A, ta dùng A.length.

Ta xét một biến tượng trưng cho mảng hoặc đối tượng như một con trỏ trỏ tới dữ liệu đại diện cho mảng hoặc đối tượng. Đối với tất cả thuộc tính f của đối tượng x, thiết lập y = x bởi y.f bằng x.f. Hơn nữa nếu chúng ta đặt x.f = 3, sau đó không chỉ x.f = 3 mà còn y.f = 3. Nói cách khác, x và y trỏ tới cùng 1 đối tượng sau khi gán y = x.

Đôi khi, một con trỏ sẽ không đề cập đến bất cứ vật gì cả. Trong trường hợp này, ta cho nó giá trị đặc biệt NIL.

Lỗi error chỉ ra rằng lỗi xảy ra vì điều kiện đã sai đối với thủ tục đã được gọi. Thủ tục gọi là trách nhiệm xử lý lỗi, vì vậy chúng tôi không chỉ ra hành động cần thực hiện.

2.2 Phân tích thuật toán

Phân tích một thuật toán có nghĩa là dự đoán các nguồn cấp mà thuật toán yêu cầu. Đôi khi, các tài nguyên như bộ nhớ, băng thông truyền thông, hoặc phần cứng máy tính là mối quan tâm chính, nhưng hơn hết thì thời gian tính toán là cái mà chúng ta muốn đo lường. Nói chung, bằng cách phân tích một số thuật toán ứng với một vấn đề, chúng ta có thể xác định được một cách hiệu quả nhất. Phân tích như vậy có thể chỉ ra nhiều hơn một ứng cử viên khả thi, nhưng chúng ta thường có thể loại bỏ một số thuật toán kém trong tiến trình.

Trước khi có thể phân tích một thuật toán, chúng ta phải có một mô hình công nghệ triển khai mà chúng ta sẽ sử dụng, bao gồm một mô hình cho các nguồn lực của công nghệ đó và chi phí của chúng. Đối với hầu hết sách này, chúng ta sẽ giả sử một bộ xử lý chung chung, RAM, và hiểu rằng các thuật toán của chúng tôi sẽ được thực hiện như các chương trình máy tính.

Nói đúng ra, chúng ta nên xác định chính xác các hướng dẫn của mô hình RAM và chi phí của chúng. Để làm như vậy, tuy sẽ là tẻ nhạt nhưng sẽ mang lại một ít hiểu biết về thiết kế thuật toán và phân tích. Tuy nhiên, chúng ta phải cẩn thận không lạm dụng mô hình RAM.

Các kiểu dữ liệu trong mô hình RAM là con trỏ kiểu số nguyên và kiểu số thực. Mặc dù chúng ta thường không quan tâm đến các điều trên nhưng trong một số ứng dụng, tính rõ ràng rất quan trọng.

Ngoài các chỉ lệnh như cộng, trừ, nhân, chia, làm tròn…, trong thực tế các máy tính còn có một vài chỉ lệnh khác, đại diên cho một vùng màu xám trong mô hình RAM. Ví dụ, phép lũy thừa có phải là một chỉ lệnh với thời gian cố định? Trong trường hợp tổng quát thì câu trả lời là không, cần một vài chỉ lệnh để tính x^y khi x và y là hai số thực. Tuy nhiên, trong một vài tình huống giới hạn thì lũy thừa là một toán tử với thời gian thực thi cố định. Nhiều máy tính có phép dịch trái bit (shift left), tức trong thời gian cố định dịch các bit của một số nguyên sang trái k đơn vị. Ở hầu hết máy tính, dịch trái bit của số nguyên một đơn vị đồng nghĩa với việc nhân hai. Vì thế dịch trái bit k đơn vị đồng nghĩa với việc nhân số nguyên đó với 2^k. Do đó những kiểu máy tính như vậy có thể tính 2^k với một chỉ lệnh thời gian cố định bằng cách dịch trái bit k đơn vị, miễn là k không lớn hơn số bit trong một mẫu tính toán. Chúng tôi sẽ cố gắng tránh các vùng màu xám như vậy trong mô hình RAM, nhưng chúng ta vẫn có thể tính 2^k như là một toán tử cố định thời gian khi k là một số nguyên dương đủ nhỏ.

Phân tích một thuật toán, ngay cả khi đó là một thuật toán đơn giản, vẫn luôn là một thách thách thức. Các công cụ toán học cần thiết có thể bao gồm cả tổ hợp, lý thuyết xác suất, kỹ năng đại số và khả năng xác định số hạng quan trọng nhất trong một công thức. Do cách ứng xử của một thuật toán có thể khác nhau với từng bộ thử nên chúng ta cần một biện pháp để tóm tắt cách ứng xử thành các công thức đơn giản và dễ hiểu.

Dù ta thường chỉ chọn mô hình một máy để phân tích một thuật toán nào đó, thì ta vẫn phải đối mặt với nhiều lựa chọn khi quyết định cách diễn tả quá trình phân tích. Chúng ta cần một biện pháp diễn tả đơn giản để viết và thao tác, nêu các đặc tính quan trọng về yêu cầu tài nguyên của một thuật toán, và hủy bỏ những chi tiết dài dòng.

Phân tích sắp xếp chèn

Thời gian thực hiện bởi thủ tục sắp xếp chèn phụ thuộc vào đầu vào: việc sắp xếp một nghìn số mất nhiều thời gian hơn so với việc sắp xếp ba số. Nói chung, thời gian thực thi một thuật tăng theo kích thước dữ liệu đầu vào, vì vậy có thể xem thời gian thực thi một chương trình như là một hàm phụ thuộc vào kích thước đầu vào. Nói chung, thời gian thực thi một thuật tăng theo kích thước dữ liệu đầu vào, vì vậy có thể xem thời gian thực thi một chương trình như là một hàm phụ thuộc vào kích thước đầu vào. Để làm như vậy, chúng ta cần phải cẩn thận hơn trong việc sử dụng thuật ngữ "thời gian chạy" và "kích thước đầu vào".

Ý tưởng tốt nhất cho kích thước đầu vào phụ thuộc vào vấn đề đang được nghiên cứu. Nếu đầu vào cho một thuật toán là một đồ thị, kích thước đầu vào có thể được mô tả bằng số các đỉnh và cạnh của đồ thị. Chúng tôi sẽ chỉ ra biện pháp kích thước đầu vào nào đang được sử dụng ứng với mỗi vấn đề chúng tôi nghiên cứu.

Thời gian chạy của một thuật toán trên một đầu vào cụ thể là số hoạt động nguyên thủy hoặc "bước" được thực hiện. Rất thuận tiện để xác định khái niệm bước để nó có thể độc lập với máy móc nhất có thể. Trong thời điểm này, chúng ta hãy chấp nhận quan điểm dưới đây. Cần phải có một khoảng thời gian nhất định để thực hiện mỗi dòng mã giả của chúng ta. Một dòng có thể mất một khoảng thời gian khác với một dòng khác, nhưng chúng ta sẽ giả định rằng mỗi lần thực thi của dòng thứ i cần thời gian ci,

trong đó ci là một hằng số. Quan điểm này phù hợp với mô hình RAM, và nó cũng phản ánh cách mã giả được thực hiện trên hầu hết các máy tính thực tế.

Chúng tôi bắt đầu trình bày về thời gian chạy của thủ tục sắp xếp chèn với chi phí của mỗi lần thực hiện câu lệnh. Cứ mỗi lần j = 2,3 … n, với n = A.length, ta gọi t_j biểu thị số lần lặp lại vòng lặp trong dòng 5 được thực hiện cho giá trị j. Khi vòng lặp for hoặc while thoát ra theo cách thông thường (nghĩa là do điều kiện trong vòng lặp), chúng tôi giả định rằng nhận xét không phải là các câu lệnh thực thi, và vì vậy chúng không mất thời gian

INSETION SORT (A)

Dòng	Code (mã)	Chi phí	Thời gian
1	for j = 2 to A.length	C_1	n
2	key = A[j]	C_2	n-1
3	// Thêm A[j] vào mảng đã sắp xếp A[1.. j-1]	0	n-1
4	i = j -1	C_4	n-1
5	while i > 0 and A[i] > key	C_5	$\sum_{j=2}^{n} tj$
6	A[i+1] =A[i]	C_6	$\sum_{j=2}^{n} (tj-1)$
7	i = i – 1	C_7	$\sum_{j=2}^{n} (tj-1)$
8	A[i+1] = key	C_8	n-1

Thời gian chạy của thuật toán là tổng thời gian chạy cho mỗi câu lệnh thực hiện. Sẽ có các bước để thực hiện và thực hiện n lần sẽ đóng góp ci vào tổng thời gian chạy. Để tính T(n), thời gian chạy của thuật toán trên dữ liệu vào với n giá trị, chúng ta tính tổng các kết quả của phần phí và thời gian.

$T(n) = c_1n + c_2(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} tj + c_6\sum_{j=2}^{n}(tj-1) + c_7\sum_{j=2}^{n}(tj-1) + c_8(n-1)$

Ngay cả đối với đầu vào của một kích thước nhất định, thời gian chạy của thuật toán có thể phụ thuộc vào đầu vào của kích thước đó được đưa ra. Ví dụ ở bài toán sắp xếp chèn, trường hợp tốt nhất xảy ra nếu mảng đã được sắp xếp. Đối với mỗi j = 2, 3, ..., n, chúng ta tìm phần tử A[j] < key ở dòng 5 khi mà i có giá trị ban đầu là j -1. Do đó t_j = 1 khi j =2, 3, …, n và thời gian chạy tốt nhất là

$T(n) = c_1n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) = (c_1 + c_2 + c_4 + c_5 + c_8)*n – (c_2 + c_4 + c_5 + c_8)$

Chúng ta có thể biểu diễn thời gian chạy này là an+b với a, b là hằng số phụ thuộc vào chi phí c_i nó là một hàm tuyến tính của n.

Nếu mảng đó được sắp xếp theo thứ tự ngược lại - tức là theo thứ tự giảm dần thì nó là trường hợp xấu nhất. Chúng ta phải so sánh mỗi phần tử A[j] với mỗi phần tử với mỗi phần tử trong toàn bộ mảng con A[1 … j-1] và vì thế, t_j = j với j = 2,3 … n. Chú ý

$$\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$$

Và

$$\sum_{j=2}^{n} (j-1) = \frac{n(n-1)}{2}$$

chúng ta thấy rằng trong trường hợp xấu nhất, thời gian chạy của INSERTION-SORT là

$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\frac{n(n+1)}{2} - 1 + c_6\frac{n(n-1)}{2} + c_7\frac{n(n-1)}{2} + c_8(n-1)$

$= \frac{c^5+c^6+c^7}{2}.n^2 + (c_1 + c_2 + c_3 + \frac{c^5-c^6-c^7}{2} + c_8).n - (c_2 + c_4 + c_5 + c_8)$

Chúng ta có thể biểu diễn thời gian chạy xấu nhất bởi an^2+bn+c với các hằng số a, b và c lần nữa phụ thuộc vào chi phí thực hiện ci; nó là hàm số bậc hai theo n.

Điển hình, như trong sắp xếp chèn, thời gian chạy của thuật toán được cố định cho một bộ dữ liệu vào, mặc dù trong chương sau chúng ta sẽ thấy những thuật toán thú vị gọi là ngẫu nhiên mà những trạng thái của nó có thể biến đổi đối với một bộ dữ liệu vào cố định.

Phân tích khả năng xấu nhất và trung bình

Trong phân tích sắp xếp chèn của chúng ta có thể thấy ở cả hai trường hợp tối ưu, là ở đó dữ liệu vào đã được sắp xếp, và trường hợp xấu nhất dự liệu vào đã bị đảo ngược thứ tự. Trong phần còn lại của sách này, mặc dù chúng ta sẽ tập trung vào việc tìm ra thời gian chạy của trường hợp xấu nhất, đó là thời gian chạy lâu nhất cho bộ dữ liệu với bất kỳ giá trị n. Chúng ta đưa ra 3 lí do cho hướng này.

Thời gian chạy xấu nhất của một thuật toán cho chúng ta một giới hạn trên về thời gian chạy cho bất kỳ bộ dữ liệu vào nào. Cho biết nó bảo đảm rằng thuật toán sẽ không bao giờ thực thi lâu hơn được nữa. Chúng ta không cần làm các ước chừng về thời gian chạy và hy vọng nó sẽ không bao giờ nhận khả năng xấu hơn nữa.

Trong một vài thuật toán, khả năng xấu nhất thường xảy ra. Ví dụ, trong tìm kiếm một phần thông tin của cơ sở dữ liệu, khả nawg xấu nhất của thuật toán tìm kiếm sẽ thường xảy ra khi thông tin không có trong cơ sở dữ liệu. Trong một số ứng dụng, việc tìm kiếm thông tin không có trong dữ liệu có thể thường xuyên.

"Trường hợp trung bình" thường là trường hợp xấu nhất. Giả sử chúng ta ngẫu nhiên chọn n số và áp dụng sắp xếp chèn. Mất bao lâu để xác định A[1…j-1] để chèn tiếp A[j]? Ở mức độ trung bình, phần nữa phần tử trong A[1..j-1] là bé hơn A[j], còn lại thì lớn hơn. Tuy nhiên khi chúng ta kiểm tra phân nữa mảng A[1..j-1] thì $t_j = j/2$. Trường hợp tồi tệ nhất của kết quả thời gian chạy hóa ra là một hàm bậc hai của kích thước đầu vào, giống như thời gian chạy tồi tệ nhất.

