






Chương 4


Một số cấu trúc dữ liệu và giải thuật căn bản


Phần 4.1 Độ qui (4LT – 2BT)

1. Độ qui

-  1.1 Khái niệm về độ qui
-  1.2 Các loại độ qui
-  1.3 Mô tả độ qui các cấu trúc dữ liệu
-  1.4 Mô tả độ qui các giải thuật
-  1.5 Các dạng độ qui đơn giản thường gặp

Khái niệm Đ/n đệ qui

 Một mô tả/định nghĩa về một đối tượng gọi là đệ qui nếu trong mô tả/định nghĩa đó ta lại sử dụng chính đối tượng này.

 Tức là mô tả đối tượng qua chính nó.

● Mô tả đệ qui tập số tự nhiên N :

- Số 1 là số tự nhiên ($1 - N$)
- Số tự nhiên bằng số tự nhiên cộng 1.

● Mô tả đệ qui cấu trúc ds(list) kiểu T :

- Cấu trúc rỗng là một ds kiểu T .
- Ghép nối một thành phần kiểu T (nút kiểu T) với một ds kiểu T ta có một ds kiểu T .

● Mô tả đệ qui cây gia phả: Gia phả của một người bao gồm người đó và gia phả của cha và gia phả của mẹ

Ví dụ

❏ Định nghĩa không đệ qui n!:

● $n! = n * (n-1) * \dots * 1$

❏ Định nghĩa đệ qui:

●
$$n! = \begin{cases} 1 & \text{nếu } n=0 \\ n * (n-1)! & \text{nếu } n>0 \end{cases}$$

❏ Mã C++:

```
int factorial(int n) {  
    if (n==0)    return 1;  
    else        return (n * factorial(n - 1));  
}
```

❏ Mô tả đệ qui thủ tục sắp tăng dãy

● $a[m:n]$ (dãy $a[m], a[m+1], \dots, a[n]$) bằng phương pháp Sort_Merge (SM):

$SM(a[m:n]) \equiv Merge(SM(a[m : (n+m) \div 2]), SM(a[(n+m) \div 2 + 1 : n]))$

➤ Với : $SM(a[x : x])$ là thao tác rỗng (không làm gì cả).

➤ Merge ($a[x : y]$, $a[(y+1) : z]$) là thủ tục trộn 2 dãy tăng $a[x : y]$, $a[(y+1) : z]$ để được một dãy $a[x : z]$ tăng.

Mô tả đệ qui gồm hai phần

- Phần neo: trường hợp suy biến (cá biệt) của đối tượng

Ví dụ: *1 là số tự nhiên, cấu trúc rỗng là danh sách kiểu T, $0 \neq 1, \dots$*

- Phần qui nạp: mô tả đối tượng (giải thuật) thông qua chính đối tượng (giải thuật) đó một cách trực tiếp hoặc gián tiếp.

Ví dụ:

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

$$SM(a[m:n]) \equiv Merge(SM(a[m:(m+n) \div 2]), SM(a[(m+n) \div 2 + 1 : n]))$$

Đệ qui gồm hai loại:

- Đệ qui trực tiếp
- Đệ qui gián tiếp

Giải thuật đệ qui

❏ Nếu ta có 1 lời giải S cho bài toán P, ta lại sử dụng lời giải ấy cho bài toán P' giống P nhưng kích cỡ nhỏ hơn thì lời giải S đó gọi là lời giải đệ qui.

❏ Biểu diễn giải thuật đệ qui

$$P \iff P[S, P]$$

Điều kiện dừng

❏ Biểu diễn tổng quát

$$P \iff \text{if } B \text{ } P[S, P]$$

$$\text{hoặc } P \iff P[S, \text{if } B \text{ } P]$$

❏ Chương trình đệ qui: Khi ta cài đặt giải thuật đệ qui, ta có chương trình đệ qui (tự nó gọi lại chính nó: $P \Rightarrow P'$)

–Hàm đệ qui

–Thủ tục đệ qui

Mô tả đệ qui các giải thuật

📖 Dãy số Fibonacci(FIBO) :{ FIBO (n) } $\equiv 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, \dots$

● FIBO(0) = FIBO (1) = 1 ;

● FIBO(n) = FIBO (n -1) + FIBO (n -2) ; với $n \geq 2$

📖 Giải thuật đệ qui tính FIBO (n) là:

FIBO(n)

if ((n = 0) or (n = 1)) return 1 ;

else return (FIBO (n -1) + FIBO (n -2)) ;

Các dạng đệ qui đơn giản thường gặp

📖 Đệ qui tuyến tính: là dạng đệ qui trực tiếp đơn giản nhất có dạng

```
P () {  
    If (B) thực hiện S;  
    else { thực hiện S* ; gọi P }  
}
```

➤ Với S , S* là các thao tác không đệ qui .

➤ Ví dụ: Hàm FAC(n) tính số hạng n của dãy n!

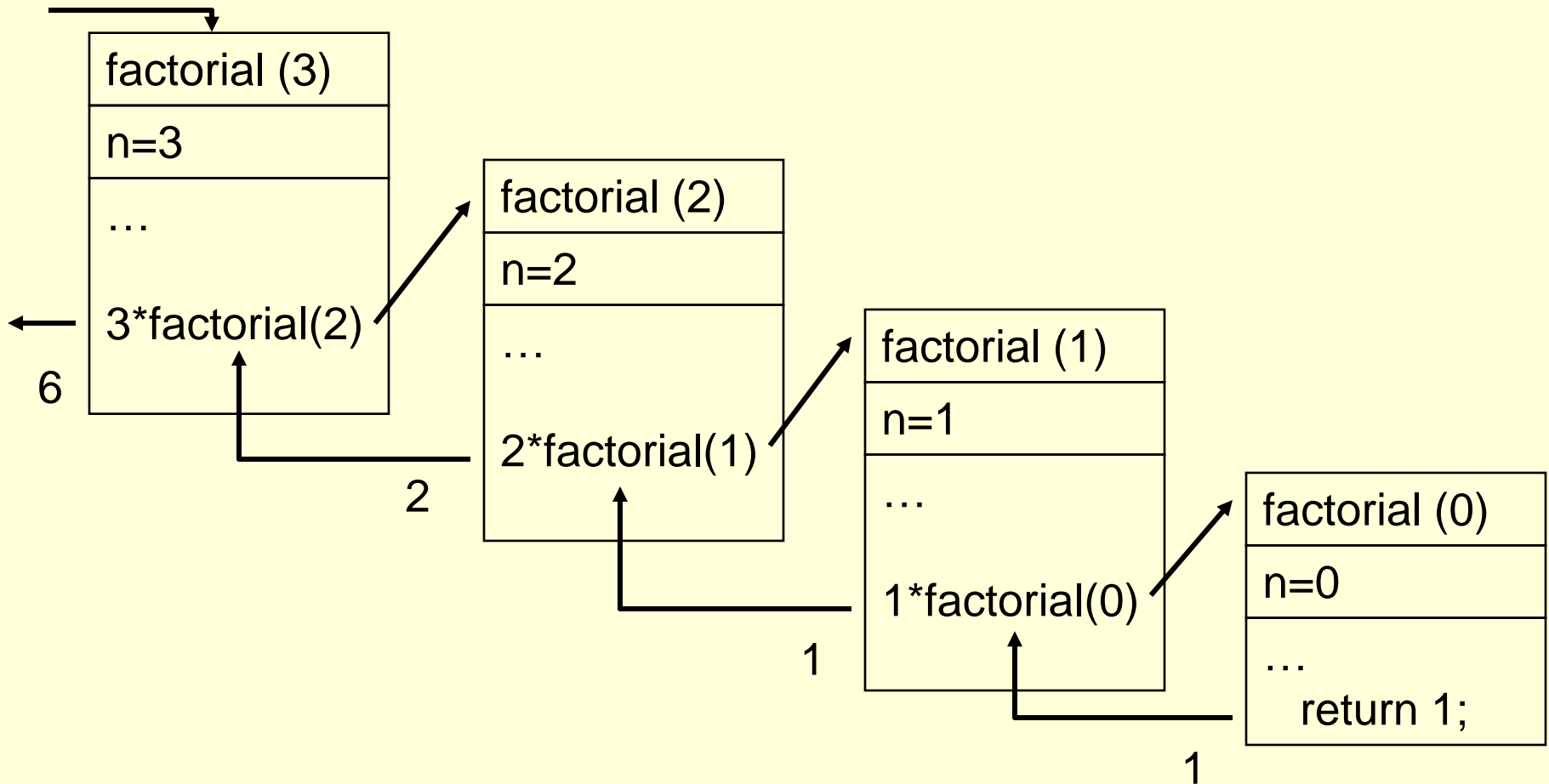
Dạng hàm trong ngôn ngữ mã giả:

```
{ Nếu n = 0 thì FAC = 1 ; /* trường hợp neo*/  
  Ngược lại FAC = n * FAC(n-1) }
```

