HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN

TUẦN 2 : ĐỆ QUY, ĐỆ QUY CÓ NHỚ

ONE LOVE. ONE FUTURE.

NỘI DUNG

- Khái niệm cơ bản
- Sơ đồ chung đệ quy
- Phân tích thuật toán đệ quy
- Đệ quy có nhớ

KHÁI NIỆM CƠ BẢN

- Đối tượng đệ quy: được xác định thông qua chính nó nhưng với quy mô nhỏ hơn
- Hàm đệ quy
  - Bước cơ sở: Xác định giá trị của hàm với một số giá trị tham số ban đầu
  - Bước đệ quy: Xác định mối quan hệ giữa hàm phụ thuộc vào chính hàm đó nhưng với tham số nhỏ hơn

- Cơ sở: F(n) = 1, với n = 1
- Đệ quyF(n) = F(n-1) + n, với n > 1

- Cơ sở: C(k, n) = 1, với k = 0 hoặc k = n
- Đệ quy: C(k, n) = C(k-1,n-1) + C(k,n-1), với các trường hợp khác

KHÁI NIỆM CƠ BẢN

- Đối tượng đệ quy: được xác định thông qua chính nó nhưng với quy mô nhỏ hơn
- Tập hợp được xác định đệ quy
  - Bước cơ sở: xác định các phần tử đầu tiên của tập hợp
  - Bước đệ quy: xác định luật cho biết các phần từ lớn hơn thuộc tập hợp từ các phần tử ban đầu

- Cơ sở: 3 thuộc tập S
- Đệ quy: Nếu x và y thuộc S thì x + y thuộc S

SƠ ĐỒ CHUNG ĐỆ QUY

- Thuật toán đệ quy là thuật toán tự gọi đến chính mình với đầu vào kích thước nhỏ hơn.
- Thuật toán đệ quy thường được dùng khi cần xử lý với các đối tượng được định nghĩa đệ quy.
  - Ví dụ: hàm tính dãy số Fibonacci được định nghĩa đệ quy:
    - $f(0) = 0, f(1) = 1,$
    - $f(n) = f(n-1) + f(n-2)$ với $n > 1$
- Các ngôn ngữ lập trình bậc cao thường cho phép xây dựng các hàm đệ quy, nghĩa là trong thân của hàm có chứa những lệnh gọi đến chính nó. Vì thế, khi cài đặt các thuật toán đệ quy, người ta thường xây dựng các hàm đệ quy.

