第二章 算法

2.1 算法的两要素

2.2 算法的特征

2.3 算法的表示

2.4 常用算法

2.5 算法的设计要求

2.6 算法的复杂度分析

解决问题一般步骤

- 实际问题--〉模型--〉算法--〉程序--〉结果

- 解决问题的核心
  - -- 算法以及算法的处理对象
  - -- 数据的结构

程序与算法

- 何谓算法：
  - 解题过程的准确、完整的描述称作解该问题的算法
- 何谓程序：就是用计算机语言表述的算法
- 流程图就是图形化了的算法
- 程序＝算法＋数据结构

2.1 算法的两要素

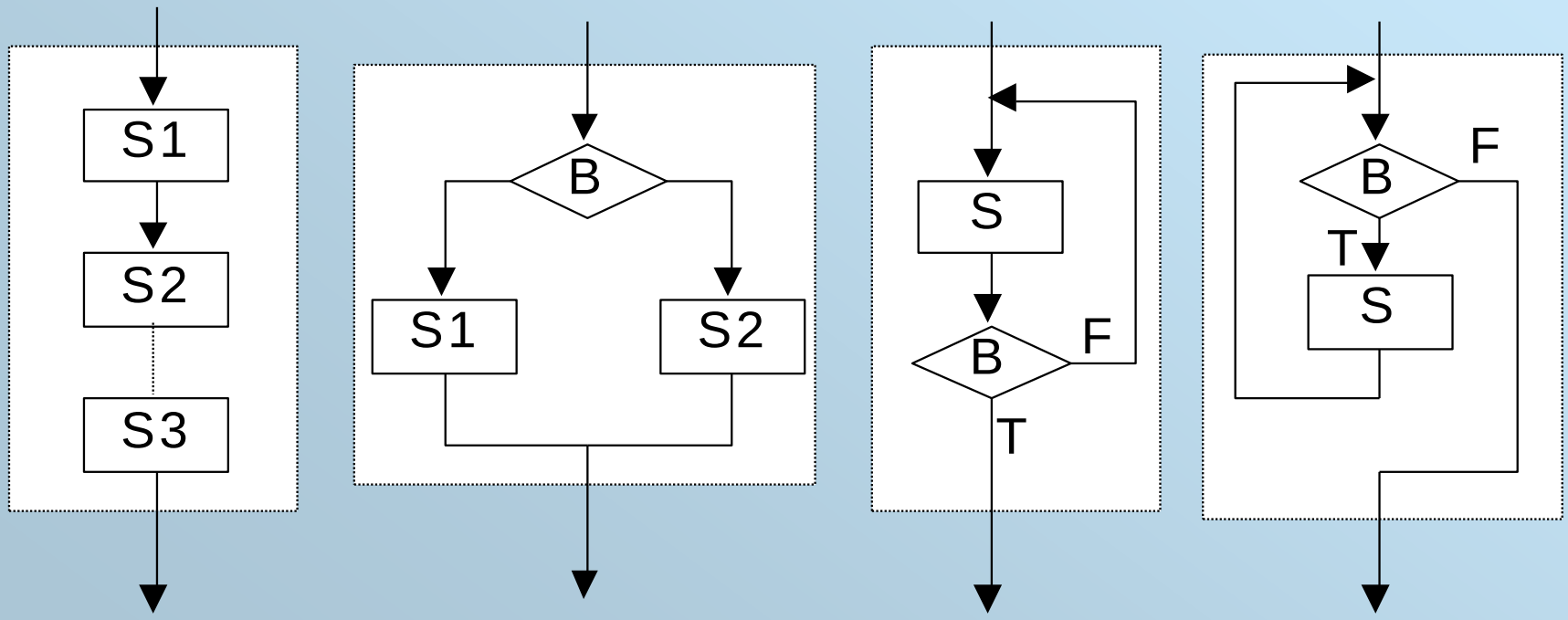
算法由对数据对象的运算和操作与算法的控制结构两要素组成

1. 算法中对数据的运算和操作

- (1) 逻辑运算：“与”、“或”、“非”；
- (2) 算术运算：加、减、乘、除；
- (3) 数据比较：大于、小于、等于、不等于；
- (4) 数据传送：输入、输出、赋值。

