5.2 递归算法的设计

5.2.1 递归算法设计的步骤

- 设计求解问题的递归模型。
- 转换成对应的递归算法。

求递归模型的步骤

（1）对原问题$f(s)$进行分析，称为“大问题”，假设出合理的“小问题”$f(s')$；

（2）假设$f(s')$是可解的，在此基础上确定$f(s)$的解，即给出$f(s)$与$f(s')$之间的关系 ⇨ 递归体。

（3）确定一个特定情况（如$f(1)$或$f(0)$）的解 ⇨ 递归出口。

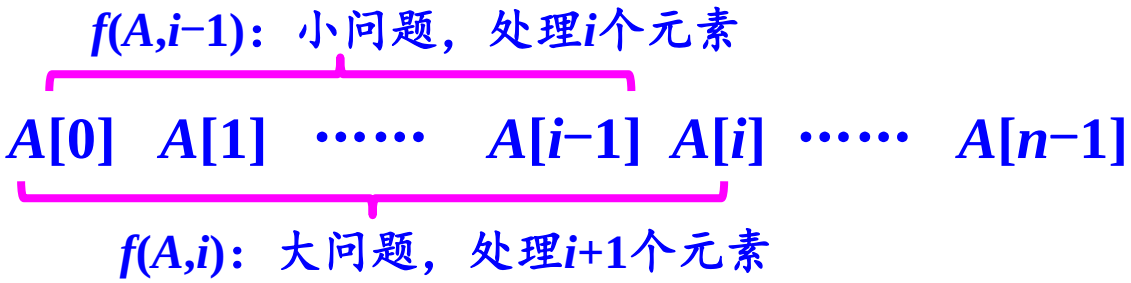
数学归纳法

⟺ 假设$n=k-1$时等式成立，求证$n=k$时等式成立

⟺ 求证$n=1$时等式成立

例如，采用递归算法求实数数组$A[0..n-1]$中的最小值。

假设$f(A,i)$求数组元素$A[0]$～$A[i]$（i+1个元素）中的最小值。



假设$f(A,i-1)$已求出，则$f(A,i)$=MIN($f(A,i-1)$，$A[i]$)，其中MIN()为求两个值较小值函数。

当i=0时，只有一个元素，有$f(A,i)=A[0]$。

因此得到如下递归模型：

$f(A,i)=A[0]$	当i=0时
$f(A,i)$= MIN($f(A,i-1)$，$A[i]$)	其他情况

由此得到如下递归求解算法：

```
float f(float A[],int i)
{     float m;
      if (i==0)
          return A[0];
      else
      {     m=f(A,i-1);
            if (m>A[i])
                return A[i];
            else
                return m;
      }
}
```