SƠ ĐỒ CHUNG ĐỆ QUY

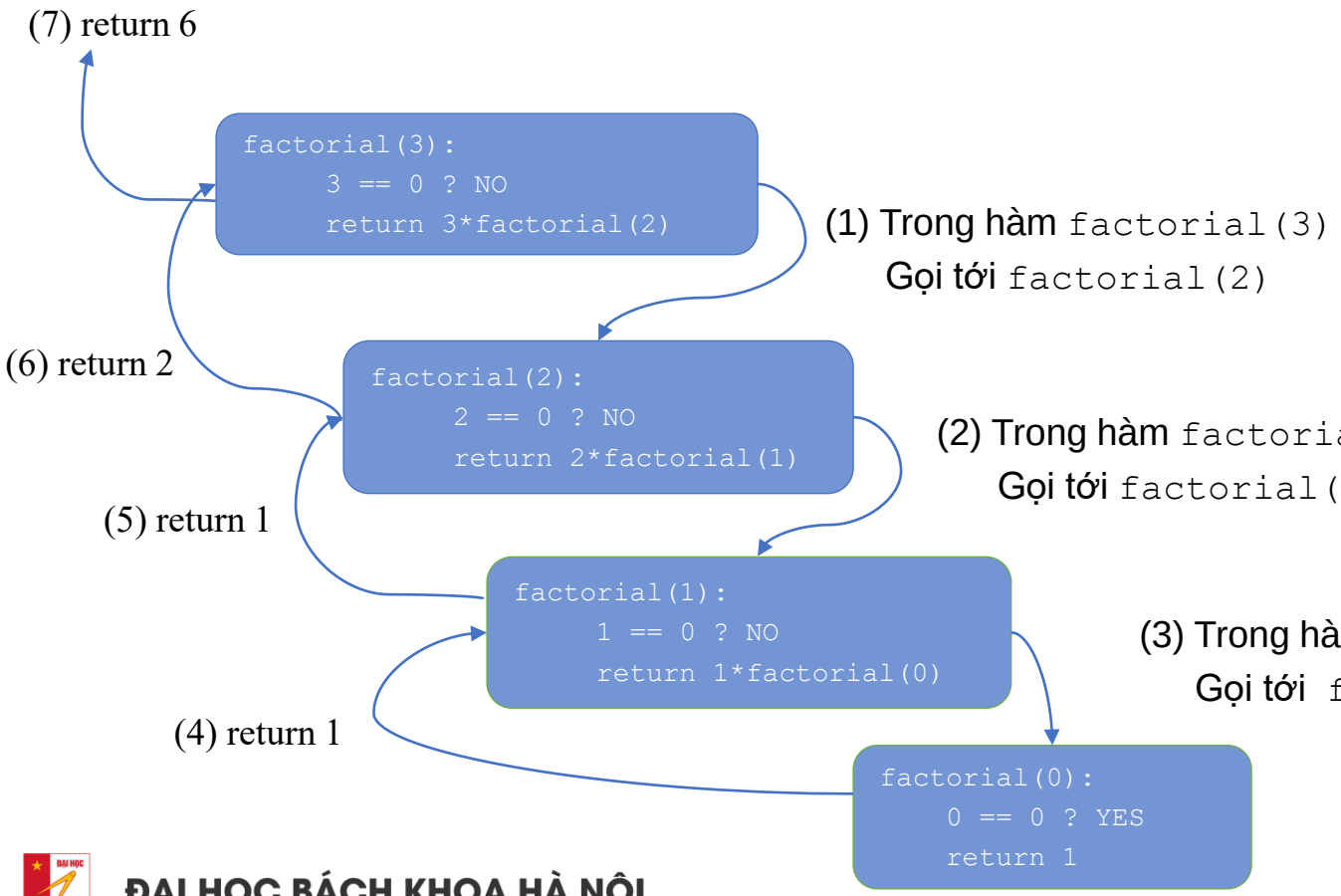
```
DeQuy(input) {
   if (kích thước của input là nhỏ nhất) then
         Thực hiện Bước cơ sở;  /* giải bài toán kích thước đầu vào nhỏ nhất */
  else  {
         DeQuy(input với kích thước nhỏ hơn);  /* bước đệ quy  */
         /* Chú ý: có thể có thêm những lệnh gọi đệ quy  */
         Tổ hợp lời giải của các bài toán con để thu được lời_giải;
         return lời_giải;
  }
}
```

VÍ DỤ

- **Ví dụ 1: Tính *n*! theo công thức đệ quy:**

$f(0) = 1$

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




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

Thực thi hàm factorial(3) sẽ dừng cho đến khi hàm factorial(2) trả về kết quả

Khi hàm factorial(2) trả về kết quả, hàm factorial(3) tiếp tục được thực hiện

VÍ DỤ

- **Ví dụ 2: Tính dãy số Fibonacci:**

$$F(0) = 0;\ F(1) = 1$$
$$F(n) = F(n\text{-}1) + F(n\text{-}2) \text{ với } n \geq 2$$

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

VÍ DỤ: BÀI TOÁN THÁP HÀ NỘI

- **Ví dụ 3: Bài toán Tháp Hà Nội:**

Bài toán Tháp Hà Nội được trình bày như sau: "*Có 3 cọc a, b, c. Trên cọc a có một chồng gồm n cái đĩa, đường kính giảm dần từ dưới lên trên. Cần phải chuyển chồng đĩa từ cọc a sang cọc c tuân thủ quy tắc:*

1. *Mỗi lần chỉ chuyển 1 đĩa*
2. *Chỉ được xếp đĩa có đường kính nhỏ hơn lên trên đĩa có đường kính lớn hơn. Trong quá trình chuyển được phép dùng cọc b làm cọc trung gian.*

Bài toán đặt ra là: Hãy liệt kê các bước di chuyển đĩa cần thực hiện để hoàn thành nhiệm vụ đặt ra của bài toán.

