CHƯƠNG 14: ĐỆ QUY

14.1 Đệ quy là gì?
14.2 Đệ quy và lặp
14.3 Tháp Hà nội
14.4 Dãy số Fibonacci
14.5 Tìm kiếm nhị phân
14.6 Chuyển số nguyên sang dãy ký tự ASCII
14.7 Cấu trúc dữ liệu cây – cây nhị phân

ĐỆ QUY LÀ GÌ?

Ví dụ 18.1: Tính tổng $\sum_{1}^{n} i$

```
int RunningSum(int n)
{
   if (n == 1)
      return 1;
   else
      return n + RunningSum(n-1);
}
```

ĐỆ QUY LÀ GÌ?

Hàm này tính tổng tất cả các số nguyên từ 1 tới thông số nhập *n*. Thí dụ, **RunningSum(4)** sẽ tính 4+3+2+1, tuy nhiên nó là việc này một cách đệ quy. Có thể dễ dàng thấy rằng, tổng của 4 chính là 4 cộng với tổng của 3, tổng của 3 chính là 3 cộng với 2, và cứ thế. Định nghĩa *đệ quy* này chính là cơ sở cho giải thuật đệ quy tổng của *n* như sau:

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

ĐỆ QUY LÀ GÌ?

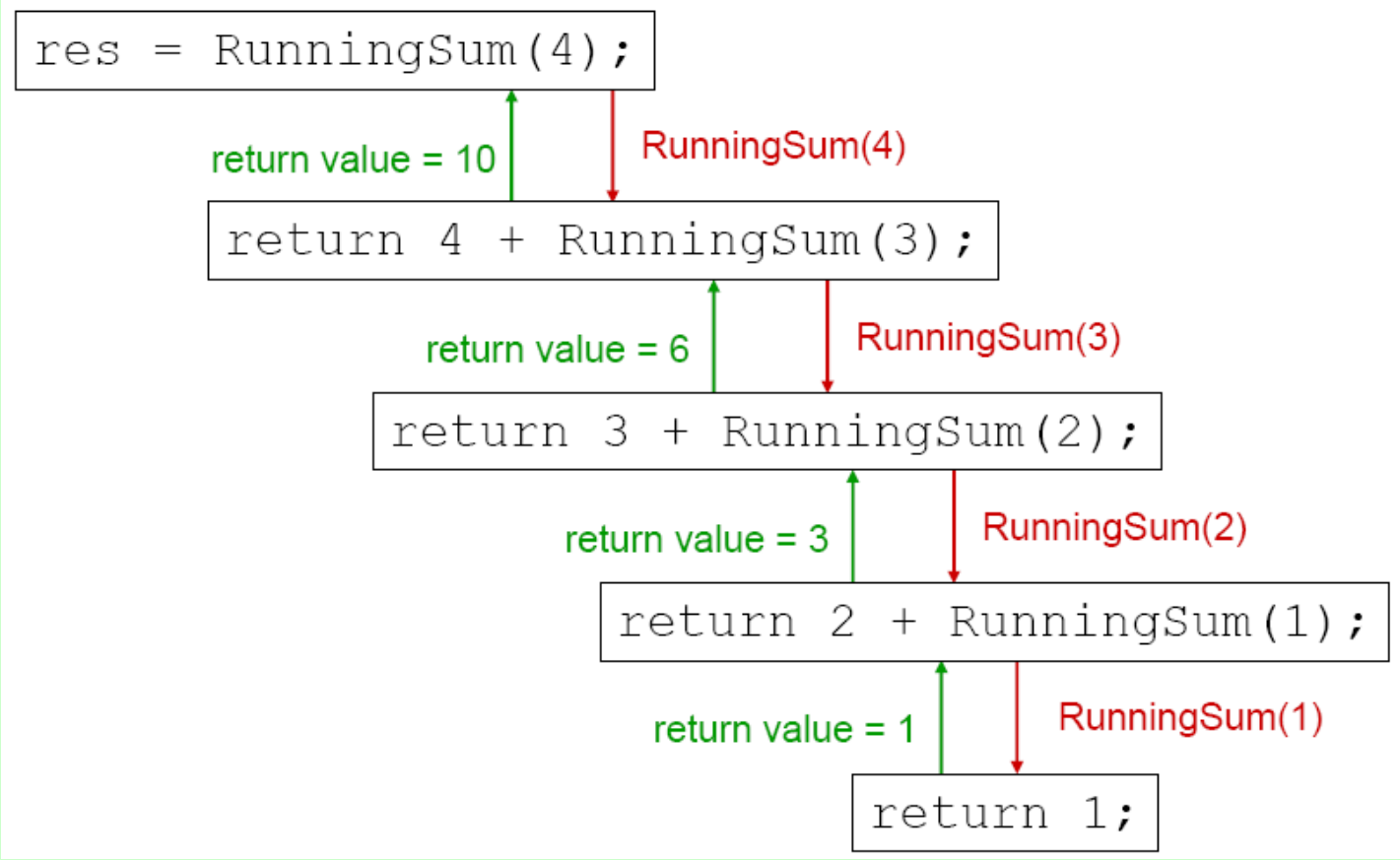
Về toán, chúng ta dùng ***các phương trình đệ quy*** để biểu diễn các hàm như vậy.
Chúng ta phải quy định *trường hợp gốc*:

