FSE598 前沿计算技术

模块 2 数据与数据处理
单元 3 编程技术
第 2 讲 递归函数

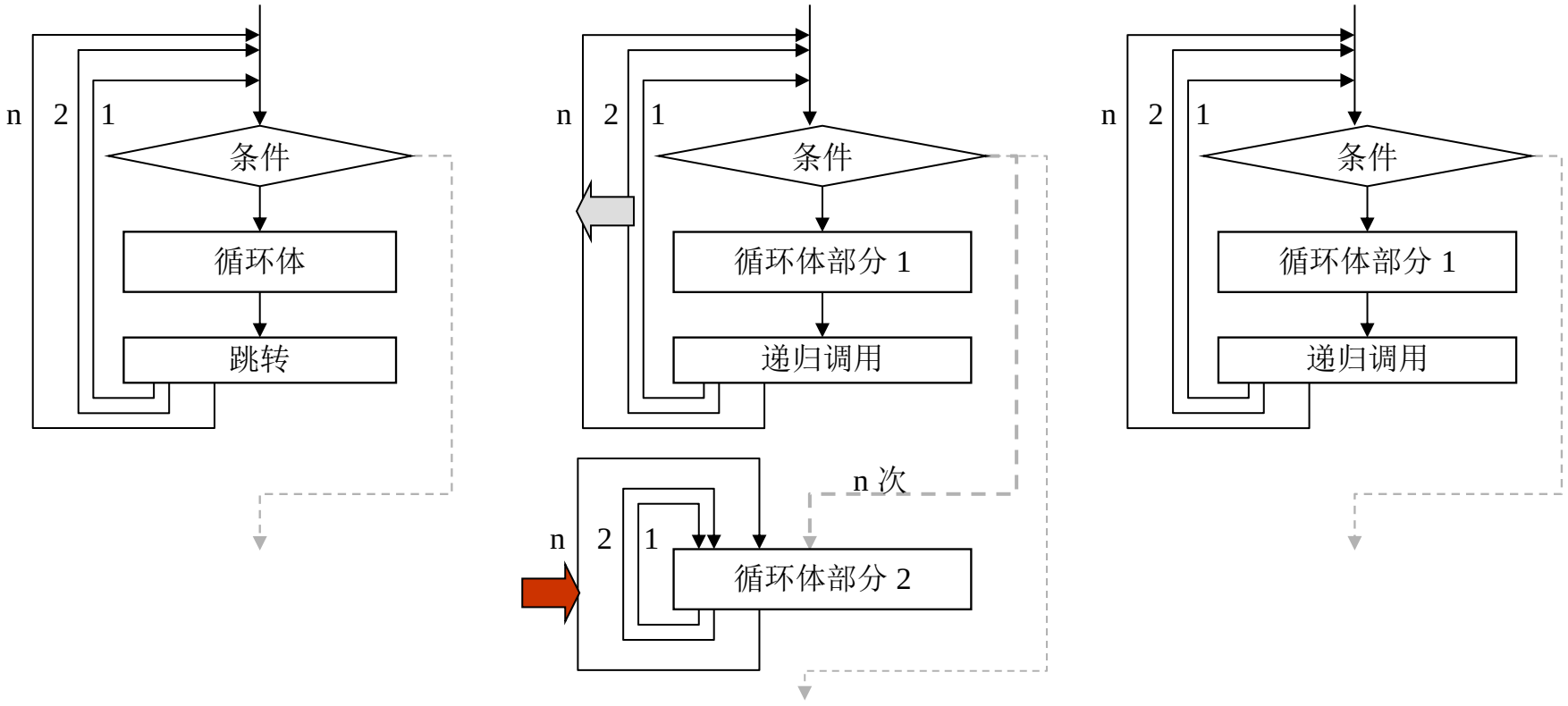
讲座的英文版内容基于本书：
Y. Chen 《编程语言入门：C、C++、Scheme、Prolog、C# 和 Python 编程》(Introduction to Programming Languages: Programming in C, C++, Scheme, Prolog, C#, and Python)，第 6 版，Kendall Hunt Publishing Company，2019 年。
https://www.public.asu.edu/~ychen10/book/IntroPl.html