Trong một số trường hợp cụ thể, chúng ta sẽ quan tâm đến thời gian chạy của một thuật toán trung bình; chúng ta sẽ thấy kỹ thuật phân tích xác suất được áp dụng cho các thuật toán khác nhau trong suốt cuốn sách này. Phạm vi phân tích các trường hợp trung bình là có hạn, bởi vì nó có thể không rõ ràng. Thông thường, chúng ta sẽ giả định rằng tất cả các đầu vào của một kích thước nhất định đều

có khả năng tương đương nhau. Trong thực tế, giả định này có thể bị vi phạm, nhưng đôi khi chúng ta có thể sử dụng một thuật toán ngẫu nhiên, đưa ra các lựa chọn ngẫu nhiên, để cho phép phân tích xác suất và mang lại thời gian chạy dự kiến. Chúng ta khám phá các thuật toán ngẫu nhiên trong chương 5 và trong một số chương tiếp theo khác.

Cấp tăng trưởng

Ta đã dùng vài kiểu trừu tưởng giản lược để tạo thuận lời cho tiến trình phân tích thủ tục INSERTION SORT. Trước tiên, ta đã bỏ qua mức hao phí thực tế của từng câu lệnh, dùng các hằng ci để biểu thị các hao phí này. Sau đó, ta nhận thấy thậm chí các hằng này còn cho ta nhiều chi tiết hơn mức cần thiết thực tế: thời gian thực hiện ca xấu nhất là an^2+bn+c với a, b là hằng số và c phụ thuộc vào cách hao phí ci. Như vậy, ta đã bỏ qua không những các hao phí câu lệnh thực tế mà cả các hao phí trừu trượng ci.

Giờ đây, ta sẽ xem xét một kiểu trừu tưởng giản lược khác. Đó là tốc độ tăng trưởng (cấp tăng trưởng) của thời gian thực hiện mà ta thực sự quan tâm. Vì vậy, ta chỉ xem xét số hạng đầu của một công thức, bởi các số hạng cấp thấp tương đối không quan trọng với n lớn. Ta cũng bỏ qua hệ số bất biến của số hạng đầu, bởi các thừa số bất biến ít quan trọng hơn tốc độ tăng trưởng trong khi xác định hiệu năng tính toán của các đầu vào lớn. Do đó, ta viết, chẳng hạn kỹ thuật sắp xếp chèn có một thời gian thực hiện trường hợp xấu nhất là $O(n^2)$.

Một thuật toán được xem là hiệu quả hơn so với một thuật toán khác khi thời gian thực hiện trường hợp xấu nhất của nó có một cấp tăng trưởng thấp hơn. Kiểu đánh giá này có thể gặp lỗi đối với các đầu vào nhỏ, song với các đầu vào đủ lớn thì trong trường hợp xấu nhất như $O(n^2)$ chẳng hạn sẽ chạy nhanh hơn so với một thuật toán $O(n^3)$.

2.3 Thiết kế thuật toán

Có nhiều cách để thiết kế các thuật toán. Phương pháp sắp xếp chèn sử dụng cách tiếp cận gia số: khi sắp xếp mảng con A[1..j-1], ta chèn một thàn phần A[j] vào đúng chỗ của nó, cho ra mảng con đã sắp xếp A[1..j]

Trong đoạn này, ta xem xét một cách tiếp cận thiết kế khác, tên là “chia để trị”. Ta sẽ dùng kỹ thuật chia để trị để thiết kế một thuật toán sắp xếp có thời gian thực hiện trường hợp xấu nhất nhỏ hơn nhiều so với kiểu sắp xếp chèn. Các thuật toán chia để trị có một ưu điểm đó là các thời gian thực hiện của chúng thường để xác định bằng các kỹ thuật mô tả trong Chương 4.

2.3.1 Cách tiếp cận chia để trị

Có nhiều thuật toán hữu ích theo cấu trúc đệ quy: để giải quyết một bài toán nhất định, chúng tự gọi theo đệ quy một hay nhiều lần để đối phó với các bài toán con có liên quan mật thiết. Các thuật toán này thường theo cách tiếp cận chia để trị: chúng tách nhỏ bài toán thành vài bài toán con tương tự như bài toán ban đầu song có kích cỡ nhỏ hơn, giải quyết các bài toán con một cách đệ quy rồi tổ hợp các nghiệm này để tạo một nghiệm cho bài toán ban đầu.

Kiểu mẫu chia để trị liên quan đến ba bước tại mỗi cấp của đệ quy:

Chia bài toán thành một số bài toán con

Trị các bài toán con bằng cách giải quyết chúng một cách đệ quy. Tuy nhiên nếu các bài toán con đủ nhỏ ta chỉ việc giải quyết các bài toán con theo cách đơn giản.

Tổ hợp các nghiệm của các bài toán con thành nghiệm cho bài toán ban đầu.

Thuật toán sắp xếp trộn (merge sort) cũng áp dụng kiểu mẫu trên. Theo trực giác, nó hoạt động như sau:

Chia: Dãy n phần tử sẽ được chia thành 2 dãy con để sắp xếp, mỗi dãy có n/2 thành phần

Trị: Dùng kỹ thuật sắp xếp trộn để sắp xếp hai dãy con theo đệ quy

Tổ hợp: Hợp nhất 2 dãy con đã sắp xếp để cho ra đáp án đã sắp xếp

Lưu ý, đệ quy sẽ không thể hiện hết tác dụng khi dãy sắp xếp có chiều dài là 1, trong trường hợp đó không có gì được thực hiện, vì mọi dãy có chiều dày là 1 thì coi như đã sắp xếp.

Tác vụ chính của thuật toán sắp xếp trộn đó là tiến trình trộn hai dãy đã sắp xếp trong bước "tổ hợp". Để thực hiện tiến trình trộn, ta dùng một thủ tục phụ trợ MERGE(A, p, q, r), ở đó A là một mảng và p, q và r là các chỉ mục đánh số các thành phần của mảng sao cho $p \le q < r$. Thủ tục mặc định các mảng con A[p..q] và A[q+1..r] đã sắp xếp. Sau đó trộn chúng lại để tạo thành một mảng hoàn thiện A[p..r] đã sắp xếp.

Ta có thể hình dung đối với MERGE, người ta sử dụng O(n), với n = r – p + 1 là số lượng thành phần đang trộn. Trở về với chủ đề chơi bài, giả sử ta có 2 đống lá bài tung ngửa trên bàn. Mỗi đống được sắp xếp sao cho lá bài nhỏ nhất nằm trên cùng. Ta muốn trộn 2 đống trên thành một đống đã sắp xếp. Bước căn bản bao gồm các tiến trình chọn lá nhỏ hơn trong 2 lá đầu tiên của các đống, gỡ bỏ nó ra khỏi đống đó và đặt lá bài này lên trên đống kết xuất. Ta lặp lại bước này cho đến khi một đống đầu vào trống rỗng, vào lúc đó ta chỉ việc lấy đống đầu vào còn lại và đặt nó lên đống kết xuất. Về mặt tính toán, mỗi bước căn bản bỏ ra một thời gian bất biến, bởi ta chỉ đang kiểm tra 2 lá bài trên cùng. Do ta thực hiện tối đa n bước căn bản, nên tiến trình trộn chiếm O(n) thời gian.

MERGE(A, p, q, r)

```
1     n1= q – p + 1
2     n2 = r – q
3     Khởi tạo mảng mới L[1..n1+1] và R[1..n2+1]
4     for i = 1 to n1
5           L[i] = A[p+i-1]
6     for j=1 to n2
7           R[j] = A[q+j]
8     L[n1+1] = ∞
9     R[n2+1] = ∞
10    i = 1
11    j = 1
12    for k = p to r
13          if L[i] ≤ R[j]
14                A[k] = L[j]
```

```
15                  i = i +1
16          else A[k] = R[j]
17                  j = j+1
```

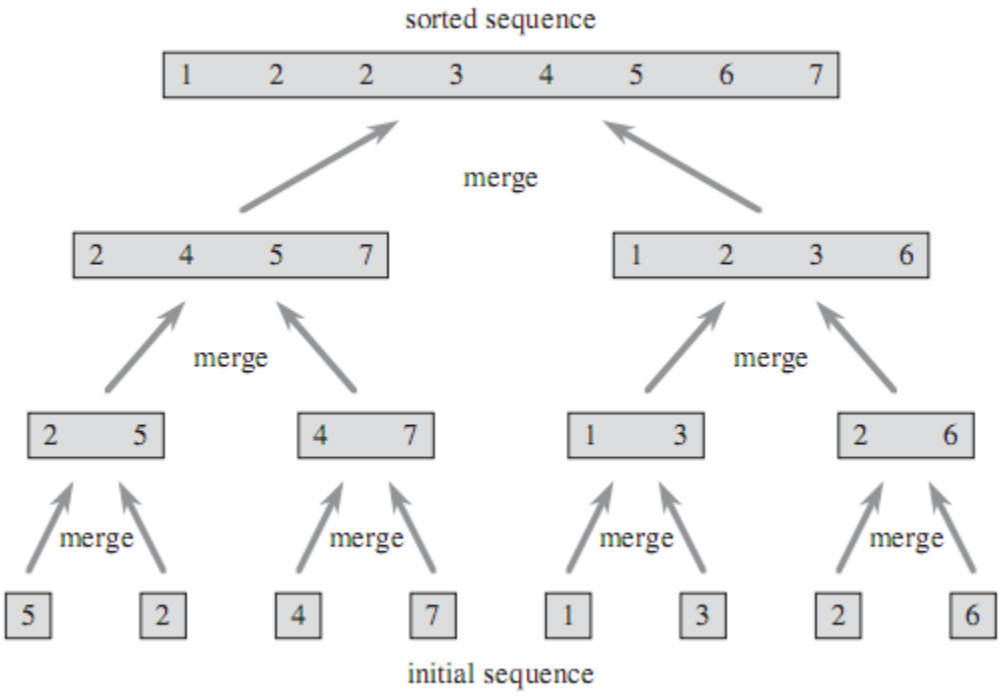
Giờ đây, có thể dùng thủ tục MERGE như một chương trình con trong thuật toán sắp xếp trộn. Thủ tục MERGE_SORT(A, p, r) sắp xếp các thành phần trong mảng caon A[p..r]. Nếu p≥r, mảng con có tối đa một thành phần và do đó đã được sắp xếp sẵn. Bằng không, bước chia đơn giản tính toán một chỉ số q phân hoạch A[p..r] thành 2 mảng con A[p..q], chứa n/2 phần tử và A[q+1..r] chứa n/2 phần tử.

MERGE_SORT(A, p, r)

```
1   if p < r
2   then q ←[(p+r)/2]
3   MERGE_SORT(A, p, q)
4   MERGE_SORT(A, q+1, r)
5   MERGE(A, p, q, r)
```

2.3.2 Phân tích các thuật toán chia để trị

Khi một thuật toán chứa một lệnh gọi đệ quy lên chính nó, thời gian thực hiện của nó thường được mô tả bởi phương trình truy hồi hoặc phép truy hồi, mô tả thời gian thực hiện chung trên một bài toán có kích cỡ n theo dạng thời gian thực hiện trên các đầu vào nhỏ hơn. Như vậy, có thể dùng các công cụ toán học để giải quyết phép truy hồi và cung cấp các giới hạn về khả năng thực hiện của thuật toán.



Hình 2.4 Ứng dụng Merge_Sort trên mảng A<5, 2, 4, 7, 1, 3, 2, 6>. Chiều dài của dãy sắp xếp đang được trộn sẽ tăng khi thuật toán tiến dần từ dưới lên

Một phép truy hồi trong thuật toán chia để trị thường dựa trên ba bước kiểu mẫu că bản. Cũng như trước, giả sử T(n) là thời gian thực hiện trên một bài toán có kích thước n. Nếu kích thước đủ nhỏ, giả sử n ≤ c với c là hằng số, ta sẽ được một thời gian thực hiện bất biến O(1). Giả sử ta chia bài toán thành a bài toán con, mỗi bài có 1/b kích cỡ so với bài toán gốc. Nếu lấy D(n) là thời gian chia bài toán thành các bài toán con và C(n) là thời gian để tổ hợp các nghiệm của bài toán con thành nghiệm cho bài toán ban đầu, ta được:

$$T(n) = \begin{cases} O(1) \ \ nếu\ n \le c \\ aT\left(\frac{n}{b}\right) + D(n) + C(n) \ \ ngược\ lại \end{cases}$$

Ở chương 4, chúng ta sẽ xem xét cách giải quyết các phép truy hồi theo dạng này.

Phân tích thuật toán sắp xếp trộn

Dưới đây là cách suy luận để xác lập phép truy hồi cho T(n) – thời gian thực hiện. TIến trình sắp xếp trộn trên chỉ một thành phần sẽ bỏ ra thời gian bất biến. Khi n > 1 thành phần, ta tách nhỏ thời gian thực hiện như sau:

Chia: Bước chia chỉ tính toán điểm giữa của mảng con, bỏ ra một thời gian bất biến. Do đó D(n) = O(1)

Trị: Ta giải quyết đệ quy hai bài toán con, mỗi bài có kích cỡ n/2, đóng góp 2T(n/2) vào thời gian thực hiện

Tổ hợp: Thủ tục MERGE trên mảng con n phần tử bỏ ra O(n) thời gian, nên C(n) = O(n)

Khi tổ hợp các hàm D(n) và C(n) của đợt phân tích sắp xếp trôn sẽ cho ra phép truy hồi cho thời gian thực hiện (trường hợp xấu nhất) T(n) của thuật toán Merge_Sort:

$$T(n) = \begin{cases} O(1) & nếu\ n = 1 \\ 2T\left(\frac{n}{2}\right) + O(n) & nếu\ n > 1 \end{cases}$$

Trong chương 4, ta sẽ thấy T(n) = O(n.log$_2$n). Nếu đầu vào đủ lớn, thuật toán sắp xếp trộn với thời gian thực hiện O(n.log$_2$n) sẽ vượt trội hơn so với thuật toán sắp xếp chèn có O(n^2).

