### 3.3. VÍ DỤ VỀ GIẢI THUẬT ĐỆ QUY

#### 3.3.1. Hàm tính giai thừa

function Factorial(n: Integer): Integer;

begin

if n = 0 then Factorial := 1

else Factorial := n \* Factorial(n - 1);

end;

Ở đây, phần neo định nghĩa kết quả hàm tại n = 0, còn phần đệ quy (ứng với n > 0) sẽ định nghĩa kết quả hàm qua giá trị của n và giai thừa của n - 1.

Ví dụ: Dùng hàm này để tính 3!, trước hết nó phải đi tính 2! bởi 3! được tính bằng tích của 3 \* 2!. Tương tự để tính 2!, nó lại đi tính 1! bởi 2! được tính bằng 2 \* 1!. Áp dụng bước quy nạp này thêm một lần nữa, 1! = 1 \* 0!, và ta đạt tới trường hợp của phần neo, đến đây từ giá trị 1 của 0!, nó tính được 1! = 1\*1 = 1; từ giá trị của 1! nó tính được 2!; từ giá trị của 2! nó tính được 3!; cuối cùng cho kết quả là 6:

3! = 3 \* 2!

🡓

2! = 2 \* 1!

🡓

1! = 1 \* 0!

🡓

0! = 1

#### 3.3.2. Dãy số Fibonacci

Dãy số Fibonacci bắt nguồn từ bài toán cổ về việc sinh sản của các cặp thỏ. Bài toán đặt ra như sau:

1) Các con thỏ không bao giờ chết

2) Hai tháng sau khi ra đời, mỗi cặp thỏ mới sẽ sinh ra một cặp thỏ con (một đực, một cái)

3) Khi đã sinh con rồi thì cứ mỗi tháng tiếp theo chúng lại sinh được một cặp con mới Giả sử từ đầu tháng 1 có một cặp mới ra đời thì đến giữa tháng thứ n sẽ có bao nhiêu cặp. Ví dụ, n = 5, ta thấy:

Giữa tháng thứ 1:1 cặp (ab) (cặp ban đầu)

Giữa tháng thứ 2:1 cặp (ab) (cặp ban đầu vẫn chưa đẻ)

Giữa tháng thứ 3:2 cặp (AB)(cd) (cặp ban đầu đẻ ra thêm 1 cặp con) Giữa tháng thứ 4:3 cặp (AB)(cd)(ef) (cặp ban đầu tiếp tục đẻ)

Giữa tháng thứ 5:5 cặp (AB)(CD)(ef)(gh)(ik) (cả cặp (AB) và (CD) cùng đẻ) Bây giờ, ta xét tới việc tính số cặp thỏ ở tháng thứ n: F(n)

Nếu mỗi cặp thỏ ở tháng thứ n - 1 đều sinh ra một cặp thỏ con thì số cặp thỏ ở tháng thứ n sẽ là:

F(n) = 2 \* F(n - 1)

Nhưng vấn đề không phải như vậy, trong các cặp thỏ ở tháng thứ n - 1, chỉ có những cặp thỏ đã có ở tháng thứ n - 2 mới sinh con ở tháng thứ n được thôi. Do đó F(n) = F(n - 1) + F(n - 2) (= số cũ + số sinh ra). Vậy có thể tính được F(n) theo công thức sau:

F(n) = 1 nếu n  2

F(n) = F(n - 1) + F(n - 2) nếu n > 2

function F(n: Integer): Integer;

begin

if n  2 then F := 1

else F := F(n - 1) + F(n - 2);

end;

**3.3.3. Giả thuyết của Collatz**

Collatz đưa ra giả thuyết rằng: với một số nguyên dương X, nếu X chẵn thì ta gán X := X div 2; nếu X lẻ thì ta gán X := X \* 3 + 1. Thì sau một số hữu hạn bước, ta sẽ có X = 1.

Ví dụ: X = 10, các bước tiến hành như sau:

1. X = 10 (chẵn) **** X := 10 div 2; (5)

2. X = 5 (lẻ) **** X := 5 \* 3 + 1; (16)

3. X = 16 (chẵn) **** X := 16 div 2; (8)

4. X = 8 (chẵn) **** X := 8 div 2 (4)

5. X = 4 (chẵn) **** X := 4 div 2 (2)

6. X = 2 (chẵn) **** X := 2 div 2 (1)

Cứ cho giả thuyết Collatz là đúng đắn, vấn đề đặt ra là: Cho trước số 1 cùng với hai phép toán \* 2 và div 3, hãy sử dụng một cách hợp lý hai phép toán đó để biến số 1 thành một giá trị nguyên dương X cho trước.

Ví dụ: X = 10 ta có 1 \* 2 \* 2 \* 2 \* 2 div 3 \* 2 = 10.

