BÀI 02

ĐỆ QUY – QUAY LUI

Design by Minh An

Email: anvanminh.haui@gmail.com

Nội dung

- Bài toán liệt kê
- Một số kiến thức về đại số tổ hợp
- Phương pháp sinh
- Đệ quy
- Quay lui
- Một số bài tập

1.4. Đệ quy

- **Khái niệm**
- **Thành phần của một mô tả đệ quy**
- **Một số ví dụ về mô tả đệ quy**
- **Phân loại đệ quy**
- **Giải quyết bài toán bằng đệ quy**

1.4.1. Khái niệm

- **Một đối tượng được mô tả thông qua chính nó gọi là mô tả đệ quy.**
- **Một bài toán có tính chất đệ quy khi nó có thể phân rã thành những bài toán nhỏ hơn nhưng có tính chất của bài toán ban đầu.**
- **Một hàm có tính chất đệ quy nếu trong thân của nó có lời gọi lại chính nó.**
- **Định nghĩa tường minh: Giải thích khái niệm mới bằng khái niệm đã có:**
- **VD: Người = Động vật cao cấp**
- **Định nghĩa lòng vòng (đệ quy):**
- **VD: Người = Con của 2 Người khác**

Thành phần của một mô tả đệ quy

- **Phần cơ sở (neo, trường hợp suy biến).**
  - Điều kiện dừng của đệ quy
- **Phần quy nạp.**
  - Mô tả đối tượng thông qua chính đối tượng đó một cách trực tiếp hoặc gián tiếp.

Thành phần của một mô tả đệ quy

- **Ví dụ**

1. Tập số tự nhiên N.
2. N!
3. Ước chung lớn nhất của 2 số nguyên dương a, b.

ucln(a, b) = $\begin{cases} b \text{ nếu } a \,\%\, b = 0 \text{ (thành phần cơ sở)} \\ ucln(b, a \,\%\, b) \text{ nếu } a \,\%\, b \neq 0 \text{ (thành phần quy nạp)} \end{cases}$

4. Dãy số Fibonaci.
5. Tổ hợp chập k của n phần tử C(k, n).
6. Tổng n số tự nhiên đầu tiên: $p(n) = \begin{cases} 0 \text{ nếu } n = 0 \\ n + p(n-1) \text{ nếu } n > 0 \end{cases}$

1.4.2. Phân loại đệ quy

- **Đệ quy tuyến tính:**
  - Mỗi lần thực thi chỉ gọi đệ quy một lần (trong hàm đệ quy chỉ có một lời gọi đệ quy).

VD: Hàm giai thừa

```
unsigned long gt(int n){
    if (n<2)
        return 1;
    else
        return n*gt(n-1);
}
```

```
rec_func( ) {
    if (condition)
        Lệnh 1;
    else {
        Lệnh 2 ;
        rec_func(…);
    }
}
```


Phân loại đệ quy

- **Đệ quy nhị phân**
  - Mỗi lần thực thi có thể gọi đệ quy hai lần (trong hàm đệ quy có hai lời gọi đệ quy).

VD: Hàm tính số C(k,n)

```
long C(int n, int k) {
    if (k == 0 || k == n)
        return 1;
    else
        return C(n-1, k) +
               C(n-1, k-1);
}
```

```
rec_func( ) {
    if (condition)
        Lệnh 1;
    else {
        Lệnh 2 ;
        rec_func(…);
        rec_func(…);
    }
}
```

Phân loại đệ quy

- **Đệ quy tương hỗ.**
  - Các hàm đệ quy gọi lẫn nhau.

Phỏng đoán Collatz

- Nếu X chẵn → X = X / 2
- Nếu X lẻ → X = X * 3 + 1

Với mọi số tự nhiên X > 0 quá trình biến đổi như trên đều đưa về số 1 (và sau đó sẽ tuần hoàn với dãy 4, 2, 1).

