
# 第6周小结

## 1 递归基础

❶ 一个递归模型由哪两部分构成？

- **递归出口**—确定递归结束情况
- **递归体**—确定大小问题的求解情况

## ❷ 递归算法如何转换为非递归算法？

- 对于尾递归，可以用循环递推方法来转换。
- 对于其他递归，可以用栈模拟执行过程来转换。

## ❸ 在Hanoi问题的递归算法中，当移动6个盘片时递归次数是多少？

$m(n) = 1$ 当 $n=1$

$m(n) = 2m(n-1)+1$ 当 $n>1$

$$
\begin{aligned}
t(6) &= 2t(5) + 1 \\
&= 2^2t(4) + 1 + 2 \\
&= 2^3t(3) + 1 + 2 + 2^2 \\
&= 2^4t(2) + 1 + 2 + 2^2 + 2^3 \\
&= 2^5t(1) + 1 + 2 + 2^2 + 2^3 + 2^4 \\
&= 1 + 2 + 2^2 + 2^3 + 2^4 + 2^5 = 2^6 - 1 = 63
\end{aligned}
$$

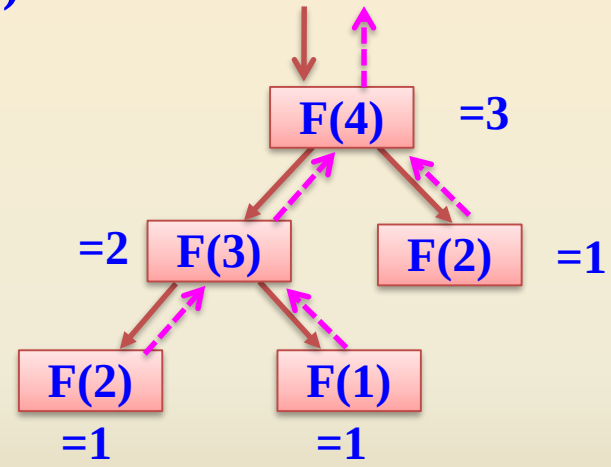
## ❹ 分析递归求Fibonacci数列时，栈的变化情况？

$F(1)=1$

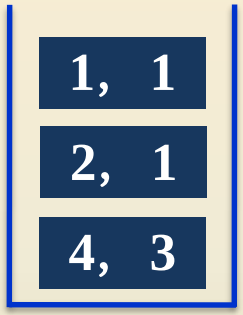
$F(2)=1$

$F(n)=F(n-1)+F(n-2)\quad n>2$

求F(4) = ？




栈




参数，函数值

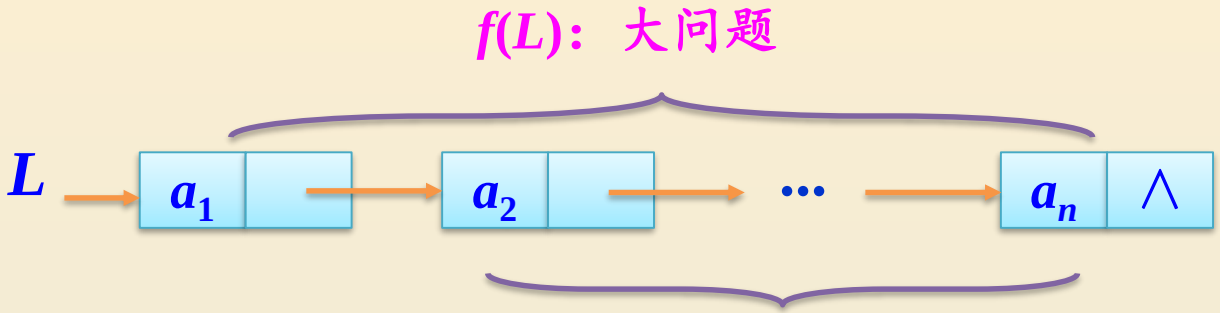
求出F(4)=3

# 2 递归算法设计

## ❶ 基于递归数据结构的递归算法设计

- 利用递归数据结构的递归特性建立递归模型
- 编写对应的递归算法

设计递归算法销毁一个不带头节点的单链表。




| | |
|---|---|
| $f(L)$ ⇔ 不做任何事件 | 当$L$为空 |
| $f(L)$ ⇔ $f(L\text{->next})$; free(L); | 当$L$非空 |

算法如下：

```
void release(LinkList *&L)
{
        if (L!=NULL)
       {      release(L->next);
              free(L);
       }
}
```

## ❷ 基于递归方法的递归算法设计

如何将递归特性不明显的问题转化为递归问题求解

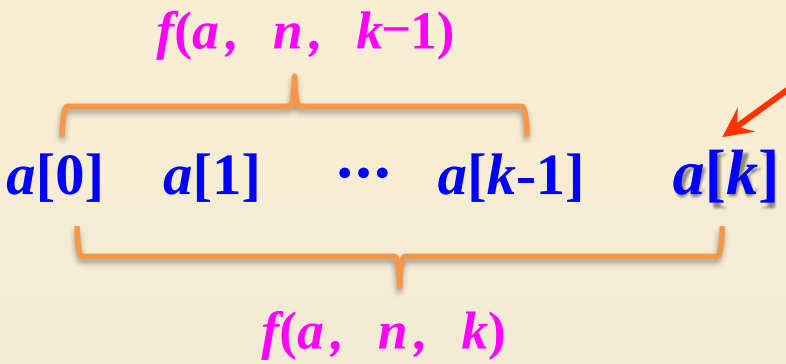
- 确定问题的形式化描述
- 确定哪些是大问题，哪些是小问题
- 确定大、小问题的关系
- 确定特殊（递归结束）情况

