

程序设计方法与实践

本课注意事项

- 培养基本编程能力。
 - 多练多写，养成编程习惯，去解决实际问题
 - 培养严谨的逻辑思维。
 - 多注意问题的边界条件，写无Bug的程序
- 8周课程，16次课，32课时
 - 从9月15到11月6日，每周1和周4。
 - 节假日占用2次课，有效课程28课时。
 - 课上偶尔点名，不到会扣平时分。
 - 有事可以请假。提前告诉我就行。

本课注意事项

- 参考书：算法设计与分析基础（第3版）
 - 绪论
 - 算法效率分析基础
 - 蛮力法
 - 减治法
 - 分治法
 - 变治法
 - 时空权衡
 - 动态规划
 - 贪婪技术



本课注意事项

- 学习途径：
 - 课堂听课
 - 课堂同学讲题
 - 鼓励上台讲自己的作业思路、走过的弯路和经验分享。
 - 每次上台1分，记入平时分
 - 乐学：程序设计方法实践-2025
 - 乐学论坛交流（有质量的发言1分，记入平时分）
 - 每周作业答题（作业分）
 - i北理的课程群
 - 4302-程序设计方法实践
 - 助教：呼延泽、林柏健

本课注意事项

- 成绩分配：总100分
 - 同学课上上台讲题+乐学论坛发言 10分
 - 每周4个编程题目占5分，8周共40分
 - 每周三下午开放。可以开始提交作业。
 - 次周周日24:00折扣——约11天，之后提交分数打8折。
 - 再次周周日24:00关闭——约18天，无法提交作业。
 - 乐学自动判分，按通过的测试用例个数给分。有提交次数限制
 - 最后闭卷考试50分

本课注意事项

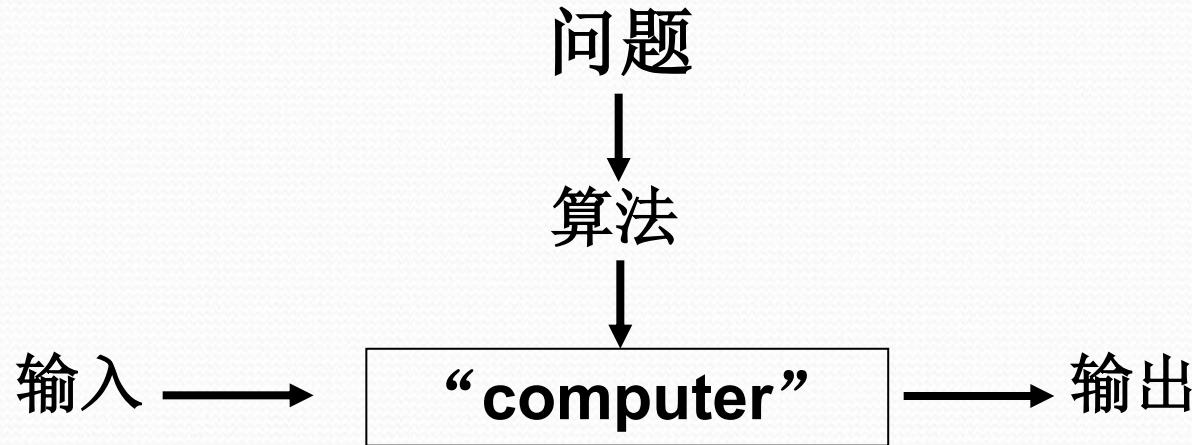
- 写程序要求
 - 运行效率:
 - 重点关注循环体内的代码效率
 - 可读性:
 - 一个函数不能太长（可截断为多个函数）
 - 不要出现重复代码（使用函数来共享重复代码）
 - 代码加注释（1、函数功能的注释；2、关键变量的注释；3、函数内关键位置的注释）

第一章 緒論

- 什么是算法
- 算法问题求解基础
- 重要的问题类型
- 基本数据结构

什么是算法

- 算法是一系列解决问题的明确指令。对于符合一定规范的输入，能够在有限的时间内获得要求的输出。



- 1、算法每个步骤没有歧义
- 2、要明确输入的值域
- 3、同一算法可用不同表述形式。（文字、伪代码、流程图...）
- 4、同一问题可有不同算法解决，效率可能不同