Dễ thấy rằng lời giải của bài toán gần như thứ tự ngược của phép biến đổi Collatz: Để biểu diễn số X > 1 bằng một biểu thức bắt đầu bằng số 1 và hai phép toán "\* 2", "div 3". Ta chia hai trường hợp:

Nếu X chẵn, thì ta tìm cách biểu diễn số X div 2 và viết thêm phép toán \* 2 vào cuối Nếu X lẻ, thì ta tìm cách biểu diễn số X \* 3 + 1 và viết thêm phép toán div 3 vào cuối

procedure Solve(X: Integer); {In ra cách biểu diễn số X}

begin

if X = 1 then Write(X)

else

if X mod 2 = 0 then

begin

Solve(X div 2);

Write(' \* 2');

end

else

begin

Solve(X \* 3 + 1);

Write(' div 3');

end;

end;

Trên đây là cách viết đệ quy trực tiếp, còn có một cách viết đệ quy tương hỗ như sau:

procedure Solve(X: Integer); forward;

procedure SolveOdd(X: Integer);

begin

Solve(X \* 3 + 1);

Write(' div 3');

end;

procedure SolveEven(X: Integer);

begin

Solve(X div 2);

Write(' \* 2');

end;

procedure Solve(X: Integer);

begin

if X = 1 then Write(X)

else

if X mod 2 = 1 then SolveOdd(X)

else SolveEven(X);

end;

Trong cả hai cách viết, để tìm biểu diễn số X theo yêu cầu chỉ cần gọi Solve(X) là xong. Tuy nhiên trong cách viết đệ quy trực tiếp, thủ tục Solve có lời gọi tới chính nó, còn trong cách viết đệ quy tương hỗ, thủ tục Solve chứa lời gọi tới thủ tục SolveOdd và SolveEven, hai thủ tục này lại chứa trong nó lời gọi ngược về thủ tục Solve.

Đối với những bài toán nêu trên, 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 quy nạp của hàm đó được xác định dễ dàng.

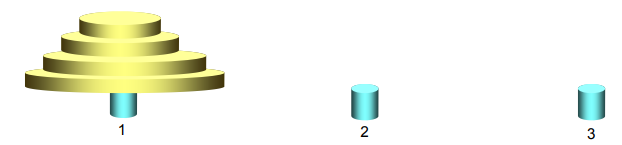
Nhưng không phải lúc nào phép giải đệ quy cũng có thể nhìn nhận và thiết kế dễ dàng như vậy. Thế thì vấn đề gì cần lưu tâm trong phép giải đệ quy?. Có thể tìm thấy câu trả lời qua việc giải đáp các câu hỏi sau:

1. Có thể định nghĩa được bài toán dưới dạng phối hợp của những bài toán cùng loại nhưng nhỏ hơn hay không ? Khái niệm "nhỏ hơn" là thế nào ?

2. Trường hợp đặc biệt nào của bài toán sẽ được coi là trường hợp tầm thường và có thể giải ngay được để đưa vào phần neo của phép giải đệ quy

**3.3.4. Bài toán Tháp Hà Nội**

Đây là một bài toán mang tính chất một trò chơi, tương truyền rằng tại ngôi đền Benares có ba cái cọc kim cương. Khi khai sinh ra thế giới, thượng đế đặt n cái đĩa bằng vàng chồng lên nhau theo thứ tự giảm dần của đường kính tính từ dưới lên, đĩa to nhất được đặt trên một chiếc cọc.



**Tháp Hà Nội**

Các nhà sư lần lượt chuyển các đĩa sang cọc khác theo luật:

* Khi di chuyển một đĩa, phải đặt nó vào một trong ba cọc đã cho
* Mỗi lần chỉ có thể chuyển một đĩa và phải là đĩa ở trên cùng
* Tại một vị trí, đĩa nào mới chuyển đến sẽ phải đặt lên trên cùng
* Đĩa lớn hơn không bao giờ được phép đặt lên trên đĩa nhỏ hơn (hay nói cách khác: một đĩa chỉ được đặt trên cọc hoặc đặt trên một đĩa lớn hơn)

Ngày tận thế sẽ đến khi toàn bộ chồng đĩa được chuyển sang một cọc khác. Trong trường hợp có 2 đĩa, cách làm có thể mô tả như sau:

Chuyển đĩa nhỏ sang cọc 3, đĩa lớn sang cọc 2 rồi chuyển đĩa nhỏ từ cọc 3 sang cọc 2.

Những người mới bắt đầu có thể giải quyết bài toán một cách dễ dàng khi số đĩa là ít, nhưng họ sẽ gặp rất nhiều khó khăn khi số các đĩa nhiều hơn. Tuy nhiên, với tư duy quy nạp toán học và một máy tính thì công việc trở nên khá dễ dàng:

Có n đĩa.

* Nếu n = 1 thì ta chuyển đĩa duy nhất đó từ cọc 1 sang cọc 2 là xong.
* Giả sử rằng ta có phương pháp chuyển được n - 1 đĩa từ cọc 1 sang cọc 2, thì cách chuyển n - 1 đĩa từ cọc x sang cọc y (1  x, y  3) cũng tương tự.
* Giả sử ràng ta có phương pháp chuyển được n - 1 đĩa giữa hai cọc bất kỳ. Để chuyển n đĩa từ cọc x sang cọc y, ta gọi cọc còn lại là z ( = 6 - x - y). Coi đĩa to nhất là … cọc, chuyển n - 1 đĩa còn lại từ cọc x sang cọc z, sau đó chuyển đĩa to nhất đó sang cọc y và cuối cùng lại coi đĩa to nhất đó là cọc, chuyển n - 1 đĩa còn lại đang ở cọc z sang cọc y chồng lên đĩa to nhất.

Cách làm đó được thể hiện trong thủ tục đệ quy dưới đây:

procedure Move(n, x, y: Integer);

begin

if n = 1 then WriteLn('Chuyển 1 đĩa từ ', x, ' sang ', y)

else

begin

Move(n - 1, x, 6 - x - y);

Move(1, x, y); {Chuyển đĩa to nhất từ x sang y}

Move(n - 1, 6 - x - y, y);

end;

end;

Chương trình chính rất đơn giản, chỉ gồm có 2 việc: Nhập vào số n và gọi Move(n, 1, 2).