CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Đệ quy

Nội dung chính

- Khái niệm
  - Sự đệ quy
  - Giải thuật đệ quy
    - Cấu tạo giải thuật đệ quy
    - Hoạt động của giải thuật đệ quy
- Xây dựng thủ tục đệ quy
  - Thủ tục đệ quy
  - Phương pháp xây dựng
    - Cấu trúc thủ tục đệ quy
    - Hoạt động của thủ tục đệ quy
    - Nguyên tắc cài đặt
- Sự khử đệ quy

Nội dung chính

- Các ví dụ minh họa
  - Tìm kiếm trong danh sách liên kết
  - Bài toán Tháp Hà Nội
  - Bài toán 8 con hậu
- Đánh giá thời gian thực hiện giải thuật
  - Khái niệm
  - Các ký hiệu
  - Các quy tắc đánh giá
  - Phân tích một số giải thuật

Khái niệm đệ quy

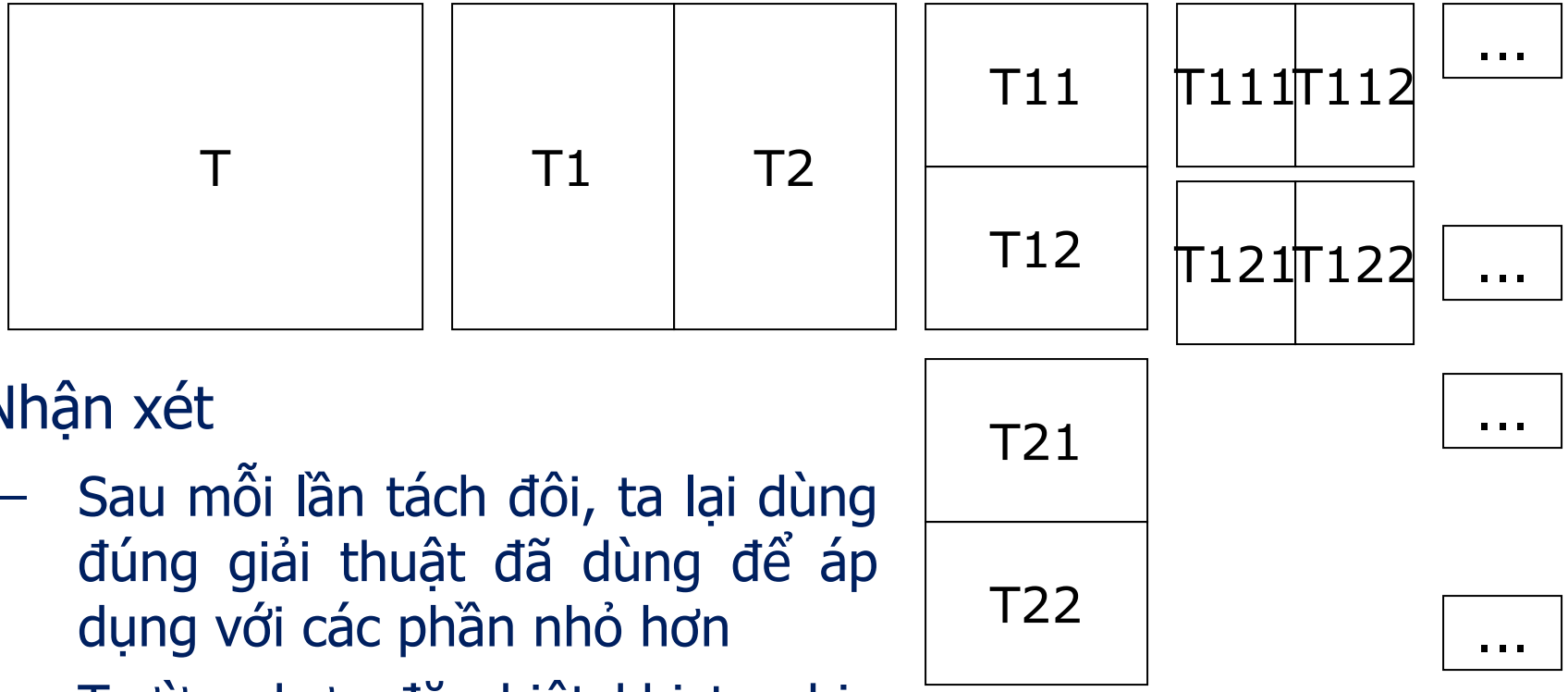
- Các ví dụ:
  - String
    - Quy tắc 1: 1 char = String
    - Quy tắc 2: String = 1 char + (sub) String
  - Số tự nhiên
    - Quy tắc 1: 1 ∈ N
    - Quy tắc 2: x ∈ N if (x-1) ∈ N
  - N!
    - Quy tắc 1: 0! = 1
    - Quy tắc 2: n! = n (n-1)!
  - Định nghĩa danh sách tuyến tính:
    - Quy tắc 1: L = rỗng là một DSTT
    - Quy tắc 2: Nếu Ln-1 là một danh sách kích thước n-1 thì cấu trúc Ln=<a,Ln-1> cũng là một DSTT, với a là một phần tử có cùng kiểu dữ liệu như các phần tử trong Ln-1, và a đứng trước Ln-1 trong danh sách Ln

