Phần 3: CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Chương 10:
Giải thuật và thủ tục đệ quy

Nội dung

- 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

- 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

- Khái niệm về đệ quy (recursion / recursive)
  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 là N
    - Quy tắc 2: x là N if (x-1) là 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 L_{n-1} là một danh sách kích thước n-1 thì cấu trúc L_n=<a,L_{n-1}> cũng là một DSTT, với a là một phần tử có cùng kiểu dữ liệu như các phần tử trong L_{n-1}, và a đứng trước L_{n-1} trong danh sách L_n

Khái niệm đệ quy

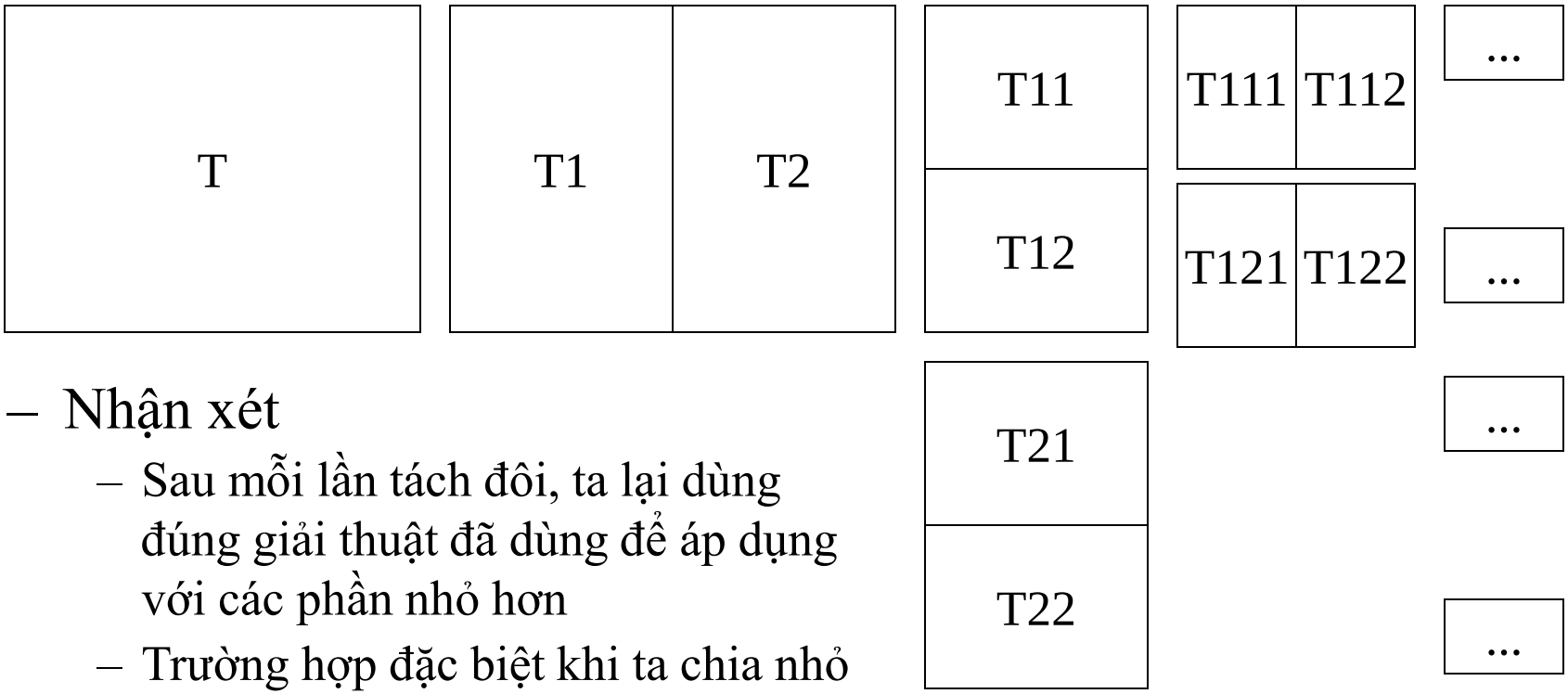
- Các khái niệm về đệ quy (recursion / recursive)
  1. **Khái niệm:** (sách *"Cấu trúc dữ liệu giải thuật"*, tác giả *Đỗ Xuân Lôi*) ta gọi một đối tượng là đệ quy nếu nó bao gồm chính nó như một bộ phận hoặc nó được định nghĩa dưới dạng của chính nó. Đệ quy bao gồm:
     - Các quy tắc cơ sở (basic rules)
     - Các quy tắc quy nạp (inductive rules)
  2. **Khái quát**

– **Khái niệm định nghĩa 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.