本讲大纲

学习

- 循环和递归的区别
- 编写递归程序的步骤
- 假设 size-m 问题已为您解决，您不需要解决它
- 理解递归程序的例子

循环和递归的一般结构

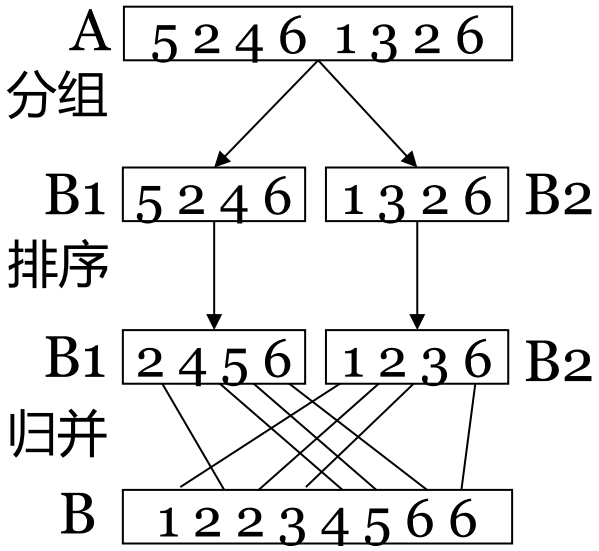


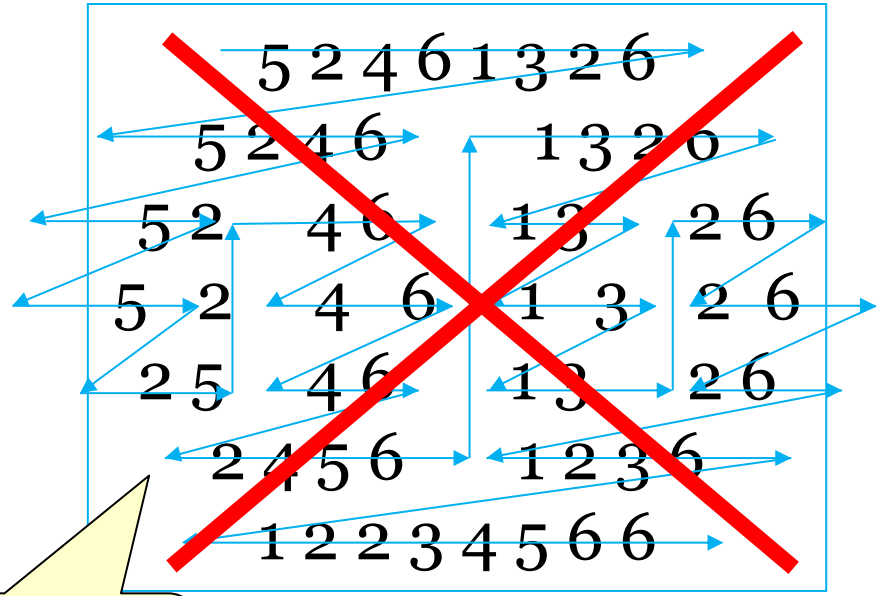
(a) (b) (c)

while-循环 递归 尾-递归

归并排序及其跟踪

1. 如数组只包含一个元素，则终止；
2. 将该数组分为两个子数组；
3. 做两个递归调用；
4. 归并两个已排序的子数组。

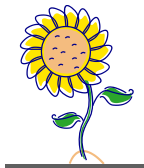




避免使用该方法来理解递归

比喻

- 孩子：我不知道怎么拿到花。太高了我够不着。
- 妈妈：让我们先跨上第一个台阶。孩子照做了，很兴奋。
- 孩子正打算跨上下一个台阶，但是
- 妈妈说：“我知道你可以跨上第二个台阶，我不想让你感到无聊和疲倦”。



- 妈妈带着她来到第 n-1 个台阶；
- 妈妈让孩子自己决定怎么做。
- 孩子从第 n-1 个台阶移动到第 n 个台阶，最后拿到了花。

用“神奇四步” 抽象方法来编写递归程序

1. 定义 size-n 问题。
2. 找到终止条件和对应的返回值。
3. 定义 size-m (m<n) 问题并找到 m。
   在很多情况下，m = n-1；
4. 通过 size-m 问题的解，构建 size-n 问题的解。