Khái niệm đệ quy

- Đối tượng đệ quy: một đối tượng được gọi là đệ quy nếu nó bao gồm chính nó như một bộ phận hoặc nó được định nghĩa dưới dạng của chính nó.
- Kiểu đệ quy: là cách định nghĩa về một đối tượng/khái niệm mà dựa vào một hay một tập các đối tượng/khái niệm có cùng bản chất với đối tượng/khái niệm cần định nghĩa nhưng có quy mô nhỏ hơn
- Tính chất đệ quy: các đối tượng/khái niệm có thể định nghĩa một cách đệ quy thì ta gọi chúng có tính chất đệ quy. Tức là các đối tượng/khái niệm này phải chứa các thành phần có cấu trúc tương tự như chính nó, chỉ khác là quy mô nhỏ hơn.

Giải thuật đệ quy

- Ví dụ 1: Tìm phần tử trong một tập hợp (tìm từ trong từ điển)



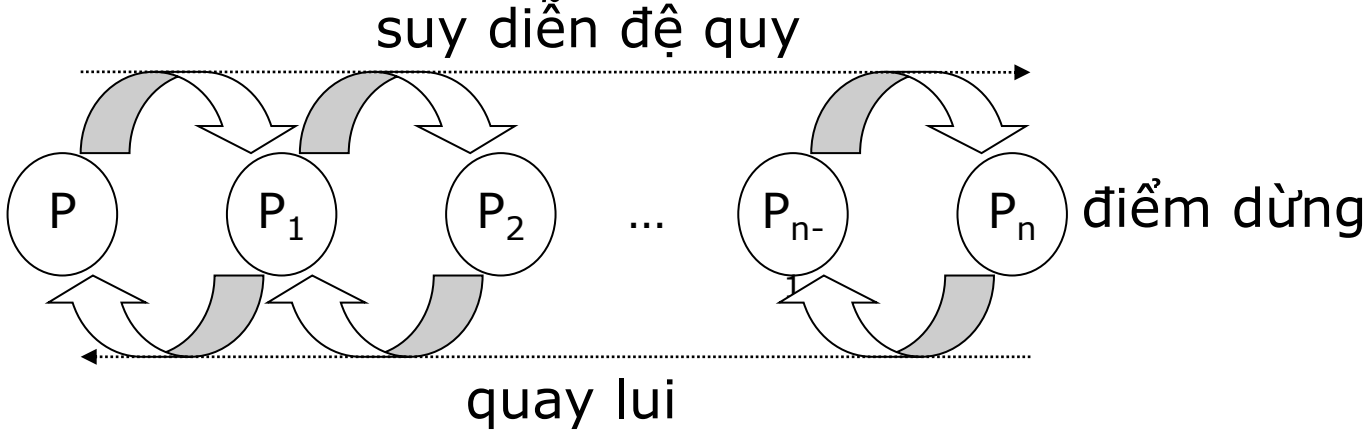
- Nhận xét
  - Sau mỗi lần tách đôi, ta lại dùng đúng giải thuật đã dùng để áp dụng với các phần nhỏ hơn
  - Trường hợp đặc biệt khi ta chia nhỏ đến 1 trang. Khi đó việc tìm kiếm sẽ trở nên dễ dàng và có thể thực hiện trực tiếp

Giải thuật đệ quy

- Giải thuật đệ quy (recursive algorithm)
  - Ví dụ 2: Tính giá trị dãy số Fibonacci
    - F (1) = 1 với n <= 2;
    - F (n) = F (n-1) + F (n-2) với n > 1;
  - Nhận xét:
    - Thay vì tính F (n) ta quy về tính các giá trị hàm F với biến n nhỏ hơn là F(n-1) và F(n-2).
    - Cách tính các hàm F(n-1), F(n-2) này cũng giống như cách tính F(n), có nghĩa là cũng quy về việc tính các hàm F với biến n nhỏ hơn nữa
    - Trường hợp đặc biệt: khi n đủ nhỏ (n <= 2) ta tính được trực tiếp giá trị của hàm F (1) = 1, F (2) = 1

Giải thuật đệ quy

- Từ P, đưa về bài toán P1 có bản chất tương tự P nhưng có quy mô bé hơn, nếu giải được P1 thì ta sẽ giải được P.
- Tương tự, từ P1 đưa về giải bài toán P2 có cùng bản chất với P1 và có quy mô nhỏ hơn P1. Quá trình cứ tiếp tục cho đến khi đưa bài toán về bài toán con Pn tương tự như Pn-1 và có quy mô nhỏ hơn Pn-1
- Pn có thể giải một cách trực tiếp
- Sau khi giải được Pn, quay lại giải các bài toán con theo trật tự ngược lại, cuối cùng giải được bài toán ban đầu P.



