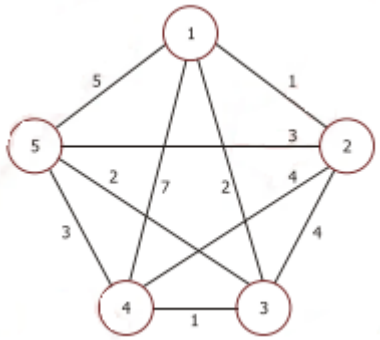
BÀI 1: GIỚI THIỆU CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT



Mục tiêu
Sau khi học bài này, các bạn có thể: • Mô tả đúng đối tượng và các phương pháp nghiên cứu môn học. • Trình bày đúng khái niệm về cấu trúc dữ liệu. • Liệt kê và xác định đúng các kiểu dữ liệu cơ bản như kiểu dữ liệu số nguyên, số thực, logic, ký tự... • Mô tả đúng các kiểu dữ liệu trừu tượng như cấu trúc dữ liệu bản ghi, kiểu dữ liệu mảng... một cách chính xác. • Mô tả ngôn ngữ diễn đạt giải thuật đúng và vận dụng nó để diễn đạt các thuật toán. • Trình bày thuật toán đệ quy, thuật toán quay lui một cách chính xác.

Nội dung
• Khái niệm • Kiểu dữ liệu • Ngôn ngữ diễn đạt giải thuật • Đệ quy

Thời lượng học
6 tiết

Chương trình máy tính là một dãy các câu lệnh để xử lý thông tin và đem lại kết quả mong muốn cho người sử dụng. Điều đó có nghĩa đối tượng xử lý của chương trình là thông tin. Thông thường đó là các thông tin về đối tượng được lấy từ trong thực tế hoặc được suy diễn từ thực tế. Tuy nhiên trong đời sống thực mỗi đối tượng thường mang rất nhiều thông tin. Ví dụ thông tin về một con người, về mặt vật lý có thể có như chiều cao, cân nặng, giới tính, nhóm máu, màu da... Về mặt xã hội có thể có trình độ học vấn, ngoại ngữ, chỉ số IQ, khả năng ngoại giao,... Vì vậy rất khó có thể mô tả một cách cụ thể chi tiết về một đối tượng vào chương trình. Thay vào đó, chỉ các thông tin về đối tượng có liên quan, tác động tới mục tiêu xử lý mới cần được biểu diễn trong chương trình. Ví dụ, để xây dựng một phần mềm quản lý nhân sự, các mô tả một nhân viên có thể sử dụng các thông tin như: quê quán, năm sinh, nơi ở, trình độ học vấn, cấp bậc tay nghề, bậc lương, vị trí...

Mặt khác, các thông tin lấy từ thực tế thường là các thông tin trừu tượng và không thể biểu diễn một cách trực tiếp trong máy tính. Vì vậy với mỗi thông tin cần biểu diễn bằng một kiểu dữ liệu phù hợp trên máy.

1.1. Khái niệm

Cấu trúc dữ liệu là cách thức tổ chức sắp xếp dữ liệu để biểu diễn thông tin trên máy tính. Dữ liệu có thể được lưu trữ tại bộ nhớ trong để trực tiếp xử lý, hoặc được lưu trong các thiết bị lưu trữ ngoài ví dụ như dưới dạng tệp dữ liệu trên ổ cứng.

Khi thông tin đã được biểu diễn hay được lưu trữ trên máy, công việc còn lại của người lập trình là phải xử lý các thông tin đó, hay nói một cách khác là thao tác các dữ liệu đó nhằm giải quyết các vấn đề đã đặt ra. Khi số lượng thông tin ít, hay lượng dữ liệu cần xử lý chỉ là một vài cá thể đơn lẻ, việc xử lý các dữ liệu đó trở nên khá đơn giản, và mức độ phức tạp của việc xử lý nói chung không ảnh hưởng nhiều đến hiệu suất của chương trình. Vấn đề trở nên đáng quan tâm khi lượng dữ liệu cần xử lý lớn, thời gian xử lý các dữ liệu đó đóng vai trò quyết định tới hiệu suất của chương trình.

Thuật ngữ giải thuật được sử dụng rộng rãi trong khoa học máy tính để mô tả các phương pháp giải quyết vấn đề phù hợp cho việc thực thi trên máy tính. Trong khuôn khổ của giáo trình này, giải thuật được hiểu là các biện pháp, cách thức để thao tác dữ liệu.

Trên thực tế tồn tại rất nhiều giải thuật và kiểu cấu trúc dữ liệu. Quyết định sử dụng cấu trúc dữ liệu và giải thuật nào cho một vấn đề nhất định phụ thuộc vào rất nhiều yếu tố và yêu cầu của chương trình cần triển khai.

Ví dụ như khả năng lưu trữ dữ liệu của máy. Trong trường hợp bộ nhớ trong của máy đủ lớn, toàn bộ dữ liệu có thể được lưu trữ trong bộ nhớ trong, thời gian truy cập dữ liệu sẽ nhanh hơn. Ngoài ra trong các máy tính hiện đại còn có thêm một loại bộ nhớ truy cập nhanh, là vùng đệm giữa bộ vi xử lý và bộ nhớ trong. Dữ liệu được đọc từ bộ nhớ trong vào bộ nhớ truy cập nhanh theo từng khối nhằm tăng tốc độ truy cập dữ liệu. Vì vậy thứ tự xử lý dữ liệu cũng đóng một vai trò nhất định. Nói chung dữ liệu thường được ưu tiên xử lý liên tiếp, như vậy tính sẵn sàng của dữ liệu trong bộ nhớ truy cập nhanh sẽ được nâng cao.

Ngược lại, trong trường hợp bộ nhớ trong của máy không đủ lớn để lưu trữ toàn bộ dữ liệu, một thiết bị lưu trữ ngoài sẽ được sử dụng để lưu trữ dữ liệu. Dữ liệu chỉ được đọc vào bộ nhớ trong khi cần phải thao tác trên chúng. Thiết bị lưu trữ ngoài còn có

một tác dụng nữa là lưu trữ dữ liệu lâu dài, qua các phiên làm việc của máy vì như chúng ta đã biết, sau mỗi phiên làm việc của máy, toàn bộ dữ liệu được lưu ở bộ nhớ trong sẽ bị xóa và sẽ không được khôi phục trong phiên làm việc tiếp theo. Trường hợp thường gặp nhất là dữ liệu được ghi vào các tệp trên ổ cứng của máy tính.

Có những giải thuật phức tạp tỏ ra hết sức hiệu quả khi áp dụng trên các cấu trúc dữ liệu phức tạp và được thực hiện rất nhiều lần. Tuy nhiên khi áp dụng với một cấu trúc dữ liệu đơn giản hơn, hoặc khi số lần áp dụng ít hơn, giải thuật đó lại kém hiệu quả hơn một giải thuật đơn giản.

Trong khuôn khổ giáo trình này, chúng tôi giới thiệu cho học viên các cấu trúc dữ liệu và giải thuật cơ bản theo một trình tự nhất định để học viên dễ tiếp cận nhất. Nắm rõ các cấu trúc và giải thuật này sẽ giúp cho học viên tìm hiểu dễ dàng hơn các cấu trúc dữ liệu và giải thuật phức tạp, nâng cao hơn.

Khi nói đến cấu trúc dữ liệu ta quan tâm đến các vấn đề cơ bản sau:

- Cách cài đặt cấu trúc dữ liệu đó.
- Cách thực hiện các thao tác cơ bản với cấu trúc dữ liệu đó.
  - Tạo mới
  - Thêm 1 phần tử
  - Xóa 1 phần tử
  - Tìm 1 phần tử…

Còn khi nói đến giải thuật ta quan tâm đến các vấn đề cơ bản sau:

- Tư tưởng của giải thuật.
- Nội dung của thuật toán và cách cài đặt thuật toán đó.
- Đánh giá độ phức tạp về thời gian và độ phức tạp về bộ nhớ.

1.2. Kiểu dữ liệu

Như đã nhắc đến ở phần trên, thông tin từ trong thực tế được biểu diễn trong máy tính dưới dạng các kiểu dữ liệu. Tuy nhiên việc lựa chọn kiểu dữ liệu nào để biểu diễn thông tin cũng không hề đơn giản. Như chúng ta đã biết, máy tính biểu diễn dữ liệu dựa trên các số nhị phân (chỉ nhận hai giá trị 0 hoặc 1). Trường giá trị có thể biểu diễn là tập hợp các giá trị rời rạc, không thích hợp cho việc biểu diễn trực tiếp các giá trị trong thực tế thường là các giá trị liên tục. Vì vậy kiểu dữ liệu được dùng để biểu diễn thông tin cần phải sát với kiểu giá trị của các thông tin đó trong thực tế.

Một yếu tố nữa cùng cần được xem xét khi lựa chọn kiểu dữ liệu là các thao tác sẽ được thực hiện trên các dữ liệu đó. Ví dụ để biểu diễn chiều dài (được tính bằng mét), trong thực tế các giá trị chiều dài thường là các giá trị thực (ví dụ 102,35m). Vì vậy kiểu dữ liệu gần nhất để biểu diễn chiều dài trong máy tính chính là số thực dấu phẩy động (floating – point number), hoặc trong một số trường hợp có thể là số thực dấu phẩy tĩnh. Nhưng khi giá trị chiều dài tương đối lớn, ví dụ vài chục km hoặc vài trăm km, chúng ta có thể sử dụng kiểu dữ liệu số nguyên để biểu diễn chiều dài theo m hoặc theo cm. Mặc dù độ chính xác của dữ liệu được biểu diễn sẽ giảm đi so với việc sử dụng kiểu dữ liệu số thực dấu phẩy động, nhưng lại giúp nâng cao tốc độ tính toán vì các thao tác số nguyên trên máy tính thường nhanh hơn nhiều so với với các thao

tác trên số thực. Quyết định sử dụng kiểu dữ liệu số thực hay số nguyên còn phụ thuộc vào độ chính xác yêu cầu của chương trình.

