FSE598 前沿计算技术

模块 3　算法设计与分析
单元 1　算法基础
第 3 讲　递归式

本课程的部分内容是基于 Thomas H.Cormen、Charles E.Leiserson 等人的“算法简介”教材

课程概要

学习内容

- ❑ 循环程序的运行时间
- ❑ 递归式的定义
- ❑ 从递归程序导出递归式
- ❑ 求解递归式的运行时间

时间复杂度：循环的运算次数

带有循环的程序/算法

```
x ← a + b                 ------------------------------>  O(1)
while i < n do            ------------------------------>  O(1)
    y ← c – d             ------------------------------>  O(n)
    i ← i+1               ------------------------------>  O(n)
end
for  i ← 1 to n do        ------------------------------>  O(1)
    for j ← 1 to n do     ------------------------------>  O(n)
        x ← a + b         ------------------------------>  O(n²)
        y ← c - d         ------------------------------>  O(n²)      +
    end
end
```

$T(n) = 2*O(n^2) + 3*O(n) + 3*O(1) = O(n^2)$

递归式的定义

递归式是一个方程式，它用一个公式来定义一个序列，该公式将下一项作为前一项的函数。也就是用 size-m 问题的执行时间来描述 size-n 问题的执行时间。它包括两个部分：

- 基础情况：$n = c$ （e.g.， $c = 0$ or $c = 1$） 所需的时间；
- 递归情况：用于解一个或多个 size-m 问题所需的时间

$$T(n)=\begin{cases} g(n) & if\ n=c \\ a_1T(f_1(n))+\ldots+a_kT(f_k(n))+h(n) & otherwise \end{cases}$$

递归式的示例是

$$T(n)=\begin{cases} 1 & if\ n=1 \\ 2T(n/2)+n & otherwise \end{cases}$$

时间复杂度：带有简单递归式的操作（步数）计数

带有简单递归式的程序/算法

```
Foo(n)                  ------------------>  T(n)
    if n = 0
        print n         ------------------>  1 only once at Foo(0)
        return          ------------------>  1 only once at Foo(0)
    else
        print n
        Foo(n-1)        ------------------>  T(n-1)   +
```

尾部递归 = while 循环

递归式

$$T(n) = \begin{cases} 2 & n = 0 \\ T(n-1) + 1 & n > 0 \end{cases}$$

定义：**递归式**是一个方程，它用一个规则来定义一个序列，该规则将下一项作为前一项的函数。

解递归式

$$
\begin{aligned}
T(n) &= 1 + T(n-1) \\
&= 1 + 1 + T(n-2) \\
&= 1 + 1 + 1 + T(n-3) \\
&\;\;\ldots \\
&= 1 + 1 + \ldots + 1 + T(0) \\
&= \underbrace{1 + 1 + \ldots + 1}_{n} + 2 \\
&= n + 2
\end{aligned}
$$

该程序用一个递归调用将 size-n 问题缩减为 size-（n-1）问题。

复杂度是：O(n)

$$T(n) = n + 2 = O(n)$$

递归和循环的时间复杂度

带有简单递归式的程序/算法

Foo(n)	T(n)
if n = 0	1 in every iteration
print i	1 only once at Foo(0)
return	1 only once at Foo(0)
else	
for i ← 1 to n do	
sum ← sum + i	n in every recursive call
print sum	1 in every recursive call
Foo(n-1)	+ T(n-1)

递归式

$$T(n) = \begin{cases} 2 & n = 0 \\ T(n-1) + n + 2 & n > 0 \end{cases}$$

解递归式

T(n) = n + 2 + T(n-1)

= n + 2 + (n-1) + 2 + T(n-2)

= n + 2 + (n-1) + 2 + (n-2) + 2 + T(n-3)

. . .

$$= \underbrace{n + (n-1) + (n-2) + (n-3) + \dots + 1}_{n} + 2*n + T(0)$$

该程序用一个递归调用将 size-n 问题缩减为 size-（n-1）问题 + n 次运算。
复杂度是：O(n²)

$$T(n) = \left(\sum_{i-1}^{n} i\right) + 2*n + 2 = \frac{n*(n-1)}{2} + 2*n + 2 = O(n^2)$$

汉诺塔程序:移动次数的时间复杂度

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

$$
\begin{aligned}
T(n) &= 2^*T(n-1) + 1 \\
&= 2^*(2^*T(n-2) + 1) + 1 \\
&= 2^*(2^*(2^*T(n-3) + 1) + 1) + 1 = 2^3{}^*T(n-3) + 2^2 + 2^1 + 1 \\
&\ldots \\
&= 2^{n-1}{}^*T(1) + 2^{n-2} + \ldots + 2^2 + 2^1 + 1 \\
&= 2^{n-1} + 2^{n-2} + \ldots + 2^2 + 2^1 + 1 \\
&= 2^n - 1 \\
&= O(2^n)
\end{aligned}
$$

该程序用两个递归调用将 size-n 问题缩减为 size-(n-1) 问题。复杂度是：$O(2^n)$

归并排序的递归式和复杂度

我们如何找到递归（带有递归调用）算法的执行时间？

```
mergesort (A[n], L,R)      —— 假设 ——>  T(n)
    If R = L
    Then return            ——>  1
    Else
        M := ⌊(R+L)/2⌋
        mergesort (A, L,M)     ——>  T(n/2)
        mergesort (A, M+1,R)   ——>  T(n/2)
        merge(A, L, M, R)      ——>  n
```

该程序用两个递归调用将 size-n 问题缩减为 size-(n/2) 问题 + n。
复杂度是： O($n$lg$n$)

$$T(n)=\begin{cases}1 & if\ n=1\\ 2T(n/2)+n & otherwise\end{cases}$$

⇒ T(n) = O($n$lg$n$)

CSE551 中有推导和证明过程