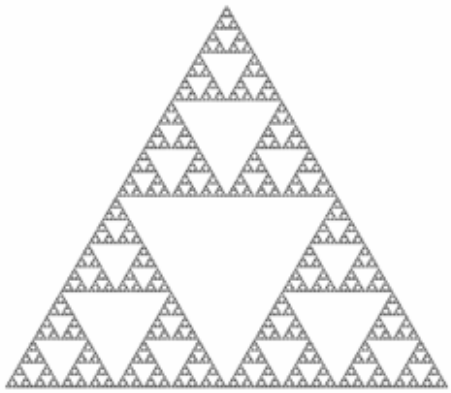
Đệ quy

Bởi:
Khoa CNTT ĐHSP KT Hưng Yên

Khái niệm đệ quy:

Đệ quy (tiếng Anh; Recursion) là một phương pháp dùng trong các chương trình

máy tính trong đó có một hàm tự gọi.



Tam giác Sierpinski

Khái niệm hình thức về đệ quy== đầu Trong toán học và khoa học máy tính, các tính chất (hoặc cấu trúc) được gọi là đệ quy nếu trong đó một lớp các đối tượng hoặc phương pháp được xác định bằng việc xác định một số rất ít các trường hợp hoặc phương pháp đơn giản (thông thường chỉ một) và sau đó xác định quy tắc đưa các trường hợp phức tạp về các trường hợp đơn giản.

Chẳng hạn, định nghĩa sau là định nghĩa đệ quy của tổ tiên:

? Bố mẹ của một người là tổ tiên của người ấy ('trường hợp cơ bản*);*

? Bố mẹ của tổ tiên một người bất kỳ là tổ tiên của người ấy (“ bước đệ quy*).*

Các định nghĩa kiểu như vậy cũng thường thấy trong toán học.

M ột kh á i niệm X đư ợ c định nghĩa theo đệ q u y nếu t r ong định ngh ĩ a X có s ử dụng

ngay chính khái n i ệm X .

? Ví dụ 1: Định nghĩa số *Sốtựnhiên*

- 0 là một số tự nhiên.

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

? Ví dụ 2: Định nghĩa *Hàmgiaithừan!*

- 0! = 1

- Nếu n > 0 thì n! = n(n - 1)!

T h u ật toán đ ệ q u i

Đôi khi chúng ta có thể quy việc giải bài toán với tập các dữ liệu đầu vào xác định về việc giải cùng bài toán đó nhưng với các giá trị đầu vào nhỏ hơn. Chẳng hạn, bài toán tìm UCLN của hai số a, b với a > b có thể rút gọn về bài toán tìm ƯCLN của hai số nhỏ hơn, a mod b và b. Khi việc rút gọn như vậy thực hiện được thì lời giải bài toán ban đầu có thể tìm được bằng một dãy các phép rút gọn cho tới những trường hợp mà ta có thể dễ dàng nhận được lời giải của bài toán. Ta sẽ thấy rằng các thuật toán rút gọn liên tiếp bài toán ban đầu tới bài toán có dữ liệu đầu vào nhỏ hơn, được áp dụng trong một lớp rất rộng các bài toán.

Địnhnghĩa:Một thuật toán được gọi là đệ quy nếu nó giải bài toán bằng cách rút gọn liên tiếp bài toán ban đầu tới bài toán cũng như vậy nhưng có dữ liệu đầu vào nhỏ hơn.

Thídụ:Tìm thuật toán đệ quy tính giá trị an với a là số thực khác không và n là

số nguyên không âm.

Ta xây dựng thuật toán đệ quy nhờ định nghĩa đệ quy của an, đó là an+1=a.an với n>0 và khi n=0 thì a0=1. Vậy để tính an ta quy về các trường hợp có số mũ n nhỏ hơn, cho tới khi n=0.

procedurepower (a: số thực khác không; n: số nguyên không âm)

ifn = 0 **then**power(a,n) := 1

elsepower(a,n) := a * power(a,n-1)

Thídụ:Tìm thuật toán đệ quy tính UCLN của hai số nguyên a,b không âm và a

> b.

procedureUCLN (a,b: các số nguyên không âm, a > b)

ifb = 0 **then**UCLN (a,b) := a

elseUCLN (a,b) := UCLN (a **mod**b, b)

Thídụ12:Hãy biểu diễn thuật toán tìm kiếm tuyến tính như một thủ tục đệ quy.

Để tìm x trong dãy tìm kiếm a1,a2,...,an trong bước thứ i của thuật toán ta so sánh x với ai. Nếu x bằng ai thì i là vị trí cần tìm, ngược lại thì việc tìm kiếm được quy về dãy có số phần tử ít hơn, cụ thể là dãy ai+1,...,an. Thuật toán tìm kiếm có dạng thủ tục đệ quy như sau.