Trong khuôn khổ thiết kế một chương trình, các kiểu dữ liệu được sử dụng không nên ở mức độ quá thấp phụ thuộc vào một cấu trúc máy tính cụ thể hay một ngôn ngữ lập trình vì như thế sẽ làm giảm độ linh động khi triển khai chương trình. Vì vậy các kiểu dữ liệu trừu tượng thường được sử dụng để thiết kế chương trình. Việc sử dụng các kiểu dữ liệu trừu tượng sẽ giúp cho người lập trình không phải quá quan tâm đến các cách thức biểu diễn cụ thể các dữ liệu đó trên máy tính.

Mặt khác, mức độ trừu tượng của dữ liệu cũng không nên quá cao vì như thế sẽ gây khó khăn cho việc triển khai chương trình. Kiểu dữ liệu trừu tượng sử dụng càng sát với thực tế càng giúp cho người triển khai dễ dàng lựa chọn kiểu dữ liệu cụ thể để biểu diễn trên máy tính. Ví dụ, không nên sử dụng một kiểu dữ liệu "tọa độ" để biểu diễn vị trí của một đối tượng trong thực tế, vì như thế người triển khai chương trình sẽ không biết dùng kiểu dữ liệu cụ thể nào. Đó có thể là kiểu tọa độ đề các, hay kiểu tọa độ trục? Mỗi trường của tọa độ sẽ được biểu diễn bằng một số nguyên hay một số thực?

Phần này sẽ giới thiệu các kiểu dữ liệu trừu tượng thường được sử dụng trong việc thiết kế chương trình, hay thiết kế giải thuật.

1.2.1. Các kiểu dữ liệu cơ bản

Kiểu dữ liệu cơ bản là các kiểu dữ liệu có sẵn trên hầu hết các máy tính, và được hỗ trợ trong hầu hết các ngôn ngữ lập trình. Chúng bao gồm kiểu dữ liệu số nguyên (INTEGER), kiểu dữ liệu số thực (REAL), kiểu dữ liệu giá trị logic (BOOLEAN), và kiểu dữ liệu ký tự (CHAR). Ngoài ra còn một kiểu dữ liệu nữa hay được sử dụng trong các thuật toán là kiểu dữ liệu con trỏ (POINTER).

1.2.1.1. Kiểu dữ liệu số nguyên (INTEGER)

Kiểu dữ liệu số nguyên biểu diễn các giá trị nguyên trong thực tế. Trên thực tế có hai kiểu dữ liệu số nguyên là số nguyên có dấu (biểu diễn cả giá trị âm và giá trị dương) và số nguyên không dấu (chỉ biểu diễn giá trị lớn hơn hoặc bằng 0). Nhưng khi nhắc đến số nguyên thường được ngầm định là kiểu giá trị số nguyên có dấu. Trong trường hợp muốn sử dụng kiểu dữ liệu số nguyên không dấu cần được khai báo rõ ràng.

Trường giá trị của kiểu dữ liệu số nguyên phụ thuộc vào số lượng bit được sử dụng để biểu diễn. Giả thiết kiểu dữ liệu số nguyên được biểu diễn bằng n bit trong hệ thống biểu diễn phần bù 2, khi đó trường giá trị của kiểu dữ liệu số nguyên sẽ nằm trong khoảng từ $-2(n-1)$ cho đến $2(n-1)$. Thông thường số bit để biểu diễn kiểu dữ liệu trên máy tính là bội số của 8, n = 8, 16, 32, 64, 128. Mỗi giá trị cụ thể của n tương ứng với một kiểu dữ liệu con trong máy tính. Việc quyết định sử dụng kiểu dữ liệu con nào để biểu diễn số nguyên phụ thuộc vào trường giá trị của thông tin trong thực tế.

Các thao tác có thể thực hiện trên kiểu dữ liệu nguyên bao gồm cộng (+), trừ (–), nhân (*), chia (/, DIV), và phần dư (MOD). Thao tác chia / để biểu diễn kết quả dưới dạng số thực, còn thao tác chia DIV (hay còn gọi là phần thương theo thuật ngữ số học) để biểu diễn kết quả dưới dạng số nguyên. Một lưu ý khi thực hiện thao tác phần thương (DIV) đối với các giá trị âm là phần thương luôn được làm tròn xuống.

Ví dụ 1.1.

1; –1; 2; –2 là các số nguyên

31 / 5 = 6.2

31 DIV 5 = 6 31 MOD 5 = 1 vì 31 chia cho 5 được thương là 6 dư 1, phép DIV là phép chia lấy thương nên 31 DIV 5 = 6, phép MOD là phép chia lấy dư nên 31 MOD 5 = 1.

–31 DIV 5 = –7 –31 MOD 5 = 4 vì –31 chia 5 được thương là 7 dư 4 (–31 = 7 * –5 + 4)

1.2.1.2. Kiểu dữ liệu số thực (REAL)

Kiểu dữ liệu số thực được dùng để biểu diễn các giá trị thực. Khác với kiểu dữ liệu số nguyên, kiểu dữ liệu số thực không biểu diễn được chính xác các giá trị thực trong thực tế. Lý do là vì kiểu dữ liệu số thực cũng được triển khai trên máy tính dựa trên biểu diễn nhị phân, vì thế trường giá trị của kiểu dữ liệu số thực là một tập hợp các giá trị rời rạc. Trong khi các giá trị thực trong thực tế là liên tục. Vì thế khi biểu diễn các giá trị thực bằng kiểu dữ liệu số thực thường gây ra các sai số làm tròn (round – off error). Hệ quả là kết quả của các thao tác trên kiểu dữ liệu số thực cũng thường không chính xác. Chúng ta không đi sâu vào vấn đề này vì đó là một lĩnh vực nghiên cứu rộng trong khoa học máy tính.

Các thao tác cơ bản trên kiểu dữ liệu số thực bao gồm cộng (+), trừ (–), nhân (*), và chia (/). Ngoài ra còn một thao tác hay được sử dụng nữa là thao tác chuyển đổi từ kiểu dữ liệu số thực sang kiểu dữ liệu số nguyên ([]), thao tác này cho phép lấy về phần trị nguyên của một số thực. Vì vậy làm tròn một số thực x tương đương với [x + 0.5].

1.2.1.3. Kiểu dữ liệu logic (BOOLEAN)

Kiểu dữ liệu logic được dùng để biểu diễn các giá trị logic, bao gồm hai giá trị là đúng (TRUE) và sai (FALSE). Kiểu dữ liệu này được sử dụng để biểu diễn kết quả so sánh giữa các giá trị thuộc kiểu dữ liệu số (nguyên hoặc thực). Ví dụ phép so sánh 5 = 9 sẽ đem lại kết quả sai (FALSE), còn phép so sánh 3.5 < 9.0 đem lại kết quả đúng (TRUE).

Các thao tác trên kiểu dữ liệu logic bao gồm phép nhân (AND), phép cộng (OR), và phép đảo (NOT). Kết quả của các phép này được cho trong bảng dưới đây:

P	Q	P AND Q	P OR Q	NOT(P)
TRUE	TRUE(1)	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE

P	Q	P AND Q	P OR Q	NOT(P)
1	1	1	1	0
1	0	0	1	0
0	1	0	1	1
0	0	1	0	1

1.2.1.4. Kiểu dữ liệu ký tự (CHAR)

Kiểu dữ liệu ký tự được dùng để biểu diễn tập hợp các ký tự. Trên thực tế có rất nhiều bộ ký tự khác nhau phụ thuộc vào từng ngôn ngữ cụ thể. Vì vậy ở đây, nếu không được nhắc đến một cách rõ ràng thì kiểu dữ liệu ký tự bao gồm các ký tự la tinh, các ký tự thường từ 'a' đến 'z' và các ký tự in hoa từ 'A' đến 'Z', và các chữ số từ '0' đến '9'. Ngoài ra còn có hai ký tự đặc biệt nữa là dấu cách ' ' và dấu kết thúc dòng.

Kiểu dữ liệu ký tự cũng được sắp xếp theo thứ tự theo quy tắc so sánh sau:

'A' < 'B' < ... < 'Z' < 'a' < 'b' < ... < 'z' < '0' < '1' < ... '9'

Lưu ý khi viết các hằng số kiểu giá trị ký tự, cần phải đặt giữa hai dấu nháy " để phân biệt với hằng số kiểu giá trị số, và phân biệt với các câu lệnh.

1.2.1.5. Kiểu dữ liệu con trỏ (POINTER)

Kiểu dữ liệu con trỏ được dùng để lưu các con trỏ đến bất kỳ một kiểu dữ liệu nào khác. Trên thực tế giá trị của "con trỏ" chính là địa chỉ đến một vùng bộ nhớ nhất định. Kiểu của con trỏ chính là kiểu dữ liệu tại vùng bộ nhớ đó.

Ví dụ một con trỏ kiểu số nguyên trỏ đến một vùng bộ nhớ lưu trữ một số nguyên nào đó. Trong khi một con trỏ số thực trỏ đến một vùng bộ nhớ khác lưu trữ một số thực. Hai thao tác cơ bản liên quan đến con trỏ là lấy dữ liệu được lưu trữ tại vị trí con trỏ (VAL), và lấy địa chỉ của một dữ liệu trong bộ nhớ để ghi lại trong một biến con trỏ (ADR).

1.2.2. Kiểu dữ liệu trừu tượng

Nói chung các đối tượng trong thực tế thường mang nhiều thông tin và đòi hỏi phải sử dụng đồng thời nhiều kiểu dữ liệu cơ bản nêu trên để biểu diễn. Do đó để tạo điều kiện thuận lợi cho người lập trình thiết kế dữ liệu, hầu hết các ngôn ngữ lập trình đều cho phép người dùng định nghĩa các kiểu dữ liệu riêng của mình dựa trên các kiểu dữ liệu cơ bản sẵn có. Những kiểu dữ liệu mới được định nghĩa đó được gọi là các kiểu dữ liệu trừu tượng, để phân biệt với các kiểu dữ liệu cơ bản. Phần này sẽ giới thiệu cho học viên một số kiểu dữ liệu trừu tượng thường dùng nhất trong quá trình thiết kế chương trình.