Khái niệm đệ quy

- Giải thuật đệ quy (recursive algorithm)
  - Ví dụ 1: Tìm phần tử trong một tập hợp (tìm từ trong từ điển)



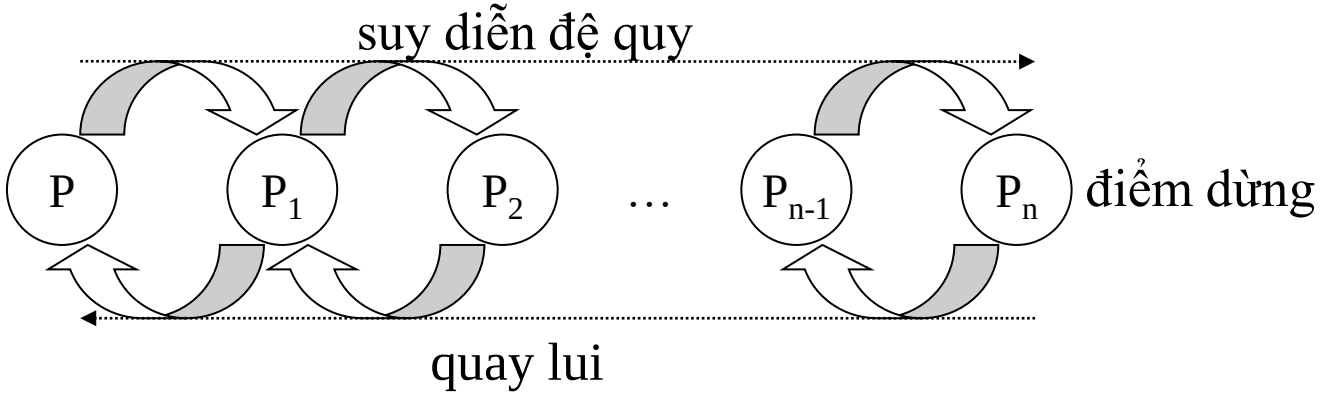
  - Nhận xét
    - Sau mỗi lần tách đôi, ta lại dùng đúng giải thuật đã dùng để áp dụng với các phần nhỏ hơn
    - Trường hợp đặc biệt khi ta chia nhỏ đến 1 trang. Khi đó việc tìm kiếm sẽ trở nên dễ dàng và có thể thực hiện trực tiếp

Khái niệm 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

Khái niệm đệ quy

- Giải thuật đệ quy (recursive algorithm)
  - Khái niệm:
    - Từ P ta đưa về một bài toán P_1 có bản chất tương tự P nhưng có quy mô bé hơn. Nghĩa là tồn tại một quan hệ giữa P và P_1, khẳng định nếu giải được P_1 thì ta sẽ giải được P.
    - Tương tự, từ P_1 ta lại đưa về giải bài toán P_2 có cùng bản chất với P_1 và cũng có quy mô nhỏ hơn P_1. Quá trình cứ tiếp tục cho đến khi ta đưa bài toán về bài toán con P_n tương tự như P_{n-1} và có quy mô nhỏ hơn P_{n-1}
    - Pn có thể giải một cách trực tiếp, tức là không cần đưa về bài toán tương tự có quy mô nhỏ hơn nữa do Pn đã đủ nhỏ và đơn giản.
    - Sau khi giải được Pn, ta quay lại giải các bài toán con theo trật tự ngược lại và cuối cùng giải được bài toán ban đầu P.



Khái niệm đệ quy

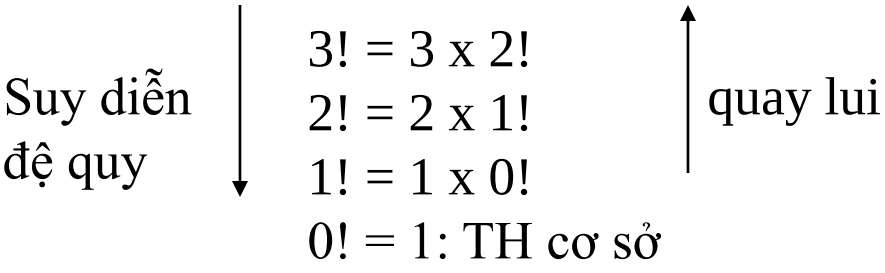
- Giải thuật đệ quy (recursive algorithm)
  - Ví dụ 3: Tìm số nhỏ nhất trong một dãy N số a_1, a_2, …, a_N
    - 1) Nếu N=1 thì min=a_1;
    - 2) Chia dãy ban đầu thành hai dãy con:
      $L_1 = a_1, a_2, \ldots, a_m$ và $L_2 = a_{m+1}, a_{m+2}, \ldots, a_N$
      với m=(1+N) DIV 2.
      Tìm min_1 trong dãy L_1 và min_2 trong dãy L_2.
      So sánh min_1 và min_2 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) ;

