

LỜI NÓI ĐẦU

Cấu trúc dữ liệu và giải thuật là một trong những môn học cơ bản của sinh viên ngành Công nghệ thông tin. Các cấu trúc dữ liệu và các giải thuật được xem như là 2 yếu tố quan trọng nhất trong lập trình, đúng như câu nói nổi tiếng của Niklaus Wirth: Chương trình = Cấu trúc dữ liệu + Giải thuật (Programs = Data Structures + Algorithms). Nắm vững các cấu trúc dữ liệu và các giải thuật là cơ sở để sinh viên tiếp cận với việc thiết kế và xây dựng phần mềm cũng như sử dụng các công cụ lập trình hiện đại.

Cấu trúc dữ liệu có thể được xem như là 1 phương pháp lưu trữ dữ liệu trong máy tính nhằm sử dụng một cách có hiệu quả các dữ liệu này. Và để sử dụng các dữ liệu một cách hiệu quả thì cần phải có các thuật toán áp dụng trên các dữ liệu đó. Do vậy, cấu trúc dữ liệu và giải thuật là 2 yếu tố không thể tách rời và có những liên quan chặt chẽ với nhau. Việc lựa chọn một cấu trúc dữ liệu có thể sẽ ảnh hưởng lớn tới việc lựa chọn áp dụng giải thuật nào.

Tài liệu “Cấu trúc dữ liệu và giải thuật” bao gồm 5 chương, trình bày về các cấu trúc dữ liệu và các giải thuật cơ bản nhất trong tin học.

Chương 1 bao gồm một số khái niệm cơ bản, tổng quan về giải thuật. Cách phân tích 1 vấn đề, cách thiết kế một giải pháp cho vấn đề theo cách giải quyết bằng máy tính. Tiếp theo, các phương pháp phân tích, đánh giá độ phức tạp và thời gian thực hiện giải thuật cũng được xem xét trong chương. Trình bày về đệ qui, một khái niệm rất cơ bản trong toán học và khoa học máy tính. Việc sử dụng đệ qui có thể xây dựng được những chương trình giải quyết được các vấn đề rất phức tạp chỉ bằng một số ít câu lệnh, đặc biệt là các vấn đề mang bản chất đệ qui.

Chương 2 trình bày về các cấu trúc dữ liệu được sử dụng rất thông dụng như mảng và danh sách liên kết, ngăn xếp và hàng đợi. Đó là các cấu trúc dữ liệu cũng rất gần gũi với các cấu trúc trong thực tiễn.

Chương 3 tập trung vào cây nhị phân và cây nhị phân tìm kiếm.

Chương 4 trình bày về các thuật toán sắp xếp và tìm kiếm. Các thuật toán này cùng với các kỹ thuật được sử dụng trong đó được coi là các kỹ thuật cơ sở cho lập trình máy tính. Các thuật toán được xem xét bao gồm các lớp thuật toán đơn giản và cả các thuật toán cài đặt phức tạp nhưng có thời gian thực hiện tối ưu.

Chương 5 trình bày về đồ thị bao gồm các phép duyệt đồ thị và một số bài toán trên đồ thị.

Cuối mỗi phần đều có các câu hỏi và bài tập để sinh viên ôn luyện và tự kiểm tra kiến thức của mình.

Về nguyên tắc, các cấu trúc dữ liệu và các giải thuật có thể được biểu diễn và cài đặt bằng bất cứ ngôn ngữ lập trình hiện đại nào. Tuy nhiên, để có được các phân tích sâu sắc hơn và có kết quả thực tế hơn, tác giả đã sử dụng ngôn ngữ lập trình C để minh họa cho các cấu trúc dữ liệu và thuật toán. Do vậy, ngoài các kiến thức cơ bản về tin học, người đọc cần có kiến thức về ngôn ngữ lập trình C.

Chương I. MỞ ĐẦU

A. Mục tiêu

Sau khi hoàn tất chương này sinh viên sẽ hiểu và nắm được các kiến thức cơ bản sau:

- Cấu trúc dữ liệu và mối quan hệ giữa cấu trúc dữ liệu và giải thuật.
- Các phương pháp mô tả giải thuật, đặc biệt phương pháp giả mã.
- Độ phức tạp của giải thuật
- Thiết kế giải thuật đệ quy và khử đệ quy

B. Nội dung

1.1. Khái niệm về cấu trúc dữ liệu và giải thuật

1.1.1. Khái niệm bài toán

Trong phạm vi Tin học, có thể quan niệm bài toán là bất cứ việc gì muốn máy tính thực hiện. Ví dụ như viết một dòng chữ ra màn hình, giải phương trình bậc hai, giải hệ phương trình bậc nhất hai ẩn số, quản lý cán bộ trong một cơ quan... được gọi là các bài toán.

Khi dùng máy tính giải bài toán cần quan tâm đến hai yếu tố: đưa vào máy tính thông tin gì (thông tin vào) và cần lấy ra thông tin gì (thông tin ra). Thông tin vào còn được gọi là Input, thông tin ra còn được gọi là Output. Phát biểu một bài toán chính là đặc tả Input và Output của bài toán.

Ví dụ:

- Bài toán tìm ước chung lớn nhất của hai số nguyên dương m, n :

+ Input: Hai số nguyên dương m và n ;

+ Output: Ước chung lớn nhất của m và n ;

- Bài toán kiểm tra tính nguyên tố của một số nguyên dương:

+ Input: Số nguyên dương n ;

+ Output: Trả lời câu hỏi n là số nguyên tố hay không;

Như vậy bài toán được cấu tạo bởi hai thành phần cơ bản:

- Thông tin vào (Input): Cho ta biết những thông tin đã có (giả thiết);

- Thông tin ra (Output): Các thông tin cần tìm từ Input (kết luận);

1.1.2. Khái niệm giải thuật (Thuật toán)

Giải thuật hay thuật toán - *Algorithm* là một khái niệm cơ sở của Tin học. Thuật ngữ “Algorithm” do nhà toán học Ả-rập Mohamed đề xuất vào năm 825. Đó là dãy các câu lệnh chặt chẽ và rõ ràng xác định một trình tự các thao tác trên một số đối tượng nào đó sao cho sau một số hữu hạn bước thực hiện đạt được kết quả mong muốn.

Hay có thể diễn tả thuật toán là một dãy hữu hạn các quy tắc, thao tác hay phép toán để giải quyết một vấn đề.

Ví dụ: Thuật toán Euclid để tìm ước chung lớn nhất (UCLN) của hai số nguyên dương m, n . Dãy các thao tác như sau:

- Tìm phần dư của phép chia m/n ;

- Nếu $r = 0$ thì $UCLN = n$;

- Trái lại thì đổi vai trò $m = n, n = r$ và lặp lại từ bước 1.

Các đặc trưng của thuật toán:

- Tính đúng đắn: Thuật toán cần đảm bảo cho một kết quả đúng sau khi thực hiện đối với các bộ dữ liệu đầu vào. Đây có thể nói là đặc trưng quan trọng nhất của thuật toán.

- Tính dừng: Thuật toán cần phải đảm bảo dừng sau một số hữu hạn bước.

- Tính xác định: Các bước của thuật toán phải được phát biểu rõ ràng, cụ thể, tránh gây nhập nhằng hoặc nhầm lẫn.

- Tính hiệu quả: Thuật toán được xem là hiệu quả nếu sau một số hữu hạn bước thực hiện sẽ giải quyết được bài toán với thời gian chấp nhận được.

- Tính phổ quát: Thuật toán được gọi là có tính phổ quát (phổ biến) nếu có thể giải quyết được một lớp các bài toán tương tự.

Ngoài ra mỗi thuật toán theo định nghĩa đều nhận các giá trị đầu vào được gọi là Input. Kết quả của thuật toán được gọi là Output.

Giải thuật chỉ phản ánh các phép xử lý, còn đối tượng để xử lý trên máy tính điện tử chính là dữ liệu chúng biểu diễn các thông tin cần thiết cho bài toán: các dữ kiện đưa vào, các kết quả trung gian... Không thể nói tới giải thuật mà mà không nghĩ tới giải thuật đó được tác động trên dữ liệu nào, còn khi xét tới dữ liệu thì cũng phải hiểu dữ liệu ấy cần được tác động giải thuật gì để đưa tới kết quả mong muốn.

1.1.3. Khái niệm cấu trúc dữ liệu và mối quan hệ với giải thuật

a. Khái niệm cấu trúc dữ liệu

Trong một bài toán, dữ liệu bao gồm một tập các phần tử cơ sở, mà ta gọi là dữ liệu nguyên tử (atoms). Nó có thể là một chữ số, một ký tự... nhưng cũng có thể là một con số, hay một từ..., điều đó tùy thuộc vào từng bài toán.

Trên cơ sở của các kiểu dữ liệu nguyên tử, các cung cách khả dĩ theo đó liên kết chúng lại với nhau sẽ dẫn tới các cấu trúc dữ liệu khác.

Khi đã xác định được cấu trúc dữ liệu thích hợp, người lập trình sẽ bắt đầu tiến hành xây dựng thuật giải tương ứng theo yêu cầu của bài toán đặt ra trên cơ sở của cấu trúc dữ liệu đã được chọn. Để giải quyết một vấn đề có thể có nhiều phương pháp, do vậy sự lựa chọn phương pháp phù hợp là một việc mà người lập trình phải cân nhắc và tính toán. Sự lựa chọn này cũng có thể góp phần đáng kể trong việc giảm bớt công việc của người lập trình trong phần cài đặt thuật toán trên một ngôn ngữ cụ thể.

Thường trong một ngôn ngữ lập trình bao giờ cũng có các cấu trúc dữ liệu tiên định. Nếu như sử dụng một ngôn ngữ mà cấu trúc tiên định của nó phù hợp với cấu trúc dữ liệu xác định bởi người dùng thì rất thuận lợi. Nhưng không phải mọi cấu trúc dữ liệu tiên định đều đáp ứng được mọi yêu cầu cần thiết. Vì vậy phải biết vận dụng linh hoạt chúng để mô phỏng các cấu trúc dữ liệu đó chọn cho bài toán cần giải.

Trong học phần này chúng ta sẽ lần lượt nghiên cứu những cấu trúc dữ liệu phức tạp hơn.

b. Mỗi quan hệ giữa cấu trúc dữ liệu với giải thuật

Khi giải quyết một bài toán trên máy tính, thường chỉ chú trọng đến việc xây dựng giải thuật mà quên đi tầm quan trọng của việc tổ chức dữ liệu trong bài toán. Giải thuật phản ánh các phép xử lý, còn đối tượng xử lý của giải thuật lại là dữ liệu. Chính dữ liệu chứa đựng các thông tin cần thiết để thực hiện giải thuật. Để xác định được giải thuật phù hợp cần phải biết nó tác động đến loại dữ liệu nào và khi lựa chọn cấu trúc dữ liệu cũng cần phải hiểu rõ những thao tác nào tác động lên dữ liệu đó. Như vậy, cấu trúc dữ liệu và giải thuật có mối quan hệ chặt chẽ với nhau, được thể hiện qua công thức:

$$\text{Cấu trúc dữ liệu} + \text{Giải thuật} = \text{Chương trình}$$

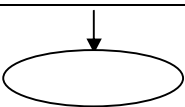
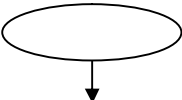
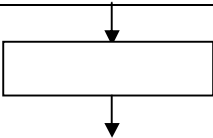
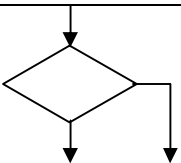
Với một cấu trúc dữ liệu đã chọn, sẽ có những giải thuật tương ứng, phù hợp. Khi cấu trúc dữ liệu thay đổi thường giải thuật cũng phải thay đổi theo để tránh việc xử lý gượng ép, thiếu tự nhiên trên một cấu trúc không phù hợp. Hơn nữa, một cấu trúc dữ liệu tốt sẽ giúp giải thuật xử lý trên đó có thể phát huy tác dụng tốt hơn, vừa nhanh vừa tiết kiệm vật tư, giải thuật cũng đơn giản và dễ hiểu hơn.

1.2. Mô tả giải thuật

1.2.1. Phương pháp sơ đồ khối

Có nhiều cách khác nhau để mô tả giải thuật. Cách dùng sơ đồ khối để minh họa giải thuật có lợi thế là rất trực quan, dễ hiểu. Tuy nhiên, với những bài toán lớn, phức tạp, việc dùng sơ đồ khối để mô tả giải thuật có những hạn chế nhất định. Hạn chế này có thể do khuôn khổ giấy, màn hình hay tầm bao quát theo dõi của mắt ta trên sơ đồ. Hạn chế này cũng nảy sinh khi bài toán của ta phức tạp, có nhiều vòng lặp lồng nhau, khi đó việc trình bày sơ đồ rất hạn chế.

Một số khối cơ bản:

			
<i>Bắt đầu</i>	<i>Kết thúc</i>	<i>Thực hiện công việc</i>	<i>Điều kiện</i>

1.2.2. Phương pháp liệt kê

Trong cách biểu diễn thuật toán theo ngôn ngữ tự nhiên, người ta sử dụng ngôn ngữ thường ngày để liệt kê các bước của thuật toán. Phương pháp biểu diễn này không yêu cầu người viết thuật toán cũng như người đọc thuật toán phải nắm các quy tắc. Tuy vậy, cách biểu diễn này thường dài dòng, không thể hiện rõ cấu trúc của thuật toán, đôi lúc gây hiểu lầm hoặc khó hiểu cho người đọc. Gần như không có một quy tắc cố định nào trong việc thể hiện thuật toán bằng ngôn ngữ tự nhiên. Tuy vậy, để dễ đọc, ta nên viết các bước con lùi vào bên phải và đánh số bước theo quy tắc phân cấp.

Ví dụ: Mô tả thuật giải tìm ước chung lớn nhất của hai số nguyên dương m và n :

- Bước 1: Chia m cho n tìm số dư r ;
- Bước 2: Nếu $r = 0$ thì thông báo kết quả UCLN là n , dừng giải thuật;
- Bước 3: Nếu $r \neq 0$ thì gán giá trị của n cho m , số dư r cho n và quay lại bước 1.

1.2.3. Phương pháp giả mã

Phương pháp thứ ba hay được dùng để mô tả giải thuật là dùng giả mã hay ngôn ngữ phỏng trình. Việc dùng ngôn ngữ phỏng trình khắc phục được nhiều nhược điểm của hai phương pháp trên, đặc biệt đối với những bài toán lớn, phức tạp.

Nói chung không có quy định bắt buộc về ngôn ngữ để trình bày thuật giải. Thông thường các giải thuật được trình bày bằng cách sử dụng một số yếu tố cấu trúc của ngôn ngữ lập trình tiêu biểu nhất, ví dụ ngôn ngữ Pascal hoặc C, kết hợp với cách diễn đạt khác, kể cả ngôn ngữ tự nhiên. Kết quả là một thứ tựa ngôn ngữ lập trình hay còn gọi là giả mã.

Ví dụ:

```
int      UCLN(m,n)
1.      r = m mod n;
2.      while(r != 0)
        {
            m = n;
            n = r;
            r = m mod n;
        }
3. return n.
```

Trong thực tế, tùy theo từng giai đoạn thiết kế giải thuật, tùy theo mức độ và đối tượng trình bày mà người lập trình chọn phương pháp thích hợp nhất trong ba phương pháp trên.

Như vậy, việc diễn tả giải thuật của một bài toán có nghĩa là trình bày các thao tác cần thực hiện theo một trình tự nào đó.

1.3. Độ phức tạp của giải thuật

Với mỗi vấn đề cần giải quyết, ta có thể tìm ra nhiều thuật toán khác nhau. Có những thuật toán thiết kế đơn giản, dễ hiểu, dễ lập trình và sửa lỗi, tuy nhiên thời gian thực hiện lớn và tiêu tốn nhiều tài nguyên máy tính. Ngược lại, có những thuật toán thiết kế và lập trình rất phức tạp, nhưng cho thời gian chạy nhanh hơn, sử dụng tài nguyên máy tính hiệu quả hơn. Khi đó, câu hỏi đặt ra là ta nên lựa chọn giải thuật nào để thực hiện?

Đối với những chương trình chỉ được thực hiện một vài lần thì thời gian chạy không phải là tiêu chí quan trọng nhất. Đối với bài toán kiểu này, thời gian để lập trình viên xây dựng và hoàn thiện thuật toán đáng giá hơn thời gian chạy của chương trình và như vậy những giải thuật đơn giản về mặt thiết kế và xây dựng nên được lựa chọn. Đối với những chương trình được thực hiện nhiều lần thì thời gian chạy của chương trình đáng giá hơn rất nhiều so với thời gian được sử dụng để thiết kế và xây dựng nó. Khi đó, lựa chọn một giải thuật có thời gian chạy nhanh hơn (cho dù việc thiết kế và xây dựng phức tạp hơn) là một lựa chọn cần thiết. Trong thực tế, trong giai đoạn đầu của việc giải quyết vấn đề, một giải thuật đơn giản thường được thực hiện trước như là 1 nguyên mẫu (prototype), sau đó nó sẽ được phân tích, đánh giá, và cải tiến thành các phiên bản tốt hơn.

1.3.1. Đánh giá thời gian thực hiện của thuật toán

Thời gian chạy của 1 chương trình phụ thuộc vào các yếu tố sau:

- Dữ liệu đầu vào
- Chất lượng của mã máy được tạo ra bởi chương trình dịch
- Tốc độ thực thi lệnh của máy
- Độ phức tạp về thời gian của thuật toán

Thông thường, thời gian chạy của chương trình không phụ thuộc vào giá trị dữ liệu đầu vào mà phụ thuộc vào kích thước của dữ liệu đầu vào. Do vậy thời gian chạy của chương trình nên được định nghĩa như là một hàm có tham số là kích thước dữ liệu đầu vào. Giả sử T là hàm ước lượng thời gian chạy của chương trình, khi đó với dữ liệu đầu vào có kích thước n thì thời gian chạy của chương trình là $T(n)$. Ví dụ, đối với một số chương trình thì thời gian chạy là an hoặc an^2 , trong đó a là hằng số. Đơn vị của hàm $T(n)$ là không xác định, tuy nhiên ta có thể xem như $T(n)$ là tổng số lệnh được thực hiện trên 1 máy tính lý tưởng.

Trong nhiều chương trình, thời gian thực hiện không chỉ phụ thuộc vào kích thước dữ liệu vào mà còn phụ thuộc vào tính chất của nó. Khi tính chất dữ liệu vào thỏa mãn một số đặc điểm nào đó thì thời gian thực hiện chương trình có thể là lớn nhất hoặc nhỏ nhất. Khi đó, ta định nghĩa thời gian thực hiện chương trình $T(n)$ trong trường hợp xấu nhất hoặc tốt nhất. Đó là thời gian thực hiện lớn nhất hoặc nhỏ nhất trong tất cả các bộ dữ liệu vào có kích thước n .

Ta cũng định nghĩa thời gian thực hiện trung bình của chương trình trên mọi bộ dữ liệu vào kích thước n . Trong thực tế, ước lượng thời gian thực hiện trung bình khó hơn nhiều so với thời gian thực hiện trong trường hợp xấu nhất hoặc tốt nhất, bởi vì việc phân tích thuật toán trong trường hợp trung bình khó hơn về mặt toán học, đồng thời khái niệm “trung bình” không có ý nghĩa thực sự rõ ràng.

Yếu tố chất lượng của mã máy được tạo bởi chương trình dịch và tốc độ thực thi lệnh của máy cũng ảnh hưởng tới thời gian thực hiện chương trình cho thấy chúng ta không thể thể hiện thời gian thực hiện chương trình dưới đơn vị thời gian chuẩn, chẳng hạn phút hoặc giây. Thay vào đó, ta có thể phát biểu thời gian thực hiện chương trình tỷ lệ với n hoặc n^2 v.v... Hệ số của tỷ lệ là 1 hằng số chưa xác định, phụ thuộc vào máy tính, chương trình dịch, và các nhân tố khác.

Ký hiệu **$O(n)$** : Để biểu thị cấp độ tăng của hàm, ta sử dụng ký hiệu $O(n)$. Ví dụ, ta nói thời gian thực hiện $T(n)$ của chương trình là $O(n^2)$, có nghĩa là tồn tại các hằng số dương c và n_0 sao cho $T(n) \leq cn^2$ với $n \geq n_0$.

Ví dụ: xét hàm $T(n) = (n+1)^2$. Ta có thể thấy $T(n)$ là $O(n^2)$ với $n_0 = 1$ và $c = 4$, vì ta có $T(n) = (n+1)^2 < 4n^2$ với mọi $n \geq 1$. Trong ví dụ này, ta cũng có thể nói rằng $T(n)$ là $O(n^3)$, tuy nhiên, phát biểu này “yếu” hơn phát biểu $T(n)$ là $O(n^2)$.

Nhìn chung, ta nói $T(n)$ là $O(f(n))$ nếu tồn tại các hằng số dương c và n_0 sao cho $T(n) < c.f(n)$ với $n \geq n_0$. Một chương trình có thời gian thực hiện là $O(f(n))$ thì được xem là có cấp độ tăng $f(n)$.

Việc đánh giá các chương trình có thể được thực hiện qua việc đánh giá các hàm thời gian chạy của chương trình, bỏ qua các hằng số tỷ lệ. Với giả thiết này, một chương trình với thời gian thực hiện là $O(n^2)$ sẽ tốt hơn chương trình với thời gian chạy $O(n^3)$. Bên cạnh các yếu tố hằng số xuất phát từ chương trình dịch và máy, còn có thêm hằng số từ bản thân chương trình.

Ví dụ, trên cùng một chương trình dịch và cùng 1 máy, chương trình đầu tiên có thời gian thực hiện là $100n^2$, trong khi chương trình thứ 2 có thời gian thực hiện là $5n^3$. Với n nhỏ, có thể $5n^3$ nhỏ hơn $100n^2$, tuy nhiên với n đủ lớn thì $5n^3$ sẽ lớn hơn $100n^2$ đáng kể.

Một lý do nữa để xem xét cấp độ tăng về thời gian thực hiện của chương trình là nó cho phép ta xác định độ lớn của bài toán mà ta có thể giải quyết. Mặc dù máy tính có tốc độ ngày càng cao, tuy nhiên, với những chương trình có thời gian thực hiện có cấp độ tăng lớn (từ n^2 trở lên), thì việc tăng tốc độ của máy tính tạo ra sự khác biệt không đáng kể về kích thước bài toán có thể xử lý bởi máy tính trong một khoảng thời gian cố định.

Bảng sau đây cho một số cấp thời gian thực hiện thuật toán được sử dụng rộng rãi và tên gọi thông thường của chúng.

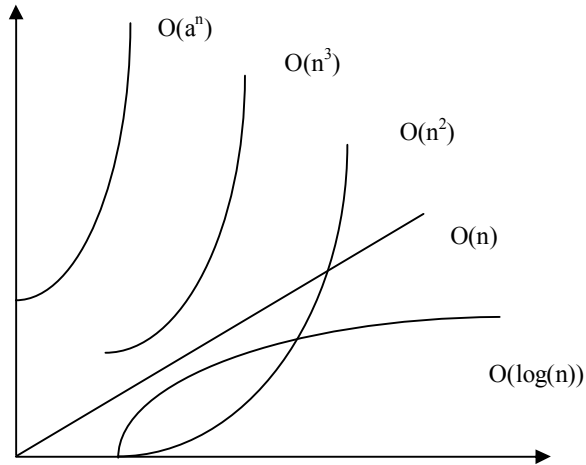
Ký hiệu ô lớn	Tên gọi thông thường
$O(\log_2 n)$	logarit
$O(n)$	Tuyến tính
$O(n \log_2 n)$	$n \log_2 n$
$O(n^2)$	Bình phương
$O(n^3)$	Lập phương
$O(2^n)$	Hàm mũ
$O(n!)$	Hàm mũ

$O(n^n)$	Hàm mũ
----------	--------

Danh sách trên sắp xếp theo thứ tự tăng dần của cấp thời gian thực hiện.

Các hàm như $\log_2 n$, n , $n \log_2 n$, n^2 , n^3 được gọi là các hàm đa thức. Giải thuật với thời gian thực hiện có cấp hàm đa thức thì thường chấp nhận được.

Các hàm như 2^n , $n!$, n^n được gọi là hàm loại mũ. Một giải thuật mà thời gian thực hiện của nó là các hàm loại mũ thì tốc độ rất chậm.



Phác họa đường biểu diễn độ phức tạp của một số giải thuật

1.3.2. Một số qui tắc xác định thời gian thuật toán

Qui tắc tổng: Giả sử $T_1(n)$ và $T_2(n)$ là thời gian thực hiện của hai giai đoạn chương trình P_1 và P_2 mà $T_1(n) = O(f(n))$; $T_2(n) = O(g(n))$ thì thời gian thực hiện đoạn P_1 rồi P_2 tiếp theo sẽ là $T_1(n) + T_2(n) = O(\max(f(n), g(n)))$.

Ví dụ: Trong một chương trình có 3 bước thực hiện mà thời gian thực hiện từng bước lần lượt là $O(n^2)$, $O(n^3)$ và $O(n \log_2 n)$ thì thời gian thực hiện 2 bước đầu là $O(\max(n^2, n^3)) = O(n^3)$. Khi đó thời gian thực hiện chương trình sẽ là $O(\max(n^3, n \log_2 n)) = O(n^3)$.

Qui tắc nhân: Nếu tương ứng với P_1 và P_2 là $T_1(n) = O(f(n))$, $T_2(n) = O(g(n))$ thì thời gian thực hiện P_1 và P_2 lồng nhau sẽ là: $T_1(n)T_2(n) = O(f(n)g(n))$

Ví dụ:

- Câu lệnh gán: $x = x + 1$ có thời gian thực hiện $O(1)$

- Câu lệnh: **for**($i = 0; i < n; i++$) $x = x + 1$;

có thời gian thực hiện $O(n.1) = O(n)$

- Câu lệnh: **for**(i = 0; i < n; i++)

for(j = 0; j < n; j++) x = x + 1;

có thời gian thực hiện được đánh giá là $O(n.n) = O(n^2)$

Giả sử rằng, các lệnh gán không chứa các lời gọi hàm. Khi đó để đánh giá thời gian thực hiện một chương trình, ta có thể áp dụng phương pháp đệ qui sau:

1. Thời gian thực hiện các lệnh đơn: gán, đọc, viết, so sánh... là $O(1)$
2. Lệnh ghép: $\{S_1, S_2, \dots, S_p\}$; thời gian thực hiện được xác định bởi luật tổng.
3. Lệnh **if**: **if** (đk) S_1 **else** S_2 . Giả sử thời gian thực hiện các lệnh S_1, S_2 là $O(f(n))$ và $O(g(n))$. Khi đó thời gian thực hiện lệnh **if** là $O(\max(f(n), g(n)))$
4. Lệnh **while**: **while** (đk) S. Giả sử thời gian thực hiện lệnh S (thân của **while**) là $O(f(n))$. Giả sử g(n) là số lần lặp tối đa. Khi đó thời gian thực hiện lệnh là $O(f(n)g(n))$.
5. Lệnh **for**: Lệnh này được đánh giá tương tự như lệnh **while**.