什么是算法

- 举例：计算两个不全为0的非负整数m,n的最大公约数(Greatest Common Divisor)。
 - 1、欧几里得算法(辗转相除法)

$\text{gcd}(m,n)=\text{gcd}(n,m \bmod n)$ 直到 $m \bmod n$ 得 0

$\text{gcd}(60,24)=\text{gcd}(24,12)=\text{gcd}(12,0)=12$

文字描述：

- 1、如果 $n=0$,返回 m 的值为结果，同时过程结束；否则进入2
- 2、 m 除以 n ，将余数赋给 r 。
- 3、将 n 的值赋给 m ，将 r 的值赋给 n ，返回1

什么是算法

- 举例：计算两个不全为0的非负整数m,n的最大公约数(Greatest Common Divisor)。

- 1、欧几里得算法(辗转相除法)

$\text{gcd}(m,n)=\text{gcd}(n,m \bmod n)$ 直到 $m \bmod n$ 得 0

$\text{gcd}(60,24)=\text{gcd}(24,12)=\text{gcd}(12,0)=12$

伪代码描述：

```
算法 Euclid(m, n) //参考教材P3
    //使用欧几里得算法计算gcd(m, n)
    //输入：两个不全为0的非负整数m, n
    //输出：m, n的最大公约数
    while n ≠ 0 do
        r ← m mod n
        m ← n
        n ← r
    return m
```

什么是算法

- 举例：计算两个不全为0的非负整数m,n的最大公约数(Greatest Common Divisor)。

- 2、 $\text{gcd}(m,n)$ 连续整数检测算法

文字描述：

- 1、将 $\min\{m,n\}$ 的值赋给t。
- 2、m除以t。如果余数为0，进入3；否则进入4。
- 3、n除以t。如果余数为0，返回t为结果；否则进入4。
- 4、t的值减1，返回2

- $m=60, n=24$ 计算

- 1) $t=24; \rightarrow 2) 60 \bmod 24 = 12; \rightarrow 4)$ $t=23; \rightarrow 2) 60 \bmod 23 = 14; \rightarrow 4-2$ 循环
- 4) $t=20; \rightarrow 2) 60 \bmod 20 = 0; \rightarrow 3) 24 \bmod 20 = 4 \rightarrow 4)$ $t=19; \rightarrow 4-2$ 循环
- 4) $t=12; \rightarrow 2) 60 \bmod 12 = 0; \rightarrow 3) 24 \bmod 12 = 0$ return 12

什么是算法

- 举例：计算两个不全为0的非负整数m,n的最大公约数(Greatest Common Divisor)。

- 3、 $\text{gcd}(m,n)$ 公因数相乘法

文字描述：

- 1、找到m的所有质因数。
- 2、找到n的所有质因数。
- 3、从1和2中的所有质因数找到公因数。
- 4、从3中找到的公因数相乘，结果为m,n的最大公约数。

