第三章
算法基础

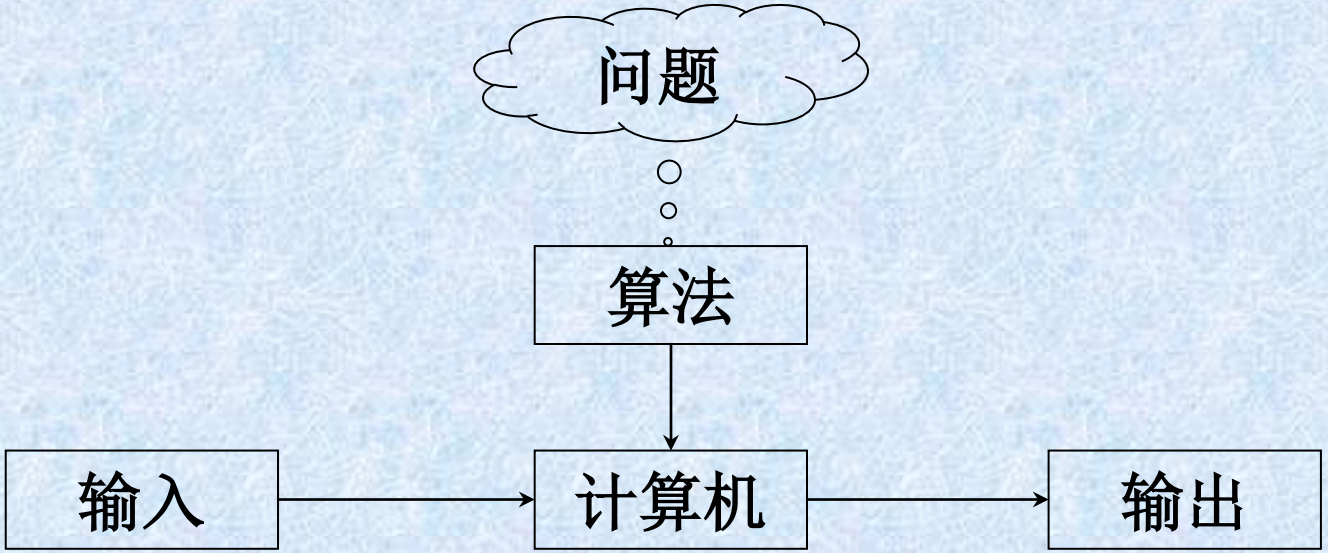
3.1 算法的概念

1974年图灵奖获得者Donald Ervin Knuth:

计算机科学就是算法的研究

The Art of Computer Programming

算法与计算机之间的关系



3.1 算法的概念

生活中的算法

- 电子商务中支付账号
- 密码的加密
- 电子地图中路线的查找
- 搜索引擎的快速信息查找

3.1 算法的概念

一、算法的起源

公元前300年：古希腊著名数学家欧几里得提出求最大公约数的一种算法,即辗转相除法又称欧几里得算法。

公元263年：三国魏人刘徽注释《九章算术》中不仅对原书的方法、公式和定理进行一般的解释和推导，而且在其论述中多有创造。如他运用割圆术得出圆周率的近似值3927/1250＝3.1416。

公元825年：波斯数学家al-Khwarizmi撰写了著名的Persian Textbook中概括了进行四则算术运算的法则。Algorithm(算法)一词就来源于这位数学家的名字。

3.1 算法的概念

二、算法的定义

[算法3.1]欧几里得算法。

输入：正整数m、n

输出：m、n的最大公约数

① r＝m mod n

② 若r＝0，输出最大公约数n

③ 若r≠0，令m＝n，n＝r，转①继续

算法：是解决某一特定问题的一组有穷规则的集合。

算法：对特定问题求解步骤的一种描述，是由若干条指令组成的有穷集合。

3.1 算法的概念

三、算法的特征

确定性：算法中每一个步骤都是清晰的、无歧义

有穷性：算法必须在有限步内终止

输　入：有零个或多个输入，作为初始状态

输　出：有一个或多个输出，作为计算结果

可行性：算法中的操作可通过有限次基本运算来实现

判断一个算法的好坏主要依据如下标准：

正确性：在合理输入下能在有限时间内得出正确结果

可读性：算法主要是为了人的阅读与交流，其次执行

健壮性：算法应具备检查错误和对错误进行处理能力

效　率：算法执行时所需计算机资源的多少

3.2 算法的描述

- **算法的描述目的**
  - 记录算法思想
  - 方便他人理解算法
- **算法的描述方法**
  - 自然语言
  - 流程图
  - 伪代码
  - 程序设计语言

3.2 算法的描述

一、自然语言

自然语言是人们日常进行交流的语言，如汉语、英语等

优点：通俗易懂，即使没有学过算法也能看懂算法执行

缺点：不够严谨，容易出现歧义和错误

[例题]利用自然语言描述欧几里得算法。

① 输入m、n

② 判断n是否为0，如果不为0，转步骤③，否则转④

③ m对n取余，其结果赋值给r，n赋给m，r赋给n，转②

④ 输出m，算法结束

3.2 算法的描述

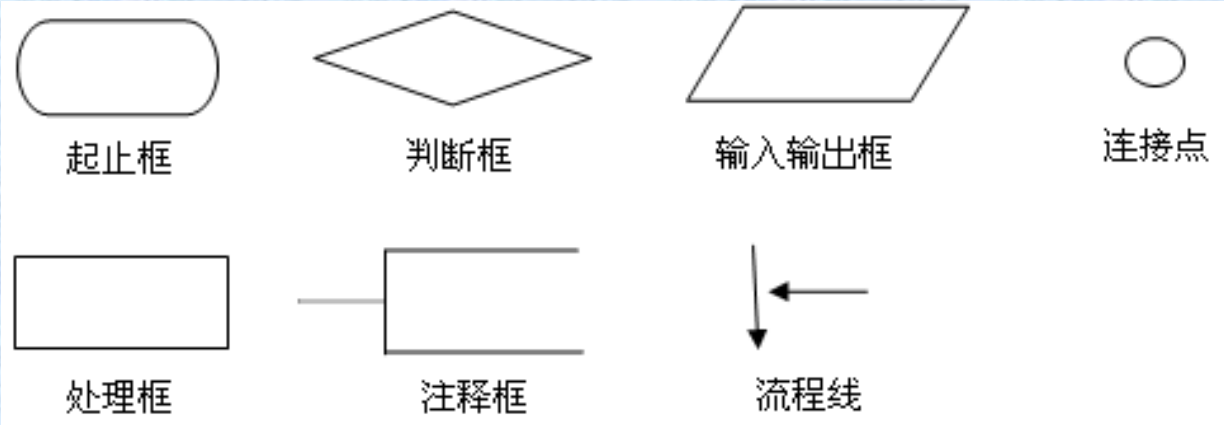
二、流程图

常用来描述算法的图形工具有：流程图或程序框图、N-S图和PAD图。

优点：直观形象，简洁明了。

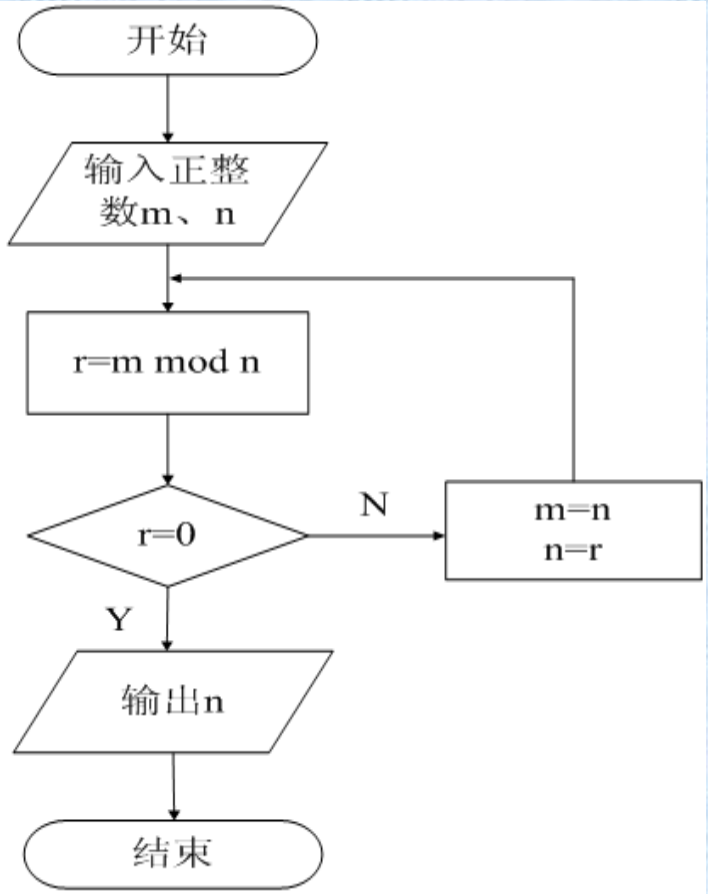
缺点：画起来费事，不易修改。

常用的流程图符号：



3.2 算法的描述

[例题]利用流程图描述欧几里得算法。



练习 1

画出计算10！的流程图

3.2 算法的描述

三、伪代码

伪代码是由带标号的指令构成，但是它不是C、C++、Java等通常使用的程序设计语言，而是算法步骤的描述。

伪代码介于自然语言和程序设计语言之间。

伪代码的具体表示：

赋值语言：←

分支语句：if…then…[else]

循环语句：while, for, repeat until

转向语句：goto

输出语句：return

调用：

注释：//…

3.2 算法的描述

[例题]利用伪代码描述欧几里得算法。

```
输入：正整数m、n
输出：m、n的最大公约数
GREATEST-COMMON-DIVISOR(m、n)
1  repeat
2     r ← m mod n
3     m ← n
4     n ← r
5  until r=0
6  return m
```

3.2 算法的描述

[例题]利用伪代码描述计算10！的算法。

```
输出:10!
FACTOR-10
1   sum ← 1
2   FOR j ← 2 TO 10
3       DO sum ← sum * j
4   RETURN sum
```

练习 2

用伪代码描述计算1+2+3+……+10的流程图

3.2 算法的描述

四、程序设计语言

程序设计语言是一个能完整、准确和规则地表达人们的意图，并用以指挥或控制计算机工作的**符号系统**，如C、C++、Java等程序设计语言可以描述算法。

优点：描述的算法能在计算机上直接执行

缺点：抽象性差、不易理解且有严格的语法限制等。

3.2 算法的描述

[例题]利用C语言描述欧几里得算法。

```
输入：正整数m、n
输出：m、n的最大公约数
int gcd(int m, int n)
{  int r;
   do
   {  r = m % n;
      m = n;
      n = r;
   } while(r);
   return m;
}
```

