**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN TP.HCM**

**KHOA CÔNG NGHỆ THÔNG TIN**

**HỆ CHÍNH QUI**

**MÔN: KỸ THUẬT LẬP TRÌNH**

**GVLT: ĐẬU NGỌC HÀ DƯƠNG**

****

SINH VIÊN: PHẠM MINH cHIẾN

mSSV: **1612052**

LỚP: 16CTT1

**CÁC KỸ THUẬT KHỬ ĐỆ QUY   
VÀ ÁP DỤNG VÀO BÀI TOÁN CỤ THỂ**

1. **Lập trình đệ quy:**
2. ***Giới thiệu:***

* Một đối tượng mô tả thông qua việc làm việc lại với chính nó được gọi là đệ quy. Trong khía cạnh lập trình, kỹ thuật đệ quy được hiểu đơn giản là “gọi hàm trong hàm”
* Một bài toán có thể giải bằng thuật đệ quy là bài có thể phân tách thành các bài toán nhỏ có cùng tính chất với bài toán ban đầu, các bài toán nhỏ được tách thành các bài toán nhỏ hơn cho tới khi gặp trường hợp đơn giản nhất cho ngay kết quả, từ đó lần ngược lại để cho ra kết quả của bài toán ban đầu.

1. ***Thành phần của một hàm đệ quy:***

* Phần cơ sở: Điều kiện thoát khỏi đệ quy.
* Phần đệ quy: Thân hàm có chứa lời gọi đệ quy.

1. ***Các loại đê quy:***

* Đệ quy tuyến tính
* Đệ quy nhị phân
* Đệ quy phi tuyến
* Đệ quy hỗ tương

1. ***Ví dụ:***

* Các ví dụ hàm có thể dùng đệ quy: tính giai thừa, tính lũy thừa, sắp xếp, tìm số Fibonacy, tính tổ hợp, chỉnh hơp, hoán vị,…
* Các bài toán kinh điển: Bài toán tháp Hà Nội, Bài toán xếp hậu, Bài toán Mã đi tuần, Giải Sudoku, Bài toán Balo,…

1. ***Ưu điểm:***

* Thuận lợi cho việc biểu diễn bài toán, đồng thời làm gọn chương trình khi cài đặt.
* Có thể thực hiện một số lượng lớn các thao tác tính toán thông qua 1 đoạn chương trình ngắn gọn.
* Định nghĩa một tập hợp vô hạn các đối tượng thông qua một số hữu hạn lời phát biểu.

1. ***Nhược điểm:***

* Nhược điểm lớn nhất của kỹ thuật đệ quy là lời giải đệ quy cho một số bài toán có thể bị chạy rất chậm do sự bùng nổ tổ hợp khi kích thước bài toán gia tăng.
* Không phải tất cả các ngôn ngữ lập trình đều có thể cài đặt thuật đệ quy (ví dụ: FORTRAN), chỉ những ngôn ngữ có khả năng tổ chức vùng nhớ kiểu stack mới cài được.

1. **Khử đệ quy**

* Từ những nhược điểm của thuật đệ quy việc thay thế một chương trình đệ quy bằng một chương trình không đệ quy cũng là một vấn đề được quan tâm nhiều trong lập trình.
* Một cách tổng quát người ta chỉ ra rằng: Mọi giải thuật đệ quy đều có thể thay thế bằng một giải thuật không đệ quy. Vấn đề còn gọi là kỹ thuật xây dựng lại giải thuật không đệ quy tương ứng thay thế giải thuật đệ quy. Công việc xây dựng lại một giải thuật không đệ quy không phải bao giờ cũng đơn giản và cho đến nay vẫn chưa có giải pháp thỏa đáng cho trường hợp tổng quát.
* Sơ đồ xây dựng chương trình cho một bài toán khó khi ta không tìm được giải thuật không đệ quy thường là:

+ Dùng quan niệm đệ quy để tìm giải thuật cho bài toán.

+ Mã hóa giải thuật đệ quy.

+ Khử đệ quy để có được một chương trình đệ quy.