Phụ lục : Nền tảng toán học

Dẫn nhập :

Sau khi phân tích thuật toán, chúng ta thường cần đến một số công cụ toán học. Nhiều đã quá quen thuộc với các bạn từ Trung học như Đại Số, nhưng một số khác sẽ mới đối với bạn. Ở phần 1, chúng ta đã tìm hiểu về cách (…). Mục phụ lục này sẽ gồm trích lục một vài khái niệm và phương pháp chúng ta sẽ dùng để phân tích những thuật toán trên. Một số lưu ý trong phần giới thiệu ở phần 1

Phục lục A : đề ra những phương pháp cho việc tính toán và (giới hạn bao hàm), những thứ thường xuyên xảy ra trong quá trình phân tích thuật toán. Nhiều công thức toán học xuất hiện ở mục này đều có trong bất kỳ quyển sách toán nào khác, nhưng bạn sẽ dễ dàng tìm được những phương pháp đó đã được biên soạn trong phần phụ lục này.

Phụ lục B : bao gồm những định nghĩa và ký hiệu cơ bản về tập hợp, cặp, quan hệ, hàm, đồ thị, và cây. Ngoài ra, phần này còn đưa ra thêm một vài tính năng cơ bản của một số đối tượng toán học (??).

Phụ lục C : khởi đầu bằng việc tiếp cận với những nguyên tắc đếm cơ bản: hoán vị, chỉnh hợp, tổ hợp,… phần còn lại thì sẽ đề cập về những định nghĩa và tính chất về xác suất. Hầu hết những thuật toán trong sách không đòi hỏi (phân tích xác suất), và vì thế bạn có thể dễ dàng bỏ qua các phần sau của chương này khi đọc lần đầu. Sau đó, khi bạn đã tiếp cận với môn phân tích xác suất chắc rằng bạn sẽ muốn tìm hiểu rõ hơn về, và ở phụ lục C được viết nhằm là nguồn phục vụ cho mục đích tham khảo.

Phụ lục D : Định nghĩa về ma trận, các phép tính trên ma trận, và một số tính chất cơ bản của ma trận. Có thể những tài liệu trong phần này đã quen thuộc đối với bạn nếu bạn đã từng học qua bộ môn đại số tuyến tính.

A. SUMMATIONS (Phép cộng) :

Mỗi thuật toán chứa một cấu trúc điều khiển vòng lặp chẳng hạn vòng lặp **while** hay vòng lặp **for**, ta có thể thể hiện thời gian chạy của thuật toán là tổng thời gian thực hiện các lệnh bên trong mỗi vòng lặp. Lấy ví dụ, ta thấy được trong mục 2.2 có j lần lặp trong thuật toán sắp xếp chèn mất thời gian tỉ lệ thuận với j trong trường hợp xấu nhất. bằng cách cộng thêm thời gian sau mỗi lần lặp, ta đạt được tổng (hay chuỗi) có dạng :

$$\sum_{j=2}^{n} j \ .$$

Khi đánh giá tổng này, ta mất một khoảng $\Theta(n^2)$ trong trường hợp thời gian chạy thuật toán xấu nhất. Ví dụ này minh họa tại sao bạn nên biết cách .

Mục **A.1.** liệt kê những công thức liên quan đến tổng (chuỗi). Mục **A.2.** đưa ra một số ký thuật hữu ích trong việc ước lượng tổng chuỗi. chúng tôi trình bày các công thức trong mục A.1. mà không có phần chứng minh, dù vậy nhiều phần chứng minh sẽ có trong mục A.2. nhằm minh họa cho các phương pháp ở mục này. Ngoài ra bạn có thể tìm thấy các phần chứng minh đó ở trong các sách toán học khác.

A.1. Công thức tổng chuỗi và các tính chất :

Cho dãy số a_1, a_2, a_3,…, a_n với n là số nguyên không âm. Chúng ta có thể biểu diễn tổng hữu hạn số hạng $a_1 + a_2 + \ldots + a_n$ bằng công thức : $\sum_{k=1}^{n} a_k$

Nếu n = 0, giá trị của tổng được xác định bằng 0. Giá trị của chuỗi hữu hạn luôn được xác định và ta có thể thêm các số hạng vào những vị trí bất kì trong chuỗi.

Cho một dãy số vô hạn a_1, a_2, a_3,… Tổng vô hạn dãy số $a_1 + a_2 + \ldots$ có thể được viết bằng công thức hay chuỗi số được kí hiệu bởi :

$$\sum_{k=1}^{\infty} a_k$$

hay có thể được viết lại dưới dạng

$$\lim_{n\to\infty} \sum_{k=1}^{n} a_k$$

Nếu chuỗi không tồn tại giới hạn, chuỗi phân kỳ; ngược lại, thì chuỗi hội tụ. Không thể thêm số hạng ở bất kỳ thứ tự nào vào trong chuỗi hội tụ. Tuy nhiên, ta có thể thực hiền điều đó đối với chuỗi hội tụ hoàn toàn, là chuỗi có dạng $\sum_{k=1}^{\infty} a_k$ và $\sum_{k=1}^{\infty} |a_k|$ đều hội tụ.

Tính tuyến tính (Linearity) :

Cho 1 số thực c bất kỳ và chuỗi hữu hạn a_1, a_2, a_3,…, a_n và b_1, b_2, b_3,…, b_n, ta có :

$$\sum_{k=1}^{n}(ca_k + b_k) = c\sum_{k=1}^{n} a_k + \sum_{k=1}^{n} b_k$$

Chuỗi số học

Tổng chuỗi n số tự nhiên liên tiếp có dạng

$$\sum_{k=1}^{n} k = 1 + 2 + \cdots + n$$

là một **chuỗi số học** và có giá trị là

$$\sum_{k=1}^{n} k = \frac{1}{2}n(n+1)$$

Tổng bình phương và tổng lập phương

$$\sum_{k=0}^{n} k^2 = \frac{n(n+1)(2n+1)}{6},$$

$$\sum_{k=0}^{n} k^3 = \frac{n^2(n+1)^2}{4}.$$

Chuỗi hình học

Cho số thực x khác 1, chuỗi có dạng

$$\sum_{k=0}^{n} x^k = 1 + x + x^2 + \cdots + x^n$$

là chuỗi hình học hay chuỗi lũy thừa và có giá trị là

$$\sum_{k=0}^{n} x^k = \frac{x^{n+1} - 1}{x - 1}. \qquad \text{(A.5)}$$

Nếu tổng chuỗi tiến ra vô cùng và |x| < 1, ta có chuỗi hình học giảm

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x} . \tag{A.6}$$

chuỗi điều hòa

Cho một số tự nhiên dương n, tổng riêng phần điều hòa thứ n là

$$\begin{aligned} H_n &= 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n} \\ &= \sum_{k=1}^{n} \frac{1}{k} \\ &= \ln n + O(1) . \end{aligned} \tag{A.7}$$

(Ta sẽ chứng minh các điều kiện liên quan ở mục A.2)

Vi phân và tích phân chuỗi

Ta lấy vi phân hay tích phân các công thức phía trên, thêm vào đó là các công thức bổ sung phát sinh. Lấy ví dụ, khi ta đạo hàm 2 vế của chuỗi hình học (A.6) và nhân thêm x ta được

$$\sum_{k=0}^{\infty} k x^k = \frac{x}{(1-x)^2} \tag{A.8}$$

Với |x| < 1.

Chuỗi Telescoping

Cho dãy số $a_0, a_1, a_2, \ldots, a_n$,

$$\sum_{k=1}^{n} (a_k - a_{k-1}) = a_n - a_0 , \tag{A.9}$$

Phép nhân

Tích của n số $a_1a_2\ldots a_n$ được kí hiệu là

$$\prod_{k=1}^{n} a_k .$$

Với n = 0 giá trị của tích được xác định bằng 1. Ta có thể chuyển đổi công thức nhân thành công thức cộng bằng đồng nhất thức

$$\lg\left(\prod_{k=1}^{n} a_k\right) = \sum_{k=1}^{n} \lg a_k .$$

A.2. Giới hạn bao hàm (Bounding summations)

Chúng tôi có nhiều kỹ thuật để tính toán thời gian chạy của thuật toán. Sau đây, tôi xin giới thiệu những phương pháp thông dụng nhất.

Quy nạp toán học

Cách đơn giản nhất để tính giá trị một chuỗi là sử dụng quy nạp toán học. Ví dụ hãy chứng minh dãy $\sum_{k=1}^{n} k$ có giá trị là $\frac{1}{2}n(n+1)$. Chúng ta có thể dễ dàng xác định được khi n = 1. Chúng ta giả sử nó đúng với n, và việc cần làm bây giờ là chứng minh nó đúng với n+1. Ta có

$$\sum_{k=1}^{n+1} k = \sum_{k=1}^{n} k + (n+1) = \frac{1}{2}n(n+1) + (n+1) = \frac{1}{2}(n+1)(n+2)$$

Bạn không cần lúc nào cũng phải tính chính xác giá trị của chuỗi, thay vào đó hãy sử dụng quy nạp toán học. Xét một ví dụ khác: Hãy chứng minh $\sum_{k=0}^{n} 3^k$ là $O(3^k)$ và $\sum_{k=0}^{n} 3^k \leq c3^n$ với c là hằng số. Đối với điều kiện ban đầu n = 0, ta có $\sum_{k=0}^{n} 3^k = 1 \leq c$ với $c \geq 1$. Giả sử, nó đúng với n, chúng ta cần chứng minh nó cũng đúng với n+1. Ta có

$$\sum_{k=0}^{n+1} 3^k = \sum_{k=0}^{n} 3^k + 3^{n+1} \leq c3^n + 3^{n+1} = \left(\frac{1}{3} + \frac{1}{c}\right)c3^{n+1} \leq c3^{n+1}$$

Với $\frac{1}{3} + \frac{1}{c} \leq 1$ hay $c \geq \frac{3}{2}$. Vì vậy, $\sum_{k=0}^{n} 3^k = O(3^k)$.

Sự ràng buộc giới hạn

Đôi khi chúng ta có thể có được một ràng buộc tốt trên một chuỗi bằng cách giới hạn mỗi chuỗi, và nó thường yêu cầu sử dụng giới hạn lớn nhất để ràng buộc các thành phần. Ví dụ:

$$\sum_{k=1}^{n} k \leq \sum_{k=1}^{n} n = n^2$$

Nói chung, đối với một chuỗi có dạng $\sum_{k=1}^{n} a_k$ nếu $a_{max} = max_{1 \leq k \leq n} a_k$ thì $\sum_{k=1}^{n} a_k \leq na_{max}$

Chúng ta cũng có thể áp dụng phương pháp này đối với chuỗi hình học. Chẳng hạn, cho chuỗi $\sum_{k=0}^{n} a_k$ với $\frac{a_{k+1}}{a_k} \leq r$ và $0 < r < 1$, r là hằng số. Ta có $a_k \leq a_0 r^k$, vì thế

$$\sum_{k=0}^{n} a_k \leq \sum_{k=0}^{\infty} a_o r^k = a_o \frac{1}{1-r}$$

Chúng ta có thể áp dụng phương pháp này để tính $\sum_{k=1}^{\infty}(\frac{k}{3^k})$. Để k = 0, ta có thể viết lại $\sum_{k=0}^{\infty}(\frac{k+1}{3^{k+1}})$. Giới hạn ban đầu a_0 là 1/3. Ta có

$$\frac{(k+2)/3^{k+2}}{(k+1)/3^{k+1}} = \frac{1}{3}.\frac{k+2}{k+1} \leq \frac{2}{3}$$

Đối với mọi $k \geq 0$. Vì vậy $\sum_{k=1}^{\infty}(\frac{k}{3^k}) = \sum_{k=0}^{\infty}(\frac{k+1}{3^{k+1}}) \leq \frac{1}{3}.\frac{1}{1-\frac{2}{3}} = 1$

Tách chuỗi

Có một cách để tính một cái chuỗi phức tạp đó là chia nhỏ nó ra. Giả sử chúng ta muốn tìm giới hạn dưới của một chuỗi số $\sum_{k=1}^{n} k$. Ta có

$$\sum_{k=1}^{n} k = \sum_{k=1}^{n/2} k + \sum_{k=\frac{n}{2}+1}^{n} k \geq \sum_{k=1}^{\frac{n}{2}} 0 + \sum_{k=\frac{n}{2}+1}^{n} \frac{n}{2} = \left(\frac{n}{2}\right)^2 = O(n^2)$$

Đây là một ràng buộc chặc chẽ, vì thế $\sum_{k=1}^{n} k = O(n^2)$

Đối với phương pháp này, chúng ta thường có thể chia tách và bỏ qua một số điều kiện không đổi ban đầu. Kỹ thuật này cũng áp dụng cho chuỗi vô hạn. Ví dụ, tìm tiệm cận trên của $\sum_{k=0}^{\infty} \frac{k^2}{2^k}$. Ta có

$$\frac{(k+1)^2/2^{k+1}}{k^2/2^k} = \frac{(k+1)^2}{2k^2} \leq \frac{8}{9}$$

Ta có thể chia

$$\sum_{k=0}^{\infty} \frac{k^2}{2^k} = \sum_{k=0}^{2} \frac{k^2}{2^k} + \sum_{k=3}^{\infty} \frac{k^2}{2^k} \leq \sum_{k=0}^{2} \frac{k^2}{2^k} + \frac{9}{8} \sum_{k=0}^{\infty} \left(\frac{8}{9}\right)^k = O(1)$$

Xấp xỉ bằng tích phân

Khi một tổng có dạng $\sum_{k=m}^{n} f(k)$ với f(x) là một hàm tăng đơn điệu, chúng ta có thể ước lượng nó bằng tích phân.

$$\int_{m-1}^{n} f(x)dx \leq \sum_{k=m}^{n} f(k) \leq \int_{m}^{n+1} f(x)dx \qquad (A.11)$$

Ở hình A.1, tổng được biểu diễn dưới dạng diện tích của các hình chữ nhật ở mặt trong, và phần tách rời là vùng bóng mờ dưới đường cong. Khi f(x) là một hàm đơn điệu giảm, chúng ta có thể sử dụng phương pháp tương tự để cung cấp các giới hạn.