2. 控制结构

- 算法的控制结构，决定了各操作的执行次序。用*流程图*可以形象地表示出算法的控制结构
- 任何复杂的算法都可以用顺序、选择、循环三种控制结构组合而成



(a) (b) (c) (d)

2．2　算法的基本特征

算法是由一套计算规则组成的一个过程

1. 确定性　算法中每一个指令须有明确的含义，不能有二义性
2. 可行性　算法中描述的操作都可实现,执行结果能达到预期目标
3. 输　出　每种算法必须有确定的结果，产生一个或多个输出
4. 输　入　每个算法必须有0个（自动生成初始数）或多个输入
5. 有穷性　解答必须在有限步内得到,不能出现“死循环”

我们可以得出如下的结论：算法是一个过程，这个过程由一套明确的规则组成，这些规则指定了一个操作的顺序，以便用有限的步骤提供特定类型问题的解答。

2. 3 算法的表示

算法设计一般是由粗到细的过程，一般可以使用下面几种类型的工具描述算法：

1. 自然语言

自然语言描述算法通俗易懂，但它有着难以克服的缺陷：

(1) 易产生歧义性

(2) 语句繁琐冗长，很难清楚地表达算法的逻辑流程

(3) 当今的计算机尚不能处理用自然语言表示的算法

2. 专用工具

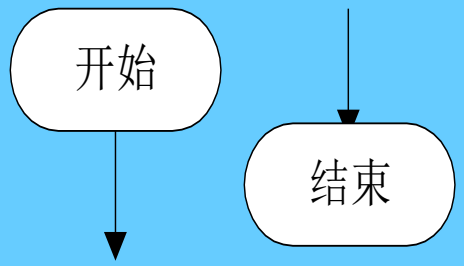
常用的有流程图、问题分析(PAD)和NS盒图、伪代码等。

3. 算法描述语言

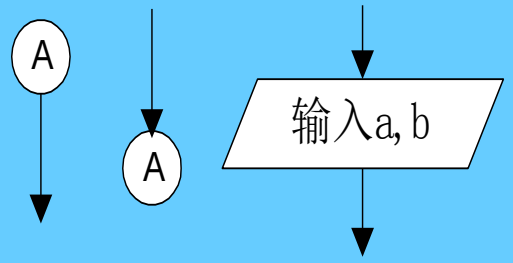
为了便于转换成某种编程语言，一般采用准程序设计语言作算法描述语言。例如，类C语言继续

流程图　是采用不同的几何图形来描述算法的逻辑结构，每个几何图形表示不同性质的操作

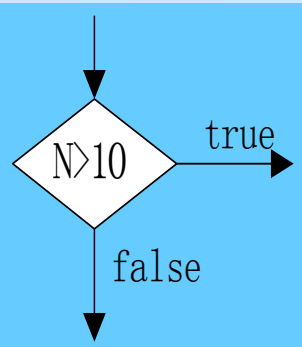
常用流程图符号：



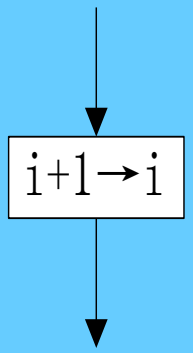
(a) 起止框、连接框



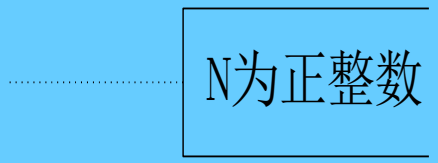
(b) 输入输出框



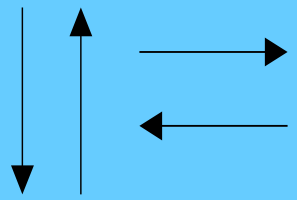
(c) 判断框



(d) 处理框



(e) 注释框



(f) 流向线

返回

2．4　常用算法

1. 枚举法（穷举法）（常用）

基本思想是：

- 先依据题目的部分条件确定答案的大致范围
- 在此范围内对所有可能的情况逐一验证，直到全部情况验证完
- 若某个情况使验证符合题目的条件，则为本题的一个答案；若全部情况验证完后均不符合题目的条件，则问题无解
- 例：百元买百鸡：　公鸡5元、母鸡3元、小鸡1元