VÍ DỤ: BÀI TOÁN THÁP HÀ NỘI

Chuyển *n* đĩa từ cọc *a* sang cọc *c* sử dụng cọc *b* làm trung gian:

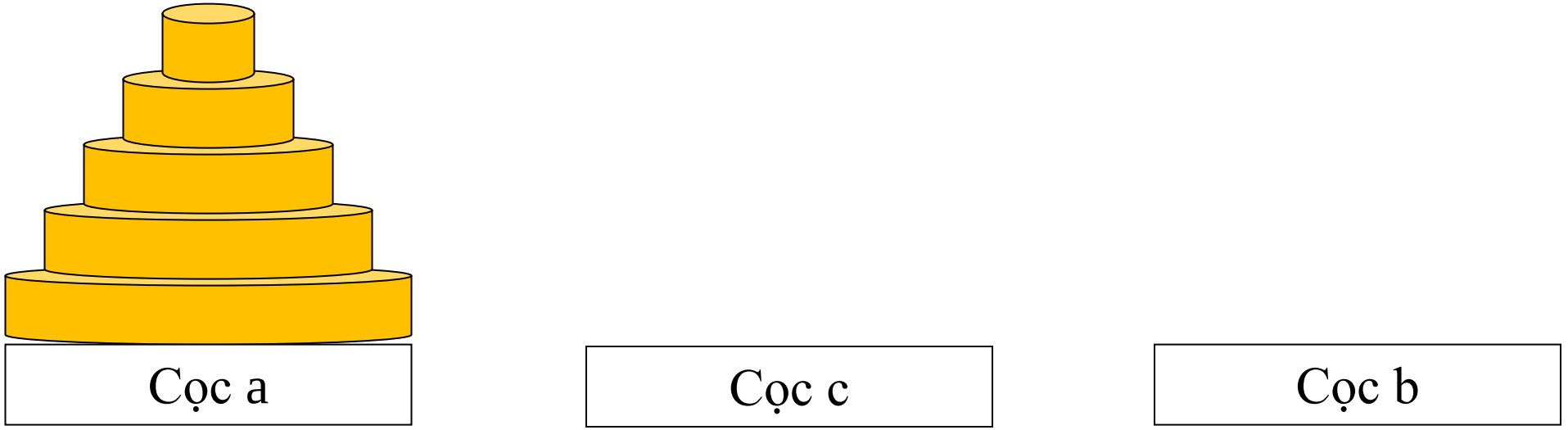
HanoiTower(n, a, c, b);

Việc di chuyển đĩa gồm 3 giai đoạn:

(1) Chuyển *n*-1 đĩa từ cọc a sang cọc b, sử dụng cọc c làm trung gian

(2) Chuyển 1 đĩa (đĩa với đường kính lớn nhất) từ cọc a sang cọc c

(3) Chuyển *n*-1 đĩa từ cọc b sang cọc c, sử dụng cọc a làm trung gian



VÍ DỤ: BÀI TOÁN THÁP HÀ NỘI

Chuyển *n* đĩa từ cọc *a* sang cọc *c* sử dụng cọc *b* làm trung gian:

```
HanoiTower(n, a, c, b);
```

Việc di chuyển đĩa gồm 3 giai đoạn:

(1) Chuyển *n*-1 đĩa từ cọc a sang cọc b, sử dụng cọc c làm trung gian

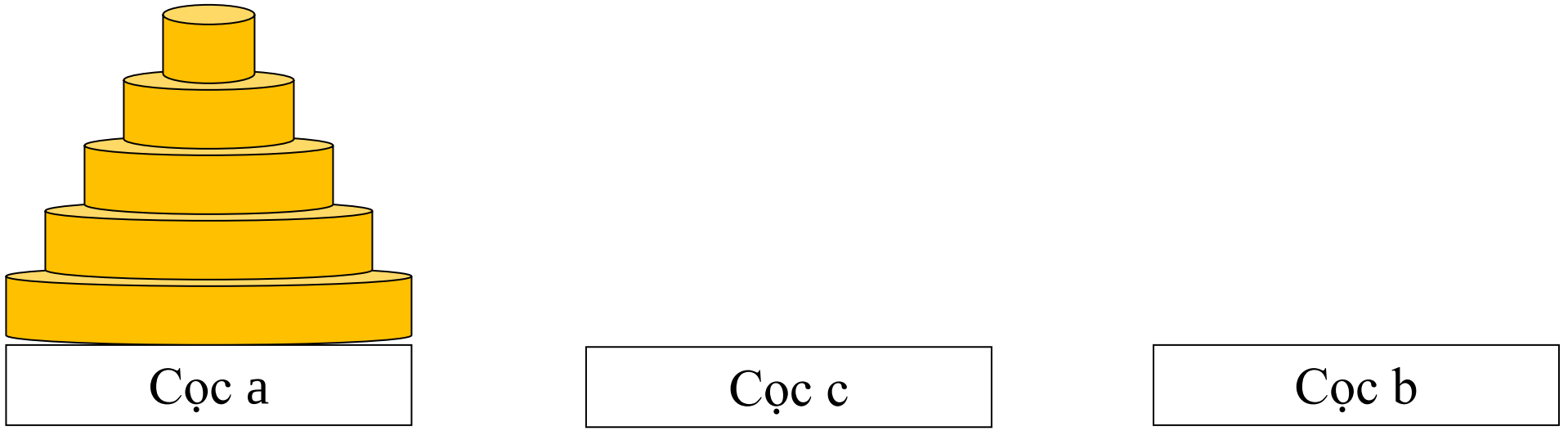
Giải bài toán kích thước *n*-1 : **`HaNoiTower(n-1, a, b, c)`**