资料来源：Yinong Chen 编程语言简介：C、C++、Scheme、Prolog、C# 和 Python 编程，第 6 版，Kendall Hunt Publishing，2019

步骤 1：定义 size-n 问题。

- 与循环一样，只有当我们想要解决一个需要多次重复相同操作的问题时，才有必要采用递归算法。
- 假设重复的次数为 n。在大多数情况下，n 是已知的。例如，n!，已给定 n 的大小。
- 在很多情况下，定义 size-n 问题，只需要选择一个函数名，并把 n 作为该函数的参数。因此，阶乘（factorial）的 size-n 问题就是

  int factorial (int n)
- size-n 的返回值是该函数要计算的解，或我们要寻找的值。

我们不需要为 size-n 问题设计解。

步骤 2： 找到终止条件和对应的返回值。

- 与 while-循环一样，递归函数会先检查终止条件。
- 如终止条件为 true，则返回对应的值并退出函数。
- 反之，则须进入递归函数体。
- 在大多数情况下，确定终止条件和对应的值都很简单。例如，factorial(n) 的终止条件是 n = 0 且对应的值为 1

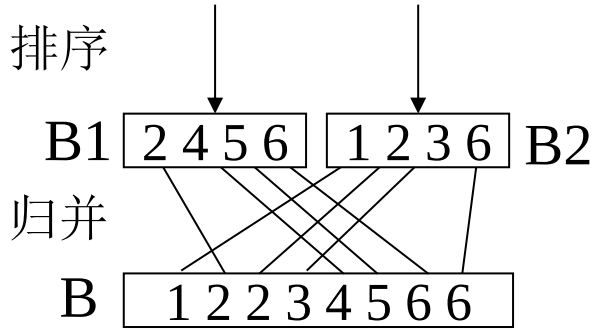
步骤 3： 定义 size-m 问题，m < n

- size-m 问题就是把 n 换成 m 的 size-n 问题，其中 m < n
  取决于我们在一次循环中可以把大小（size）减少多少。
- 如果我们只能将问题大小减少 1 个数，则 m 就是 n-1。
  例如： factorial(n-1)
- 在某些情况下，我们需要找到一个不是 n-1 的 m。如何找到适当的 m 是与具体应用有关的。
  对于归并排序，m = ½ n。
- 在此步骤中，不要试图找到 size-m 问题的解或返回值！
  我们要做的只是：**假设** size-m 问题将返回一个值，我们可以使用这个解，例如 n***factorial (n-1);**

步骤 4: 通过假设的 size-m 问题的解，构建 size-n 问题的解

- 在这个步骤中，我们将用 size-m 或 size-(n-1) 问题的假定解或返回值来构建 size-n 问题的解。
- 此步骤因具体应用而异。对于阶乘问题，size-n 问题的解为 **n*factorial (n-1)**;
- 有时，我们要用到多个 size-m 问题的返回值，其中用 0≤m<n 来构建 size-n 问题的解。

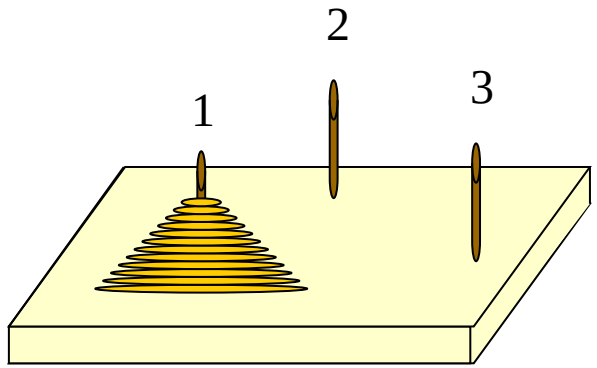
针对归并排序的情况：



Factorial(n)

```
def factorial(n):    # size-n problem
   if (n == 0):      # Stopping condition
      return 1;      # Return value at the stopping condition
   else:
      return n * factorial(n - 1); # use size-(n-1) problem's
   # assumed solution to construct size-n problem's solution
def main():
     print(F"factorial(4) = {factorial(4)}")
   # output: factorial(4) = 24
if __name__ == "__main__":
  main()
```