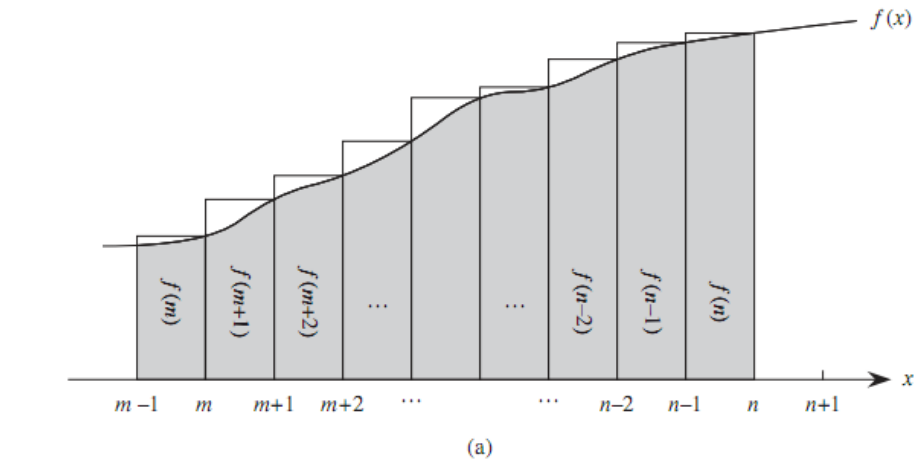
$$\int_{m}^{n+1} f(x)dx \leq \sum_{k=m}^{n} f(k) \leq \int_{m-1}^{n} f(x)dx \qquad (A.12)$$

Các xấp xỉ tích phân A.12 ước lượng gần đúng của số điều hòa thứ n. Đối với cận dưới, ta được

$$\sum_{k=1}^{n} \frac{1}{k} \geq \int_{1}^{n+1} \frac{dx}{x} = \ln(n+1) \qquad (A.13)$$

Đối với cận trên, ta có bất phương trình

$$\sum_{k=2}^{n} \frac{1}{k} \leq \int_{1}^{n} \frac{dx}{x} = \ln(n)$$



Hình A.1 Xấp xỉ của$\sum_{k=m}^{n} f(k)$ bởi tích phân

B. Tập hợp (Sets) :

Nhiều chương trong sách này sẽ đề cập đến toán rời rạc. Phần phụ lục này điểm lại tất cả các kí hiệu, định nghĩa và những tính chất cơ bản của tập hợp, quan hệ, hàm số, đồ thị và cây. Nếu bạn đã rất thành thạo với các nội dung trên, bạn có thể lướt sơ qua ở phần này.

B.1. Tập hợp

Tập hợp là một tập kết hợp của các đối tượng phân biệt, gọi là các thành phần hay phần tử. Nếu đối tượng x là một thành phần của tập S, ta kí hiệu x ∈ S (đọc là "x là phần tử thuộc tập S" hay ngắn gọn hơn là "x thuộc S"). Nếu x không phải là thành phần của tập S, ta kí hiệu x ∉ S. Ngoài ra, ta có thể mô tả một tập hợp bằng cách liệt kê tất cả các phần tử của tập hợp đó bên trong một cặp ngoặc. Thí dụ, ta định nghĩa tập S chứa các số 1, 2, 3 bằng cách viết S = {1, 2, 3}. Bởi vì 2 là 1 phần tử của S, nên ta viết 2 ∈ S, và 4 không phải là phần tử của S, nên ta viết 4 ∉ S.

Một tập không thể chứa nhiều hơn một đối tượng giống nhau, và các phần tử đó không có thứ tự. Hai tập hợp A và B bằng nhau, kí hiệu là A = B, nếu chúng chứa các phần tử giống nhau.

Một số kí hiệu cho những tập hợp đặc biệt :

- Kí hiệu Ø biểu thị cho tập rỗng, tập hợp không có phần tử nào.

- Kí ihệu $\mathbb{Z}$ biểu thị tập hợp các số nguyên {..., -2, -1, 0, 1, 2,...}.
- Kí hiệu $\mathbb{R}$ biểu thị tập hợp các số thực.
- Kí hiệu $\mathbb{N}$ biểu thị tập hợp các số tự nhiên.

Nếu mọi phần tử của tập A đều thuộc tập B, ta gọi A là tập con của B, kí hiệu $A \subseteq B$. A là tập con hoàn toàn của tập B, kí hiệu là $A \subset B$, khi $A \subseteq B$ và $A \neq B$.

- Cho tập A bất kỳ,ta có $A \subseteq A$.
- Cho 2 tập A và B bất kỳ, A = B khi và chỉ khi $A \subseteq B$ và $B \subseteq A$.
- Cho 3 tập bất kỳ A, B và C, nếu $A \subseteq B$ và $B \subseteq C$ theo quy tắc bắc cầu thì $A \subseteq C$.
- Với bất kỳ tập A nào, ta luôn có $\emptyset \subseteq A$.

Cho 2 tập A và B, ta xác định được tập mới bằng cách áp dụng một số phép toán trên tập hợp :

- **Phép giao 2 tập hợp**

$$A \cap B = \{x : x \in A \text{ and } x \in B\} .$$

- **Phép hợp 2 tập hợp**

$$A \cup B = \{x : x \in A \text{ or } x \in B\} .$$

- **Phép lấy phần bù của A trong B**

$$A - B = \{x : x \in A \text{ and } x \notin B\} .$$

Những phép toán tuân theo một số quy luật :

- **Luật với tập rỗng**

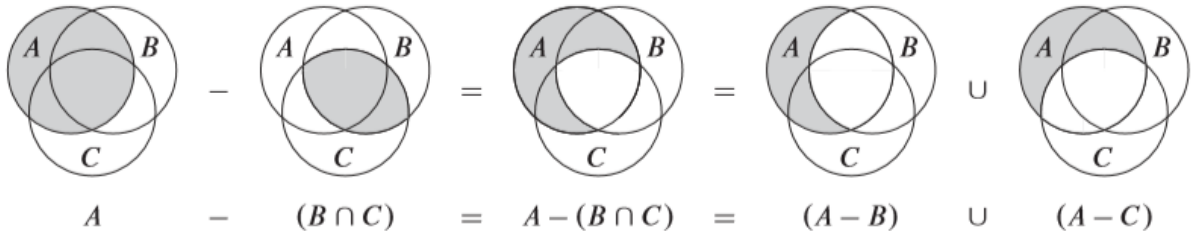
$$A \cap \emptyset = \emptyset ,$$
$$A \cup \emptyset = A .$$

- **Luật lũy đẳng**

$$A \cap A = A ,$$
$$A \cup A = A .$$

- **Luật giao hoán**

$$A \cap B = B \cap A ,$$
$$A \cup B = B \cup A .$$



Hình B.1 Biểu đồ Venn minh họa luật DeMorgan thứ nhất (B.2). Mỗi tập hợp A, B và C được biểu diễn bởi một hình tròn

- **Luật liên kết**

$$A \cap (B \cap C) = (A \cap B) \cap C ,$$
$$A \cup (B \cup C) = (A \cup B) \cup C .$$

- **Luật phân phối**

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C) ,$$
$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C) . \quad \text{(B.1)}$$

- **Luật hấp thụ**

$$A \cap (A \cup B) = A ,$$
$$A \cup (A \cap B) = A .$$

- **Luật Demorgan**

$$A - (B \cap C) = (A - B) \cup (A - C) ,$$
$$A - (B \cup C) = (A - B) \cap (A - C) . \quad \text{(B.2)}$$

Hình B.1 biểu diễn luật DeMorgan thứ nhất bằng cách sử dụng biểu đồ Venn minh họa tập hợp.

Thường thì tất cả các tập hợp được xem là tập con của một tập lớn **U** gọi là tập vũ trụ hay không gian. Ví dụ, nếu đang xét nhiều tập hợp chỉ bao gồm các số nguyên, thì tập $\mathbb{Z}$ xem như là tập vũ trụ của các tập đó. Cho một tập vũ trụ **U**, ta định nghĩa các thành phần không thuộc tập A là $\bar{A} = U - A$ = {x : x ∈ U và x ∉ A}.
Cho tập $A \subseteq U$, ta phải tuân theo một số luật :

- $\bar{\bar{A}} = A$,
- $A \cap \bar{A} = \emptyset$,
- $A \cup \bar{A} = \text{U}$.

Luật DeMorgan đối với tập bù. Cho 2 tập $B, C \subseteq U$, ta có :

$$\overline{B \cap C} = \overline{B} \cup \overline{C} ,$$
$$\overline{B \cup C} = \overline{B} \cap \overline{C} .$$

Hai tập hợp A và B rời nhau nếu chúng không có phần tử trùng nhau, hay $A \cap B = \emptyset$. Một tập hợp $\mathcal{S}$ = {S_i} của tập khác rỗng biểu diễn một phân vùng của tập S khi

- Tập hợp rời nhau từng đôi một, với $S_i, S_j \in \mathcal{S}$ và $i \neq j$ thì $S_i \cap S_j = \emptyset$ và
- Tất cả các tập hợp đó hợp lại bằng S, hay

$$S = \bigcup_{S_i \in \mathcal{S}} S_i .$$

Nói cách khác, $\mathcal{S}$ biểu thị một phân hoạch của S, nếu mỗi phần tử của S xuất hiện đúng một lần trong $S_i \in \mathcal{S}$.

Số lượng phần tử của một tập hợp được gọi là số lượng phần tử (hay kích thước) tập hợp, kí hiệu là |S|. Hai tập có cùng kích thước nếu phần tử của chúng ghép với nhau thành từng đôi một tương ứng. Kích thước tập rỗng là 0, |∅| = 0. Nếu kích thước của một tập là một số tự nhiên, thì tập đó là tập hữu hạn; ngược lại, thì nó là tập vô hạn. Trong tập vô hạn chia làm 2 loại

là tập vô hạn đếm được và tập vô hạn không đếm được. Ví dụ, tập số nguyên Z là tập đếm được, còn tập số thực R là tập không đếm được.
Cho 2 tập hữu hạn A và B, ta có :

$$|A \cup B| = |A| + |B| - |A \cap B| \ , \qquad \text{(B.3)}$$

hay ta có thể kết luận

$$|A \cup B| \leq |A| + |B| \ .$$

Nếu A và B là 2 tập rời nhau, khi đó $A \cap B = \emptyset$ hay $|A \cap B| = 0$. Nên |A B| = |A| + |B|. Nếu $A \subseteq B$, thì $|A| \leq |B|$.
Một tập hữu hạn n phần tử còn được gọi là "**n-set**". "1-set" được gọi là tập đơn. Tập con có k phần tử là "**k-subset**".
Ta biểu thị tập chứa tất cả các tập con của S, bao gồm tập rỗng và chính tập S, là 2^S (được gọi là tập mũ của tập S). ví dụ, $2^{\{a, b\}}$ = {, {a}, {b}, {a, b}}. Kích thước tập mũ của tập S là $2^{|S|}$.
Tích Cartesian (DesCartes) của 2 tập A và B, kí hiệu A x B, là tập tất cả các cặp có thứ tự (a, b), với $a \in A$ và $b \in B$.

$$A \times B = \{(a, b) : a \in A \text{ and } b \in B\} \ .$$

Ví dụ, {a, b} x {a, b, c} = {(a, a), (a, b), (a, c), (b, a), (b, b), (b, c), (c, a), (c, b), (c, c)}. Khi đó, số lượng phần tử của tích Cartesian được tính là |A x B| = |A|.|B|.
Tích Cartesian của n tập A_1, A_2, …., A_n là tập của "**n-tuples**"

$$A_1 \times A_2 \times \cdots \times A_n = \{(a_1, a_2, \ldots, a_n) : a_i \in A_i \text{ for } i = 1, 2, \ldots, n\} \ ,$$

Kích thước của nó là

$$|A_1 \times A_2 \times \cdots \times A_n| = |A_1| \cdot |A_2| \cdots |A_n|$$

Xét trong tập hữu hạn. Ta biểu thị tích Cartesian của n tập đơn A là
A^n = A x A x… x A,
kích thước của tích đó $|A^n| = |A|^n$ với A là tập hữu hạn. Ta cũng có thể xem một n-tuple như là một chuỗi có độ dài n.

B.2. Quan hệ

Một quan hệ 2 ngôi **R** trên 2 tập A và B là một tập con của tích DeCartes A x B. Với (a, b) thuộc R, ta viết **a R b**. Khi ta gọi R là mối quan hệ 2 ngôi trên tập A, ta biết rằng R là tập con của A x A. Ví dụ, "<" là quan hệ trên tập số tự nhiên là một tập {(a, b) : $a, b \in N$ và a < b}. Một quan hệ n ngôi trên các tập A_1, A_2, …., A_n là một tập con của tích DeCartes A_1 x A_2 x …. x A_n.

Cho R là quan hệ trên A, ta có một số tính chất sau :

- Tính phản xạ : R **phản xạ** khi a R a với mọi phần tử $a \in A$. Ví dụ, "=" hay "≤" là quan hệ phản xạ trên **N** (tập số tự nhiên), còn "<" thì không.
- Tính đối xứng : R **đối xứng** khi a R b thì b R a, với mọi $a, b \in A$. Ví dụ, "=" đối xứng, còn "<" hay "≤" thì không phải.
- Tính bắc cầu (truyền) : R bắc cầu (hay truyền) khi a R b và b R c thì a R c, với mọi $a, b, c \in A$. Ví dụ, quan hệ "≤", "<", "=" có tính bắc cầu, còn quan hệ R = {(a, b) : $a, b \in N$ và a – b = 1} thì không phải, vì 3 R 4 và 4 R 5 nhưng không dẫn đến 3 R 5.
- Tính phản xứng : R phản xứng khi a R b và b R a thì a = b. Ví dụ như, "≤" quan hệ trên tập số tự nhiên là phản xứng, vì a ≤ b và b ≤ a thì a = b.