- $m=60, n=24$ 计算

$$60 = 2 \times 2 \times 3 \times 5; \quad 24 = 2 \times 2 \times 2 \times 3$$

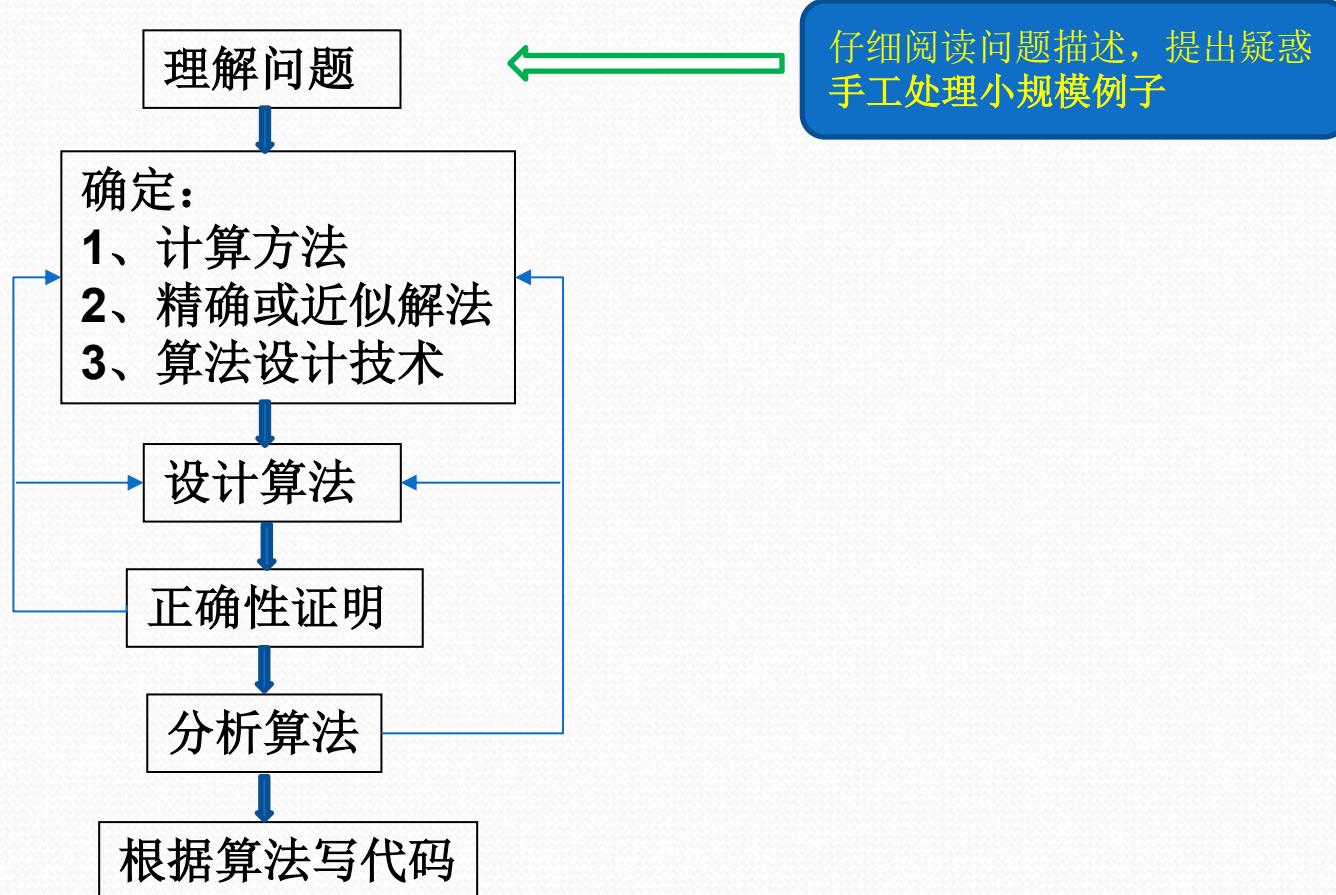
$$\text{gcd}(60, 24) = 2 \times 2 \times 3 = 12$$