递归出口

递归体

5.2.2 基于递归数据结构的递归算法设计

递归数据结构的数据特别适合递归处理 ⇨递归算法

种瓜得瓜：递归性

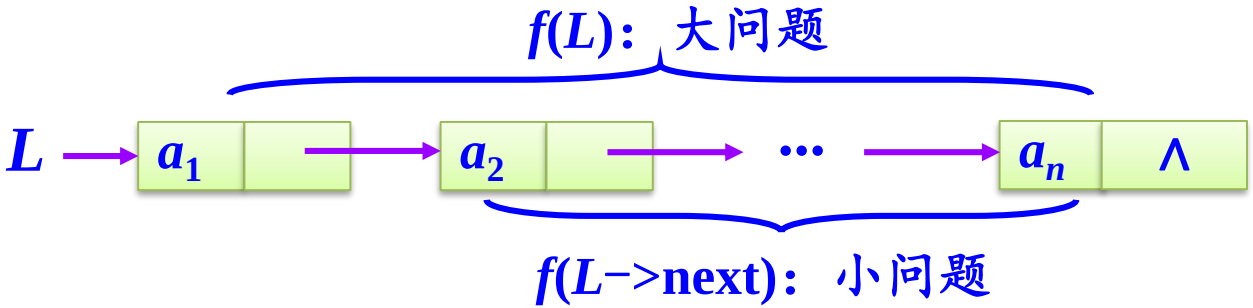
数据：D={瓜的集合}

运算：Op={种瓜}

递归性：

$\mathrm{Op}(x \in D) \in D$

【例5-1】设计不带头节点的单链表的相关递归算法。



把“大问题”转化为若干个相似的“小问题”来求解。

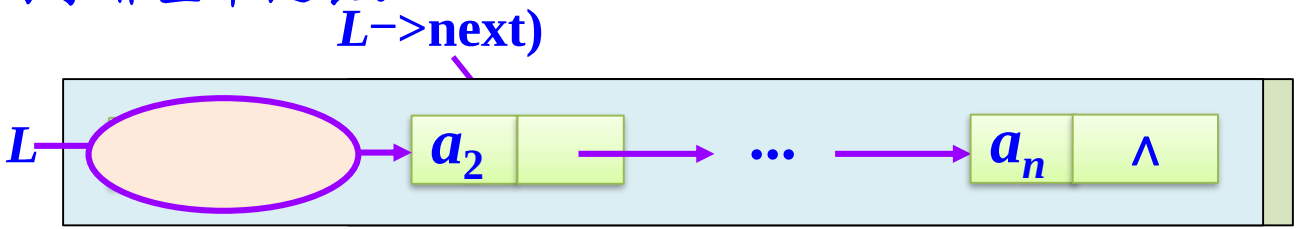
为什么在这里设计单链表的递归算法时不带头节点？

❶ 求单链表中数据节点个数。

设$f(L)$为单链表中数据节点个数。

☑ 空单链表的数据节点个数为0 ⟹ $f(L)=0$ 当L=NULL

☑ 对于非空单链表：



$f(L)$ = $f(L->next)$ + 1

递归模型如下：

$f(L)=0$	当L=NULL
$f(L)=f(L->next)+1$	其他情况

求单链表中数据节点个数递归算法如下：

```
int count(Node *L)
{     if (L==NULL)
            return 0;
      else
            return count(L->next)+1;
}
```

不带头节点单链表L

❷ 正向显示所有节点值。　　❸ 反向显示所有节点值。

$f(L)$：大问题，输出$a_n,\ldots,a_2a_1$

$f(L->next)$：小问题，输出$a_n,\ldots,a_2$

假设$f(L->next)$已求解

$f(L)$ ⇨ $f(L->next)$;输出L->data;

不带头节点单链表L

正向显示所有节点值。

递归模型如下：

f(L) ≡ 不做任何事件

当L=NULL

f(L) ≡ 输出L->data;f(L->next)

其他情况

```
void traverse(Node *L)
{    if (L==NULL) return;
     printf("%d ",L->data);
     traverse(L->next);
}
```

反向显示所有节点值。

递归模型如下：

f(L) ≡ 不做任何事件

当L=NULL

f(L) ≡ f(L->next);输出L->data

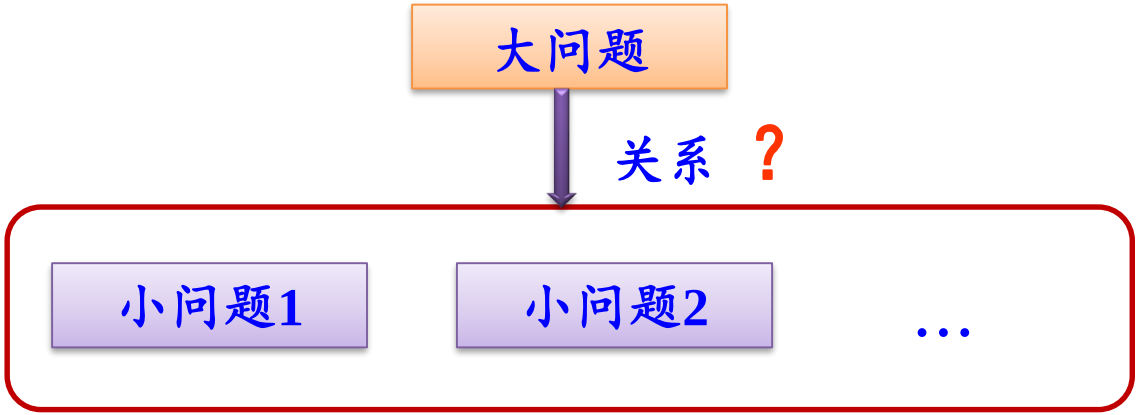
其他情况

```
void traverseR(Node *L)
{    if (L==NULL) return;
     traverseR(L->next);
     printf("%d ",L->data);
}
```