Chú ý 1: Dựa vào các quy tắc xác định độ phức tạp tính toán của giải thuật phải chú ý tới các bước tương ứng với một phép toán gọi là phép toán tích cực. Phép toán tích cực là phép toán mà thời gian thực hiện nó không ít hơn các phép toán khác.

Ví dụ: Cho đoạn chương trình:

```
for(i = 0; i < n-1; i++)  
    for(j = i+1; j < n; j++)  
        if(a[j] < a[i])  
        {  
            tam = a[i];  
            a[i] = a[j];  
            a[j] = tam;  
        }
```

Phép toán tích cực ở đây là phép so sánh $a[j] < a[i]$.

Chú ý 2: Việc xác định độ phức tạp tính toán nhiều khi còn phụ thuộc vào tình trạng của dữ liệu. Do đó, phải xét $T(n)$ trong trường hợp thuận lợi nhất, trong trường hợp xấu nhất và trường hợp trung bình. Tuy nhiên việc xác định độ phức tạp trung

bình thường khó khăn nên người ta thường lấy độ phức tạp trong trường hợp xấu nhất để làm độ phức tạp tính toán của giải thuật.

1.3.3. Phân tích một số thuật toán

Ví dụ 1: Phân tích thuật toán Euclid

int UCLN(m,n) (1)

1. $r = m \bmod n;$ (2)

2. **while**($r \neq 0$) (3)

{

$m = n;$ (4)

$n = r;$ (5)

$r = m \bmod n;$ (6)

}

3. **return** n;

Lệnh (2) có thời gian thực hiện là $O(1)$ vì chúng là câu lệnh đọc và câu lệnh gán. Do đó ta đánh giá thời gian thực hiện câu lệnh (3). Thân của lệnh này, là khối gồm ba lệnh (4), (5) và (6). Mỗi lệnh có thời gian thực hiện là $O(1)$. Do đó khối có thời gian thực hiện là $O(1)$. Ta còn phải đánh giá số lớn nhất các lần thực hiện lặp khối.

Ta có: $m = n.q_1 + r_1, 0 \leq r_1 < n$

$$n = r_1.q_2 + r_2, 0 \leq r_2 < r_1$$

- Nếu $r_1 \leq n/2$ thì $r_2 < r_1 \leq n/2$, do đó $r_2 < n/2$

- Nếu $r_1 > n/2$ thì $q_2 = 1$, tức là $n = r_1 + r_2$, do đó $r_2 < n/2$.

Tóm lại, ta luôn có $r_2 < n/2$.

Như vậy cứ hai lần thực hiện khối lệnh thì phần dư r giảm đi còn một nửa của n . Gọi k là số nguyên lớn nhất sao cho $2^k \leq n$. Suy ra số lần lặp tối đa là $2k + 1 \leq 2\log_2 n + 1$. Do đó thời gian thực hiện lệnh **while** là $O(\log_2 n)$. Đó cũng là thời gian thực hiện của thuật toán.

Ví dụ 2: Giải thuật tính giá trị của e^x tính theo công thức gần đúng:

$$e^x = 1 + x/1! + x^2/2! + \dots + x^n/n!, \text{ với } x \text{ và } n \text{ cho trước}$$

Câu lệnh (1) có thời gian thực hiện là $O(1)$. Do đó thời gian thực hiện giải thuật phụ thuộc vào câu lệnh (2). Vì hai câu lệnh (3) và (4) đều có thời gian thực hiện là $O(n)$ nên thời gian thực hiện của giải thuật là $O(n)$

Như vậy từ hai giải thuật trên ta có thể nói rằng giải thuật thứ hai tốt hơn giải thuật thứ nhất với n đủ lớn (với n nhỏ thì thời gian thực hiện hai giải thuật này tương đương nhau).

1.4. Độ quy

1.4.1. Khái niệm đệ quy

Đệ quy là một khái niệm cơ bản trong toán học và khoa học máy tính. Một đối tượng là đệ quy nếu nó bao gồm chính nó như một bộ phận hoặc nó được định nghĩa dưới dạng của chính nó.

Ví dụ:

- Số tự nhiên:

+ 1 là số tự nhiên.

+ n là số tự nhiên nếu $n-1$ là số tự nhiên.

- Hàm giai thừa: $n!$

+ $0! = 1$

+ $n! = (n-1)! * n$

1.4.2. Giải thuật đệ quy và thủ tục đệ quy

Nếu lời giải của của một bài toán T được giải bằng lời giải của một bài toán T' , có dạng giống như T , thì lời giải đó được gọi là lời giải đệ quy. Giải thuật tương ứng với lời giải đệ quy gọi là giải thuật đệ quy.

Ở đây T' có dạng giống T nhưng theo một nghĩa nào đó T' phải “nhỏ” hơn T . Cách hiểu nhỏ hơn ở đây là nhỏ hơn về kích thước dữ liệu.

Chẳng hạn với bài toán tính $n!$ thì $n!$ là bài toán T còn $(n-1)!$ là bài toán T' ta thấy T' cùng dạng với T nhưng nhỏ hơn ($n-1 < n$).

Một hàm hay thủ tục đệ quy bao gồm hai phần chính sau:

- Phần neo (ứng với trường hợp suy biến): là phần mà tác động của hàm hay thủ tục được đặc tả cho một hay nhiều tham số. Phần neo nhằm đảm bảo tính dừng của giải thuật.

- Phần đệ quy (phần quy nạp): là phần mà tác động cần được thực hiện cho giá trị hiện thời của các tham số được định nghĩa bằng các giá trị được định nghĩa trước.

Trong các ngôn ngữ lập trình cấp cao nếu một giải thuật chứa lời gọi đến chính nó thì gọi là đệ quy trực tiếp, còn nếu lời gọi đến chính nó phải thông qua một hay một số giải thuật khác thì gọi là đệ quy gián tiếp.

1.4.3. Thiết kế giải thuật đệ quy

Khi bài toán đang xét hoặc dữ liệu đang xử lý được định nghĩa dưới dạng đệ quy thì việc thiết kế các giải thuật đệ quy tỏ ra rất thuận lợi. Hầu như nó phản ánh rất sát nội dung của định nghĩa đó.

Ta xét một số bài toán sau:

a. Bài toán n!

Hàm này được định nghĩa như sau:

$$Factorial(n) = \begin{cases} 1 & n = 0 \\ n * Factorial(n-1) & n > 0 \end{cases}$$

Giải thuật đệ quy được viết dưới dạng hàm dưới đây:

long GT(n)

```
1.      if(n=0)      GT=1
           else      GT = n*GT(n-1);
```

```
2. ReturnGT;
```

Trong hàm trên lời gọi đến nó nằm ở câu lệnh gán sau else.

Mỗi lần gọi đệ quy đến GT, thì giá trị của n giảm đi 1. Ví dụ, GT(4) gọi đến GT(3), gọi đến GT(2), gọi đến GT(1), gọi đến GT(0) đây là trường hợp suy biến, nó được tính theo cách đặc biệt $GT(0) = 1$.

b. Bài toán dãy số Fibonacci.

$$F(n) = \begin{cases} 1 & n \leq 2 \\ F(n-2) + F(n-1) & n > 2 \end{cases}$$

Dãy số thể hiện $F(n)$ ứng với các giá trị của $n = 1, 2, 3, 4, \dots$, có dạng: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, được gọi là dãy số Fibonacci.

Sau đây là thủ tục đệ quy thể hiện giải thuật tính $F(n)$.

int F(n)

```
1.      if(n<=2)      F=1  
      else F = F(n-2) + F(n-1);
```

```
2. ReturnF;
```

Ở đây trường hợp suy biến ứng với 2 giá trị $F(1) = 1$ và $F(2) = 1$.

Chú ý:

Đối với hai bài toán nêu trên thì việc thiết kế các giải thuật đệ quy tương ứng khá thuận lợi vì cả hai đều thuộc dạng tính giá trị hàm mà định nghĩa của nó xác định được dễ dàng.

Nhưng không phải lúc nào tính đệ quy trong cách giải bài toán cũng thể hiện rõ nét và đơn giản như vậy. Việc thiết kế một giải thuật đệ quy đòi hỏi phải giải đáp được các câu hỏi sau:

- Có thể định nghĩa được bài toán dưới dạng một bài toán cùng loại, nhưng nhỏ hơn như thế nào?

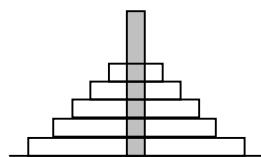
- Như thế nào là kích thước của bài toán được giảm đi ở mỗi lần gọi đệ quy?

- Trường hợp đặc biệt nào của bài toán được gọi là trường hợp suy biến?

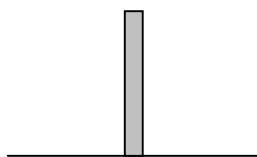
Sau đây ta xét thêm bài toán phức tạp hơn.

c. Bài toán “Tháp Hà Nội”.

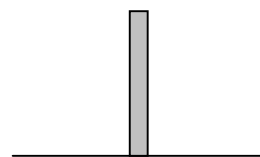
Bài toán: Có n đĩa, kích thước nhỏ dần, và có thể xếp chồng lên nhau xuyên qua một cọc, đĩa to ở dưới, đĩa nhỏ ở trên để cuối cùng có một chồng đĩa dạng như hình tháp như hình dưới đây.



A



B



C

Yêu cầu đặt ra là:

Chuyển chồng đĩa từ cọc A sang cọc khác, chẳng hạn cọc C, theo những điều kiện:

- Mỗi lần chỉ được chuyển một đĩa.

- Không khi nào có tình huống đĩa to ở trên đĩa nhỏ (dù là tạm thời).
- Được phép sử dụng một cọc trung gian, chẳng hạn cọc B để đặt tạm đĩa (gọi là cọc trung gian).

Để đi tới cách giải tổng quát, trước hết ta xét vài trường hợp đơn giản.

*) Trường hợp có 1 đĩa: Chuyển đĩa từ cọc A sang cọc C.

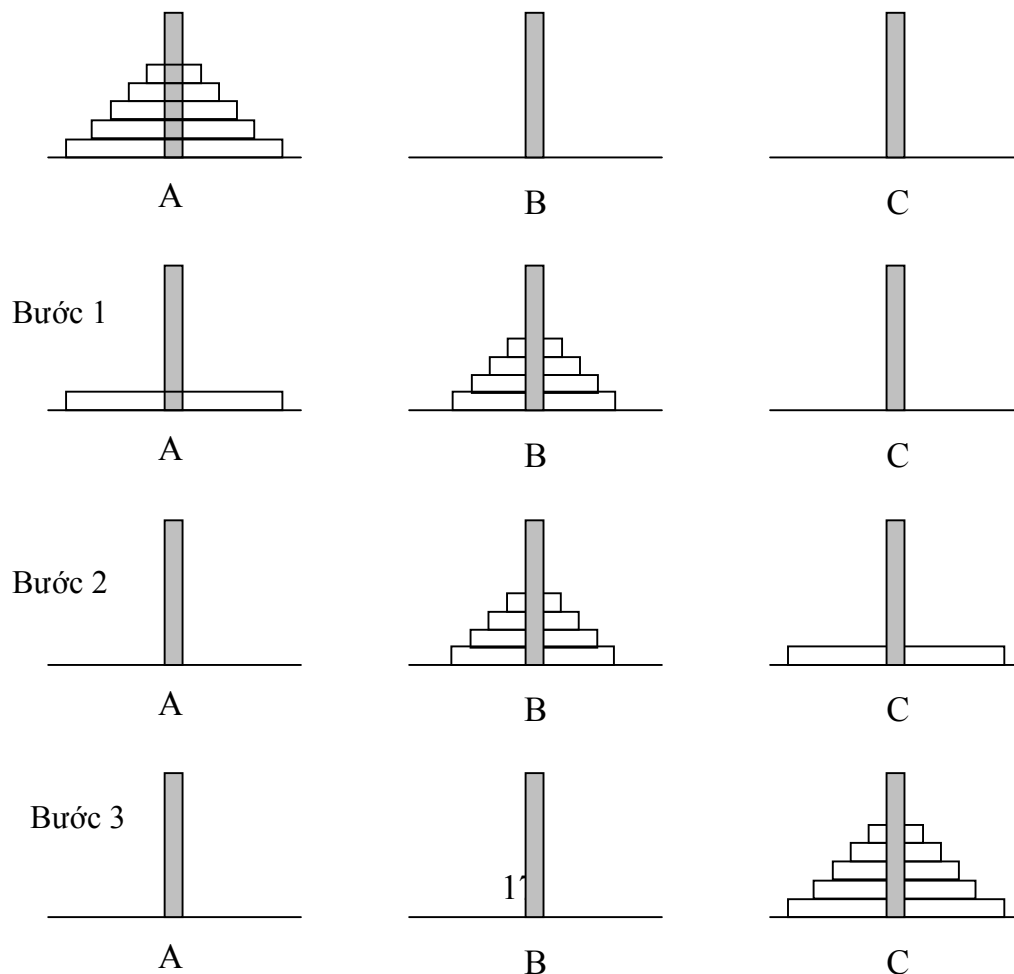
*) Trường hợp 2 đĩa:

- Chuyển đĩa thứ nhất từ cọc A sang cọc B.
- Chuyển đĩa thứ hai từ cọc A sang cọc C.
- Chuyển đĩa thứ nhất từ cọc B sang cọc C.

Ta thấy với trường hợp n đĩa ($n > 2$) nếu coi $n-1$ đĩa ở trên, đóng vai trò như đĩa thứ nhất thì có thể xử lý giống như trường hợp 2 đĩa được, nghĩa là:

- Chuyển $n-1$ đĩa trên từ A sang B.
- Chuyển đĩa thứ n từ A sang C.
- Chuyển $n-1$ đĩa từ B sang C.

Lược đồ thể hiện 3 bước này như sau:



Như vậy, bài toán “Tháp Hà Nội” tổng quát với n đĩa đã được dẫn đến bài toán tương tự với kích thước nhỏ hơn là chuyển $n-1$ đĩa và cứ như thế cho tới khi trường hợp suy biến xảy ra, đó là trường hợp ứng với bài toán chuyển 1 đĩa.

Các đặc điểm của đệ quy trong giải thuật đã được xác định do đó có thể viết giải thuật đệ quy của bài toán “Tháp Hà Nội” như sau:

```
void ThapHN(n, A, B, C)
```

```
1. if(n==1) chuyển đĩa từ A sang C
```

```
2. else
```

```
{ ThapHN(n-1, a, C, B);  
  ThapHN(1, A, B, C);  
  ThapHN(n-1, B, A, C) ;  
}
```

```
3. return;
```

Cài đặt:

```
#include "conio.h"  
#include "stdio.h"  
void ThapHN(int n,int a,int b,int c)  
{if (n==1)  
    printf("%d->%d\n",a,c);  
else  
{  
    ThapHN(n-1,a,c,b);  
    ThapHN(1,a,b,c);  
    ThapHN(n-1,b,a,c);  
}  
}  
void main()  
{int x;  
printf("\nNhap so dia:");
```

```
scanf ("%d", &x) ;
ThapHN (x, 1, 2, 3) ;
getch() ;
}
```

1.4.4. Hiệu lực của đệ quy

Có những bài toán, bên cạnh giải thuật đệ quy vẫn có những giải thuật lặp khá đơn giản và hữu hiệu. Chẳng hạn giải thuật lặp tính $n!$ có thể viết:

```
long Factorial(n)
```

```
1. if (n=0 || n=1)
```

```
    gt=1
```

```
2. else
```

```
    {    gt=1;
```

```
        for(i=2;i<=n;i++)
```

```
            gt = gt*i;
```

```
    }
```

```
2. return gt;
```

Hoặc ta xét giải thuật lặp tính số Fibonacci thứ n :

```
int Fibonacci(n)
```

```
1. if(n<=2)
```

```
    Fibonacci = 1
```

```
2. else
```

```
    {    Fib1 = 1; Fib2 = 1;
```

```
        for(i=3; i<=n; i++)
```

```
            {    Fibn = Fib1 + Fib2;
```

```
                Fib1 = Fib2;
```

```
                Fib2 = Fibn;
```

```
            }
```

```
        Fibonacci = Fibn;
```

```
2. return Fibonacci;
```

End;

Tuy vậy, đệ quy vẫn có vai trò xứng đáng của nó. Có những bài toán việc nghĩ ra giải thuật đệ quy thuận lợi hơn nhiều so với giải thuật lặp và có những giải thuật đệ quy thực sự có hiệu lực cao, chẳng hạn giải thuật sắp xếp kiểu phân đoạn (Quick Sort) hoặc các giải thuật duyệt cây nhị phân.

Một điều nữa cần nói thêm là: về mặt định nghĩa, công cụ đệ quy đã cho phép xác định một tập vô hạn các đối tượng bằng một phát biểu hữu hạn. Ta sẽ thấy vai trò của công cụ này trong định nghĩa văn phạm, định nghĩa cú pháp ngôn ngữ, định nghĩa một số cấu trúc dữ liệu v.v...

Chú ý: Khi thay các giải thuật đệ quy bằng các giải thuật lặp tương ứng ta gọi là khử đệ quy. Tuy nhiên có những bài toán việc khử đệ quy tương đối đơn giản (ví dụ: giải thuật tính $n!$, tính số fibonacci ...), nhưng có những bài toán việc khử đệ quy là rất phức tạp (ví dụ: bài toán tháp Hà Nội, giải thuật sắp xếp phân đoạn...).

1.5. Phân tích và thiết kế giải thuật

1.5.1. Mô-đun hoá và việc giải quyết bài toán

Các bài toán giải được trên máy tính điện tử ngày càng đa dạng và phức tạp. Các giải thuật và chương trình để giải chúng cũng ngày càng có quy mô lớn và càng khó khi thiết lập cũng như khi muốn tìm hiểu.

Tuy nhiên, ta cũng thấy mọi việc sẽ đơn giản hơn nếu có thể phân chia bài toán lớn thành các bài toán nhỏ. Điều đó có nghĩa là nếu coi bài toán của ta như một mô-đun chính thì cần phân chia nó thành các mô-đun con. Mỗi mô-đun con này lại được chia thành nhiều mô-đun tương ứng với các phần việc cơ bản mà ta đã biết cách giải quyết. Như vậy việc tổ chức lời giải các bài toán được thể hiện theo một cấu trúc phân cấp.

Chiến thuật giải quyết bài toán theo tinh thần như vậy chính là chiến thuật “Chia để trị”. Để thể hiện chiến thuật đó, người ta dùng cách thiết kế “từ đỉnh xuống” (top-down). Đó là phân tích tổng quát toàn bộ vấn đề xuất phát từ dữ liệu và các mục tiêu đặt ra để đề ra các công việc chủ yếu rồi mới đi dần vào giải quyết các vấn đề cụ thể một cách chi tiết hơn.

Cách thiết kế giải thuật theo kiểu top-down giúp cho việc giải quyết bài toán được định hướng rõ ràng, tránh sa đà vào các chi tiết phụ. Nó cũng là nền tảng cho lập trình có cấu trúc.

Thông thường, đối với các bài toán lớn, việc giải quyết nó phải do nhiều người cùng làm. Chính phương pháp mô-đun hóa sẽ cho phép tách bài toán ra thành các phần độc lập tạo điều kiện cho các nhóm giải quyết phần việc của mình mà không ảnh hưởng gì đến nhóm khác. Với chương trình được xây dựng trên cơ sở của các giải thuật được thiết kế theo cách này thì việc tìm hiểu cũng như sửa chữa chính lý sẽ dễ dàng hơn.

Việc phân bài toán thành các bài toán con không phải là một việc dễ dàng. Chính vì vậy mà có những bài toán nhiệm vụ phân tích và thiết kế giải thuật mất nhiều thời gian hơn nhiệm vụ lập trình.

1.5.2. Phương pháp tinh chỉnh từng bước (Stepwise refinement)

Tinh chỉnh từng bước là phương pháp thiết kế giải thuật gắn liền với lập trình. Nó phản ánh tinh thần của quá trình mô-đun hoá bài toán và thiết kế kiểu top-down.

Đầu tiên giải thuật được trình bày bằng ngôn ngữ tự nhiên phản ánh ý chính của công việc cần làm. Từ các bước sau, những lời những ý đó sẽ được chi tiết hoá dần dần tương ứng với những công việc nhỏ hơn.

Ta gọi đó là các bước tinh chỉnh, sự tinh chỉnh này sẽ được hướng về phía ngôn ngữ lập trình mà ta đã chọn ở các bước trung gian pha tạp cả ngôn ngữ tự nhiên lẫn ngôn ngữ lập trình.

Trong quá trình tinh chỉnh này dữ liệu cũng được tinh chế dần từ dạng cấu trúc đến dạng lưu trữ (cách biểu diễn 1 cấu trúc dữ liệu trong bộ nhớ được gọi là cấu trúc dữ liệu)

Ví dụ: Lập trình sắp xếp 1 dãy n số nguyên khác nhau theo thứ tự tăng dần

Có thể phác thảo giải thuật như sau:

Từ dãy số nguyên chưa được sắp xếp chọn ra số nhỏ nhất đặt ở cuối dãy đã được sắp xếp. Cứ lặp lại quy trình đó cho tới khi dãy chưa được sắp xếp trở thành rỗng.

* Bước tinh chỉnh đầu tiên sẽ là như sau:

for ($i=$; $i \leq n$; $i++$)

- { - xét từ a_i đến a_n để tìm số nhỏ nhất a_j
- Đổi chỗ giữa a_i và a_j
- }

Ta thấy có hai nhiệm vụ:

- Tìm số nguyên nhất a_j trong các số từ a_i đến a_n .
- Đổi chỗ a_j với a_i

Nhiệm vụ đầu có thể thực hiện bằng cách:

“Thoạt đầu tiên coi a_i là “số nhỏ nhất” tạm thời; lần lượt so sánh a_i với a_{i+1}, a_{i+2}, \dots khi thấy số nào nhỏ nhất thì lại coi đó là “số nhỏ nhất” mới. Khi đã so sánh với a_n rồi thì số nhỏ nhất sẽ được xác định”.

Nhưng xác định bằng cách nào?

- Có thể bằng cách chỉ ra chỗ của nó, nghĩa là nắm được chỉ số của phần tử ấy. Ta có bước tinh chỉnh:

```
j=i;
for (k=j+1; k<=n; k++)
    if (a_k < a_j)
        j=k;
```

Với nhiệm vụ thứ hai ta có bước tinh chỉnh:

$B = a_i; \quad a_i = a_j; \quad a_j = B;$

Chương trình sắp xếp dưới dạng thủ tục như sau:

Void SORT(A,n)

1. **for** (i=1; i<=n; i++)
2. {Chọn số nhỏ nhất}


```
j=i
for (k=j+1; k<=n; k++)
    if (A[k]<A[j]) j=k;
```
3. {Đổi chỗ} $B=A[i]; A[i]=A[j]; A[j]=B$
4. **Return**

Bài tập chương I

1.1 . Tìm ví dụ minh họa mối quan hệ giữa cấu trúc dữ liệu và giải thuật.

1.2. Các cấu trúc dữ liệu tiên định trong một ngôn ngữ có đủ đáp ứng yêu cầu về tổ chức dữ liệu không?

1.3. Hãy nêu một giải thuật mà độ phức tạp tính toán là $O(1)$.

1.4. Cho $T(n) = O(n)$. Chứng minh rằng $T(n) = O(n^2)$.

1.5. Với các đoạn chương trình dưới đây hãy xác định độ phức tạp tính toán của giải thuật bằng ký pháp chữ O lớn trong trường hợp tồi nhất.

a) `sum = 0;`

```
for(i = 1; i<=n; i++)  
{  
    scanf(x);  
    sum = sum + 1;  
}
```

b) `for(i = 1; i<=n; i++)`

```
    for(j = 1; j<=n; j++)  
    {  
        C[i,j] = 0;  
        for(k = 1; k<=n; k+)  
            C[i,j] = C[i,j] + A[i,k] + B[k,j];  
    }
```

c) `for(i = 1; i<n; i++)`

```
{  
    for(j = 1; j<n; j++)  
        if (X[j] > X[j + 1])  
        {  
            tg = X[j];
```

```

        X[j] = X[j + 1];
        X[j + 1] = tg;
    }
}

```

1.6. Cho một ma trận kích thước $M \times N$ gồm các số nguyên (có cả số âm và dương). Hãy viết chương trình tìm ma trận con của ma trận đã cho sao cho tổng các phần tử trong ma trận con đó lớn nhất có thể được (bài toán maximum sum plateau). Hãy đưa ra đánh giá về độ phức tạp của thuật toán sử dụng.

1.7. Việc chia bài toán ra thành các bài toán nhỏ có những thuận lợi gì?

1.8. Hàm $C(n, k)$ với n, k là các giá trị nguyên không âm và $k \leq n$, được định nghĩa:

$$C(n, n) = 1$$

$$C(n, 0) = 1$$

$$C(n, k) = C(n - 1, k - 1) + C(n - 1, k) \text{ nếu } 0 < k < n$$

Viết giải thuật tính $C(n, k)$ khi biết n, k .

Chương II. CẤU TRÚC DỮ LIỆU TUYẾN TÍNH VÀ CÁCH CÀI ĐẶT

A. Mục tiêu

Sau khi hoàn tất chương này sinh viên sẽ hiểu và vận dụng được các kiến thức cơ bản sau:

- Đặc điểm của cấu trúc dữ liệu tuyến tính.
- Các giải thuật cơ bản trên danh sách nối đơn, nối kép. Cài đặt thành công các giải thuật.
- Ngăn xếp, hàng đợi và ứng dụng.

B. Nội dung

2.1. Phân loại các cấu trúc dữ liệu

Kiểu dữ liệu là một tập các giá trị cùng với tập các thao tác được xác định trên những giá trị này. Những kiểu dữ liệu đơn giản (còn gọi là kiểu vô hướng hay kiểu nguyên thủy) là những kiểu được tạo nên từ những giá trị không thể phân tách thành những phần tử nhỏ hơn.

Ví dụ: Trong ngôn ngữ lập trình Pascal thì Integer, Real, Char, Boolean là những kiểu dữ liệu đơn giản.