3.3 算法的设计

算法是解决问题的方案，由于实际问题千奇百怪，因而制定出的解决方案也将千差万别。

常见的算法有：

- 穷举法
- 回朔法
- 递归
- 分治法
- 贪心法
- 动态规划

3.3 算法的设计

一、穷举法(又称蛮力算法)

穷举法指在问题的解空间范围内逐一测试，找出问题的解。它是一种简单而有效的算法设计策略同时也是一种很容易应用的方法。

- 穷举法的应用
  - 国王的婚姻中国王使用的算法
  - 旅行商问题中逐条路线计算
  - 密码学中的暴力破解法
  - 图论中四色定理的证明
  - 百钱买百鸡问题

3.3 算法的设计

[案例一]暴力破解法是一种用穷举法实现的密码破译方法。

- 最原始、最基本的攻击方式，对密码进行逐一测试直到找到真正的密码。
- 原则上可以破译所有密码，但费时费力。
- 密码暴力破解软件：89Winrar
  QQ密码暴力破解软件

3.3 算法的设计

[案例二]四色定理(又称四色问题或四色猜想)。

四色问题描述：任何一张地图只用四种颜色就能使具有共同边界的国家着上不同的颜色。

数学语言表示：将平面任意地细分为不相重叠的区域，每一个区域总可以用1、2、3、4这四个数字之一来标记，而不会使相邻的有公共边界的两个区域得到相同的数字。

证明四色定理(穷举法)：

①利用数学理论推出证明所有例子可以归约到证明有限个特例上；

②利用计算机程序产生了所有特例(大约1700个例子)，通过穷举发现所有特例都是四着色的。

3.3 算法的设计

[案例三]百钱买百鸡问题

百钱买百鸡：鸡翁一，值钱五
鸡母一，值钱三
鸡雏三，值钱一
问翁、母、雏各几何？

意思是：公鸡每只5元、母鸡每只3元、小鸡3只1元，用100元钱买100只鸡，求公鸡、母鸡、小鸡的只数。

3.3 算法的设计

设鸡翁、鸡母、鸡雏的个数分别为x、y、z，根据题意可得如下方程组：

$$\begin{cases} 5x+3y+z/3=100 \\ x+y+z=100 \\ 1\leqslant x<20,\ 1\leqslant y<33,\ 3\leqslant z<100,\ z \bmod 3=0 \end{cases}$$

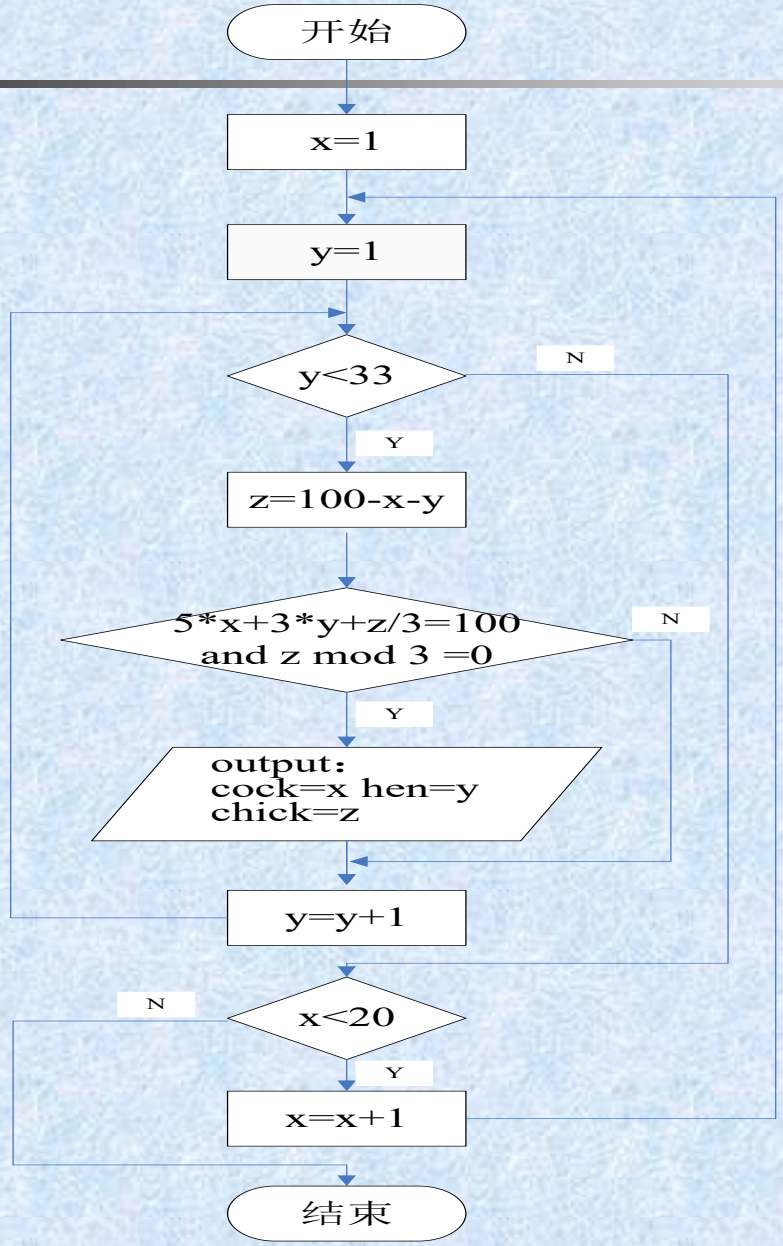
测试集合：$1\leqslant x<20,\ 1\leqslant y<33,$

测试条件：$x+y+z=100$

$5x+3y+z/3=100$

$z \bmod 3=0$

用穷举法求解，即在以上有限测试集中，对每组x、y的值，找到满足测试条件的z，如果有则得到百钱百鸡问题的解（这个解不是唯一的）。


开始
x=1
y=1
y<33
N
Y
z=100-x-y
5*x+3*y+z/3=100
and z mod 3 =0
N
Y
output:
cock=x hen=y
chick=z
y=y+1
x<20
N
Y
x=x+1
结束

3.3 算法的设计

巧妙和高效的算法很少来自于穷举法，但基于以下因素，穷举法仍是一种重要的算法设计策略：

①穷举法几乎可以通用于任何领域的问题求解，可能是唯一一种解决所有问题的一般性方法；

②即使效率低下，仍可用穷举法求解一些小规模的问题实例；

③如果解决的问题实例不多，而穷举法可用一种可接受的速度对问题求解，那么花时间去设计一个更高效地算法是得不偿失的。

[思考题]举例说明生活中的穷举法应用。

3.3 算法的设计

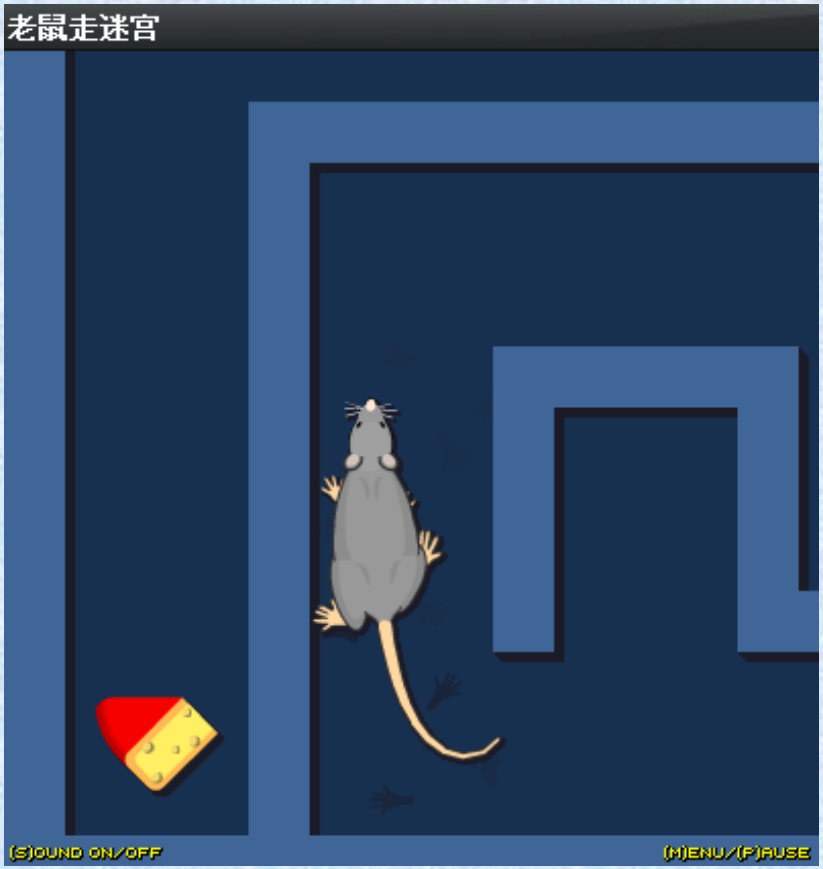
二、回溯法

回溯法是一种选优搜索法，按选优条件向前搜索，以达到目标。在搜索过程中，能进则进，不能进则退回来，换一条路再试，通过此种方式提高搜索效率，减少不必要的测试。

- 回溯法的应用
  - 迷宫问题
  - 搜索引擎中的网络爬虫
  - 八皇后问题

3.3 算法的设计

[案例一]老鼠走迷宫



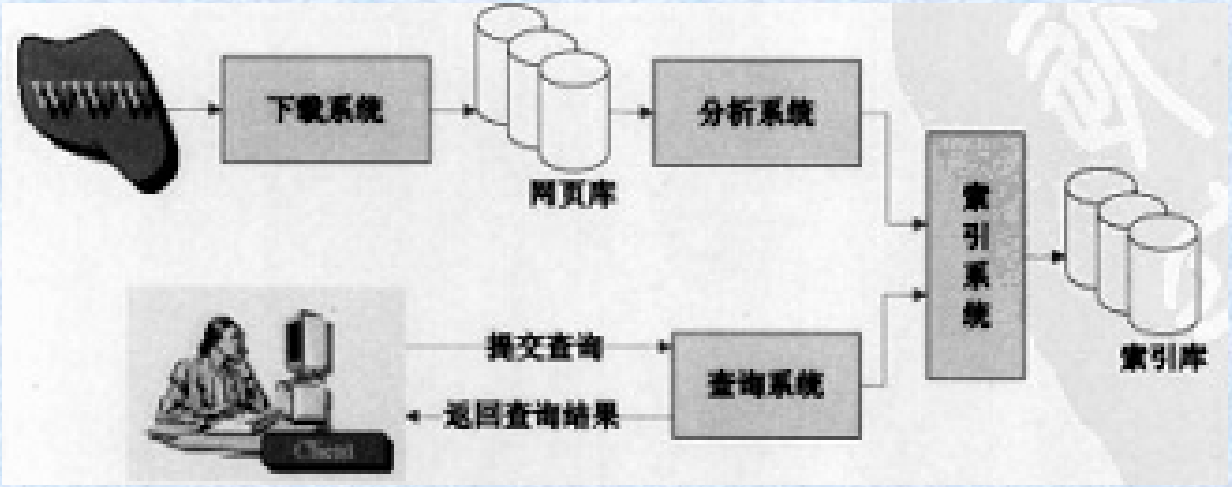
老鼠从迷宫入口出发，任选一条路线向前走，在到达一个岔路口时，任选一个路线走下去…，如此继续，直到前面没有路可走时，老鼠退回到上一个岔路口，重新在没有走过的路线中任选一条路线往前走。按这种方式走下去，直到走出迷宫。