5.3.3 基于递归求解方法的递归算法设计

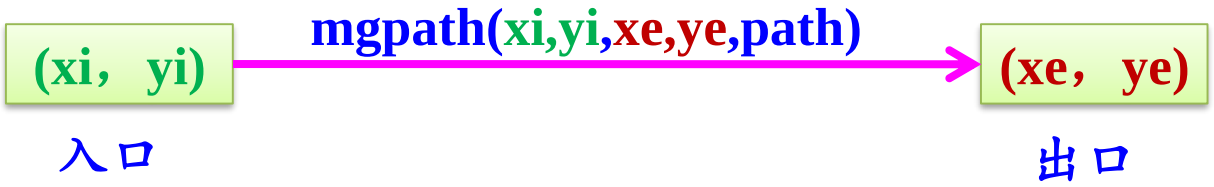
有些问题可以采用递归方法求解（求解方法之一）。

采用递归方法求解问题时，需要对问题本身进行分析，确定大、小问题解之间的关系，构造合理的递归体。



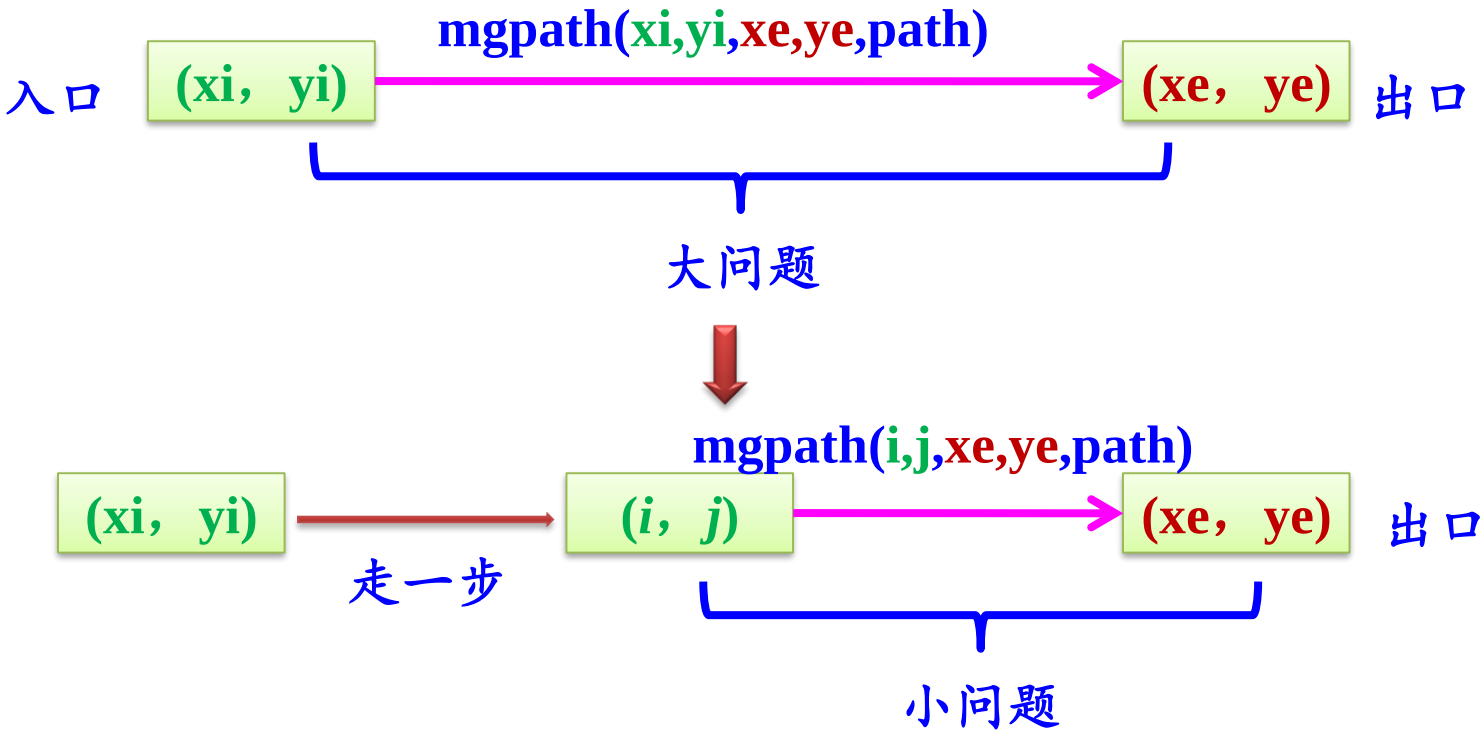
【例5-2】 采用递归算法求解迷宫问题，并输出从入口到出口的所有迷宫路径。

求解问题描述：



mgpath(int xi,int yi,int xe,int ye,PathType path)：

求从(xi,yi)到(xe,ye)的迷宫路径，用path变量保存迷宫路径。


mgpath(xi,yi,xe,ye,path)
入口
(xi，yi)
(xe，ye)
出口
大问题
mgpath(i,j,xe,ye,path)
入口
(xi，yi)
走一步
(i，j)
(xe，ye)
出口
小问题

大问题 ≡ 走一步 + 小问题

求解迷宫问题的递归模型如下：

mgpath(xi,yi,xe,ye,path) ≡ 将(xi,yi)添加到path中;输出path中的迷宫路径;

若(xi,yi)=(xe,ye)

mgpath(xi,yi,xe,ye,path) ≡ 对于(xi,yi)四周的每一个相邻方块(*i*,*j*)：

❶将(xi,yi)添加到path中；

❷ 置mg[xi][yi]=−1;

❸ mgpath(*i*,*j*,xe,ye,path);

❹ path回退一步并置mg[xi][yi]=0;

若(xi,yi)不为出口且可走

在一个“小问题”执行完后回退找所有解

迷宫路径用顺序表存储，它的元素由方块构成的。

其PathType类型定义如下：

```
typedef struct
{      int i;                    //当前方块的行号
       int j;                    //当前方块的列号
}   Box;

typedef struct
{     Box data[MaxSize];
      int length;               //路径长度
}  PathType;                    //定义路径类型
```

```
void mgpath(int xi,int yi,int xe,int ye,PathType path)
//求解路径为:(xi,yi) ⇨ (xe,ye)
{      int di,k,i,j;
       if  (xi==xe && yi==ye)
       {    path.data[path.length].i = xi;
            path.data[path.length].j = yi;
            path.length++;
            printf("迷宫路径%d如下:\n",++count);
            for (k=0;k<path.length;k++)
            {    printf("\t(%d,%d)",path.data[k].i, path.data[k].j);
                 if ((k+1)%5==0)            //每输出每5个方块后换一行
                     printf("\n");
            }
            printf("\n");
       }
```