RunningSum(1) = 1

Để tính **RunningSum(4)** chúng ta có quá trình sau:

RunningSum(4) = 4 + RunningSum(3)
= 4 + 3 + RunningSum(2)
= 4 + 3 + 2 + RunningSum(1)
= 4 + 3 + 2 + 1

ĐỆ QUY LÀ GÌ?

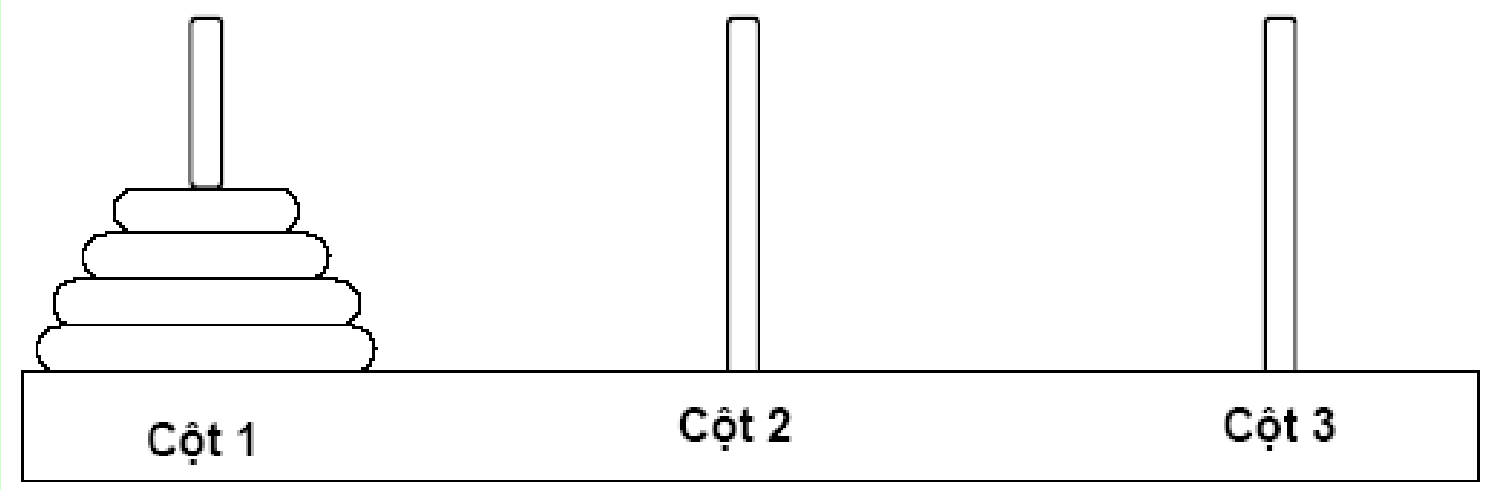

res = RunningSum(4);
return value = 10
RunningSum(4)
return 4 + RunningSum(3);
return value = 6
RunningSum(3)
return 3 + RunningSum(2);
return value = 3
RunningSum(2)
return 2 + RunningSum(1);
return value = 1
RunningSum(1)
return 1;

ĐỆ QUY VÀ LẶP

Tất cả các hàm đệ quy đều có thể được viết bằng vòng lặp.
Việc sử dụng đệ quy sẽ **dễ dàng và trong sáng hơn** khi dùng vòng lặp.
Bản đệ quy tương đối chậm vì các hàm đệ quy chịu sự gọi hàm còn vòng lặp thì không.