假设$a$数组含有1，2，…，$n$，求其全排列。

**解：** 设$f(a，n，k)$为$a[0..k]$（$k$+1个元素）的所有元素的全排序，为大问题。

则$f(a，n，k\text{-}1)$为$a[0..k\text{-}1]$（$k$个元素）的所有元素的全排序，为小问题。

假设$f(a，n，k-1)$可求，对于$a[k]$位置，可以取$a[0..k]$任何元素值，再组合$f(a，n，k-1)$，则得到$f(a，n，k)$。




此位置可以取$a[0]$～$a[k]$中任何值，但不重复！

采用循环$i$：0～$k$，$a[i]$⇔$a[k]$

$f(a，n，k)$ ⇔ 输出$a$　　当$k$=0时(一个元素的全排列)

$f(a，n，k)$ ⇔ $a[k]$位置取$a[0..k]$任何之值，并组合$f(a，n，k-1)$的结果;　　其他情况

```
void perm(int a[], int n, int k)
{    int i, j;
     if (k==0)
     {   for (j=0;j<n;j++)
             printf("%d", a[j]);
         printf("\n");
     }
    else
    {   for (i=0;i<=k;i++)
        {   swap(a[k], a[i]);
            perm(a, n, k-1);
            swap(a[k], a[i]);
        }
    }
}
```

```
void main()
{   int n=3, k=2;
    int a[]={1,2,3};
    perm(a, n, k);
}
```

输出结果：

231

321

312

132

213

123

# 3 递归函数设计中几个问题

## ❶ 递归函数中的引用形参可以用全局变量代替

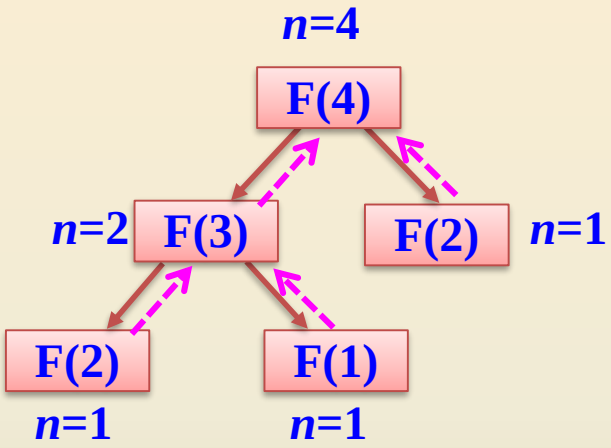
例如，求1+2+…+*n*

```
void add(int n，int &s)    //s=1+2+…+n
{    int s1;
     if (n==1)
          s=1;
     else
     {    add(n-1，s1);
          s=s1+n;
     }
}
```

可以用全局变量代替：

```
int s=0;                    //全局变量
void add1(int n)    //理解为:s与add(n)绑定，  s=1+2+…+n
{     if (n==1)
            s=1;
      else
      {     add1(n-1);
            s+=n;
      }
}
```

## ❷ 递归函数中的非引用形参作为状态变量，可以自动回溯



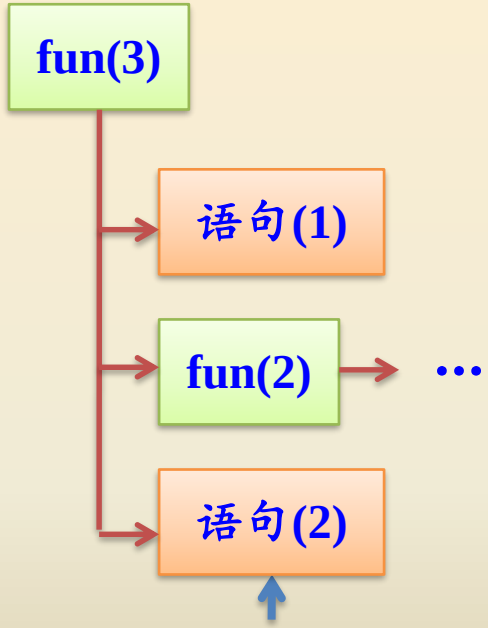

- n表示状态
- 状态自动回溯

## ❸ 递归调用后面的语句表示该子问题执行完毕后要完成的功能

```
void fun(int n)
{
    if (n>=1)
    {
        printf("n1=%d\n",n);  //(1)
        fun(n-1);
        printf("n2=%d\n",n);  //(2)
    }
}
```

fun(3)的输出结果

```
n1=3
n1=2
n1=1
n2=1
n2=2
n2=3
```




掌握递归函数的执行过程有助于递归算法设计