Đệ quy tương hỗ

```
void xuly_le(int x)  {
      cout<<x*3+1<<", ";
      xuly_so (x*3+1);
}
void xuly_chan(int x){
      cout<<x/2<<", ";
      xuly_so (x/2);
}
void xuly_so(int x){
      if (x%2==0)
            xuly_chan(x);
      else if (x>1) xuly_le(x);
}
```

Đệ quy tương hỗ

```
rec_func1(){
   if (ĐK)
              Lệnh 1;
   else{
        Lệnh 2 ;
        rec_func2(…);
   }
}
```

```
rec_func2(){
   if (ĐK)
        Lệnh 1;
   else{
        Lệnh 2;
        rec_func1(…);
   }
}
```


Phân loại đệ quy

- **Đệ quy phi tuyến.**
  - Hàm đệ quy được gọi trong vòng lặp.

Ví dụ:

Xác định dãy { An } theo công thức truy hồi:

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

Đệ quy phi tuyến

```
int A(int n) {
  if (n==0)
     return 1 ;
  else {
     int tg= 0 ;
     for (int i=0; i<n; i++)
          tg = tg + (n-i)^2 * A(i);
          return tg;
  }
}
```


Phân loại đệ quy

- Đệ quy lồng.
  - Tham số trong lời gọi đệ quy là một lời gọi đệ quy.

Ví dụ: **Hàm số Ackermann**

Cho hàm A(x, y), với miền giá trị là R

A (0, y) = 1, nếu y ≥ 0
A (1, 0) = 2
A (x, 0) = x + 2, nếu x ≥ 0
A (x, y) = A (A (x - 1, y), y - 1), nếu x ≥ 1 và y ≥ 1

Đệ quy lồng

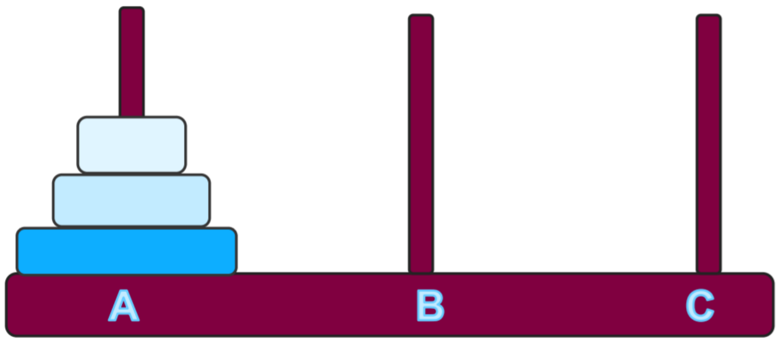
```
long Acker(int m, int n)
{
      if (m == 0)
            return (n + 1);
      else if (n == 0)
            return Acker(m - 1, 1);
      else
            return Acker(m-1, Acker(m, n-1));
}
```


1.4.3. Giải quyết bài toán bằng đệ quy

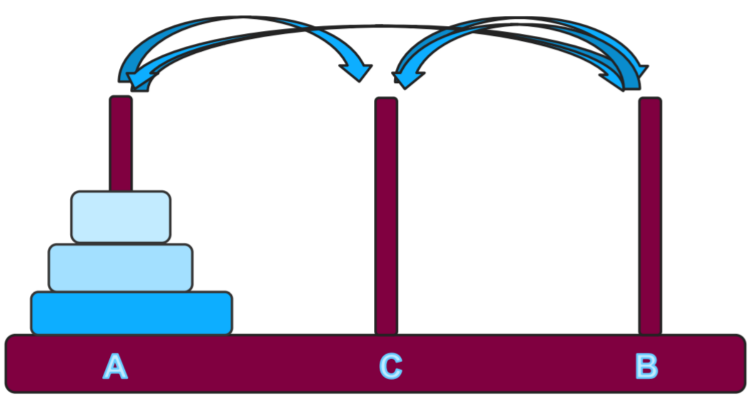
- **Tổng quát hoá bài toán.**
- **Tìm trường hợp suy biến (neo, cơ sở).**
- **Tìm giải thuật trong trường hợp tổng quát bằng phân rã bài toán theo kiểu đệ quy.**

Giải quyết bài toán bằng đệ quy

- Bài toán Tháp Hà Nội.