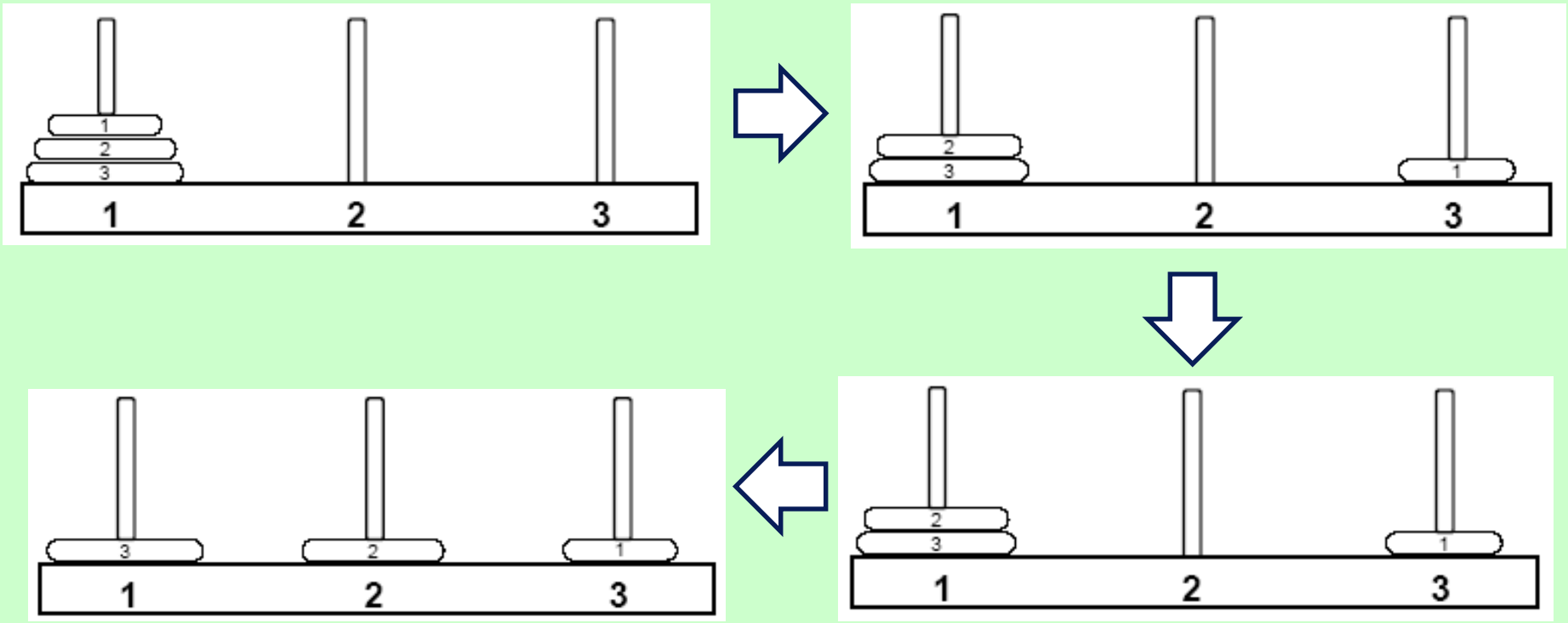
THÁP HÀ NỘI

Bài toán đố gồm một nền có ba cột, một trong ba cột có các đĩa gỗ sắp theo thứ tự đĩa nhỏ ở trên đĩa lớn ở dưới.
Chúng ta phải chuyển tất cả các đĩa từ cột hiện thời qua một trong hai cột kia theo hai luật sau: mỗi lần chỉ được di chuyển một đĩa và đĩa lớn không được đặt trên đĩa nhỏ.