Dạng hàm trong C++ :

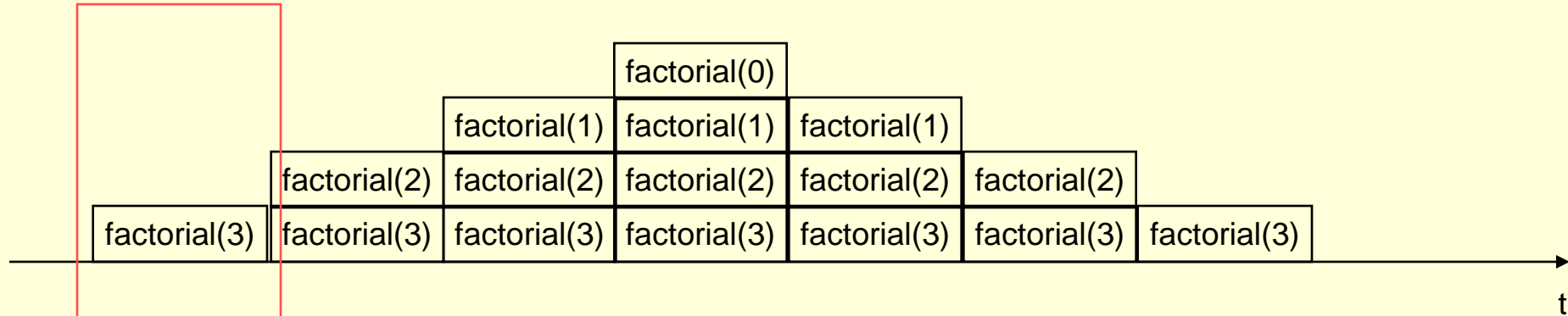
```
int FAC( int n )  
{  
    if ( n == 0 ) return 1 ;  
    else return ( n * FAC(n-1) );  
}
```


Thi hành hàm tính giai thừa

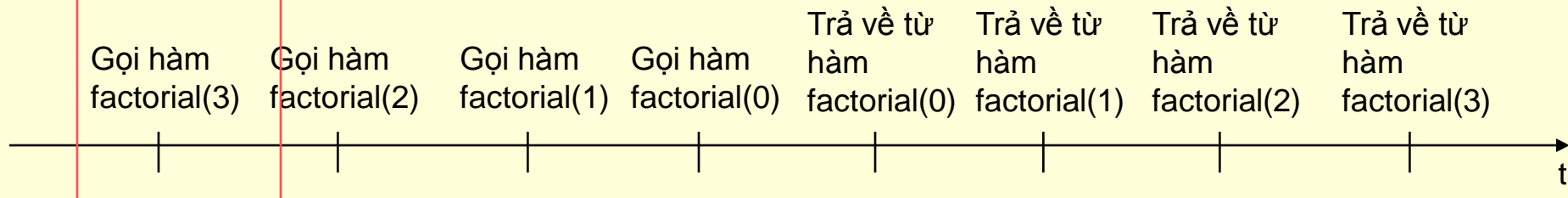


Trạng thái hệ thống khi thi hành hàm tính giai thừa

Stack hệ thống



Thời gian hệ thống



Các dạng đệ qui đơn giản thường gặp (tiếp)

📖 Đệ qui nhị phân: là đệ qui trực tiếp có dạng như sau

```
P () {  
    If (B) thực hiện S;  
    else { thực hiện S* ; gọi P ; gọi P...}  
}
```

- Với S , S* là các thao tác không đệ qui .

- Ví dụ: Hàm FIBO(n) tính số hạng n của dãy FIBONACCI

Dạng hàm trong C++ :

```
int F(int n)  
{ if ( n < 2 ) return 1 ;  
  else return (F(n -1) + F(n -2)) ; }
```

Các dạng đệ qui đơn giản thường gặp (tiếp)

▣ Đệ qui phi tuyến: là đệ qui trực tiếp mà lời gọi đệ qui được thực hiện bên trong vòng lặp.

```
P () {  
  for (<giá trị đầu> to <giá trị cuối>)  
  {  
    thực hiện S ;  
    if ( thỏa điều kiện dừng ) then thực hiện S*;  
    else gọi P; }  
}
```

● Với S , S* là các thao tác không đệ qui .

Ví dụ: Cho dãy { A_n } xác định theo công thức truy hồi :

$$A_0 = 1 ; A_n = n^2 A_0 + (n-1)^2 A_1 + \dots + 2^2 A_{n-2} + 1^2 A_{n-1}$$

Dạng hàm đệ qui tính A_n trên ngôn ngữ C++ là:

```
int A( int n ) {  
  if ( n == 0 ) return 1 ;  
  else {  
    int tg = 0 ;  
    for (int i = 0 ; i < n ; i++ ) tg = tg + sqr(n-i) * A(i);  
    return ( tg ) ;  
  }  
}
```

3 bước để tìm giải thuật đệ qui

Tham số hóa bài toán .

- Tổng quát hóa bài toán cụ thể cần giải thành bài toán tổng quát (một hoặc các bài toán chứa bài toán cần giải)
- Tìm ra các tham số cho bài toán tổng quát
 - Các tham số điều khiển: các tham số mà độ lớn của chúng đặc trưng cho độ phức tạp của bài toán, và giảm đi qua mỗi lần gọi đệ qui.
 - Ví dụ
n trong hàm $FAC(n)$;
a , b trong hàm $USCLN(a,b)$.

Tìm các trường hợp *neo* (“Trivial”) cùng giải thuật giải tương ứng


- trường hợp suy biến của bài toán tổng quát
- các trường hợp tương ứng với các giá trị biên của các biến điều khiển

Vd : $FAC(1) = 1$

$USCLN(a,b) = b$ nếu a chia hết cho b

Tìm giải thuật giải trong trường hợp tổng quát bằng phân rã bài toán theo kiểu đệ qui.

3 bước (tiếp)

 Phân rã bài toán tổng quát theo phương thức đệ qui

● Phân rã bài toán thành các bài toán giống BT




Top-Down!