Giải thuật đệ quy

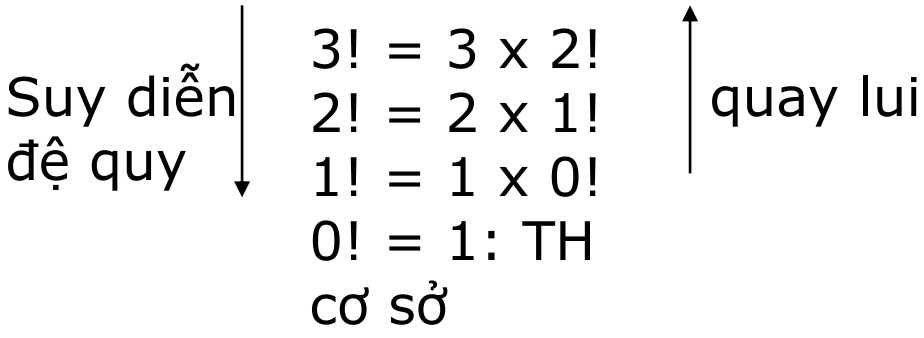
- Ví dụ 3: Tìm số nhỏ nhất trong một dãy N số a1, a2, …, aN
  - 1) Nếu N=1 thì min=a1;
  - 2) Chia dãy ban đầu thành hai dãy con:
    L1 = a1, a2,…, am và L2 = am+1, am+2,…, aN
    với m=(1+N) DIV 2.
    Tìm min1 trong dãy L1 và min2 trong dãy L2.
    So sánh min1 và min2 để tìm ra min của dãy ban đầu
- Lưu ý:
  - Trong trường hợp khái quát hơn, từ bài toán ban đầu ta có thể phải đưa về nhiều bài toán con tương tự.
  - Trong trường hợp không có trường hợp đặc biệt, giải thuật thường dễ rơi vào vòng lặp vô hạn (không có tính dừng) và bài toán không thể giải được. Ví dụ vòng lặp vô hạn - Ví dụ 4: Tính giá trị dãy số Fibonacci
    F (n) = F (n-1) + F (n-2) ;

Giải thuật đệ quy

- Cấu tạo giải thuật đệ quy:
  - Trường hợp cơ sở: là trường hợp bài toán Pn, có quy mô đủ nhỏ để ta có thể giải trực tiếp. Nó đóng vai trò điểm dừng trong quá trình suy diễn đệ quy, và cũng quyết định tính dừng của giải thuật đệ quy.
  - Trường hợp đệ quy: là trường hợp khái quát chứa cơ chế đệ quy, là cơ chế đưa bài toán cần giải về một hay nhiều bài toán tương tự nhưng có quy mô nhỏ hơn. Cơ chế này được áp dụng nhiều lần để thu nhỏ quy mô bài toán cần giải, từ bài toán ban đầu cho đến bài toán nhỏ nhất ở trường hợp cơ sở.
  - Hai trường hợp trên có mối quan hệ khăng khít để tạo thành giải thuật đệ quy và đảm bảo giải thuật đệ quy có thể giải được và có điểm dừng.

Giải thuật đệ quy

- Hoạt động của giải thuật bao gồm hai quá trình:
  - Quá trình suy diễn đệ quy: là quá trình thu nhỏ bài toán ban đầu về các bài toán trung gian tương tự nhưng có quy mô giảm dần bằng cách áp dụng cơ chế đệ quy, cho đến khi gặp trường hợp cơ sở (điểm dừng của quá trình suy diễn đệ quy).
  - Quá trình quay lui (hay suy diễn ngược): là quá trình từ kết quả thu được trong trường hợp cơ sở, thực hiện giải các bài toán trung gian ở quá trình suy diễn đệ quy theo trật tự ngược lại, cho đến khi giải quyết được bài toán ban đầu.
  - Ví dụ 5: tính 3! theo giải thuật đệ quy



Thủ tục đệ quy

- Khái niệm: để cài đặt các giải thuật đệ quy một cách đơn giản và hiệu quả, đa số các ngôn ngữ lập trình hiện nay đều có cơ chế, công cụ để hỗ trợ công việc này, đó là thủ tục đệ quy.
- Thủ tục đệ quy là một chương trình con trực tiếp cài đặt cho giải thuật đệ quy. Tuỳ theo ngôn ngữ lập trình, nó có thể có các tên gọi khác nhau như: trong Pascal ta có thể có thủ tục đệ quy hay hàm con đệ quy, trong C/C++ chỉ có hàm con đệ quy, …

Thủ tục đệ quy

- Ví dụ 6: Tính giá trị n!
  - 0! = 1
  - n! = n (n-1)!

```
int Fact (int n){
        if  (n <= 1) return 1;
        else return n * Fact (n-1);
}
```