Khái niệm đệ quy

- Giải thuật đệ quy (recursive algorithm)
  - Cấu tạo giải thuật đệ quy: gồm hai thành phần sau:
    - *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.

Khái niệm đệ quy

- Giải thuật đệ quy (recursive algorithm)
  - 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



Xây dựng thủ tục 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, …
  - Để thực hiện được trường hợp đệ quy trong giải thuật đệ quy, thủ tục đệ quy sẽ cung cấp khả năng *gọi đệ quy*, là lời gọi đến chính thủ tục đệ quy đang được định nghĩa.

Xây dựng thủ tục 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;

Gọi đệ quy

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

Xây dựng thủ tục đệ quy

- Thủ tục đệ quy - Phương pháp xây dựng

  Nhận xét: qua các ví dụ trên ta có thể rút ra các đặc điểm chung cho các thủ tục (hàm) đệ quy như sau:

  – 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

- Thủ tục đệ quy – Phương pháp xây dựng
  - **Cấu trúc của thủ tục đệ quy**: Viết dưới dạng tựa C cấu trúc chung của một thủ tục đệ quy như sau:

```
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

- Thủ tục đệ quy - Phương pháp xây dựng

  Nhận xét cấu trúc của thủ tục đệ quy:

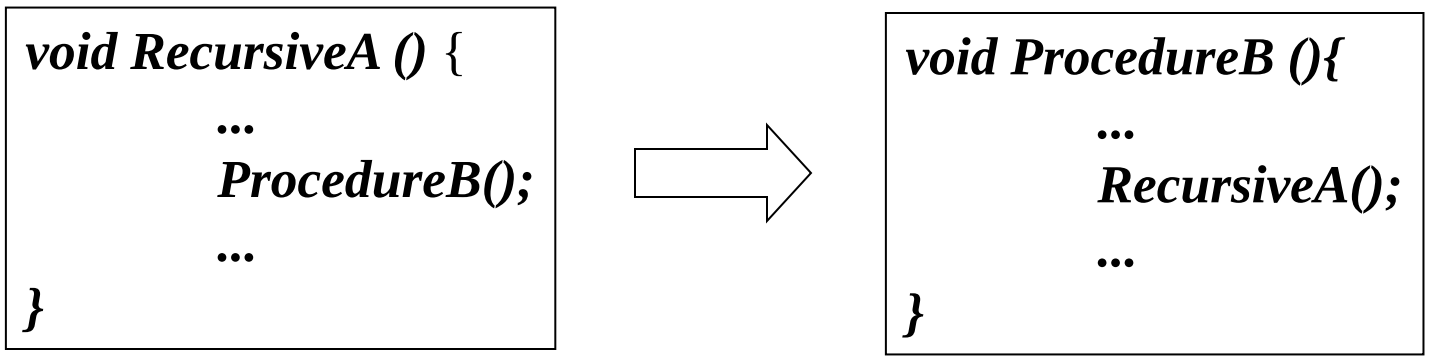
  – Phần đầu thủ tục: thủ tục đệ quy luôn luôn phải có tham số. Ngoài vai trò biểu diễn các thông tin vào/ra như các thủ tục thông thường không đệ quy, tham số của thủ tục đệ quy còn biểu diễn quy mô bài toán cần giải. Tham số này phải có khả năng thu nhỏ khi gọi đệ quy và có quy mô nhỏ nhất khi đến trường 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 - Phương pháp xây dựng
  - Thủ tục đệ quy: gọi đến chính nó