第一章 緒論

- 什么是算法
- 算法问题求解基础
- 重要的问题类型
- 基本数据结构

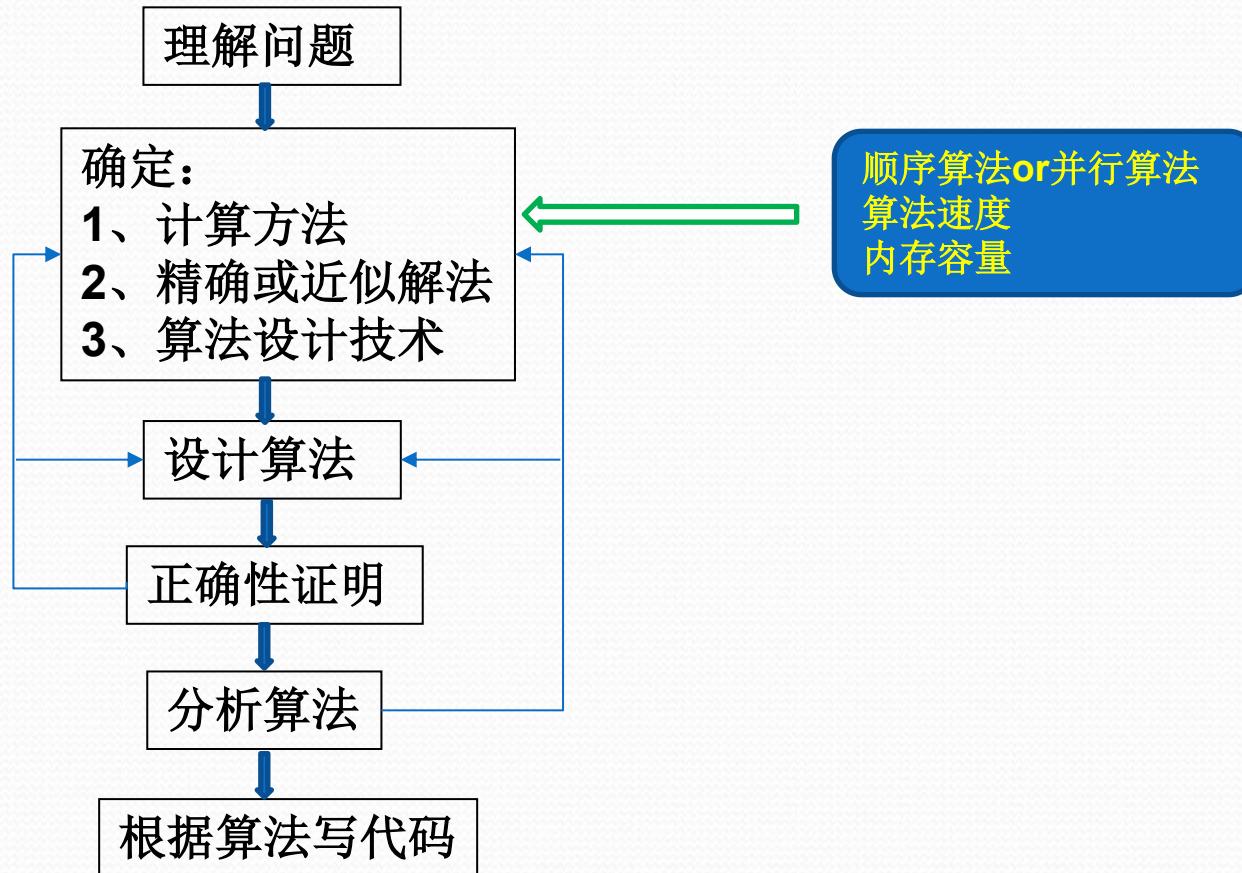
算法问题求解基础