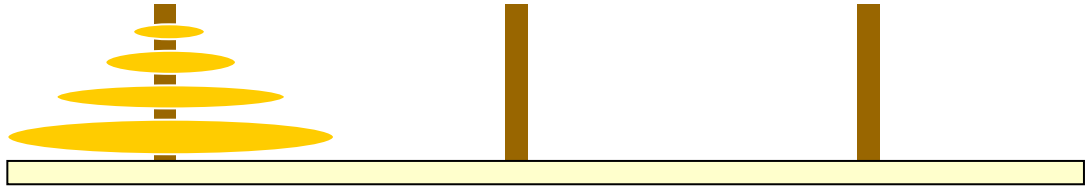
汉诺塔游戏



- 有史以来最棒的益智游戏之一；
- François Anatole Lucas
  于1883年发明了该游戏；
- 想法源自《De Subtiliate》
  1550年 Girolamo Cardano 的著作
- Hanoi （河内, 又译为“汉诺”）的一座修道院有一块立着三根木桩的金板。最初，第一根桩子上有64块金圆盘。僧侣们接到命令，将圆盘从第一根木桩移到第三根木桩上，用第二个木桩作为缓存。
- 僧侣们每次只可移动一块圆盘，且不得把大圆盘放在小圆盘上面……

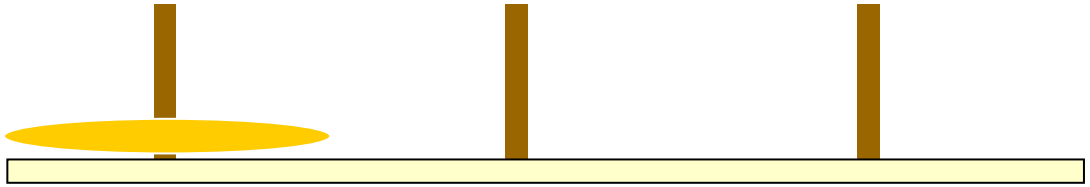
汉诺塔：根据四步法设计步骤

步骤 1
size-n 问题



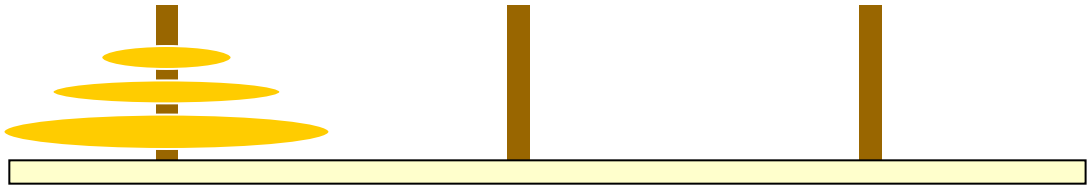
void Hanoi(**n**, source, center, destination)

步骤 2
终止条件
（size-1 问题）
和解法：

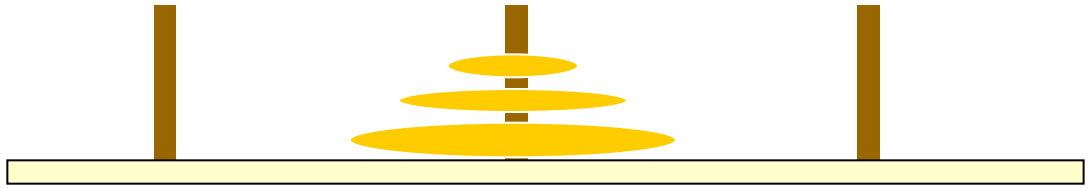


“把圆盘从source移动到destination”

步骤 3：
size-(n-1) 问题
有两种情况：



hanoi (n-1, source, center, destination)