(2) Chuyển 1 đĩa (đĩa với đường kính lớn nhất) từ cọc a sang cọc c

Giải bài toán kích thước n = 1 (chỉ cần 1 bước di chuyển đĩa): **`HaNoiTower(1, a, c, b)`**

(3) Chuyển *n*-1 đĩa từ cọc b sang cọc c, sử dụng cọc a làm trung gian

Giải bài toán kích thước *n*-1: **`HaNoiTower(n-1, b, c, a)`**



VÍ DỤ: BÀI TOÁN THÁP HÀ NỘI

Ví dụ 3: Bài toán tháp Hà Nội: *n* = 5

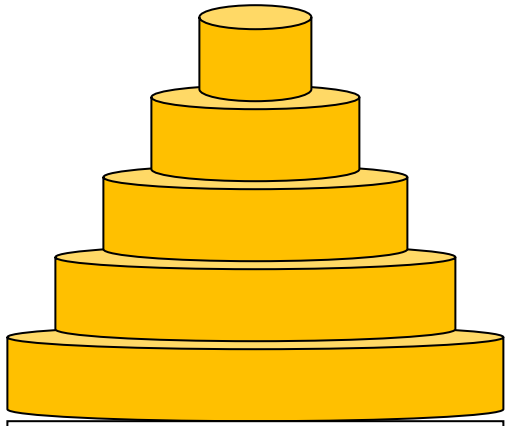
(1) Chuyển *n*-1 đĩa từ cọc a sang cọc b, sử dụng cọc c làm trung gian

Giải bài toán kích thước *n*-1 : **`HaNoiTower(n-1, a, b, c)`**

(2) Chuyển 1 đĩa (đĩa với đường kính lớn nhất) từ cọc a sang cọc c

(3) Chuyển *n*-1 đĩa từ cọc b sang cọc c, sử dụng cọc a làm trung gian

Giải bài toán kích thước *n*-1: **`HaNoiTower(n-1, b, c, a)`**



VÍ DỤ: BÀI TOÁN THÁP HÀ NỘI

Thuật toán có thể mô tả trong thủ tục đệ qui sau đây:

```
HanoiTower(n, a, c, b) {//chuyển n đĩa từ cọc a sang cọc c sử dụng cọc b làm trung gian:
    if (n==1) then <chuyển đĩa từ cọc a sang cọc c>
    else {
        HanoiTower(n-1,a,b,c);
        HanoiTower(1,a,c,b);
        HanoiTower(n-1,b,c,a);
    }
}
```

Việc di chuyển đĩa gồm 3 giai đoạn:

(1) Chuyển *n*-1 đĩa từ cọc a sang cọc b, sử dụng cọc c làm trung gian

Giải bài toán kích thước *n*-1 : **`HaNoiTower(n-1, a, b, c)`**

(2) Chuyển 1 đĩa (đĩa với đường kính lớn nhất) từ cọc a sang cọc c

Giải bài toán kích thước n = 1 (chỉ cần 1 bước di chuyển đĩa): **`HaNoiTower(1, a, c, b)`**