● Ví dụ

$$\text{FAC}(n) = n * \text{FAC}(n - 1) .$$

$$\text{Tmax}(a[1:n]) = \max(\text{Tmax}(a[1:(n-1)]) , a[n])$$

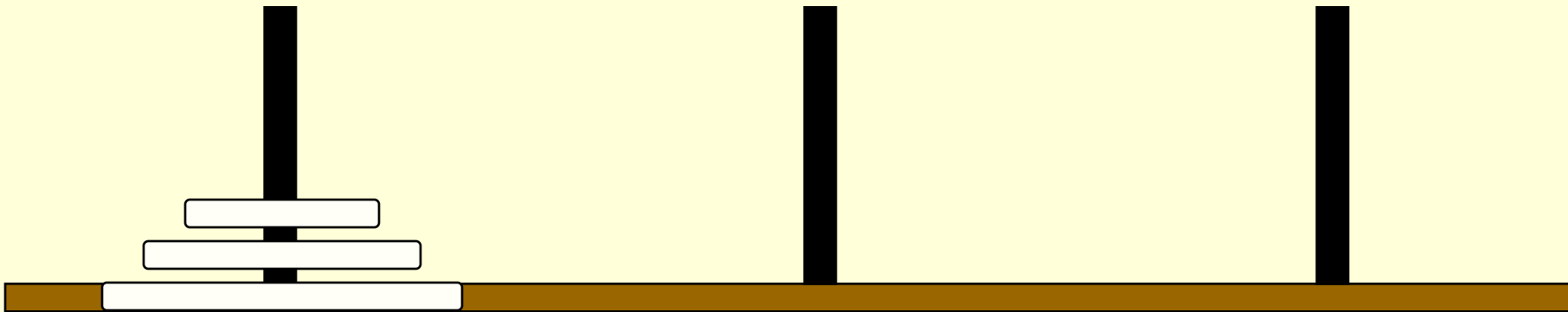
Một số bài toán giải bằng đệ qui

-  Bài toán tháp Hà Nội
-  Bài toán chia phần thưởng
-  Bài toán hoán vị

Bài toán Tháp Hà nội

 Luật:

- Di chuyển mỗi lần một đĩa
- Không được đặt đĩa lớn lên trên đĩa nhỏ



Với chồng gồm n đĩa cần $2^n - 1$ lần chuyển

–Giả sử thời gian để chuyển 1 đĩa là t giây thì thời gian để chuyển xong chồng 64 đĩa sẽ là:

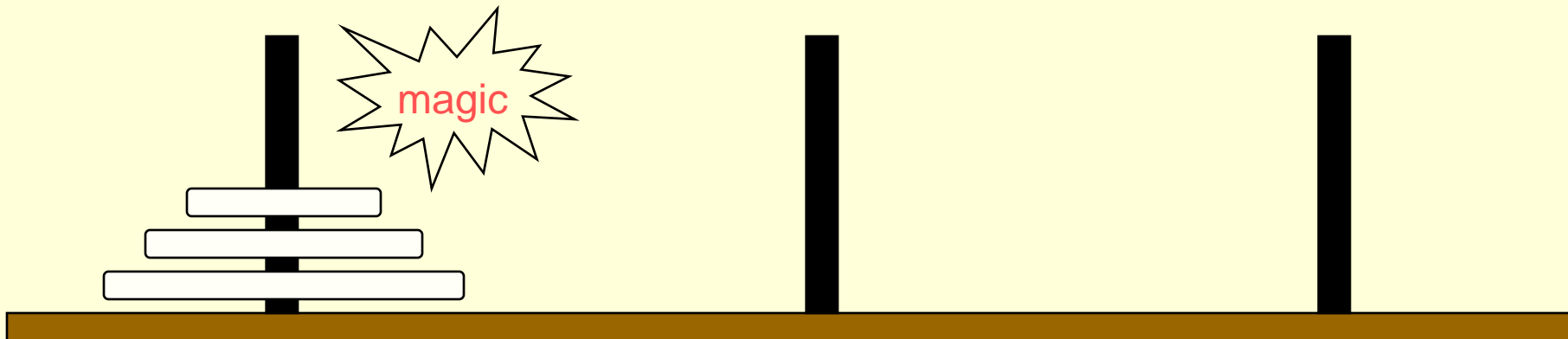
– $T = (2^{64} - 1) * t = 1.84 \cdot 10^{19} t$

–Với $t = 1/100$ s thì $T = 5.8 \cdot 10^9$ năm = 5.8 tỷ năm .

Bài toán Tháp Hà nội

Hàm đệ qui:

- Chuyển $(n-1)$ đĩa trên đỉnh của cột start sang cột temp
- Chuyển 1 đĩa (cuối cùng) của cột start sang cột finish
- Chuyển $n-1$ đĩa từ cột temp sang cột finish



Bài toán Tháp Hà nội

Giải bài toán bằng đệ qui

● Thông số hóa bài toán

- Xét bài toán ở mức tổng quát nhất : chuyển n ($n \geq 0$) đĩa từ cột A sang cột B lấy cột C làm trung gian .
- $THN(n, A, B, C) \rightarrow$ với 64 đĩa gọi $THN(64, A, B, C)$
- n sẽ là thông số quyết định bài toán – n là tham số điều khiển

● Trường hợp suy biến và cách giải

- Với $n = 1$: $THN(1, A, B, C)$
 - Giải thuật giải bt $THN(1, A, B, C)$ là thực hiện chỉ 1 thao tác cơ bản : Chuyển 1 đĩa từ A sang B (ký hiệu là $Move(A, B)$) .
- $THN(1, A, B, C) \equiv \{ Move(A, B) \}$.
- $THN(0, A, B, C) \equiv \{ \varnothing \}$

Bài toán Tháp Hà nội

Phân rã bài toán

- Ta có thể phân rã bài toán TH N (k, A, B, C) : chuyển k đĩa từ cột A sang cột B lấy cột C làm trung gian thành dãy tuần tự 3 công việc sau :

- Chuyển ($k - 1$) đĩa từ cột A sang cột C lấy cột B làm trung gian :
 - THN ($k - 1, A, C, B$) (bài toán THN với $n = k - 1, A = A, B = C, C = B$)
- Chuyển 1 đĩa từ cột A sang cột B : Move (A, B) (thao tác cơ bản).
- Chuyển ($k - 1$) đĩa từ cột C sang cột B lấy cột A làm trung gian :
 - THN ($k - 1, C, B, A$) (bài toán THN với $n = k - 1, A = B, B = A, C = C$) .

Vậy giải thuật trong trường hợp tổng quát ($n > 1$) là:

THN(n, A, B, C)

```
{  
    THN ( $n - 1, A, C, B$ ) ;  
    Move ( $A, B$ ) ;  
    THN ( $n - 1, C, B, A$ ) ;  
}
```