Quan hệ thỏa mãn các tính chất **phản xạ, đối xứng và bắc cầu** là một **quan hệ tương đương**. Ví dụ, "=" là một quan hệ tương đương trên tập số tự nhiên, còn "<" không phải. Cho R là quan hệ tương đương trên A và $a \in A$. Khi đó, tập hợp tất cả các phần tử trong A có quan hệ

với a được gọi là lớp tương đương, kí hiệu [a], [a] = { $x \in A$ | x R a}. Dưới đây là định lý cơ bản (B.1) của lớp tương đương

Định lý B.1 (một quan hệ tương đương là một sự phân hoạch tập hợp)

Lớp tương đương của một quan hệ tương đương R trên tập A biểu diễn tập A thành hợp của các phân hoạch của A, và mỗi phân hoạch A xác định một quan hệ tương đương trên A cho tập hợp các phân vùng là lớp tương đương.

Quan hệ thỏa mãn các tính chất **phản xạ, phản xứng và bắc cầu** là một **quan hệ thứ tự**. Ta gọi tập chứa các phân hoạch thứ tự là tập phân hoạch thứ tự. Trong tập phan hoạch thứ tự A, sẽ không chỉ có một "cực đại" duy nhất a sao cho $b\mathcal{R}a$ với mọi $b \in A$. Thay vào đó, tập hợp sẽ các phân tử cực đại a sao cho $b \notin A$, khi mà $b \neq a$, trong trường hợp $a\mathcal{R}b$.

Quan hệ R trên tập A là **quan hệ toàn phần** nếu với mọi $a, b \in A$, ta có aRb hoặc bRa (hay cả hai đều thỏa), đó là, mỗi cặp các phần tử trong A có quan hệ R. Một quan hệ thứ tự cũng là một quan hệ toàn phần, còn được gọi là **thứ tự toàn phần** hay **thứ tự tuyến tính**. Một quan hệ toàn phần có phải tính bắc cầu, mà không cần phải có tính phản xạ hay phản xứng, là quan hệ toàn phần tiền thứ tự.

B.3. Hàm số

Cho 2 tập A và B, một hàm f là một quan hệ 2 ngôi của A và B sao cho với mỗi phần tử $a \in A$, luôn tồn tại chính xác một phần tử $b \in B$ ứng với a, kí hiệu là $(a, b) \in f$. Tập A được gọi là **tập nguồn** của f và B là **tập đích** của f. Ta còn có thể viết $f : A \rightarrow$ B hay b = f(a), vì b là duy nhất được xác định bằng cách chọn a.

Nhận thấy rằng, hàm f xác định một phần tử của tập B ứng với mỗi phần tử của tập A. Không có phần tử thuộc A được xác định bằng (...) , nhưng một vái phần tử của B có thể được xác định bởi 2 phần tử khác nhau của A. Ví dụ, cho quan hệ 2 ngôi $f = \{(a, b) : a, b \in \mathrm{N}\ và\ b = a\ mod\ 2\}$ (mod : phép lấy chia phần dư) là một hàm số $f : \mathrm{N} \rightarrow \{0, 1\}$, vì với mỗi số tự nhiên a, luôn tồn tại một giá trị b thuộc tập {0, 1} sao cho b = a mod 2, (f(0) = 0, f(3) = 1, ...). Ngược lại, với quan hệ 2 ngôi $g = \{(a, b) : a, b \in \mathrm{N}\ và\ a + b\ chẵn\}$ thì không phải là hàm số, vì (1, 3) và (1, 5) thỏa g, nhưng với một giá trị a = 1 lại không có duy nhất giá trị b thỏa $(a, b) \in g$.

Cho hàm $f : A \rightarrow B$, nếu $b = f(a)$, ta nói a là đối số của f và b là giá trị của f tại a. Ta xác định hàm bằng các giá trị được ánh xạ từ các phần tử của tập nguồn (??). Thí dụ, (...)

Một dãy hữu hạn n phần tử là một hàm số tập nguồn là tập n số nguyên {0, 1, ..., n-1}. Ta kí hiệu dãy đó bởi danh sách các giá trị của nó : $\langle f(0), f(1), f(2), \dots, f(n-1)\rangle$. Một dãy vô hạn là một hàm mà tập nguồn của nó tập số tự nhiên **N**. Ví dụ, dãy Fibonacci, được xác định bằng đệ quy, là dãy vô hạn $\langle 0, 1, 1, 2, 3, 5, 8, \dots\rangle$.

Nếu tập nguồn của hàm số là tích Cartesian (tích DeCartes), ta thường bỏ qua dấu ngoặc đơn bao quanh đối số. Lấy ví dụ, $f : A_1 \times A_2 \times A_3 \times \dots \times A_n \rightarrow B$, ta sẽ viết $b = f(a_1, a_2, a_3, \dots, a_n)$ thay vì $b = f((a_1, a_2, a_3, \dots, a_n))$. Ta cúng có thể coi a_i là một đối số của hàm f, mặc dù về kĩ thuật đối số của hàm f là một n-tuple $(a_1, a_2, a_3, \dots, a_n)$.

Nếu $f : A \rightarrow B$ và $b = f(a)$, thì ta còn gọi b là ảnh của a qua hàm f. Ảnh là tập hợp $A' \subseteq A$ qua hàm f được xác định bởi $f(A') = \{b \in B : b = f(a)\ (for\ some) a \in A'\}$.

Miền giá trị của hàm f là ảnh của tập nguồn, đó là, **f(A)**. Ví dụ, miền giá trị của hàm $f : \mathbb{N} \rightarrow \mathbb{N}$ được định nghĩa bởi $f(n) = 2n$ là $f(\mathbb{N}) = \{m: m = 2n\ với\ mọi\ n \in \mathbb{N}\}$, hay là tập số nguyên không âm chẵn.

Một hàm số là **toàn ánh** khi miền giá trị của nó chính là tập đích.

Một hàm số $f : A \rightarrow B$ là một **đơn ánh** nếu với các đối số khác nhau thì giá trị của hàm f khác nhau, hay, nếu $a \neq a'$ thì $f(a) \neq f(a')$. Đơn ánh còn được gọi là **hàm 1-1**.

Một hàm số $f : A \rightarrow B$ là một **song ánh** nếu nó vừa đơn ánh vừa toàn ánh.

Cho một hàm f là song ánh, ta định nghĩa hàm nghịch đảo của hàm f là f^{-1} với $f^{-1}(b) = a$ khi và chỉ khi $f(a) = b$. Lấy ví dụ, hàm nghịch đảo của hàm $f(n) = (-1)^n(\frac{n}{2})$ là

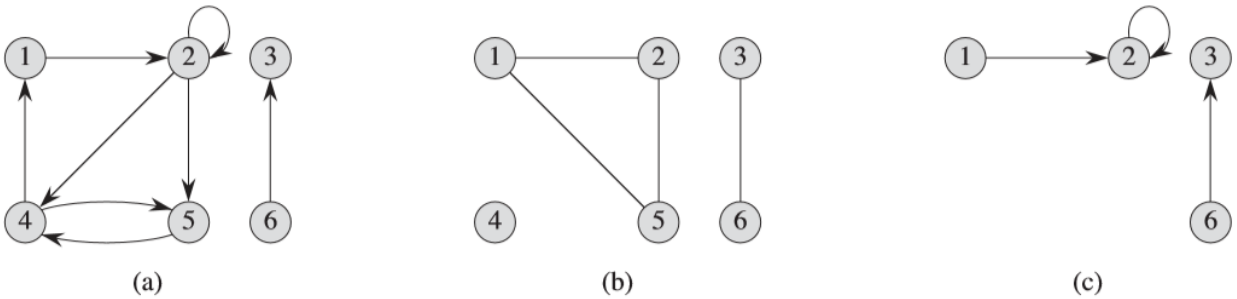
$$f^{-1}(m) = \begin{cases} 2m & \text{if } m \geq 0\,, \\ -2m-1 & \text{if } m < 0\,. \end{cases}$$

B.4. Đồ thị

Ở mục này trình bày 2 loại đồ thị : **đồ thị vô hướng** và **đồ thị có hướng**. Chắc rằng một số định nghĩa trong tài liệu khác sẽ khác với những định nghĩa sắp được đề cập ở đây, nhưng phần lớn sẽ không khác nhau nhiều.

Đồ thị có hướng G là một cặp (V, E), V là một tập giới hạn và E là tập quan hệ của V. Tập V gọi là **tập đỉnh** của G, các phần tử của nó gọi là các đỉnh. Tập E gọi là **tập cạnh** của G, các phần tử của nó gọi là các cạnh. Hình B.2(a) là một hình minh họa cho đồ thị có hướng tập đỉnh {1, 2, 3, 4, 5, 6}. Các đỉnh được biểu diễn bởi hình tròn với số bên trong, và cạnh biểu diễn bằng các mũi tên. Chú ý, **cạnh lặp tại một đỉnh** có thể tồn tại.

Trong đồ thị vô hướng G = (V, E), tập cạnh E bao gồm các cặp đỉnh không thứ tự. Do đó, một cạnh là một tập {u, v}, với $u, v \in V$ và $u \neq v$. Theo qui ước, ta sử dụng kí hiệu (u, v) cho một cạnh thay vì kí hiệu tập hợp {u, v}, và (u, v) hay (v, u) được xem là như nhau. Trong đồ thị vô hướng, cạnh lặp không tồn tại, vì vậy mỗi cạnh bao gồm 2 cạnh khác nhau. Hình B.2(b) biểu diễn minh họa cho đồ thị vô hướng với tập đỉnh {1, 2, 3, 4, 5, 6}.



Hình B.2 Đồ thị có hướng và vô hướng. **(a)** là đồ thị có hướng G = (V, E), với V = {1, 2, 3, 4, 5, 6} và E = {(1,2), (2, 2), (2, 4), (2, 5), (4, 1), (4, 5), (5, 4), (6, 3)}. Cạnh (2, 2) là cạnh lặp. **(b)** đồ thị vô hướng G = (V, E) với V = {1, 2, 3, 4, 5, 6} và E = {(1, 2), (1, 5), (2, 5), (3, 6)}. Đỉnh 4 là đỉnh cô lập.

Nhiều định nghĩa trong đồ thị có hướng và vô hướng là như nhau, mặc dù các thuật ngữ khác nha đôi chút giữa 2 ngữ cảnh. Nếu (u, v) là 1 cạnh trong đồ thị có hướng G = (V, E), ta nói rằng (u, v) là **liên thuộc từ hay rời khỏi** đỉnh u và **liên thuộc đến hay vào** đỉnh v.

Nếu (u, v) là một cạnh của đồ thị G = (V, E), ta gọi đỉnh v **kề** đỉnh u. Trong đồ thị vô hướng, quan hệ kề có tính đối xứng. Khi đồ thị có hướng, quan hệ kề không có tính đối xứng. Nếu v kkè u trong đồ thị có hướng, ta kí hiệu $u \rightarrow v$. ở hình (a) và (b) của hình B.2, đỉnh 2 kề với đỉnh 1, vì cạnh (1, 2) có trong đồ thị. Nhưng đỉnh 1 không kề đỉnh 2 trong hình B.2(a), vì cạnh (2, 1) không thuộc đồ thị.

Bậc của đỉnh trong đồ thị vô hướng là số cạnh liên thuộc trong đồ thị. Ví dụ, đỉnh 2 trong hình B.2(b) có bậc là 2. Một đỉnh có bậc chính nó là 0, giống như đỉnh 4 trong hình B.2(b), vì nó cô lập. Trong đồ thị có hướng **bậc ra** của một đỉnh là số cạnh ra đỉnh đó, và **bậc vào** của một đỉnh là số cạnh vào đỉnh đó. Bậc của một đỉnh trong đồ thị có hướng là tổng của bậc ra và bậc vào của nó. Đỉnh 2 trong hình B.2(a) có 2 bậc vào, 3 bậc ra nên có 5 bậc.

Một lộ trình có độ dài k từ đỉnh u đến đỉnh u' trong đồ thị G = (V, E) là một dãy $\langle v_0, v_1, v_2, \ldots, v_k \rangle$ các đỉnh với $u = v_0$, $u' = v_k$ và $(v_{i-1}, v_i) \epsilon E$ (i = 1, 2, …, k).Độ dài lộ trình là số cạnh trong lộ trình đó. Lộ trình chứa các đỉnh $v_0, v_1, v_2, \ldots, v_k$ và các cạnh (v_0, v_1), (v_1, v_2), …, (v_{k-1}, v_k). (độ dài từ u đến chính nó là 0).

Nếu có một đường đi p từ u đến u', ta nói u' có thể đến được từ u theo p, ta kí hiệu $u \rightsquigarrow u'$ nếu G có hướng. Một đường là **đường đơn** khi tất cả đỉnh trong đường phân biệt nhau. Hình B.2(a), đường đi $\langle 1, 2, 5, 4 \rangle$ là đường đơn có độ dài 3. Trong khi đó, đường đi $\langle 2, 5, 4, 5 \rangle$ không đơn.

Một đường con của p = $\langle v_0, v_1, v_2, \dots, v_k \rangle$ là một dãy con liên tiếp các đỉnh của p. Đó là, với $0 \le i \le j \le k$, dãy con các đỉnh $\langle v_i, v_{i+1}, v_{i+2}, \dots, v_j \rangle$ là đường con của p.