3.3 算法的设计

[案例二]搜索引擎中的网络爬虫。

搜索引擎是指根据一定的策略、运用特定的计算机程序从互联网上搜集信息，在对信息进行组织和处理后，为用户提供检索服务，将用户检索相关的信息展示给用户的系统。百度和谷歌等是搜索引擎的代表。

搜索引擎的组成：下载、索引和查询。

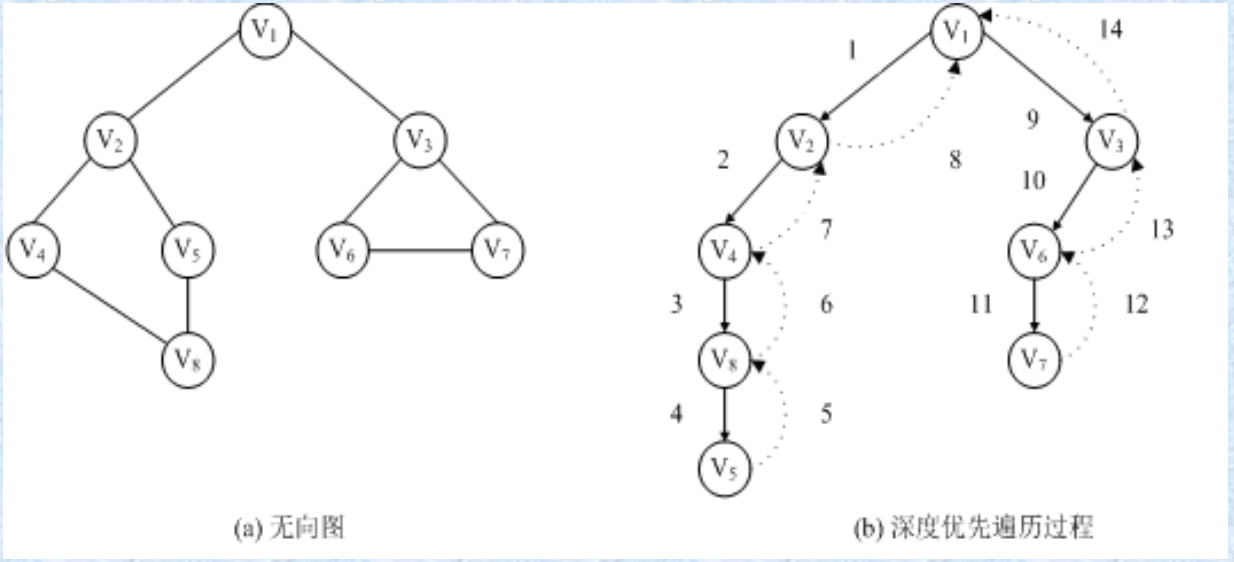


3.3 算法的设计

网络爬虫：自动下载互联网所有网页。

网络爬虫原理：图的遍历，从图中某一顶点出发访遍图中所有顶点，且使每个顶点仅被访问一次。

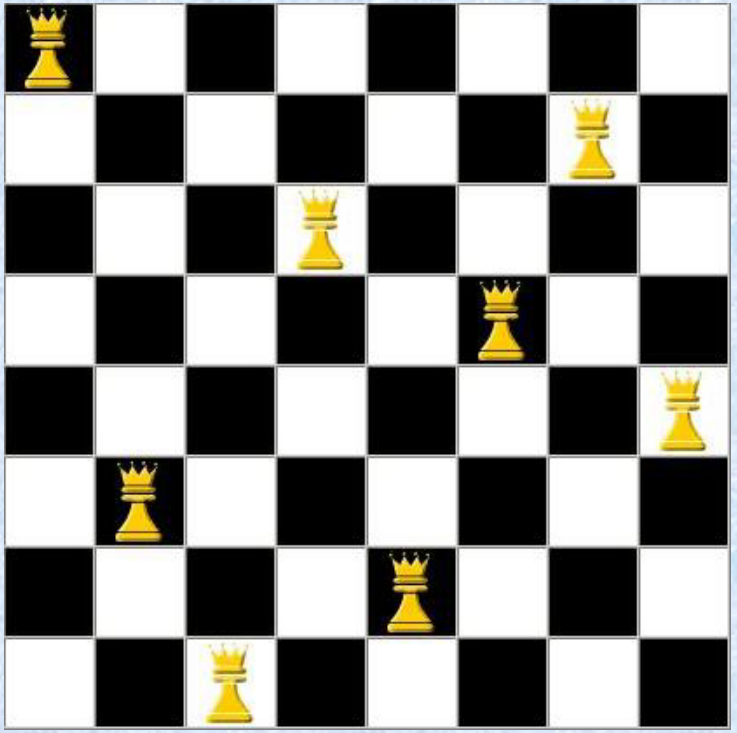
回溯算法：图的深度优先遍历。



深度优先遍历顺序：$V_1, V_2, V_4, V_8, V_5, V_3, V_6, V_7$

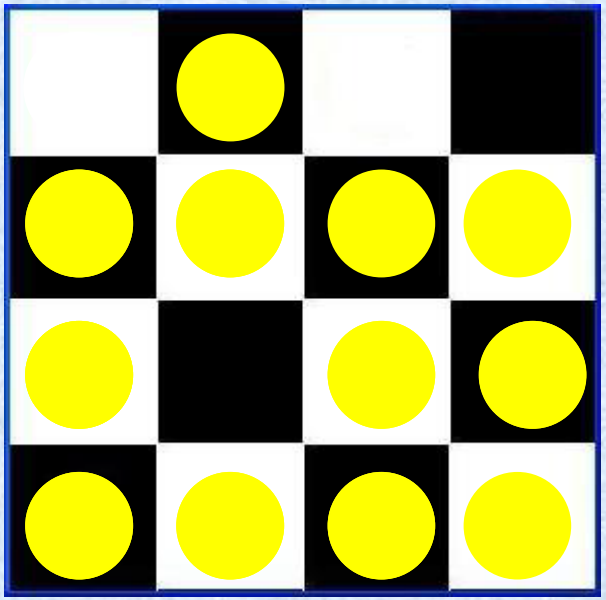
3.3 算法的设计

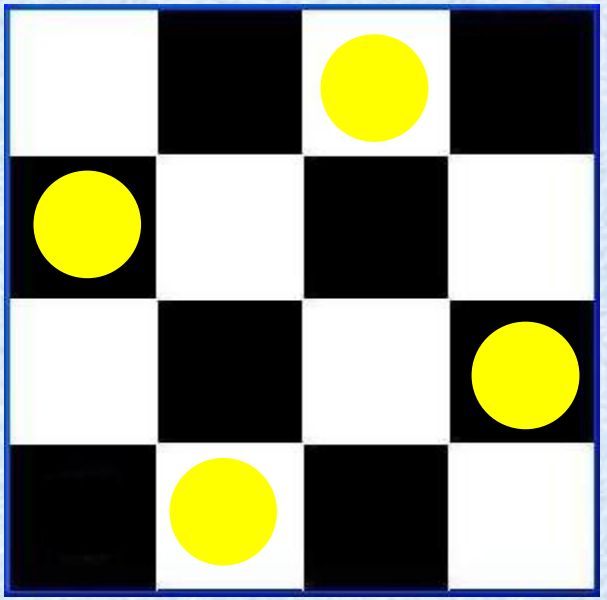
[案例三]八皇后问题。在8×8格国际象棋的棋盘上摆放八个皇后，使其不能互相攻击，即任意两个皇后都不能处于同一行、同一列或同一斜线上问有多少种摆法?



3.3 算法的设计

回溯法解八皇后问题思路：逐行摆放皇后。初始第1行皇后放第1列；摆放第i行皇后时，从第1列开始，逐列判定是否与前i-1行皇后攻击，直到找到一个不攻击的位置，继续第i+1行的摆放；若第i行无摆放位置，则拿掉该行皇后，回溯至第i-1行，第i-1行皇后从当前位置的下一列开始判定，继续搜索。当第1行皇后的摆放位置超出棋盘时，全部求解过程结束。(92种)





思考

两皇后或三皇后问题是否有解?

3.3 算法的设计

回溯法有通用解法之称，当一个问题没有显而易见的解法时，可尝试使用回溯法求解，这实际是与穷举法一致的，因其本质仍是穷举。需要注意，回溯和穷举虽然能解很多问题，但其算法效率可能很低。

回溯法的基本思想是能进则进，不能进则退。为了求得问题的解，先选择某一种可能情况向前探索，在探索过程中，一旦发现原来的选择是错误的，就退回一步重新选择，继续向前探索，如此反复进行，直至得到解或确定该问题无解。

回溯法是求解实际问题的一个重要算法，很多无法使用贪心算法和动态规划算法进行求解的问题，都可以使用回溯算法进行求解，并且可以保证得到问题的最优解。

3.3 算法的设计

三、递归算法

递归：直接或间接地调用自身的算法。

递归思想就是用与自身问题相似但规模较小的问题来描述自己。

- 递归算法的应用
  - 盗梦空间(美国影片)
  - 欧几里得算法
  - 德罗斯特效应(Droste Effect)
  - 斐波纳契数列(Fibonacci数列)

3.3 算法的设计

[案例一]德罗斯特效应。递归的一种视觉形式，它指一张图片的某个部分与整张图片相同，如此产生无限循环。

3.3 算法的设计

[案例二]1202年，意大利数学家斐波纳契出版了他的算盘全书。他在书中提出了一个关于兔子繁殖问题：如果一对兔子每月能生一对小兔(一雄一雌)，而每对小兔在它们出生后的第三个月里，又能开始生一对小兔，假定在不发生死亡的情况下，由一对出生的小兔开始，50月后会有多少对兔子？

分析：第一个月只有一对兔子，第二个月仍只有一对兔子，第三个月兔子对数为第二个月兔子对数加第一月兔子新生的对数。同理，第i个月兔子对数为第i-1月兔子对数加第i-2月兔子新生的对数。即从第一个月开始计算，每月兔子对数依次为：
1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, …。

$$F_n = \begin{cases} 1 & n \le 2 \\ F_{n-1} + F_{n-2} & n > 2 \end{cases}$$

3.3 算法的设计

兔子繁殖的规律

月数	小兔子对数	中兔子对数	老兔子对数	兔子总数
1	1	0	0	1
2	0	1	0	1
3	1	0	1	2
4	1	1	1	3
5	2	1	2	5
6	3	2	3	8
7	5	3	5	13
⋮	⋮	⋮	⋮	⋮