Bài toán tháp hà nội



Design by Minh An

Bài toán tháp hà nội

- Tổng quát thành bài toán: Chuyển n đĩa từ A sang C
- Tìm trường hợp suy biến: Chỉ có 1 đĩa
- Tìm giải thuật trong trường hợp tổng quát bằng phân rã bài toán theo kiểu đệ quy:
  - Chuyển n - 1 đĩa từ A sang B (C làm trung gian)
  - Chuyển 1 đĩa từ A sang C
  - Chuyển n - 1 đĩa từ B sang C (A làm trung gian)

Bài toán tháp hà nội

Mô tả đệ quy

- Nếu n = 1:
  - Chuyển 1 đĩa từ A sang C
- Ngược lại:
  - Chuyển n – 1 đĩa từ A sang B
  - Chuyển 1 đĩa từ A sang C
  - Chuyển n – 1 đĩa từ B sang C

Bài toán tháp hà nội

Thuật toán

```
void Chuyen(n, A, B, C){
   if  (n == 1)
      Chuyển 1 đĩa từ A sang C;
   else{
      Chuyen(n-1, A, C, B);
      Chuyen(1, A, B, C);
      Chuyen(n-1, B, A, C);
   }
}
```


Giải quyết bài toán bằng đệ quy

- **Thuật toán Loang**
  - Bài toán tìm Miền liên thông: Cho một lưới hình chữ nhật kích thước MxN gồm các ô có giá trị 0 hoặc 1.
    - Mỗi ô (i, j) có 4 ô liền kề là (i-1, j), (i+1, j), (i, j-1), (i, j+1).
    - Hai ô trong lưới được gọi là cùng một miền liên thông, nếu chúng có cùng giá trị, đồng thời có thể "đi đến" được nhau thông qua các ô liền kề.
  - Yêu cầu: Tìm số miền liên thông của lưới.

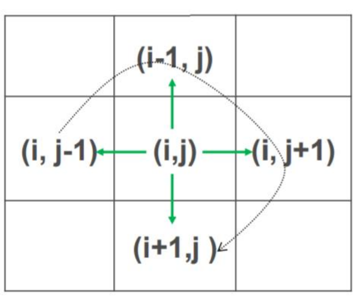
Thuật toán loang

- **Bài toán tìm Miền liên thông**
  - Ví dụ: Lưới kích thước 5x6 dưới đây có 8 miền liên thông.

1	1	0	0	0	1
1	0	0	0	0	0
1	0	0	1	1	0
0	1	1	0	0	1
0	0	1	0	1	1

Thuật toán loang

- **Phân tích:**



- Từ mỗi ô (i, j) sẽ lần lượt "thử đi" sang các ô liền kề theo một thứ tự nhất định (VD: trái -> trên -> phải -> dưới).
- Nếu "đi được" và ô đó có cùng giá trị với ô (i, j) đang xét thì:
  - Ghi nhận ô đó là cùng miền với ô (i, j).
  - Và tiếp tục đệ quy với các ô này.

Thuật toán loang

- **Thuật toán: "loang" từ ô (i, j):**

```
void loang(int i, int j) {
    Đánh dấu là Ô[i][j] đã xét
    if ((Ô[i-1][j]==Ô[i][j]) && (Ô[i-1][j] chưa xét) && Ô[i-1][j]
    nằm trong bảng)
        loang(i-1, j);
    //Tương tự với các Ô[i+1][j], Ô[i][j-1] và Ô[i][j+1]
}
```


Thuật toán loang

- **Cấu trúc dữ liệu:**
  - Biến so_mien (kiểu int) để đếm số miền liên thông.
  - Mảng 2 chiều O[M][N] (kiểu int) để biểu diễn lưới.
  - Mảng 2 chiều flag[M][N] (kiểu bool) để biểu diễn việc đánh dấu ô trong lưới đã được xét.