Bài toán Tháp Hà nội – Mã C++

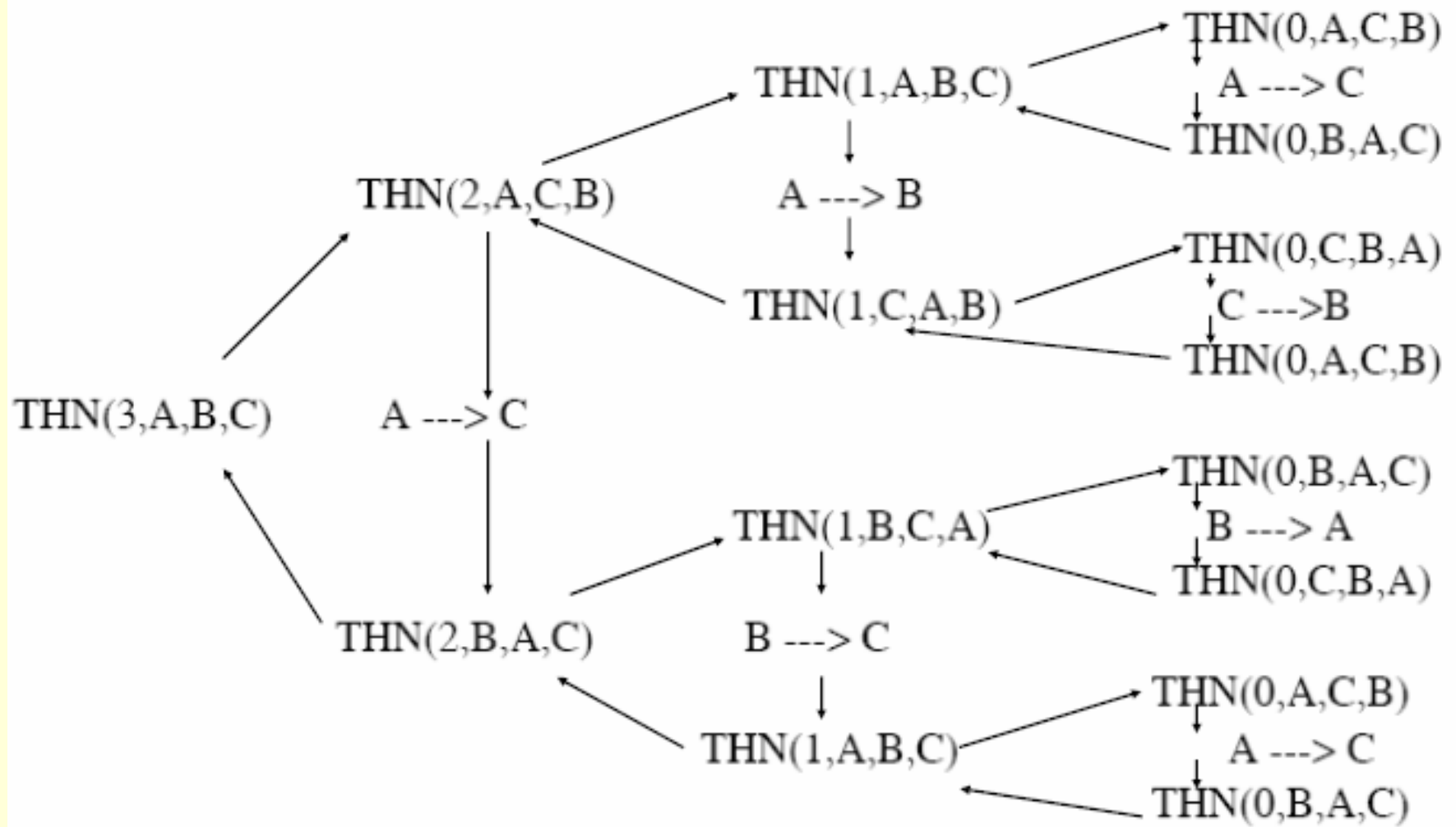
```
void move(int count, int start, int finish, int temp) {  
    if (count > 0) {  
        move(count - 1, start, temp, finish);  
        cout << "Move disk " << count << " from " << start  
            << " to " << finish << "." << endl;  
        move(count - 1, temp, finish, start);  
    }  
}
```

Lời gọi c/0



Lời gọi c/1

Lời gọi c/2

Lời gọi c/3



Bài toán chia phần thưởng

-  Có **100 phần thưởng** đem chia cho **12 học sinh giỏi đã được xếp hạng**. Có bao nhiêu cách khác nhau để thực hiện cách chia?
-  Tìm giải thuật giải bài toán bằng phương pháp đệ quy.

Bài toán chia phần thưởng (tự đọc)

Giải bài toán bằng đệ qui

- Nhìn góc độ bài toán tổng quát: Tìm số cách chia m vật (phần thưởng) cho n đối tượng (học sinh) có thứ tự.

- $PART(m, n)$
- n đối tượng đã được sắp xếp $1, 2, \dots, n$
- S_i là số phần thưởng mà i nhận được

$$S_i \geq 0$$

$$S_1 \geq S_2 \geq \dots \geq S_n.$$

$$S_1 + S_2 + \dots + S_n = m$$

- Ví dụ:

Với $m = 5$, $n = 3$ ta có 5 cách chia sau :

5 0 0, 4 1 0, 3 2 0, 3 1 1, 2 2 1

Tức là $PART(5, 3) = 5$

Các trường hợp suy biến

- $m = 0$: mọi học sinh đều nhận được 0 phần thưởng .
 $PART(0, n) = 1$ với mọi n
- $n = 0, m \neq 0$: không có cách chia
 $PART(m, 0) = 0$ với mọi $m \neq 0$.

Phân rã bài toán trong trường hợp tổng quát

- $m < n$: $n - m$ học sinh xếp cuối sẽ luôn không nhận được gì cả trong mọi cách chia .
Vậy: $n > m$ thì $PART(m, n) = PART(m, m)$
- $m \geq n$: là tổng
 - Học sinh cuối cùng không có phần thưởng
 $PART(m, n - 1)$
 - Học sinh cuối cùng có ít nhất 1
 $PART(m - n, n)$Vậy: $m > n \Rightarrow PART(m, n) = PART(m, n - 1) + PART(m - n, n)$

Dạng hàm PART trong NN LT C++

```
int PART( int m , int n ) {  
    if ((m == 0 ) || (n == 0) ) return 1 ;  
    else if(m < n ) retrun ( PART(m , m )) ;  
    else  
        return ( PART(m , n -1 ) + PART( m -n , n ) ) ;  
}
```

Bài toán tìm tất cả hoán vị của một dãy các phần tử (giảng lướt)

Thông số hóa bài toán.

- Gọi $HV(v, m)$

- v : array[1 .. N] of T

- m : integer ; $m \leq N$

- T là một kiểu dữ liệu

- Ví dụ: $N = 4$, $A[1] = 1$, $A[2] = 2$, $A[3] = 3$, $A[4] = 4$ thì lời gọi $HV(A, 3)$ xuất tất cả hoán vị của A có được bằng cách hoán vị 3 phần tử đầu (có 6 h vị) :

1 2 3 4

2 1 3 4

1 3 2 4

3 1 2 4

3 2 1 4

2 3 1 4

Trường hợp neo

i m = 1 : $HV(v, 1)$: xuất v

$HV(v, 1) \equiv \text{Display}(v)$ //for (k= 1 to N do) Display(v[k])

Phân rã bài toán

- ❏ Giữ nguyên các phần tử cuối $V[m], \dots, V[N]$ hoán vị $m-1$ phần tử đầu
 - gọi đệ quy $HV(V, m-1)$
- ❏ Đổi chỗ $V[m]$ cho $V[m-1]$, giữ nguyên các phần tử cuối $V[m], \dots, V[N]$ hoán vị $m-1$ phần tử đầu
 - gọi đệ quy $HV(V, m-1)$
- ❏ Đổi chỗ $V[m]$ cho $V[m-2]$, giữ nguyên các phần tử cuối $V[m], \dots, V[N]$ hoán vị $m-1$ phần tử đầu
 - gọi đệ quy $HV(V, m-1)$
- ❏
 -
 -
 - ❏ Đổi chỗ $V[m]$ cho $V[2]$, giữ nguyên các phần tử cuối $V[m], \dots, V[N]$ hoán vị $m-1$ phần tử đầu
 - gọi đệ quy $HV(V, m-1)$.
 - ❏ - Đổi chỗ $V[m]$ cho $V[1]$, giữ nguyên các phần tử cuối $V[m], \dots, V[N]$ hoán vị $m-1$ phần tử đầu
 - gọi đệ quy $HV(V, m-1)$.