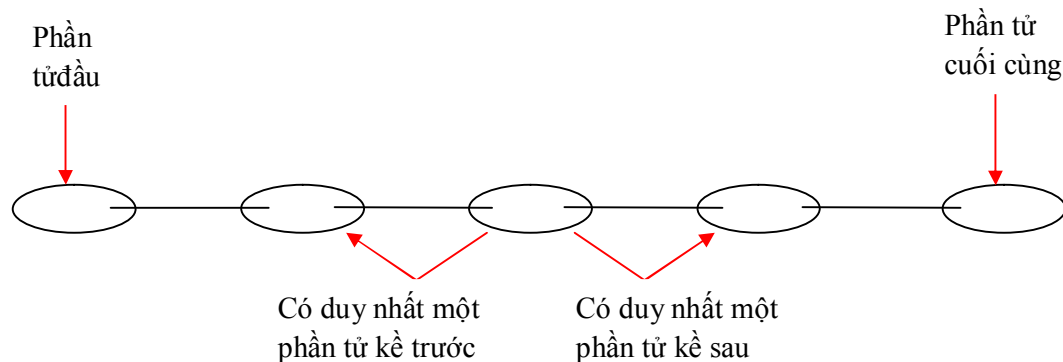
Những kiểu dữ liệu phức hợp (còn được gọi là những cấu trúc dữ liệu) được tạo nên từ những phần tử mà mỗi phần tử đó đều có thể phân tách được thành các thành phần thuộc kiểu dữ liệu đơn giản hay thuộc kiểu dữ liệu phức hợp khác.

Ví dụ: Kiểu mảng, kiểu bản ghi là kiểu dữ liệu có cấu trúc.

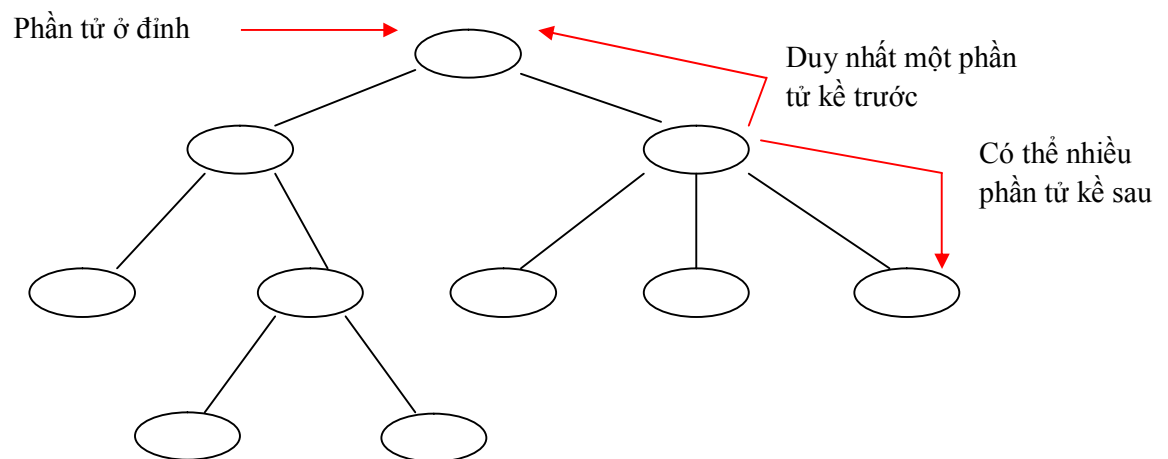
Tuy nhiên đó vẫn là những kiểu dữ liệu do một ngôn ngữ lập trình cung cấp sẵn. Việc lập trình giải quyết một bài toán thực tế có thể được tiến hành qua hai giai đoạn. Trước hết chúng ta thể hiện thế giới thực của bài toán bằng mô hình toán học phù hợp, tức là chọn và xây dựng kiểu dữ liệu trừu tượng phù hợp. Sau đó ta cài đặt các kiểu dữ liệu trừu tượng này từ các kiểu dữ liệu có sẵn của một ngôn ngữ lập trình cụ thể, rồi trên cơ sở đó cài đặt các thuật toán để có kết quả nhờ máy tính điện tử. Những kiểu dữ liệu mới được xây dựng từ những kiểu dữ liệu tiền định của một ngôn ngữ lập trình cũng là những kiểu dữ liệu có cấu trúc.

Các kiểu dữ liệu có cấu trúc có thể được phân loại dựa vào mối quan hệ giữa các phần tử của kiểu dữ liệu đó. Có bốn loại cấu trúc dữ liệu:

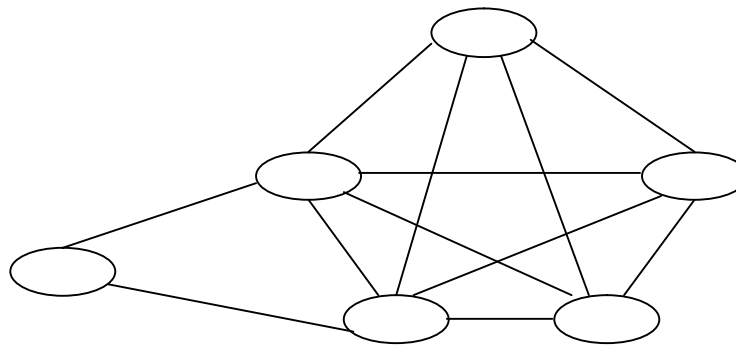
- **Cấu trúc dữ liệu tuyến tính:** Đặc điểm của những cấu trúc dữ liệu thuộc loại này là giữa các phần tử có mối quan hệ một - một, hiểu theo nghĩa: mỗi phần tử (trừ phần tử đầu và phần tử cuối) đều có đúng một phần tử kề trước và có một phần tử kề sau.



- **Cấu trúc dữ liệu phân cấp:** Đặc điểm của những cấu trúc dữ liệu thuộc kiểu này là giữa các phần tử có mối quan hệ một - nhiều, hiểu theo nghĩa: mỗi phần tử có thể có nhiều phần tử kề sau (gọi là các phần tử con) nhưng chỉ có đúng một phần tử kề trước (gọi là phần tử cha), trừ một phần tử đặc biệt gọi là gốc thì không có cha. Mỗi cấu trúc phân cấp như vậy còn được gọi là một cây.



- **Cấu trúc đồ thị:** Đặc điểm của những cấu trúc đồ thị là giữa các phần tử có mối quan hệ nhiều - nhiều, hiểu theo nghĩa: mỗi phần tử có thể nối với nhiều phần tử khác. Cấu trúc đồ thị là kiểu biểu diễn phức tạp nhất.



Nhiều phần tử kế trước

Nhiều phần tử kế sau

- **Cấu trúc tập hợp:** Khác với ba loại cấu trúc đã nêu, trong một tập hợp không có mối quan hệ trực tiếp giữa các phần tử. Tất cả các phần tử của tập hợp đều có chung một mối quan hệ duy nhất: chúng đều là thành viên của của cùng một tập hợp tạo nên bởi chúng.

2.2. Mảng (Array)

2.2.1. Khái niệm

Mảng là một tập có thứ tự gồm một số cố định các phần tử có cùng kiểu. Ngoài giá trị, một phần tử của mảng còn được đặt trưng bởi chỉ số thể hiện thứ tự của phần tử đó trong mảng. Vector là mảng một chiều, mỗi phần tử a_i của nó ứng với một chỉ số i . Ma trận là mảng hai chiều, mỗi phần tử a_{ij} ứng với hai chỉ số i và j . Tương tự người ta mở rộng ra mảng ba chiều, mảng bốn chiều,..., mảng n chiều.

Các phép toán trên mảng bao gồm: phép tạo lập, phép tìm kiếm... Không có phép bổ sung và loại bỏ phần tử được thực hiện đối với mảng.

2.2.2. Cấu trúc lưu trữ của mảng

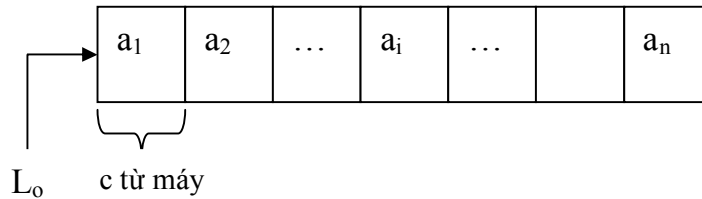
Mảng một chiều dùng *địa chỉ được tính* để thực hiện việc lưu trữ và tìm kiếm phần tử.

Từ máy: Mỗi một địa chỉ mà ứng với nó chứa từ 8 đến 64bit gọi là từ máy. Ta có thể hình dung bộ nhớ trong như một dãy có thứ tự các từ máy (words) mà việc tham khảo đến nội dung của nó được thông qua địa chỉ.

Để xác định địa chỉ của một phần tử trong mảng ta dựa vào những đặc tả của dữ liệu cần tìm. Địa chỉ thuộc loại này được gọi là địa chỉ được tính.

Thông thường một số từ máy kế tiếp sẽ được dành ra để lưu trữ các phần tử của mảng và gọi là lưu trữ kế tiếp. Do số lượng phần tử của mảng là cố định cho nên không gian nhớ cũng được ấn định trước.

Một mảng A có n phần tử, nếu mỗi phần tử a_i ($1 \leq i \leq n$) chiếm c từ máy thì nó sẽ được lưu trữ trong cn từ máy kế tiếp như sau:



Ta có: $a_1 \rightarrow L_o + 0 * c$

$a_2 \rightarrow L_o + 1 * c$

$a_3 \rightarrow L_o + 2 * c$

....

$a_i \rightarrow L_o + (i - 1) * c$

Vậy địa chỉ của phần tử a_i được tính theo công thức:

$$\text{Loc}(a_i) = L_o + (i - 1) * c$$

Trong đó:

- L_o là địa chỉ gốc
- $f(i) = (i - 1) * c$ gọi là hàm địa chỉ

Đối với mảng nhiều chiều việc lưu trữ thông qua một vector lưu trữ kế tiếp. Với mảng hai chiều tùy theo từng loại ngôn ngữ lập trình mà nó được lưu trữ theo thứ tự ưu tiên hàng hay theo thứ tự ưu tiên cột.

Ngôn ngữ lập trình Pascal lưu trữ mảng hai chiều theo thứ tự ưu tiên hàng.

Cho mảng A có m hàng, n cột và mỗi phần tử chiếm một từ máy. Khi đó công thức tính địa chỉ của một phần tử a_{ij} như sau:

$$\text{Loc}(a_{ij}) = L_o + (i - 1) * n + (j - 1)$$

Chú ý: Khi mảng được lưu trữ kế tiếp thì việc truy nhập vào phần tử của mảng được thực hiện trực tiếp dựa vào địa chỉ được tính nên tốc độ nhanh và đồng đều đối với mọi phần tử.

2.3. Danh sách

2.3.1. Danh sách móc nối đơn

Khái niệm và các phép toán

- Danh sách là một tập có thứ tự nhưng bao gồm một số biến động các phần tử. Một danh sách mà quan hệ lân cận giữa các phần tử hiện thị ra thì được gọi là danh sách tuyến tính.

- Mỗi phần tử của danh sách được lưu trữ trong một phần tử nhớ mà ta gọi là nút (node). Mỗi nút bao gồm một số từ máy kế tiếp. Trong mỗi nút, ngoài phần thông tin ứng với mỗi phần tử, còn chứa địa chỉ của phần tử đứng sau nó trong danh sách:

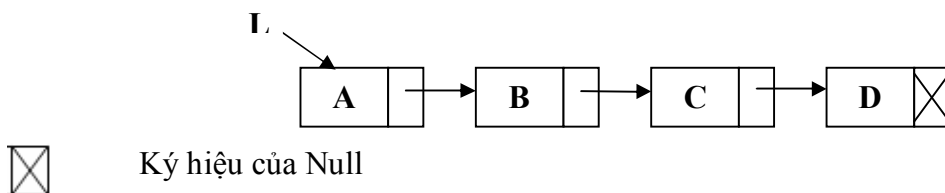
INFO	LINK
------	------

Trường INFO chứa thông tin của phần tử

Trường LINK chứa địa chỉ (mối nối) nút tiếp theo

- Riêng nút cuối cùng thì không có nút đứng sau nên mối nối ở nút này phải là một “địa chỉ đặc biệt” riêng chỉ dùng để đánh dấu nút kết thúc danh sách, ta gọi là “mối nối không” và ký hiệu là Null.

- Để có thể truy nhập đến các phần tử của danh sách phải truy nhập được vào nút đầu tiên, do đó cần sử dụng thêm một con trỏ L trỏ vào nút đầu tiên:



Nếu danh sách rỗng thì $L = \text{Null}$

Khai báo cấu trúc dữ liệu kiểu danh sách:

Typedef struct node

{ **Kieu_DL** Info;

struct node *Link;

}ds;

Một số quy ước:

- Việc cấp phát một nút có địa chỉ P được thực hiện bởi thủ tục New(P); Còn phép thu hồi không dùng nữa thì được thực hiện bởi thủ tục Dispose(P) (hoặc Free(P))

- Khi viết Info(P), Link(P) nghĩa là truy nhập đến trường Info, Link của nút có địa chỉ P.

- Khi thực hiện phép gán $P = \text{Link}(P)$; nghĩa là cho con trỏ P trỏ đến phần tử liền sau phần tử đang trỏ bởi P.

- Khi thực hiện phép gán $P = Q$ nghĩa là con trỏ P trỏ đến phần tử mà Q đang trỏ

Một số phép toán

a. Bổ sung một nút mới vào danh sách

Yêu cầu: Lập giải thuật bổ sung một nút mới có Info=X vào sau nút trỏ bởi M của danh sách nối đơn trỏ bởi L.

void INSERT(L,M,X)

1. *{Tạo nút mới}*

New(P);

Info(P) = X;

2. *{Nếu danh sách rỗng thì bổ sung nút mới vào thành nút đầu tiên, nếu danh sách không rỗng thì nắm lấy M và bổ sung nút mới vào sau nút đó}*

if(L == Null)

{ L = P;

Link(P) = Null;

}

else

{ Link(P) = Link(M);

Link(M) = P;

}

3. **Return**

b. Loại bỏ một nút ra khỏi danh sách

Yêu cầu: Lập giải thuật loại bỏ nút trỏ bởi M ra khỏi danh sách nối đơn trỏ bởi L.

void DELETE (L,M)

1. *{Kiểm tra danh sách rỗng}*

```
    if(L==Null)
    {
        cout("Danh sach rong");
        return;
    }
```

2. *{TH loại bỏ nút đầu tiên}*

```
    if(M == L)
        L = Link(M);
    else
```

3. *{ Tìm nút liền trước nút M}*

```
    {
        P = L;
        while(Link(P) != M)
            P = Link(P);
        Link(P) = Link(M);
    }
```

4. **Free**(M);

5. **Return**

c. Ghép hai danh sách nối đơn

Yêu cầu: Cho hai danh sách nối đơn trở bởi P và Q. Lập giải thuật ghép hai danh sách thành một danh sách mới và cho P trở tới nó

void COMBINE(P,Q)

1. *{TH danh sách trở bởi Q rỗng}*

```
    if (Q==Null)        Return;
```

2. *{TH danh sách trở bởi P rỗng}*

```
    if (P==Null)
    {
        P=Q;
        return;
```

```
}
```

3. *{Tìm đến nút cuối danh sách}*

```
P1= P;
```

```
while(Link(P1) !=Null)
```

```
    P1=Link(P1);
```

4. *{Ghép}*

```
    Link(P1)=Q;
```

5. **Return;**

Cài đặt:

```
#include<conio.h>
#include<stdio.h>
#include<iostream.h>
#include<alloc.h>
typedef struct node
{int info;
  struct node *link;
}ds;
ds *L;

void bosung(int x,int k)    //Bổ sung sau nút thứ k của danh sách
{int d=1;
  ds *p,*m;
  p=new ds(); //p=(ds*)malloc(sizeof(ds));
  p->info=x;   p->link=NULL;
  if(L==NULL)
      L=p;
  else
  {m=L;
    while(d++<k && m->link!=NULL)
        m=m->link;
    p->link=m->link;
```



```

    m->link=p;

}

}

void nhapds()
{int y,n;
  int ch;
  do{
    printf("\nNhap gia tri va vi tri:");
    scanf("%d%d",&y,&n);
    bosung(y,n);
    printf("Co tiep tục tạo ds?(C/K):");
    ch=getch();putchar(ch);
    }while(ch=='c' || ch=='C');
}

void inds()
{ds *p;
  p=L;
  printf("\nDs noi don:");
  while(p!=NULL)
  {printf("%5d",p->info);
    p=p->link;
  }
  printf("\n");
}

int dem()
{int d=0;
  ds*p;
  p=L;
  while(p!=NULL)
  {d++;
    p=p->link;

```

```

    }
    return d;
}

void xoa(int k) //Xóa nút thứ k của danh sách
{if(L==NULL)
    {printf("\nDS rong");
    return;
    }
ds *m,*p;
m=L;
if(k==1)
{L=L->Link;
    free(m);
    return;
    }
while(int i<k && m->Link!=NULL)
    {i++;
    p=m;
    m=m->Link;
    }
p->Link=m->Link;
    free(m);
    }
}

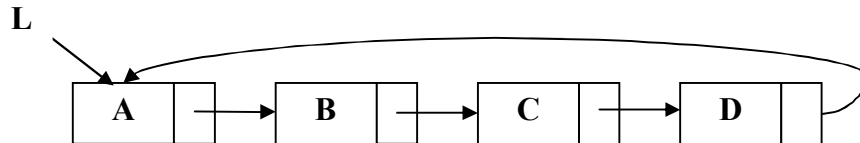
void main()
{ clrscr();
    nhapds();inds();
    dem();
    getch();
}

```

2.3.2. Một số dạng danh sách móc nối khác

a. Danh sách móc nối vòng

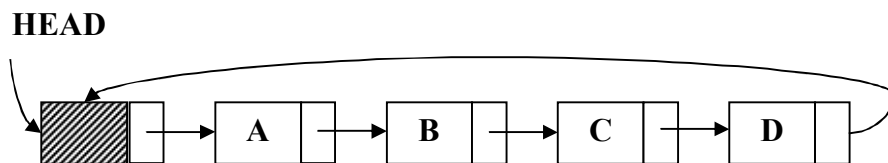
Một cải tiến của danh sách nối đơn là danh sách nối vòng. Nó khác với danh sách nối đơn ở chỗ: mỗi nút ở nút cuối cùng trong danh sách không phải là “mỗi nút không” mà là chứa địa chỉ của nút đầu tiên.



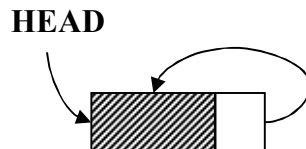
- Ưu điểm: Truy nhập vào các nút trong danh sách linh hoạt hơn. Có thể truy cập vào mọi nút của danh sách bắt đầu từ nút nào cũng được, tức là nút nào cũng có thể coi là nút đầu tiên.

- Nhược điểm: Trong xử lý nếu không cẩn thận sẽ dẫn tới một chu trình không kết thúc vì không biết được chỗ kết thúc của danh sách.

- Để khắc phục nhược điểm đó người ta bổ sung thêm một nút mà trường info của nút này không chứa thông tin gọi là nút đầu danh sách và do con trỏ Head quản lý.



- Việc bổ sung thêm nút đầu danh sách khiến cho danh sách về mặt lưu trữ là không bao giờ rỗng nhưng về mặt xử lý ta thấy danh sách rỗng khi: $\text{Link}(\text{Head}) = \text{Head}$.



- Sau đây là giải thuật bổ sung một nút vào thành nút đầu tiên của một danh sách nối vòng:

$\text{New}(\text{P}); \text{Info}(\text{P}) = \text{X};$

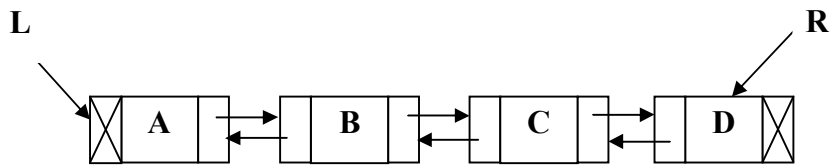
$\text{Link}(\text{P}) = \text{Link}(\text{Head}); \text{Link}(\text{Head}) = \text{P};$

b. Danh sách nối kép

- Đối với danh sách nối đơn ta chỉ có thể duyệt danh sách theo một chiều. Trong nhiều ứng dụng, đôi khi xuất hiện duyệt danh sách theo cả hai chiều. Do đó, mỗi phần tử ngoài phần chứa thông tin còn có hai phần liên kết, một phần chứa địa chỉ của nút liền trước và một phần chứa địa chỉ của nút liền sau gọi là danh sách nối kép.

LPTR	INFO	RPTR
------	------	------

- Trong đó:
 - + Info: Chứa thông tin của nút
 - + LPTR: Con trỏ trái, trỏ tới nút đứng trước.
 - + RPTR: Con trỏ phải, trỏ tới nút đứng sau.
- Như vậy danh sách móc nối sẽ có dạng:



- LPTR của nút cực trái và RPTR của nút cực phải là Null. Để truy nhập danh sách cả hai chiều phải dùng hai con trỏ:

- + Con trỏ L trỏ vào nút cực trái
 - + Con trỏ R trỏ vào nút cực phải.
- Danh sách rỗng khi $L=R=Null$.

Khai báo cấu trúc dữ liệu kiểu danh sách:

Typedef struct node

{

Kieu_DL Info;

struct node *LPTR, *RPTR;

}ds;

Một số phép toán tác động trên danh sách nối kép:

Bổ sung một nút mới vào danh sách:

Yêu cầu: Lập giải thuật bổ sung một nút mới có info=X vào sau nút trỏ bởi M của danh sách nối kép trỏ bởi L và R

void DOUBLE_INSERT(L,R,M,X)

1. *{Tạo nút mới}*

New(P);

Info(P) =X;

2. *{TH danh sách rỗng}*

if (L == R == Null)

{

LPTR(P) =RPTR(P)= Null;

L =R=P;

return;

}

3. *{TH M là nút cuối cùng}*

if(M ==R)

{

RPTR(P) = Null;

LPTR(P)=M;

RPTR(M) =P;

R=P;

return;

}

4. *{Bổ sung vào giữa}*

RPTR(P) =RPTR(M);

LPTR(P) =M;

RPTR(M) =P;

LPTR(RPTR(P))=P;

5. **return;**

Loại một nút ra khỏi danh sách

Yêu cầu: Lập giải thuật loại bỏ nút trở bởi M ra khỏi danh sách nối kép được trở bởi L và R.

```
void DOUBLE_DELETE;
```

1. *{TH ds rỗng}*

```
if(L == R == Null)
```

```
{ printf("Danh sach rong");
```

```
return;
```

```
}
```

2. *{Xét các trường hợp}*

```
if (L == R) {Ds chỉ có một nút}
```

```
L = R = Null;
```

```
else
```

```
if (M == L) {Loại nút đầu tiên}
```

```
{L = RPTR(L);
```

```
LPTR(L) = Null;
```

```
}
```

```
else
```

```
if (M == R) {Loại nút cuối cùng}
```

```
{R = LPTR(R);
```

```
RPTR(R) = Null;
```

```
}
```

```
else {Loại bỏ nút ở giữa}
```

```
{RPTR(LPTR(M)) = RPTR(M);
```

```
LPTR(RPTR(M)) = LPTR(M);
```

```
}
```

3. *{Loại bỏ}*

```
free(M);
```

4.Return;

Cài đặt:

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
#include<alloc.h>
typedef struct nut
{
    int info;
    struct nut *lptr,*rptr;
}ds;
ds *L,*R;
void taods()
{
    ds *p;
    char ch;
    L=NULL; R=NULL;
    do
    {
        p=new ds();
        cout<<"\nNhap du lieu cho nut:";
        cin>>p->info;
        p->lptr=NULL;
        p->rptr=NULL;
        if (L==NULL && R==NULL)
        {
            L=p; R=p;
        }
        else
        {
```

```

        R->rptr=p;
        p->lptr=R;
        R=p;
    }
    cout<<"\nCo tiep tuc tao ds?(C/K):";
    ch=getch();putchar(ch);
}while(ch=='c' || ch=='C');
}

void inds()
{
    ds *p;
    p=L;
    cout<<"\nDS noi kep:" ;
    while (p!=NULL)
        {cout<<p->info<<"    ";
        p=p->rptr;
        }
}

void bosung()
{
    ds *p,*M;
    int d,k;
    p=new ds();
    cout<<"\nNhap du lieu can bo sung:";
    cin>>p->info;
    if (L==NULL && R==NULL)
    {
        L=p; R=p;
        p->lptr=NULL;
        p->rptr=NULL;
    }
}

```



```

else
{
    M=L;    d=1;

    cout<<"Can bo sung sau nut thu may?";

    cin>>k;

    while (d<k && M!=NULL)
    { M=M->rptr;
      d=d+1;
    }

    if (M==R)
    { R->rptr=p;
      p->lptr=R;
      p->rptr=NULL;
      R=p;
    }

    else
    { p->rptr=M->rptr;
      p->lptr=M;
      M->rptr=p;
      p->rptr->lptr=p;
    }
}

}

void main()
{clrscr();

  taods();

  inds();

  bosung();

  inds();

  getch();