Thuật toán loang

- **Hàm "loang" từ ô (i, j):**

```
void loang(int i, int j) {
   flag[i][j] = true; //Loang tới Ô(i, j)
   if ((a[i-1][j]==a[i][j]) && (! flag[i-1][j]))
      loang(i-1, j); //Loang tiếp
   if ((a[i+1][j]==a[i][j]) && (! flag[i+1][j]))
      loang(i+1, j);
   if ((a[i][j-1]==a[i][j]) && (! flag[i][j-1]))
      loang(i, j-1);
   if ((a[i][j+1]==a[i][j]) && (! flag[i][j+1]))
      loang(i, j+1);
}
```

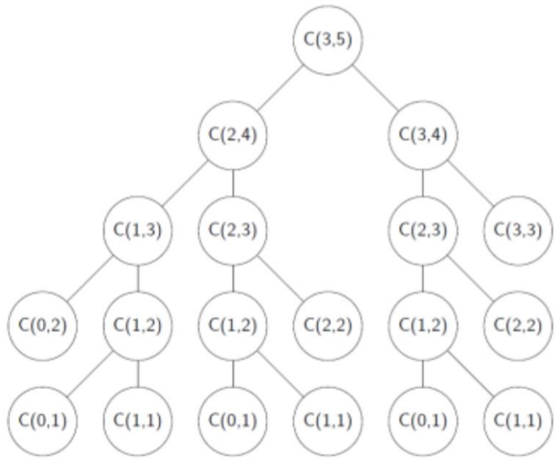

1.4.3. Đệ quy có nhớ

- **Xét bài toán tổ hợp chập k của n, C(k, n)**

```
long C(int k, int n) {
   if (k == 0) || (k == n)
      return 1;
   else return C(k, n-1) + C(k-1, n-1);
}
```

Đệ quy có nhớ

- **Sơ đồ gọi đệ quy của C(k, n)**



Đệ quy có nhớ

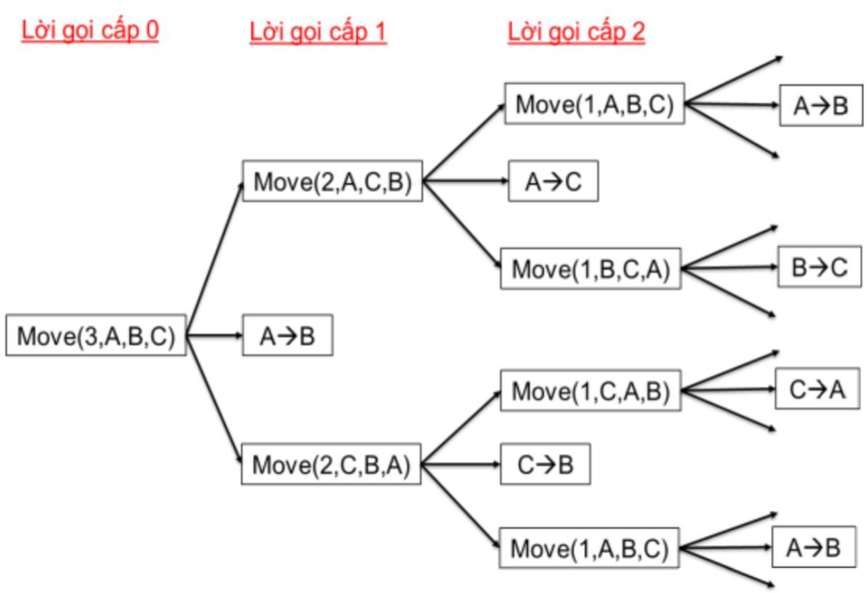
- **Nhận xét**
  - Có một số (công thức tính) lời gọi bị lặp lại: C(2, 3), C(1, 2), …
  - Nếu dùng đệ quy để tính lại các công thức này sẽ mất nhiều thời gian.
  - Để giảm bớt thời gian tính ta lưu các giá trị đã tính toán vào một mảng để sau này khi gặp lại ta chỉ việc lấy giá trị đã lưu mà không phải tính lại.
  - Cần một mảng kích thước (k+1)x(n+1) cho bài toán C(k, n).