- Quay lại ví dụ 2: Tính giá trị dãy Fibonacci
  - F (1) = 1 với n <= 2;
  - F (n) = F (n-1) + F (n-2) với n > 1;

```
int Fibo1 (int n) {
        if (n <= 2) return 1;
        else return (Fibo1 (n-1) + Fibo1 (n-2));
}
```

Xây dựng thủ tục đệ quy

- Trong thủ tục đệ quy có lời gọi đến chính thủ tục đó
- Mỗi lần thực hiện gọi lại thủ tục thì kích thước của bài toán (tham số n) lại thu nhỏ hơn trước: bài toán được chia nhỏ
- Trường hợp cơ sở - Trường hợp suy biến (degenerate case): các trường hợp mà ta có thể giải quyết bài toán một cách khác, một cách dễ dàng và hiển nhiên (ở 2 ví dụ trên, các hàm được tính một cách dễ dàng) và bài toán kết thúc.
- Việc chia nhỏ dần bài toán đảm bảo dẫn tới tính trạng suy biến nói trên

Xây dựng thủ tục đệ quy

- Cấu trúc của thủ tục đệ quy:

```
void  P (A) {
    if A==A0 then
        Trường hợp cơ sở
    else {                  //Trường hợp đệ quy
        Q1();
        P(A1);  //Lời gọi đệ quy
        Q2();
        P(A2);  //Lời gọi đệ quy
    }
}
```

Xây dựng thủ tục đệ quy

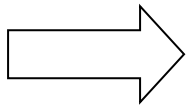
- Phần đầu thủ tục: thủ tục đệ quy luôn luôn phải có tham số. Ngoài vai trò biểu diễn các thông tin vào/ra như các thủ tục thông thường không đệ quy, tham số của thủ tục đệ quy còn biểu diễn quy mô bài toán cần giải. Tham số này phải có khả năng thu nhỏ khi gọi đệ quy và có quy mô nhỏ nhất khi đến trường hợp cơ sở.
- Phần thân thủ tục: bao gồm hai nhánh của một cấu trúc rẽ nhánh tương ứng với hai trường hợp của giải thuật đệ quy:
  - Trường hợp cơ sở: khi quy mô bài toán suy biến đủ nhỏ, bài toán có thể được giải trực tiếp, nên trong trường hợp này sẽ chứa các lệnh thi hành việc giải trực tiếp này.
  - Trường hợp đệ quy: nó chứa một hay nhiều lời gọi đệ quy, là lời gọi đến chính thủ tục đang xây dựng, nhưng có các tham số khác với (mà thường là nhỏ hơn) tham số ban đầu.

Xây dựng thủ tục đệ quy

- Thủ tục đệ quy: gọi đến chính nó theo 1 trong 2 cách
  - Gọi trực tiếp (Đệ quy trực tiếp)
    Ví dụ: xem ví dụ 6, ví dụ 2
  - Gọi gián tiếp: gọi tới chính nó thông qua việc gọi đến một thủ tục (hàm) khác

    Ví dụ:

```
void RecursiveA () {
        ...
        ProcedureB();
        ...
}
```



```
void ProcedureB (){
        ...
        RecursiveA();
        ...
}
```

Xây dựng thủ tục đệ quy

- Thủ tục đệ quy hoạt động theo hai giai đoạn:
  - Giai đoạn gọi đệ quy:
    - Bắt đầu từ lời gọi thủ tục đầu tiên, CT sẽ đi theo nhánh gọi đệ quy trong thân thủ tục để liên tục gọi các lời gọi đệ quy trung gian cho đến khi gặp trường hợp cơ sở.
    - Khi đến điểm dừng này, hệ thống sẽ tự động kích hoạt giai đoạn thứ hai là giai đoạn quay lui.
  - Giai đoạn quay lui:
    - Hệ thống sẽ thi hành các thủ tục đệ quy trung gian trong giai đoạn đầu theo thứ tự ngược lại, cho đến khi thi hành xong thủ tục được gọi đầu tiên thì kết thúc, đồng thời kết thúc hoạt động của thủ tục đệ quy.
- Vấn đề cài đặt: làm thế nào hệ thống lưu giữ các lời gọi đệ quy trung gian và các kết quả xử lý trung gian trong giai đoạn gọi đệ quy để có thể lấy chúng ra để xử lý trong giai đoạn quay lui?

Xây dựng thủ tục đệ quy

- Việc lưu trữ, xử lý các lời gọi đệ quy có đặc điểm sau:
  - Tính vào sau ra trước (LIFO):
    - Việc lưu trữ các lời gọi theo đúng thứ tự của quá trình gọi đệ quy, lời gọi trước lưu trữ trước và ngược lại.
    - Trong quá trình quay lui, các lời gọi được lấy ra theo thứ ngược lại để xử lý => Sử dụng cấu trúc ngăn xếp để lưu trữ các lời gọi đệ quy.
  - Tính đồng nhất:
    - Do cấu trúc các lời gọi đệ quy và các kết quả trung gian giống nhau, nên cấu trúc ngăn xếp được sử dụng cũng có tính đồng nhất.
  - Trong thực tế
    - Các ngôn ngữ lập trình có hỗ trợ cài đặt thủ tục đệ quy đều đã tự động cài đặt các cấu trúc ngăn xếp thích hợp để phục vụ cho quá trình cài đặt và hoạt động của thủ tục đệ quy.