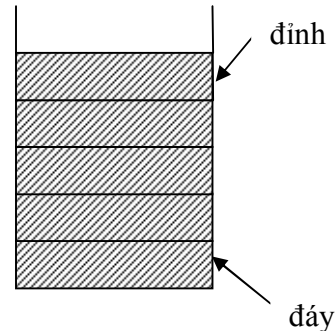
}

```

2.4. Ngăn xếp (Stack)

2.4.1. Khái niệm và các phép toán

- Stack là một kiểu danh sách tuyến tính đặc biệt mà phép bổ sung và loại bỏ luôn luôn được thực hiện ở một đầu gọi là đỉnh (Top).



- Nguyên tắc vào sau ra trước của Stack đưa tới một tên gọi khác là danh sách kiểu LIFO – Last in First out.

- Stack có thể rỗng hoặc bao gồm một số phần tử.

Một số phép toán:

Giải thuật bổ sung

Yêu cầu: Lập giải thuật bổ sung một phần tử X vào Stack.

void PUSH(S,X);

1. **New(T); Info(T) = X;**

2. *{Đẩy vào đỉnh Stack}*

Link(T) = S;

3. *{Ghi nhận lại đỉnh}*

S = T;

4. **Return;**

Giải thuật loại bỏ

Yêu cầu: Lập giải thuật loại bỏ một phần tử ở đỉnh Stack.

Procedure POP(S,X);

1. *{KT Stack rỗng}*

if(S == Null)

{ printf(‘Stack rỗng’);

return;

}

2. **else**

{ X = Info(S);

T=S;

S =link(S);

}

3. *{Loại bỏ}*

free(T);

4. **Return;**

2.4.2. Ví dụ về ứng dụng của Stack

a. Đổi cơ số

Lập giải thuật chuyển đổi số nguyên dương N từ hệ đếm thập phân sang hệ đếm nhị phân.

void CHUYEN_DOI(N);

1. *{Tìm số dư và đẩy vào stack}*

while (N != 0)

{ R = N mod 2;

PUSH(S,R);

N = N div 2;

}

2. *{Lấy và hiện thị số dư từ Stack}*

while(S != Null)

{POP(S,R);

printf(R);

}

3. **Return;**

b. Định giá biểu thức số học theo ký pháp nghịch đảo Ba Lan

- Thông thường trong các biểu thức số học toán tử hai ngôi được đặt giữa hai toán hạng, cặp dấu ngoặc đơn thể hiện sự ưu tiên của toán tử gọi là ký pháp trung tố (hay biểu thức trung tố).

- Nhà bác học Ba Lan J.Lukasiewicz đã đưa ra biểu thức dưới dạng ký pháp tiền tố và ký pháp hậu tố mà dấu ngoặc không còn trong biểu thức gọi là ký pháp nghịch đảo Ba Lan.

- Biểu thức tiền tố là biểu thức tương ứng ký pháp tiền tố mà trong đó toán tử đứng trước toán hạng. Biểu thức hậu tố là biểu thức tương ứng ký pháp hậu tố mà trong đó toán tử đứng sau toán hạng.

Ví dụ:

+ Biểu thức trung tố: $2*(3+4)$

+ Biểu thức tiền tố là: $* 2 + 3 4$

+ Biểu thức hậu tố là: $2 3 4 + *$

Chuyển đổi biểu thức trung tố sang biểu thức hậu tố:

1. Khởi động Stack rỗng để chứa các toán tử.

2. Trong khi chưa xảy ra lỗi và chưa xét hết biểu thức trung tố thì làm:

- Đọc phần tử tiếp theo kể từ trái qua phải trong biểu thức trung tố.

- Nếu phần tử được đọc là :

+ Dấu ngoặc trái thì đẩy vào Stack.

+ Là dấu ngoặc phải thì lấy và hiện thị các phần tử từ Stack cho đến khi dấu ngoặc trái đầu tiên được đọc.

+ Là toán tử:

- Nếu Stack rỗng hoặc toán tử đang xét có độ ưu tiên cao hơn thì đẩy toán tử đang xét vào Stack.
- Nếu toán tử đang xét có độ ưu tiên không cao hơn toán tử ở đỉnh Stack thì lấy và hiện thị các phần tử từ Stack cho đến khi gặp toán tử có độ ưu tiên thấp hơn, sau đó đẩy toán tử đang xét vào Stack.

+ Là toán hạng thì hiện thị toán hạng đó.

3. Sau khi đã xét hết biểu thức trung tố thì lấy và hiện thị các phần tử từ Stack cho đến khi Stack rỗng.

Chú ý:

- Dấu ngoặc trái được xem có độ ưu tiên thấp hơn các toán tử khác.
- Dấu ngoặc trái được lấy ra từ Stack nhưng không được hiện thị trong biểu thức kết quả.

Định giá biểu thức hậu tố:

1. Khởi động Stack rỗng để chứa các toán hạng.
2. Trong khi chưa xét hết biểu thức hậu tố thì làm:
 - Đọc 1 phần tử tiếp theo kể từ trái qua phải trong biểu thức hậu tố.
 - Nếu phần tử được đọc là toán hạng thì đẩy vào Stack. Nếu là toán tử thì thực hiện như sau:

- + Lấy từ đỉnh Stack 2 toán hạng.
- + Tác động toán hạng ra sau lên toán hạng ra trước qua thông qua toán tử.
- + Đẩy kết quả thu được vào Stack.

3. Sau khi đã xét hết biểu thức hậu, giá trị còn lại và duy nhất trong Stack chính là kết quả của biểu thức hậu tố cần định giá.

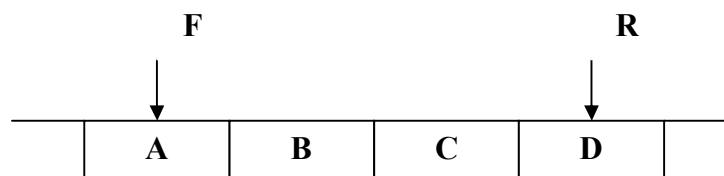
Ví dụ: Cho biểu thức: $(a+b)/m * (c - d*b) + n$

- a. Minh họa quá trình chuyển biểu thức trên về dạng hậu tố.
- b. Minh họa quá trình định giá biểu thức hậu tố thu được với $a = 7, b = 3, m = 2, c = 15, d = 4, n = 1$

2.5. Hàng đợi (Queue)

2.5.1. Khái niệm

- Queue là một kiểu danh sách tuyến tính mà phép bổ sung được thực hiện ở một đầu gọi là lỗi sau còn phép loại bỏ được thực hiện ở một đầu gọi là lỗi trước.



- Nguyên tắc vào trước ra trước của Queue đưa tới một tên gọi khác là danh sách kiểu FIFO – First in First out.

2.5.2. Một số phép toán

a. Giải thuật bổ sung vào hàng đợi

void Q_INSERT(X)

```
1.   New(Q);
      Info(Q) = X;
2.   if(F == Null)
      {F = Q;
       Link(Q) = Null;
       R = F;
      }
      else
      {   Link(R) = Q;
          R = Q;
          Link(Q) = Null;
      }
```

3. **Return**

b. Giải thuật loại bỏ

void Q_DELETE (X);

```
1. if(F == Null) printf(“Queue rỗng”);
2. else
   {   X = info(F);
       Q = F;
       F = Link(F);
       free(Q);
   }
```

3. **Return;**

2.5.3. Ứng dụng của hàng đợi

- Trong các hệ điều hành:
 - + Hàng đợi các công việc hoặc các tiến trình đang đợi để được thực hiện
 - + Hàng đợi các tiến trình chờ các tín hiệu từ các thiết bị IO
 - + Các file được gửi tới máy in
- Mô phỏng các hệ thống hàng đợi thời trong thực tế:
 - + Các khách hàng trong các cửa hàng tạp hóa, trong các hệ thống ngân hàng
 - + Các đơn đặt hàng của một công ty
 - + Các cuộc gọi điện thoại hoặc các đặt hàng vé máy bay, các đặt hàng của khách hàng ...
- Các ứng dụng khác: Thứ tự topo: với một tập các sự kiện, và các cặp (a, b) trong đó sự kiện a có độ ưu tiên cao hơn so với sự kiện b (bài toán lập lịch), duyệt đồ thị theo chiều rộng (Breadth First Search).

Bài tập chương II

2.1. Cho danh sách nối đơn nút đầu tiên được trỏ bởi L. Hãy lập các giải thuật thực hiện:

- + Tìm nút đầu tiên thỏa mãn $\text{info} = X$.
- + Chuyển nút thứ k thành nút cuối cùng của danh sách.

2.2. Cho danh sách nối đơn nút đầu tiên được trỏ bởi L. Giá trị của trường info trong các nút là các số khác nhau và đã được sắp xếp tăng dần. Hãy lập các giải thuật thực hiện:

- + Đếm số nút của danh sách.
- + Bỏ sung một nút mới có $\text{info} = X$ vào danh sách.

2.3. Cho danh sách nối đơn nút đầu tiên được trỏ bởi L. Hãy lập các giải thuật thực hiện:

- + Tính số lượng các nút có $\text{info} = X$.
- + Loại bỏ nút đứng trước nút thứ k của danh sách.

2.4. Cho danh sách nối đơn nút đầu tiên được trỏ bởi P. Hãy lập các giải thuật thực hiện:

- + In ra màn hình trường info của các nút.
- + Có một danh thứ 2, nút đầu tiên trỏ bởi Q. Hãy chèn danh sách Q vào sau nút thứ k trong danh sách trên.

2.5. Cho danh sách nối kép do con trỏ L trỏ vào đầu danh sách và con trỏ R trỏ vào cuối danh sách, trường INFO là các số nguyên. Hãy lập các giải thuật:

- a. In ra trường INFO của tất cả các nút trong danh sách.
- b. Loại bỏ nút mà trường INFO của nó có giá trị bằng K cho trước.

2.6. Cho danh sách nối kép do con trỏ L trỏ vào đầu danh sách và con trỏ R trỏ vào cuối danh sách, trường INFO là các số nguyên. Hãy lập các giải thuật:

- a. Tìm nút thứ k của danh sách, nếu có nút thứ k thì cho biết trường INFO của nút đó, nếu không thì thông báo không tồn tại nút thứ k.
- b. Bỏ sung một nút mới có INFO bằng X vào trước nút thứ k.

2.7. Cho danh sách các mặt hàng, thông tin về một mặt hàng gồm có: Tên hàng, giá.

a) Hãy biểu diễn CTDL của danh sách nói trên dưới dạng một danh sách liên kết đơn.

b) Dựa vào CTDL đã biểu diễn ở trên hãy viết các thủ tục thực hiện các yêu cầu sau:

- Tính tổng giá trị của các mặt hàng có giá từ 5000 trở xuống.
- Bổ sung một mặt hàng mới vào cuối danh sách với tên hàng và giá nhập từ bàn phím.

2.8. Cho danh sách cán bộ, thông tin về một cán bộ gồm có: Họ tên, số năm công tác, giới tính.

a) Hãy biểu diễn CTDL của danh sách nói trên dưới dạng một danh sách liên kết đơn.

b) Dựa vào CTDL đã biểu diễn ở trên hãy viết các thủ tục thực hiện các yêu cầu sau:

- Đếm xem có bao nhiêu cán bộ có giới tính là nữ.
- Loại bỏ những người có từ 30 năm công tác trở lên.

2.9. Minh họa quá trình chuyển biểu thức trung tố sau về biểu thức hậu tố:

$$(a/2+b) * (c - d*e) + f/g$$

2.10. Cho biểu thức hậu tố: $a \ b \ c \ + \ / \ d \ - \ e \ +$

Minh họa quá trình định giá biểu thức hậu tố trên ứng với $a=12$, $b=5$, $c=1$, $d=2$, $e=8$.

2.11. Đọc hiểu chương trình:

```
#include<conio.h>
#include<stdio.h>
#include<iostream.h>
#include<alloc.h>
typedef struct sv
{char ma[5];
char ht[25];
```

```

float dtb;

struct sv *tiep;

}ds;

ds *L;

void taods()
{
    ds *p,*m;
    char ch;
    L=NULL;
    do
    {
        p=new ds();
        cout<<"\nMa sinh vien:";
        gets(p->ma);
        cout<<"\nHo ten:";
        gets(p->ht);
        cout<<"\nDiem TB:";
        cin>>p->dtb;
        p->tiep=NULL;
        if(L==NULL)
        {
            L=p;
            m=L;
        }
    }else
    {
        m->tiep=p;
        m=p;
    }

    cout<<"\nCo tiep tuc tao ds?(C/K):";
    ch=getch();putchar(ch);

```

```

    }while(ch=='c' || ch=='C');
}

void inds()
{
    ds *p;
    p=L;
    cout<<"\nDs noi don:\n";
    while(p!=NULL)
    {
        cout<<p->ma<<"    "<<p->ht<<"    "<<p->dtb<<"\n";
        p=p->tiep;
    }
}

void bosung()
{
    ds *p,*m;
    int k,d;
    cout<<"\nBo sung SV vao danh sach:";
    cout<<"\nNhap vi tri can bo sung:";
    cin>>k;
    p=new ds();
    cout<<"\nMa sinh vien:";
    gets(p->ma);
    cout<<"\nHo ten:";
    gets(p->ht);
    cout<<"\nDiem TB:";
    cin>>p->dtb;
    p->tiep=NULL;
    if(L==NULL)
        L=p;
    else

```

```

{
m=L; d=1;
while(d++<k && m->tiiep!=NULL)m=m->tiiep;
p->tiiep=m->tiiep;
m->tiiep=p;
}
}
void xoa()
{
ds *p,*m;
char t[5];
cout<<"\nNhap ma sinh vien can xoa:";
gets(t);
cout<<"\n"<<t;
p=L; //m=L;
while(strcmp(p->ma,t)==0 && p->tiiep!=NULL)
{
if (p->ma==t)
break;
else
{m=p;
p=p->tiiep;
}
}
cout<<"\n"<<p->ma;
if(p==L)
L=L->tiiep;
else
m->tiiep=p->tiiep;
delete(p);
}

```

```
void main()  
{  
    clrscr();  
    taods();  
    inds();  
    xoa();  
    inds();  
    getch();  
}
```

Chương III. CÂY VÀ CÂY TÌM KIẾM NHỊ PHÂN

A. Mục tiêu

Sau khi hoàn tất chương này sinh viên cần nắm và vận dụng được các kiến thức cơ bản sau:

- Các khái niệm cơ bản về cây.
- Cây nhị phân, các phép duyệt trên cây, cài đặt cây nhị phân.
- Cây nhị phân tìm kiếm và các giải thuật trên cây.

B. Nội dung

3.1. Một số khái niệm

- Một cây là một tập hữu hạn các nút trong đó có một nút đặc biệt được gọi là gốc. Giữa các nút có mối quan hệ phân cấp được gọi là “quan hệ cha con”.

- Ta có thể định nghĩa cây một cách đệ quy như sau:

+ Một nút là một cây và cũng là gốc của cây đó.

+ Nếu n là một nút; T_1, T_2, \dots, T_k là các cây có gốc tương ứng n_1, n_2, \dots, n_k .

Khi đó ta có thể tạo ra cây T bằng cách cho trở thành cha của các nút n_1, n_2, \dots, n_k . n gọi là gốc của cây T ; T_1, T_2, \dots, T_k gọi là các cây con của T ; n_1, n_2, \dots, n_k gọi là các nút con của nút n .

+ Cây rỗng là cây không có nút nào cả.

Các khái niệm:

- Cấp của nút: Số con của một nút gọi là cấp của nút. Các nút có cấp 0 gọi là lá. Các nút không phải là lá gọi là nút nhánh.

- Cấp của cây: Là cấp của nút có cấp cao nhất trên cây

- Mức: Gốc của cây có mức là 1, nếu một nút có mức là i thì các nút con của nó có mức là $i+1$.

- Chiều cao hay chiều sâu của cây là số mức lớn nhất có trên cây đó.

- Đường đi: Nếu n_1, n_2, \dots, n_k là dãy các nút thì nó được gọi là đường đi nếu thỏa mãn n_i là cha của n_{i+1} ($1 \leq i \leq k$). Độ dài của đường đi bằng số các nút trên đường đi đó trừ đi 1.

- Cây có thứ tự: Nếu các cây con của một cây mà thứ tự của nó được coi trọng gọi là cây có thứ tự. Ngược lại gọi là cây không có thứ tự.

- Rừng: Nếu có một tập hữu hạn các cây phân biệt thì tập đó được gọi là rừng. Nếu một cây mà loại bỏ đi nút gốc của nó thì có thể thu được rừng.

3.2. Cây nhị phân

3.2.1. Định nghĩa và tính chất

- Cây nhị phân là một dạng quan trọng của cấu trúc cây. Nó có đặc điểm là mọi nút trên cây chỉ có tối đa 2 nút con. Đối với các cây con của một nút người ta phân biệt cây con trái và cây con phải. Như vậy, cây nhị phân là một cây có thứ tự.

- Một số dạng đặc biệt của cây nhị phân:

- + Cây nhị phân lệch trái.
- + Cây nhị phân lệch phải.
- + Cây ziczăc

- Cây nhị phân hoàn chỉnh là cây mà số nút trên mọi mức, trừ mức cuối cùng đều đạt tối đa.

- Cây nhị phân đầy đủ là cây mà số nút đạt tối đa trên mọi mức. Nó là trường hợp đặc biệt của cây nhị phân hoàn chỉnh.

Trong các cây nhị phân có cùng số lượng nút thì cây nhị phân suy biến có chiều cao lớn nhất, còn cây nhị phân hoàn chỉnh có chiều cao nhỏ nhất.

Bổ đề:

1. Số các nút tối đa ở mức i của cây nhị phân là 2^{i-1} ($i \geq 1$).
2. Số các nút tối đa của cây nhị phân có chiều cao h là $2^h - 1$ ($h \geq 1$)

Chứng minh

1. Sẽ được chứng minh bằng qui nạp

Bước cơ sở: Với $i = 1$, cây nhị phân có tối đa $1 = 2^0$ nút.

Vậy mệnh đề đúng với $i = 1$

Bước qui nạp: Giả sử kết quả đúng với mức i , nghĩa là ở mức này cây nhị phân có tối đa 2^{i-1} nút, ta chứng minh mệnh đề đúng với mức $i + 1$.

Theo định nghĩa cây nhị phân thì tại mỗi nút có tối đa hai cây con nên mỗi nút ở mức i có tối đa hai con. Do đó theo giả thiết qui nạp ta suy ra tại mức $i+1$ ta có:

$$2^{i-1} \times 2 = 2^i \text{ nút.}$$

2. Ta đã biết rằng chiều cao của cây là số mức lớn nhất có trên cây đó. Theo i) ta suy ra số nút tối đa có trên cây nhị phân với chiều cao h là :

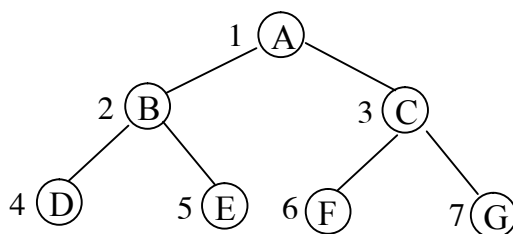
$$2^0 + 2^1 + \dots + 2^{h-1} = 2^h - 1.$$

3.2.2. Biểu diễn cây nhị phân

a. Lưu trữ kế tiếp

Nếu có một cây nhị phân hoàn chỉnh (ưu tiên trái) hoặc đầy đủ ta có thể đánh số thứ tự cho các nút bắt đầu từ 1 kể từ mức 1 trở đi, hết mức này đến mức khác, trên mỗi mức các nút được đánh số kể từ trái qua phải.

Ví dụ:



Khi đó ta có:

- Các nút con của nút i có chỉ số thứ tự là $2i$ và $2i + 1$.
- Nút cha của nút j là nút có chỉ số thứ tự $[j/2]$ ($j \text{ div } 2$).

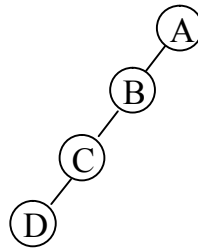
Như vậy với các đánh số ở trên ta có thể lưu trữ cây bằng một vectơ lưu trữ V theo nguyên tắc nút thứ i được lưu trữ ở $V[i]$, kí hiệu: $V: i \rightarrow V[i]$. Cách lưu trữ như vậy gọi là lưu trữ kế tiếp của cây nhị phân. Với cách lưu trữ này ta có thể biết được địa chỉ các nút con của một nút và ngược lại.

Ví dụ: Với cây đã cho ở trên hình ảnh lưu trữ kế tiếp là:

A	B	C	D	E	F	G
$v[1]$	$v[2]$	$v[3]$	$v[4]$	$v[5]$	$v[6]$	$v[7]$

Nếu cây nhị phân không đầy đủ thì cách lưu trữ này không thích hợp vì sẽ gây ra lãng phí bộ nhớ do có nhiều phần tử bỏ trống (ứng với cây con rỗng). Ta hãy xét cây

như hình dưới. Để lưu trữ cây này ta phải dùng mảng gồm 31 phần tử mà chỉ có 5 phần tử khác rỗng; hình ảnh lưu trữ miền nhớ của cây này như sau:



Cây nhị phân đặc biệt

A	B	∅	C	∅	∅	∅	D	∅	∅	∅	∅	∅	∅	∅	E	∅	...
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	-----

(∅: chỉ chỗ trống)

Nếu cây nhị phân luôn biến động nghĩa là có phép bổ sung, loại bỏ các nút thường xuyên tác động thì cách lưu trữ này gặp phải một số nhược điểm như tốn thời gian khi phải thực hiện các thao tác này, độ cao của cây phụ thuộc vào kích thước của mảng...

b. Lưu trữ móc nối

Cách lưu trữ này khắc phục được các nhược điểm của cách lưu trữ trên đồng thời phản ánh được dạng tự nhiên của cây.

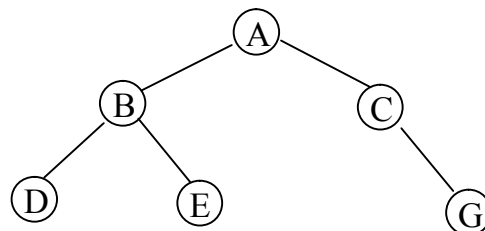
Trong cách lưu trữ này mỗi nút tương ứng với một phần tử nhớ có quy cách như sau:

left	info	right
------	------	-------

Trường info ứng với thông tin (dữ liệu) của nút

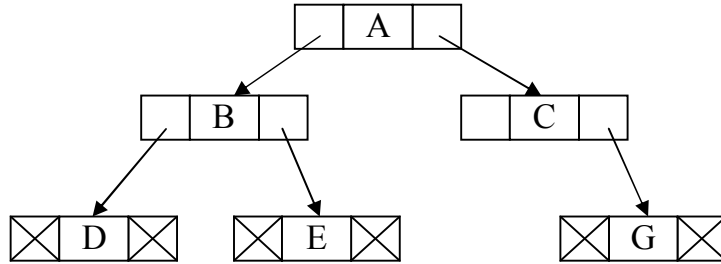
Trường left ứng với con trỏ, trỏ tới cây con trái của nút đó

Trường right ứng với con trỏ, trỏ tới cây con phải của nút



Nếu các con trỏ của các nút không trỏ đến nút con nào cả thì có giá trị Null. Ngoài ra người ta thường sử dụng thêm một biến trỏ T trỏ đến nút gốc của cây và quy ước cây rỗng khi và chỉ khi T = Null.

Ví dụ: Hình ảnh lưu trữ móc nối của cây nhị phân ở trên là:



Với cách lưu trữ móc nối ta dễ dàng truy nhập từ nút cha đến nút con, tuy nhiên điều ngược lại thì khó có thể thực hiện được.

3.2.3. Phép duyệt cây nhị phân

Phép xử lý các nút trên cây gọi chung là phép thăm các nút một cách hệ thống, sao cho mỗi nút được thăm đúng một lần, gọi là phép duyệt cây. Chúng ta thường duyệt cây nhị phân theo một trong ba thứ tự: duyệt trước, duyệt giữa và duyệt sau, các phép này được định nghĩa đệ quy như sau:

Duyệt theo thứ tự trước (preorder traversal)

- Thăm gốc
- Duyệt cây con trái theo thứ tự trước
- Duyệt cây con phải theo thứ tự trước

Duyệt theo thứ tự giữa (inorder traversal)

- Duyệt cây con trái theo thứ tự giữa
- Thăm gốc
- Duyệt cây con phải theo thứ tự giữa

Duyệt theo thứ tự sau (postorder traversal)

- Duyệt cây con trái theo thứ tự sau
- Duyệt cây con phải theo thứ tự sau
- Thăm gốc

Tương ứng với ba phép duyệt ta có ba hàm duyệt cây nhị phân như sau:

```
void Preorder (Tree T);
```

```
1. if (T != Null)
```

```

    {
        printf(info(T));
        Preorder(LPTR(T));
        Preorder(RPTR(T));
    }

```

2. return

void Inorder (Tree T);

1. if(T != Null)

```

    {
        Inorder(LPTR(T));
        printf(info(T));
        Inorder(RPTR(T));
    }

```

2. return

void Postorder (Tree T);

1. if(T != Null)

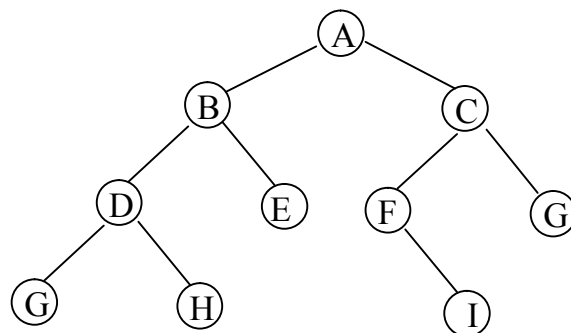
```

    {
        Postorder(LPTR(T));
        Postorder(RPTR(T));
        printf(info(T));
    }

```

2. return

Ví dụ:



Với cây nhị phân ở hình vẽ này, dãy các nút được thăm trong các phép duyệt là:

a) Duyệt theo thứ tự trước:

A B D G H E C F I G

b) Duyệt theo thứ giữa:

G D H B E A F I C G

c) Duyệt theo thứ tự sau:

G H D E B I F G C A

3.2.3. Cài đặt cây nhị phân

Cấu trúc dữ liệu cây cài đặt theo kiểu móc nối:

```
struct Node
```

```
{
```

```
    Kieu_DL info;
```

```
    Node *LPTR;
```

```
    Node *RPTR;
```

```
};
```

```
typedef struct Node *tree;
```

3.3. Cây nhị phân tìm kiếm

Cây nhị phân được sử dụng vào nhiều mục đích khác nhau. Tuy nhiên việc sử dụng cây nhị phân để lưu giữ và tìm kiếm thông tin vẫn là một trong những áp dụng quan trọng nhất của cây nhị phân. Trong phần này chúng ta sẽ nghiên cứu một lớp cây nhị phân đặc biệt phục vụ cho việc tìm kiếm thông tin, đó là cây nhị phân tìm kiếm.

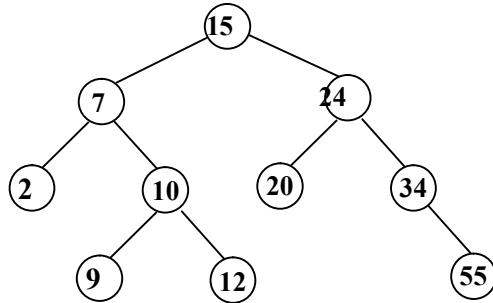
Trong thực tế, một lớp đối tượng nào đó có thể được mô tả bởi một kiểu bản ghi, các trường của bản ghi biểu diễn các thuộc tính của đối tượng. Trong bài toán tìm kiếm thông tin thường quan tâm đến một nhóm các thuộc tính nào đó của đối tượng mà thuộc tính này hoàn toàn xác định được đối tượng. Chúng ta gọi các thuộc tính này là khoá. Như vậy, khoá là một nhóm các thuộc tính của một lớp đối tượng sao cho hai đối tượng khác nhau cần phải có giá trị khác nhau trên nhóm thuộc tính đó.

3.3.1. Định nghĩa

Cây nhị phân tìm kiếm là cây nhị phân thoả mãn đồng thời các điều kiện sau:

- Khoá của các đỉnh thuộc cây con trái nhỏ hơn khoá nút gốc
- Khoá của nút gốc nhỏ hơn khoá của các đỉnh thuộc cây con phải của gốc
- Cây con trái và cây con phải của gốc cũng là cây nhị phân tìm kiếm

Ví dụ biểu diễn một cây nhị phân tìm kiếm, trong đó khoá của các đỉnh là các số nguyên:



3.3.2. Giải thuật tìm kiếm

Tìm kiếm trên cây là một trong các phép toán quan trọng nhất đối với cây nhị phân tìm kiếm. Xét bài toán sau:

Giả sử mỗi đỉnh trên cây nhị phân tìm kiếm là một bản ghi, biến con trỏ T chỉ tới gốc của cây và X là khoá cho trước. Tìm xem trên cây có chứa đỉnh với khoá X hay không.

Giải thuật đệ qui

```
void BST(Tree T; keytype X; tree P);
```

```
{ Nếu tìm thấy thì P chỉ tới nút có trường khoá bằng, ngược lại P=NULL}
```

```
1. P=T;
```

```
2. while (P!=Null)
```

```
    if (X < info(P))
```

```
        BST(LPTR(P), X, P);
```

```
    else
```

```
        if X > info(P)
```

```
            BST(RPTR(P), X, P);
```

```
3. return
```

Giải thuật lặp

Trong giải thuật này sẽ sử dụng biến found có kiểu boolean để điều khiển vòng lặp, nó có giá trị ban đầu là false. Nếu tìm kiếm thành công thì found nhận giá trị true, vòng lặp kết thúc và đồng thời P trở đến nút có trường khoá bằng X. Còn nếu không tìm thấy thì giá trị của found vẫn là false và giá trị của p là nil

```
void BST (Tree T; keytype X; Tree P);
```

```
1. kt =False;P=T;
```

```
2. while (P != Nulll && not kt )
```

```
    if (X > Info(P))
```

```
        P = RPTR(P);
```

```
else
```

```
    if(X < Info(P)
```

```
        P = LPTR(P);
```

```
else        kt = True;
```

```
3.return
```

Duyệt cây nhị phân tìm kiếm

Như ta đã biết cây nhị phân tìm kiếm cũng là cây nhị phân nên các phép duyệt trên cây nhị phân cũng vẫn đúng trên cây nhị phân tìm kiếm. Lưu ý là khi duyệt theo thứ tự giữa thì được dãy khoá theo thứ tự tăng dần.

Chèn một nút vào cây nhị phân tìm kiếm

Việc thêm một nút có trường khoá bằng X vào cây phải đảm bảo điều kiện ràng buộc của cây nhị phân tìm kiếm. Ta có thể thêm vào nhiều chỗ khác nhau trên cây, nhưng nếu thêm vào một nút lá sẽ là tiện lợi nhất do ta có thể thực hiện quá trình tương tự như thao tác tìm kiếm. Khi kết thúc việc tìm kiếm cũng chính là lúc tìm được chỗ cần chèn.

Giải thuật đệ quy

```
void Insert (TreeT; keytypeX);
```

```
1. New(P);
```

```
    Info(P) = X; Q=T;
```

```
2. if(T==Null)
```

```
    {
```

```
        T=P;
```

```

        LPTR(P)= Null;
        RPTR(P)= Null;
    }
3. else
if (X > Info(Q)
        Insert (RPTR(Q), X)

else
        if(X < Info(Q)
            I      Insert (LPTR(Q), X);

4. return

```

Giải thuật lặp

Trong giải thuật này ta sử dụng biến con trỏ Q chạy trên các đỉnh của cây bắt đầu từ gốc. Khi đang ở một đỉnh nào đó, Q sẽ xuống đỉnh con trái (phải) tùy theo khoá ở đỉnh lớn hơn (nhỏ hơn) khoá X.

Tại một đỉnh nào đó khi Q muốn xuống đỉnh con trái (phải) thì phải kiểm tra xem đỉnh này có đỉnh con trái (phải) không. Nếu có thì tiếp tục xuống, ngược lại thì bổ sung đỉnh mới vào bên trái (phải) đỉnh đó. Điều kiện Q = Null sẽ kết thúc vòng lặp. Quá trình này được lặp lại khi có đỉnh mới được chèn vào.