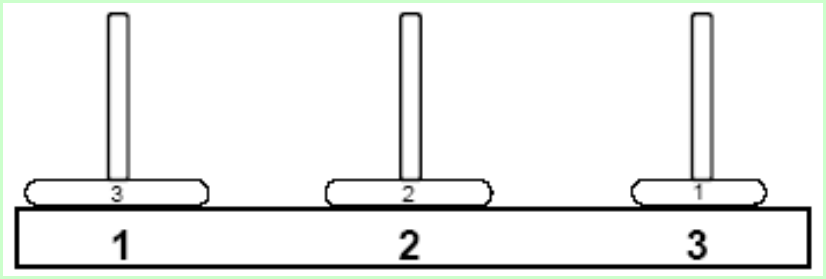


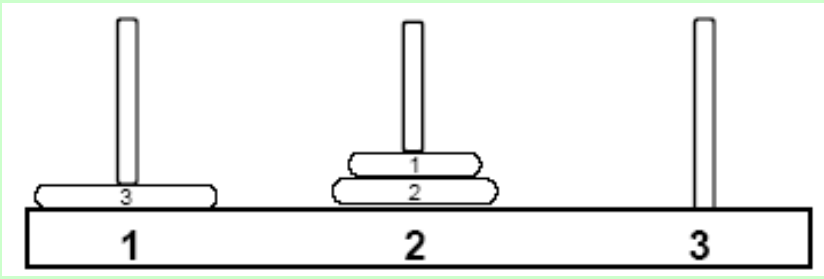
THÁP HÀ NỘI

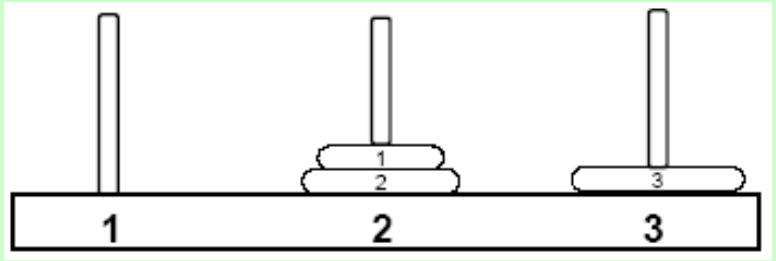
Với ý tưởng đó, ta xét bài toán có 3 đĩa thì quy trình như sau:

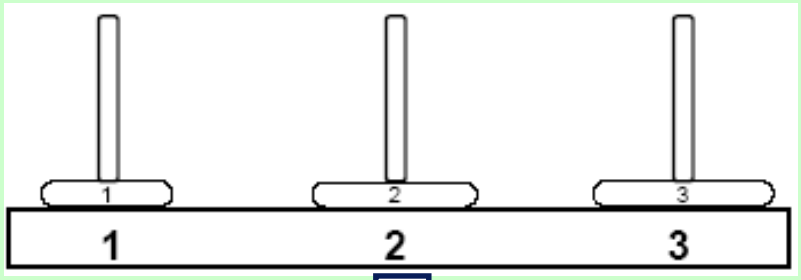


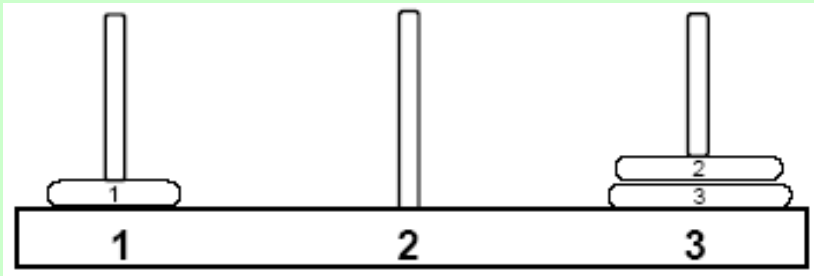
THÁP HÀ NỘI

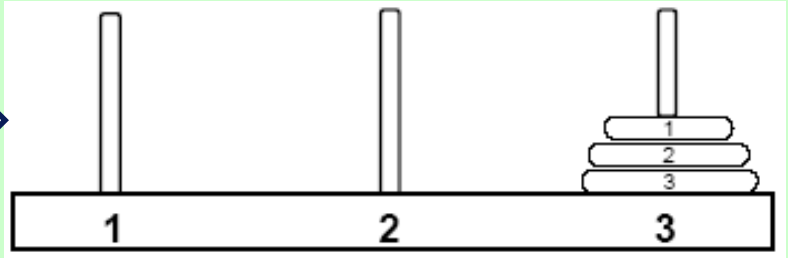












THÁP HÀ NỘI

Đoạn chương trình C của hàm Movedisk:

/* Dữ liệu nhập:

- ***diskNumber*** là số hiệu của đĩa cần chuyển chỗ (đĩa 1 là đĩa nhỏ nhất)
- ***startPost*** là cột mà hiện thời đĩa đang ở trên đó
- ***endPost*** là cột mà chúng ta muốn đĩa chuyển đĩa tới
- ***midPost*** là cột trung gian