Xây dựng thủ tục đệ quy

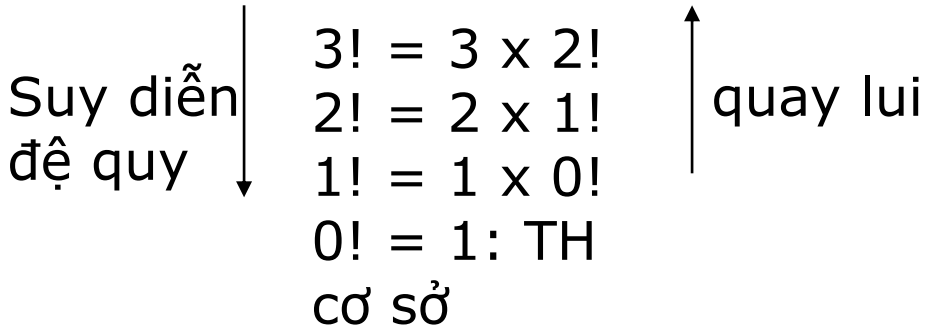
- Hoạt động của thủ tục đệ quy trong chương trình:

(1) Khởi tạo ngăn xếp
(2) Giai đoạn gọi đệ quy
  - Bắt đầu từ lời gọi thủ tục đầu tiên, đi theo nhánh gọi đệ quy trong thân thủ tục để liên tục gọi các lời gọi đệ quy trung gian cho đến khi gặp trường hợp cơ sở.
  - Song song với quá trình gọi đệ quy, hệ thống sẽ lưu các lời gọi đệ quy và các kết quả trung vào ngăn xếp.
  - Khi đến điểm dừng, hệ thống sẽ tự động kích hoạt giai đoạn quay lui.
(3) Giai đoạn quay lui
  - Hệ thống lần lượt lấy các lời gọi đệ quy và các kết quả trung gian trong ngăn xếp ra để xử lý cho đến khi hết ngăn xếp.
  - Khi đó, giai đoạn quay lui kết thúc, đồng thời cũng kết thúc hoạt động của thủ tục đệ quy. Kết quả cuối cùng sẽ là kết quả của thủ tục đệ quy.

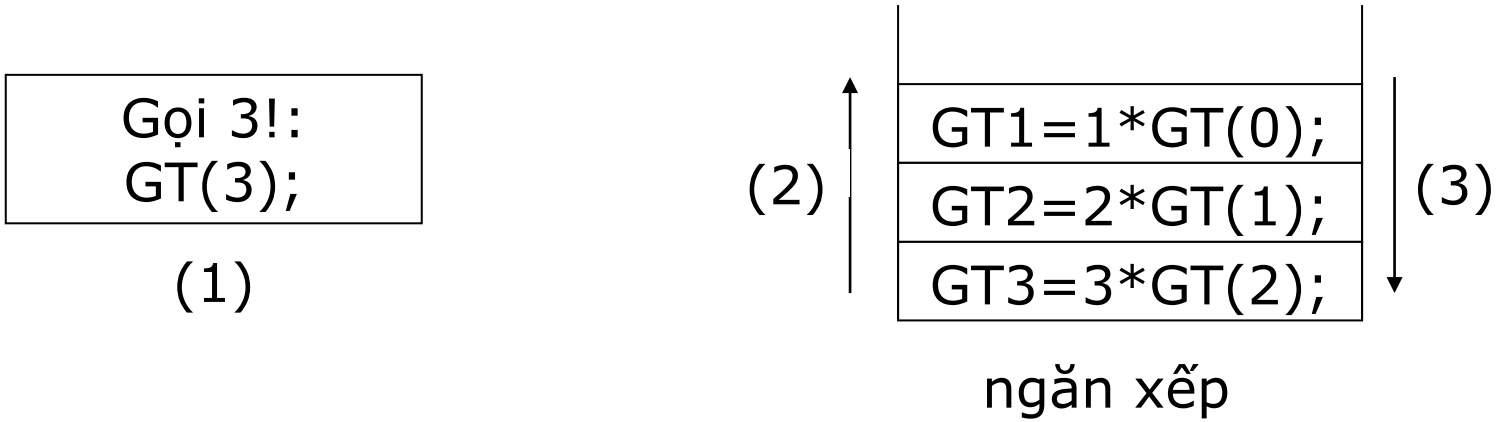
Xây dựng thủ tục đệ quy

- Ví dụ: minh hoạ hoạt động của thủ tục đệ quy tính 3!.