  - Trong một số trường hợp, việc gọi đến chính nó có thể được thực hiện qua hai cách:
    - Gọi trực tiếp (2 ví dụ trên): ta có Đệ quy trực tiếp (***directly recursive***)
      Ví dụ: xem ví dụ 6, ví dụ 2
    - Gọi gián tiếp: gọi tới nó thông qua việc ta gọi đến một thủ tục (hàm) khác và tại đó, ta mới gọi đến thủ tục (hàm) đệ quy. Ta có Đệ quy gián tiếp (***indirectly recursive***)
      Ví dụ:



Xây dựng thủ tục đệ quy

- Thủ tục đệ quy - Hoạt động của thủ tục đệ quy

  Quá trình hoạt động của thủ tục đệ quy gồm 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 có thể lưu giữ các lời gọi đệ quy trung gian và các kết quả xử lý trung gian trong giai đoạn gọi đệ quy để ta có thể lấy chúng ra để xử lý trong giai đoạn quay lui.

Xây dựng thủ tục đệ quy

- Thủ tục đệ quy - Nguyên tắc cài đặt thủ tục đệ quy

  Nhận xét: 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. Vậy cần 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

- Thủ tục đệ quy - Nguyên tắc cài đặt 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 thứ hai là giai đoạn quay lui.*

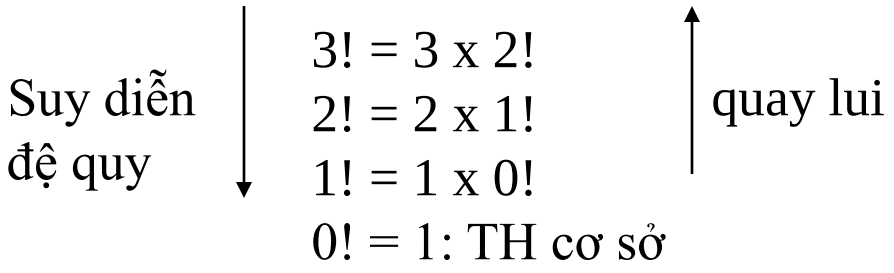
(3) Giai đoạn quay lui

- *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 và 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.*

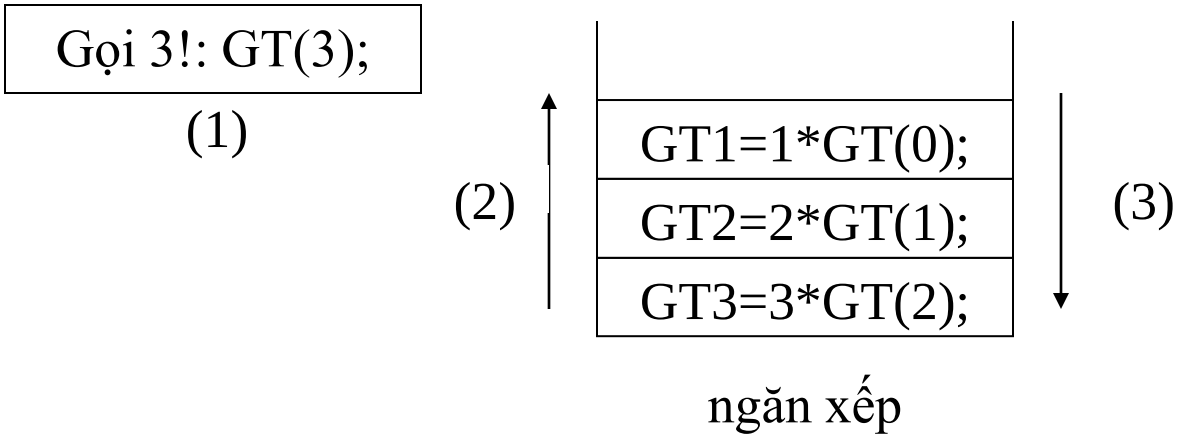
Xây dựng thủ tục đệ quy

- Thủ tục đệ quy - Nguyên tắc cài đặt 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

- Thủ tục đệ quy – Ưu nhược điểm
  - Ưu điểm: viết chương trình dễ dàng, dễ hiểu, ngắn gọn
  - Nhược điểm: theo nguyên tắc cài đặt của thủ tục đệ quy, có thể thấy các nhược điểm sau:
    - 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

- Khái niệm
  - 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;
}
```