1.2.2.1. Cấu trúc dữ liệu bản ghi (RECORD)

Cách thức đơn giản và thông thường nhất để tổ chức dữ liệu là nhóm tất cả các phần tử thuộc bất kỳ một kiểu dữ liệu nào vào một tổ hợp. Các phần tử có thể thuộc một kiểu dữ liệu cơ bản, hoặc có thể là một kiểu dữ liệu trừu tượng khác.

Các ví dụ đơn giản về cấu trúc bản ghi như kiểu dữ liệu số phức bao gồm phần thực và phần ảo đều thuộc kiểu dữ liệu số thực. Tương tự là kiểu dữ liệu tọa độ, có thể bao gồm 2, 3 hoặc nhiều tọa độ khác nhau tùy thuộc đó là tọa độ 2 chiều, 3 chiều hay nhiều chiều. Hoặc kiểu dữ liệu để mô tả một cá nhân bao gồm các thông tin đơn giản như họ, tên, ngày sinh, giới tính.

Thuật ngữ bản ghi (record) xuất phát từ lĩnh vực xử lý dữ liệu. Các kiểu dữ liệu tổng hợp thường xuất hiện trong các tệp hoặc các ngân hàng dữ liệu để ghi lại các đặc tính,

thông tin liên quan về một cá nhân hoặc một đối tượng nào đó. Thuật ngữ bản ghi do đó được sử dụng rộng rãi để mô tả một tập hợp dữ liệu hoặc thông tin.

Các dữ liệu lưu trong bản ghi được gọi là trường dữ liệu (field) của bản ghi. Để định nghĩa một cấu trúc bản ghi mới, chúng ta dùng quy tắc dưới đây:

Định nghĩa kiểu

```
struct ten_cua_cau_truc
{
        kiểu_dữ_liệu_1 tên_trường_dữ_liệu_1;
        kiểu_dữ_liệu_2 tên_trường_dữ_liệu_2;
         ……..
};
```

Để truy cập một trường dữ liệu của bản ghi, chúng ta viết như sau:

TÊN_CẤU_TRÚC.TÊN_TRƯỜNG_DỮ_LIỆU

Ví dụ 1.2. Kiểu dữ liệu số phức:

Định nghĩa kiểu dữ liệu: struct

so_phuc

{

float **phan_thuc;**

float **phan_ao;**

};

Để truy cập phần thực và phần ảo của một phần tử x thuộc kiểu dữ liệu so_phuc, chúng ta viết như sau: x.phan_thuc, x.phan_ao

1.2.2.2. Cấu trúc dữ liệu mảng (ARRAY)

Khác với cấu trúc dữ liệu bản ghi dùng để tập hợp các kiểu dữ liệu khác nhau, trong nhiều trường hợp, người dùng muốn lưu trữ một số lượng nhất định các dữ liệu thuộc cùng một kiểu dữ liệu. Đó chính là mục đích của cấu trúc dữ liệu mảng.

Cấu trúc dữ liệu mảng được sử dụng để lưu trữ liên tiếp các phần tử thuộc cùng một kiểu dữ liệu. Mỗi phần tử của mảng được xác định bởi vị trí của nó trong mảng.

Trên thực tế mỗi ngôn ngữ lập trình có một kiểu riêng để đánh số thứ tự mảng. Ví dụ FORTRAN đánh số bắt đầu từ 1, trong khi C và Java đánh số bắt đầu từ 0.

Cấu trúc mảng có lẽ là cấu trúc được sử dụng nhiều nhất trong các chương trình. Cấu trúc mảng còn được gọi là cấu trúc đồng nhất vì các phần tử của mảng bắt buộc phải cùng một kiểu dữ liệu. Ngoài ra nó còn được gọi cấu trúc dữ liệu truy nhập ngẫu nhiên, vì mỗi phần tử của mảng có thể được truy xuất một cách dễ dàng nhanh chóng dựa vào vị trí của nó trong mảng. Vị trí của mỗi phần tử được gọi là chỉ số (index) của phần tử trong mảng.

Cách thao tác cấu trúc mảng nói chung là thao tác trên từng phần tử của mảng. Vì vậy thao tác cơ bản nhất đối với cấu trúc mảng là thao tác truy xuất ([???]). Ví dụ, nếu x là một biến thuộc kiểu cấu trúc mảng, để truy xuất phần tử thứ 5 trong mảng đó chúng ta sử dụng: x[5].

Một trường hợp đặc biệt của cấu trúc dữ liệu mảng chính là kiểu dữ liệu xâu ký tự (STRING). Mỗi một xâu ký tự (STRING) là một mảng của kiểu dữ liệu ký tự (CHAR), và luôn được kết thúc bằng một ký tự đặc biệt. Thông thường ký tự kết thúc dòng trong các ngôn ngữ lập trình có giá trị mã hóa bằng 0. Các thao tác trên xâu ký tự có thể kể ra như hiển thị xâu ký tự, so sánh xâu ký tự, ghép nối xâu ký tự, ... Mỗi thao tác trên xâu ký tự đều dựa trên các thao tác mảng.

Ví dụ để hiển thị xâu ký tự, cần phải duyệt qua tất cả phần tử trong mảng, và in lần lượt từng phần tử (ký tự) đó cho đến khi hết xâu.

Mảng 2 chiều

Cấu trúc mảng còn thường được sử dụng để mã hóa các đối tượng thuộc kiểu ma trận trong toán học. Ma trận một chiều, hay vector được lưu trữ bằng mảng một chiều như đã trình bày ở trên. Để biểu diễn ma trận hai chiều chúng ta phải sử dụng mảng hai chiều. Các phần tử trong mảng hai chiều được sắp xếp theo hàng và cột. Các phần tử thuộc cùng một hàng được sắp xiếp liên tiếp nhau. Để truy xuất đến một phần tử trong mảng hai chiều, ta phải sử dụng hai chỉ số: chỉ số cột và chỉ số hàng. Ví dụ, để tham chiếu đến phần tử thuộc hàng thứ 3, cột thứ 5 của mảng hai chiều a, ta sử dụng cú pháp: a[3][5]. Vì vậy để duyệt mảng hai chiều ta phải duyệt lần lượt từng hàng và từng phần tử trong mảng. Tương tự ta có thể mở rộng ra khái niệm mảng nhiều chiều. Tuy nhiên trong thực tế ít khi phải sử dụng đến mảng nhiều chiều. Hơn nữa việc tổ chức mảng nhiều chiều trong bộ nhớ sẽ phức tạp hơn nhiều so với mảng một chiều và mảng hai chiều.

1.3. Ngôn ngữ diễn đạt giải thuật

Cũng giống như với cấu trúc dữ liệu, với mục đích thiết kế chương trình không phụ thuộc vào một máy tính hay một ngôn ngữ nhất định, các giải thuật cũng phải được diễn đạt hay biểu diễn bằng một ngôn ngữ chung. Ngôn ngữ đó vừa phải mang tính trừu tượng giúp cho người dùng không cần quan tâm tới những đặc tả của máy tính hay các câu lệnh cụ thể của bất kỳ ngôn ngữ nào. Hơn nữa, ngôn ngữ diễn đạt giải thuật càng gần với ngôn ngữ tự nhiên thì càng dễ cho học viên nắm bắt các giải thuật được viết. Nhưng ngôn ngữ diễn đạt giải thuật cũng phải càng gần với các ngôn ngữ lập trình hiện có càng tốt để giúp cho người triển khai phần mềm dễ dàng chuyển từ thiết kế của chương trình sang một triển khai trên một ngôn ngữ lập trình cụ thể được lựa chọn.

Nói chung không có một quy chuẩn nhất định cho ngôn ngữ diễn đạt phần mềm. Vì thế tồn tại rất nhiều phiên bản của ngôn ngữ diễn đạt phần mềm. Có những phiên bản được xây dựng ít nhiều dựa trên một ngôn ngữ lập trình cụ thể mà nhiều người biết tới. Có thể kể ở đây như ngôn ngữ lập trình Pascal hay ngôn ngữ lập trình C.

Ngôn ngữ diễn đạt giải thuật còn phụ thuộc vào ngôn ngữ tự nhiên, hay ngôn ngữ giao tiếp chung giữa người viết và người sử dụng. Các ngôn ngữ diện đạt giải thuật thường sử dụng các thuật ngữ tiếng Anh với hai lý do:

- Câu lệnh của các ngôn ngữ lập trình thường dựa trên các thuật ngữ tiếng Anh.
- Phần lớn các tài liệu về công nghệ thông tin được viết bằng tiếng Anh, trong đó bao gồm cả các tài liệu về cấu trúc dữ liệu và giải thuật.

Việc lựa chọn sử dụng các thuật ngữ tiếng Anh có ưu điểm là một mặt giúp cho học viên làm quen dần với các thuật ngữ công nghệ thông tin tiếng Anh, tiện cho việc đọc các tài liệu cũng như các giải thuật được diễn tả bằng tiếng Anh. Mặt khác những học viên đã có kiến thức về ngôn ngữ lập trình sẽ cảm thấy ngôn ngữ diễn đạt giải thuật gần với ngôn ngữ lập trình hơn.

Mục đích của phần này chính là giới thiệu cho học viên ngôn ngữ diễn đạt giải thuật sẽ được sử dụng để mô tả các giải thuật được nghiên cứu trong các phần sau.

1.3.1. Cấu trúc dữ liệu giải thuật được viết bằng ngôn ngữ diễn đạt giải thuật

Một giải thuật được viết bao gồm hai phần:

- Phần định nghĩa, hay phần đầu giải thuật. Phần này mô tả khái quát giải thuật bao gồm dữ liệu đầu vào, điều kiện áp dụng giải thuật, và kết quả trả về mong muốn của việc thực hiện giải thuật.
- Phần triển khai, hay phần thân của giải thuật. Phần này bao gồm việc khai báo các biến dữ liệu sẽ được sử dụng trong giải thuật, và mô tả các lệnh cụ thể cần được thực hiện để triển khai giải thuật.