```

void Insert (Tree T; keytype X)
1. New(P);
    Info(P) = X;
2. if(T == Null)
{
    T = P;
    LPTR(P) = Null;
    RPTR(P) = Null;
}
3. else
{
    Q=T;
    while(Q != Null)
        if(X < Info(Q)
            if (LPTR(Q)!= Null)
                Q = LPTR(Q);
        else
            {LPTR(Q)=P;
              Q = Null;
            }

```

```

    }
    else
        if(X > Info(Q))
            if (RPTR(Q) != Null)
                Q = RPTR(Q);
            else
                { RPTR(Q) = P;
                  Q = Null;
                }
    }
    4.          return

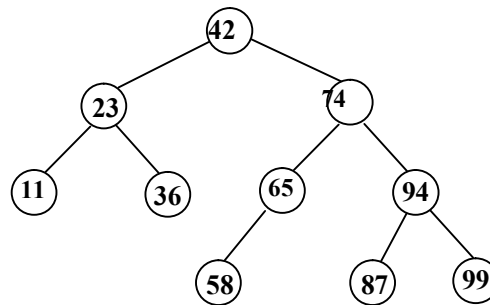
```

Nhận xét:

Để dựng được cây nhị phân tìm kiếm ứng với một dãy khoá đưa vào bằng cách liên tục bỏ các nút ứng với từng khoá, bắt đầu từ cây rỗng. Ban đầu phải dựng lên cây với nút gốc là khoá đầu tiên sau đó đối với các khoá tiếp theo, tìm trên cây không có thì bổ sung vào.

Ví dụ:

Với dãy khoá: 42 23 74 11 65 58 94 36 99 87
thì cây nhị phân tìm kiếm dựng được sẽ có dạng



3.3.3. Phân tích đánh giá

Trong mục này ta sẽ đánh giá thời gian để thực hiện các phép toán trên cây nhị phân tìm kiếm. Ta có nhận xét rằng, thời gian thực các phép tìm kiếm là số phép so sánh giá trị khoá x cho trước với khoá của các đỉnh nằm trên đường đi từ gốc tới đỉnh nào đó trên cây. Do đó thời gian thực hiện các phép tìm kiếm, bổ sung và loại bỏ là độ dài đường đi từ gốc tới một đỉnh nào đó trên cây.

Với giải thuật tìm kiếm nêu trên, ta thấy dạng cây nhị phân tìm kiếm dựng được hoàn toàn phụ thuộc vào dãy khoá đưa vào. Như vậy nghĩa là trong quá trình xử lý

động ta không thể biết trước được cây sẽ phát triển ra sao, hình dạng của nó sẽ như thế nào.

Trong trường hợp nó là một cây nhị phân hoàn chỉnh (ta gọi là cân đối ngay cả khi nó chưa đầy đủ) thì chiều cao của nó là $\lceil \log_2(n+1) \rceil$, nên chi phí tìm kiếm có cấp độ là $O(\log_2 n)$.

Trong trường hợp nó là một cây nhị phân suy biến thành một danh sách liên kết (khi mà mỗi nút chỉ có một con trừ nút lá). Lúc đó các thao tác trên cây sẽ có độ phức tạp là $O(n)$.

Người ta chứng minh được rằng số lượng trung bình các phép so sánh trong tìm kiếm trên cây nhị phân tìm kiếm là:

$$C_{tb} \approx 1,386 \log_2 n$$

Do đó cấp độ lớn của thời gian thực hiện trung bình các phép toán cũng chỉ là $O(\log_2 n)$.

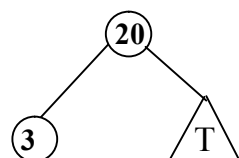
3.3.4. Loại bỏ nút trên cây nhị phân tìm kiếm

Đối lập với phép toán chèn vào là phép toán loại bỏ. Chúng ta cần phải loại bỏ khỏi cây nhị phân tìm kiếm một đỉnh có khoá X (ta gọi tắt là nút X) cho trước, sao cho việc huỷ một nút ra khỏi cây cũng phải bảo đảm điều kiện ràng buộc của cây nhị phân tìm kiếm.

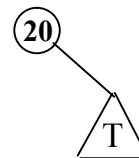
Có ba trường hợp khi huỷ một nút X có thể xảy ra:

- X là nút lá
- X là nút nửa lá (chỉ có một con trái hoặc con phải)
- X có đủ hai con (trường hợp tổng quát)

Trường hợp thứ nhất: chỉ đơn giản huỷ nút X vì nó không liên quan đến phần tử nào khác.

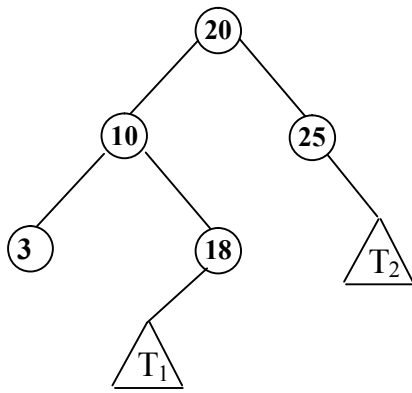


Cây trước khi xoá

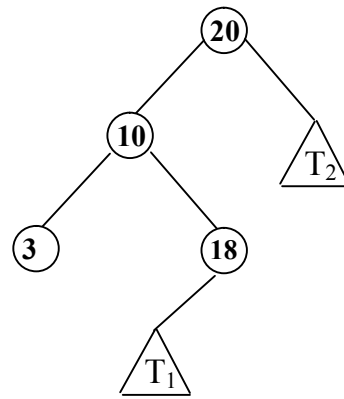


Cây sau khi xoá

Trường hợp thứ hai: Trước khi xoá nút X cần móc nối cha của X với nút con (nút con trái hoặc nút con phải) của nó



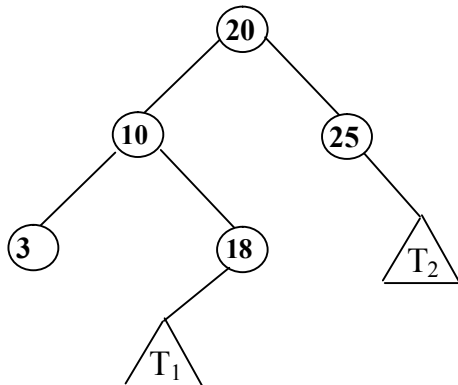
Cây trước khi xoá



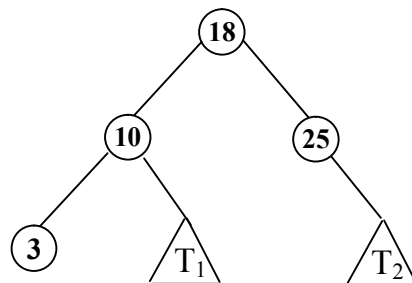
Cây sau khi xoá đỉnh (25)

Trường hợp tổng quát: khi nút bị loại bỏ có cả cây con trái và cây con phải, thì nút thay thế nó hoặc là nút ứng với khoá nhỏ hơn ngay sát trước nó (nút cực phải của cây con trái nó) hoặc nút ứng với khoá lớn hơn ngay sát sau nó (nút cực trái của cây con phải nó). Như vậy ta sẽ phải thay đổi một số mối nối ở các nút:

- Nút cha của nút bị loại bỏ
- Nút được chọn làm nút thay thế
- Nút cha của nút được chọn làm nút thay thế

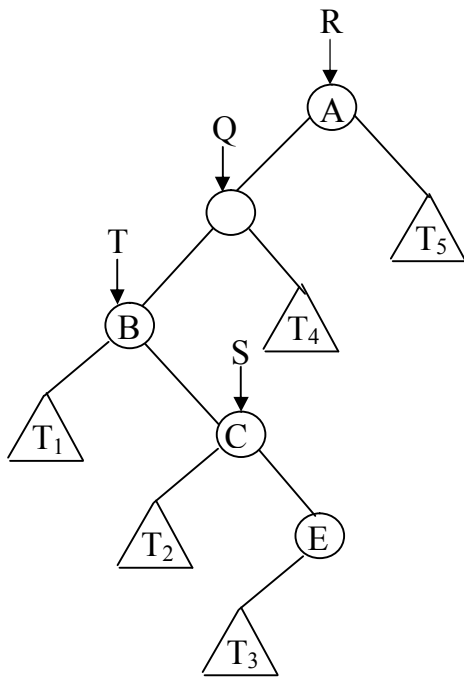


Cây trước khi xoá đỉnh 20

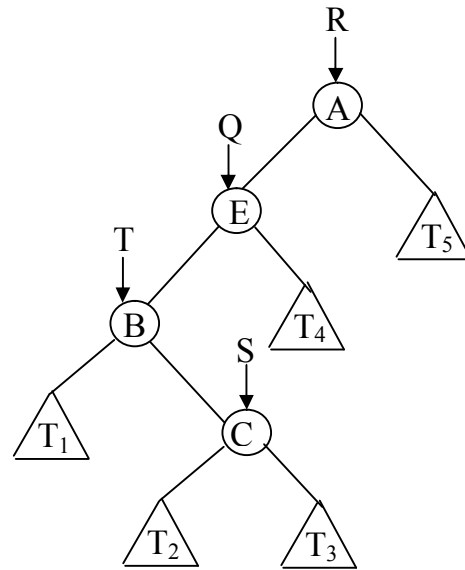


Cây sau khi xoá đỉnh 20

Trong ví dụ này chọn nút thay thế nút bị xoá là nút cực phải của cây con trái. Giải thuật sau thực hiện việc loại bỏ một nút trở bởi Q. Ban đầu Q chính là nối trái hoặc nối phải của một nút R trên cây nhị phân tìm kiếm mà giả sử đã biết rồi.



Cây trước khi xoá nút trở bởi Q



Cây sau khi xoá nút trở bởi Q

void Del (Tree Q); {xoá nút trở bởi Q}

1. **P = Q;** {Xử lý trường hợp nút lá và nút nửa lá}

2. **if**(LPTR(P) = Null)

{

Q=RPTR(P) ;

free(P);

}

2. **else**

if (RPTR(P) = Null)

{Q =LPTR(P);

free (P);

}

3. **else** { Xử lý trường hợp tổng quát}

{ T = LPTR(P);

if (RPTR(T) = Null)

{ RPTR(T) = RPTR(P);

Q = T;

```

        free(P);
    }
    else
    {S = RPTR(T); {Tìm nút thay thế, là nút cực phải của cây }
    while (RPTR(S)!= Null)
    {T = S;
        S = RPTR(T);
    }
    RPTR(S)= RPTR(P)
    RPTR(T) = LPTR(S);
    LPTR(S)= LPTR(P);
    Q =S;
    free(p);
}
}

```

4. return

Giải thuật xoá nút có trường dữ liệu bằng X:

- Tìm đến nút có trường dữ liệu bằng X
- Áp dụng thủ tục Del để xoá

Giải thuật loại khỏi cây gốc T đỉnh có khoá X cho trước. Đó là thủ tục đệ qui, nó tìm ra đỉnh có khoá X, sau đó áp dụng thủ tục Del để loại đỉnh đó ra khỏi cây.

```

void Delete (Tree T ; keytype X);
1.P=T;
2. while(T != Null)
    if(X < Info(P))
        Delete (LPTR(P), X);
    else
        if (X > Info(P))
            Delete (RPTR(P), X);
        else
            Del(P);
3. return

```

Nhận xét: Việc huỷ toàn bộ cây có thể thực hiện thông qua thao tác duyệt cây theo thứ sau. Nghĩa là ta sẽ huỷ cây con trái, cây con phải rồi mới huỷ nút gốc.

```
void      RemoveTree (Tree T);  
  
1. if (T != Null)  
{ RemoveTree(LPTR(T));  
      RemoveTree(RPTR(T));  
      free (T);
```

2. return

Cài đặt cây nhị phân tìm kiếm:

```
#include<conio.h>  
#include<stdio.h>  
#include<iostream.h>  
struct  Node  
{  
    int info;  
    Node  *LPTR;  
    Node  *RPTR;  
};  
typedef struct Node *tree;  
void ktao (tree &T)  
{  
    T=NULL;  
}  
void  insert (tree &T,int x)  
{tree P;  
  P=new Node;  
  P->info=x;  
  if (T==NULL)  
  {  
      T=P;  
      P->LPTR=NULL;  
      P->RPTR=NULL;  
  }
```

```

else
    if (x>T->info)
        insert(T->RPTR,x);
    else
        if (x<T->info)
            insert(T->LPTR,x);
}

void taocay(tree &T)
{
    int y,ch;
    ktao(T);
    do
    {
        cout<<"\nNhap gia tri nut:" ;
        cin>>y;
        insert(T,y);
        cout<<"An phim bat ky de TT - An ESC de thoat";
        ch=getch();
    }while (ch!=27);
}

void duyetttruoc(tree &T)
{
    if (T!=NULL)
    {
        cout<<T->info<<" ";
        duyetttruoc(T->LPTR);
        duyetttruoc(T->RPTR);
    }
}

void duyetgiua(tree &T)
{
    if (T!=NULL)
    {
        duyetgiua(T->LPTR);
        cout<<T->info<<" ";

```

```

        duyetsau(T->RPTR);
    }
}

void duyetsau(tree &T)
{
    if (T!=NULL)
    {
        duyetsau(T->LPTR);
        duyetsau(T->RPTR);
        cout<<T->info<<"    ";
    }
}

int dem(tree &T)
{
    if (T==NULL)
        return 0;
    return 1+dem(T->LPTR)+dem(T->RPTR);
}

int demla(tree &T)
{
    if (T==NULL)
        return 0;
    int d=demla(T->LPTR)+demla(T->RPTR);
    if (T->LPTR==NULL && T->RPTR==NULL)
        d++;
    return d;
}

void main()
{tree H;
  clrscr();
  int chon,n;
  do
  {
      cout<<"\n-----CAY NHI PHAN TIM KIEM-----";
      cout<<"\n1. Tao cay";

```

```

cout<<"\n2. Duyet cay theo thu tu truoc";
cout<<"\n3. Duyet cay theo thu tu giua:";
cout<<"\n4. Duyet cay theo thu tu sau:";
cout<<"\n5. Bo sung them mot nut vao cay";
cout<<"\n6. Dem so nut cua cay";
cout<<"\n7. Dem nut la";
cout<<"\n9. Thoat";
cout<<"\nMoi chon chuc nang:";
cin>>chon;
switch (chon)
{
    case (1): taocay(H);
                break;
    case (2): cout<<"\nCAY NPTK DUYET THU TU TRUOC:";
                duyetttruoc(H);
                cout<<"\n"      ;
                break;
    case (3): cout<<"\nCAY NPTK DUYET THU TU GIUA:";
                duyetgiua(H);
                cout<<"\n";
                break;
    case (4): cout<<"\nCAY NPTK DUYET THU TU SAU:";
                duyetsau(H);
                break;
        case (5): cout<<"\nNhap gia tri nut can bo sung:";
                    cin>>n;
                    insert(H,n);
                    break;
    case (6): cout<<"\nSo nut:"<<dem(H);
                break;
    case (7): cout<<"\nSo nut la:"<<demla(H);
                break;
}
}while(chon!=9);
}

```


3.4. Cây biểu thức

Cây biểu thức là cây nhị phân mà nút gốc và các nút nhánh chứa các toán tử (phép toán) còn các nút lá thì chứa các toán hạng.

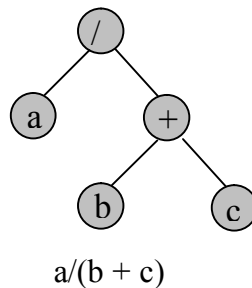
Cách dựng cây biểu thức:

Đối với phép toán hai ngôi (chẳng hạn $+$, $-$, $*$, $/$) được biểu diễn thành cây nhị phân. Quá trình biểu diễn biểu thức T thành cây nhị phân, kí hiệu $B(T)$ được định nghĩa đệ quy như sau:

- Gốc của cây là phép toán thực hiện sau cùng của biểu thức.
- Khi đó biểu thức được chia làm hai phần:
 - + T_1 bên trái.
 - + T_2 bên phải.

Gốc của cây con trái là $B(T_1)$, cây con phải là $B(T_2)$.

Ví dụ:



Đối với phép toán một ngôi như "phủ định" hoặc "lấy giá trị đối", hoặc các hàm chuẩn như $\exp()$ hoặc $\cos()$...thì cây con bên trái rỗng (tức toán hạng tham gia phải được biểu diễn là nút con phải của phép toán đó). Còn với các phép toán một toán hạng như phép "lấy đạo hàm" $()'$ hoặc hàm "giai thừa" $()!$ thì cây con bên phải rỗng.

Nhận xét

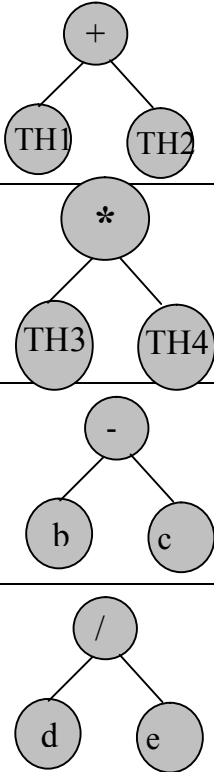
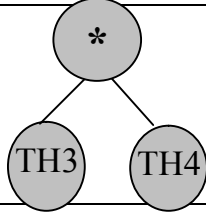
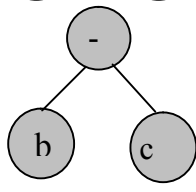
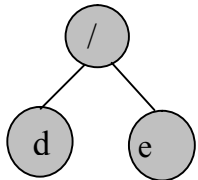
Nếu ta duyệt cây biểu thức theo thứ tự trước thì ta được biểu thức Balan dạng tiền tố (prefix). Nếu duyệt cây nhị phân theo thứ tự sau thì ta có biểu thức Balan dạng hậu tố (postfix); còn theo thứ giữa thì ta nhận được cách viết thông thường của biểu thức (dạng trung tố).

Ví dụ:

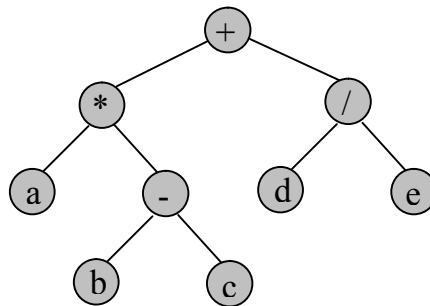
Cho biểu thức $P = a*(b - c) + d/e$

a) Hãy dựng cây biểu thức biểu diễn biểu thức trên

b) Đưa ra biểu thức ở dạng tiền tố và hậu tố

Phân tích	Cây biểu thức
$TH1 = a * (b - c)$ $TH2 = d / e$	
$TH1 = TH3 * TH4$ $TH3 = a$	
$TH4 = b - c$	
$TH2 = d / e$	

Tổng hợp cây biểu thức của các toán hạng ta được cây biểu thức sau:



Duyệt theo thứ tự trước : $+ * a - b c / d e$

Duyệt theo thứ sau: $a b c - * d e / +$

3.5. Cây tổng quát

3.5.1. Biểu diễn cây tổng quát

- Đối với cây tổng quát cấp m nào đó có thể sử dụng cách biểu diễn móc nối tương tự như đối với cây nhị phân. Như vậy ứng với mỗi nút ta phải dành ra m trường mỗi

nổi để trở tới các con của nút đó và như vậy số mỗi nổi không sẽ rất nhiều: nếu cây có n nút sẽ có tới $n(m-1) + 1$ "mỗi nổi không" trong số $m.n$ mỗi nổi.

- Nếu tùy theo số con của từng nút mà định ra mỗi nổi nghĩa là dùng nút có kích thước biến đổi thì sự tiết kiệm không gian nhớ này sẽ phải trả giá bằng những phức tạp của quá trình xử lý trên cây.

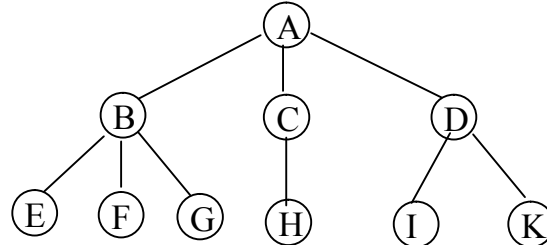
- Để khắc phục các nhược điểm trên là dùng cách biểu diễn cây nhị phân để biểu diễn cây tổng quát.

Ta có thể biến đổi một cây bất kỳ thành một cây nhị phân theo qui tắc sau:

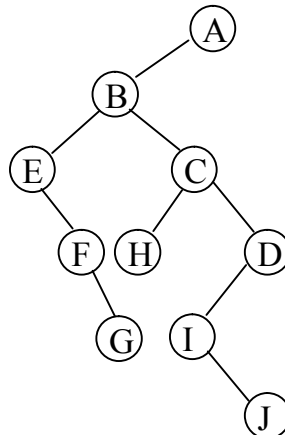
- + Nút gốc của cây tổng quát trở thành nút gốc của cây nhị phân tương đương.
- + Nút con cực trái của một nút trở thành nút con trái của nút đó.
- + Nút em kế cận phải của một nút trở thành nút con phải của nút đó.

Như vậy, trong cây nhị phân mới, con trái thể hiện quan hệ cha con và con phải thể hiện quan hệ anh em trong cây tổng quát ban đầu. Khi đó cây nhị phân này được gọi là cây nhị phân tương đương.

Ví dụ: Giả sử có cây tổng quát như hình vẽ dưới đây:



Cây nhị phân tương đương sẽ như sau



3.5.2. Phép duyệt cây tổng quát

Phép duyệt cây tổng quát cũng được đặt ra tương tự như đối với cây nhị phân. Tuy nhiên có một điều cần phải xem xét thêm, khi định nghĩa phép duyệt, đó là:

- Sự nhất quán về thứ tự các nút được thăm giữa phép duyệt cây tổng quát và phép duyệt cây nhị phân tương đương của nó
- Sự nhất quán giữa định nghĩa phép duyệt cây tổng quát với định nghĩa phép duyệt cây nhị phân. Vì cây nhị phân cũng có thể coi là cây tổng quát và ta có thể áp dụng định nghĩa phép duyệt cây tổng quát cho cây nhị phân.

Ta có thể xây dựng được định nghĩa của phép duyệt cây tổng quát T như sau

Duyệt theo thứ tự trước

a) Nếu T rỗng thì không làm gì.

b) Nếu T khác rỗng thì:

- Thăm gốc của T
- Duyệt các cây con thứ nhất T_1 của gốc của cây T theo thứ tự trước
- Duyệt các cây con còn lại T_2, T_3, \dots, T_n của gốc T theo thứ tự trước

Duyệt theo thứ tự giữa

a) Nếu T rỗng thì không làm gì.

b) Nếu T khác rỗng thì:

- Duyệt các cây con thứ nhất T_1 của gốc của cây T theo thứ tự giữa
- Thăm gốc của T
- Duyệt các cây con còn lại T_2, T_3, \dots, T_n của gốc T theo thứ tự giữa

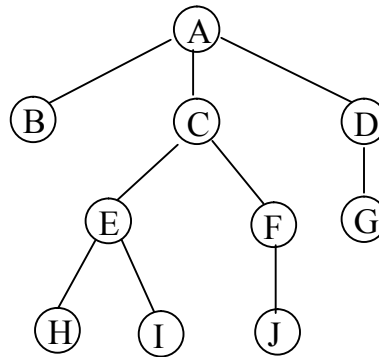
Duyệt theo thứ tự sau

a) Nếu T rỗng thì không làm gì.

b) Nếu T khác rỗng thì:

- Duyệt các cây con thứ nhất T_1 của gốc của cây T theo thứ tự sau
- Duyệt các cây con còn lại T_2, T_3, \dots, T_n của gốc T theo thứ tự sau
- Thăm gốc của T

Ví dụ:



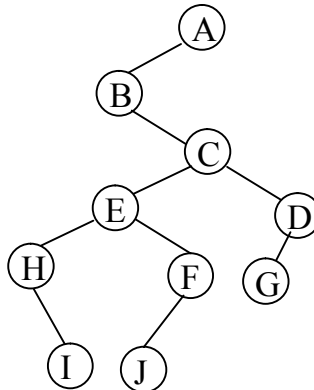
Thứ tự các nút được thăm:

Thứ tự trước: A B C E H I F J D G

Thứ tự giữa : B A H E I C J F G D

Thứ tự sau : B H I E J F C G D A

Cây nhị phân tương đương với cây tổng quát ở hình trên:



Thứ tự các nút được thăm khi duyệt trên cây nhị phân tương đương:

Thứ tự trước: A B C E H I F J D G

Thứ tự giữa: B H I E J F C G D A

Thứ tự sau: I H J F E G D C B A

Nhận xét

Với thứ tự trước phép duyệt cây tổng quát và phép duyệt cây nhị phân tương đương của nó đều cho một dãy tên như nhau. Phép duyệt cây tổng quát theo thứ tự sau cho dãy tên giống như dãy tên các nút được duyệt theo thứ tự giữa trong phép duyệt cây nhị phân. Còn phép duyệt cây tổng quát theo thứ tự giữa thì cho dãy tên không giống bất kỳ dãy nào đối với cây nhị phân tương đương. Do đó đối với cây tổng quát, nếu định nghĩa phép duyệt như trên người ta thường chỉ nêu hai phép duyệt theo thứ tự trước và phép duyệt theo thứ tự sau

Bài tập chương III

3.1. Dựng cây nhị phân biết thứ tự các đỉnh khi duyệt theo

a) Thứ tự trước: A D F G H K L P Q R W Z

Thứ tự giữa : G F H K D L A W R Q P Z

b) Theo thứ tự sau: F G H D A L P Q R Z W K

Thứ tự giữa : G F H K D L A W R Q P Z

3.2. Với mỗi biểu thức số học dưới đây, hãy vẽ cây nhị phân biểu diễn biểu thức ấy, rồi dùng các kiểu duyệt để tìm biểu thức tiền tố và hậu tố tương đương.

a. $a / (b - (c - (d - (e - f))))$

b. $((a * (b + c)) / (d - (e + f))) * (g / (h / (i * j)))$

3.3. Hãy trình bày các vấn đề sau

a. Định nghĩa và đặc điểm của cây nhị phân tìm kiếm

b. Thao tác thực hiện tốt nhất trong kiểu này

c. Hạn chế của kiểu này là gì?

3.4. Xét giải thuật tạo cây nhị phân tìm kiếm. Nếu thứ tự các khoá nhập vào là như sau:

8 3 5 2 20 11 30 9 18 4

thì hình ảnh cây được tạo như thế nào?

Sau đó, nếu huỷ lần lượt các nút theo thứ tự như sau:

11 20 8

thì cây sẽ thay đổi như thế nào trong từng bước huỷ, vẽ cây minh hoạ qua từng bước.

3.5. a. Nêu khái niệm cây nhị phân và phép duyệt sau.

b. Vẽ cây biểu diễn biểu thức sau đây và viết chúng dưới dạng tiền tố, hậu tố:

$(a \uparrow 2 + b) * (c + d * \theta e)$

c. Cho dãy khoá: 34, 66, 17, 25, 71, 75, 50, 68

- Biểu diễn dãy khoá trên thành cây nhị phân tìm kiếm.

- Nếu loại bỏ các nút: 68, 71, 34 thì hình ảnh cây nhị phân tìm kiếm như thế nào?

3.6. Cài đặt cây nhị phân:

- Cài đặt ba phép duyệt trên cây
- Đếm số nút của cây
- Đếm số nút lá.
- Tính chiều cao của cây.

3.7. Cài đặt cây nhị phân tìm kiếm:

- Đếm số nút của cây
- Đếm số nút lá.
- Tính chiều cao của cây.
- Tìm giá trị X có trên cây không, nếu không có bổ sung vào
- Xóa nút trên cây.

Chương IV. SẮP XẾP VÀ TÌM KIẾM

A. Mục tiêu

Sau khi hoàn tất chương này sinh viên cần hiểu và vận dụng được các kiến thức cơ bản sau:

- Các giải thuật sắp xếp cơ bản, sắp xếp nhanh, sắp xếp vun đống, hòa nhập. Mỗi giải thuật cần nắm: ý tưởng, giải thuật và minh họa.
- Tìm kiếm tuần tự và tìm kiếm nhị phân

B. Nội dung

4.1. Bài toán sắp xếp

- Sắp xếp là quá trình bố trí lại các phần tử của một tập đối tượng nào đó theo một thứ tự ấn định.

- Bài toán sắp xếp được đặt ra ở đây là sắp xếp đối với một bảng gồm n bản ghi R_1, R_2, \dots, R_n . Tuy nhiên không phải toàn bộ các trường dữ liệu của các bản ghi đều được xét đến trong quá trình sắp xếp mà chỉ một hay một vài trường nào đó được chú ý đến mà thôi. Trường như vậy được gọi là khoá sắp xếp và quá trình sắp xếp được tiến hành dựa trên khoá này.

Khoá đóng vai trò đặc biệt trong quá trình sắp xếp nên khi trình bày các phương pháp, giải thuật hay ví dụ minh họa để cho đơn giản ta chỉ nói tới các giá trị khoá. Bài toán sắp xếp được coi như sắp xếp n khoá K_1, K_2, \dots, K_n tương ứng với các bản ghi R_1, R_2, \dots, R_n và $K_i \neq K_j$ với mọi $i \neq j$.

Để minh họa cho các phương pháp ta giả sử thứ tự sắp xếp là tăng dần và sử dụng dãy khoá sau làm ví dụ:

42 23 74 11 65 58 94 36 99 87

4.2. Một số phương pháp sắp xếp đơn giản

4.2.1. Sắp xếp kiểu lựa chọn (Slection Sort)

Nội dung phương pháp: Ở lượt thứ i ($i = 1, 2, \dots, n - 1$) ta chọn khoá nhỏ nhất trong dãy khoá K_i, K_{i+1}, \dots, K_n và đổi chỗ nó với K_i .

Giải thuật:


```
void Selection_Sort(K, n);
```

```
1. {Xét các lượt}
```

```
for (i = 1; i < n; i++)
```

```
{
```

```
2. {Tìm khoá nhỏ nhất}
```

```
    m = i;
```

```
    for (j = i + 1; j <= n; j++)
```

```
        if (K[j] < K[m])
```

```
            m = j;
```

```
3. {Đổi chỗ}
```

```
    if (m != i)
```

```
        K[m] ↔ K[i];
```

```
}
```

```
4. return.
```

Minh họa:

K _i	42	23	74	11	65	58	94	36	99	87
Lượt 1	<u>11</u>	23	74	<u>42</u>	65	58	94	36	99	87
Lượt 2	11	<u>23</u>	74	42	65	58	94	36	99	87
Lượt 3	11	23	<u>36</u>	42	65	58	94	<u>74</u>	99	87
Lượt 4	11	23	36	<u>42</u>	65	58	94	74	99	87
Lượt 5	11	23	36	42	<u>58</u>	<u>65</u>	94	74	99	87
Lượt 6	11	23	36	42	58	<u>65</u>	94	74	99	87
Lượt 7	11	23	36	42	58	65	<u>74</u>	<u>94</u>	99	87
Lượt 8	11	23	36	42	58	65	74	<u>87</u>	99	<u>94</u>
Lượt 9	11	23	36	42	58	65	74	87	<u>94</u>	<u>99</u>

4.2.2. Sắp xếp kiểu thêm dần (Insert Sort)

Nội dung phương pháp:

- Đầu tiên khoá K_1 được coi như bản ghi gồm 1 khoá đã được sắp xếp.
- Xét khoá K_2 , so sánh K_2 với K_1 để xác định chỗ chèn ta thu được bảng gồm hai khoá đã được sắp xếp.
- Xét khoá K_3 , so sánh với K_2, K_1 để xác định chỗ chèn ta thu được bảng gồm ba khoá đã sắp xếp.

Quá trình được tiếp tục tương tự với các khoá K_4, \dots, K_n .

Giải thuật:

void Insertion_Sort(K, n);

1. *{Xét các khoá}*

for ($i = 2; i \leq n; i++$)

{ $X = K[i];$

$j = i - 1;$

while ($j > 0 \ \&\& \ X < K[j]$)

{ $K[j+1] = K[j];$

$j = j - 1; \}$

$K[j+1] = X; \}$

4. return.

Minh họa:

Lượt	1	2	3	4	5	6	7	8	9	10
Khóa đưa vào	42	23	74	11	65	58	94	36	99	87
	42	23	23	11	11	11	11	11	11	11
	-	42	42	23	23	23	23	23	23	23
	-	-	74	42	42	42	42	36	36	36
	-	-	-	74	65	58	58	42	42	42
	-	-	-	-	74	65	65	58	58	58
	-	-	-	-	-	74	74	65	65	65
	-	-	-	-	-	-	94	74	74	74
	-	-	-	-	-	-	-	94	94	87
	-	-	-	-	-	-	-	-	99	94
	-	-	-	-	-	-	-	-	-	99

Cài đặt:

```
#include<conio.h>
#include<iostream.h>
void nhap(int k[],int n)
{
    for(int i=0; i<n; i++)
    {
        cout<<"a["<<i<<"]=";
        cin>>k[i];
    }
}
void indl(int k[],int n)
{
    cout<<"\nDay so:";
```

```

    for (int i=0; i<n; i++)
        cout<<k[i]<<" ";
    cout<<"\n";
}

void sx(int k[], int n)
{
    for (int i=0; i<n-1; i++)
    {
        int m=i;
        for (int j=i+1; j<n; j++)
            if (k[j]<k[m])
                m=j;
        if (m!=i)
        {
            int tg=k[i];
            k[i]=k[m];
            k[m]=tg;
        }
    }
}

void is(int k[], int n)
{
    int i,j,x;
    for (i=1;i<n;i++)
    {
        x=k[i];
        j=i-1;
        while ((j>=0) && (x<k[j]))
            {k[j+1]=k[j];
            j=j-1;
            }
        k[j+1]=x;
    }
}

```

```

void main()
{
    clrscr();
    int a[20]; //= {42, 23, 11, 74, 65, 58, 94, 36, 99, 87};
    int m, chon; //= 10
    do
    {
        cout<<"\n-----Menue-----";
        cout<<"\n1. Nhap day so";
        cout<<"\n2. In day so";
        cout<<"\n3. Sap xep kieu lua chon";
        cout<<"\n4. Sap xep kieu them dan";
        cout<<"\n9. Thoat";
        cout<<"\n Moi chon chuc nang:";
        cin>>chon;
        switch (chon)
        {
            case (1):
                cout<<"\nNhap so ptu cua day:";
                cin>>m;
                nhap(a,m);
                break;
            case (2):
                indl(a,m);
                break;
            case (3):
                sx(a,m);
                break;
            case (4):
                is(a,m);
                break;
            case (5):

```

```

        break;
    }
    getch();
}while (chon!=9);
}

```

4.2.3. Sắp xếp kiểu đổi chỗ (Exchange Sort)

Nội dung phương pháp: Bảng các khoá được duyệt từ đáy lên đến đỉnh, dọc đường nếu gặp hai khoá kề cận ngược thứ tự thì đổi chỗ chúng cho nhau. Như vậy ở lượt đầu tiên khoá nhỏ nhất sẽ được chuyển dần lên đỉnh. Quá trình thực hiện tương tự cho đến khi dãy khoá được sắp xếp xong.

Giải thuật:

Procedure Exchange_Sort(K, n);

1. *{Xét các lượt và đổi chỗ}*

for (i = 1; i < n; i++)

for (j = n; j > i; j--)

if (K[j] < K[j-1])

 K[j] ↔ K[j-1];

2. **return.**

4.3. Sắp xếp nhanh

4.3.1. Giới thiệu phương pháp

- Sắp xếp kiểu phân đoạn là một cải tiến của phương pháp sắp xếp kiểu đổi chỗ. Nó dựa trên nguyên tắc ưu tiên đổi chỗ trên các khoảng cách lớn để đạt được hiệu quả cao nhất.

- Ý chủ đạo của phương pháp là chọn một khoá ngẫu nhiên trong dãy khoá làm khoá chốt. Mọi khoá có giá trị nhỏ hơn khoá chốt được xếp vào vị trước khoá chốt, mọi khoá có giá trị lớn hơn khoá chốt được xếp vào vị trí sau khoá chốt. Muốn vậy các khoá trong dãy khoá phải so sánh với khoá chốt và đổi chỗ cho nhau hoặc cho khoá chốt. Khi việc đổi chỗ được thực hiện xong thì khoá chốt sẽ nằm đúng vị trí của nó và dãy khoá được chia thành hai phân đoạn. Một phân đoạn gồm các khoá nhỏ hơn khoá

chốt nằm ở vị trí trước khoá chốt và một đoạn gồm các khoá lớn hơn khoá chốt và nằm sau khoá chốt.

- Ở các lượt tiếp theo cũng áp dụng kỹ thuật tương tự đối với các phân đoạn cho đến khi dãy khoá được sắp xếp xong.

4.3.2. Ví dụ và giải thuật

Xét dãy khoá:

42 23 74 11 65 58 94 36 99 87

- Ở đây ta chọn khoá 42 làm chốt. Sử dụng hai biến chỉ số i và j với giá trị ban đầu $i=2, j=10$ để duyệt. Quá trình so sánh và đổi chỗ được thực hiện như sau:

- So sánh K_i với 42, nếu $K_i < 42$ thì tăng i lên 1 đơn vị, quá trình được lặp cho đến khi $K_j > 42$.

- So sánh K_j với 42, nếu $K_j > 42$ thì giảm j xuống 1 đơn vị, quá trình được lặp cho đến khi $K_j < 42$.

- Nếu $i < j$ thì hai khoá K_i và K_j được đổi chỗ cho nhau. Nếu $i > j$ thì khoá 42 được đặt vào đúng vị trí của nó

void QSORT(l,r);

1. *{Khởi tạo}*

$i = l; j = r;$

$key = K[(l+r) \div 2];$

2. *{Lặp các bước so sánh và đổi chỗ}*

while($i \leq j$)

{

while($K[i] < key$)

$i = i+1;$

while ($K[j] > key$)

$j = j-1;$

if ($i < j$)

$\{K[i] \leftrightarrow K[j]\}$

```

        i = i+1;
        j = j-1;
    }
}

```

3. $\text{Key} \leftrightarrow K[j]$

4. *{Thực hiện đối với các phân đoạn}*

```

    if (l < j)
        QSORT(l,j); (j-1)

    if (i < r)
        QSORT(i,r);

```

5. return;

Cài đặt:

```

#include<conio.h>
#include<iostream.h>
int n;
void nhap(int k[])
{
    for(int i=0; i<n; i++)
    {
        cout<<"a["<<i<<"]=";
        cin>>k[i];
    }
}
void indl(int k[])
{
    cout<<"\nDay so:";
    for (int i=0; i<n; i++)
        cout<<k[i]<<" ";
    cout<<"\n";
}
void nhanh(int k[],int l,int r)
{

```



```

int i=l,j=r;
int key=k[(l+r) % 2];
while(i<=j)
{
    while (k[i]<key)
        i++;
    while (k[j]>key)
        j--;
    if(i<=j)
    { int tg=k[i];
      k[i]=k[j];
      k[j]=tg;
      i++;
      j--;
    }
}
if (l<j)
    nhanh(k,l,j);
if (i<r)
    nhanh(k,i,r);
}

void main()
{
    clrscr();
    int a[20];
    cout<<"\nNhap so ptu cua day can sx:";
    cin>>n;
    nhap(a);
    indl(a);
    nhanh(a,1,n);
    indl(a);
    getch;
}

```

4.3.3. Nhận xét và đánh giá

- Khoá chốt có thể chọn tùy ý trong dãy khoá nhưng khi thể hiện giải thuật ta phải định ra một cách chọn cụ thể. Một phương pháp được đưa ra để chọn khoá chốt là chọn trung vị của 3 khoá $K[l]$, $K[(l+r) \div 2]$, $K[r]$; trong đó l và r là chỉ số của khoá đầu và khoá cuối của dãy đã cho.

- Người ta đã chứng minh được rằng:

+ TH xấu nhất xảy ra khi dãy khoá đã có thứ tự sắp xếp và khi đó $T(n) = O(n^2)$;

+ TH tốt nhất khi dãy khoá luôn luôn được chia đôi và khi đó $T(n) = O(n \cdot \log_2 n)$;

+ TH trung bình thì $T(n) = O(n \cdot \log_2 n)$;

4.4. Sắp xếp kiểu vun đống (Heap Sort)

Với phương pháp sắp xếp kiểu phân đoạn cho ta thời gian thực hiện trung bình khá tốt, tuy nhiên trong trường hợp xấu nhất vẫn là $O(n^2)$. Sau đây ta sẽ xét tới phương pháp mà độ phức tạp trong mọi trường hợp đều là $O(n \cdot \log_2 n)$. Bảng các khoá được tổ chức dưới dạng cây nhị phân hoàn chỉnh và lưu trữ kế tiếp trong máy.

4.4.1. Giới thiệu phương pháp

Sắp xếp kiểu vun đống được chia thành hai giai đoạn như sau:

- Đầu tiên cây nhị phân biểu diễn bảng khoá được biến đổi để trở thành một đống, gọi là giai đoạn tạo đống. Đống là một cây nhị phân hoàn chỉnh mà mỗi nút của nó được gán một giá trị khoá sao cho khoá ở nút cha bao giờ cũng lớn hơn khoá ở các nút con của nó.

Như vậy khoá ở nút gốc của đống là khoá có giá trị lớn nhất so với mọi khoá trên cây và ta gọi là khoá trội.

- Giai đoạn thứ hai là giai đoạn sắp xếp, ở đây nhiều lượt xử lý được thực hiện, mỗi lượt ứng với các phép sau:

+ Đưa khoá trội về đúng vị trí của nó bằng cách đổi chỗ với khoá sau cùng.

+ Loại bỏ khoá trội ra khỏi cây.

+ Vun lại thành đống với cây gồm các khoá còn lại.

- Vd: Cho dãy khoá sau: 42 23 74 11 65 58

- a. Biểu diễn dãy khoá thành cây nhị phân hoàn chỉnh.
- b. Vẽ hình ảnh của đồng qua các bước sắp xếp kiểu vun đồng.

4.4.2. Giải thuật

Để đi tới giải thuật sắp xếp trước hết ta phải xây dựng giải thuật vun đồng. Việc vun đồng được tiến hành từ đáy lên.

Chú ý:

- Nếu một cây nhị phân hoàn chỉnh đã là đồng thì các cây con của các nút (nếu có) cũng là cây nhị phân hoàn chỉnh và cũng là đồng.
- Trên cây nhị phân có n nút thì chỉ có $\lfloor n/2 \rfloor$ nút được là “cha” thôi.
- Một nút lá bao giờ cũng có thể coi là đồng.

Giải thuật vun đồng sẽ thực hiện chỉnh lý một cây nhị phân hoàn chỉnh có gốc được đánh số thứ tự là i , và gốc đã có 2 cây con đã là đồng rồi.

```
void      Vun_dong(i,n);
```

```
1. {Khởi tạo}
```

```
    j = 2*i;
```

```
2. {Thực hiện vun đồng}
```

```
    while(j <= n)
```

```
    {      {Tìm nút con có khoá lớn nhất}
```

```
        if (j < n && K[j] < K[j+1])
```

```
            j = j+1;
```

```
            {So sánh với nút cha}
```

```
        if(K[i] > K[j])
```

```
            return;
```

```
        else  K[i] ↔ K[j];
```

```
        i = j;
```

```
        j = 2*i;
```

```
    end;
```

```
3. return;
```

Giải thuật sắp xếp kiểu vun đống được viết như sau:

```
void      Heap_Sort(K,n);
```

1. *{Vun đống lần đầu}*

```
    for(i = n div 2; i > 0; i--)
```

```
        Vun_dong(i,n);
```

2. *{Sắp xếp}*

```
    for(i = n; i > 1; i--)
```

```
        { K[1] ↔ K[i]; {Khoá lớn nhất chuyển ra cuối dãy}
```

```
        Vun_dong(1,i-1); {Điều chỉnh cây còn lại thành đống}
```

```
    }
```

3. **return;**

Cài đặt:

```
#include<conio.h>
```

```
#include<iostream.h>
```

```
void nhap(int k[],int n)
```

```
{
```

```
    for(int i=0; i<n; i++)
```

```
    {
```

```
        cout<<"a["<<i<<"]=";
```

```
        cin>>k[i];
```

```
    }
```

```
}
```

```
void indl(int k[],int n)
```

```
{
```

```
    cout<<"\nDay so:";
```

```
    for (int i=0; i<n; i++)
```

```
        cout<<k[i]<<"  ";
```

```
    cout<<"\n";
```

```
}
```

```
void dc(int &x,int &y)
```

```

{
    int tam=x;
        x=y;
        y=tam;
}

void sx(int k[], int n)
{
    for (int i=0; i<n-1; i++)
        {
            int m=i;
            for (int j=i+1; j<n; j++)
                if (k[j]<k[m])
                    m=j;
            if (m!=i)
                dc(k[i],k[m]);
        }
}

void is(int k[], int n)
{
    int i,j,x;
    for (i=1;i<n;i++)
        {x=k[i];
            j=i-1;
            while ((j>=0) && (x<k[j]))
                {k[j+1]=k[j];
                    j=j-1;
                }
            k[j+1]=x;
        }
}

void qs(int k[],int l,int r)
{

```

```

int i=l;
int j=r;
int key=k[(l+r)/2];
while (i<=j)
{
    while ((k[i]<key)&&(i<r))
        i++;
    while ((k[j]>key)&&(j>l))
        j--;
    if(i<=j)
    {
        dc(k[i],k[j]);
        i++;
        j--;
    }
}
if(l<j)
    qs(k,l,j);
if(i<r)
    qs(k,i,r);
}

void vundong(int k[],int i, int n)
{
    int j=2*i+1;
    while (j<=n-1)
    {
        if ((j<n-1)&&(k[j]<k[j+1]))
            j++;
        if (k[i]<k[j])
            dc(k[i],k[j]);
        j=2*j+1;
    }
}

```

```

    }
}

void sxvd(int k[],int n)
{
    int i;
    for(i=n/2; i>=0; i--)
        vundong(k,i,n);
    for(i=n-1;i>0;i--)
    {
        dc(k[0],k[i]);
        vundong(k,0,i);
    }
}

void main()
{
    clrscr();
    int a[20];
    int m;//=10;
    int chon;
    do
    {
        cout<<"-----MENUE-----";
        cout<<"\n1. Nhap day can sap xep";
        cout<<"\n2. In day";
        cout<<"\n3. Sap xep kieu lua chon";
        cout<<"\n4. Sap xep kieu chen";
        cout<<"\n5. Sap xep kieu doi cho";
        cout<<"\n6. Sap xep nhanh";
        cout<<"\n7. Sap xep vun dong";
        cout<<"\n9. Thoat";
        cout<<"\nMoi chon chuc nang:"; cin>>chon;

```

```

        switch (chon)
    {
        case(1):
            cout<<"\nNhap so ptu:";
            cin>>m;
            nhap(a,m);
            break;
        case(2):
            indl(a,m);
            break;
        case(3):
            sx(a,m);
            break;
        case(4):
            is(a,m);
            break;
        case(6):
            qs(a,0,m-1);
            break;
        case(7):
            sxvd(a,m);
            break;
        case(9):
            break;
    }

    getch();
}while(chon!=9);
}

```


4.5. Sắp xếp kiểu hoà nhập (Merge Sort)

4.5.1. Phép hoà nhập hai đường

Giả sử ta đã có hai dãy A và B đều đã được sắp xếp, ta có thể “trộn” hai dãy này thành một dãy C cũng đã được sắp xếp. Ý tưởng cơ bản của nó như sau:

- So sánh hai khoá nhỏ nhất của A và B chọn khoá nhỏ hơn để đưa vào C. Khoá được chọn đó bị loại ra khỏi dãy chứa nó.

- Quá trình cứ tiếp tục như vậy cho đến khi một trong hai dãy đã hết. Lúc đó chỉ cần chuyển toàn bộ phần đuôi của dãy còn lại ra sau dãy sắp xếp là xong.

Giải thuật:

```
void Merging(A,r,B,s,C);
```

```
1. {Khởi tạo các chỉ số}
```

```
    i = 1; j = 1; k = 1;
```

```
2. {So sánh để chọn phần tử nhỏ hơn}
```

```
    while (i<r && j<s)
```

```
        if(A[i]<A[j])
```

```
            {C[k] = A[i];
```

```
            i = i+1;
```

```
            k = k+1;
```

```
        }
```

```
    else
```

```
        { C[k] = A[j];
```

```
        j = j+1;
```

```
        k = k+1;
```

```
    }
```

```
3. {Một trong hai dãy đã hết}
```

```
    if (i>r)
```

```
        for(t = 0; t<= s-j; t++)
```

```
            C[k+t] = B[j+t];
```

else for (t = 0; t <= r - i; t++)

C[k+t] := A[i+t];

4. return;

Chú ý: Trường hợp chỉ số phần tử đầu của A, B, C không phải là 1 thì giải thuật hợp nhất hai đường vẫn tương tự như trên với một vài thay đổi nhỏ:

void Merge(A,r,LBA,B,s,LBB,C,LBC)

1. i = LBA; j = LBB; k = LBC;

UBA = LBA + r - 1; LBB = LBB + s - 1;

{Các bước còn lại tương tự, chỉ thay r bởi UBA, s bởi UBB}

4.5.2. Sắp xếp kiểu hoà nhập hai đường trực tiếp

Từ phương pháp hoà nhập hai dãy, có thể dễ dàng cải tiến để hoà nhập nhiều dãy.

Mỗi khoá trong dãy khoá có thể được coi là một dãy có độ dài bằng 1.

Nếu hoà nhập hai dãy như vậy sẽ được một dãy mới có độ dài bằng 2. Lại hoà nhập hai dãy có độ dài bằng 2 ta sẽ được dãy có độ dài bằng 4; và cứ tương tự như thế, cuối cùng ta sẽ có dãy có độ dài bằng n, đó chính là dãy khoá đã được sắp xếp.

Q là số cặp dãy có độ dài bằng 1 thì $Q = n \text{ div } (2*1)$.

Gọi $S = 2*1*Q$ thì S là số các phần tử của Q cặp dãy.

Suy ra $R = n - S$ là số các phần tử còn lại của dãy K.

void MPASS(K,n,l,X);

1. $Q = n \text{ div } (2*1)$;

$S = 2*1*Q$;

$R = n - S$;

2. *{Hoà nhập từng dãy}*

for(j=1; j <= Q; j++)

*{ i = 1 + (2*j - 2)*1; {xác định chỉ số phần tử đầu của dãy thứ nhất}*

Merge(K,1,LB,K,1,LB+1,X,LB);

end;

3. *{Chỉ còn một dãy}*

```

    if(R<=1)
        for(j=1; j<= R; j++)
            X[S+j] := A[S+j];
    elseMerge(K,1,S+1,K,R-1,1+S+1,X,S+1);
4. return;
void Hoa_Nhap(K,n);
1. {Khởi tạo}
    l=1;
2. while(l<n)
    { MPASS(K,n,l,X);
      MPASS(X,n,2*l,K);
      l = 4*l;
    }
3. return;

```

4.6. Bài toán tìm kiếm

Bài toán tìm kiếm được phát biểu như sau: Cho một bảng gồm n bản ghi R_1, R_2, \dots, R_n ; mỗi bản ghi tương ứng với một khoá K_i ($1 \leq i \leq n$). Hãy tìm bản ghi có giá trị khoá bằng X cho trước. X được gọi là khoá tìm kiếm.

Công việc tìm kiếm sẽ kết thúc khi một trong hai tình huống sau xuất hiện:

- Tìm thấy bản ghi có giá trị khoá bằng X . Khi đó ta nói phép tìm kiếm được thoả mãn.

- Không tìm thấy bản ghi nào có giá trị khoá bằng X . Khi đó ta nói phép tìm kiếm không thoả mãn. Nếu phép tìm kiếm không thoả có thêm yêu cầu bổ sung một bản ghi mới có giá trị khoá bằng X thì giải thuật tương ứng được gọi là giải thuật tìm kiếm có bổ sung..

- Tương tự như các phương pháp sắp xếp, ta sẽ coi các giá trị khoá đại diện cho các bản ghi

4.7. Một số phương pháp tìm kiếm

4.7.1. Tìm kiếm tuần tự

Nội dung phương pháp: Bắt đầu từ vị trí khoá thứ nhất ta lần lượt so sánh khoá tìm kiếm với các khoá trong bảng cho đến khi tìm được khoá mong muốn hoặc hết bảng mà không tìm thấy.

Lập giải thuật tìm kiếm trong dãy khoá gồm n khoá xem có khoá nào bằng X hay không. Nếu có thì trả về giá trị chỉ số thứ tự của khoá tìm được, ngược lại trả về giá trị không.