- 算法是问题的程序化解决方案



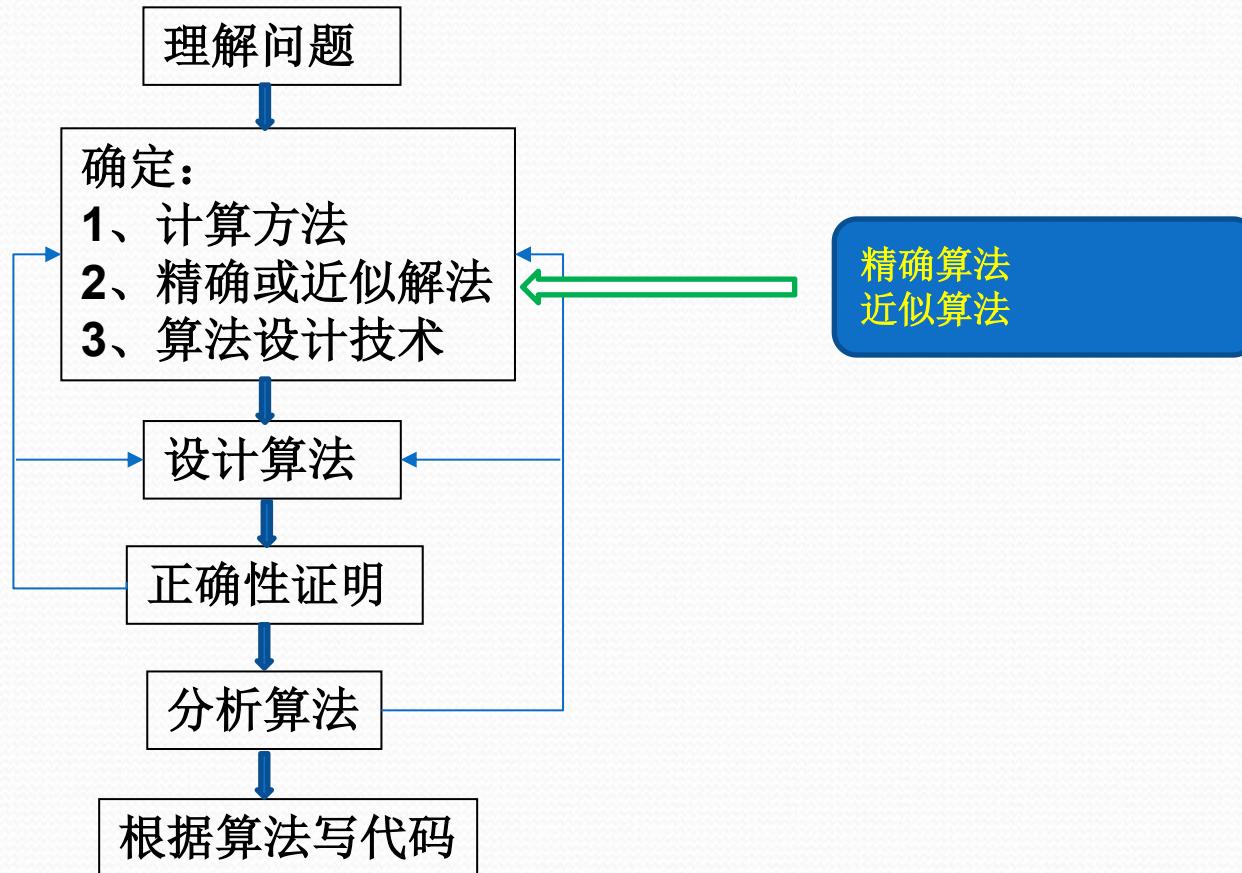
算法问题求解基础

- 算法是问题的程序化解决方案



