EC1003 程式設計

Spring 2018

Homework 3

Due: 23:55PM, May. 10

1. (Ch. 5, Prob. 5.36) ***(Towers of Hanoi)*** Every budding computer scientist must grapple with certain classic problems, and the Towers of Hanoi (see Fig. 5.19) is one of the most famous of these. Legend has it that in a temple in the Far East, priests are attempting to move a stack of disks from one peg to another. The initial stack had 64 disks threaded onto one peg and arranged from bottom to top by decreasing size. The priests are attempting to move the stack from this peg to a second peg under the constraints that exactly one disk is moved at a time, and at no time may a larger disk be placed above a smaller disk. A third peg is available for temporarily holding the disks. Supposedly the world will end when the priests complete their task, so there is little incentive for us to facilitate their efforts.

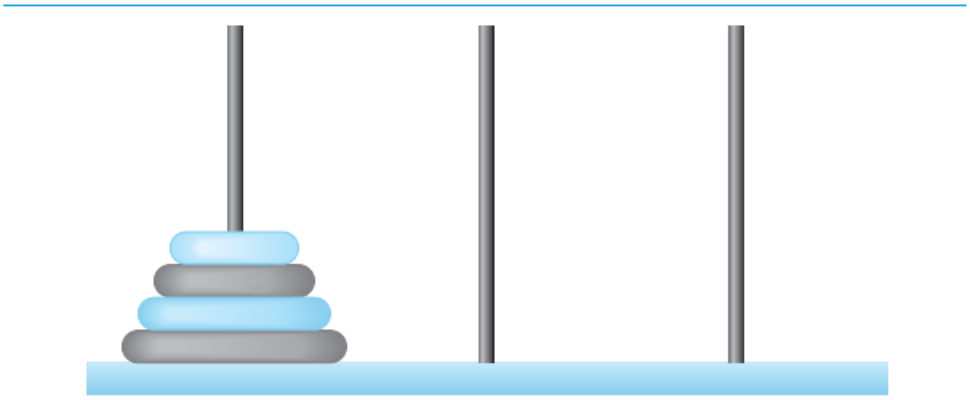

Fig. 5.19 | Towers of Hanoi for the case with four disks.

Let us assume that the priests are attempting to move the disks from peg 1 to peg 3. We wish to develop an algorithm that will print the precise sequence of disk-to-disk peg transfers. If we were to approach this problem with conventional methods, we would rapidly find ourselves hopelessly knotted up in managing the disks. Instead, if we attack the problem with recursion in mind, it immediately becomes tractable. Moving *n* disks can be viewed in terms of moving only *n* – 1 disks (and hence the recursion) as follows:

a) Move *n* – 1 disks from peg 1 to peg 2, using peg 3 as a temporary holding area.

b) Move the last disk (the largest) from peg 1 to peg 3.

c) Move the *n* – 1 disks from peg 2 to peg 3, using peg 1 as a temporary holding area.

The process ends when the last task involves moving *n* = 1 disk, i.e., the base case. This is accomplished by trivially moving the disk without the need for a temporary holding area.

Write a program to solve the Towers of Hanoi problem. Use a recursive function with four parameters:

a) The number of disks to be moved

b) The peg on which these disks are initially threaded

c) The peg to which this stack of disks is to be moved

d) The peg to be used as a temporary holding area

Your program should print the precise instructions it will take to move the disks from the starting peg to the destination peg. For example, to move a stack of three disks from peg 1 to peg 3, your program should print the following series of moves:

1 –> 3 (This means move one disk from peg 1 to peg 3.)

1 –> 2

3 –> 2

1 –> 3

2 –> 1

2 –> 3

1 –> 3

Please show your results for moving six disks from peg 1 to peg 3.

2. (Ch 5. Prob. 5.32) ***(Guess the Number)*** Write a C program that plays the game of "guess the number" as follows: Your program chooses the number to be guessed by selecting an integer at random in the range 1 to 1000. The program then types:

I have a number between 1 and 1000 Can you guess my number? Please type your first guess.

The player the types a first guess. The program responds with one of the following:

1. Excellent! You guessed the number. Would you like to play again (y or n)? 2. Too low! Try again. 3. Too high! Try again.

If the player's guess is incorrect, your program should loop until the player finally gets the number. Your program should keep telling the player Too high or Too low to help the player "zero in" on the correct answer.

3. (Ch 5, Prob. 5.29) ***(Least Common Multiple)*** The *lease common multiple (LCM)* of two integers is the smallest positive integer that is a multiple of both the numbers. Write function lcm that returns the least common multiple of two integers. Write a program to test your lcm function for the following integer pairs: (23, 483), (61, 17), and (19, 82).

4. (Ch. 6, Prob. 6.30) ***(The Sieve of Eratosthenes)*** A prime integer is any integer greater than 1 that can be divided evenly only by itself and 1. The Sieve of Eratosthenes is a method of finding prime numbers. It works as follows:

   a) Create an array with all elements initialized to 1 (true). Array elements with prime subscripts will remain 1. All other array elements will eventually be set to zero.

b) Starting with array subscript 2 (subscript 1 is not prime), every time an array element is found whose value is 1, loop through the remainder of the array and set to zero every element whose subscript is a multiple of the subscript for the element with value 1. For array subscript 2, all elements beyond 2 in the array that are multiples of 2 will be set to zero (subscripts 4, 6, 8, 10, and so on.). For array subscript 3, all elements beyond 3 in the array that are multiples of 3 will be set to zero (subscripts 6, 9, 12, 15, and so on.).

When this process is complete, the array elements that are still set to 1 indicate that the subscript is a prime number. Write a program that uses an array of 1000 elements to determine and print the prime numbers between 1 and 999. Ignore element 0 of the array.

Hand in:

1. Source code (You must write comments to explain your codes.)
2. Running results