```
int Sequen_Search(K,n,X);
```

```
1. {Tìm kiếm}
```

```
    i = 1;
```

```
    while (i<=n && K[i]!=X)
```

```
        i = i+1;
```

```
2. {Trả về giá trị}
```

```
    if (i=n+1)
```

```
        return(0);
```

```
    elsereturn(i);
```

4.7.2. Tìm kiếm nhị phân

- Tìm kiếm nhị phân là một phương pháp tìm kiếm khá thông dụng, nếu số lượng bản ghi lớn thì thời gian tìm kiếm sẽ được rút ngắn dựa trên sự ứng dụng của sơ đồ chia để trị.

- Phép tìm kiếm nhị phân luôn luôn chọn khoá ở giữa dãy khoá đang xét để so sánh với dãy khoá tìm kiếm. Giả sử dãy khoá đang xét là $K[l], K[l+1], \dots, K[r]$ thì khoá ở giữa dãy khoá là $K[i]$, với $i=(l+r) \text{ div } 2$. Việc tìm kiếm sẽ kết thúc khi $K[i]=X$. Nếu $X < K[i]$, phép tìm kiếm được thực hiện tương tự trên dãy khoá $K[l], K[l+1], \dots, K[i-1]$. Nếu $X > K[i]$, phép tìm kiếm được thực hiện tương tự trên dãy khoá $K[i+1], \dots, K[r]$.

- Cho dãy khoá K đã được sắp xếp tăng dần, lập giải thuật trả về vị trí khoá có giá trị bằng X cho trước, ngược lại trả về giá trị 0.

```
int Binary_Search(K,n,X);
```

1. *{Khởi tạo}*

$l = 1; r = n;$

2. *{Tìm kiếm}*

while($l \leq r$)

{ $m = (l+r) \text{ div } 2;$

if ($X < K[m]$)

$r = m - 1;$

else

if ($X > K[m]$)

$l = m + 1;$

else return(m);

}

3. *{Tìm kiếm không thoả}*

if ($l > r$) **return**(0);

Nhận xét:

- Phương pháp tìm kiếm tuần tự cho độ phức tạp trung bình và xấu nhất đều là $O(n)$.

- Phương pháp tìm kiếm nhị phân trong trường hợp giải thuật được viết dưới dạng đệ quy cho độ phức tạp trong trường hợp trung bình và xấu nhất đều là $O(\log_2 n)$. So với tìm kiếm tuần tự, chi phí tìm kiếm nhị phân ít hơn khá nhiều.

- Đối với giải thuật tìm kiếm nhị phân cần lưu ý dãy khoá phải được sắp xếp rồi, do đó chi phí thời gian dành cho sắp xếp cũng phải được kể đến. Nếu dãy khoá luôn biến động thì chi phí thời gian dành cho sắp xếp nổi lên rất rõ, và đây chính là nhược điểm của phương pháp tìm kiếm nhị phân.

Bài tập chương IV

4.1. Cài đặt các thuật toán sắp xếp cơ bản bằng ngôn ngữ lập trình C trên 1 mảng các số nguyên, dữ liệu của chương trình được nhập vào từ file text được sinh ngẫu nhiên (số phần tử khoảng 10000) và so sánh thời gian thực hiện thực tế của các thuật toán.

4.2. Cài đặt các thuật toán sắp xếp nâng cao bằng ngôn ngữ C với một mảng các cấu trúc sinh viên (tên: xâu ký tự có độ dài tối đa là 50, tuổi: số nguyên, điểm trung bình: số thực), khóa sắp xếp là trường tên. So sánh thời gian thực hiện của các thuật toán, so sánh với hàm qsort() có sẵn của C.

4.3. Cài đặt của các thuật toán sắp xếp có thể thực hiện theo nhiều cách khác nhau. Hãy viết hàm nhận input là mảng $a[0..i]$ trong đó các phần tử ở chỉ số 0 tới chỉ số $i-1$ đã được sắp xếp tăng dần, $a[i]$ không chứa phần tử nào, và một số x , chèn x vào mảng $a[0..i-1]$ sao cho sau khi chèn kết quả nhận được là $a[0..i]$ là một mảng được sắp xếp. Sử dụng hàm vừa xây dựng để cài đặt thuật toán sắp xếp chèn.

Gợi ý: Có thể cài đặt thuật toán chèn phần tử vào mảng như phần cài đặt của thuật toán sắp xếp chèn đã được trình bày hoặc sử dụng phương pháp đệ quy.

4.4. Viết chương trình nhập vào 1 mảng số nguyên và một số nguyên k , hãy đếm xem có bao nhiêu số bằng k . Nhập tiếp 2 số $x < y$ và đếm xem có bao nhiêu số lớn hơn x và nhỏ hơn y .

4.5. Cài đặt thuật toán tìm kiếm tuyến tính theo kiểu đệ quy.

4.6. Viết chương trình nhập một mảng các số nguyên từ bàn phím, nhập 1 số nguyên S , hãy đếm xem có bao nhiêu cặp số của mảng ban đầu có tổng bằng S , có hiệu bằng S .

Chương V. ĐỒ THỊ

A. Mục tiêu

Sau khi hoàn tất chương này sinh viên sẽ hiểu và nắm được các kiến thức cơ bản sau:

- Đồ thị, cách biểu diễn đồ thị, các phép duyệt trên đồ thị.
- Một số bài toán trên đồ thị như: bao đóng truyền ứng, tìm đường đi ngắn nhất, xây dựng cây khung tối thiểu.

B. Nội dung

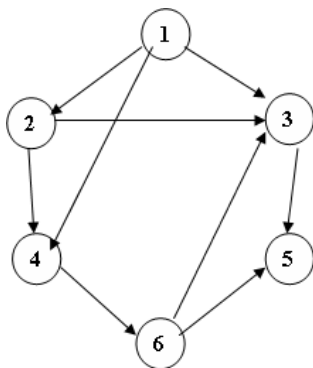
5.1. Khái niệm và biểu diễn đồ thị

5.1.1. Khái niệm

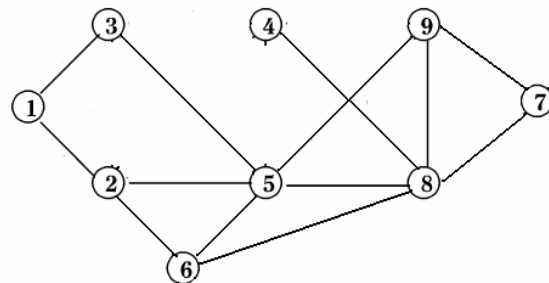
Định nghĩa đồ thị: Một đồ thị $G(V, E)$ bao gồm một tập hữu hạn V các nút hay đỉnh và một tập hữu hạn E các cặp đỉnh mà ta gọi là cung.

Nếu $(v_1, v_2) \in E$ thì ta nói có một cung nối v_1 và v_2 . Nếu cung (v_1, v_2) khác với cung (v_2, v_1) thì ta có một đồ thị định hướng, lúc đó (v_1, v_2) được gọi là cung định hướng từ v_1 đến v_2 . Nếu thứ tự các nút trên cung không được coi trọng thì ta có đồ thị không định hướng.

Bảng hình vẽ có thể biểu diễn đồ thị như sau:



a. Đồ thị định hướng



b. Đồ thị không định hướng

Nếu $(v_1, v_2) \in E$ thì v_1, v_2 gọi là lân cận của nhau.

Một đường đi từ v_p đến v_q trong đồ thị G là một dãy các đỉnh $v_p, v_{i1}, v_{i2}, \dots, v_{in}, v_q$ mà $(v_p, v_{i1}), (v_{i2}, v_{i3}), \dots, (v_{in}, v_q)$ là các cung trong $E(G)$. Số lượng các cung trên đường đi ấy gọi là độ dài đường đi.

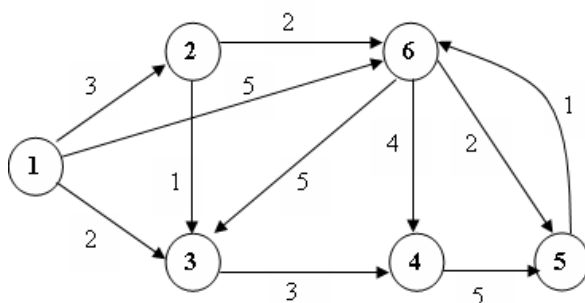
Một đường đi đơn là đường đi mà mọi đỉnh trên đó, trừ đỉnh đầu và đỉnh cuối đều khác nhau.

Một chu trình là một đường đi đơn mà đỉnh đầu và đỉnh cuối trùng nhau.

Trong đồ thị G , hai đỉnh v_i và v_j gọi là liên thông nếu có một đường đi từ v_i tới v_j .

Một đồ thị G gọi là liên thông nếu đối với mọi cặp đỉnh phân biệt v_i, v_j trong $V(G)$ đều có một đường đi từ v_i tới v_j .

Có khi mỗi cung của đồ thị người ta gán một giá trị thể hiện một thông tin nào đó liên quan tới cung - được gọi là trọng số, trong trường hợp này đồ thị được gọi là đồ thị có trọng số. Ví dụ: mạng lưới giao thông đường bộ nối các tỉnh với trọng số ứng với mỗi tuyến đường nối giữa 2 tỉnh là độ dài của tuyến đường đó.



5.1.2. Biểu diễn đồ thị

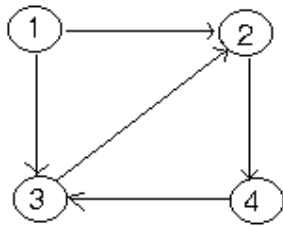
Có nhiều cấu trúc để biểu diễn đồ thị, việc lựa chọn cấu trúc nào tùy thuộc vào các ứng dụng và các phép xử lý cần tác động lên đồ thị trong ứng dụng đó.

a. Biểu diễn bằng ma trận kề

Xét đồ thị $G(V, E)$ với V gồm n đỉnh ($n \geq 1$), giả sử các đỉnh đã được đánh số thứ tự theo một quy định nào đó.

Ma trận lân cận A biểu diễn G là một ma trận vuông cấp n , $a_{ij} = 0$ nếu không tồn tại cung từ đỉnh thứ i đến đỉnh thứ j , $a_{ij} = 1$ nếu tồn tại cung từ đỉnh i tới đỉnh j .

Ví dụ:



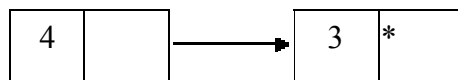
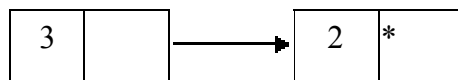
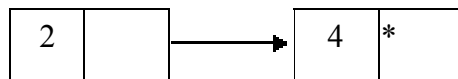
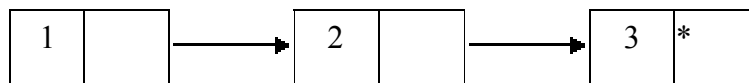
$\begin{matrix} j \\ i \end{matrix}$	1	2	3	4
1	0	1	1	0
2	0	0	0	1
3	0	1	0	0
4	0	0	1	0

b. Biểu diễn bằng danh sách lân cận kề

Trong cách biểu diễn này, n hàng của ma trận lân cận được thay bởi n danh sách móc nối.

Với mỗi đỉnh của đồ thị có một danh sách tương ứng. Các nút trong danh sách i biểu diễn các đỉnh lân cận của nút i. Mỗi nút có 2 trường là Vertex và Link. Trường Vertex chứa chỉ số của các đỉnh lân cận của đỉnh i. Trường Link chứa con trỏ, trỏ tới nút tiếp theo trong danh sách.

Ví dụ: Với đồ thị như hình vẽ trên có thể biểu diễn:



5.2. Duyệt và tìm kiếm trên đồ thị

5.2.1. Tìm kiếm theo chiều sâu

Xuất phát từ đỉnh v được thăm. Tiếp theo một đỉnh w chưa được thăm mà là lân cận của v sẽ được chọn và một phép tìm kiếm theo chiều sâu xuất phát từ w lại được thực hiện.

Khi một đỉnh u được với tới mà mọi đỉnh lân cận của nó đều đã được thăm rồi thì ta sẽ quay ngược lên đỉnh cuối cùng vừa được thăm và phép tìm kiếm theo chiều sâu lại được thực hiện.

Giải thuật

void DFS(v)

{Dùng mảng Visited(n), các phần tử nhận giá trị 0, nếu đỉnh nào được thăm thì phần tử có chỉ số đó nhận giá trị 1 }

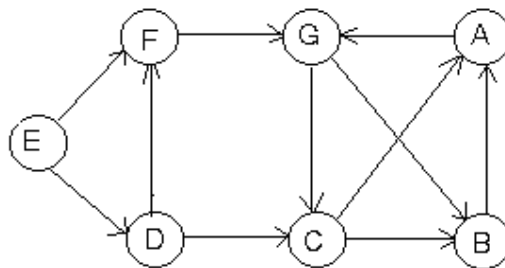
1. **Visited**(v)=1

2. **for**(mỗi đỉnh w là lân cận của v)

if(Visited(w)=0) DFS(w)

3. **Return**

Ví dụ:



Giả sử ta bắt đầu duyệt từ đỉnh A, tức là dfs(A). Giải thuật sẽ đánh dấu là A đã được duyệt, rồi chọn đỉnh đầu tiên trong danh sách các đỉnh kề với A, đó là G. Tiếp tục duyệt đỉnh G, G có hai đỉnh kề với nó là B và C, theo thứ tự đó thì đỉnh kế tiếp được duyệt là đỉnh B. B có một đỉnh kề đó là A, nhưng A đã được duyệt nên phép duyệt dfs(B) đã hoàn tất. Bây giờ giải thuật sẽ tiếp tục với đỉnh kề với G mà còn chưa duyệt là C. C không có đỉnh kề nên phép duyệt dfs(C) kết thúc vậy dfs(A) cũng kết thúc. Còn lại 3 đỉnh chưa được duyệt là D,E,F và theo thứ tự đó thì D được duyệt, kế đến là F. Phép duyệt dfs(D) kết thúc và còn một đỉnh E chưa được duyệt. Tiếp tục duyệt E và kết thúc. Nếu ta in các đỉnh của đồ thị trên theo thứ tự được duyệt ta sẽ có danh sách sau: AGBCDFE.

5.2.2. Tìm kiếm theo chiều rộng

Đỉnh xuất phát v được thăm sau đó các đỉnh chưa được thăm mà là lân cận của v sẽ được thăm kế tiếp rồi mới đến các đỉnh chưa được thăm mà là lân cận của các đỉnh này.

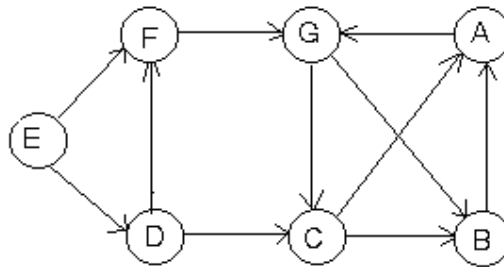
Giải thuật

void BFS(*v*);