* Tuy nhiên do việc khử đệ quy không phải bao giờ cũng dễ và vì vậy trong nhiều trường hợp ta cũng phải chấp nhận sử dụng chương trình đệ quy.

Dưới đây trình bày một số cách khử đệ quy:

1. **Khử đệ quy bằng vòng lặp:**

Áp dụng cho các hàm tính giá trị bằng công thức truy hồi

* *Xét ví dụ hàm tìm ước chung lớn nhất của 2 số nguyên dương a, b:*

|  |  |
| --- | --- |
| **Sử dụng đệ quy** | **Không đệ quy** |
| int timUCLN(int a, int b)  {  if (a == b)  return a;  if (a > b)  return timUCLN(a - b, a);  return timUCLN(a, b - a);  } | int timUCLN2(int a, int b)  {  while (a != b)  {  if (a > b) a -= b;  else b -= a;  }  return a;  } |

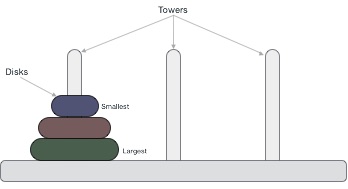
* *Hàm tìm số Fibonacy:*

|  |  |
| --- | --- |
| **Sử dụng đệ quy** | **Không đệ quy** |
| int fibo(int n)  {  if (n < 2)  return n;  return fibo(n-1)+fibo(n-2);  } | int fibo(int n)  {  int a=1, b=2, kq;  if (n == 1) kq= 1;  else if (n == 2) kq = 2;  else  for (int i=1;i<=n-2;i++)  {  kq = a + b;  a = b;  b = kq;  }  return kq;  } |

Như vậy đối với các hàm tính toán có dạng truy hồi, đa phần ta đều có thể cài đặt khử đệ quy bằng vòng lăp.

1. ***Khử đệ quy bằng cấu trúc dữ liệu Stack:***

* Xét bài toán Tháp Hà Nội:



Yêu cầu của bài toán là di chuyển các đĩa có kích cỡ khác nhau sang cột khác sao cho vẫn đảm bảo thứ tự ban đầu của các đĩa: đĩa nhỏ nằm trên đĩa lớn. Dưới đây là một số qui tắc cho trò chơi toán học Tháp Hà Nội (Tower of Hanoi):

* Mỗi lần chỉ có thể di chuyển một đĩa từ cột này sang cột khác.
* Chỉ được di chuyển đĩa nằm trên cùng (không được di chuyển các đĩa nằm giữa).
* Đĩa có kích thước lớn hơn không thể được đặt trên đĩa có kích thước nhỏ hơn

Phân tích bài toán:

Bài toán yêu cầu chuyể n cột từ cột nguồn (I) sang cột thứ ba (III) với cột trung gian là là cột thứ hai (II). Theo yêu cầu bài toán thì ta cần thực hiện các bước sau (giải thuật đệ qui):

* Bước 1: Chuyển n-1 đĩa từ cột (I) sang cột (II)
* Bước 2: Chuyển đĩa thứ n từ cột (I) sang cột (III)
* Bước 3: Chuyển n-1 đĩa từ cột (II) sang cột (III)

Ta thấy tại bước thứ 3 là một sự lặp lại công việc chuyển đĩa như bài toán ban đầu, chỉ có khác ở chỗ là số đĩa và đâu là cột trung gian, đâu là cột đích. Như vậy ta hoàn toàn giải được bài toàn bằng thuật đệ qui. ( Hàm kết thúc tại neo n = 1 đĩa).

|  |
| --- |
| **Tháp Hà Nội – Đệ Qui** |
| /\*c1: cột nguồn  c2: cột trung gian  c3: cột đích\*/  void ThapHaNoi(int n, char c1, char c2, char c3)  {  if (n == 1)  {  printf("Dua dia tu %c qua %c \n", c1, c3);  return;  }  else  {  ThapHaNoi (n - 1, c1, c3, c2);  printf("Dua dia tu %c qua %c \n", c1, c3);  ThapHaNoi(n - 1, c2, c1, c3);  }    } |