*/

THÁP HÀ NỘI

Đoạn chương trình C của hàm Movedisk:

```
MoveDisk(diskNumber, startPost, endPost, midPost)
  {
     if (diskNumber > 1)
     {
        MoveDisk(diskNumber-1, startPost, midPost, endPost);
         printf("Move disk number %d from post %d to post
%d.\n", diskNumber, startPost, endPost);
MoveDisk(diskNumber-1, midPost, endPost, startPost);
      }
      else
          printf("Move disk number 1 from post %d to post %d.\n",
startPost, endPost);
   }
```

THÁP HÀ NỘI

Chúng ta tóm tắt lại các thao tác đệ quy:

```
MoveDisk(3, 1, 3, 2)   /*  Gọi khởi động  */
MoveDisk(2, 1, 2, 3)
MoveDisk(1, 1, 3, 2)
MoveDisk(1, 2, 3, 1)
MoveDisk(2, 2, 3, 1)
MoveDisk(1, 2, 1, 3)
MoveDisk(1, 1, 3, 2)
```

DÃY SỐ FIBONACCI

Ta có phương trình toán truy hồi sau

$$f(n) = f(n - 1) + f(n - 2)$$

$$f(1) = 1$$

$$f(0) = 1$$

hàm đệ quy để tính số Fibonacci thứ n là phương trình truy hồi trên.

Fibonacci(n) = Fibonacci(n-1) + Fibonacci(n-2).

Trường hợp gốc của đệ quy: ***Fibonacci(1)*** và ***Fibonacci(0)*** đều bằng 1.

DÃY SỐ FIBONACCI

Ví dụ 18.3: Chương trình tính số Fibonacci thứ n.

```
#include <stdio.h>
int Fibonacci(int n);
int main()
{   int in;
    int number;
     printf (“Which Fibonacci number? ”);
    scanf (“%d”, &in);
    number = Fibonacci(in);
    printf (“That Fibonacci number is %d\n”, number);        }
```

DÃY SỐ FIBONACCI

```
int Fibonacci(int n)
{
    if ((n == 0) || (n == 1))
        return 1;
    else
        return Fibonacci(n-1) + Fibonacci(n-2);
}
```

DÃY SỐ FIBONACCI

Xem quá trình đệ quy khi gọi hàm Fibonacci(3) một cách đại số như sau:

Fibonacci(3) = Fibonacci(2) + Fibonacci(1)
= Fibonacci(1) + Fibonacci(0) + Fibonacci(1)
= 1 + 1 + 1 = 3

TÌM KIẾM NHỊ PHÂN

Kỹ thuật tìm kiếm nhị phân (binary search) là cách rất nhanh chóng để tìm ra phần tử trong danh sách với khóa tham khảo đã được sắp xếp theo một thứ tự từ nhỏ đến lớn (hay ngược lại).

TÌM KIẾM NHỊ PHÂN

Ví dụ 18.4:

```
int BinarySearch(int item, int list[], int start, int end)
{
 int middle = (end + start) / 2;
if (end < start)
    return -1;
else if (list[middle] == item)
    return middle;
else if (item < list[middle])
    return BinarySearch(item, list, start, middle - 1);
return BinarySearch(item, list, middle + 1, end);
}
```

Chuyển số nguyên sang dãy ký tự ASCII

Để biểu diễn một trị nguyên lên màn hình, mỗi ký số của trị phải được trích ra một cách riêng lẻ, được chuyển sang mã ASCII và rồi được đưa tới thiết bị xuất liệu.

Hàm đệ quy ***IntToAscii*** làm việc như sau: để in ra được một số, ví dụ 21669, hàm sẽ chia nhỏ vấn đề ra làm hai phần. Đầu tiên số 2166 phải được in ra nhờ gọi đệ quy tới hàm ***IntToAscii***, và khi gọi đã xong, số 9 sẽ được in ra.

Chuyển số nguyên sang dãy ký tự ASCII

Ví dụ 18.5:

```
#include <stdio.h>
void IntToAscii(int i);
int main()
{
  int in;
  printf("Input number: ");
  scanf("%d", &in);
  IntToAscii(in);
  printf("\n");
}
```

Chuyển số nguyên sang dãy ký tự ASCII

```
void IntToAscii(int num)
{
  int prefix;
  int currDigit;
   if (num < 10)              /* The terminal case     */
    printf("%c", num + '0');
  else {
    prefix = num / 10;        /* Convert the number    */
    IntToAscii(prefix);       /* without last digit    */
     currDigit = num % 10;    /* Then print last digit */
    printf("%c", currDigit + '0');
  }
}
```

Cấu trúc dữ liệu cây – cây nhị phân

Xem sách giáo khoa

KẾT THÚC CHƯƠNG 14