3.3 算法的设计

Fibonacci数列递归算法的伪代码描述：

Fibonacci数列的递归算法

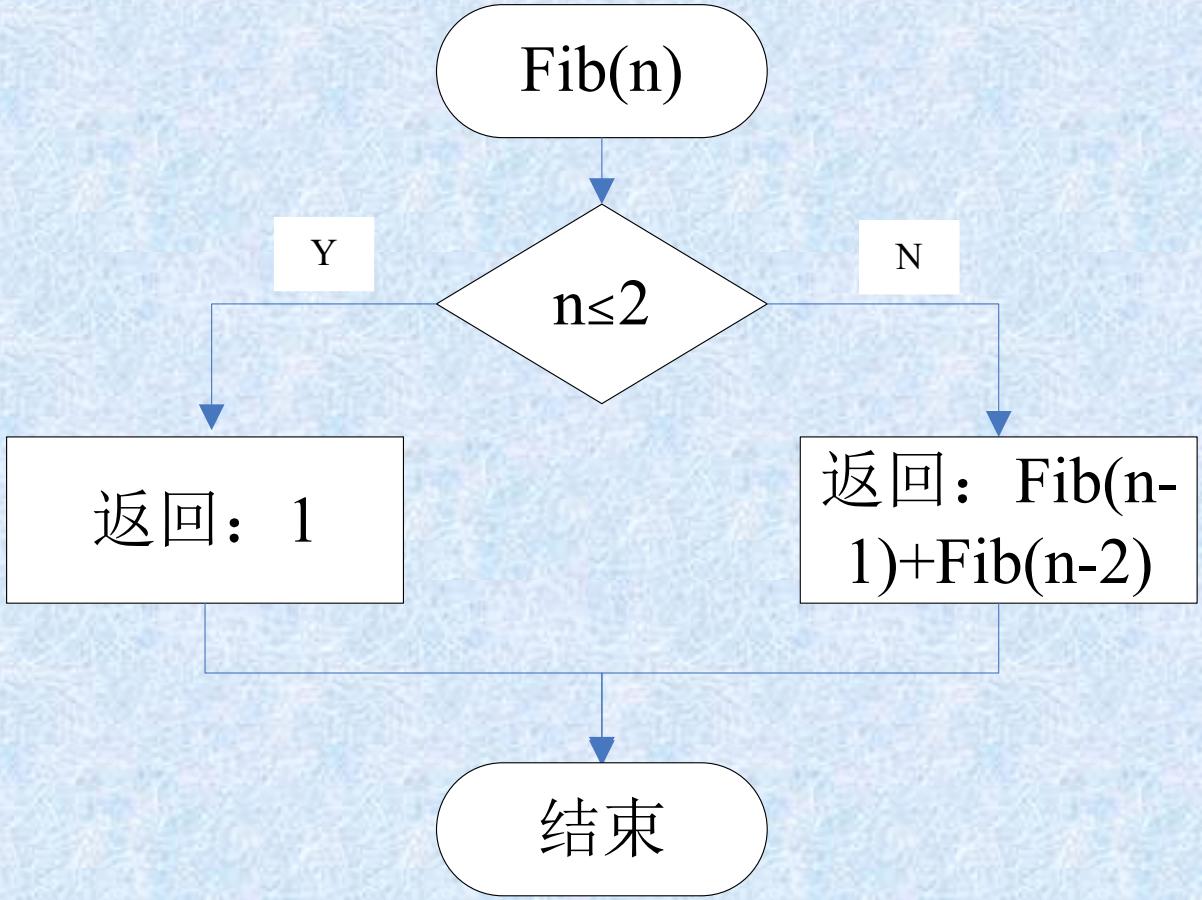
输入：正整数n

输出：Fibonacci数列的第n项

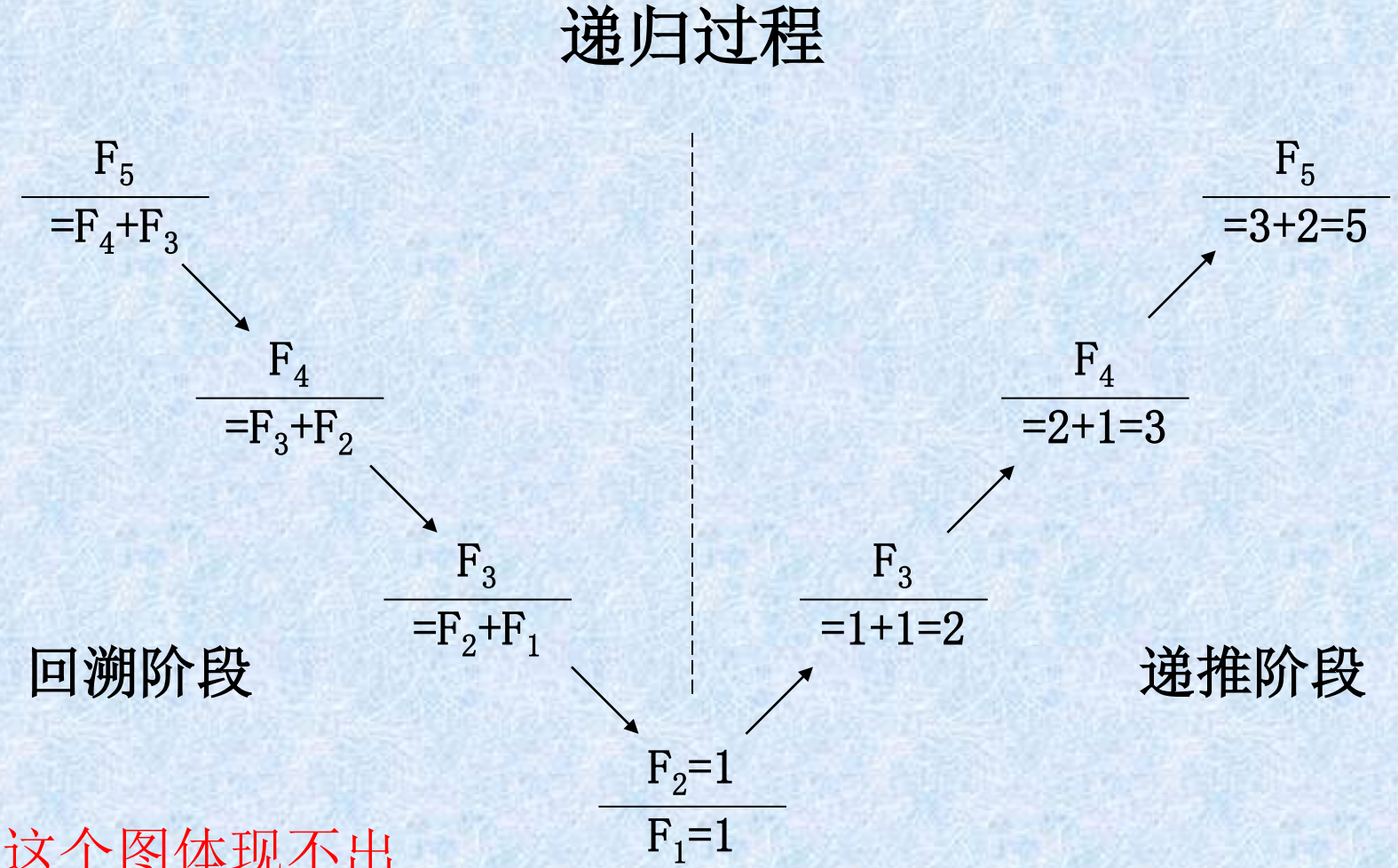
```
Fib(n)
1   IF n≤2
2         THEN RETURN 1
3   RETURN Fib(n-1)+Fib(n-2)   //调用自身
```