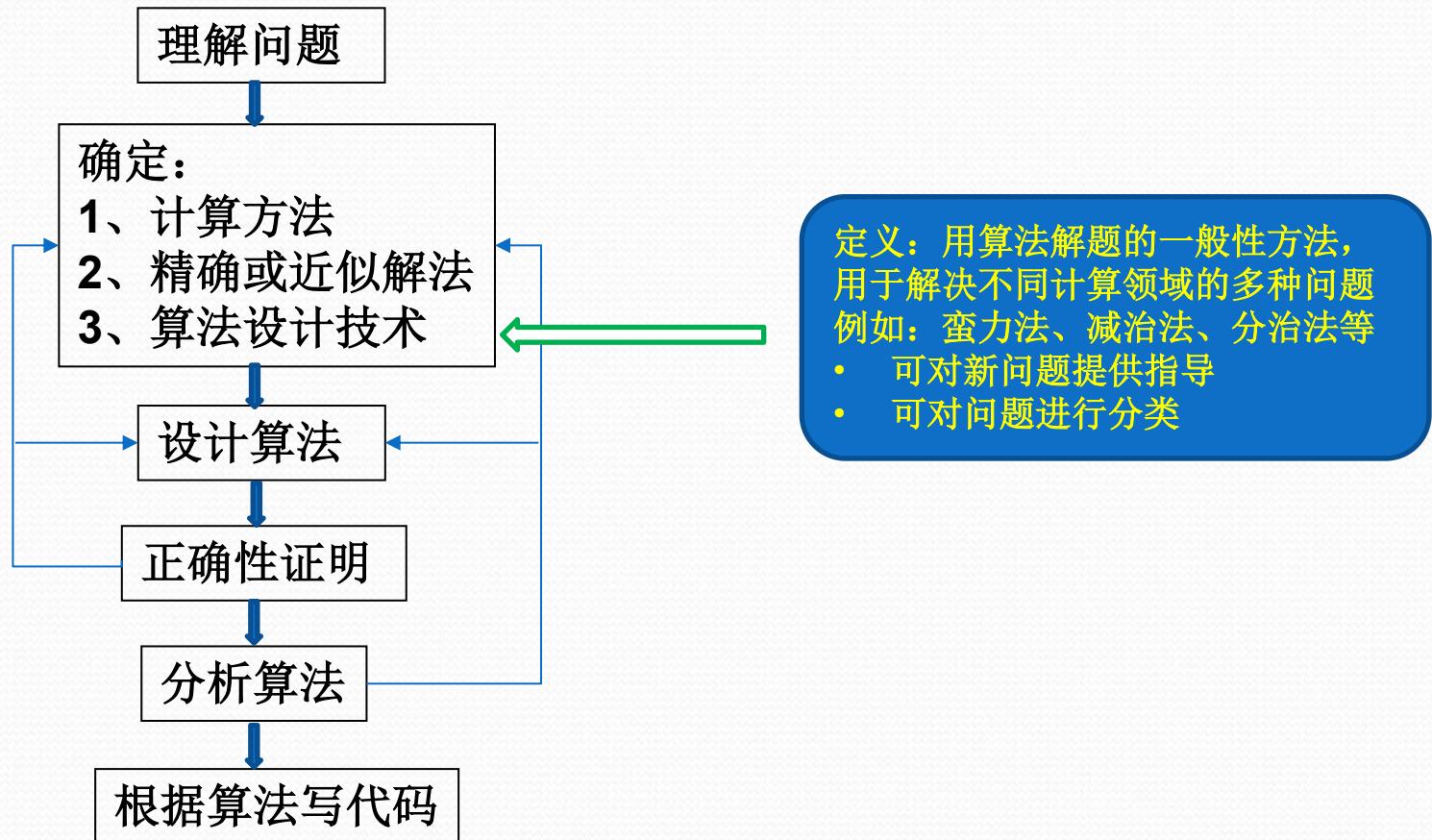
算法问题求解基础

- 算法是问题的程序化解决方案



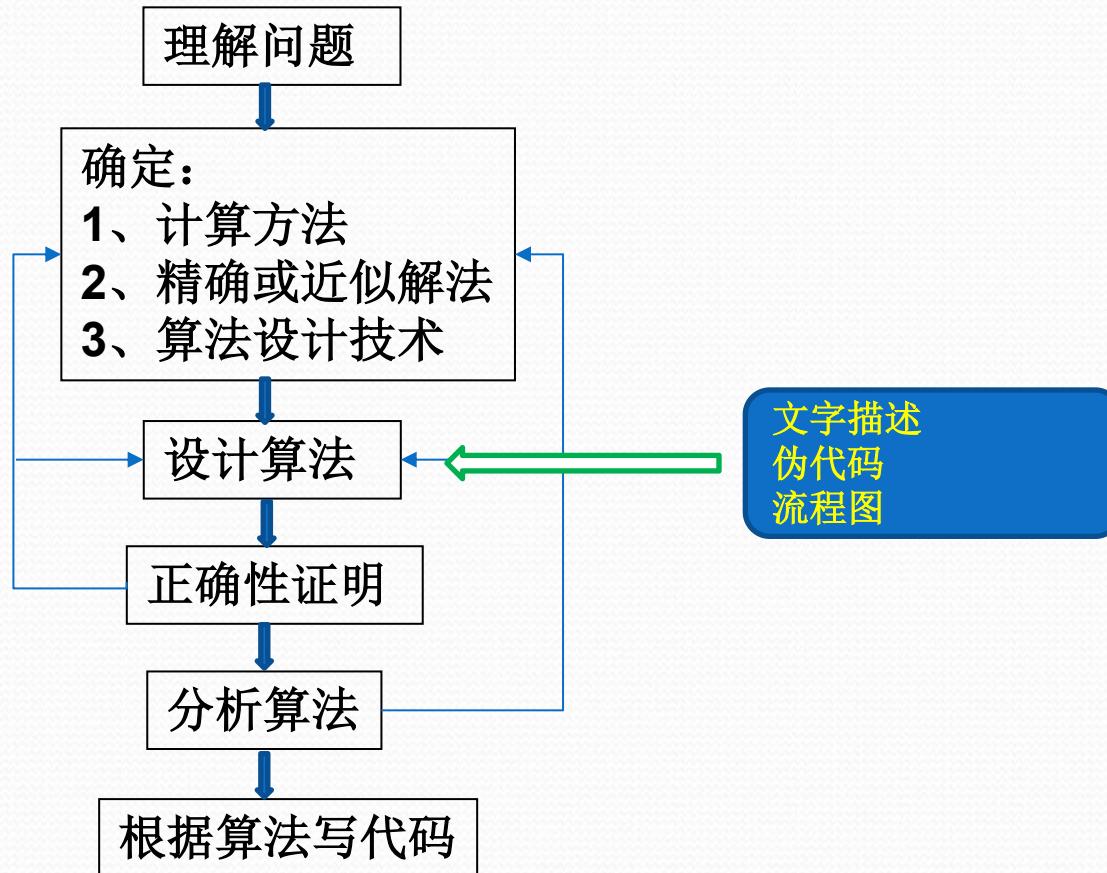