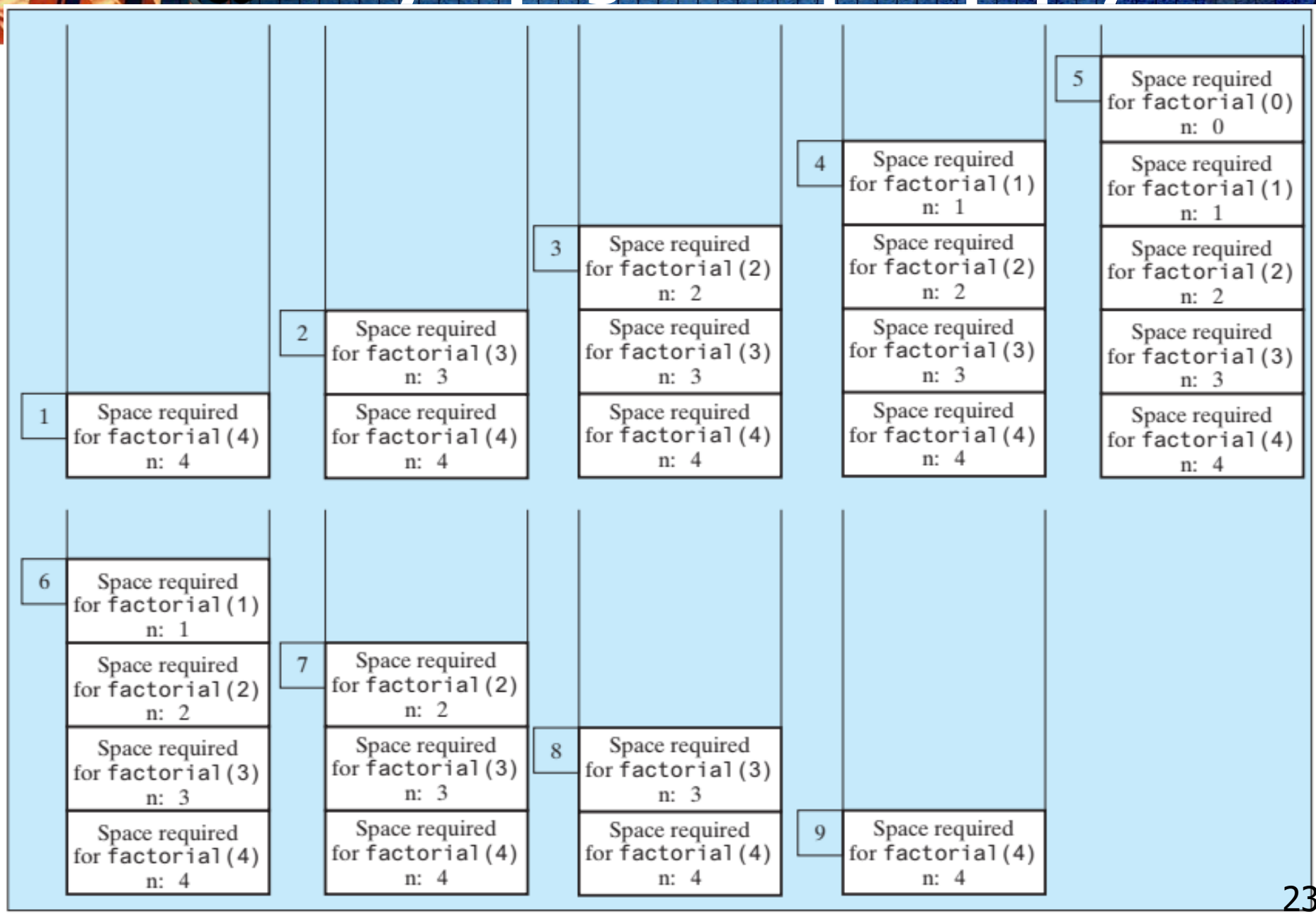
Giải thuật đệ quy



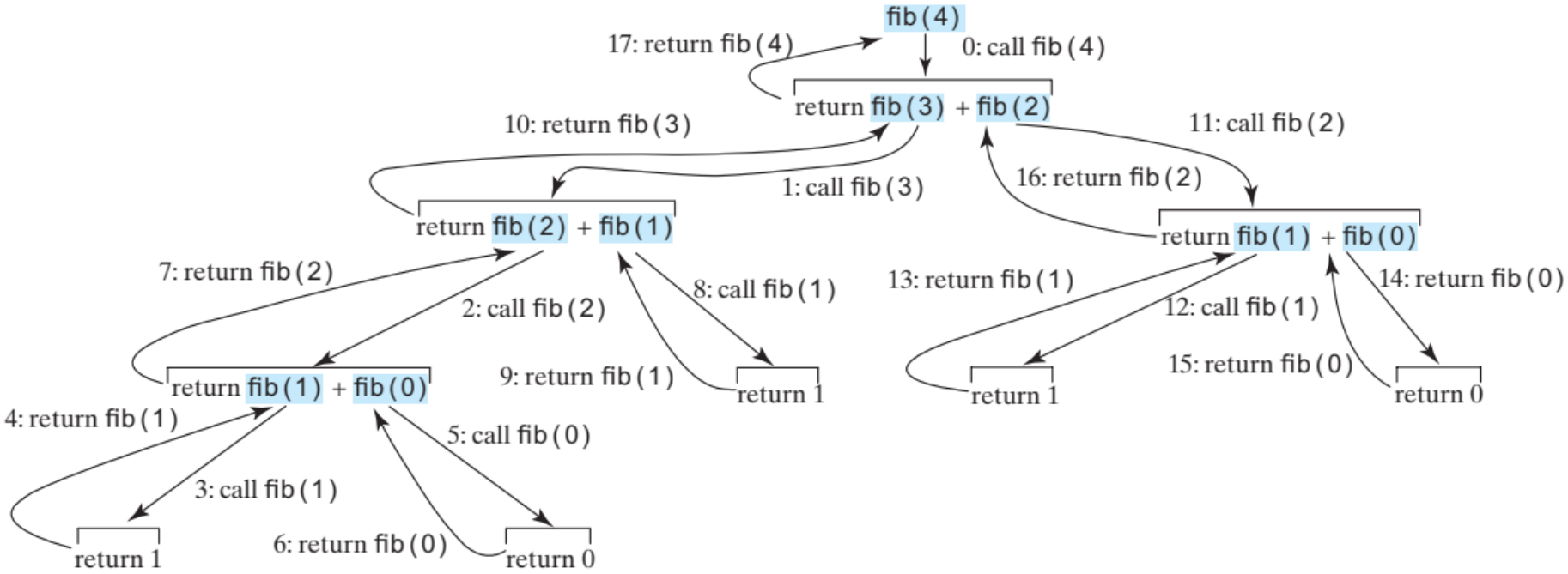
Hoạt động của thủ tục đệ quy



Xây dựng thủ tục đệ quy



Xây dựng thủ tục đệ quy



Xây dựng thủ tục đệ quy

- Ưu điểm: viết chương trình dễ dàng, dễ hiểu, ngắn gọn
- Nhược điểm:
  - Thời gian thực hiện: tốn thời gian
  - Bộ nhớ: tốn bộ nhớ
- Các vấn đề khác: không phải lúc nào cũng có thể xây dựng bài toán theo giải thuật và thủ tục đệ quy một cách dễ dàng, các vấn đề có thể là:
  - Có thể định nghĩa bài toán dưới dạng bài toán cùng loại nhưng nhỏ hơn như thế nào?
  - Làm thế nào để đảm bảo kích thước bài toán giảm đi sau mỗi lần gọi?
  - Xem xét và định nghĩa các trường hợp đặc biệt (trường hợp suy biến) như thế nào?

Sự khử đệ quy