3.3 算法的设计

Fibonacci数列递归算法的流程图：



3.3 算法的设计

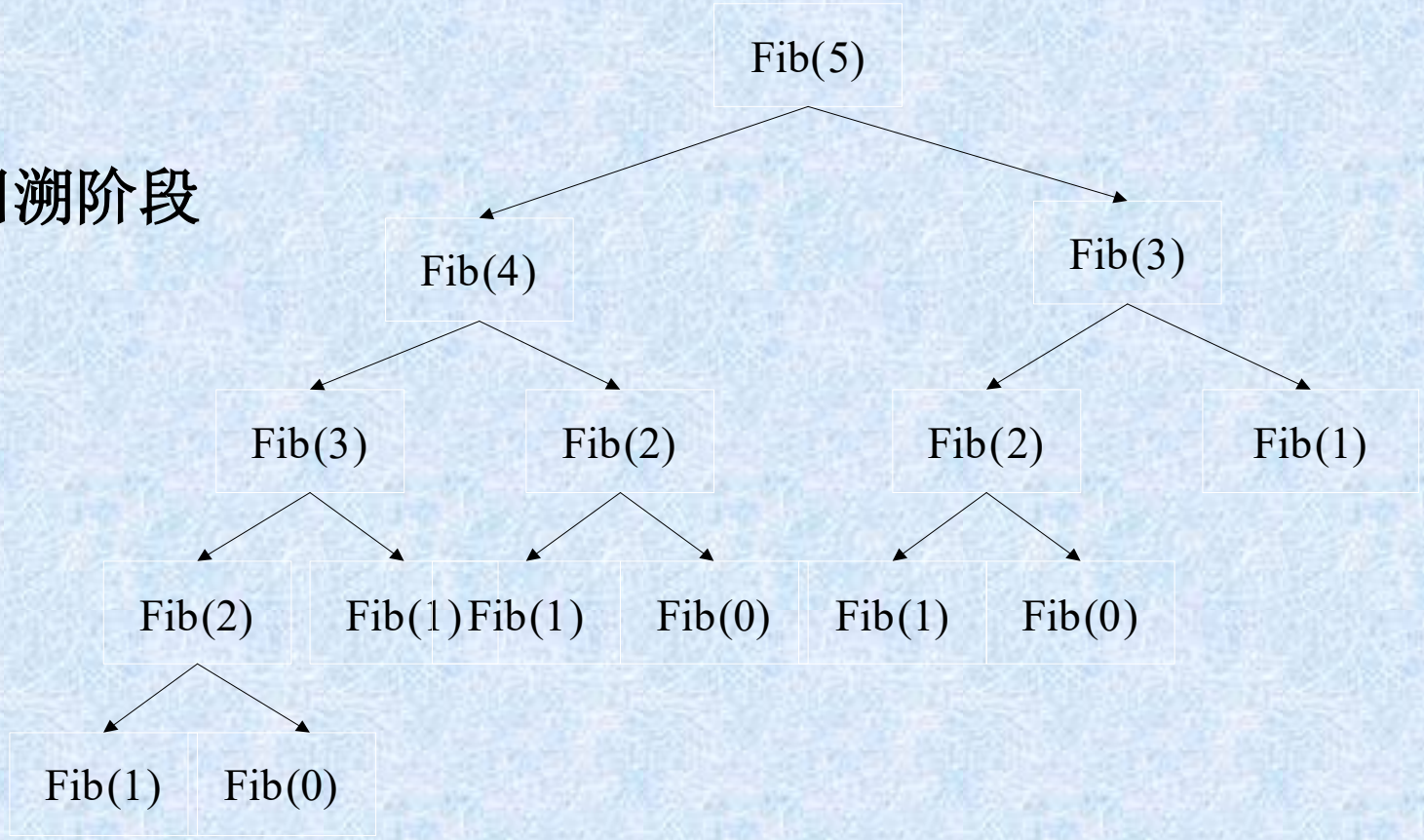


（这个图体现不出重复计算）

3.3 算法的设计

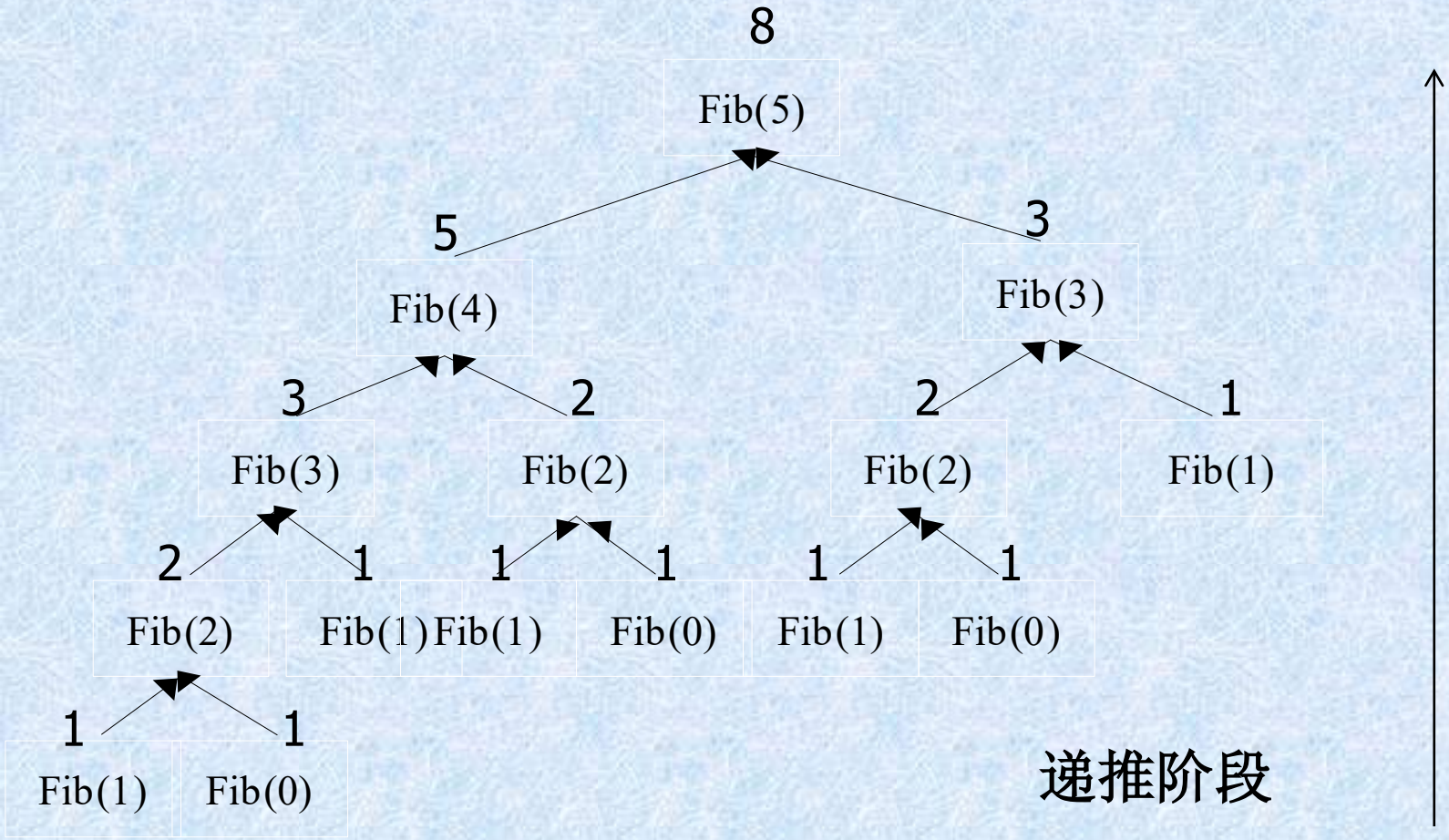
递归过程

回溯阶段



3.3 算法的设计

递归过程



3.3 算法的设计

递归算法的主要优点：结构清晰、可读性强，而且容易用数学归纳法来证明算法的正确性，因此它为设计算法、调试程序带来了很大方便。

递归算法的主要缺点：递归算法的运行效率相对较低，无论是耗费的计算时间还是占用的存储空间都比非递归算法要多。通常的解决方法是消除递归算法中的递归调用，使递归算法转化为非递归算法。

3.3 算法的设计

四、分治法

分治算法：将一个难以直接解决的大问题，分解成一些规模较小的子问题，以便各个击破，分而治之。如果子问题还比较大，可反复使用分治算法，直到最后的子问题可以直接得出它们的结果。由于分治算法的子问题类型常与原来的相同，因而很自然地使用递归算法。

- 分治法的应用
  - 国王的婚姻中宰相的策略
  - 二分查找
  - 大型企业的组织架构
  - 大型体育赛事的赛程安排（预赛和决赛）
  - 用于组织管理和军事等领域

3.3 算法的设计

[案例一]世界杯足球赛的赛事安排

世界杯足球赛报名参赛的球队达200多支球队，需从中选出成绩最好的32支球队，难度很大，成本也高。

通过分区预选赛选出成绩最好的32支球队进入决赛圈，这种做法也包含分治思想并降低了难度和复杂度。

3.3 算法的设计

[案例二]二分查找

常用的二分查找是一个典型的分治算法。

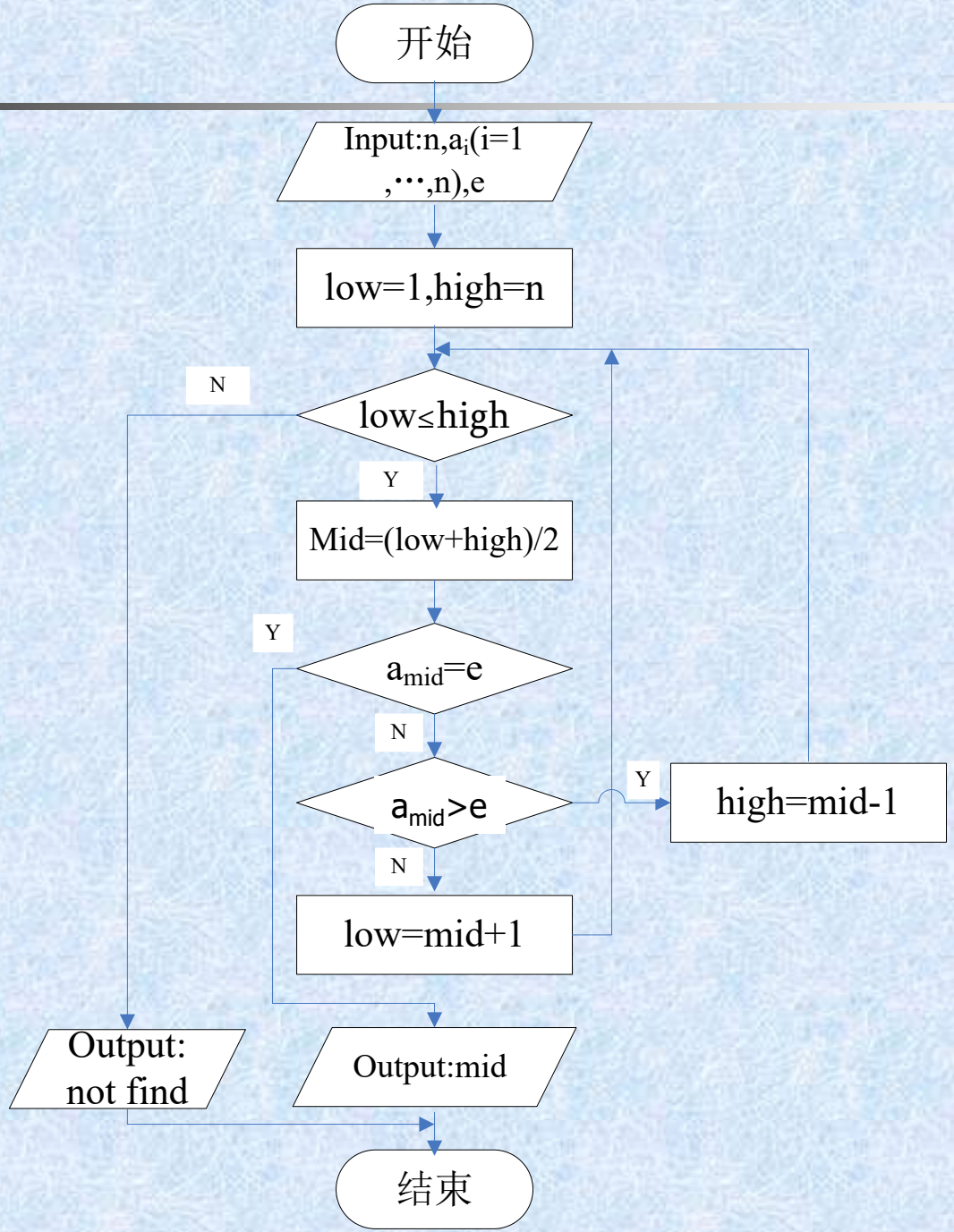
二分查找基本原理：用于在n个元素的有序序列中查找指定元素e。将n个元素分成个数大致相同的两半，取$a_{n/2}$与欲查找的e作比较，

若$e=a_{n/2}$，则找到e，算法终止

若$e<a_{n/2}$，则只需在数组a的前半部分继续二分查找e

若$x>a_{n/2}$，则只需在数组a的后半部分继续二分查找e

二分查找每次比较将数据减少一半，也称折半查找。


开始
Input:n,ai(i=1,…,n),e
low=1,high=n
low≤high
N
Y
Mid=(low+high)/2
amid=e
Y
N
amid>e
Y
high=mid-1
N
low=mid+1
Output: not find
Output:mid
结束