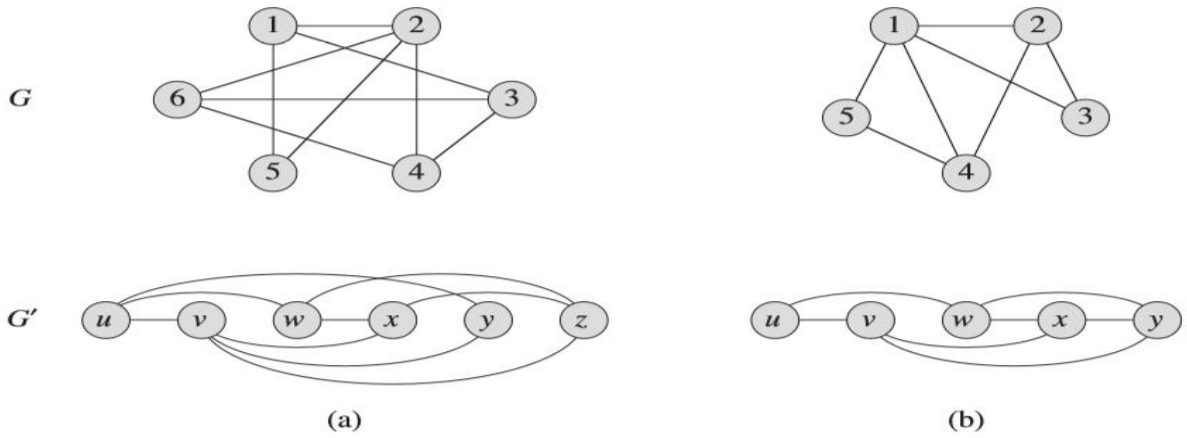
Trong đồ thị có hướng, một con đường $\langle v_0, v_1, v_2, \dots, v_k \rangle$ trở thành **chu trình** khi $v_o = v_k$ và đường phải chứa ít nhất một cạnh. Chu trình là **chu trình đơn** nếu thêm $v_0, v_1, v_2, \dots, v_k$ phân biệt nhau. Một cạnh tự lặp có có độ dài là 1. Hai đường đi $\langle v_0, v_1, v_2, \dots, v_k, v_0 \rangle$ và $\langle {v_0}', {v_1}', {v_2}', \dots, {v_k}', {v_0}' \rangle$ là 2 chu trình giống nhau nếu tồn tại một số nguyên j sao cho ${v'}_i = v_{(i+j) \bmod k}$ khi i = 0, 1, …, k-1.

Trong đồ thị vô hướng, một đường đi $\langle v_0, v_1, v_2, \dots, v_k \rangle$ trở thành chu trình nếu $k \ge 3$ và $v_0 = v_k$, chu trình là chu trình đơn nếu thêm $v_0, v_1, v_2, \dots, v_k$ phân biệt nhau.

Một đồ thị vô hướng là **liên thông** nếu mỗi đỉnh đều có thể đến được từ tất cả các đỉnh còn lại. **Thành phần liên thông** của đồ thị là các lớp tương đương của các đỉnh thông qua quan hệ "có thể đến được". Một đồ thị vô hướng liên thông nếu chỉ có đúng một thành phần liên thông. Các cạnh trong thành phần liên thông thì liên thuộc với các đỉnh trong đó. Nói cách khác, cạnh (u, v) là một cạnh thuộc thành phần liên thông chỉ khi cả u và v đều là đỉnh thuộc thành phần liên thông đó.

Một đồ thị có hướng **liên thông mạnh** nếu trong 2 đỉnh bất kì có thể đến được từ mỗi đỉnh còn lại. **Thành phần liên thông mạnh** trong đồ thị vô hướng là lớp tương đương của các đỉnh thông qua quan hệ "có khả năng đến được". một đồ thị có hướng liên thông mạnh nếu chỉ có một thành phần liên thông mạnh.

Hai đồ thị G = (V, E) và G' = (V', E') là **đẳng cấu (cùng cấu hình)** nếu tồn tại một song ánh $f : V \to V'$ sao cho $(u, v) \in E'$. Hay ta có thể đánh nhãn lại các đỉnh trong G thành các đỉnh trong G', tương ứng với các cạnh. Hình B.3(a) minh họa cặp đồ thị đẳng cấu G và G' với tập đỉnh tương ứng V = {1, 2, 3, 4, 5, 6} và V' = {u, v, w, x, y, z}. Ánh xạ từ V sang V' cho ra $f(1) = u, f(2) = v, f(3) = w, f(4) = x, f(5) = y, f(6) = z$.



Hình B.3 (a) cặp đồ thị đẳng cấu. Các đỉnh của đồ thị bên trên được ánh xạ với các đỉnh đỏ thị bên dưới với $f(1) = u, f(2) = v, f(3) = w, f(4) = x, f(5) = y, f(6) = z$. **(b)** hai đồ thị không đẳng cấu. Vì đỉnh 4 của đồ thị phía trên không ánh xạ phía dưới.

Ta nói G' = (V', E') là **đồ thị con** của G = (V, E) khi $V' \subseteq V$ và $E' \subseteq E$. Cho tập $V' \subseteq V$, đồ thị con của G tạo bởi V' là đồ thị G' = (V', E'), Khi $E' = \{(u, v) \in E : u, v \in V'\}$. Đồ thị con tạo bởi tâph đỉnh {1, 2, 3, 6} trong hình B.2(a) có trong hình B.2(c) và tập cạnh {(1, 2), (2, 2), (6, 3)}.

Cho đồ thị vô hướng G = (V, E), **phiên bản có hướng** của G là đồ thị có hướng G' = (V', E'), trong đó $(u, v) \in E'$ khi và chỉ khi $(u, v) \in E$. Đó là, khi ta thay mỗi cạnh vô hướng (u, v) bằng 2 cạnh có hướng (u, v) và (v, u) trong phiên bản có hướng.

Một vài loại đồ thị đặc biệt. **Đồ thị hoàn chỉnh** là đồ thị vô hướng mà mỗi cặp đỉnh thì kề nhau. **Đồ thị hai phía** là đồ thị vô hướng G = (V, E) trong đó, V được phân hoạch thành 2 tập V_1, V_2 sao cho $(u, v) \in E$ thì $u \in V_1$ và $v \in V_2$ hoặc ngược lại. Một đồ thị vô hướng, không có chu trình là một **rừng** và một đồ thị vô hướng, liên thông, không có chu trình là một **cây** (Mục B.5).

Có 2 biến thể của đồ thị mà đôi khi bạn gặp phải. **Đa đồ thị** là đồ thị vô hướng, nhưng nó có **đa cạnh** nối với các đỉnh và có cạnh lặp. **Siêu đồ thị** là đồ thị vô hướng, nhưng mỗi **siêu cạnh** nối nhiều hơn 2 đỉnh, nối với một tập con các đỉnh tùy ý. Có nhiều thuật toán được viết cho đồ thị vô hướng và có hướng áp dụng chạy trên các cấu trúc đồ thị đó.

Sự co lại của đồ thị vô hướng G = (V, E) bởi một cạnh e = (u, v) là đồ thị đồ thị G' = (V', E'), trong đó, $V' = V - \{u, v\} \cup \{x\}$, x là đỉnh mới. Tập cạnh E' trở thành tập E bằng cách bỏ đi cạnh (u, v) và với mỗi cạnh w liên thuộc u và v, ta xóa luôn cạnh (u,w) và (w,v) trong tập E và thêm vào cạnh (x, w). Khi áp dụng, u và v "bị co lại" khi nó là đỉnh đơn.

B.5. Cây

Giống như đồ thị, có nhiều sự liên hệ nhưng sẽ hơi khác so với đồ thị. Mục này trình bày định nghĩa và một số tính chất toán học của vài loại cây. Mục 10.4 và 22.1 mô tả cách chúng ta mô hình hóa cây trong máy tính.

B.5.1. Cây tự do

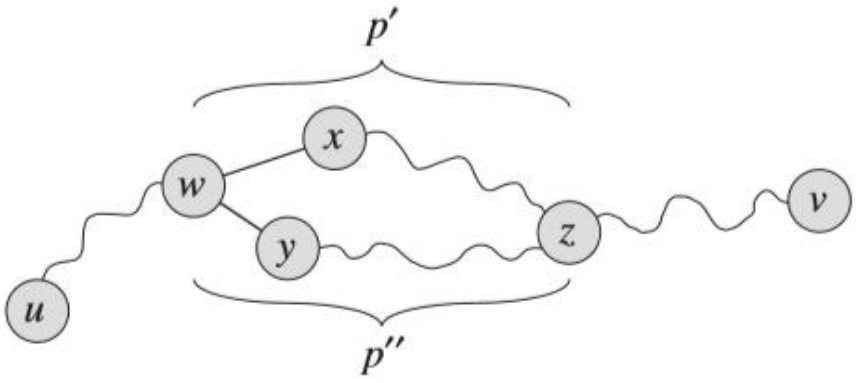
Như đã định nghĩa ở mục B.4, một cây tự do là một đồ thị vô hướng, liên thông, không có chu trình. Ta thường bỏ đi từ "tự do" khi nói đồ thị đó là một cây. Nếu đồ thị vô hướng không có chu trình nhưng có thể không liên thông, đồ thị đó là rừng. Có rất nhiều thuật toán áp dụng cho cây và rừng.

Hình B.4 (a) cây tự do (cây) **(b)** rừng **(c)** đồ thị chứa một chu trình và do đó không phải là cây hay rừng.

Định lý B.2 (Tính chất của cây)

Cho G = (V, E) là một đồ thị vô hướng. Những ý dưới đây là tương đương.

1. G là một cây.
2. Hai đỉnh bất kỳ của G liên thông bởi một chu trình đơn độc lập.
3. G liên thông, nhưng nếu xóa bất kỳ cạnh nào từ tập E, đồ thị sẽ không liên thông.
4. G liên thông và |E| = |V| - 1.
5. G không có chu trình và |E| = |V| - 1.
6. G không có chu trình, nhưng thêm bất kỳ cạnh nào vào tập E, đồ thị sẽ chứa một chu trình.



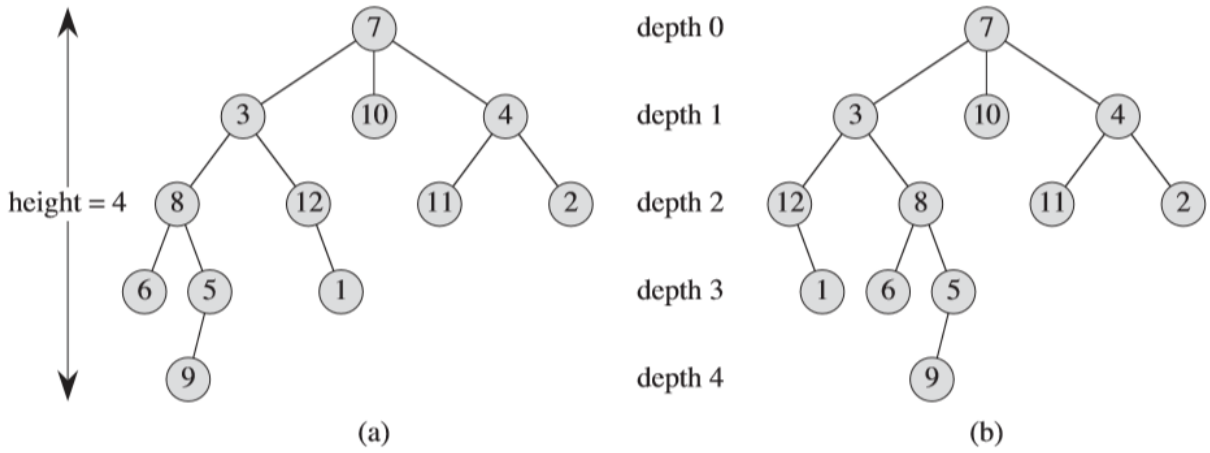
Hình B.5 Một bước trong qua trình chứng minh định lý B.2

B.5.2 Cây cùng gốc và cây thứ tự

Một **cây cùng gốc** là cây mà mỗi đỉnh trong cây phân biệt với các đỉnh còn lại. ta gọi đỉnh phân biệt đó là **gốc**. Gọi đỉnh của cây có gốc là nút của cây. Hình B.6(a) biểu thị một cây có gốc với tập 12 nút và gốc là 7.

Xét một nút x nằm trong cây T với gốc r. Ta gọi nút y bất kì trên đường đi đơn từ r đến x là **tổ tiên** của x. Nếu y là **tổ tiên** của x, thì x là **hậu duệ** của y (Mỗi nút vừa là tổ tiên vừa là hậu duệ của chính nó). **Cây con cùng gốc tại x** là cây được tạo bởi các con cháu của x và gốc x. Ví dụ, cây con cùng gốc tại nút 8 trong hình B.6(a) chứa các nút 8, 6, 5, 9.

Nếu cạnh cuối cùng của đường đi đơn từ gốc r của cây T tới nút x là (y, x), thì y là **cha** x, và x là **con** y. Gốc chỉ có 1 nút trong T thì không có cha. Nếu 2 nút cùng cha, chúng là **anh em**. Nút mà không có con gọi là **lá** hay **nút bên ngoài**. Nút không có lá gọi là **nút trong**.



Hình B.6 Cây có nút và cây có thứ tự

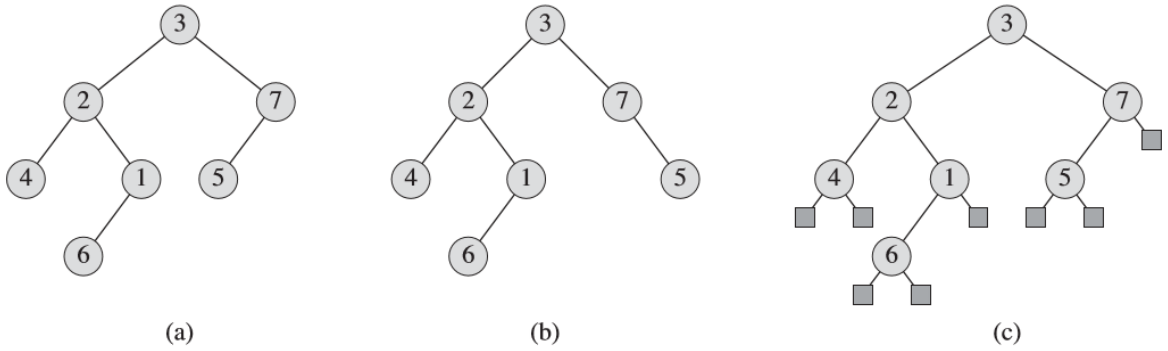
Số con của nút x trong cây có gốc T bằng **số bậc** của x. Độ dài đường đi đơn từ gốc r đến x là **độ sâu** của x. **Mức** của cây bao gồm tất cả các nút có cùng độ sâu. **Chiều cao** của một nút là số cạnh trên đường đơn dài nhất đi xuống từ nút đó đến lá, và **chiều cao cây** là chiều cao gốc. Chiều cao cây cúng bằng độ sâu lớn nhất của nút bất kỳ trong cây.