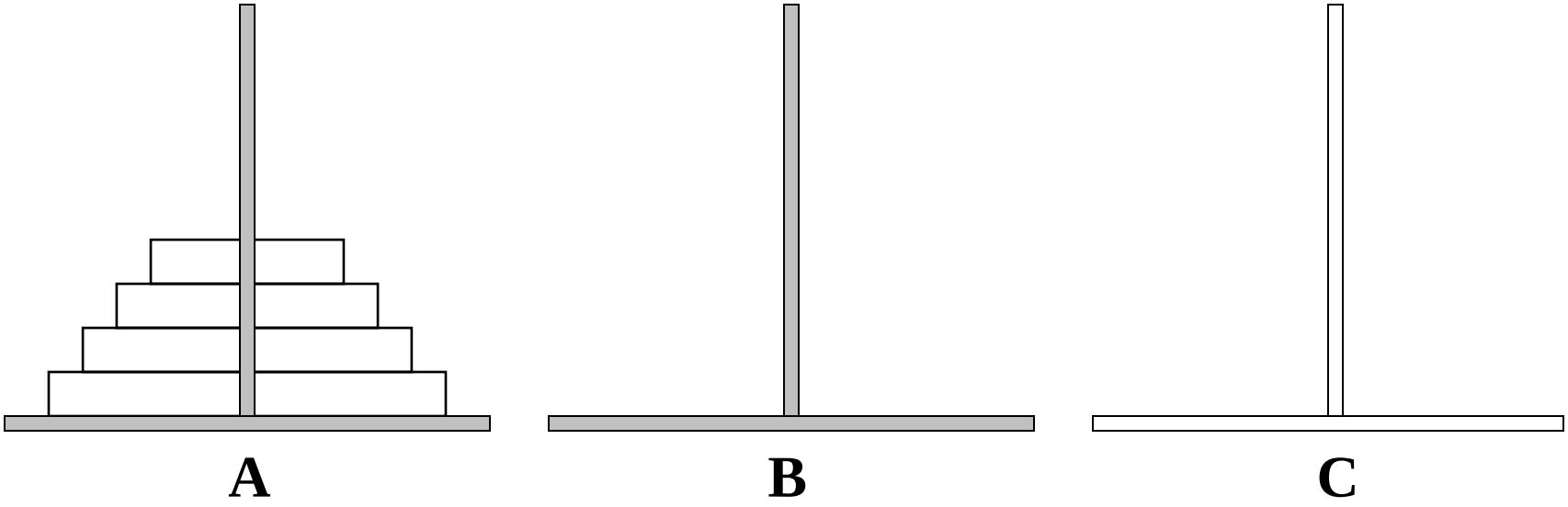
Các ví dụ minh họa

- Tìm kiếm trong danh sách liên kết:
  - Tìm kiếm một phần tử x trong danh sách có H là con trỏ hiện đang trỏ đến nút đầu tiên. Hàm trả về con trỏ trỏ vào nút tìm thấy; Trái lại nếu không tìm thấy thì trả về NULL.

```
PNode Search (Item x, PNode H) {
    if (H == NULL) return NULL;
    else
        if  (H->info == x) return H;
        else  return Search (x; H->next) ;
}
```