3.3 算法的设计

二分查找在[illegible]的计算过程

low=1,high=15；

mid=(low+high)/2=8,

a_8="Harry", e="John"

a_8<e,

low=mid+1=9,

high=15；

mid=(low+high)/2=12,

a_{12}="Larry", e="John"

a_{12}>e,

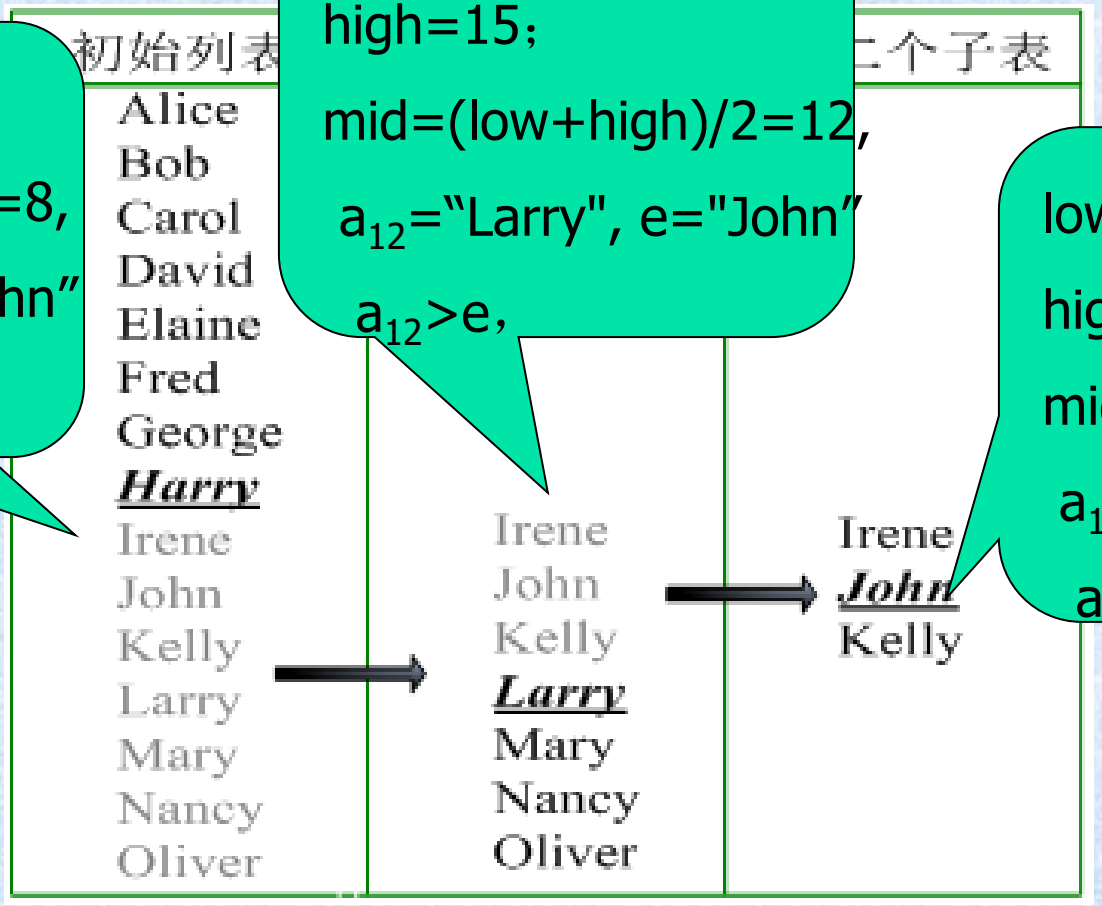
low=9,

high=mid-1=11；

mid=(low+high)/2=10,

a_{10}="John", e="John"

a_{10}=e,



3.3 算法的设计

分治策略是解决工作、学习和生活中常见问题的一种思维方法，它在组织管理和军事领域得到广泛的应用

例如：某大企业的销售公司，由于其许多产品优质而非常畅销，总部会到各地建立分支机构，这其中就蕴涵着分治思想。

又如：各种大型体育赛事通常分为初赛和决赛，世界杯足球赛要从报名参赛的200多支球队中选出成绩最好的32支球队，难度很大，成本也高。因此通过分区预选赛选出成绩最好的32支球队进入决赛圈，这种做法也包含分治思想并降低了难度和复杂度。

再如：中国革命战争时期经常遇到敌军强大，因此采用集中优势兵力，逐个击破的分治策略往往能产生以弱胜强的优异战果

3.3 算法的设计

五、贪心法

1.问题的提出

假设有3种硬币，它们的面值分别是1元、5角、1角。现在有一个小孩买了价值6元2角的东西，并给售货员10元钱。当售货员找给小孩零钱时，希望她找给小孩的硬币数目最少。

3	3	2	2	2	2	
1	0	3	2	1	0	----
3	8	3	8	13	18	

这种简单地从具有最大面值的币种开始，按递减的顺序考虑各种币种的方法称为贪心法，或启发式搜索法。

3.3 算法的设计

2.贪心算法

贪心法的基本思想：将待求解的问题分解成若干个子问题进行分步求解，且每一步总是做出当前最好的选择，即得到局部最优解，再将各个局部最优解整合成问题的解。

贪心法体现了一种快刀斩乱麻的思想，以当前和局部利益最大化为导向的问题求解策略。

利用贪心法求解问题的过程：

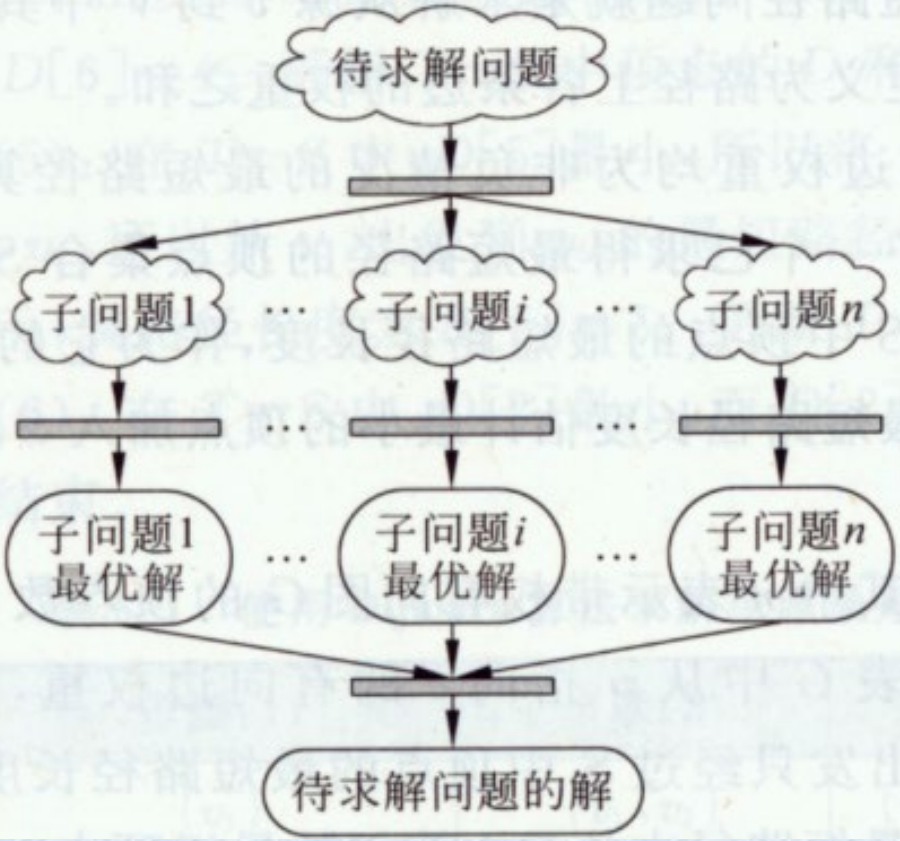
①分解：将原问题分解为若干个相互独立的阶段；

②解决：对每个阶段求局部的最优解，即进行贪心选择

③合并：把各个阶段的解合并为原问题的一个可行解。

3.3 算法的设计

利用贪心法对问题进行求解的过程



(1) 将待求解问题细分成若干个子问题

(2) 分别对每一子问题进行求解，得到子问题的局部最优解。

(3) 把子问题的局部最优解整合成待求解问题的解。

3.3 算法的设计

3.贪心法的应用

[案例一]田忌赛马

战国时期，齐威王与大将田忌赛马，齐威王和田忌各有三匹好马：上马、中马与下马。比赛分三次进行，每次赛马以千金作赌。由于两者的马力相差无几，而齐威王的马分别比田忌相应等级的马要好，所以大家都认为田忌必输无疑。

田忌采纳了门客孙膑的意见，用下马对齐威王的上马，用上马对齐威王的中马，用中马对齐威王的下马，结果田忌以2比1胜齐威王而得千金。

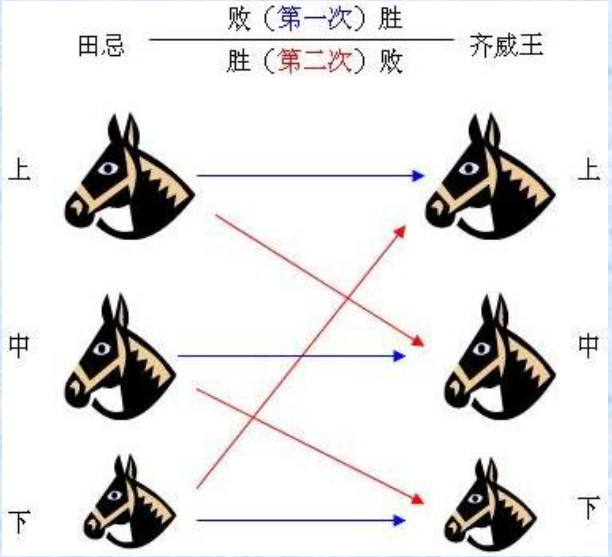
3.3 算法的设计

将齐王的马、田忌的马均按上、中、下马顺序排列，齐王依次出马，孙膑的贪心选择策略：

①若剩下的最强的马都赢不了齐王剩下的最强的马，选择用最差的一匹马对阵齐王最强的马；

②若剩下的最强的马可以赢齐王剩下的最强的马，选择用这匹马去赢齐王剩下的最强的马；

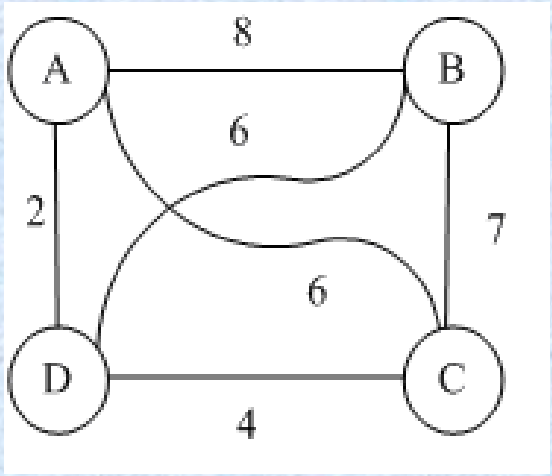
③若剩下的最强的马和齐王剩下的最强的马打平的话，可以选择打平或者用最差的马输掉比赛。