Một **cây có thứ tự** là một cây có gốc mà con của mỗi nút được sắp thứ tự.Đó là, nếu một nút có k con, thì sẽ có con thứ nhất, con thứ hai, … con thứ k. 2 cây trong hình B.6 thì khác nhau khi xét là cây tứ tự, nhưng giống nhau khi xét chúng là cây cùng gốc.

B.5.3 Cây nhị phân và cây vị trí

Ta xác định cây nhị phân bằng đệ quy. một **cây nhị phân T** là một cấu trúc được xác định với tập hữu hạn nút thỏa

- Không chứa nút nào, hoặc
- Bao gồm 3 tập hợp nút rời nhau : một nút gốc, một cây nhị phân là cây con trái, một cây nhị phân là cây con phải.

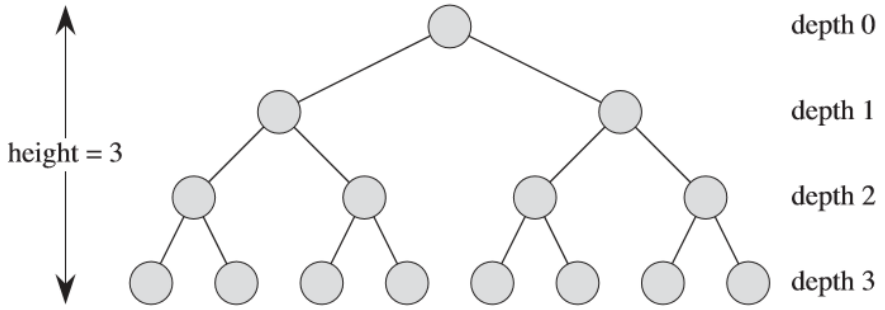


Hình B.7 Các cây nhị phân. **(a)** Cây nhị phân vẽ theo cách chuẩn. Con bên trái của nút được vẽ bên dưới trái nút, con bên phải của nút được vẽ bên dưới phải nút. **(b)** Một cây nhị phân khác với cây nhị phân (a). **(c)** Cây nhị phân (a) được trình bày như một cây nhị phân đầy đủ: mỗi (ordered tree) có trong nút có bậc là 2.

Cây nhị phân không chứa nút gọi là **cây rỗng**, hoặc có thể kí hiệu là **NIL**. Nếu cây con trái khác rỗng, nút gốc của nó được gọi là **con bên trái** của gốc của toàn bộ cây. Tương tự, đối với cây con phải cũng vậy. Nếu một cây con rỗng, ta gọi con bên đó bị **khuyết** hay **mất**. Hình B.7(a) biểu diễn một cây nhị phân.

Một cây nhị phân không đơn thuần là một cây thứ tự mà mỗi nút của nó chỉ có tối đa 2 bậc (2 con). **Cây nhị phân đầy đủ** là cây nhị phân mà mỗi nút của nó hoặc là lá hoặc có bậc là 2, không có nút có bậc là 1. Do đó, thứ tự các con của nút được nút đó giữ thông tin vị trí.

Cây k-phân là cây vị trí mà với mỗi nút, tất cả các con có nhãn lớn hơn hoặc bằng k đều không có. Vì thế, cây nhị phân là cây k-phân với k = 2.



Hình B.8 một cây nhị phân hoàn chỉnh có độ cao 3, với 8 nút lá và 7 nút bên trong

Một cây k-phân hoàn chỉnh là cây k-phân mà mọi lá của nó có cùng độ sâu và các nút bên trong có cùng bậc là k. Hình B.8 cho thấy được một cây nhị phân hoàn chỉnh có độ cao là 3. Số lá tạo đọ sâu h là $\mathbf{k^h}$. Số nút bên trong của một cây k phân có độ cao h là

$$1 + k + k^2 + \cdots + k^{h-1} = \sum_{i=0}^{h-1} k^i = \frac{k^h - 1}{k - 1}$$

Vì thế, cây nhị phân hoàn chỉnh có $\mathbf{2^k - 1}$ nút bên trong.

C. Phép đếm và xác suất

Mục phụ lục C xem xét lại vè tổ hợp cơ bản và lý thuyết xác suất. Nếu bạn có căn bản tốt trong những phần này bạn có thể lướt qua phần mở đầu và taaph trung vào phần sau. Hầu hết những chương trong sách này không cần lý thuyết về xác suất, nhưng một vài chương khác sẽ rất cần. **C.1** tóm tắt những tổng quan cơ bản về lý thuyết đếm bao gồm công thức đếm chuẩn hoán vị và tổ hợp. Các tiên đề xác suất và các sự kiện xác suất được được phân phối trong phần **C.2**. Biến ngẫu nhiên được giới thiệu ở mục **C.3**, cùng với đó các tính chất kỳ vọng và phương sai. Mục **C.4** (investigate) xác suất hình học và phân phối nhị thức phát sinh từ nghiên cứu thử nghiệm Bernoulli (phân phối Bernoulli). Mục **C.5** thảo luận về một số vấn đề nâng cao của phân phối xác suất.

C.1. Đếm

Lý thuyết đếm được dùng để trả lời cho câu hỏi "có bao nhiêu?" mà không cần liệt kê tát cả các sự lựa chọn (??). Ví dụ "Có bao nhiêu số n-bit khác nhau?" hay "Có bao nhiêu cách sắp xếp các phân tử khác nhau vào vị trí ?". Ở mục này, chúng ta cùng xem qua lý thuyết đếm.

Quy tắc cộng và quy tắc nhân

Xâu

Một **xâu** của một tập hữu hạn S. Ví dụ, có 8 xâu nhị phân có có độ dài 3: 000, 001, 010, 011, 100, 101, 110, 111.

Chúng ta thường gọi 1 xâu độ dài k là 1 k-xâu. Một xâu con s' của xâu s là một cách sắp xếp (…)

Hoán vị

Một hoán vị của một tập hợp hữu hạn S là một cách sắp xếp tất cả các phần tử trong S, mỗi phần tử xuất hiện đúng 1 lần. Ví dụ, nếu S = {a, b, c}, ta có 6 hoán vị tập S: abc,acb, bac, bca, cab, cba.

Có n! hoán vị của 1 tập hợp n phần tử, bởi vì với vị trí thứ nhất có n cách chọn, vị trí thú 2 có (n - 1) cách chọn, tương tự..

(…)

Tổ hợp

Một tổ hợp chập k của tập hợp n phần tử S là tập con k phần tử của S. ví dụ, tập hợp 4 phần tử {a, b, c, d} có 6 tổ hợp 2 phần tử : ab, ac, ad, bc, bd, cd.

(…)

C.2. Xác suất

Xác suất là nột công cụ quan trọng cho việc thiết kế và phân tích xác suất cùng với các thuật toán ngẫu nhiên. Phần này sẽ giới xem xét về các lý thuyết xác suất.

Ta định nghĩa xác suất là về một ***không gian mẫu*** *S,* mà chứa gồm các phần tử là các biến cố cơ bản. Chúng ta nghĩ rằng mỗi biến cố cơ bản này được coi như là một khả năng có thể xảy ra khi thử nghiệm. Với thí nghiệm tung hai đồng xu có hai mặt khác nhau, với mỗi trường hợp tung

được sẽ ra mặt sấp (S) hoặc ngửa (N), chúng ta thể hiện không gian mẫu bao gồm bộ tất cả các khả năng với chuỗi 2 ký tự bao gồm {S,N}:

S = {SS, SN, NS, NN}.

Một sự kiện là một tập hợp con[1] của không gian mẫu *S.* Ví dụ, với thí nghiệm tung hai đồng xu, sự kiện bao gồm một mặt sấp và một mặt ngửa là {SN, NS}. với biến cố S được gọi là biến cố chắc chắn, và biến cố Ø được gọi là biến trỗng. Chúng ta nói rằng hai biến cố A và B là xung khắc nếu $A \cap B$ =Ø. Đôi lúc chúng ta biểu thị một sự kiện cơ bản $s \in S$ là biến cố {*s*}. Theo định nghĩa, tất cả các biến cố cơ bản đều là xung khắc.

Tiên đề của xác suất

Một ***phân phối xác suất*** Pr{} trên không gian mẫu *S* là cách để chuyển những biến cố của S thành số thực thỏa mãn các tiên đề của xác suất:

1. Pr{*A*} ≥ 0 với mọi biến cố *A*.

2. Pr{*S*} = 1.

3. Pr{ $A \cup B$ } = Pr{*A*} + Pr{*B*} với bất kỳ hai biến cố xung khắc *A* và *B.* Nhìn chung, với bất kỳ (có giới hạn hoặc vô hạn) trình tự các biến cố A_1,A_2,… mà xung khắc từng đôi một,

$$\Pr\left\{\bigcup_i A_i\right\} = \sum_i \Pr\{A_i\} .$$

Ta gọi Pr{*A*} là xác suất của biến cố *A.* lưu ý rằng tiên đề 2 là điều kiện tiêu chuẩn: không thật sự có một tiêu chuẩn nào cho việc chọn 1 là xác suất biến cố chắc chắn, ngoài việc nó tự nhiên và thuận tiện.

Một vài hệ quả theo sau các tiên đề và định lý cơ bản (ở Mục B.1). Biến cố rỗng Ø là biến cố có Pr{Ø} = 0. Nếu $A \subseteq B$, thì Pr{*A*} ≤ Pr{*B*}. sử dụng $\overline{A}$ để ký hiệu biến cố *S* - *A* (phần bù của *A*), khi ta có Pr{ $\overline{A}$ } = 1-{*A*}. Với hai biến cố bất kỳ *A* và *B,*

Pr{ $A \cup B$ } = Pr{*A*} + Pr{*B*} -Pr{ $A \cap B$ } (C.12)

≤ Pr{*A*} + Pr{*B*} . (C.13)

Với ví dụ tung đồng xu, giả sử rằng mỗi 4 biến cố cơ bản có xác suất là 1/4. Và xác suất để có ít nhất nột mặt sấp là

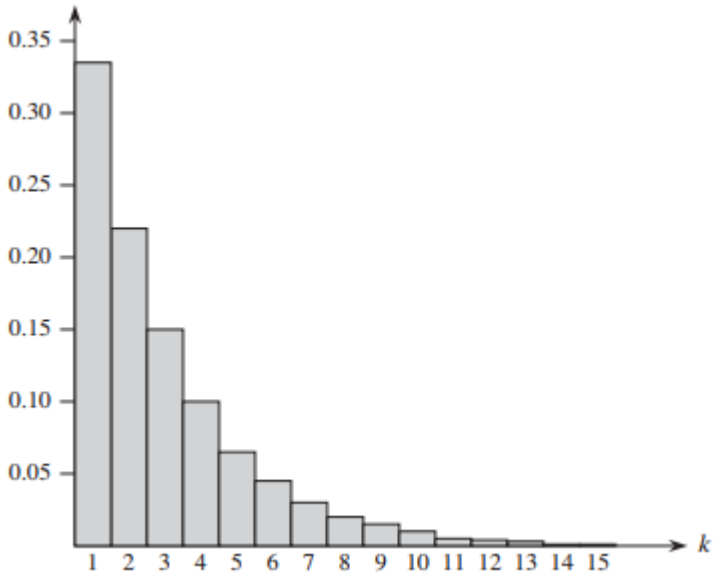
Pr{SN,NS,SS} = Pr{SS} + PR{SN} + Pr{NS}

= 3/4

Cách khác, vì xác suất để nhận được ít hơn một mặt sấp Pr{NN} = 1/4, và xác suất để có ít nhất một mặt sấp là 1 - 1/4 = 3/4.

C.3. Biến ngẫu nhiên rời rạc

Chúng ta có thể nghĩ tung đồng xu là một ***phép thử Bernoulli***, là một thí nghiệm chỉ có hai kết quả xảy ra: thành công, với xác suất là *p*, và thất bại, với xác suất *q* = 1-*p*. khi ta nói ***phép thử Bernoulli*** một cách chung chung, có nghĩa là các phép thử đó là độc lập với nhau, trừ khi chúng ta nói một cách đặc biệt khác, mà mỗi xác suất p thành công. Hai phân bố quan trọng xuất hiện từ phép thử Bernoulli: phân bố hình học và phân bố nhị thức.



Hình C.1 một phân bố nhị thức với xác suất *p* = 1/3 khả năng thành công và *q* = 1-*p* khả năng thất bại. Kỳ vọng của phân bố lá 1/*p*=3

C.4. Phân phối hình học và phân phối nhị thức

Giả sử ta có một chuỗi phép thử Bernoulli, mỗi xác suất *p* là xác suất thành công và *q*= 1-*p* là xác suất thất bại. Có bao nhiêu phép thử xảy ra trước khi chúng ta có một phép thử thành công? Hãy định nghĩa một biến ngẫu nhiên X là một số mà phép thử cần để thành công. Khi đó *X* có giá trị trong tầm {1,2,…}, và với *k*≥1,

$$\Pr\{X = k\} = q^{k-1}p\,, \qquad \text{(C.31)}$$

khi ta có k-1 lần thất bại trước khi có một lần thành công. Một phân bố xác suất thỏa mãn biểu thức (C.31) được gọi là ***phân bố hình học***. Hình C.1 biểu diễn loại phân bố đó.