Phần định nghĩa giải thuật có cấu trúc như sau:

```
Algorithm name
Input…..
Pre-condition
Post-condition
Output
```

Ví dụ 1.3. Để diễn tả giải thuật tính giai thừa, phần định nghĩa sẽ được viết như sau:

Algorithm factorial

Input: n an integer

Precondition: n >= 0

Post-condition: calculate the factorial of n

Phần triển khai giải thuật bao gồm hai phần nhỏ là khai báo dữ liệu và khối các lệnh cụ thể để thực hiện giải thuật được đặt giữa hai dấu móc lệnh { và } vậy về mặt tổng quan, cấu trúc một giải thuật được viết sẽ như sau:

```
Algorithm  name
Input: ...
Pre-condition
Post-condition
Declarations
{
    List of operations
}
```

Phần tiếp theo của bài này sẽ lần lượt giới thiệu cách khai báo biến, và các kiểu câu lệnh trong ngôn ngữ diễn đạt giải thuật.

1.3.2. Khai báo biến

Để khai báo một biến (Variable) để sử dụng trong giải thuật, chúng ta dùng cấu trúc:

TYPE variable;

Ví dụ: int n;

Ví dụ 1.4. Các ví dụ về khai báo biến:

int n; // Kiểu số nguyên **float** x; // Kiểu số thực **char** c; // Kiểu ký tự

Việc khai báo một biến thuộc kiểu dữ liệu bản ghi cũng được thực hiện theo quy tắc trên. Tuy nhiên cần lưu ý là cần phải định nghĩa kiểu dữ liệu bản ghi (như đã nêu trong phần cấu trúc bản ghi) trước khi sử dụng kiểu dữ liệu bản ghi. Phần khai báo kiểu dữ liệu được đặt bên trên phần định nghĩa giải thuật.

Khai báo một biến thuộc cấu trúc mảng

TYPE variable[size];

Ví dụ: **int** arr [10];

Ví dụ 1.5.

a : MẢNG 25 phần tử thuộc KIỂU SỐ NGUYÊN	**int** arr [25];

Kích thước của một mảng hai chiều được khai báo theo cú pháp: số hàng x số cột. Ví dụ để khai báo một mảng có 10 hàng và 25 cột, ta sử dụng

A : MẢNG 10x25 phần tử thuộc KIỂU SỐ NGUYÊN	**int** arr [10][25];

Lưu ý: để giải thuật được ngắn gọn, các biến có cùng kiểu dữ liệu có thể được khai báo đồng thời. Ví dụ để khai báo 3 số nguyên a, b, c ta có thể dùng:

int a,b,c; //khai báo 3 số nguyên a, b, c.

int a,b,c[10]; //khai báo 2 số nguyên a, b và mảng số nguyên c có 10 phần tử.

1.3.3. Câu lệnh

1.3.3.1. Câu lệnh gán

Câu lệnh gán dùng để gán giá trị cho một biến. Giá trị có thể là một hằng số hoặc một biểu thức có cùng kiểu giá trị với biến. Cấu trúc của một câu lệnh gán:

variable = constant/expression; Ví dụ: n = 5;

Ví dụ 1.6. Phép gán giá trị cho một biến kiểu số nguyên:

```
int a,b,c;....
b =5;
c = 8;
a = b + c;
```

Trong đoạn lệnh trên, a, b, c là 3 biến thuộc kiểu số nguyên. b và c được gán giá trị bằng các hằng số nguyên, còn a được gán giá trị bằng biểu thức giữa b và c. Sau khi thực hiện đoạn lệnh này, các biến sẽ mang giá trị: b = 5, c = 8, a = 13.

1.3.3.2. Câu lệnh điều kiện NẾU THÌ (IF THEN ELSE)

Câu lệnh điều kiện được sử dụng để quản lý logic của thuật toán. Cụ thể, với câu lệnh điều kiện, một lệnh hoặc đoạn lệnh chỉ được thực hiện trong trường hợp một hoặc nhiều điều kiện được thỏa mãn. Trong trường hợp ngược lại, một lệnh / đoạn lệnh thay thế sẽ được thực hiện.

Cấu trúc của câu lệnh điều kiện:

```
if(condition)
{
  list of operation…
}
else
{
    list of operation….
}
```

Đoạn lệnh else có thể có hoặc không. Trong trường hợp không có đoạn lệnh else, cấu trúc câu lệnh điều kiện được rút gọn:

```
if(condition)
{
  list of Operation…
}
```

Điều kiện của câu lệnh điều kiện bắt buộc phải thuộc kiểu dữ liệu logic. Thông thường đó là kết quả của các phép toán so sánh, hoặc các phép toán logic.

Trong trường hợp có nhiều điều kiện cần phải kiểm tra, để tránh sử dụng nhiều câu lệnh điều kiện lồng nhau, chúng ta có thể sử dụng câu lệnh nhiều tầng. Ứng với mỗi điều kiện được thỏa mãn, thực hiện một đoạn lệnh tương ứng. Trong trường hợp không điều kiện nào được thỏa mãn, một đoạn lệnh thay thế sẽ được thực hiện. Cấu trúc câu lệnh điều kiện nhiều tầng như sau:

```
if(condition1)
{
  list of operation…
}
else
{
  if(condition 2)
  {
    list of operation…
  }
   ……..
    else
    {
      list of operation…
    }
}
```

1.3.3.3. Câu lệnh hiển thị (Display)

Câu lệnh hiển thị được dùng để hiển thị (in ra màn hình, ghi vào file ...) một xâu ký tự hằng số, hoặc để hiển thị giá trị của một biến. Cấu trúc của câu lệnh hiển thị như sau:

printf(a string / variable / expression);

Lưu ý, để phân biệt một xâu ký tự hằng số với các câu lệnh và biến số, xâu ký tự phải được đặt giữa hai dấu ngoặc kép " ".

Ví dụ 1.8.

```
printf("The value of x is: ");
printf(x);
```

Câu lệnh hiển thị thường được dùng với mục đích thông báo, nhắc nhở, hiển thị kết quả tính toán.

1.3.3.4. Câu lệnh vòng lặp xác định (For)

Câu lệnh lặp xác định được dùng để thực hiện lặp lại một đoạn lệnh với số lần lặp xác định trước. Câu lệnh lặp xác định thường sử dụng một biến chạy để kiểm soát số lần lặp. Biến chạy này thường thuộc kiểu số nguyên. Cấu trúc câu lệnh lặp xác định có thể được mô tả như sau:

```
for(i = start_value; i < end_value; i+ = step_value)
//step_value: bước nhảy
{
   list of operation
}
```

Giá trị của biến chạy được gán bằng start_value và tăng dần đến end_value. Với mỗi một giá trị của biến chạy thì đoạn lệnh sẽ được thực hiện một lần. Biến chạy có thể được sử dụng trong đoạn lệnh để thay đổi hiệu ứng của đoạn lệnh tùy theo mỗi giá trị của biến chạy. Lưu ý giá trị của biến chạy có thể bị thay đổi trong quá trình đoạn lệnh, trong trường hợp đó số lần lặp của vòng lặp sẽ thay đổi, và có thể trở thành không xác định. Người sử dụng được khuyến cáo không nên thay đổi giá trị của biến chạy trong vòng lặp xác định trừ khi đã biết được kết quả chính xác của việc thay đổi giá trị của biến chạy.

Câu lệnh lặp xác định đặc biệt được sử dụng với hầu hết các thao tác của mảng, vì chiều dài của mảng đã được xác định trước. Ví dụ đoạn lệnh sau cho phép hiển thị giá trị của một mảng các số thực:

```
int i;
float a[20];
….
for(i = 0; i < 19; i++)
      printf(a[i]);
```

Giá trị của biến chạy i sẽ thay đổi lần lượt từ 0, 1, 2,... đến 19. Với mỗi giá trị của i, chúng ta thực hiện in phần tử thứ i của mảng a. Vì vậy sau khi kết thúc vòng lặp, toàn bộ 20 phần tử của mảng a sẽ được hiển thị.

Về mặt ngầm định, giá trị của biến chạy sẽ được tăng lên 1 sau mỗi lần lặp. Tuy nhiên trong một số trường hợp người dùng không muốn sử dụng tất cả các giá trị của biến chạy, ví dụ như người dùng chỉ muốn hiển thị các phần tử ở vị trí lẻ của mảng. Trong trường hợp đó ta có thể thay đổi cách ứng xử của vòng lặp bằng cách quy định bước nhảy của vòng lặp. Khi đó vòng lặp xác định sẽ có cấu trúc đầy đủ hơn như sau:

```
for(i = start_value; i < end_value; i+ = step_value)
//step_value: bước nhảy
{
  list of operation
}
```

Ví dụ để in các phần tử ở vị trí lẻ của mảng ta thực hiện đoạn lệnh sau:

```
int i;
float a[20];
….
for(i = 0; i < 19; i++)
If (i% 2 = = 1 printf(a[i]);
```

Trong trường hợp giá trị của bước nhảy khác 1, end_value không thực sự là giá trị cuối cùng của biến chạy mà đóng vai trò là điều kiện kết thúc. Vòng lặp sẽ được thực hiện cho đến khi giá trị của biến chạy vượt quá end_value. Trong ví dụ trên, giá trị của biến chạy i sẽ chạy từ 0, 3, 5, 7,... đến 19. Số lần lặp sẽ là 10. Mặc dù end_value đóng vai trò là điều kiện kết thúc nhưng do bước nhảy của biến chạy đã được xác định nên số lần lặp của vòng lặp vẫn luôn được xác định.