3.3 算法的设计

[案例二]电缆铺设

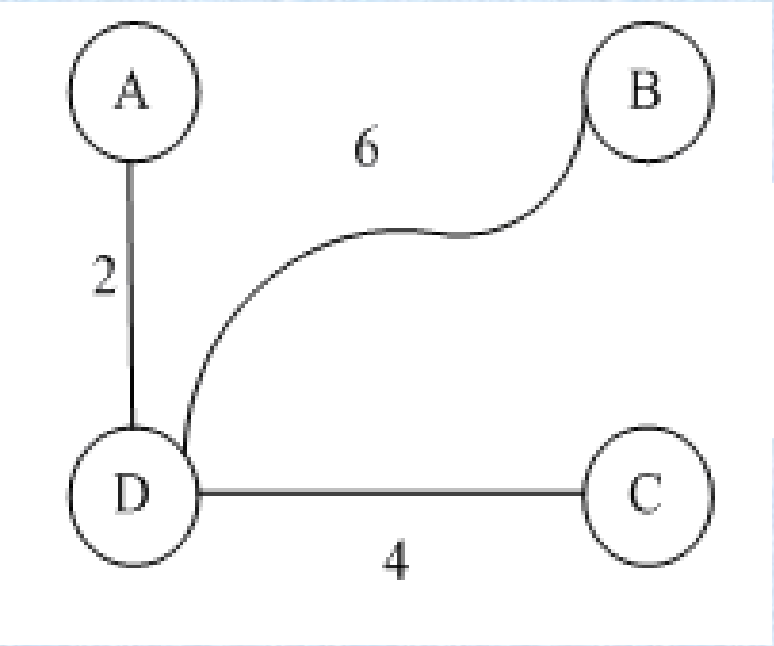
假设要在n个城市之间铺设光缆,铺设光缆费用很高，且各个城市之间铺设光缆的费用不同，问如何铺设,使得n个城市的任意两个之间都可以通信，且使铺设光缆的总费用最低?



- 可用图论中的最小生成树求解
- 求解最小生成树算法是贪心法

3.3 算法的设计

利用贪心法求解最小生成树，其中一种贪心选择策略是：贪心选择权值最小的边，若与之前加入的边构成回路，则放弃；否则，加入最小生成树。



电缆铺设的最小生成树

3.3 算法的设计

[案例三]数字删除

给定n位正整数a，去掉其中任意k≤n个数字后，剩下的数字按原来次序排列组成一个新的最小正整数。

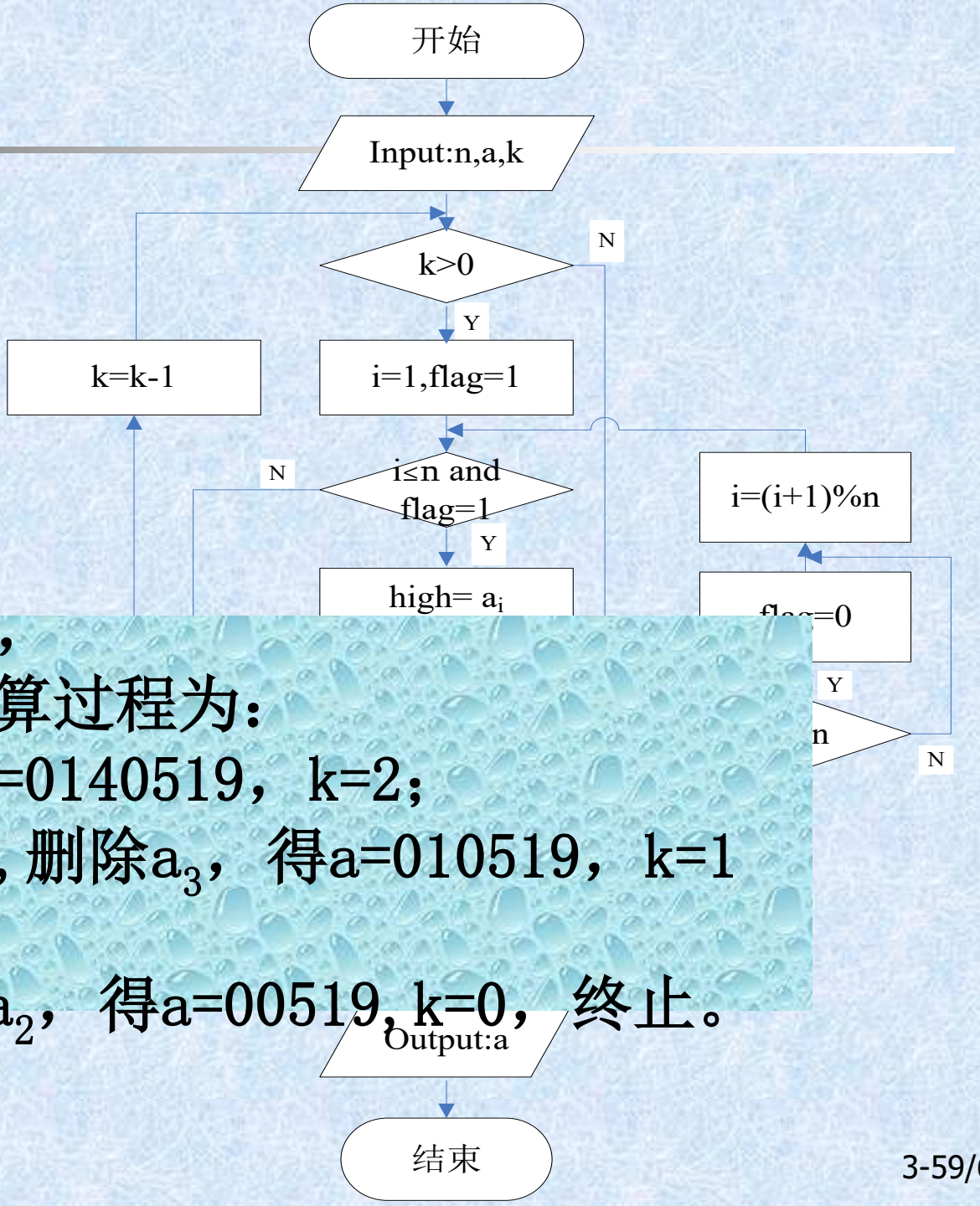
例如a=20140519,k=3,删除第一位2、第三位1、第四位4后得到的新数最小，为519。

3.3 算法的设计

[解] 令n位正整数$a=a_1a_2...a_{i-1}a_i...a_n$。显然，要想删除1个数字后的新数小于原数字，应依次比较相邻两位，贪心选择第一个满足$a_i>a_{i+1}$的数字a_i，删除a_i。对第a_n，应比较a_n和a_1。重复删除k个数字，即得到问题的解。

若遍历a的n位，均未找到满足$a_i>a_{(i+1)\%n}$的数字a_i，则意味着a的每一位均相等，默认删除a_1。

流程图



以a=20140519, k=3为例，

贪心求解删除问题的计算过程为：

（1）$a_1>a_2$, 删除a_1, 得a=0140519， k=2；

（2）$a_1<a_2$, $a_2<a_3$, $a_3>a_4$, 删除a_3，得a=010519，k=1；

（3）$a_1<a_2$, $a_2>a_3$, 删除a_2，得a=00519, k=0，终止。

思考

如果：

a=123456789

k=3

则应该删除哪几个数？得到的数是多少？

3.3 算法的设计

贪心算法是最接近于人类日常思维的一种问题求解方法，它已在人类工作和生活的各个领域得到广泛的应用。

例如：公司招聘新员工是从一批应聘者中招收最能干的人。

再如：学校招生是从众多报考者中招收一批最好的学生。

这种按照某种标准挑选最接近该标准的人或物的做法就是贪心算法。

3.3 算法的设计

六、动态规划

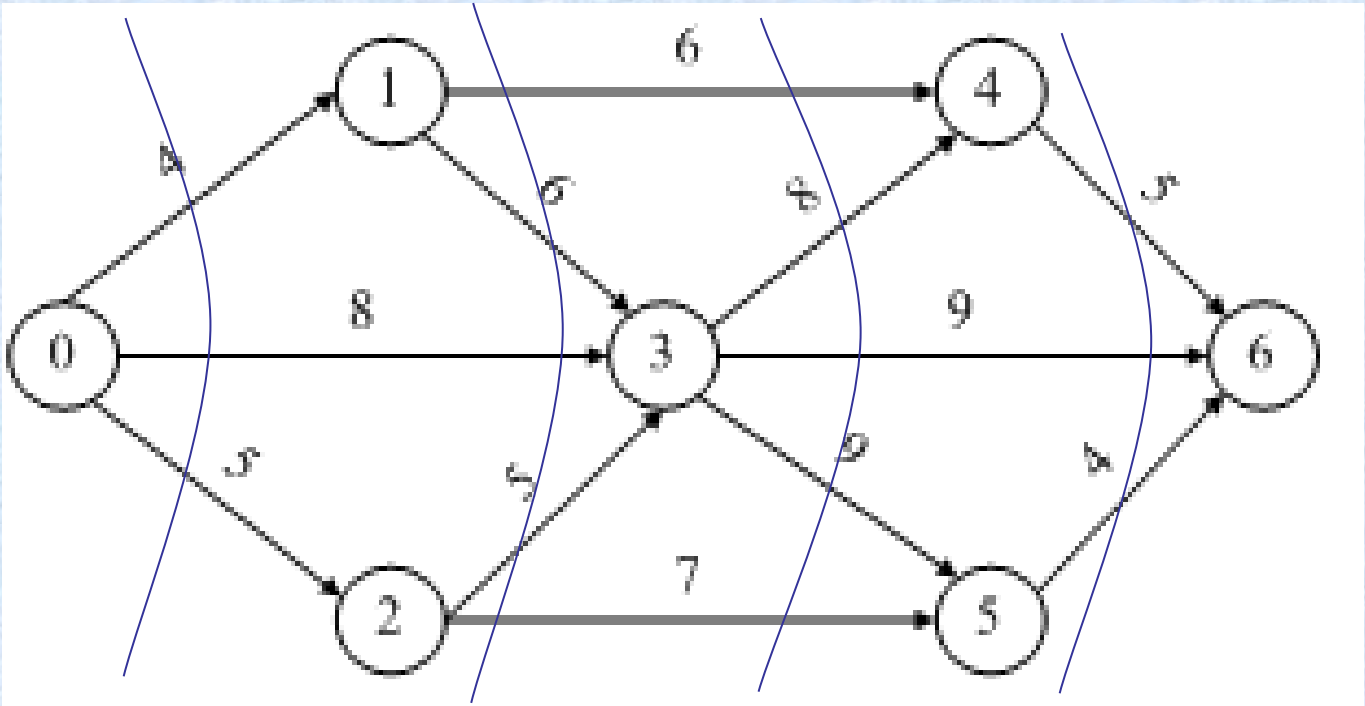
1.问题的提出

动态规划是解决多阶段决策最优化问题的一种方法。

[案例一]GPS中的最优路径。全球定位系统GPS(Global Positioning System)可以为我们计算出满足各种不同要求的、从出发地到目的地最优路径，可能是花费时间最短，也可能是过路费最少。GPS寻找最优路径的算法就是动态规划算法。