(3) Chuyển *n*-1 đĩa từ cọc b sang cọc c, sử dụng cọc a làm trung gian

Giải bài toán kích thước *n*-1: **`HaNoiTower(n-1, b, c, a)`**

VÍ DỤ: BÀI TOÁN THÁP HÀ NỘI

```c
#include<stdio.h>

int i = 0;

void HanoiTower (int n, char xuatphat, char dich, char trunggian)
{
    if (n == 1){
          printf("Dich chuyen dia tu coc %c den coc %c\n",xuatphat,dich);
          i++;
          return;
    } else {
        HanoiTower (n-1, xuatphat, trunggian, dich);
        HanoiTower (1, xuatphat, dich, trunggian);
        HanoiTower (n-1, trunggian, dich, xuatphat);
    }
}

int main()
{
    int n;
    printf("Nhap so dia n = "); scanf("%d",&n);
    HanoiTower (n, 'a', 'c', 'b');
    printf("Tong so lan di chuyen dia = %d",i);
    return 0;
}
```

```
Nhap so dia n = 3
Dich chuyen dia tu coc a den coc c
Dich chuyen dia tu coc a den coc b
Dich chuyen dia tu coc c den coc b
Dich chuyen dia tu coc a den coc c
Dich chuyen dia tu coc b den coc a
Dich chuyen dia tu coc b den coc c
Dich chuyen dia tu coc a den coc c
Tong so lan di chuyen dia = 7
```

PHÂN TÍCH THUẬT TOÁN ĐỆ QUY

- Để phân tích thuật toán đệ qui ta thường tiến hành như sau:
  - Gọi ***T(n)*** là thời gian tính của thuật toán
  - Xây dựng công thức đệ quy cho ***T(n)***
  - *Giải công thức đệ quy* thu được để đưa ra đánh giá cho ***T(n)***
- Nói chung ta chỉ cần một đánh giá sát cho tốc độ tăng của $T(n)$ nên việc ***giải công thức đệ quy*** đối với $T(n)$ là đưa ra đánh giá tốc độ tăng của $T(n)$ trong ký hiệu tiệm cận.

PHÂN TÍCH THUẬT TOÁN ĐỆ QUY

Ví dụ: Tính *n*! theo công thức đệ quy:

$f(0) = 1$

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

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

- Gọi $T(n)$ là số phép toán nhân phải thực hiện trong lệnh gọi factorial(n).
- Ta có:

$T(0) = 0,$

$T(n) = T(n-1) + 1, n \geq 1$

PHÂN TÍCH THUẬT TOÁN ĐỆ QUY

- Gọi $T(n)$ là số phép toán nhân phải thực hiện trong lệnh gọi factorial(n). Ta có:

  $T(0) = 0$,

  $T(n) = T(n\text{-}1) + 1$, $n \geq 1$

- Giải công thức đệ quy T(n), ta có:

$T(n) = T(n\text{-}1) + 1$ — thay thế $T(n\text{-}1)$

$= T(n\text{-}2) + 1 + 1$ — thay thế $T(n\text{-}2)$

$= T(n\text{-}3) + 1 + 1 + 1$

$= T(n\text{-}3) + 3$

$= \ldots$

$= T(n\text{-}k) + k$

$T(n) = T(n\text{-}n) + n$ — *chọn $k = n$*

$= T(0) + n$

$= n$

Vậy $T(n) = O(n)$

ĐỆ QUY CÓ NHỚ

- Bài toán con trùng lặp

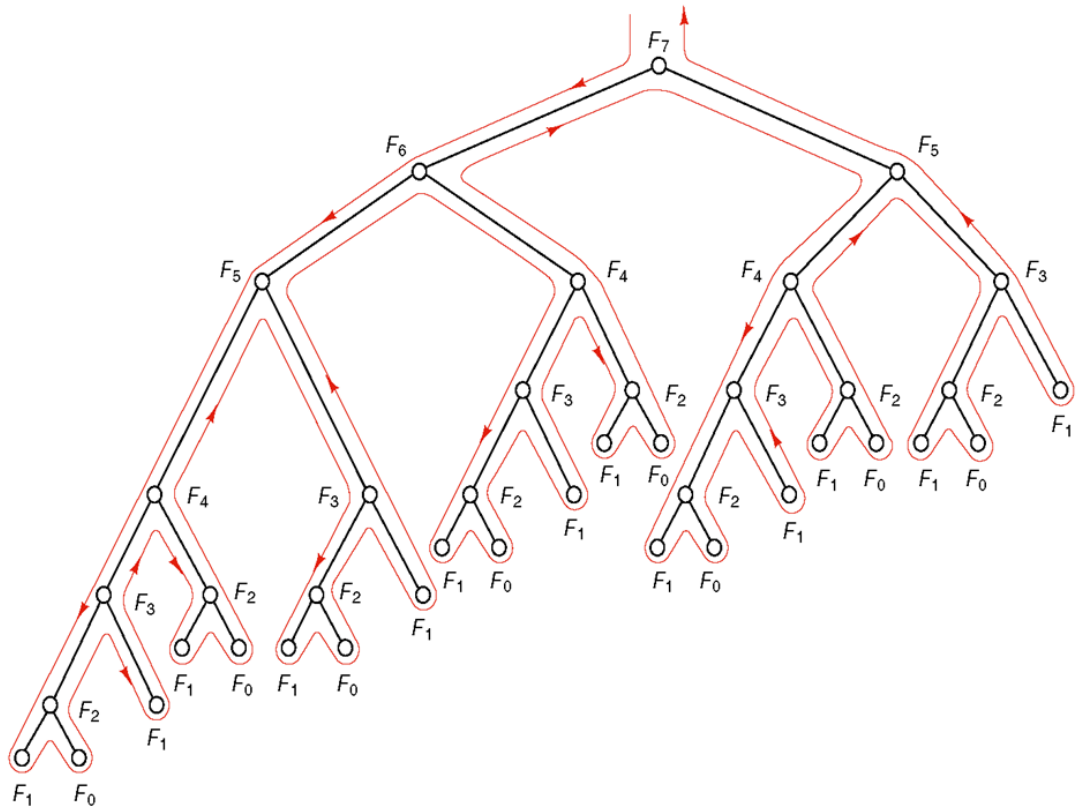
  Ví dụ 1: Tính dãy số Fibonacci:

  $F(0) = 0; F(1) = 1$

  $F(n) = F(n-1) + F(n-2)$ với $n \geq 2$

- Trong thuật toán đệ quy, mỗi khi cần đến lời giải của một bài toán con ta lại phải trị nó một cách đệ quy. Do đó, có những bài toán con bị giải đi giải lại nhiều lần. Điều đó dẫn đến tính kém hiệu quả của thuật toán. Hiện tượng này gọi là hiện tượng bài toán con trùng lặp.