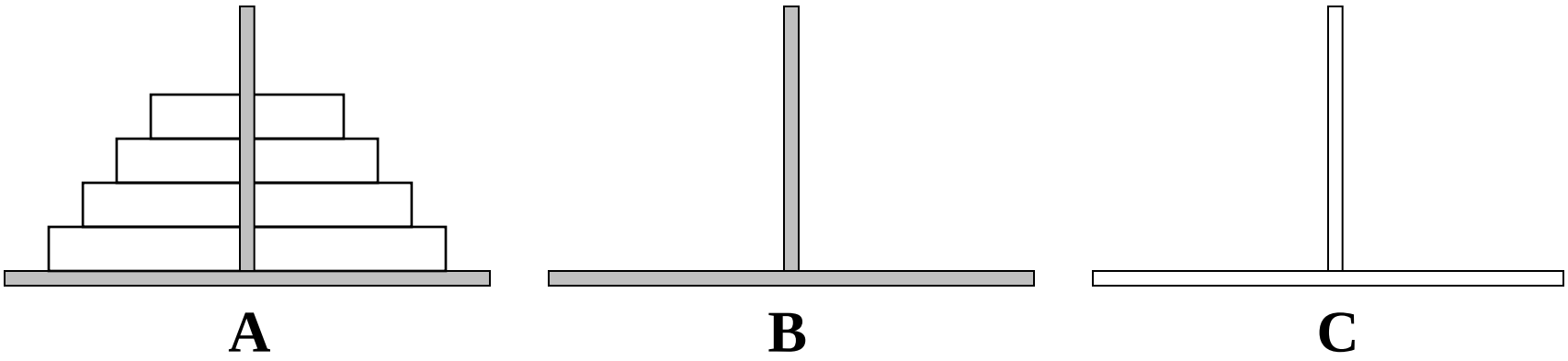
Các ví dụ minh họa

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



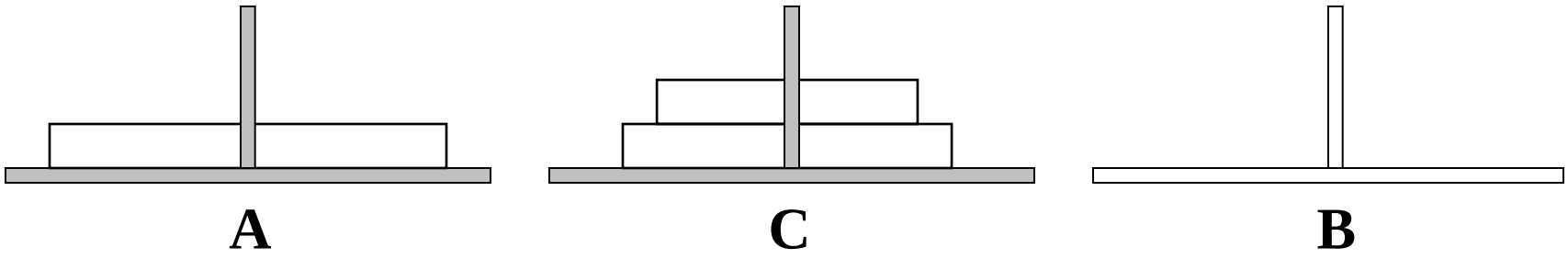
Các ví dụ minh họa

- 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

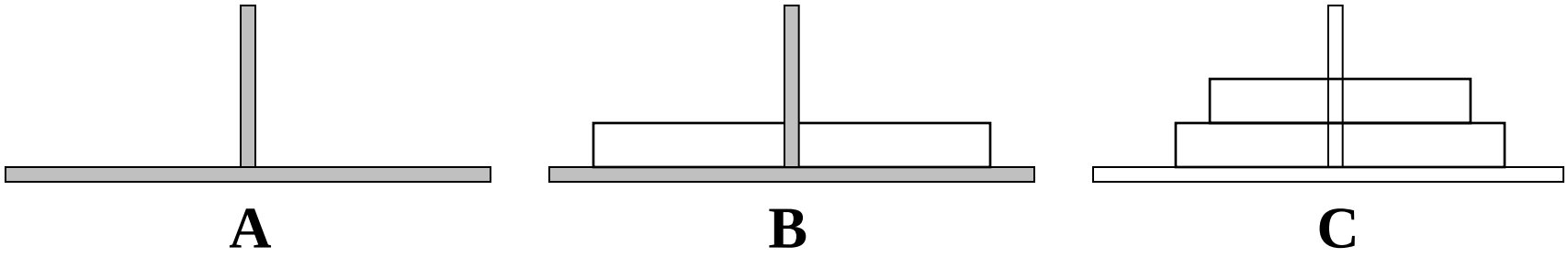


Các ví dụ minh họa

- Bài toán Tháp Hà Nội
  - Ví dụ 3 đĩa
    - 1, Quy về chuyển 2 đĩa trên cùng từ A sang C: ***A -> B, A -> C, B -> C***



    - 2, Quy về chuyển 1 đĩa từ A sang B: ***A -> B***



    - 3, Quy về chuyển 2 đĩa từ C sang B: ***C -> A, C -> B, A -> B***

Các ví dụ minh họa

- Bài toán Tháp Hà Nội
  - Trường hợp đơn giản (suy biến): 1 đĩa: A -> 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

```
void  TowerHN (int n, Tower A, B, C){
    if (n == 1) Transfer (A, B);
    else {
                    TowerHN (n-1, A, C, B);
                    TowerHN (1, A, B, C);
                    TowerHN (n-1, C, B, A);
    }
}
```

Giải thuật quay lui

- Ý tưởng giải thuật
- Cách cài đặt

Giải thuật quay lui

- Ý tưởng giải thuật:
  - *Bài toán khái quát*: cho một tập N biến $x_1, x_2, \ldots, x_N$ có thể nhận các giá trị tương ứng trong các tập giá trị hữu hạn $D_1, D_2, \ldots, D_N$. Yêu cầu hãy tìm một tập các giá trị thích hợp cho dãy N biến để thoả mãn một tiêu chuẩn K nào đó.
  - Giải thuật:
    - Đặt $V_i = <x_1, x_2, \ldots, x_i>$, với $i=1..N$.
    - Cần tìm vec tơ V_N thoả mãn điều kiện K, với $x_i \in D_i$ và $i=1..N$, hoặc trái lại khẳng định không có vec tơ nào thoả mãn.
    - Giải thuật sẽ thi hành trong N bước, mỗi bước ta sẽ tìm giá trị thích hợp cho một thành phần của vec tơ V.