- Khi thay các giải thuật đệ quy bằng các giải thuật không tự gọi chúng, ta gọi đó là sự khử đệ quy
- Sự khử đệ quy thông qua các vòng lặp (for, while) và phân nhánh (if...then...else)
- Ví dụ 1:

```
int Fact (int n){
        if  (n <= 1) return 1;
        else return n * Fact (n-1);
}
```

```
int Fact (int n){
    if  (n <= 1) return 1;
    else {
        int x=1;
        for (int i=2;i<=n;i++) x*=i;
        return x;
    }
}
```

Sự khử đệ quy

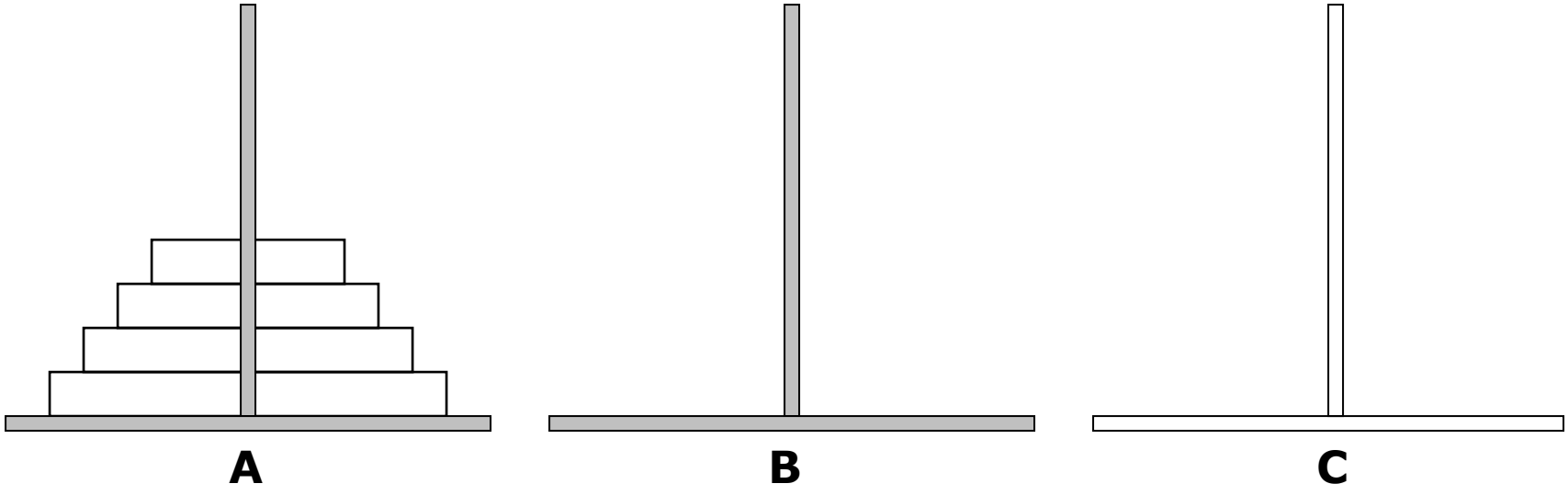
- Ví dụ 2:

```
int Fibo1 (int n){
    if (n <= 2) return 1;
    else
        return (Fibo1 (n-1) + Fibo1 (n-2));
}
```

```
int  Fibo2 (int n) {
    int i, f, f1, f2;
    f1 = 1 ;
    f2 = 1 ;
    if  (n <= 2)  f = 1;
    else for (i = 3;i<=n;i++){
            f = f1 + f2;
            f1 = f2;
            f2 = f;
    }
    return f;
}
```