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



ĐỆ QUY CÓ NHỚ

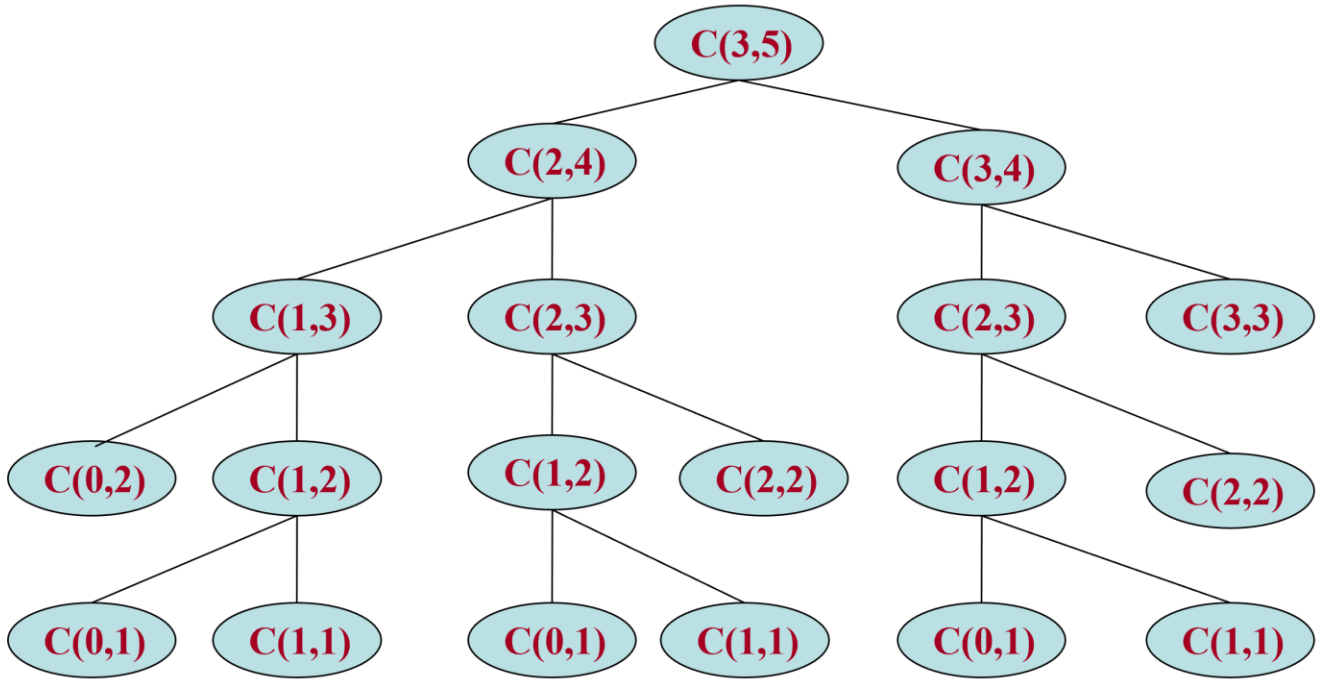
- Bài toán con trùng lặp

  Ví dụ 2: Tính hệ số nhị thức:

  $C(0,n) = 1,\ C(n,n) = 1;$ với mọi $n \geq 0,$

  $C(k,n) = C(k-1,n-1)+C(k,n-1),\ 0 < k < n$

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



ĐỆ QUY CÓ NHỚ

- Trong hai ví dụ trên, ta đã thấy các thuật toán đệ quy để tính số Fibonacci và tính hệ số nhị thức là kém hiệu quả.
- Để tăng hiệu quả của các thuật toán đệ quy, ta có thể sử dụng kỹ thuật đệ quy có nhớ.
  - Sử dụng kỹ thuật đệ quy có nhớ, trong nhiều trường hợp, ta giữ nguyên được cấu trúc đệ quy của thuật toán và đồng thời lại đảm bảo được hiệu quả của nó. Nhược điểm lớn nhất của cách làm này là đòi hỏi về bộ nhớ.

ĐỆ QUY CÓ NHỚ

- Ý tưởng của đệ quy có nhớ:
  - Dùng biến ghi nhớ lại thông tin về lời giải của các bài toán con ngay sau lần đầu tiên nó được giải. Điều đó cho phép rút ngắn thời gian tính của thuật toán, bởi vì, mỗi khi cần đến có thể tra cứu mà không phải giải lại những bài toán con đã được giải trước đó.

ĐỆ QUY CÓ NHỚ

- Ví dụ 1: Tính dãy số Fibonacci:

$F(0) = 0; F(1) = 1$

$F(n) = F(n-1) + F(n-2)$ với $n \geq 2$

Đệ quy không có nhớ:

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

Đệ quy có nhớ:

```
void init() {
   M[0] = 0; M[1] = 1;
   for (int i = 2; i <= n; i++) M[i] = 0;
}
int F(int n){
  if (n!= 0 && M[n] == 0)
      M[n] = F(n-1) + F(n-2);
  return M[n];
}
```

Trước khi gọi hàm F(*n*) cần gọi hàm init() để khởi tạo các phần tử trong mảng M[ ] như sau:

M[0] = 0, M[1]=1,

M[*i*] = 0, với $i \geq 2$.

ĐỆ QUY CÓ NHỚ

- Ví dụ 2: Tính hệ số nhị thức:

$C(0,n) = 1,\ C(n,n) = 1;$ với mọi $n \geq 0,$

$C(k,n) = C(k-1,n-1) + C(k,n-1),\ 0 < k < n$

Đệ quy không có nhớ:

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

Đệ quy có nhớ:

```
void init() {
   for (int i = 0; i <= k; i++)
     for(int j = 0; j <= n; j++) M[i][j] = 0;
}
int C(int k, int n) {
   if (k == 0 || k == n) M[k][n] = 1;
   else {
     if(M[k][n] == 0) M[k][n] = C(k-1,n-1) + C(k,n-1);
   }
   return M[k][n];
}
```

Trước khi gọi hàm C(*k*, *n*) cần gọi hàm init() để khởi tạo các phần tử trong mảng M[ ][ ] = 0

HUST

hust.edu.vn fb.com/dhbkhn

THANK YOU !