3.3 算法的设计

假设计算下图中顶点0到顶点6的最短路径。



3.3 算法的设计

定义cost[i]：从顶点0到顶点i的最短路径。

第0阶段：cost[0]=0

第1阶段：cost[1]=cost[0]+4=4

cost[2]=cost[0]+5=5

第2阶段：cost[3]=min{cost[0]+8,cost[1]+4,cost[2]+5}=8

第3阶段：cost[4]=min{cost[1]+6,cost[3]+8}=10

cost[5]=min{cost[2]+7,cost[3]+9}=12

第4阶段：cost[6]=min{cost[4]+5,cost[3]+9,cost[5]+4}=15

根据计算，从顶点0到顶点6的最短路径值为15。

从顶点6向前回溯，最短路径为0→1→4→6。

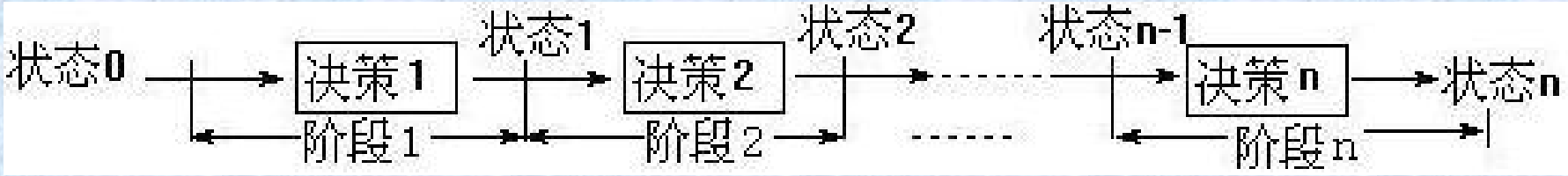
3.3 算法的设计

2.动态规划算法

动态规划是美国数学家R.Bellman等人于1951年在研究多阶段决策过程的优化问题时创立的一种解决问题的新方法。

在现实生活中，有一类问题可以将其活动过程分解成若干个相互联系的阶段，在它的每一阶段都需要作出决策，从而使整个过程达到最好的活动效果。这种将一个问题看作是一个前后相互关联且具有链状结构的多阶段过程称为多阶段决策过程

将解决多阶段决策的最优化的过程称为动态规划算法。



3.3 算法的设计

动态规划法主要适用于最优化问题的求解：这类问题会有多种可能的解，每个解都有一个值，而动态规划找出其中最优（最大或最小）值的解。若存在若干个最优值的解的话，它只取其中的一个。

动态规划问题求解的基本思想：将待求解的问题分解为若干个互相联系的子问题，然后按自底向上的顺序推导出原问题的解。通过存储子问题的解，可以避免在求解过程中重复多次求解同一个子问题，从而可以提高该算法的求解效率。动态规划算法实质是分治思想和冗余解决方法的结合。

3.3 算法的设计

3.动态规划的应用

[案例二]Fibonacci数列。

F[1]=1,F[2]=1,F[i]=F[i-1]+F[i-2],计算F[n](n≥3)

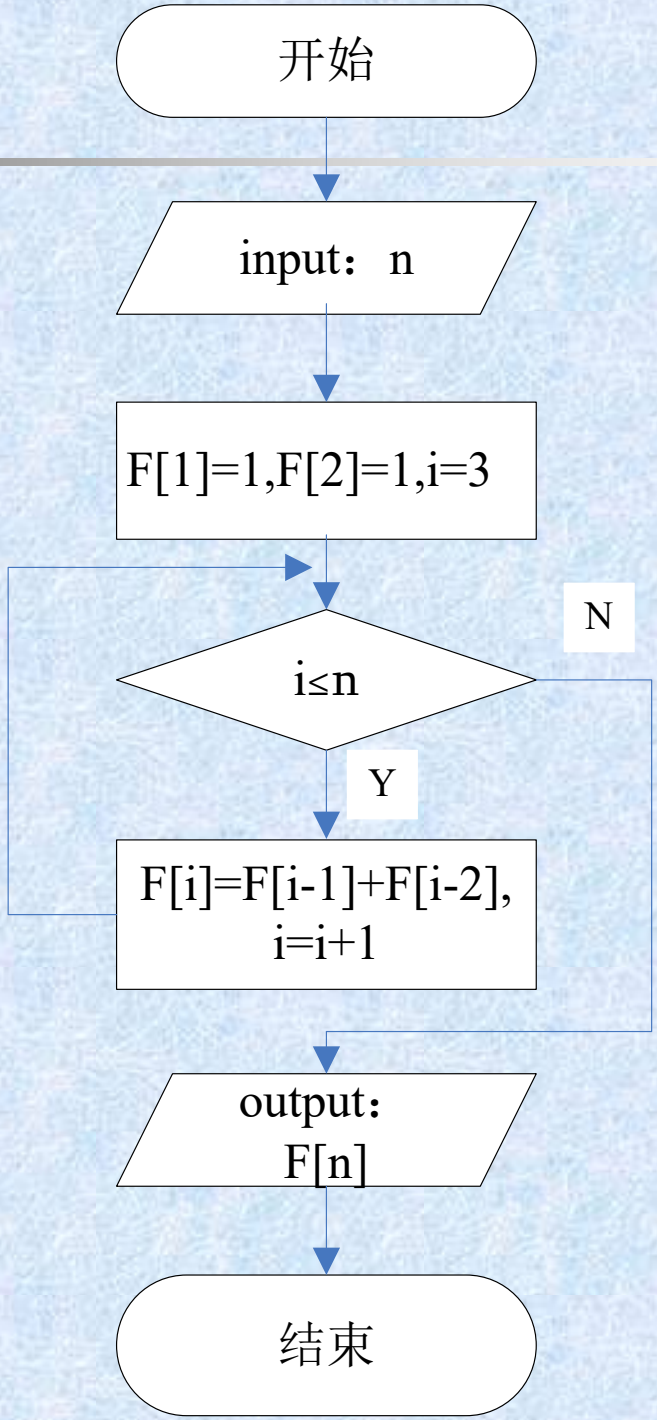
动态规划求Fibonacci数列的伪代码描述如下：

输入：正整数n

输出：Fibonacci数列的第n项

```
Fib(n)
1    F[1]←F[2]←1
2    FOR i=3 to n
3       DO F[i]←F[i-1]+F[i-2]
4    RETURN F[n]
```

流程图



3.3 算法的设计

动态规划已在经济管理、生产调度、工程技术和最优控制等方面得到了广泛的应用，最短路线、库存管理、资源分配、设备更新、排序和装载等问题运用动态规划算法求解比较方便

例如：将动态规划方法运用于经济学领域的最优投资与消费选择策略的求解，可以得到连续时间下两类资产的最优投资与消费问题的解决方案。

再如：动态规划也适用于人生规划，它是人类智慧的体现

千里之行，始于足下，任何一项伟大事业的完成总是从小事做起的，小目标的达成是实现大目标的基础。

3.4 算法的评价和分析

实际应用中，一个问题常常有多种解法。如Fibonacci数列，有递归和动态规划两种求解方法。如何选择算法？哪个算法是最优的？这涉及算法的评价和分析。

一、算法的正确性

二、算法的复杂性

3.4 算法的评价和分析

一、算法的正确性

一个正确的算法是对每一个输入数据产生对应的正确结果并且终止。而错误的算法对于某些输入数据要么不会终止，要么在终止前给出的不是预期的正确结果。

算法确认：设计出算法后，证明该算法对所有可能的合法输入都能计算出正确结果的工作过程。

程序证明：用算法语言描述构成的程序在计算机上运行，也应证明该程序是正确的。

算法确认和程序证明的研究难度很大(D.Gries,吴文俊)。

3.4 算法的评价和分析

二、算法的复杂性

算法的复杂度衡量一个算法的效率。对所设计的算法，通常关心的是算法的执行时间和算法所需要的存储空间。即：

1. **算法的时间复杂度**
2. **算法的空间复杂度**

3.4 算法的评价和分析

算法的时间复杂性

事后统计法：计算算法开始时间与完成时间差值

缺点：1．必须执行程序

2．其它因素掩盖算法本质

事前统计

n，是常

和算法执行**时间**相关的**因素**：

1．**算法选用的策略**

2．**问题的规模**

3．编写程序的**语言**

4．**编译**程序产生的机器代码**的质量**

5．**计算机**执行指令**的速度**

3.4 算法的评价和分析

算法的时间复杂性

算法的时间复杂度：度量算法的运行时间，记做T(n)，它是输入数据量n的函数，它忽略了运行时间的低阶项和高阶项系数，只关心运行时间的高阶项。

多项式时间复杂度：1、$\log_2 n$、n、$n\log_2 n$、n^2、n^3

指数时间复杂度：2^n、n!

伟大的成就：

Fibonacci数列动态规划算法，它是多项式时间复杂度n：用普通台式机，在一秒钟内就会计算出F_{200}

[illegible]数列，也只能多计算7个数

3.4 算法的评价和分析

算法的空间复杂性

算法的空间复杂度：为解问题实例而需要的存储空间。

算法的存储量包括：

对比Fibonacci数列的递归实现和动态规划实现。
前者指数时间复杂度，运行效率低；
后者多项式时间复杂度，运行效率高。
分析算法可以看出，动态规划实现较递归实现增加了一个数组空间，用于记录已得到的数列值，从而避免重复计算，节省了计算时间。它以增加空间开销换取了时间效率的提高。

Fibonacci数列递归和动态规划算法

	递归算法	动态规划
时间复杂度	指数	多项式
空间复杂度	?	多项式

本章小结

- **算法的概念**：起源，定义，特征
- **算法的描述**：目的，方法(四种)
  自然语言、流程图、伪代码、程序设计语言
- **算法的设计**(一般步骤，分类)
  穷举法：暴力破解法、四色定理、百钱买百鸡
  回溯法：老鼠走迷宫、网络爬虫、八皇后问题
  递归法：欧几里得、德罗斯特效应、Fibonacci
  分治法：国王的婚姻、MapReduce、二分查找
  贪心法：货币支付、田忌赛马、电缆铺设
  动态规划：GPS最优路径、Fibonacci数列
- **算法的评价和分析**
  算法的正确性，算法的复杂性(时间、空间)