Cho *search*(i,j,x) là thủ tục tìm số x trong dãy ai, ai+1,..., aj. Dữ liệu đầu vào là bộ ba (1,n,x). Thủ tục sẽ dừng khi số hạng đầu tiên của dãy còn lại là x hoặc là khi dãy còn lại chỉ có một phần tử khác x. Nếu x không là số hạng đầu tiên và còn có các số hạng khác thì lại áp dụng thủ tục này, nhưng dãy tìm kiếm ít hơn một phần tử nhận được bằng cách xóa đi phần tử đầu tiên của dãy tìm kiếm ở bước vừa qua.

procedure*search*(i,j,x)

ifai = x **then**loacation := i

elseifi = j **then**loacation := 0

else*search*(i+1,j,x)

Thídụ:Hãy xây dựng phiên bản đệ quy của thuật toán tìm kiếm nhị phân.

Giả sử ta muốn định vị x trong dãy a1, a2, ..., an bằng tìm kiếm nhị phân. Trước tiên ta so sánh x với số hạng giữa a[(n+1)/2]. Nếu chúng bằng nhau thì thuật toán kết thúc, nếu không ta chuyển sang tìm kiếm trong dãy ngắn hơn, nửa đầu của dãy nếu x nhỏ hơn giá trị giữa của của dãy xuất phát, nửa sau nếu ngược lại. Như vậy ta rút gọn việc giải bài toán tìm kiếm về việc giải cũng bài toán đó nhưng trong dãy tìm kiếm có độ dài lần lượt giảm đi một nửa.

procedure*binarysearch*(x,i,j)

m := [(i+j)/2]

ifx = am **then**loacation := m

elseif (x < am and i < m) **then***binarysearch*(x,i,m-1)

elseif (x > am and j > m) **then***binarysearch*(x,m+1,j)

elseloacation := 0

Đ ệ quy v à lặ p :

Thídụ. Thủ tục đệ quy sau đây cho ta giá trị của n! với n là số nguyên dương.

procedure*factorial*(n: positive integer)

ifn = 1 **then***factorial(n)*:= 1

elsefactorial(n) := n * factorial(n-1)

Có cách khác tính hàm giai thừa của một số nguyên từ định nghĩa đệ quy của nó. Thay cho việc lần lượt rút gọn việc tính toán cho các giá trị nhỏ hơn, ta có thể xuất phát từ giá trị của hàm tại 1và lần lượt áp dụng định nghĩa đệ quy để tìm giá trị của hàm tại các số nguyên lớn dần. Đó là thủ tục lặp.

procedureiterative factorial (n: positive integer)

x := 1

fori := 1 **to**n x := i * x

{x là n!}

Thông thường để tính một dãy các giá trị được định nghĩa bằng đệ quy, nếu dùng phương pháp lặp thì số các phép tính sẽ ít hơn là dùng thuật toán đệ quy (trừ khi dùng các máy đệ quy chuyên dụng). Ta sẽ xem xét bài toán tính số hạng thứ n của dãy Fibonacci.

procedure*fibonacci*(n: nguyên không âm)

ifn = 0 **the***fibonacci*(n) := 0

elseif n = 1 **then***fibonacci*(n) := 1

else*fibonacci*(n) := *fibonacci*(n - 1) + *fibonacci*(n - 2)

Theo thuật toán này, để tìm fn ta biểu diễn fn = fn-1 + fn-2. Sau đó thay thế cả hai số này bằng tổng của hai số Fibonacci bậc thấp hơn, cứ tiếp tục như vậy cho tới khi f0 và

f1 xuất hiện thì được thay bằng các giá trị của chúng theo định nghĩa. Do đó để tính fn cần fn+1-1 phép cộng.

Bây giờ ta sẽ tính các phép toán cần dùng để tính fn khi sử dụng phương

pháp lặp. Thủ tục này khởi tạo x là f0 = 0 và y là f1 = 1. Khi vòng lặp được duyệt qua tổng của x và y được gán cho biến phụ z. Sau đó x được gán giá trị của y và y được gán giá trị của z. Vậy sau khi đi qua vòng lặp lần 1, ta có x = f1 và y = f0 + f1 = f2. Khi qua vòng lặp lần n-1 thì x = fn-1. Như vậy chỉ có n – 1 phép cộng được dùng để tìm fn khi n > 1.

procedure*Iterativefibonacci*(n: nguyên không âm)

ifn = 0 **then**y := 0

el s e b egin

x := 0 ; y := 1

fori := 1 **to**n - 1

b egin

e n d

e n d

z := x + y

x := y ; y := z

{y là số Fibonacci thứ n}

Ta đã chỉ ra rằng số các phép toán dùng trong thuật toán đệ quy nhiều hơn khi dùng phương pháp lặp. Tuy nhiên đôi khi người ta vẫn thích dùng thủ tục đệ quy hơn ngay cả khi nó tỏ ra kém hiệu quả so với thủ tục lặp. Đặc biệt, có những bài toán chỉ có thể giải bằng thủ tục đệ quy mà không thể giải bằng thủ tục lặp.