1.3.3.5. Câu lệnh vòng lặp không xác định

Khi số lần lặp chưa được biết trước, chúng ta không thể sử dụng câu lệnh vòng lặp xác định. Khi đó câu lệnh vòng lặp không xác định sẽ được sử dụng. Cấu trúc của câu lệnh vòng lặp không xác định như sau:

```
while (condition)
{
  list of operation...
}
```

Từ cấu trúc của câu lệnh ta có thể dễ dàng rút ra ý nghĩa của câu lệnh như sau: một đoạn lệnh sẽ được thực hiện lặp đi lặp lại trong khi một điều kiện cho trước vẫn được thỏa mãn. Cũng như với câu lệnh điều kiện, điều kiện ở đây phải có giá trị kiểu logic. Hơn nữa, giá trị của điều kiện phải thay đổi ít nhất một lần trong quá trình lặp từ ĐÚNG (TRUE) sang SAI (FALSE), vì nếu không đoạn lệnh sẽ được thực hiện

mãi mãi, đoạn lệnh rơi vào trường hợp vòng lặp vô hạn, và có thể dẫn đến treo chương trình.

Số lần lặp của vòng lặp này có thể thay đổi từ 0 (điều kiện sai ngay từ đầu) cho đến vô cùng (điều kiện luôn luôn đúng). Ngoài ra còn một biến thể của vòng lặp vô hạn trong đó số lần lặp thay đổi từ 1 đến vô cùng như sau:

```
do
{
  list of operation……
} while (condition);
```

Như chúng ta thấy, ngay cả khi điều kiện sai ngay từ đầu, thì đoạn lệnh vẫn được thực hiện ít nhất 1 lần. Đó là do việc kiểm tra điều kiện được thực hiện sau đoạn lệnh.

Để hiểu rõ câu lệnh vòng lặp không xác định, chúng ta hãy xem xét một ví dụ đơn giản về giải thuật tìm phần dư của một phép chia số nguyên dưới đây.

```
Giải thuật PHẦN_DƯ
Dữ liệu đầu vào:
a, b: KIỂU SỐ NGUYÊN
Điều kiện áp dụng: a, b > 0
Kết quả trả về: số dư của phép chia a cho b (a MOD b).
Ví dụ 10 MOD 3 = 1
Khai báo biến
        int m;
  {
        m = a;
        while(m >= b)
        {
                m = m – b
        }
        //KẾT THÚC VÒNG LẶP while
        TRẢ VỀ m
  }
```

Giải thuật được diễn giả như sau:

- Một biến mkiểu số nguyên được khai báo để lưu giữ kết quả trả về.
- Giá trị của m ban đầu được gán bằng giá trị của đầu vào a.
- Trong khi giá trị của m còn lớn hơn hoặc bằng b thì giảm giá trị của mđi một lượng bằng b.

Vì mỗi lần giá trị của m được giảm đi một lượng bằng b, nên m luôn đồng dư với a theo mô đun b.

Vì điều kiện thực hiện vòng lặp là mlớn hơn hoặc bằng b, nên điều kiện kết thúc vòng lặp sẽ là m < b. Do đó sau khi thực hiện vòng lặp msẽ có giá trị nhỏ hơn b.

Giá trị ban đầu của m bằng a lớn hơn 0, trong khi đó ta chỉ thực hiện việc giảm giá trị của m đi 1 lượng bằng b khi mà m còn lớn hơn hoặc bằng b. Do đó giá trị của m luôn lớn hơn hoặc bằng 0.

Vì giá trị của m giảm đi sau mỗi lần lặp, nên số lần lặp của vòng lặp bị giới hạn. Bằng kiến thức suy luận logic, ta có thể suy ra số lần lặp của vòng lặp sẽ bằng phần thương của a chia cho b. Vì vậy vòng lặp **while** luôn kết thúc với mọi giá trị dương của avà b.

Kết thúc vòng lặp, giá trị của m thỏa mãn

$$0 <= m < b$$

trong khi đó

m luôn đồng dư với atheo mô đun b.

Vì vậy kết thúc vòng lặp, giá trị của m chính là phần dư của atheo mô đun b.

Điều đó cho phép ta kết luận rằng kết quả trả về của thuật toán PHẦN_DƯ là chính xác, hay nói một cách khác thuật toán tính phần dư đã được thiết kế đúng.

Với các kiến thức cơ bản về cấu trúc dữ liệu và ngôn ngữ diễn đạt giải thuật được trình bày trong phần đầu của chương này, học viên đã bắt đầu có thể viết các giải thuật cơ bản, và có thể đọc được các giải thuật sẵn có. Phần cuối cùng của chương này, chúng tôi sẽ giới thiệu một kỹ thuật hết sức hiệu quả và được sử dụng rộng rãi để thiết kế giải thuật: kỹ thuật đệ quy.

1.4. Đánh giá hiệu quả của thuật toán

Tính hiệu quả của thuật toán thông thường được đo bởi thời gian tính (thời gian được sử dụng để tính bằng máy hoặc bằng phương pháp thủ công) khi các giá trị đầu vào có kích thước xác định. Tính hiệu quả của thuật toán cũng được xem xét theo thước đo dung lượng bộ nhớ đã sử dụng để tính toán khi kích thước đầu vào đã xác định.

Hai thước đo đã nêu trên liên quan đến độ phức tạp tính toán của một thuật toán, được gọi là độ phức tạp thời gian và độ phức tạp không gian(còn gọi là độ phức tạp dung lượng nhớ).

Việc xem xét độ phức tạp về mặt không gian gắn liền với việc xem xét các cấu trúc dữ liệu đặc biệt được dùng để thực hiện thuật toán. Chúng ta sẽ chỉ đề cập đến độ phức tạp thời gian của thuật toán.

Độ phức tạp thời gian của một thuật toán thường được biểu diễn thông qua số phép toán trong khi thực hiện thuật toán khi các giá trị dữ liệu đầu vào có kích thước xác định. Thông thường số các phép tính được thực hiện phụ thuộc vào cỡ của bài toán, tức là độ lớn của đầu vào. Vì thế **độ phức tạp thuật toán** là một hàm phụ thuộc đầu vào. Tuy nhiên trong những ứng dụng thực tiễn, chúng ta không cần biết chính xác hàm này mà chỉ cần biết một ước lượng đủ tốt của chúng.

Để ước lượng độ phức tạp của một thuật toán ta thường dùng khái niệm bậc O-lớn

- Bậc 0 lớn:

  Gọi *n* là độ lớn đầu vào. Tùy thuộc từng bài toán mà *n* có thể nhận những giá trị khác nhau. Chẳng hạn, bài toán tính giai thừa thì *n* chính là số cần tính giai thừa. Nhiều bài toán số trị, chẳng hạn tính sai phân thì *n* là số chữ số có nghĩa cần đạt được. Trong các phép tính đối với ma trận thì *n* là số hàng hoặc cột của ma trận.

Độ phức tạp của bài toán phụ thuộc vào *n*. Ở đây ta không chỉ đặc trưng độ phức tạp bởi số lượng phép tính, mà dùng một đại lượng tổng quát là *tài nguyên cần dùng R*(*n*). Đó có thể là số lượng phép tính (có thể tính cả số lần truy nhập bộ nhớ, hoặc ghi vào bộ nhớ); nhưng cũng có thể là thời gian thực hiện chương trình (*độ phức tạp về thời gian*) hoặc dung lượng bộ nhớ cần phải cấp để chạy chương trình (*độ phức tạp về không gian*).

Xét quan hệ giữa tài nguyên và độ lớn đầu vào, nếu như tìm được hằng số C > 0, C không phụ thuộc vào n, sao cho với n đủ lớn, các hàm R(n), g(n) đều dương và

$$R(n) \square C.g(n)$$

thì ta nói thuật toán có độ phức tạp cỡ O(g(n)).

- **Diễn giải**

Độ phức tạp không phải là độ đo chính xác lượng tài nguyên máy cần dùng, mà đặc trưng cho động thái của hệ thống khi kích thước đầu vào tăng lên. Chẳng hạn với thuật toán có độ phức tạp tuyến tính O(n), nếu kích thước đầu vào tăng gấp đôi thì có thể ước tính rằng tài nguyên cần dùng cũng tăng khoảng gấp đôi. Nhưng với thuật toán có độ phức tạp bình phương $O(n^2)$ thì tài nguyên sẽ tăng gấp bốn. Mặt khác, với thuật toán có độ phức tạp hàm mũ $O(2^n)$ thì chỉ cần công thêm 2 đơn vị vào độ lớn đầu vào cũng đã làm tài nguyên tăng gấp 4 lần (tức là theo cấp số nhân).

- Các độ phức tạp thường gặp đối với các thuật toán thông thường gồm có:

Độ phức tạp hằng số, O (1). Số phép tính/thời gian chạy/dung lượng bộ nhớ không phụ thuộc vào độ lớn đầu vào. Chẳng hạn như các thao tác hệ thống: đóng, mở file.

Độ phức tạp tuyến tính, O(n). Số phép tính/thời gian chạy/dung lượng bộ nhớ có xu hướng tỉ lệ thuận với độ lớn đầu vào. Chẳng hạn như tính tổng các phần tử của một mảng một chiều.

Độ phức tạp đa thức, O(P(n)), với P là đa thức bậc cao (từ 2 trở lên). Chẳng hạn như các thao tác tính toán với mảng nhiều chiều (tính định thức ma trận).

Độ phức tạp logarit, O(logn) (chú ý: bậc của nó thấp hơn so với O(n)). Chẳng hạn thuật toán Euclid để tìm ước số chung lớn nhất.

Độ phức tạp hàm mũ, $O(2^n)$. Trường hợp này bất lợi nhất và sẽ rất phi thực tế nếu thực hiện thuật toán với độ phức tạp này.

Ví dụ: Xét thuật toán tìm kiếm phần tử lớn nhất trong dãy số nguyên cho trước

Input: a là mảng các số nguyên, n > 0, là số các số trong mảng a.

Output: Max, số lớn nhất trong mảng a.

Giải thuật:

1. Max = a[0];
2. for (int i = 0; i < n - 1; i++) if (Max < a[i]) Max
   = a[i];

Vì các phép toán dùng ở đây là các phép toán so sánh sơ cấp nên ta sẽ dùng số các phép toán sơ cấp để đo độ phức tạp của thuật toán. Ta dễ dàng thấy được số các phép toán so sánh sơ cấp được sử dụng ở đây là 2(n – 1). Vì vậy ta nói rằng độ phức tạp của thuật toán nói trên là O(n)(gọi là độ phức tạp tuyến tính.)