Bài toán tìm tất cả hoán vị của một dãy các phần tử

```
HV(V,m) ≡{  
    SWAP( V[m],V[m] ) ; HV(V,m -1) ;  
    SWAP( V[m],v[m-1] ) ; HV(V,m -1) ;  
    SWAP( V[m],v[m-2 ] ) ; HV(V,m -1) ;  
    .....  
    .....  
    SWAP (V[m],v[2] ) ; HV(V,m -1) ;  
    SWAP( V[m],v[1] ) ; HV(V,m -1) ;  
}
```

SWAP(x , y) là thủ tục hoán đổi giá trị của 2 đối tượng dữ liệu x ,y)

```
Vậy :HV(V , m ) ≡ for (k = m;k>0; k--) {  
    SWAP( V[m], V[k] ) ;  
    HV(V,m -1) ;  
} ;
```

***const size = Val ; // Val là hằng giá trị
typedef typebase vector[size] ; // typebase là một kiểu dữ liệu có thứ tự***

***void Swap(typebase &x , typebase &y) {
 typebase t ;
 t = x ; x = y ; y = t ;
}***

***void print(const vector &A) {
 for(int j= 0 ; j <size ; j++) cout<< A[j] ;
 cout << “....”;
}***

















***void HV(const vector &V , int m) {
 if (m == 1) print(V);
 else for(int k = m-1 ; k >= 0 ; k--) {
 swap(V[m-1] , V[k]) ;
 HV(V,m-1) ;
 }
}***

Lời giảiĐệ qui khác của bài toán Hoán vị (giảng cho SV)

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#define STOP 27 // phim esc
const MAX =100;
int a[MAX];
char b[MAX];// mang danh dau
int shv, n;

Hoanvi (int k)
{ int i,j;
  for (i=1;i<=n;i++)
    if (b[i])
      { a[k] = i;
        b[i] = 0;
        if (k==n)
          { // Da chon du n so => 1 hoan vi
            shv ++;
            printf("\n Hoan vi %2d la : ", shv);
            for (j=1; j <=n;j++)
              printf("%d", a[j]);
          }
        else // goi tang len 1 so
          Hoanvi (k+1);
      }
    b[i] = 1;
  }
  return 0;
}
```

Lời giả không đệ qui của bài toán Hoán vị (tiếp)

```
 main ()  
 { char ch;  
 int i;  
 do  
 { // danh dau mang b  
   for (i=1;i <= MAX;i++)    b[i] = 1;  
   clrscr();  
   printf("\n n=");  
   scanf("%d", &n);  
   shv = 0;  
   if ( n > 0)  
       Hoanvi(1);  
   printf("\n Lam tiep :Y Thoat: ESC>");  
   ch = getch();  
   } while (ch !=STOP);  
 }
```

KHỬ ĐỆ QUI

- 1 Cơ chế thực hiện đệ qui
- 2 Tổng quan về khử đệ qui
- 3 Các trường hợp khử đệ qui đơn giản

Cơ chế thực hiện đệ qui

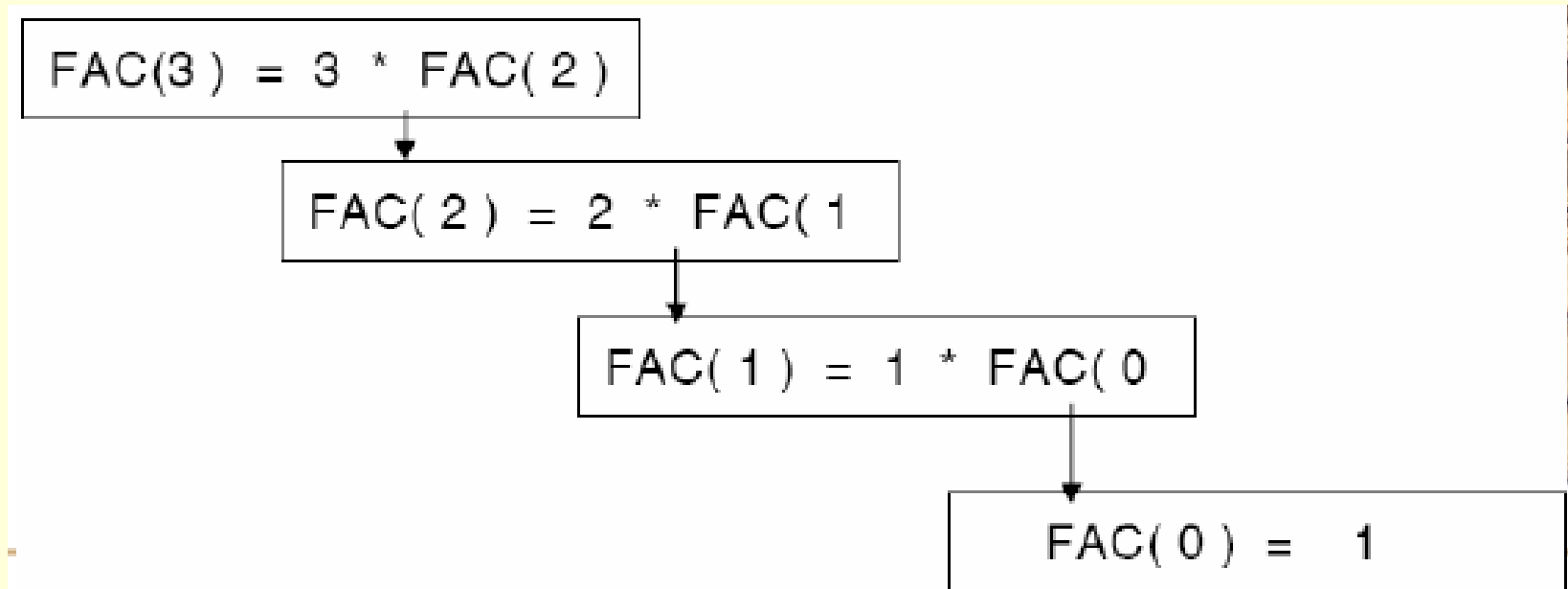
- Trạng thái của tiến trình xử lý một giải thuật: nội dung các biến và lệnh cần thực hiện kế tiếp.
- Với tiến trình xử lý một giải thuật đệ qui ở từng thời điểm thực hiện, cần lưu trữ cả các trạng thái xử lý đang còn dang dở .

Xét giải thuật giai thừa

–Giải thuật

*FAC (n) \equiv if(n = 0) return 1;
else return (n * FAC (n – 1));*

–Sơ đồ thực hiện



Xét thủ tục đệ qui tháp HàNội THN (n , X , Y , Z)

- *trung gian*

– Giải thuật :

THN (n, X ,Y , Z)