Ý tưởng giải thuật (tiếp)

Giải thuật sẽ thực hiện đệ quy như sau:

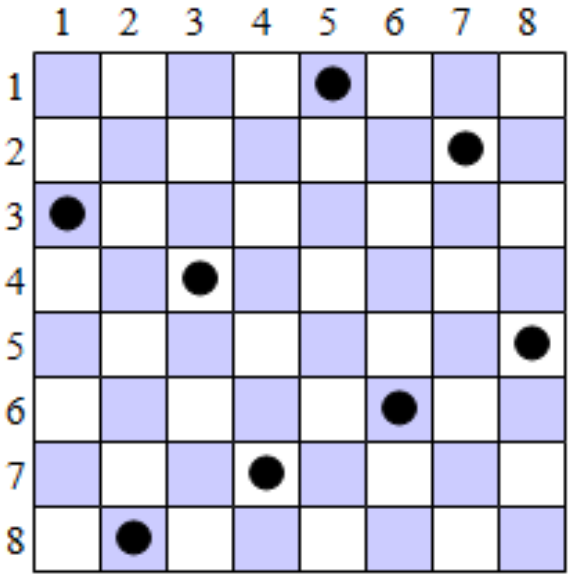
- Giả sử ta đã thực hiện đến bước thứ i, tức là $V_i = <x_1, x_2, \ldots, x_i>$ thoả mãn K. Nếu $i=N$ thì dừng, đưa ra kết quả là V_N. Trái lại, lặp lại giải thuật cho bước tiếp theo. Ở mỗi bước này, hai khả năng có thể xảy ra:
  - Nếu tìm được giá trị thích hợp, tức là $\exists p \in D_{i+1}$ sao cho với $x_{i+1}=p$ thì V_{i+1} thoả K, gán $x_{i+1}=p$. Lặp lại giải thuật ở bước $i+1$ (suy diễn đệ quy).
  - Trái lại, nếu $\forall p \in D_{i+1}$ ta đều có V_{i+1} không thoả K thì phải quay lại thử tiếp ở bước thứ i (quay lui). Tức là phải tìm vị trí thích hợp tiếp theo của x_i (vì x_i đang nhận giá trị thích hợp). Nếu không có vị trí tiếp theo thích hợp và $i=1$ thì giải thuật cũng kết thúc với kết luận không có giải pháp nào.

Ví dụ

- **Bài toán 8 con hậu**

  Hãy tìm cách bố trí 8 con hậu trên bàn cờ sao cho không có hai con hậu nào ăn được nhau. Bài toán này có thể mở rộng cho N con hậu.

- Ví dụ về một trong số 92 đáp án của bài toán:



Ví dụ: Bài toán 8 con hậu

- *Giải thuật*:

Ý tưởng chung:

- Coi bàn cờ là một mảng hai chiều a_{NxN}, (với bàn cờ thực tế thì *N=8)*;
- Gọi N con hậu lần lượt là: h_1, h_2,…, h_N.
- Để các con hậu không ăn nhau, ta sẽ đặt con hậu thứ *i* (h_i) vào hàng thứ *i*, với *i=1..N*.
- Ta sẽ phải chọn vị trí cột thích hợp của từng con hậu để chúng không ăn nhau.