1.5. Đệ quy

1.5.1. Giới thiệu về đệ quy (Recurrence)

Recurrence trong tiếng Anh có nghĩa là gọi lại. Bản chất của việc thiết kế giải thuật theo phương pháp đệ quy là việc gọi lại chính giải thuật đó trong quá trình thiết kế giải thuật. Các giải thuật được thiết kế theo phương pháp đệ quy được gọi chung một tên là giải thuật đệ quy.

Vì giải thuật đệ quy là gọi lại chính mình, nên để thiết kế giải thuật đệ quy cần phải quy định cách gọi một giải thuật trong ngôn ngữ thiết kế giải thuật. Lời gọi một giải thuật có cấu trúc như sau:

TÊN_GIẢI_THUẬT (danh sách các tham số đầu vào);

Ví dụ để thực hiện lời gọi giải thuật phần dư được thiết kế ở trên:

a = PHẦN DƯ (10,3);

Sau lời gọi trên, giá trị trả về của thuật toán PHẦN_DƯ sẽ được gán vào biến a. Do đó biến asẽ mang giá trị bằng 1.

Nói chung một giải thuật đệ quy thường được thiết kế theo cấu trúc:

- Kiểm tra điều kiện kết thúc đệ quy và trả về kết quả tương ứng.
- Chia nhỏ dữ liệu đầu vào.
- Áp dụng giải thuật đệ quy trên từng phần dữ liệu con.
- Tổng hợp các kết quả thu được.

Trong toán học có một phép toán gần tương đương với giải thuật đệ quy là phép toán truy hồi: mỗi một giá trị được tính toán dựa trên một hoặc một vài giá trị đã được tính toán trước đó. Trên thực tế việc triển khai các phép toán truy hồi của toán học trên máy tính được thực hiện hết sức trực quan bằng các thuật toán đệ quy. Hãy xem xét một ví dụ với dãy số Fibonacci.

Các phần tử của dãy số Fibonacci được định nghĩa như sau:

F[1] = 1

F[2] = 1

F[n] = F[n – 1] + F[n – 2] với mọi n > 2

Theo như định nghĩa, mỗi phần tử của dãy số Fibonacci được tính bằng cách lấy tổng của hai phần tử đứng ngay trước nó. Vì vậy, để tính được giá trị của một phần tử trước tiên cần phải biết giá trị của hai phần tử đứng ngay trước nó.

Với công thức truy hồi nêu trên chúng ta có thể tính được giá trị của bất kỳ phần tử nào trong dãy Fibonacci bằng cách lùi dần về vị trí đầu dãy. Điều đó cũng có nghĩa là để xác định được dãy thì điều kiện bắt buộc là phải xác định được giá trị của 2 phần tử đầu dãy. Điều kiện đó được gọi là *điều kiện đầu*.

Bây giờ chúng ta hãy xem xét cách triển khai giải thuật tính giá trị của dãy Fibonacci dựa trên giải thuật đệ quy. Phần định nghĩa của giải thuật sẽ như sau:

```
Giải thuật Fibonaci
Dữ liệu đầu vào: int n ;
Điều kiện áp dụng: n > 0 ;
```

Kết quả trả về: giá trị của phần tử thứ n trong dãy Fibonacci F[n].

Để thiết kế giải thuật này, chúng ta không cần phải sử dụng thêm biến phụ, nên phần khai báo biến có thể được bỏ qua. Trước tiên cần phải triển khai điều kiện đầu của dãy, hay xác định 2 giá trị ban đầu của dãy. Việc này tương đương với kiểm tra điều kiện kết thúc lời gọi đệ quy và được thực hiện hết sức đơn giản bằng câu lệnh điều kiện để kiểm tra giá trị của n:

```
if(n == 1)
  return 1 ;
else
{
if(n == 2)
  return 1 ;
}
```

Cuối cùng, để tính giá trị phần tử thứ n trong dãy Fibonacci, hai phần tử thứ n – 1 và n – 2 đứng ngay trước nó được tính bằng cách gọi đệ quy đến chính giải thuật Fibonacci.

Tổng kết lại, giải thuật Fibonacci được viết đầy đủ như sau:

```
Giải thuật Fibonacci:
        Dữ liệu đầu vào: int n ;
Điều kiện áp dụng n > 0
Điều kiện trả về: giá trị phần tử thứ n trong dãy
Fibonacci F[n]
        {
                if(n == 1) return 1 ;
                else
                {
                        if(n == 2) return 1 ;
                }
                return Fibonacci(n – 1) + Fibonacci(n – 2) ;

        }
```

Để minh họa giải thuật, chúng ta hãy xem cách thức hoạt động của giải thuật với một giá trị của thể n = 6.

```
{
n = 6
- điều kiện n = 1 không thỏa mãn
- điều kiện n = 2 không thỏa mãn
- gọi giải thuật FIBONACCI với n = 4:
        - điều kiện n = 1 không thỏa mãn
        - điều kiện n = 2 không thỏa mãn
        - gọi giải thuật FIBONACCI với n = 3:
                - điều kiện n = 1 không thỏa mãn
                - điều kiện n = 2 không thỏa mãn
                - gọi giải thuật FIBONACCI với n = 2:
                - điều kiện n = 2 thỏa mãn, trả về giá trị 2
                - gọi giải thuật FIBONACCI với n = 1:
                - điều kiện n = 1 thỏa mãn, trả về giá trị 1
                - tính tổng FIBONACCI(2) + FIBONACCI(1) = 2 + 1 = 3
                - trả về giá trị 3
        - gọi giải thuật FIBONACCI với n = 2:
                - điều kiện n = 2 thỏa mãn, trả về giá trị 2
        - tính tổng FIBONACCI(3) + FIBONACCI (2) = 3 + 2 = 5
        - trả về giá trị 5
- gọi giải thuật FIBONACCI với n = 3:
                - điều kiện n = 1 không thỏa mãn
        - điều kiện n = 2 không thỏa mãn
        - gọi giải thuật FIBONACCI với n = 2:
                - điều kiện n = 2 thỏa mãn, trả về giá trị 2
        - gọi giải thuật FIBONACCI với n = 1:
                - điều kiện n = 1 thỏa mãn, trả về giá trị 1
        - tính tổng FIBONACCI(2) + FIBONACCI(1) = 2 + 1 = 3
        - trả về giá trị 3
- tính tổng FIBONACCI(4) + FIBONACCI(3) = 5 + 3 = 8
- trả về giá trị 8
}
```

Quá trình thực hiện giải thuật cũng có thể được minh hoạ bằng cấu trúc cây như sau. Mỗi nút của cây tương ứng với một lời gọi đến giải thuật.

Như vậy để tính giá trị phần tử thứ 5 của dãy FIBONACCI chỉ cần thực hiện lời gọi Fibonacci(5). Nhưng trên thực tế, theo như minh họa ở trên, thì số lời gọi thực tế đến

giải thuật này là 9, vì bên trong lời gọi Fibonacci(5) có chứa 2 lời gọi đến Fibonacci(4) và Fibonacci(3),... Tuy nhiên, việc thực hiện các lời gọi con này là hoàn toàn "trong suốt" đối với người sử dụng giải thuật.

Ưu điểm lớn nhất của phương pháp đệ quy là tính hiệu quả trong thiết kế và lập trình. Như đã thấy trong ví dụ trên, với việc sử dụng lời đệ quy, logic của chương trình được biểu diễn hết sức đơn giản và trong sáng. Giải thuật đệ quy đặc biệt thích hợp để giải quyết các vấn đề, để triển khai các hàm hay để xử lý các dữ liệu vốn dĩ đã được định nghĩa bằng phương pháp truy hồi.

Tuy nhiên giải thuật đệ quy không thật sự hiệu quả nếu xét về mặt thực hiện chương trình. Như ở trong ví dụ trên, để tính Fibonacci(5), Fibonacci(3) và Fibonacci(1) được gọi 2 lần, còn Fibonacci(2) được gọi 3 lần. Số lần lời gọi con đến giải thuật cũng có thể được tính dựa trên thuật toán truy hồi. Nếu gọi số lần gọi truy hồi đến giải thuật Fibonacci cho vị trí thứ n là CALL (n), thì CALL (n)sẽ được tính theo công thức :

$$Call(n) = 2 + CALL(n - 1) + CALL\ (n - 2)$$

Như vậy số lần gọi truy hồi để tính phần tử thứ n của dãy Fibonacci còn tăng nhanh hơn giá trị của nó. Theo cách giải hệ thức truy hồi ta tính được:

$$F(n) = \left(\frac{\sqrt{5}-1}{2}\right)^2 + \left(\frac{\sqrt{5}+1}{2}\right)^n$$

Vì vậy số lần gọi truy hồi của thuật toán trên sẽ tăng theo cấp số mũ. Thời gian thực hiện giải thuật sẽ tăng rất nhanh khi giá trị của n tăng lên.

Còn một vấn đề nữa khi số lần gọi truy hồi của thuật toán tăng lên là dung lượng bộ nhớ yêu cầu. Trên thực tế, mỗi khi thực hiện một lời gọi đệ quy, môi trường thực thi hiện thời sẽ được ghi lại để chương trình có thể tiếp tục chạy sau khi lời gọi kết thúc. Bộ nhớ cần thiết để chạy chương trình sẽ tăng lên khi độ sâu của lời gọi đệ quy tăng lên. Như trong hình minh họa ở trên, độ sâu của lời gọi đệ quy chính bằng giá trị của n. Hạn chế này có thể chấp nhận được đối với các thuật toán đòi hỏi ít sử dụng bộ nhớ. Vấn đề sẽ trở nên trầm trọng khi chương trình cần sử dụng một lượng lớn bộ nhớ nhất định cho mỗi lần gọi đệ quy.

Vì vậy, khi một chương trình có thể được thiết kế bằng một giải thuật trực tiếp mà không quá phức tạp thì không nên sử dụng giải thuật đệ quy để thiết kế chương trình.