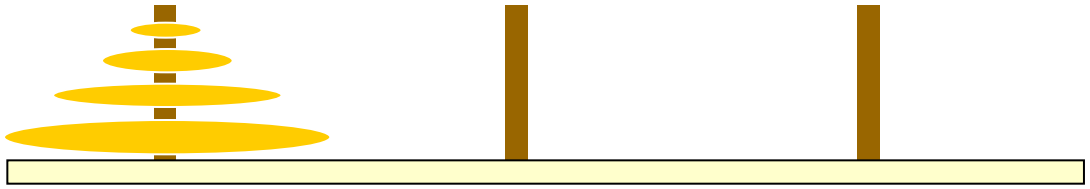


hanoi (n-1, center, source, destination）

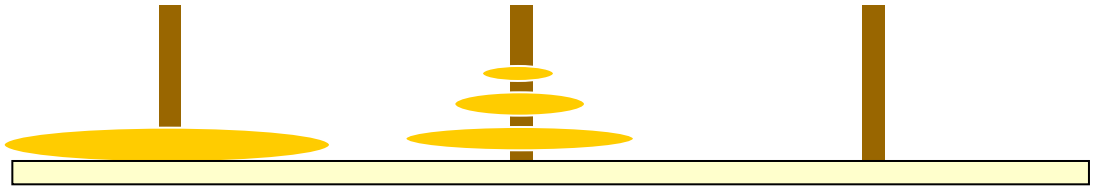
汉诺塔示例：

步骤 4：通过 size-m 问题，构建 size-n 问题的解法。

(1):
把 n-1 个圆盘
移动到中间的桩子上



(2):
把 1 个圆盘
移动到右侧的桩子上



(3):
把 n-1 个圆盘
移动到右侧的桩子上

汉诺塔程序

```
def Hanoi(n , left, center, right):
      if n==1:
         print ("Move disk 1 from Left",left,"to destination",Right)
         return
      Hanoi(n-1, Left, Right, Center)
      print ("Move disk",n,"from source", Left,"to destination", Right)
      Hanoi(n-1, Center, Left, Right)
```

汉诺塔程序的输出

hanoi(3) 函数输出:

```
move top from Left to Right
move top from Left to Center
move top from Right to Center
move top from Left to Right
move top from Center to Left
move top from Center to Right
move top from Left to Right
```

```
For  n = 3
move top from Left to Right
move top from Left to Center
move top from Right to Center
move top from Left to Right
move top from Center to Left
move top from Center to Right
move top from Left to Right
For  n = 4
move top from Source to Spare
move top from Source to Destination
move top from Spare to Destination
move top from Source to Spare
move top from Destination to Source
move top from Destination to Spare
move top from Source to Spare
move top from Source to Destination
move top from Spare to Destination
move top from Spare to Source
move top from Destination to Source
move top from Spare to Destination
move top from Source to Spare
move top from Source to Destination
move top from Spare to Destination
```

斐波那契函数

- 类似于阶乘函数，我们可以实现斐波那契函数：

$$fib(N) = \begin{cases} 0 & if\ N = 0 \\ 1, & if\ N = 1 \\ fib(N-1) + fib(N-2), & if N \geq 2 \end{cases}$$

斐波那契函数迭代实现

```
n = int(input("Please enter a positive integer n = "))
n1, n2 = 0, 1 # base cases (stopping conditions)
count = 0
while n <= 0: # check if the number is valid
   n = int(input("Please enter a positive integer that is 1 or
bigger "))
if n == 1:     # if the input number is 1, return n1
   print("Fibonacci sequence from 0 to ",n,":", n1)
else:          # generate fibonacci sequence in a while loop
   print("Fibonacci sequence starting from 0:")
   while count < n:
       print(n1)
       nth = n1 + n2
       n1 = n2    # update values to the next iteration
       n2 = nth
       count += 1 # update the counter
```

斐波那契函数递归实现

$$fib(N) = \begin{cases} 0 & if\ N = 0 \\ 1, & if\ N = 1 \\ fib(N-1) + fib(N-2), & if N \geq 2 \end{cases}$$

```
def fib(n:int):
    if n <= 1:
        return n
    else:
        return(fib(n-1) + fib(n-2))
        print("Fibonacci sequence:")
def main():
    n = int(input("Please enter a positive integer n = "))
    while n <= 0:
        n = int(input("Please enter a positive integer"))
        # if the number is 1, return n1
    for i in range(1,n):
        print(fib(i))
main()
```