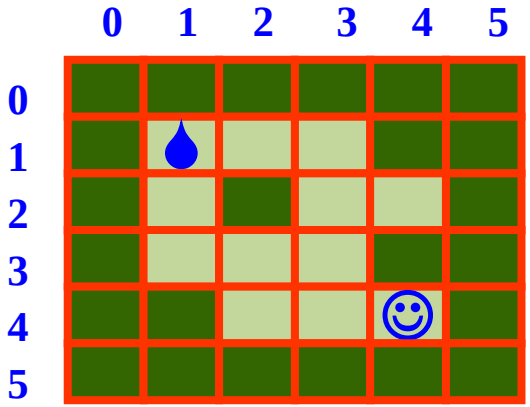
找到了出口，输出一条路径（递归出口）

```
else                                    //(xi,yi)不是出口
{       if (mg[xi][yi]==0)              //(xi,yi)是一个可走方块
        {    di=0;
             while (di<4)               //对于(xi,yi)四周的每一个相邻方位di
             {    switch(di)            //找方位di对应的方块(i,j)
                  {
                  case 0:i=xi-1; j=yi;   break;
                  case 1:i=xi;   j=yi+1; break;
                  case 2:i=xi+1; j=yi;   break;
                  case 3:i=xi;   j=yi-1; break;
                  }
                  ❶ path.data[path.length].i = xi;
                    path.data[path.length].j = yi;
                    path.length++;      //路径长度增1
                  ❷ mg[xi][yi]=-1;      //避免来回重复找路径
```

```
                ❸ mgpath(i,j,xe,ye,path);
                ❹ path.length--;                    //回退一个方块
                  mg[xi][yi]=0;                     //恢复(xi,yi)为可走
              di++;
            } //-while
        } //- if (mg[xi][yi]==0)
    } //-递归体
}
```

本算法输出所有的迷宫路径，可以通过进一步比较找出最短路径（可能存在多条最短路径）。

应用

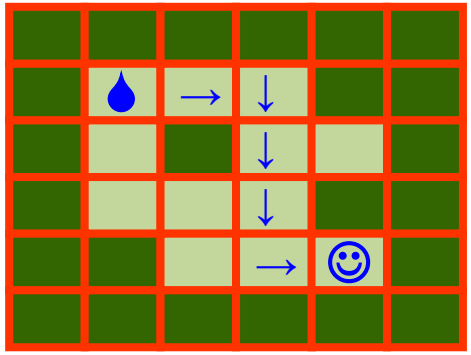



```
int mg[M+2][N+2]=        //M=4,  N=4
{   {1, 1, 1, 1, 1, 1},
    {1, 0, 0, 0, 1, 1},
    {1, 0, 1, 0, 0, 1},
    {1, 0, 0, 0, 1 ,1},
    {1, 1, 0, 0, 0, 1},
    {1, 1, 1, 1, 1, 1}  };
```

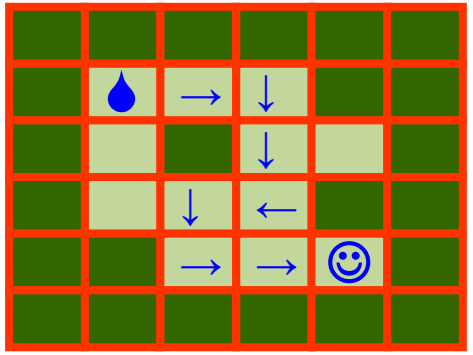
```
void main()
{     PathType path;
      path.length=0;
      mgpath(1,1,4,4,path);
}
```

得到如下4条迷宫路径：

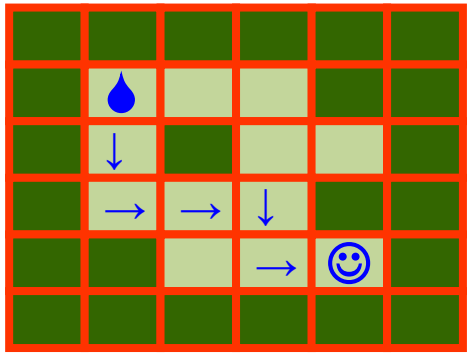
迷宫路径1



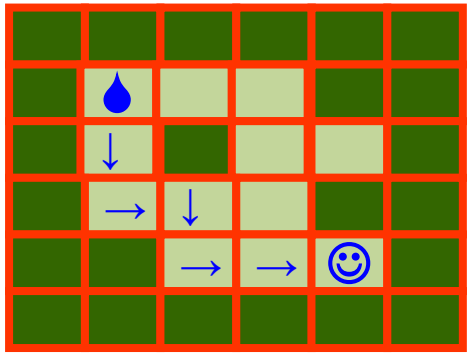
迷宫路径2



迷宫路径3



迷宫路径4



思考题：

迷宫问题的递归求解与用栈和队列求解有什么异同？

——本章完——