Ví dụ, để tính phần tử thứ n của dãy Fibonacci, ta có thể sử dụng giải thuật trực tiếp sau:

```
Giải thuật Fibonacci
Dữ liệu đầu vào: n: KIỂU SỐ NGUYÊN
Điều kiện áp dụng: n > 0
Kết quả trả về: giá trị của phần tử thứ n trong dãy Fibonacci F[n]
Khai báo biến
     int   i, x, y, z ;
        if (n == 1)return 1 ;
        else
              {
                     If (n == 2)return 1 ;
              }
        x = 1 ;
        y = 1 ;
        i = 3 ;
        while (i < n)
              {
                     z = x + y ;
                     x = y ;
                     y = z ;
                     i = i + 1 ;
              }
        return (x + y);
        }
```

1.5.2. Phương pháp quay lui

Trong thực tế có nhiều vấn đề trong đó không thể xây dựng lời giải bằng các quy tắc, các công thức tính toán nhất định. Để tìm lời giải cho các vấn đề đó, người ta không còn cách nào khác là phải thử lần lượt các trường hợp có thể của lời giải.

Một trong những phương pháp để giải quyết các vấn đề chung chung đó là phương pháp thử và quay lại nếu sai (trial and error). Vấn đề được chia nhỏ thành các vấn đề con nhỏ hơn, và chương trình sẽ chọn thử lần lượt các giá trị có thể của một vấn đề con. Với mỗi giá trị có thể đó chương trình sẽ tìm cách kiểm tra xem có tồn tại lời giải với các vấn đề con còn lại hay không. Nếu không tồn tại lời giải cho các vấn đề con còn lại, chương trình sẽ quay lại thử giá trị có thể tiếp theo của vấn đề con hiện thời. Vấn đề thường được chia nhỏ thành nhiều tầng theo hướng đệ quy. Việc tìm lời giải cho tầng dưới sẽ được thực hiện bằng một lời gọi đệ quy. Nếu không tìm được lời giải cho tầng dưới thì quay lui lên tầng trên để thử lời giải tiếp theo. Vì vậy phương pháp này được gọi là phương pháp quay lui.

Để hiểu rõ hơn phương pháp xây dựng giải thuật quay lui, ta hãy xem xét phân tích một ví dụ cụ thể: bài toán Tám Hậu. Mục tiêu của bài toán là tìm cách bố trí 8 con hậu trên một bàn cờ vua sao cho không con nào có thể ăn con nào.

Đặt vấn đề:

Bàn cờ vua có kích thước 8□8. Để 8 con hậu không con nào ăn con nào thì nhất thiết không có con nào được sắp xếp theo cùng một hàng hay cùng một cột.

Để mô tả bàn cờ vua, ta sẽ sử dụng một mảng hai chiều có kích thước tương đương gồm 8 hàng và 8 cột, gọi là mảng ban_co. Mảng sẽ được khởi tạo với tất cả các phần tử mang giá trị bằng 0.

Thay vì tìm cách tìm vị trí cho từng con hậu trên bàn cờ, ta sẽ tìm lần lượt vị trí của từng con hậu trên từng hàng từ 1 đến 8. Nhớ rằng trên mỗi hàng chỉ có một con hậu. Mỗi lần xếp một con hậu vào một hàng, ta sẽ đánh dấu các vị trí ở các hàng tiếp theo mà có thể bị con hậu ở hàng đó ăn bằng cách gán giá trị cho các phân tử trong mảng ban_cobằng giá trị của hàng hiện tại.

Ví dụ, nếu ở hàng 2 ta xếp một con hậu ở cột thứ 3 thì các phần tử của mảng ban_coở các vị trí: (3, 2) (3, 3) (3, 4) (4, 1) (4, 3) (4, 5) (5, 3) (5, 6) (6, 3) (6, 7) (7, 3) (7, 8) (8, 3) sẽ được gán giá trị bằng 2 nếu như nó chưa được gán giá trị.

Bàn cờ

o o o o o o o o
o o x o o o o o
o x x x o o o o
x o x o x o o o
o o x o o x o o
o o x o o o x o
o o x o o o x o
o o x o o o o x

Theo quy tắc chơi cờ vua, tất cả các phần tử nằm cùng cột, hoặc cùng đường chéo với phần tử ở vị trí (2, 3) sẽ bị đánh dấu. Lưu ý:

- Phần tử ở vị trí (i, j) thuộc cùng cột với (2, 3) nếu j = 3
- Phần tử ở vị trí (i, j) thuộc cùng đường chéo trái với (2, 3) nếu: i + j = 2 + 3
- Phần tử ở vị trí (i, j) thuộc cùng đường chéo phải với (2, 3) nếu: j – i = 3 – 2

Sau đó ta đi xuống hàng tiếp theo để tiếp thục việc kiểm thử. Nếu tìm được lời giải ở hàng tiếp theo thì giá trị ở hàng hiện thời là đúng, ta đã tìm được lời giải cho bài toán. Ngược lại, nếu không xếp được hậu cho hàng tiếp theo, ta phải tiếp thục thử vị trí tiếp theo cho con hậu ở hàng hiện thời. Lưu ý trước khi thử vị trí tiếp theo cần phải xóa bỏ các đánh dấu của vị trí trước. Để thuật toán Tám Hậu được rõ ràng, ở đây chúng tôi xây dựng các giải thuật toán con: KHOI_TAO để khởi tạo mảng ban đầu, DANH_DAU để đánh dấu bàn cờ mỗi lần đặt một con hậu mới, XOA_DAU để xóa bỏ các đánh dấu, và giải thuật THU_HAU để thử xếp hậu vào từng hàng trong bàn cờ

```
Giải thuật KHOI_TAO
Dữ liệu đầu vào: int ban_co[8][8];
Điều kiện ban đầu: không có
Kết quả trả về: tất cả các phần tử của mảng ban_co đều có giá trị
bằng 0
Khai báo biến
        int i, j;
        {
        for(i = 0; i < 7; i++)
            for(j = 0; i < 7; j++)
                  ban_co[i][j] = 0;
        }
```

Giải thuật DANH_DAU

Dữ liệu đầu vào:

```
int ban_co[8][8];
int m.n;
```

Điều kiện ban đầu: m >= 1, n <= 8;

Kết quả trả về: tất cả các phần tử trong mảng ban_co nằm trong hàng ở vị trí lớn hơn m nằm cùng trên cùng một cột hoặc nằm trên cùng đường chéo với phần tử (m, n) có giá trị bằng 0 (chưa bị đánh dấu) sẽ được đánh dấu bằng giá trị m

Khai báo biến

```
int i, j;
{
    for(i = m + 1; i < 7; i++)
    {
        if (ban_co[i][n] == 0)
            ban_co[i][n] = m;
    }
    for(i = 0; i < n - 1; i++)
    {
        j = m + n - i;
        if ( (j >= 1) && (ban_co[i][j] == 0))
            ban_co[i][j] = m;
    }
    for(i = n  + 1; i < 7; i++)
    {
        j = n - m + i;
        if ((j <= 8) && (ban_co[i][j] == 0))
            ban_co[i][j] =m;
    }
}
```

Giải thuật XÓA DẤU

Dữ liệu đầu vào:

```
int ban_co[8][8];
int m;
```

Điều kiện áp dụng m >= 1; m <= 8;

Kết quả trả về: gán tất cả các phần tử của mảng ban_co bị đánh dấu bởi giá trị m về giá trị 0

Khai báo biến

```
int i,j;
```

```
    {
        for(i = m + 1; i < 7; i++)
        {
            for(j = 0; j < 7; j++)
            {
                if (ban_co[i][j] == m)
                    ban_co[i][j] = 0;
            }
        }
    }
```

```
Giải thuật THU_HAU
Dữ liệu đầu vào:
        int ban_co[8][8];
        int vi_tri[8];
        int ;
Điều kiện áp dụng: m >= 1; m <= 8;
Kết quả trả về: int bằng m nếu tìm được vị trí cho con hậu ở hàng thứ m và vị trí được ghi vào phần tử thứ m của mảng vị trí, 0 nếu không tìm được vị trí cho hậu ở hàng thứ m
Khai báo biến
    int i;
    tim_thay: kiểu int (C ko có kiểu Boolean, mặc định C hiểu 0 là false còn khác 0 là true)
{
    i = 1 ;
    while(i <= 8)
    {
       DANH_DAU(ban_co, m, i);
            tim_thay = THU_HAU(ban_co, vi_tri, m+1);
       if (tim_thay )
          {
             vi_tri[m] = i;
             return m;
          }
       else
             XOA_DAU(ban_co, m);
       i = i + 1;
    }
}
```

```
Giải thuật TAM_HAU
Dữ liệu đầu vào: không có
Điều kiện áp dụng: không có
Kết quả trả về: hiển thị một cách bố trí 8 con hậu trên ban cờ sao
cho không con nào ăn còn nào.
Khai báo biến:
     int ban_co[8][8];
     int vi_tri[8];
     int  tim_thay;
     int i;
   {
     KHOI_TAO(ban_co)
     tim_thay = THU_HAU(ban_co, vi_tri, 1);
     if (tim_thay)
     {
         for(i = 0; i < 7; i++)
              printf("Đặt hậu ở hàng thứ ");
              printf (i);
              printf("cột thứ ");
              printf (vi_tri[i]);
          }
     else
          printf ("Không bố trí được 8 hậu trên bàn cờ sao cho không
con nào ăn con nao");
   }
```

Một ví dụ khá điển hình mỗi khi đề cập đến giải thuật đệ quy là bài toán Tháp Hà Nội.

1.5.3. Bài toán tháp Hà Nội

Nội dung của bài toán có thể được phát biểu như sau:

Có ba cọc xếp đĩa A, B, C. Cọc B và cọc C rỗng. Cọc A có xếp n đồng xu với kích thước từ lớn đến nhỏ được xếp từ dưới lên trên (xem hình vẽ minh họa).