Đệ quy có nhớ

- **Hàm đệ quy có nhớ cho bài toán C(k, n)**

```
long **d;
long C(int k, int n) {
   if (k == 0 || k == n)
      d[k][n] = 1;
   else if (d[k][n] < 0)
      d[k][n] = C(k, n-1) + C(k-1, n-1);
   return d[k][n];
}
```


1.4.4. Khử đệ quy



Khử đệ quy

- Với bài toán Tháp Hà nội, khi n = 64 đĩa, thì số lần gọi đệ quy và thực hiện lệnh là $2^{64} - 1$ (chương trình sẽ hoàn thành trong khoảng 100 triệu năm với máy tính hiện đại nhất hiện nay).

Khử đệ quy

- Ví dụ 1: Tính n!

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

```
long Factorial(int n) {
	int gt = 1;
	for (int i=2; i<=n; i++)
		gt *= i;
	return gt;
}
```

Khử đệ quy

- Ví dụ 1: Tính n!

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

```
long Factorial(int n) {
     int gt = 1;
     for (int i=2; i<=n; i++)
          gt *= i;
     return gt;
}
```


Ví dụ 2: Tìm số fibonaci thứ n

```
long f(int n) {
     if (n < 3) return 1;
     else return f(n-1) + f(n-2);
}
```

```
long f(int n) {
     int fn, fn1 = 1, fn2 = 1;
     for (int i=3; i<=n; i++) {
          fn = fn1 + fn2;
          fn2 = fn1; fn1 = fn;
     }
     return fn;
}
```

Ví dụ 3: Tìm ước số chung lớn nhất

```
int ucln(int a, int b) {
     if (a % b == 0) return b;
     else return ucln(b, a % b);
}
```

```
int ucln(int a, int b) {
     int r = a % b;
     while (r != 0) {
          a = b; b = r;
          r = a % b;
     }
     return b;
}
```

Design by Minh An

Ví dụ 4: Tháp Hà Nội

```
void chuyen(int n, char a, char b, char c) {
   if (n == 1)
      cout<<"Chuyen 1 dia tu A sang B";
   else {
      chuyen(n-1, a, c, b);
      chuyen(1, a, b, c);
      chuyen(n-1, b, a, c);
   }
}
```

Design by Minh An

Ví dụ 4: Tháp Hà Nội

```
void chuyen(int n, char a, char b, char c) {
   push(s, (n, a, b, c));
   while (!empy(s)){
      pop(s, (n, a, b, c));
      if (n == 1)
         cout<<"Chuyen 1 dia tu A sang B";
      else {
         push(s, (n-1, b, a, c));
         push(s, (1, a, b, c));
         push(s, (n-1, a, c, b);
   }
}
```


Bài tập

1) Cài đặt hàm tìm số finonaci thứ n bằng đệ quy có nhớ. Áp dụng hàm để hiển thị dãy n số fibonaci và tổng của dãy.
2) Khởi tạo lưới (mảng 2 chiều) kích thước 12 x 12 gồm các số ngẫu nhiên trong đoạn [0, 5], hiển thị lưới. Đếm số miền liên thông chứa các ô có giá trị 5 trong lưới.
3) Thiết kế giải thuật tính tổng các chữ số của một số nguyên dương theo cách:
   - Đệ quy
   - Lặp
   - Viết chương trình ứng dụng.

Công thức khởi tạo số ngẫu nhiên dùng hàm rand():

a[i][j] = n + rand() % (m - n + 1); //So ngau nhien trong doan [n, m]