Giả sử rằng *q* < 1, ta có thể tính được kỳ vọng của phân bố hình học sử dụng hàm (A.8):

$$E[X] = \sum_{k=1}^{\infty} kq^{k-1}$$

$$= \frac{p}{q} \sum_{k=0}^{\infty} kq^{k}$$

$$= \frac{p}{q} \cdot \frac{q}{(1-q)^2}$$

$$= \frac{p}{q} \cdot \frac{q}{p^2}$$

$$= 1/p \qquad \text{(C.32)}$$

Do đó, trung bình, nó cần 1/*p* lần thử trước khi ta có được một lần thành công, một kết quả trực quan.

Phương sai, có thể được tính tương tự, nhưng sử dụng bài tập A.1-3. là

$$\text{Var}[X] = q/p^2 \qquad \text{(C.33)}$$

Ví dụ, giả sử lăn liên tục 2 xúa xắc cho tới khi có được là bảy hoặc mười một. Trong 36 kết quả có thể xảy ra, 6 phần của bảy và 2 phần chủa mười một. Do đó, xác suất thành công là p = 8/36 = 2/9, và ta phải lăn 1/p = 9/2 =4.5 lần so với trung bình của một lần bảy hoặc mười một.

Phân bố nhị thức

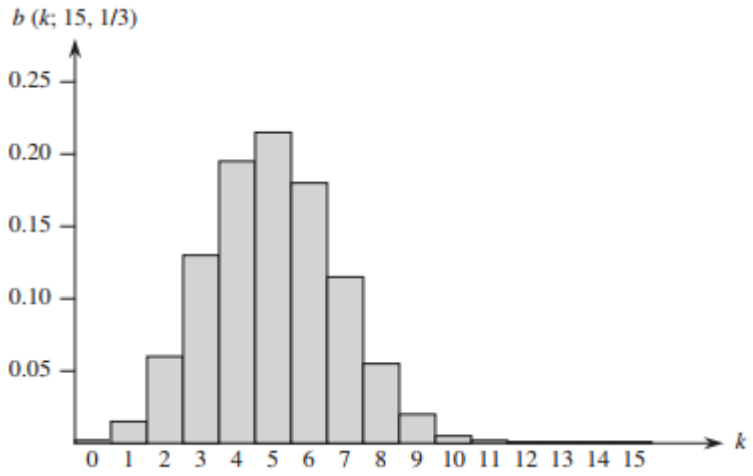
Có bao nhiêu lần thành công xảy ra trong *n* phép thử Bernoulli, một lần thành công xảy ra với xác suất *p* và một lần thất bại với xác suất là q = 1 - p? xác định biến ngẫu nhiên với số lần thành công trong n lần thử. Khi đó X có gái trị trong khoảng {0,1,…,n}, và với k = 0,1,…,n,

$$\Pr\{X = k\} = \binom{n}{k} p^k q^{n-k} \qquad \text{(C.34)}$$

Vì ta có $\binom{n}{k}$ cách để chọn k trong n lần thử là thành công, và xác suất mỗi lần xảy ra là $p^k q^{n-k}$. Phân bố xác suất thỏa mãn đẳng thức (C.34) và được gọi là ***phân bố nhị phân.*** Để thuận tiện, ta định nghĩa các phân bố nhị phân sử dụng ký hiệu sau

$$b(k; n, p) = \binom{n}{k} p^k (1-p)^{n-k} \qquad \text{(C.35)}$$

Hình C.2 biểu thị một phân bố nhị phân. Từ “nhị phân” đến từ phía bên phải của đẳng thức (C.34) với lần thứ *k* là khai triển của $(p + q)^n$. Hệ quả là, *p* + *q* = 1,



Hình C.2 Phân bố nhị thức với $b(k;12,1/3)$ kết quả từ $n = 15$ phép thử Bernoulli, mỗi lần với xác suất $p = 1/3$ khả năng thành công. Kỳ vọng của phân bố là $np = 5$

$$\sum_{k=0}^{n} b(k; n, p) = 1 \qquad \text{(C.36)}$$

như tiên đề 2 của xác suất yêu cầu.

Chúng ta tính được kỳ vọng biến ngẫu nhiên có phân bố nhị phân từ biểu thức (C.8) và (C.36). Lấy X là một biến ngẫu nhiên theo sau phân nố nhị phân b(k;n,p), và lấy q = 1 - p. Bằng định nghĩa của kỳ vọng , ta có

$$\begin{aligned}
E[X] &= \sum_{k=0}^{n} k.\Pr\{X = k\} \\
&= \sum_{k=0}^{n} k.b(k; n, p) \\
&= \sum_{k=1}^{n} k\binom{n}{k} p^k q^{n-k} \\
&= np\sum_{k=1}^{n}\binom{n-1}{k-1} p^{k-1} q^{n-k} \\
&= np\sum_{k=0}^{n-1}\binom{n-1}{k} p^{k} q^{(n-1)-k} \\
&= np\sum_{k=0}^{n-1}\binom{n-1}{k} b(k; n-1, p)
\end{aligned}$$

= $n\,p$ (biểu thức (C.36)) (C.38)

Ta có thể dùng cách này để tính phương sai của phân phối. Sử dụng đẳng thức (C.27), ta có Var[X_i] = E[X_i^2] - E[X_i]. vì X_i chỉ nhận giá trị là 0 hoặc 1, ta có $X_i^2 = X_i$, tương đương với E[X_i^2] = E[X_i] = p. Vì thế,

Var[X_i] = $p - p^2 = p(1 - p) = pq$. (C.39)

Để tính phương sai của X, ta có thể lấy từ sự độc lập của n lần thử; bởi biểu thức (C.29),

$$\begin{aligned} Var[X] &= Var\left[\sum_{i=1}^{n} X_i\right] \\ &= \sum_{i=1}^{n} Var[X_i] \\ &= \sum_{i=1}^{n} pq \\ &= n\,p\,q \end{aligned} \qquad \text{(C.40)}$$

Như hình C.2 biểu diễn, phân phối nhị phân $b(k,n,p)$ tăng với k cho tới giá trị np, sau đó sẽ giảm. Ta có thể chứng minh rằng phân bố luôn diễn ra theo hướng ở trên bằng cách nhìn vào các tỷ lệ sau:

$$\begin{aligned} \frac{b(k;n,p)}{b(k-1;n,p)} &= \frac{\binom{n}{k} p^k q^{n-k}}{\binom{n}{k-1} p^{k-1} q^{n-k+1}} \\ &= \frac{n!(k-1)!(n-k+1)!\,p}{k!(n-k)!n!q} \\ &= \frac{(n-k+1)\,p}{kq} \\ &= 1 + \frac{(n+1)\,p-k}{kq} \end{aligned}$$

Thỉ lệ này lớn hơn 1 khi (n + 1)p - k là dương. Hệ quả là, $b(k,n,p) > b(k\text{-}1,n,p)$ khi $k < (n + 1)p$ (hàm phân bố tăng), và $b(k;n,p) < b(k\text{-}1;n,p)$ khi $k > (n + 1)p$ (hàm phân phối giảm). Nếu $k = (n + 1)p$ là số nguyên, khi đó $b(k;n,p) = b(k\text{-}1;n,p)$, và hàm phân phối có hai cực trị tại k = $(n + 1)p$ và k - 1= $(n + 1)p$ - 1 = $np - q$. Mặt khác, ta đạt được cựa đại tại điểm k duy nhất nằm trong khoảng $np - q < k < (n + 1)p$.

Suy luận sau cung cấp cận trên cho phân phối nhị phân

Suy luận C.1

Để $n \geq 0$, và $0 < p < 1$, $q = 1 - p$, và $0 \leq k \leq n$. Khi đó

$$b(k; n, p) \leq \left(\frac{np}{k}\right)^k \left(\frac{nq}{n-k}\right)^{n-k}.$$

Chứng minh Sử dụng (C.6), ta có

$$\begin{aligned} b(k; n, p) &= \binom{n}{k} p^k q^{n-k} \\ &\leq \left(\frac{n}{k}\right)^k \left(\frac{n}{n-k}\right)^{n-k} p^k q^{n-k} \\ &= \left(\frac{np}{k}\right)^k \left(\frac{nq}{n-k}\right)^{n-k} \end{aligned}$$

D. Ma trận

Ma trận phát sinh trong nhiều ứng dụng, bao gồm, nhưng không có nghĩa là chỉ giới hạn (???), khoa học máy tính. Nếu bạn đã từng biết đến ma trận, phần lớn tài liệu trong phụ lục này sẽ quen thuộc với bạn, nhưng sẽ có một số điều mới lạ hơn. Phần D.1 bao gồm định nghĩa cơ bản về ma trận và các phép tính toán trên ma trận, và phần D.2 sẽ trình bày một số đặc tính ma trận cơ bản.

D.1. Ma trận và các phép toán trên ma trận

Ở mục này, ta điểm qua một vài concepts cơ bản vè lý thuyết ma trận và những tính chất cơ bản của ma trận.

Ma trận và vector

Ma trận là một mảng số hình chữ nhật, ví dụ

$$\begin{aligned} A &= \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix} \\ &= \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \end{aligned} \tag{D.1}$$

là một ma trận 2 x 3 A = (a_{ij}), với i = 1,2 và j = 1, 2, 3, ta kí hiệu(?) phần tử tại hàng i, cột j là a_{ij}. Ta dùng kí tự in hoa để kí hiệu cho ma trận và kí tự thường tương ứng để kí hiệu cho các phần tử. Tập hợp các ma trận kích thước m x n với các giá trị thực được bao hàm bởi $\mathbf{R^{m\ x\ n}}$ và tập hợp các ma trận kích thước m x n nói chung được ký hiệu là S hay $\mathbf{S^{m\ x\ n}}$.

Ma trận chuyển vị của ma trận A là ma trận $\mathbf{A^T}$ bao gồm dong và cột đã được đảo vị trí của A. Cho ma trận A như đẳng thức D.1

$$A^T = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}.$$

Một vector là một mảng 1 chiều. Ví dụ:

$$x = \begin{pmatrix} 2 \\ 3 \\ 5 \end{pmatrix}$$

là một vector có kích thước là 3. Ta còn có thể gọi vector có độ dài n là n-vector. Dùng chữ thường để kí hiệu cho các vector, phần tử thứ i thuộc vector x kí hiệu là x_i (i = 1, 2, …, n). Quy ước, vector chuẩn là các vector cột, các vector tương ứng là các vector dòng.(….)

Vector đơn vị e_i là vector mà phần tử thứ i của nó bằng 1 và các phần tử còn lại bằng 0.

Ma trận vuông

D.2. Những đặc tính cơ bản của ma trận

Ma trận nghịch đảo, hạng ma trận và định thức

Nghịch đảo của ma trận A $n \times n$ là một ma trận $n \times n$, kí hiệu là A^{-1} (nếu nó tồn tại), sao cho $AA^{-1} = I_n = A^{-1}A$, ví dụ :

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{-1} = \begin{pmatrix} 0 & 1 \\ 1 & -1 \end{pmatrix}$$

Nhiều ma trận $n \times n$ khác ma trận 0 không có nghịch đảo. Một ma trận không có nghịc đảo được gọi là ma trận không khả nghịch hay ma trận suy biến. Dưới đây là ví dụ một ma trận suy biến :

$$\begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}$$

Nếu ma trận có nghịch đảo, nó được gọi là ma trận khả nghịch hay ma trận không suy biến.

Ma trận nghịch đảo, nếu có, là duy nhất. Nếu A và B là 2 ma trận $n \times n$ không suy biến thì $(BA)^{-1} = A^{-1}B^{-1}$

Phép nghịch đảo kết hợp với phép chuyển vị : $(A^T)^{-1} = (A^{-1})^T$.

Định lí D.1

Một ma trận vuông có số hạng tối đa khi và chỉ khi nó là ma trận khả nghịch.

Định lí D.2

Một ma trận có đầy đủ hạng theo cột nếu nó không chứa vecto 0

Hệ quả D.3

Ma trận A không khả nghịch khi và chỉ khi nó có 1 vecto 0 trong A

Chúng ta định nghĩa "định thức" của A là một số thực được xác định bằng

$$\det(A) = \begin{cases} a_{11} & khi\ n = 1 \\ \sum_{j=1}^{n} (-1)^{1+j} a_{1j} \det(A_{ij}) & khi\ n > 1 \end{cases}$$

Có thể xem $(-1)^{1+j} \det(A_{ij})$ là a_{ij}

Định lí D.4 (Tính chất của định thức)

Định thức của một ma trận vuông có những tính chất sau:

Nếu bất kì dòng hoặc cột nào bằng 0 thì det(A)=0

Ta có det(A) * λ sẽ bằng tất cả phần tử đầu của cột (hoặc dòng) nhân với λ

Một định thức sẽ không thay đổi nếu ta thục hiện nhân một dòng hoặc một cột nào đó với một số khác 0 rồi cộng vào các dòng hoặc các cột khác.

Ta có det(A)=det(A^T).

Nếu đổi vị trí hai dòng hoặc hai cột của một định thức thì giá trị định thức sẽ đổi dấu

Với 2 ma trận vuông A, B ta có det(AB)=det(A).det(B).

Định lí D.5

Ma trận không khả nghịch khi và chỉ det(A)=0.

Ma trận vuông A (n x n) gọi là xác định dương nếu x^T.Ax > 0 với x là n-vecto khác 0

Đặt x là vecto khác 0, x = (x1 x2 … xn), ta có $x^T I_n x = x^T x = \sum_{i=1}^{n} x_i^2 > 0$

Định lí D.6

Với mọi ma trận có đầy đủ hạng theo cột, ta có A^T.A xác định dương