Ví dụ: Bài toán 8 con hậu (tiếp)

- Giải thuật sẽ thực hiện trong N bước, mỗi bước sẽ tìm cách đặt một con hậu vào vị trí thích hợp.
  - Giả sử ở bước thứ *i*, ta đã xếp được *i* con hậu $h_1, h_2, \ldots, h_i$ vào *i* hàng đầu tiên thoả mãn yêu cầu đầu bài. Với $1 \le i \le N$. Nếu *i=N* thì kết thúc, đưa ra kết quả, trái lại thì sang bước tiếp theo.
  - Tại bước thứ *i+1*, tìm vị trí cột thích hợp ở hàng *i+1*để đưa h_{i+1} vào đó. Hai khả năng có thể xảy ra:
    - Nếu tìm được vị trí cột *j* thích hợp (không bị một trong *i* con hậu đầu tiên ăn) thì đưa h_{i+1} vào vị trí *j*. Lặp lại giải thuật ở bước *i+1*.
    - Nếu cả N cột đều không thích hợp thì phải quay lại thử tiếp ở bước thứ i (*quay lui*). Tức là phải tìm vị trí thích hợp tiếp theo của h_i (vì h_i đã đang ở một vị trí thích hợp). Nếu không có vị trí tiếp theo thích hợp và *i=1* thì giải thuật cũng kết thúc với kết luận không có giải pháp nào.

Giải thuật quay lui

- Cách cài đặt: có 2 cách
  - Dùng giải thuật đệ quy
  - Dùng vòng lặp (giải thuật không đệ quy)

Giải thuật quay lui – Cách cài đặt

- Dùng giải thuật đệ quy

```
void  QuayLui (vector V, int buoc)
{
   if (buoc == N) {
      V là một nghiệm;
   }
   else
      while (Tồn tại x_i thuộc D_i mà V+{x_i} thoả K)
      {
         D_i = D_i \ {x_i} ;
         V = V+ {x_i};
         QuayLui(V, buoc+1);
         V = V \ {x_i};   //Chuan bi Quay lui
      }
}
```

Giải thuật quay lui – Cách cài đặt

- Dùng giải thuật không đệ quy (vòng lặp)

```
int  QuayLui ()
{
    int  i = 1;
    int tong = 0;           //biến đếm tổng số giải pháp
    vector  V = ∅ ;         //V ban đầu rỗng
    do {
        while (Tồn tại x_i thuộc D_i mà V+{x_i} thoả K)
        {
            D_i = D_i \ {x_i} ;
            V = V+ {x_i};
            if  (i == N)  {
            V là một nghiệm;
            tong++;
        }
        else  i++;
        }
        V = V \ {x_i};
        i-- ;  //Quay lui
    }
    while  (i >= 1);
    return  tong;
}
```

Phương pháp tính độ phức tạp cho giải thuật đệ quy

- Công thức truy hồi:

  Là công thức có dạng:

  $$f_n = a_1 f_{n-1} + a_2 f_{n-2} + \ldots + a_k f_{n-k} \quad (1)$$

  Tức là giá trị thứ n của f phụ thuộc vào các giá trị của chính nó tại các điểm trước đó. Đồng thời, một số giá trị ban đầu của f cũng xác định trước. k được gọi là bậc của công thức. Nếu $a_1, a_2, \ldots, a_k$ là các hằng số thì công thức được gọi là tuyến tính thuần nhất.

- Ví dụ:
  - Công thức tính dãy Fibonaci:

    $$F_n = F_{n-1} + F_{n-2}; \quad F_0 = F_1 = 1;$$

  - Công thức tính số lần chuyển đĩa của bài toán Tháp Hà Nội:

    $$T_n = 2T_{n-1} + 1; \quad T_1 = 1;$$

Cách tính công thức truy hồi

- Công thức truy hồi (1) ở trên có phương trình đặc trưng:

$$r^k - a_1 r^{k-1} - a_2 r^{k-2} - \ldots - a_k = 0 \quad (2)$$

Nếu (2) có k nghiệm phân biệt $r_1, r_2, \ldots, r_k$, thì khi đó:

$$f_n = \alpha_1 . r_1^n + \alpha_2 . r_2^n + \ldots + \alpha_k . r_k^n \quad (3)$$

Trong đó:

$\alpha_1, \alpha_2, \ldots, \alpha_k$ là các hằng số mà tính được từ các giá trị ban đầu của f.

Cách tính công thức truy hồi

- Trong trường hợp k=2, và phương trình đặc trưng có nghiệm kép r_0, thì khi đó:

$$f_n = \alpha_1 . r_0^n + \alpha_2 . n . r_0^n$$

Bài tập

- Bài 1: Viết thủ tục đệ quy tính tổng của một dãy N số
- Bài 2: Viết thủ tục đệ quy tìm số nhỏ nhất trong một dãy N số.
- Bài 3: Viết thủ tục đệ quy thực hiện giải thuật sắp xếp chèn
- Bài 4: Viết thủ tục đệ quy thực hiện giải thuật sắp xếp nhanh.

Xin cảm ơn!