算法问题求解基础

- 算法是问题的程序化解决方案



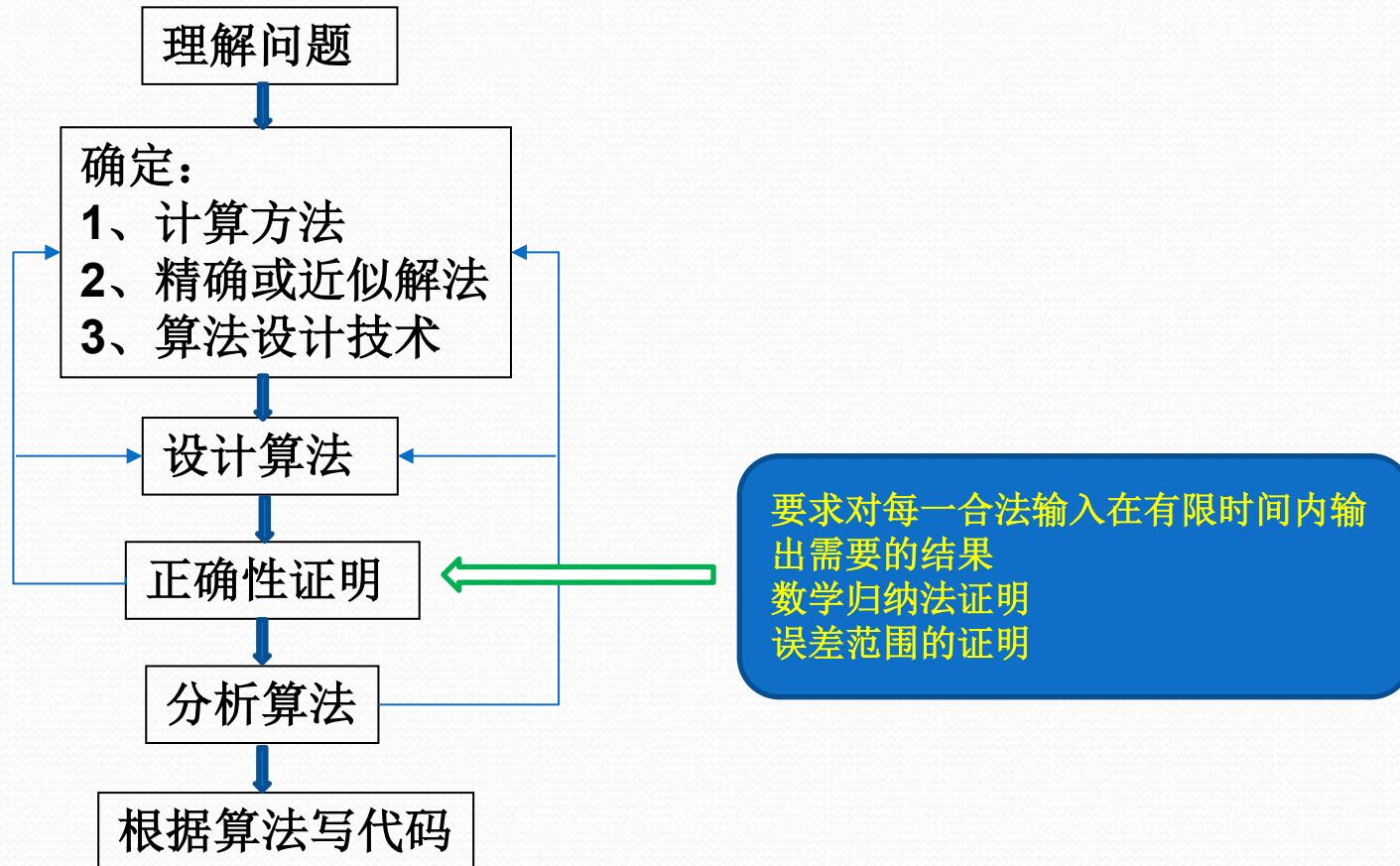
算法问题求解基础