```
1. Visited(v) = 1;  
2. Q_INSERT(v,Q);  
3. while(Q không rỗng)  
{ Q_DELETE(Q);  
  for (mỗi đỉnh w lân cận với v)  
    if (Visited(w) = 0)  
    { Visited(w) = 1;  
      Q_INSERT(w,Q);  
    }  
}
```

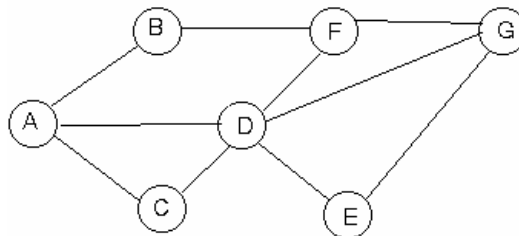
4. **return**

Ví dụ:



A chỉ có một đỉnh kề G, nên ta duyệt G. Kế đến duyệt tất cả các đỉnh kề với G; đó là B, C. Sau đó duyệt tất cả các đỉnh kề với B, C theo thứ tự đó. Các đỉnh kề với B, C đều đã được duyệt, nên ta tiếp tục duyệt các đỉnh chưa được duyệt. Các đỉnh chưa được duyệt là D, E, F. Duyệt D, kế đến là F và cuối cùng là E. Vậy thứ tự các đỉnh được duyệt là: AGBCDFE.

Ví dụ:



Giả sử bắt đầu duyệt từ A. Duyệt A, kế đến duyệt tất cả các đỉnh kề với A; đó là B, C, D theo thứ tự đó. Kế tiếp là duyệt các đỉnh kề của B, C, D theo thứ tự đó. Vậy các

nút được duyệt tiếp theo là F, E, G. Có thể minh hoạ hoạt động của hàng đợi trong phép duyệt trên như sau:

Duyệt A nghĩa là đánh dấu Visited và đưa nó vào hàng đợi:

A

Kế đến duyệt tất cả các đỉnh kề với đỉnh đầu hàng đợi mà chưa được duyệt; tức là loại A khỏi hàng đợi, duyệt B, C, D và đưa chúng vào hàng đợi, bây giờ hàng đợi chứa các đỉnh B, C, D.

B
C
D

Kế đến B được lấy ra khỏi hàng đợi và các đỉnh kề với B mà chưa được duyệt, đó là F, sẽ được duyệt, và F được đưa vào hàng đợi.

C
D
F

Kế đến thì C được lấy ra khỏi hàng đợi và các đỉnh kề với C mà chưa được duyệt sẽ được duyệt. Không có đỉnh nào như vậy, nên bước này không có thêm đỉnh nào được duyệt.

D
F

Kế đến thì D được lấy ra khỏi hàng đợi và duyệt các đỉnh kề chưa duyệt của D, tức là E, G được duyệt. E, G được đưa vào hàng đợi.

F
E
G

Tiếp tục, F được lấy ra khỏi hàng đợi. Không có đỉnh nào kề với F mà chưa được duyệt. Vậy không duyệt thêm đỉnh nào.

E
G

Tương tự như F, E rồi đến G được lấy ra khỏi hàng đợi. Hàng đợi trở thành rỗng và giải thuật kết thúc.

5.3. Một số bài toán trên đồ thị

5.3.1. Bài toán bao đóng truyền ứng

Trong một số trường hợp ta chỉ cần xác định có hay không có đường đi nối giữa hai đỉnh i, j bất kỳ. Có thể coi ma trận lân cận như một ma trận Bool. Như vậy có thể tác động lên ma trận đó các phép toán logic:

- Phép cộng (Phép tuyển, phép hoặc): \vee
- Phép nhân (Phép hội, phép và): \wedge

a	b	$a \vee b$	$a \wedge b$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

Trong ma trận lân cận A, nếu $a_{ij} = 1$ thì có đường đi có độ dài bằng 1 từ i tới j.

Xét $A^{(2)} = A \wedge A$, nếu phần tử ở hàng i, cột j của $A^{(2)}$ bằng 1 thì ít nhất cũng có một đường đi có độ dài bằng 2 từ i tới j vì:

$$A^{(2)}_{ij} = \vee (a_{ik} \wedge a_{kj}) = 1 \quad (k=1..n)$$

thì ít nhất cũng có một giá trị k mà $a_{ik} \wedge a_{kj} = 1$, mà $a_{ik} \wedge a_{kj}$ chỉ bằng 1 khi cả hai bằng 1, nghĩa là khi có đường đi độ dài bằng 1 từ đỉnh i tới đỉnh k và có đường đi độ dài bằng 1 từ đỉnh k tới đỉnh j.

Từ đó suy ra điều tương tự cho $A^{(r)} = A \wedge A^{(r-1)}$, $r=3,4,\dots$, nghĩa là nếu $a_{ij}^{(r)} = 1$ thì ít nhất có một đường đi độ dài r từ i tới j.

Như vậy, nếu ta lập am trận:

$$P = A \vee A^{(2)} \vee \dots \vee A^{(n)} = \vee A^{(k)} \quad (k=1..n)$$

Thì P sẽ cho biết có hay không đường đi, có độ dài lớn nhất là n, từ đỉnh i tới đỉnh j. P được gọi là ma trận đường đi của đồ thị. Đồ thị tương ứng với P được gọi là bao đóng truyền ứng của đồ thị đã cho.

Giải thuật Warshall tính ma trận P

void Warshall(A,P,n);

1. **for**(i=1; i<= n; i++)

for(j=1; j< n; j++)

P[i,j]=A[i,j];

2. **for** (k=1; k<=n; k++)

for(i=1; i<= n; i++)

for(j=1; j<=n; j++)

P[i,j]=P[i,j]**or**P[i,k] **and** P[k,j];

3.return

5.3.2. Bài toán một nguồn, mọi đích

Cho đồ thị G với tập các đỉnh V và tập các cạnh E (đồ thị có hướng hoặc vô hướng). Mỗi cạnh của đồ thị có một nhãn, đó là một giá trị không âm, nhãn này còn gọi là giá (cost) của cạnh. Cho trước một đỉnh v xác định, gọi là đỉnh nguồn. Vấn đề là tìm đường đi ngắn nhất từ v đến các đỉnh còn lại của G; tức là các đường đi từ v đến các đỉnh còn lại với tổng các giá (cost) của các cạnh trên đường đi là nhỏ nhất. Chú ý rằng nếu đồ thị có hướng thì đường đi này là đường đi có hướng.

Ta có thể giải bài toán này bằng cách xác định một tập hợp S chứa các đỉnh mà khoảng cách ngắn nhất từ nó đến đỉnh nguồn v đã biết. Khởi đầu $S=\{v\}$, sau đó tại mỗi bước ta sẽ thêm vào S các đỉnh mà khoảng cách từ nó đến v là ngắn nhất. Với giả thiết mỗi cung có một giá không âm thì ta luôn luôn tìm được một đường đi ngắn nhất như vậy mà chỉ đi qua các đỉnh đã tồn tại trong S. Để chi tiết hoá giải thuật, giả sử G có n đỉnh và nhãn trên mỗi cung được lưu trong mảng hai chiều C, tức là $C[i,j]$ là giá (có thể xem như độ dài) của cung (i,j), nếu i và j không nối nhau thì $C[i,j]=\infty$. Ta dùng mảng 1 chiều D có n phần tử để lưu độ dài của đường đi ngắn nhất từ mỗi đỉnh của đồ thị đến v. Khởi đầu khoảng cách này chính là độ dài cạnh (v,i), tức là $D[i]:=C[v,i]$. Tại

mỗi bước của giải thuật thì $D[i]$ sẽ được cập nhật lại để lưu độ dài đường đi ngắn nhất từ đỉnh v tới đỉnh i , đường đi này chỉ đi qua các đỉnh đã có trong S .

Để cài đặt giải thuật dễ dàng, ta giả sử các đỉnh của đồ thị được đánh số từ 1 đến n , tức là $V=\{1,...,n\}$ và đỉnh nguồn là 1. Dưới đây là giải thuật Dijkstra để giải bài toán trên.

void Dijkstra;

1. $S=[1]$; { S chỉ chứa một đỉnh nguồn}

2. **for**($i=2$; $i \leq n$; $i++$)

$D[i]=C[1,i]$; {khởi đầu các giá trị cho D }

3. **for**($i=1$; $i < n$; $i++$)

{

Lấy đỉnh w trong $V-S$ sao cho $D[w]$ nhỏ nhất;

Thêm w vào S ;

for (mỗi đỉnh u thuộc $V-S$)

$D[u] = \min(D[u], D[w] + C[w,u]);$

}

4. return

Nếu muốn lưu trữ lại các đỉnh trên đường đi ngắn nhất để có thể xây dựng lại đường đi này từ đỉnh nguồn đến các đỉnh khác, ta dùng một mảng P . Mảng này sẽ lưu $P[u]=w$ với u là đỉnh “trước” đỉnh w trong đường đi. Lúc khởi đầu $P[u]=1$ với mọi u .

Giải thuật Dijkstra được viết lại như sau:

void Dijkstra;

1. $S=[1]$; { S chỉ chứa một đỉnh nguồn}

2. **for**($i=2$; $i \leq n$; $i++$)

{ $P[i]:=1$; {khởi tạo giá trị cho P }

$D[i]:=C[1,i]$; {khởi đầu các giá trị cho D }

}

3. **for**($i=1$; $i < n$; $i++$)

{ Lấy đỉnh w trong $V-S$ sao cho $D[w]$ nhỏ nhất;

Thêm w vào S;

for (mỗi đỉnh u thuộc V-S)

if ($D[w] + C[w,u] < D[u]$)

{ $D[u] := D[w] + C[w,u]$;

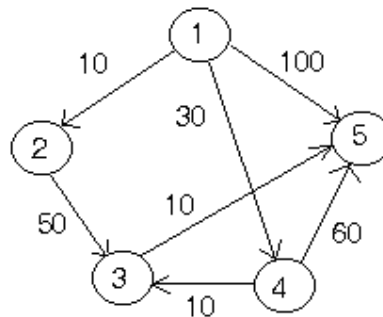
$P[u] := w$;

}

}

4. return

Ví dụ: Áp dụng giải thuật Dijkstra cho đồ thị:



Kết quả khi áp dụng giải thuật

Lần lặp	S	W	D[2]	D[3]	D[4]	D[5]
khởi đầu	{1}	-	10	∞	30	100
1	{1,2}	2	10	60	30	100
2	{1,2,4}	4	10	40	30	90
3	{1,2,3,4}	3	10	40	30	50
4	{1,2,3,4,5}	5	10	40	30	50

Mảng P có giá trị như sau:

P	1	2	3	4	5
		1	4	1	3

Từ kết quả trên ta có thể suy ra rằng đường đi ngắn nhất từ đỉnh 1 đến đỉnh 3 là $1 \rightarrow 4 \rightarrow 3$ có độ dài là 40. Đường đi ngắn nhất từ 1 đến 5 là $1 \rightarrow 4 \rightarrow 3 \rightarrow 5$ có độ dài 50.

5.3.3. Cây khung tối thiểu

Giả sử $G=\{V,E\}$ là một đồ thị vô hướng, liên thông, trong đó các cạnh có gắn trọng số.

T được gọi là cây khung của G nếu cây T có chứa tất cả các đỉnh của G và có chứa các cạnh mà không tạo thành chu trình.

Tìm cây khung tối thiểu: Dùng thuật toán Prim hoặc Kruskal

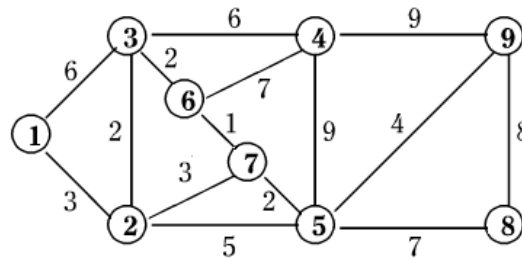
PRIM

- Ý tưởng thuật toán Prim:

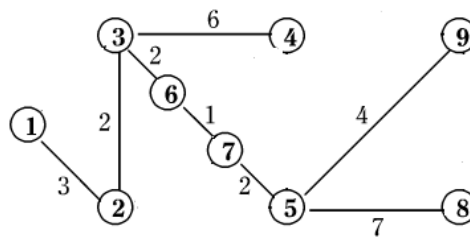
+ Từ một đỉnh v nào đó, trong các lân cận của v , tìm đỉnh gần nhất, nghĩa là (v,u) có trọng số nhỏ nhất.

+ Tiếp theo, trong số các cạnh kề với v hoặc u , ta chọn cạnh có trọng số nhỏ nhất mà không tạo thành chu trình với (v,u) . Tiếp tục quá trình đến khi cây gồm n đỉnh và $n-1$ cạnh. Đó chính là cây khung bé nhất.

Ví dụ: Cho đồ thị:



Cây khung tối thiểu:



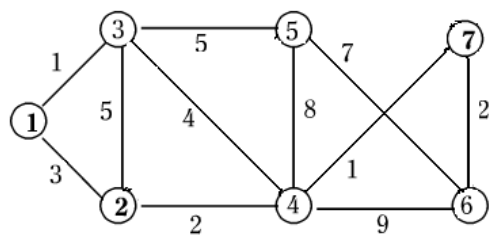
KRUSKAL

- Ý tưởng thuật toán Kruskal:

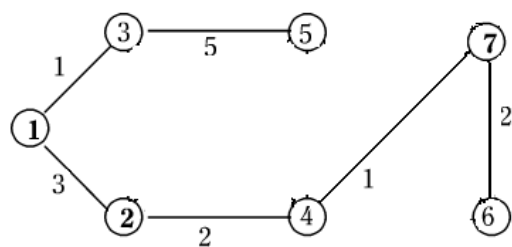
+ Giả sử $G=(V,E)$ là một đồ thị vô hướng, liên thông, trong đó các cung có gắn trọng số.

+ Cây khung tối thiểu được xây dựng dần dần từng cung một. Một cung được đưa vào cây khung nếu nó có giá trị thấp nhất và không tạo thành chu trình với các cung đã có trong cây khung. Tiếp tục quá trình đến khi thu được đồ thị có n đỉnh và $n-1$ cung, đó là cây khung tối thiểu.

Ví dụ: Cho đồ thị:

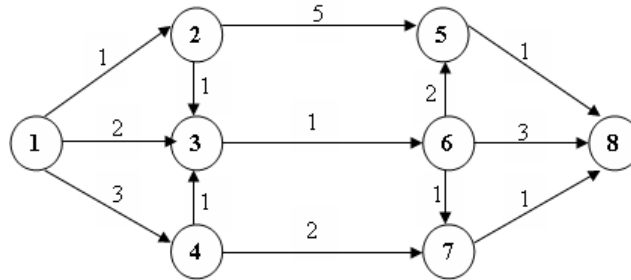


Cây khung tối thiểu:



Bài tập chương V

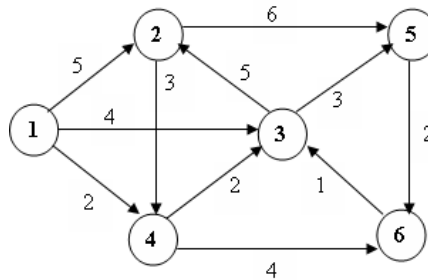
5.1. Cho đồ thị sau:



a. Dùng thuật toán Dijkstra tìm đường đi ngắn nhất từ đỉnh 1 đến các đỉnh còn lại trong đồ thị.

b. Trình bày ý tưởng thuật toán Kruskal để tìm cây khung cực tiểu, lấy đỉnh xuất phát là đỉnh 1.

5.2. Cho đồ thị sau:



a. Dùng thuật toán Dijkstra tìm đường đi ngắn nhất từ đỉnh 1 đến các đỉnh còn lại trong đồ thị.

b. Trình bày ý tưởng duyệt đồ thị theo chiều rộng. Áp dụng đối với đồ thị trên.

5.3. Cho ma trận kề (trọng số) của một đồ thị có hướng như sau:

A_{ij}	1	2	3	4	5	6
1	-	2	∞	3	∞	∞
2	∞	-	5	∞	4	∞
3	∞	∞	-	∞	∞	2
4	∞	1	∞	-	2	∞
5	∞	∞	1	∞	-	∞
6	∞	∞	∞	∞	4	-

- Vẽ đồ thị trên.
- Dùng thuật toán Dijkstra tìm đường đi ngắn nhất từ đỉnh 1 đến các đỉnh còn lại của đồ thị.

- Nêu ý tưởng giải thuật Prim, vẽ từng bước quá trình tìm cây khung tối thiểu của đồ thị trên theo giải thuật Prim, lấy đỉnh xuất phát là đỉnh 1.

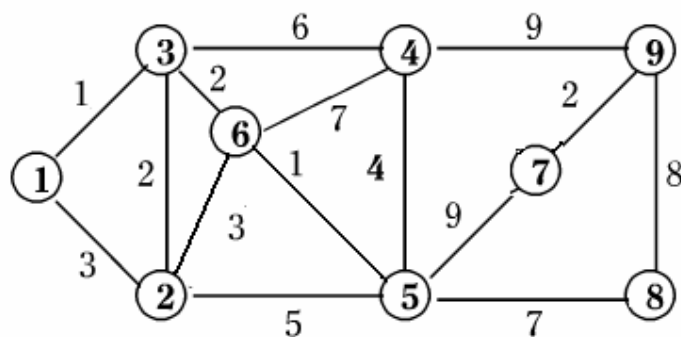
5.4. Cho ma trận kề (trọng số) của một đồ thị có hướng như sau:

A_{ij}	1	2	3	4	5	6
1	-	2	5	∞	∞	∞
2	∞	-	1	8	∞	3
3	∞	∞	-	4	7	∞
4	∞	∞	∞	-	∞	2
5	∞	∞	∞	∞	-	∞
6	∞	∞	∞	∞	1	-

- Vẽ đồ thị trên.
- Dùng thuật toán Dijkstra tìm đường đi ngắn nhất từ đỉnh 1 đến các đỉnh còn lại của đồ thị.
- Nêu ý tưởng giải thuật Kruskal, vẽ từng bước quá trình tìm cây khung tối thiểu của đồ thị trên theo giải thuật Kruskal, lấy đỉnh xuất phát là đỉnh 1.

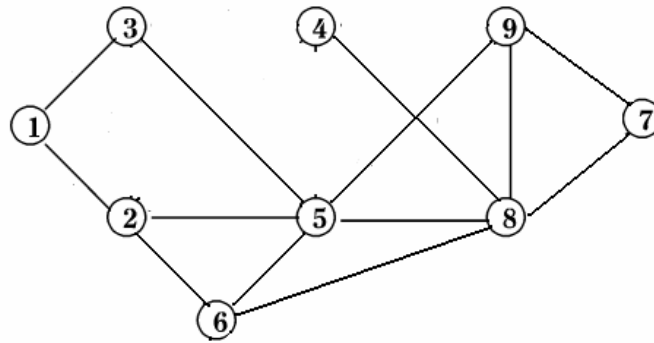
5.5.a. Trình bày khái niệm đồ thị, đồ thị liên thông, cây khung.

b. Cho đồ thị sau:



- Biểu diễn đồ thị bằng ma trận kề.
- Tìm cây khung nhỏ nhất của đồ thị bằng thuật toán Prim, lấy đỉnh xuất phát là đỉnh 1.

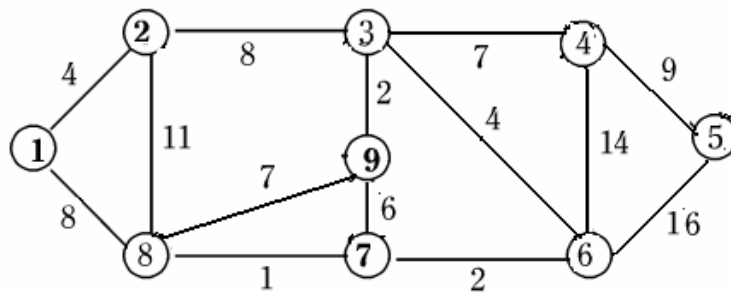
5.6. Cho đồ thị sau:



a. Trình bày nội dung phương pháp và giải thuật duyệt theo chiều sâu, nêu thứ tự các đỉnh thu được khi duyệt theo chiều sâu đồ thị trên.

b. Biểu diễn đồ thị bằng ma trận kề.

5.7. Cho đồ thị sau:



a. Trình bày nội dung phương pháp duyệt theo chiều rộng và áp dụng với đồ thị trên.

b. Trình bày ý tưởng thuật toán Prim và tìm cây khung nhỏ nhất của đồ thị bằng thuật toán Prim, lấy đỉnh xuất phát là đỉnh 1.

TÀI LIỆU THAM KHẢO

1. Đỗ Xuân Lôi, Cấu trúc dữ liệu và giải thuật, Nhà xuất bản Đại học Quốc gia Hà Nội
2. Nguyễn Đình Hoá, Cấu trúc dữ liệu và thuật giải. Nhà xuất bản Đại học Quốc gia Hà Nội.
3. Nguyễn Đức Nghĩa, Nguyễn Tô Thành, Toán rời rạc, Nhà xuất bản giáo dục
4. Phạm Văn Át, Kỹ thuật lập trình C. Nhà xuất bản thống kê.
5. Nguyễn Thị Tĩnh (Chủ biên), Cấu trúc dữ liệu và giải thuật. Nhà xuất bản Đại học Sư phạm.



ĐỀ CƯƠNG MÔN HỌC: CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Ngành đào tạo: CNTT

Bộ môn Công nghệ Thông tin; Khoa Kỹ thuật – Công nghệ

1. Thông tin về giảng viên:



❖ **Họ và tên: Bùi Thị Thu Hoài**

- Chức danh, học hàm, học vị: Thạc Sĩ
- Thời gian, địa điểm làm việc: Thứ 2 hàng tuần tại Khoa Kỹ thuật – Công nghệ, Trường Đại học Hà Tĩnh
- Địa chỉ liên hệ: Khoa Kỹ thuật – Công nghệ, Trường Đại học Hà Tĩnh
- Điện thoại: 0979707783
- E-mail: hoai.buithithu@htu.edu.vn
- Các hướng nghiên cứu chính: Cấu trúc dữ liệu và giải thuật, Đồ họa máy tính.

❖ **Họ và tên: Trần Thị Hương**

- Chức danh, học hàm, học vị: Thạc Sĩ
- Thời gian, địa điểm làm việc: Thứ 2 hàng tuần tại Khoa Kỹ thuật – Công nghệ, Trường Đại học Hà Tĩnh
- Địa chỉ liên hệ: Khoa Kỹ thuật – Công nghệ, Trường Đại học Hà Tĩnh
- Điện thoại: 0988747289
- E-mail: huong.tranhti@htu.edu.vn
- Các hướng nghiên cứu chính: Hệ quản trị cơ sở dữ liệu, Lập trình hướng đối tượng, Trí tuệ nhân tạo.

2. Thông tin chung về môn học

- Tên môn học: Cấu trúc dữ liệu và giải thuật
- Mã môn học: CN21023
- Số tín chỉ: 3
- Môn học: - Bắt buộc: 
- Tự chọn: 
- Các môn học tiên quyết: Sinh viên đã học học phần lập trình C
- Các môn học kế tiếp:

- Các yêu cầu đối với môn học (nếu có): Có máy tính
- Giờ tín chỉ đối với các hoạt động:
 - + Nghe giảng lý thuyết: 20 tiết
 - + Thảo luận: 5 tiết
 - + Thực hành: 20 tiết
 - + Tự học: 90
- Khoa/ bộ môn quản lý môn học: Kỹ thuật – Công nghệ

3. Mục tiêu của môn học

Sau khi học xong phần lý thuyết và thực hành môn cấu trúc dữ liệu và giải thuật, sinh viên sẽ có kiến thức cơ bản về một số cấu trúc dữ liệu, các thuật toán cơ bản trên các cấu trúc đó. Sinh viên có thể áp dụng các cấu trúc dữ liệu đã học và tư duy thuật toán để có thể thiết kế và cài đặt một số chương trình bằng ngôn ngữ C.

4. Tóm tắt nội dung môn học

Học phần trình bày các phương pháp tổ chức và những thao tác cơ sở trên từng cấu trúc dữ liệu, kết hợp với việc phát triển tư duy giải thuật để hình thành nên chương trình máy tính. Công cụ được sử dụng là các ngôn ngữ lập trình bậc cao như Pascal, C. Các khái niệm: cấu trúc dữ liệu, giải thuật; Những cấu trúc dữ liệu tuyến tính; Phân tích và thiết kế thuật giải; Sắp xếp và tìm kiếm; Cây và cây tìm kiếm nhị phân; Đồ thị.

5. Nội dung chi tiết môn học

Chương 1. MỞ ĐẦU

(Lý thuyết: 4 tiết, Thảo luận: 2 tiết, Bài tập: 1 tiết, Thực hành: 1 tiết, Tự học: 16 giờ)

1.1. Khái niệm về cấu trúc dữ liệu và giải thuật.

1.1.1. Khái niệm bài toán.

1.1.2. Khái niệm giải thuật

1.1.3. Khái niệm về cấu trúc dữ liệu và mối quan hệ với giải thuật

1.2. Mô tả giải thuật.

1.2.1. Phương pháp sơ đồ khối.

1.2.2. Phương pháp liệt kê.

1.2.3. Phương pháp giả mã

1.3. Độ phức tạp của giải thuật.

1.3.1. Đánh giá thời gian thực hiện của thuật toán.

1.3.2. Một số quy tắc xác định thời gian thực hiện thuật toán.

1.3.3. Phân tích một số thuật toán

1.4. Đề quy

1.4.1. Khái niệm đề quy

1.4.2. Giải thuật đề quy và thủ tục đề quy

- 1.4.3. Thiết kế giải thuật đệ quy
- 1.4.4. Hiệu lực của đệ quy
- 1.5. Phân tích và thiết kế giải thuật.
 - 1.5.1. Mô-đun hoá và việc giải quyết bài toán
 - 1.5.2. Phương pháp tinh chỉnh từng bước
- 1.6. Bài tập chương I

Chương 2. CẤU TRÚC DỮ LIỆU TUYẾN TÍNH VÀ CÁCH CÀI ĐẶT

(Lý thuyết: 5 tiết, Thảo luận: 1 tiết, Bài tập: 1 tiết, Thực hành: 6 tiết, Tự học: 26 giờ)

- 2.1. Phân loại các cấu trúc dữ liệu
- 2.2. Mảng
 - 2.2.1. Khái niệm
 - 2.2.2. Cấu trúc lưu trữ của mảng
- 2.3. Danh sách
 - 2.3.1. Danh sách móc nối đơn
 - 2.3.2. Một số dạng danh sách móc nối khác
- 2.4. Ngăn xếp (Stack)
 - 2.4.1. Khái niệm và các phép toán
 - 2.4.2. Ví dụ về ứng dụng của Stack
- 2.5. Hàng đợi (Queue)
 - 2.5.1. Khái niệm
 - 2.5.2. Một số phép toán
 - 2.5.3. Ứng dụng của hàng đợi
- 2.6. Bài tập chương II

Chương 3. CÂY VÀ CÂY TÌM KIẾM NHỊ PHÂN

(Lý thuyết: 4 tiết, Thảo luận: 1 tiết, Bài tập: 1 tiết, Thực hành: 4 tiết, Tự học: 22 giờ)

- 3.1. Một số khái niệm
- 3.2. Cây nhị phân
 - 3.2.1. Định nghĩa và tính chất.
 - 3.2.2. Biểu diễn cây nhị phân.
 - 3.2.3. Phép duyệt cây nhị phân.
 - 3.2.4. Cài đặt cây nhị phân
- 3.3. Cây tìm kiếm nhị phân
 - 3.3.1. Định nghĩa cây nhị phân tìm kiếm.
 - 3.3.2. Giải thuật tìm kiếm.
 - 3.3.3. Loại bỏ trên cây nhị phân tìm kiếm.
- 3.4. Cây biểu thức
- 3.5. Cây tổng quát
 - 3.5.1. Biểu diễn cây tổng quát
 - 3.5.2. Phép duyệt cây tổng quát
- 3.6. Bài tập chương III

Chương 4. SẮP XẾP VÀ TÌM KIẾM

(Lý thuyết: 4 tiết, Thảo luận: 1 tiết, Bài tập: 1 tiết, Thực hành: 3 tiết, Tự học: 18 giờ)

- 4.1. Bài toán sắp xếp
- 4.2. Một số phương pháp sắp xếp cơ bản
 - 4.2.1. Sắp xếp kiểu lựa chọn
 - 4.2.2. Sắp xếp kiểu thêm dần chèn
 - 4.2.3. Sắp xếp kiểu đổi chỗ
- 4.4. Sắp xếp nhanh
 - 4.4.1. Giới thiệu phương pháp.
 - 4.4.2. Ví dụ và giải thuật.
- 4.3. Sắp xếp bằng phương pháp vun đống
 - 4.3.1. Giới thiệu phương pháp
 - 4.3.2. Ví dụ và giải thuật.
- 4.5. Sắp xếp kiểu hoà nhập (Merge Sort)
 - 4.5.1. Phép hoà nhập hai đường
 - 4.5.2. Sắp xếp kiểu hoà nhập hai đường trực tiếp
- 4.6. Bài toán tìm kiếm
- 4.7. Một số phương pháp tìm kiếm
 - 4.7.1. Tìm kiếm tuần tự
 - 4.7.2. Tìm kiếm nhị phân
- 4.8. Bài tập chương IV

Chương 5. ĐỒ THỊ

(Lý thuyết: 3 tiết, Bài tập: 1 tiết, Thực hành: 1 tiết, Tự học: 10 giờ)

- 5.1. Khái niệm và biểu diễn đồ thị
- 5.2. Duyệt và tìm kiếm trên đồ thị
 - 5.2.1. Tìm kiếm theo chiều sâu.
 - 5.2.2. Tìm kiếm theo chiều rộng.
- 5.3. Một số bài toán trên đồ thị
- 5.4. Bài tập chương 5

6. Học liệu

- Tài liệu chính

- [1a]. Bài giảng Cấu trúc dữ liệu và giải thuật – Bùi Thị Thu Hoài (Tài liệu lưu hành nội bộ)
- [2a]. Đỗ Xuân Lôi, Cấu trúc dữ liệu và giải thuật, Nhà xuất bản Đại học Quốc gia Hà Nội.
- [3a]. Nguyễn Đình Hoà, Cấu trúc dữ liệu và thuật giải. Nhà xuất bản Đại học Quốc gia Hà Nội.

- Học liệu tham khảo:

- [1b]. Phạm Văn Ất, Kỹ thuật lập trình C. Nhà xuất bản thống kê.
- [3a]. Nguyễn Thị Tĩnh (Chủ biên), Cấu trúc dữ liệu và giải thuật. Nhà xuất bản Đại học Sư phạm.

MỤC LỤC

LỜI NÓI ĐẦU.....	1
Chương I. MỞ ĐẦU.....	2
1.1. Khái niệm về cấu trúc dữ liệu và giải thuật.....	2
1.1.1. Khái niệm bài toán.....	2
1.1.2. Khái niệm giải thuật.....	3
1.1.3. Khái niệm cấu trúc dữ liệu và mối quan hệ với giải thuật.....	4
1.2. Mô tả giải thuật.....	5
1.2.1. Phương pháp sơ đồ khối.....	5
1.2.2. Phương pháp liệt kê.....	6
1.2.3. Phương pháp giả mã.....	6
1.3. Độ phức tạp của giải thuật.....	7
1.3.1. Đánh giá thời gian thực hiện của thuật toán.....	7
1.3.2. Một số qui tắc xác định thời gian thuật toán.....	10
1.3.3. Phân tích một số thuật toán.....	12
1.4. Đệ quy.....	14
1.4.1. Khái niệm đệ quy.....	14
1.4.2. Giải thuật đệ quy và thủ tục đệ quy.....	14
1.4.3. Thiết kế giải thuật đệ quy.....	15
1.4.4. Hiệu lực của đệ quy.....	19
1.5. Phân tích và thiết kế giải thuật.....	20
1.5.1. Mô-đun hoá và việc giải quyết bài toán.....	20
1.5.2. Phương pháp tinh chỉnh từng bước.....	21
Bài tập chương I.....	23
Chương II. CẤU TRÚC DỮ LIỆU TUYẾN TÍNH.....	25
VÀ CÁCH CÀI ĐẶT.....	25
2.1. Phân loại các cấu trúc dữ liệu.....	25
2.2. Mảng (Array).....	27
2.2.1. Khái niệm.....	27
2.2.2. Cấu trúc lưu trữ của mảng.....	27
2.3. Danh sách.....	29
2.3.1. Danh sách móc nối đơn.....	29
2.3.2. Một số dạng danh sách móc nối khác.....	35
2.4. Ngăn xếp (Stack).....	42

2.4.1. Khái niệm và các phép toán	42
2.4.2. Ví dụ về ứng dụng của Stack	43
2.5. Hàng đợi (Queue)	45
2.5.1. Khái niệm	45
2.5.2. Một số phép toán	46
2.5.3. Ứng dụng của hàng đợi.....	47
Bài tập chương II.....	48
Chương III. CÂY VÀ CÂY TÌM KIẾM NHỊ PHÂN	54
3.1. Một số khái niệm	54
3.2. Cây nhị phân.....	55
3.2.1. Định nghĩa và tính chất.....	55
3.2.2. Biểu diễn cây nhị phân.....	56
3.2.3. Phép duyệt cây nhị phân	58
3.2.3. Cài đặt cây nhị phân	60
3.3. Cây nhị phân tìm kiếm	60
3.3.1. Định nghĩa	60
3.3.2. Giải thuật tìm kiếm	61
3.3.3. Phân tích đánh giá.....	64
3.3.4. Loại bỏ nút trên cây nhị phân tìm kiếm.....	65
3.4. Cây biểu thức.....	73
3.5. Cây tổng quát.....	74
3.5.1. Biểu diễn cây tổng quát	74
3.5.2. Phép duyệt cây tổng quát.....	76
Bài tập chương III.....	78
Chương IV. SẮP XẾP VÀ TÌM KIẾM.....	80
4.1. Bài toán sắp xếp.....	80
4.2. Một số phương pháp sắp xếp đơn giản	80
4.2.1. Sắp xếp kiểu lựa chọn (Slection Sort)	80
4.2.2. Sắp xếp kiểu thêm dần (Insert Sort)	82
4.2.3. Sắp xếp kiểu đổi chỗ (Exchange Sort).....	86
4.3. Sắp xếp nhanh.....	86
4.3.1. Giới thiệu phương pháp	86
4.3.2. Ví dụ và giải thuật	87
4.3.3. Nhận xét và đánh giá	90
4.4. Sắp xếp kiểu vun đống (Heap Sort).....	90
4.4.1. Giới thiệu phương pháp	90
4.4.2. Giải thuật.....	91

4.5. Sắp xếp kiểu hoà nhập (Merge Sort)	97
4.5.1. Phép hoà nhập hai đường.....	97
4.5.2. Sắp xếp kiểu hoà nhập hai đường trực tiếp.....	98
4.6. Bài toán tìm kiếm.....	99
4.7. Một số phương pháp tìm kiếm	100
4.7.1. Tìm kiếm tuần tự	100
4.7.2. Tìm kiếm nhị phân.....	100
Bài tập chương IV	102
Chương V. ĐỒ THỊ	103
5.1. Khái niệm và biểu diễn đồ thị	103
5.1.1. Khái niệm.....	103
5.1.2. Biểu diễn đồ thị	104
5.2. Duyệt và tìm kiếm trên đồ thị.....	105
5.2.1. Tìm kiếm theo chiều sâu	105
5.2.2. Tìm kiếm theo chiều rộng.....	106
5.3. Một số bài toán trên đồ thị	109
5.3.1. Bài toán bao đóng truyền ứng.....	109
5.3.2. Bài toán một nguồn, mọi đích	110
5.3.3. Cây khung tối thiểu.....	112
Bài tập chương V.....	115
TÀI LIỆU THAM KHẢO.....	118