2. 迭代法

- 使一个复杂问题的求解过程转化为相对简单的迭代算式的重复执行过程。
- 基本思想：通过列举少量的特殊情况，经过分析，最后找出一般的关系。
- 基本方法：
  - 首先确定一个合适的迭代公式，选取一个初始近似值以及解的误差
  - 然后用循环处理实现迭代过程，终止循环过程的条件是前后两次得到的近似值之差的绝对值小于或等于预先给定的误差
  - 并认为最后一次迭代得到的近似值为问题的解。
- 例：数值计算方法

3. 递归法

- 基本思想：将复杂问题逐层分解，最后归结为一些简单的问题。
- 如果一个过程直接或间接地调用它自身，则称该过程是递归的
- 例：输出自然数1到n。

```
#include “stdio.h”
Wrt1(int n)
{if (n!=0)
    {wrt1(n-1);printf(“%d\n”,n); }
    return:;
}
```

递归过程必须有一个递归终止条件，

当n=0时定义为1，是阶乘递归定义的递归出口

递归则是从函数本身出发，逐次上溯调用其本身求解过程，直到递归的出口，然后再从里向外倒推回来，得到最终的值

4. 递推法

- 基本思想：从已知的初始条件出发，逐次推出所要求的中间结果和最后结果。（本质上属归纳法）
- 所谓递推法，它的数学公式也是递归的。只是在实现计算时与递归相反。从给定边界出发逐步迭代到达指定计算参数。

例：求阶乘

- f(n)＝n！＝n×(n-1)！＝n×f(n-1)
- 要计算10!，可以从递推初始条件f(0)=1出发，应用递推公式f(n)=n×f(n-1)逐步求出f(1)、f(2)...、f(9)、最后求出f(10)的值
- 递推操作是提高递归函数执行效率最有效的方法，科技计算中最常见