```
{  
  if (n > 0 ) {  
    THN(n-1,X ,Z ,Y) ;  
    Move(X, Y)  
    THN(n-1,Z,Y,X) ;  
  }  
}
```

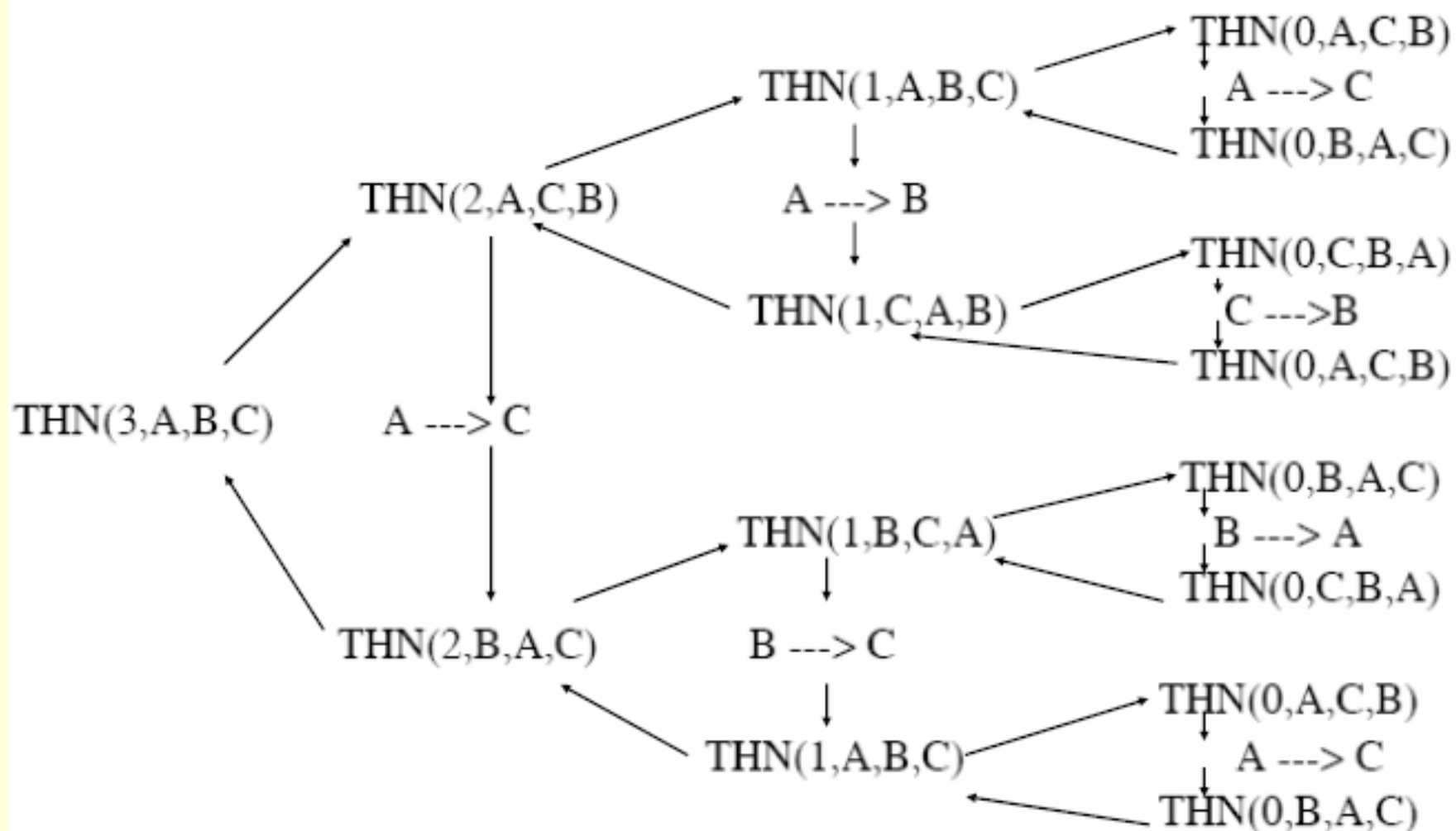
– Sơ đồ thực hiện THN(3,A,B,C)

Lời gọi c/0

Lời gọi c/1

Lời gọi c/2

Lời gọi c/3



Nhận xét

- Lời gọi đệ qui sinh ra lời gọi đệ qui mới cho đến khi gặp trường hợp suy biến (neo)
- 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 trường hợp neo) sẽ được hoàn tất đầu tiên.
- ng: cấu trúc Stack (LIFO)

Tạo ngăn xếp S

- Thủ tục Creatstack(S) : Tạo S rỗng .
- Thủ tục Push(x,S) : thêm x vào đỉnh stack S
 - (x là dữ liệu kiểu đơn giản hoặc có cấu trúc)
- Thủ tục Pop(x,S) : Lấy giá trị đang lưu ở đỉnh S
 - Lưu trữ vào x
 - Loại bỏ giá trị này khỏi S (lùi đỉnh S xuống một mức)
- Hàm Empty(S): (kiểu boolean) Kiểm tra tính rỗng của S : cho giá trị đúng nếu S rỗng , sai nếu không.

Cai dat stack :

```
#define MAX 100
```

```
typedef struct {
```

```
    int top;
```

```
    int nodes(MAX);
```

```
} stack;
```

```
int Empty(stack *ps) {
```

```
    if (ps->top == - 1)
```

```
        return (true);
```

```
    return(false);
```

```
}
```

```

void Push(stack *ps , int x) {
    if (ps->top ==MAX -1) {
        printf("\n stack full");
        return;
    }
    ps->top +=1 ;
    ps->nodes[ps->top] =x;
}

int Pop(stack *ps) {
    if(Empty(ps) {
        printf("\n stack is empty");
        return (0);
    }
    return(ps->nodes[ps->top--]);
}

```


Tổng quan về khử đệ qui

t code

- Nhược điểm: tốn không gian nhớ và thời gian xử lý.
- Mọi giải thuật đệ qui đều có thể thay thế bằng một giải thuật không đệ qui.
- Sơ đồ xây dựng chương trình cho một bài toán khó khi ta không tìm được giải thuật không đệ qui thường là:
 - i) Dùng quan niệm đệ qui để tìm giải thuật cho bài toán .
 - ii) Mã hóa giải thuật đệ qui .
 - ii) Khử đệ qui để có được một chương trình không đệ qui .
- Tuy nhiên : khử đệ qui không phải bao giờ cũng dễ => trong nhiều trường hợp ta cũng phải chấp nhận sử dụng chương trình đệ qui.

1. Khử đệ qui bằng vòng lặp

A. *a trị của dãy dữ liệu mô tả bằng hồi qui*

• Ý tưởng

- Xét một vòng lặp trong đó sử dụng 1 tập hợp biến $W = (V, U)$
 - i) Tập hợp U các biến bị thay đổi
 - ii) V là các biến còn lại.
- Dạng tổng quát của vòng lặp là: **$W = W_0 ; \{ W_0 = (U_0, V_0) \}$**
 $\text{while } C(U) \text{ do } U := g(W)$
- Gọi U_0 là trạng thái của U ngay trước vòng lặp
- U_k với $k > 0$ là trạng thái của U sau lần lặp thứ k (giả sử còn lặp đến lần k)
 - U_0 mang các giá trị được gán ban đầu
 - $U_k = g(W) = g(U_{k-1}, V_0) = f(U_{k-1})$ với $k = 1 \dots n$
 - Với n là lần lặp cuối cùng, tức $C(U_k)$ đúng với mọi $k < n$, $C(U_n)$ sai
- Sau vòng lặp W mang nội dung (U_n, V_0) .

Giải thuật hồi qui thường gặp

$$f(n) = C \text{ khi } n = n_0 \text{ (} C \text{ là một hằng)}$$
$$= g(n, f(n-1)) \text{ khi } n > n_0$$

• Ví dụ:

- Hàm giai thừa FAC $(n) = n! = 1$ khi $n = 0$
 $= n * \text{FAC}(n-1)$ khi $n > 0$

- Tổng n số đầu tiên của dãy đan dấu sau :

$$S_n = 1 - 3 + 5 - 7 \dots + (-1)^{n+1} * (2n-1)$$

$$S(k) = \begin{cases} 1 & \text{khi } k = 1 \\ S(k-1) + (-1)^{k+1} * (2*k-1) & \text{với } k > 1 \end{cases}$$

.Giải thuật đệ qui tính giá trị $f(n)$

$f(n) = \text{if}(n = n_0) \text{ return } C ;$
 $\text{else return } (g(n, f(n - 1))) ;$

•Giải thuật lặp tính giá trị $f(n)$

$k := n_0 ; F := C ;$
 $\{ F = f(n_0) \}$
 $\text{while}(k < n) \{$
 $k := k + 1 ;$
 $F := g(k, F) ;$
 $\}$
 $\text{return } F;$

•Trong trường hợp này :

$W = U = (k , F)$
 $W_0 = U_0 = (n_0, C)$
 $C(U) = (k < n)$
 $f(W) = f(U) = f(k, F) = (k+1, g(k, F))$

Khử đệ qui với hàm tính giai thừa

```
long int FAC ( int n ) {  
    int k = 0 ;  
    long int F = 1 ;  
    while ( k < n ) F = ++k * F ;  
    return (F) ;  
}
```

Với hàm tính S(n)