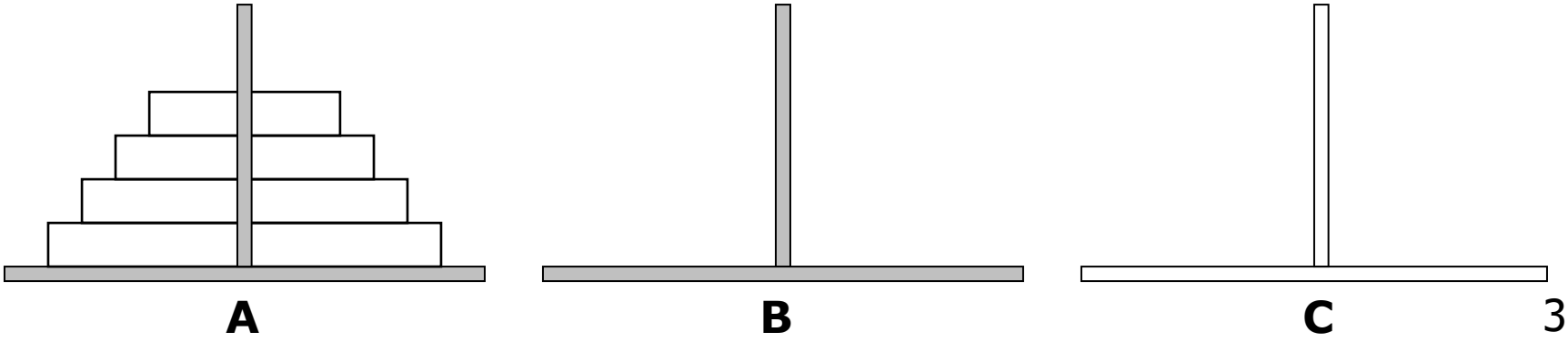
Ví dụ minh họa

- Bài toán Tháp Hà Nội:
  - Chuyển cột đĩa từ A sang B
  - Mỗi lần chỉ được chuyển 1 đĩa
  - Không được đĩa to trên đĩa nhỏ, dù chỉ tạm thời
  - Được phép chuyển qua một cột trung gian C



Ví dụ minh họa

- Bài toán Tháp Hà Nội
  - Xác định trường hợp đơn giản (suy biến)
    - 1 đĩa: A -> B
    - 2 đĩa: A -> C, A -> B, C -> B
  - Tổng quát: quy về trường hợp 1 hoặc 2 đĩa:
    - 1, Quy về bài toán chuyển (n-1) đĩa A sang C
    - 2, Quy về bài toán chuyển 1 đĩa A sang B
    - 3, Quy về bài toán chuyển (n-1) đĩa C sang B



Ví dụ minh họa

- Bài toán Tháp Hà Nội

```
Procedure Tower(n, A, B, C)
If n=1 then chuyen dia tu A sang B
Else begin
        call Tower(n-1, A, C, B);
        call Tower(1, A, B, C);
        call Tower(n-1, C, B, A)
        end
End.
```

```cpp
#include <iostream>
using namespace std;
void move(int n, char A, char B, char C)
//A nguon, B dich, C trung gian
{
    if (n == 1) {
        cout << A << " ==> " << B << "\n";
    }
    else {
        move(n - 1, A, C, B);
        cout << A << " ==> " << B << "\n";
        move(n - 1, C, B, A);
    }
}
int main() {
    move(3, 'A', 'B', 'C');
}
```