* Việc dùng thuật đệ qui như trên giải quyết được bài toán tuy nhiên với số đĩa n lớn thì việc gọi đệ quy nhiều lần như thế tốn rất nhiều thời gian và không gian (bộ nhớ). Như vậy cần phải cải tiến bằng việc khử đệ qui.
* Do đặc điểm của qúa trình xử lý một giải thuật đệ quy là : việc thực thi lời gọi đệ quy

sinh ra lời gọi đệ quy mới cho đến khi gặp trường hợp suy biến (neo) cho nên để thực

thi giải thuật đệ quy cần có cơ chế lưu trử thông tin thỏa các yêu cầu sau :

+ Ở mỗi lần gọi phải lưu trữ thông tin trạng thái con dang dở của tiến trình xử lý ở thời

điểm gọi. Số trạng thái này bằng số lần gọi chưa được hoàn tất.

+ Khi thực hiện xong (hoàn tất) một lần gọi, cần khôi phục lại toàn bộ thông tin trạng thái trước khi gọi

+ Lệnh gọi cuối cùng (ứng với trương hợp neo) sẽ được hoàn tất đầu tiên, thứ tự dãy các lệnh gọi được hoàn tất ngược với thứ tự gọi, tương ứng dãy thông tin trạng thái được hồi phục theo thứ tự ngược với thứ tự lưu trữ.

* Cấu trúc dữ liệu cho phép lưu trữ dãy thông tin thỏa 3 yêu cầu trên là cấu trúc lưu trữ thỏa luật LIFO (Last In First Out). Một kiểu cấu trúc lưu trử thường được sử dụng trong trường hợp này là cấu trúc chồng (stack): bao gồm các thao tác như kiểm tra cấu trúc rỗng, thêm đối tượng vào stack, lấy đối tượng ra khỏi stack,…

|  |
| --- |
| **Cài đặt Stack** |
| #define max 100  struct stack  {  int e[max];  char a[max];  char b[max];  char c[max];  int top;  };  //-------------------  void Init(stack &s)  {  s.top = -1;  }  //-------------------  int isEmptyStack(stack s)  {  return (s.top == -1);  }  //-------------------  void push(stack &s, int &x, char &a, char &b, char &c)  {  if (s.top == max - 1) exit(0);  memcpy(&s.e[++s.top], &x, sizeof(int));  memcpy(&s.a[s.top], &a, sizeof(char));  memcpy(&s.b[s.top], &b, sizeof(char));  memcpy(&s.c[s.top], &c, sizeof(char));  }  //-------------------  void pop(stack &s, int &x, char &a, char &b, char &c)  {  if (s.top == -1) exit(0);  memcpy(&x, &s.e[s.top], sizeof(int));  memcpy(&a, &s.a[s.top], sizeof(char));  memcpy(&b, &s.b[s.top], sizeof(char));  memcpy(&c, &s.c[s.top--], sizeof(char));  } |

|  |
| --- |
| **Hàm xử lí Tháp Hà Nội - Stack** |
| void ThapHaNoi(int n,stack &s, char c1, char c2, char c3)  {  int k = 1;  push(s, n, c1,c2,c3);  while (!isEmptyStack(s))  {  pop(s, n, c1, c2, c3);  if (n == 1)  cout << "\nChuyen " << c1 << " qua " << c3;  else  {  int l = n - 1;  push(s, l, c2, c1, c3);  push(s, k, c1, c2, c3);  push(s, l, c1, c3, c2);  }  }  } |

Như vậy cũng với tư tưởng lập trình của đệ qui nhưng bằng cách cài đặt bằng cấu trúc dữ liệu khác thì vấn đề tồn đọng của việc giải bằng đệ qui được giải quyết, độ phức tập của thuật toán giảm đi rất nhiều so với đệ qui.

Trên đây là 2 cách khử đệ qui căn bản mà em tìm được!

-----------------------------Hết-------------------------------