```
int S ( int n ) {  
    int k = 1 , tg = 1 ;  
    while ( k < n ) {  
        k ++ ;  
        if (k%2) tg += 2 * k +1 ;  
        else tg -= 2 * k + 1 ;  
    }  
    return ( tg ) ;  
}
```

2. Các thủ tục đệ qui dạng đệ qui đuôi

- Xét thủ tục P dạng :

```
P(X) ≡ if B(X) then D(X)
      else {
            A(X) ;
            P(f(X)) ;
          }
```

- Trong đó: X là tập biến (một hoặc một bộ nhiều biến).
- P(X) là thủ tục đệ qui phụ thuộc X
- A(X) ; D(X) là các thao tác không đệ qui
- f(X) là hàm biến đổi X

- Xét quá trình thi hành $P(X)$:
 - gọi P_0 là lần gọi P thứ 0 (đầu tiên) $P(X)$
 - P_1 là lần gọi P thứ 1 (lần 2) $P(f(X))$
 - P_i là lần gọi P thứ i (lần $i + 1$) $P(f(f(...f(X)...))$
 - ($P(f_i(X))$) hợp i lần hàm f)
- Gọi P_i nếu $B(f_i(X))$
 - (false) { A và gọi P_{i+1} }
 - (true) { D }
- Giả sử P được gọi đúng $n + 1$ lần . Khi đó ở trong lần gọi cuối cùng (thứ n) P_n thì $B(f_n(X)) = \text{true}$, lệnh D được thi hành và chấm dứt thao tác gọi thủ tục P .

Sơ đồ thực hiện giải thuật trên bằng vòng lặp:

```
while ( ! B(X) ) {
    A(X) ;
    X = f(X) ;
}
D(X) ;
```

Ví dụ:

Đổi 1 số nguyên không âm Y ở cơ số 10 sang dạng cơ số k ($2 \leq k \leq 9$)

–Dùng mảng $A[n]$

–Convert(x, m) để tạo dãy giá trị: $A[0]$, $A[1]$, . . . , $A[m]$

–Giải thuật

```
Convert(n,m)  $\equiv$  if  $n \neq 0$  {  
     $A[m] = n \% k$  ;  
    Convert( $n/k$  ,  $m - 1$ ) ;  
}
```

–Trong ví dụ này ta có

- X là (n, m) ;
- $B(X)$ là biểu thức boolean $\text{not}(n \neq 0)$
- $A(X)$ là lệnh gán $A[m] := n \% k$;
- $f(X)$ là hàm $f(n, m) = (n/k, m - 1)$;
- $D(X)$ là lệnh rỗng

–Đoạn lệnh lặp tương ứng với thủ tục Convert(x, m) là:

```
while (n != 0) {  
     $A[m] = n \% k$  ; //  $A(X)$   
     $n = n / k$  ; //  $X := f(X)$   
     $m = m - 1$  ;  
}
```


Ví dụ: Tìm ước số chung lớn nhất của hai số

- Giải thuật đệ qui

$$\text{USCLN}(m, n, \text{var } us) \equiv \text{if } (n = 0) \text{ then } us := m \\ \text{else } \text{USCLN}(n, m \bmod n, us);$$

=> t trên c

unsigned USCLN2(int a, int b)

```
{ // Dùng đệ qui theo định nghĩa của USCLN
  if ((a % b) == 0) return b;
  else return USCLN2(b, a % b);
}
```

qui

```
unsigned USCLN1(int a, int b)
{
    // Dùng vòng lặp theo thuật toán O'clide
    while (a != b)
    {
        if (a > b) a-=b;
        else b-=a;
    }
    return b;
}
```

ng Stack

- Để thực hiện một chương trình con đệ qui thì hệ thống phải tổ chức vùng lưu trữ thỏa qui tắc LIFO (Stack).=> So sánh với gọi CTC thông thường.
- Vậy ta chủ động tạo ra cấu trúc dữ liệu stack đặc dụng cho từng chương trình con đệ qui cụ thể phù hợp cơ chế LIFO.

A. Độ qui chỉ có một lệnh gọi trực tiếp

•Độ qui có dạng sau:

```
P(X)  $\equiv$  if C(X) D(X)
else {
    A(X) ;
    P(f(X)) ;
    B(X) ;
}
```

X là một biến đơn hoặc biến véc tơ.

C(X) là một biểu thức boolean của X .

A(X) , B(X) , D(X):không đệ qui

f(X) là hàm của X (hàm đơn điệu giảm)

Giải thuật thực hiện $P(X)$ với việc sử dụng Stack có dạng :

```
P(X)  $\equiv$  {  
    Create_Stack (S) ; ( tạo stack S )  
    while(not(C(X))  
    { A(X) ;  
      Push(S,X) ;  
      X := f(X) ;  
    }  
    D(X) ;  
    while (not Empty(S)) {  
      POP(S,X) ;  
      B(X) ;  
    }  
}
```

- Ví dụ: Thủ tục đệ qui chuyển biểu diễn số từ cơ số thập phân sang nhị phân có dạng :

```
Binary(m)  $\equiv$  if ( m > 0 ) {  
    Binary( m div 2 ) ;  
    write( m mod 2 ) ;  
}
```

- Trong trường hợp này :

X là m .

P(X) là Binary(m) .

A(X) ; D(X) là lệnh rỗng .

B(X) là lệnh Write(m mod 2) ;

C(X) là (m \leq 0) .

f(X) = f(m) = m div 2

Giải thuật thực hiện Binary(m) không đệ qui là:

Binary (m)

```
{ Create_Stack (S) ;  
  while ( m > 0 ) {  
    sdu := m mod 2 ;  
    Push(S,sdu) ;  
    m := m div 2 ;  
  }  
  while ( not Empty(S)) {  
    POP(S,sdu) ;  
    Display(sdu) ;  
  }  
}
```

B. Thủ tục đệ qui với hai lần gọi đệ qui

–Đệ qui có dạng sau

```
P(X) ≡ if C(X) D(X) ;  
else {  
    A(X) ; P(f(X)) ;  
    B(X) ; P(g(X)) ;  
}
```

-Thuật toán khử đệ qui tương ứng với thủ tục đệ quy

P(X) là:{

```
    Creat_Stack (S) :  
    Push (S, (X,1)) ;  
    do  
    {      while ( not C(X) ) {  
            A(X) ;  
            Push (S, (X,2)) ;  
            X := f(X) ;  
        }  
  
        D(X) ;  
        POP (S, (X,k)) ;  
        if ( k <> 1) {  
            B(X) ;  
            X := g(X) ;  
        }  
  
    } while ( k = 1 ) ;  
}
```


Khử đệ qui thủ tục Tháp Hà Nội .

- Dạng đệ qui





```
void THN(n , X , Y, Z )  
{ if( n > 0 ) {  
    THN ( n -1 , X , Z , Y ) ;  
    Move ( X , Y ) ;  
    THN ( n -1 , Z , Y , X ) ;  
}  
}
```

- Giải thuật không đệ qui tương đương là:

THN (n, X, Y, Z)