Hãy tìm cách chuyển toàn bộ tiền xu từ cọc A sang cọc B theo quy tắc:

- Mỗi lần chỉ được chuyển 1 đồng xu ở trên cùng từ cọc này sang cọc kia
- Không bao giờ được để đồng xu to hơn nằm trên đồng xu nhỏ hơn

Phân tích bài toán:

Trước tiên ta hãy đánh dấu các đồng xu lần lượt từ 1 đến n theo thứ tự kích thước tăng dần.

Vì đồng xu nhỏ hơn luôn được xếp lên trên, nên sau khi chuyển toàn bộ đồng xu từ cọc A sang cọc B, các đồng xu cũng được xếp theo thứ tự giống như ban đầu:

Ta hãy xem xét di chuyển của đồng xu n. Vì n là đồng xu lớn nhất nên n luôn nằm ở vị trí cuối cùng.

- Để n có thể di chuyển được thì n phải nằm ở vị trí trên cùng của cọc. Khi đó n sẽ là đồng xu duy nhất của cọc vì nó là đồng trên cùng và cũng là đồng dưới cùng.
- Để n có thể chuyển sang 1 cọc thứ hai thì đồng trên cùng của cọc thứ hai phải có kích thước lớn hơn n. Vì n là đồng lớn nhất nên không còn đồng xu nào có kích thước lớn hơn n. Điều đó có nghĩa là n chỉ có thể được chuyển sang một cọc trống.

Vì vậy, để chuyển được đồng xu thứ n từ cọc A (vị trí ban đầu của nó) sang cọc B thì khi đó trạng thái của các cọc sẽ là:

- Cọc A chỉ có đồng xu n
- Cọc B trống
- Như vậy tất cả các đồng xu còn lại sẽ nằm ở cọc C. Với quy tắc sắp xếp đồng xu, n – 1 đồng xu đó cũng phải được sắp xếp theo tứ tự trên nhỏ dưới to. Do đó để đạt được trạng thái này, ta phải chuyển toàn bộ n – 1 đồng xu trên cùng của cọc A sang cọc C. Sau khi chuyển đồng xu n từ A sang B, thì cọc A sẽ trống, cọc B chỉ có đồng xu n, và toàn bộ n – 1 đồng xu vẫn nằm ở cọc C. Việc còn lại là phải chuyển toàn bộ n – 1 đồng xu này từ cọc C sang cọc B. Ta có thể thấy rằng bài toán đã được giải quyết theo hướng đệ quy. Việc chuyển n đồng xu đã được chia nhỏ thành 2 lần chuyển n – 1 đồng xu. Điều kiện kết thúc của chuỗi đệ quy là khi chỉ còn 1 đồng xu. Khi đó ta sẽ không cần phải chia nhỏ bài toán mà có thể thực hiện việc chuyển đồng xu đó một cách trực tiếp.

Biểu diễn giải thuật:

Dựa vào phân tích ở trên, ta có thể biểu diễn giải thuật bằng ngôn ngữ diễn đạt giải thuật một cách hình thức như sau:

```
Giải thuật THAP_HA_NOI
Dữ liệu đầu vào:
            int n;
            char coc_1,coc_2,coc_3;
Điều kiện áp dụng: n > 0
Kết quả trả về: hiển thị cách chuyển n đồng xu từ cọc được đánh dấu
bằng coc_1 sang cọc được đánh dấu bằng coc_2
    {
        if (n == 1)
        printf( "Chuyển đồng xu % d từ Cọc % d sang Cọc % d”,
n, coc_1, coc_2";
        else
        {
            THAP_HA_NOI(n - 1, coc_1, coc_3, coc_2);
         printf("Chuyển đồng xu % d từ Cọc % d sang Cọc % d", n, " từ cọc
", coc_1, "sang cọc ",
        coc_2);
            THAP_HA_NOI(n-1, coc_3, coc_2, coc_1);
        }
    }
```

THAP_HA_NOI(4, 'A', 'B', 'C'):

- Điều kiện (n=1) không thỏa mãn
- thực hiện lời gọi tới THAP_HA_NOI(3, 'A', 'C', 'B')
- điều kiện (n=1) không thỏa mãn
- thực hiện lời gọi tới THAP_HA_NOI(2, 'A','B','C')
- điều kiện (n=1) không thỏa mãn
- thực hiện lời gọi tới THAP_HA_NOI(1, 'A','C','B')
- điều kiện (n=1) thỏa mãn;
- printf("Chuyển đồng xu ", 1, " từ cọc" , 'A', " sang cọc ",'C')
- printf("Chuyển đồng xu ", 2, " từ cọc ", 'A', " sang cọc ",'B');
- thực hiện lời gọi tới THAP_HA_NOI(1, 'C','B','A')
- điều kiện (n=1) thoả mãn
- printf("Chuyển đồng xu ", 1, " từ cọc ", 'C', " sang cọc ",'B');
- printf("Chuyển đồng xu ", 3, " từ cọc ", 'A', " sang cọc ",'C');
- thực hiện lời gọi tới THAP_HA_NOI(2, 'B','C','A')
- điều kiện (n=1) không thỏa mãn
- thực hiện lời gọi tới THAP_HA_NOI(1, 'B','A','C')
- điều kiện (n=1) thỏa mãn
- printf("Chuyển đồng xu ", 1, " từ cọc" , 'B', " sang cọc ",'A');
- printf("Chuyển đồng xu ", 2, " từ cọc ", 'B', " sang cọc ",'C');
- thực hiện lời gọi tới THAP_HA_NOI(1, 'A','C','B')
- điều kiện (n=1) thoả mãn
- printf("Chuyển đồng xu ", 1, " từ cọc ", 'A', " sang cọc ",'C');
- printf("Chuyển đồng xu ", 4, " từ cọc ", 'A', " sang cọc ",'B');
- thực hiện lời gọi tới THAP_HA_NOI(3, 'C', 'B', 'A')
- điều kiện (n=1) không thỏa mãn
- thực hiện lời gọi tới THAP_HA_NOI(2, 'C','A','B')
- điều kiện (n=1) không thỏa mãn
- thực hiện lời gọi tới THAP_HA_NOI(1, 'C','B','A')
- điều kiện (n=1) thỏa mãn
- printf("Chuyển đồng xu ", 1, " từ cọc" , 'C', " sang cọc ",'B');
- printf("Chuyển đồng xu ", 2, " từ cọc ", 'C', " sang cọc ",'A');
- thực hiện lời gọi tới THAP_HA_NOI(1, 'B','A','C')
- điều kiện (n=1) thoả mãn
- printf("Chuyển đồng xu ", 1, " từ cọc ", 'B', " sang cọc ",'A');
- printf("Chuyển đồng xu ", 3, " từ cọc ", 'C', " sang cọc ",'B');
- thực hiện lời gọi tới THAP_HA_NOI(2, 'A','B','C')
- điều kiện (n=1) không thỏa mãn
- thực hiện lời gọi tới THAP_HA_NOI(1, 'A','C','B')
- điều kiện (n=1) thỏa mãn
- printf("Chuyển đồng xu ", 1, " từ cọc" , 'A', " sang cọc ",'C');
- printf("Chuyển đồng xu ", 2, " từ cọc ", 'A', " sang cọc ",'B');
- thực hiện lời gọi tới THAP_HA_NOI(1, 'C','B','A')
- điều kiện (n=1) thoả mãn
- printf("Chuyển đồng xu ", 1, " từ cọc ", 'C', " sang cọc ",'B');

Để minh họa giải thuật, chúng tôi sẽ minh hoạ các bước thực hiện của giải thuật với một trường hợp cụ thể.
Tổng kết lại, kết quả hiển thị của giải thuật sẽ như sau:

```
Chuyển đồng xu 1 từ cọc A sang cọc C
Chuyển đồng xu 2 từ cọc A sang cọc B
Chuyển đồng xu 1 từ cọc C sang cọc B
Chuyển đồng xu 3 từ cọc A sang cọc C
Chuyển đồng xu 1 từ cọc B sang cọc A
Chuyển đồng xu 2 từ cọc B sang cọc C
Chuyển đồng xu 1 từ cọc A sang cọc C
Chuyển đồng xu 4 từ cọc A sang cọc B
Chuyển đồng xu 1 từ cọc C sang cọc B
Chuyển đồng xu 2 từ cọc C sang cọc A
Chuyển đồng xu 1 từ cọc B sang cọc A
Chuyển đồng xu 3 từ cọc C sang cọc B
Chuyển đồng xu 1 từ cọc A sang cọc C
Chuyển đồng xu 2 từ cọc A sang cọc B
Chuyển đồng xu 1 từ cọc C sang cọc B
```

Học viên có thể tự kiểm tra tính đúng đắn của kết quả trên.

TÓM LƯỢC CUỐI BÀI

Các bạn đã được học các kiến thức cơ bản về cấu trúc dữ liệu và giải thuật.

Các bạn cần ghi nhớ các vấn đề sau:

- Nắm được khái niệm về cấu trúc dữ liệu và giải thuật.
- Nắm được các kiểu dữ liệu cơ bản.
- Nắm được các kiểu dữ liệu trừu tượng.
- Nắm được ngôn ngữ diễn đạt giải thuật.
- Nắm được thuật toán Đệ quy.
- Nắm được phương pháp quay lui.

Bài tiếp theo các bạn sẽ được học về Danh sách.

BÀI TẬP

1. Thế nào là kiểu dữ liệu có cấu trúc? Thế nào là kiểu dữ liệu do người định nghĩa. Hãy cho một vài ví dụ trong Pascal.

2. Hãy nêu định nghĩa về giải thuật? Thế nào là ngôn ngữ diễn đạt giải thuật?

3. Hãy nêu lên và mô tả ba cấu trúc điều khiển dùng trong phát triển các thuật toán có cấu trúc?

4. Giai thừa của số nguyên không âm n, ký hiệu bởi n! được định nghĩa như sau:

0! = 1

n! = 1 * 2 * ... * n đối với n > 0

Hãy trình bày thuật toán để tính n! theo giải thuật đệ quy và không đệ quy.

5. Dựa theo thuật toán phần dư, viết một thuật toán cho phép tính ước số chung lớn nhất của hai số nguyên: theo giải thuật đệ quy và không đệ quy.