5. 分治法

- 解一个复杂的问题时，尽可能地把这个问题分解为较小部分，找出各个的解，然后再把各部分的解组合成整个问题的解，这就是所谓的分治法
- 分治法的基本步骤

  1.分解：将原问题分解为若干个规模较小，相互独立，与原问题形式相同的子问题；

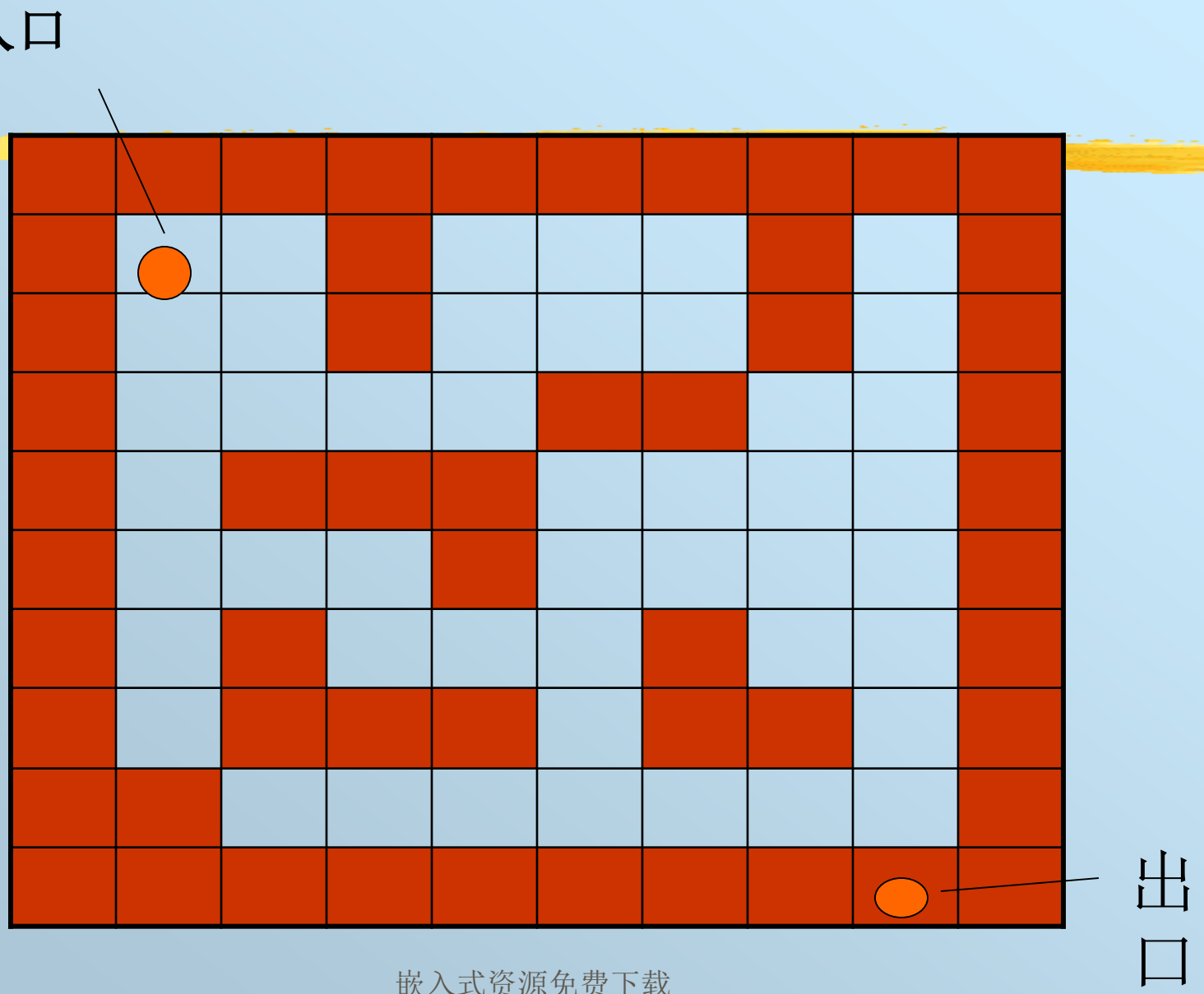
  2.解决：若子问题规模较小而容易被解决则直接解，否则递归地解各个子问题；

  3.合并：将各个子问题的解合并为原问题的解。
- 常用于：人工智能、查找、检索
- 例：将无序的N个元素按递增次序排序（P79）

6. 回溯法

- **基本思想：**
  - **通过对问题的分析，找出一个解决问题的线索；**
  - **然后沿着这个线索逐步试探。**
    - 若成功，就得到问题的解
    - 若失败，就逐步回退，换别的路线再进行试探
- **例：求解骑士周游（P80）、老鼠走迷宫等**

迷宫求解



回溯法的算法：

```
Proc Backtracking(succ : Boolean)
确定起始状态值走第一步
确定下一步还有几种可能
选一可能走下一步，记住可能和本步特征
做完新一步应做的事
While 目标未达到 do
  确定下一步有几种可能
  While 没有可能and 还有上一步 do
    回退上一步
    查有无下一可能
  Enddo
  If 上一步没有了Then
        return  (SUCC=FALSE)
  EndIf
    选一可能走一步，记住可能和本步特征
      做完新一步应做的事
Enddo
return (SUCC=TRUE)
End Backtracking
```

2.5 算法的设计要求

（1）正确性（Correctness）

算法应满足具体问题的需求。

（2）可读性（Readability）

算法的第一目的是为了阅读和交流

有助于对算法的理解、调试和修改。

（3）健壮性（Robustness）

算法应具有容错处理。

当输入非法数据时，算法应对其作出反应，而不是产生莫名其妙的输出结果。

（4）高效率与低存储量

实际问题的求解往往是求得时间和空间的统一、折中。

2.6 算法的复杂度分析

时间复杂度：指执行算法所需要的计算工作量
（算法中基本操作重复执行的次数）

用“O（数量级）”来表示，称为“阶”。
常见的时间复杂度有： O（1） O（logn） O（n ） O(n2)
常数阶 对数阶 线性阶 平方阶

不同算法的时间复杂度可有不同

```
for (i=1; i<=n; ++i)
    for (j=1; j<= n; ++j)
        c[i][j]= 0;
```

算法时间复杂度表示为$f(n) = O(n^2)$

空间复杂度：指执行算法程序所需要的内存空间、初始数据、执行时所需额外空间

度量同时间复杂度

```
for (i=1; i<=n; ++i)
    for (j=1; j<= n; ++j)
        c[i][j]= 0;
```

小结

算法是由一套计算规则组成的一个过程
算法与程序的区别
算法的两要素:运算和操作，控制结构
算法的特征：确定，可行，有穷，IN/OUT
算法的表示:自然语言，流程图和程序设计语言
常用算法设计方法
算法的设计要求

- （1）正确性（Correctness）
- （2）可读性（Readability）
- （3）健壮性（Robustness）
- （4）高效率与低存储量