```
{ Creat_Stack (S) ;  
  Push (S ,(n,X,Y,Z,1)) ;  
  do  
    while ( n > 0 ) {  
      Push (S ,(n,X,Y,Z,2)) ;  
      n := n -1 ;  
      Swap (Y,Z) ;  
    }  
    POP (S,(n,X,Y,Z,k)) ;  
    if ( k <> 1 ) {  
      Move (X ,Z ) ;  
      n := n -1 ;  
      Swap (X ,Y ) ;  
    }  
  } while ( k = 1 ) ;  
}
```

Bài tập

-  Liệt kê mọi tập con của tập $1, 2, 3, \dots, n$, với n nhập từ bàn phím
-  Liệt kê mọi hoán vị của Từ COMPUTER (mở rộng, 1 từ bất kỳ nhập từ bàn phím)
-  Một nhà thám hiểm đem theo 1 cái túi với trọng lượng tối đa là B . Có n đồ vật cần mang theo, mỗi đồ vật có trọng lượng a_i và giá trị c_i tương ứng. Hãy viết CT tìm cách bỏ vào túi các đồ vật sao cho giá trị sử dụng là lớn nhất.
-  Bài toán Người du lịch : 1 người du lịch muốn đi thăm các thành phố khác nhau. Xuất phát tại 1 thành phố nào đó, họ muốn lần lượt qua tất cả các thành phố (1 lần) rồi trở lại thành phố ban đầu. Biết chi phí đi lại từ thành phố I đến J là C_{ij} . Hãy tìm hành trình với tổng chi phí thấp nhất

8 x 8

sao cho chúng không ăn được nhau.

Bài toán 8 con Hậu – Giải thuật

Algorithm Solve

Input trạng thái bàn cờ

Output

1. **if** trạng thái bàn cờ chứa đủ 8 con hậu
 - 1.1. In trạng thái này ra màn hình
2. **else**
 - 2.1. **for** mỗi ô trên bàn cờ mà còn an toàn
 - 2.1.1. thêm một con hậu vào ô này
 - 2.1.2. dùng lại giải thuật Solve với trạng thái mới
 - 2.1.3. bỏ con hậu ra khỏi ô này

End Solve

Vết cặn



Bài toán 8 con Hậu – Thiết kế phương thức

```
bool Queens::unguarded(int col) const;
```

Post: Returns **true** or **false** according as the square in the first unoccupied row (row count) and column col is not guarded by any queen.

```
void Queens::insert(int col);
```

Pre: The square in the first unoccupied row (row count) and column col is not guarded by any queen.

Post: A queen has been inserted into the square at row count and column col; count has been incremented by 1.

```
void Queens::remove(int col);
```

Pre: There is a queen in the square in row count – 1 and column col.

Post: The above queen has been removed; count has been decremented by 1.

```
bool Queens::is_solved() const;
```

Post: The function returns **true** if the number of queens already placed equals board_size; otherwise, it returns **false**.

Bài toán 8 con Hậu – Thiết kế dữ liệu đơn giản

```
const int max_board = 30;

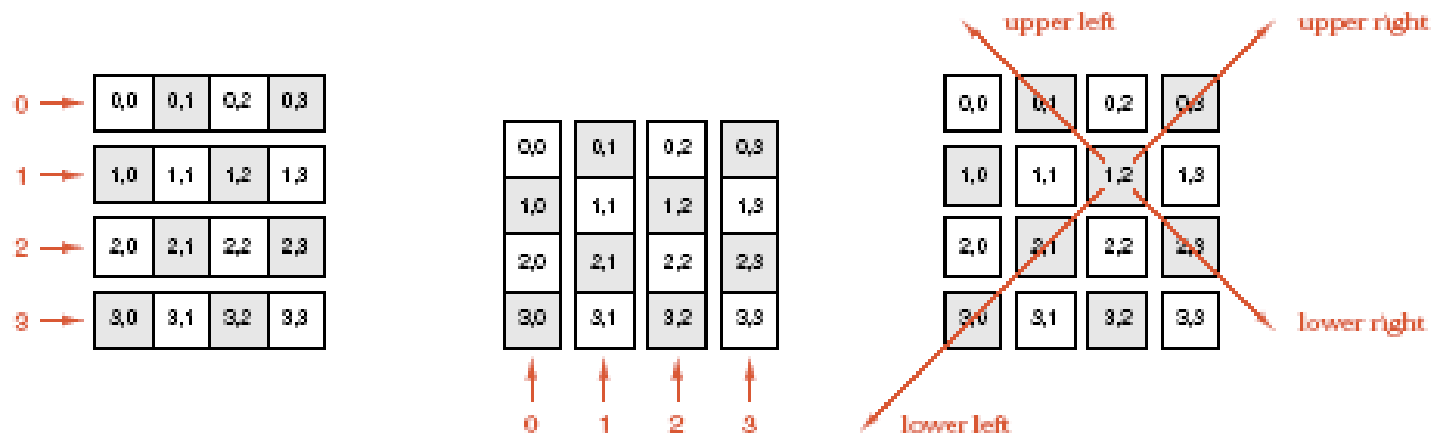
class Queens {
public:
    Queens(int size);
    bool is_solved( ) const;
    void print( ) const;
    bool unguarded(int col) const;
    void insert(int col);
    void remove(int col);
    int board_size; // dimension of board = maximum number of queens
private:
    int count; // current number of queens = first unoccupied row
    bool queen_square[max_board][max_board];
};
```

Bài toán 8 con Hậu – Mã C++

```
void Queens :: insert(int col) {  
    queen_square[count++][col] = true;  
}
```

```
bool Queens :: ungarded(int col) const {  
    int i;  
    bool ok = true;  
    for (i = 0; ok && i < count; i++)           //kiểm tra tại một cột  
        ok = !queen_square[i][col];  
    //kiểm tra trên đường chéo lên  
    for (i = 1; ok && count - i >= 0 && col - i >= 0; i++)  
        ok = !queen_square[count - i][col - i];  
    //kiểm tra trên đường chéo xuống  
    for (i = 1; ok && count - i >= 0 && col + i < board_size; i++)  
        ok = !queen_square[count - i][col + i];  
    return ok;  
}
```

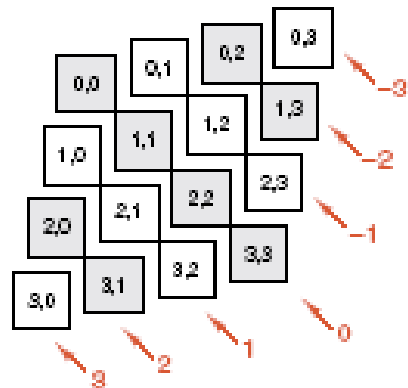
Bài toán 8 con Hậu – Góc nhìn khác



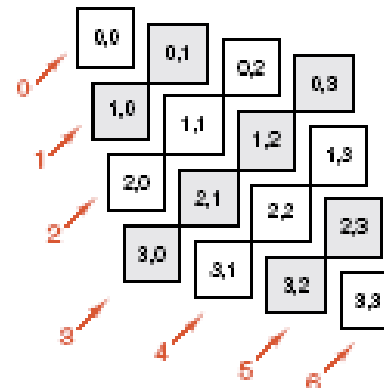
(a) Rows

(b) Columns

(c) Diagonal directions



difference = row - column



sum = row + column

Bài toán 8 con Hậu – Thiết kế mới

```
const int max_board = 30;
class Queens {
public:
    Queens(int size);
    bool is_solved( ) const;
    void print( ) const;
    bool unguarded(int col) const;
    void insert(int col);
    void remove(int col);
    int board size;
private:
    int count;
    bool col_free[max_board];
    bool upward_free[2 * max_board - 1];
    bool downward_free[2 * max_board - 1];
    int queen_in_row[max_board]; //column number of queen in each row
};
```

Bài toán 8 con Hậu – Mã C++ mới

```
Queens :: Queens(int size) {  
    board_size = size;  
    count = 0;  
    for (int i = 0; i < board_size; i++)  
        col_free[i] = true;  
    for (int j = 0; j < (2 * board_size - 1); j++)  
        upward_free[j] = true;  
    for (int k = 0; k < (2 * board_size - 1); k++)  
        downward_free[k] = true;  
}
```

```
void Queens :: insert(int col) {  
    queen_in_row[count] = col;  
    col_free[col] = false;  
    upward_free[count + col] = false;  
    downward_free[count - col + board_size - 1] = false;  
    count++;  
}
```