- 算法是问题的程序化解决方案



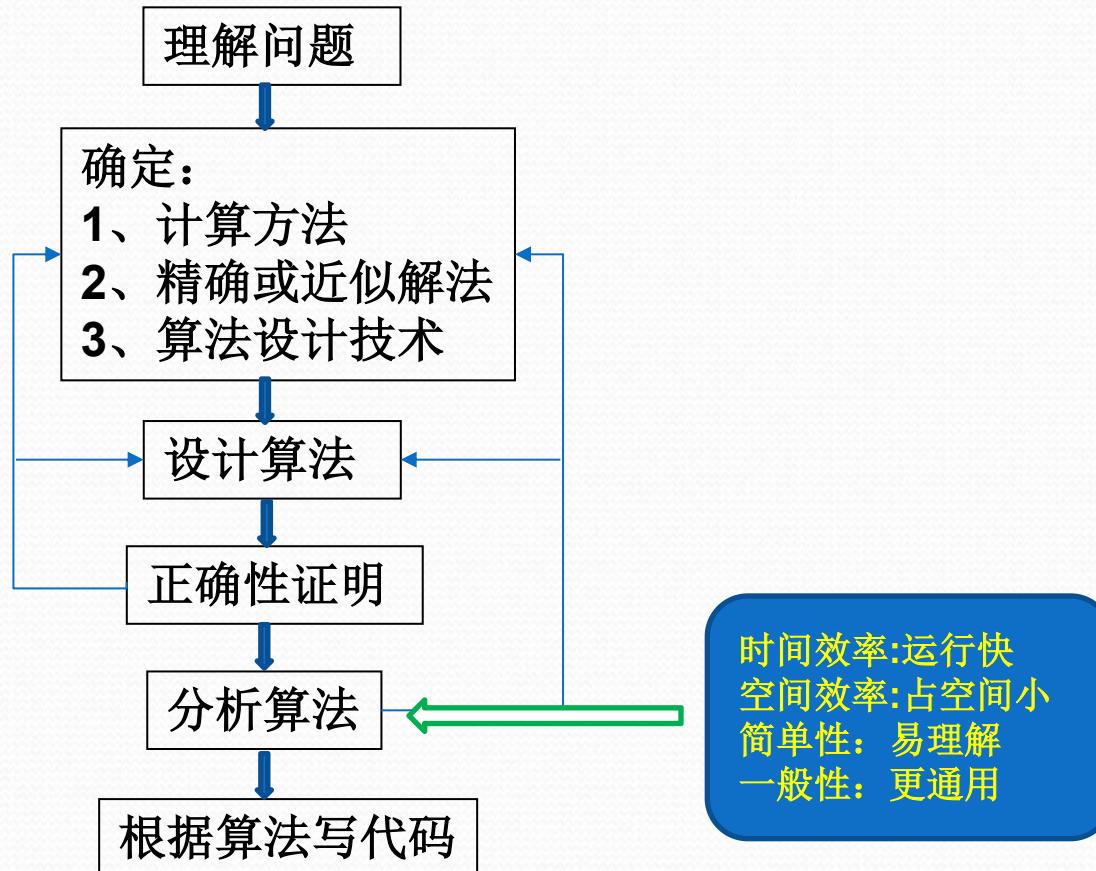
算法问题求解基础

- 算法是问题的程序化解决方案



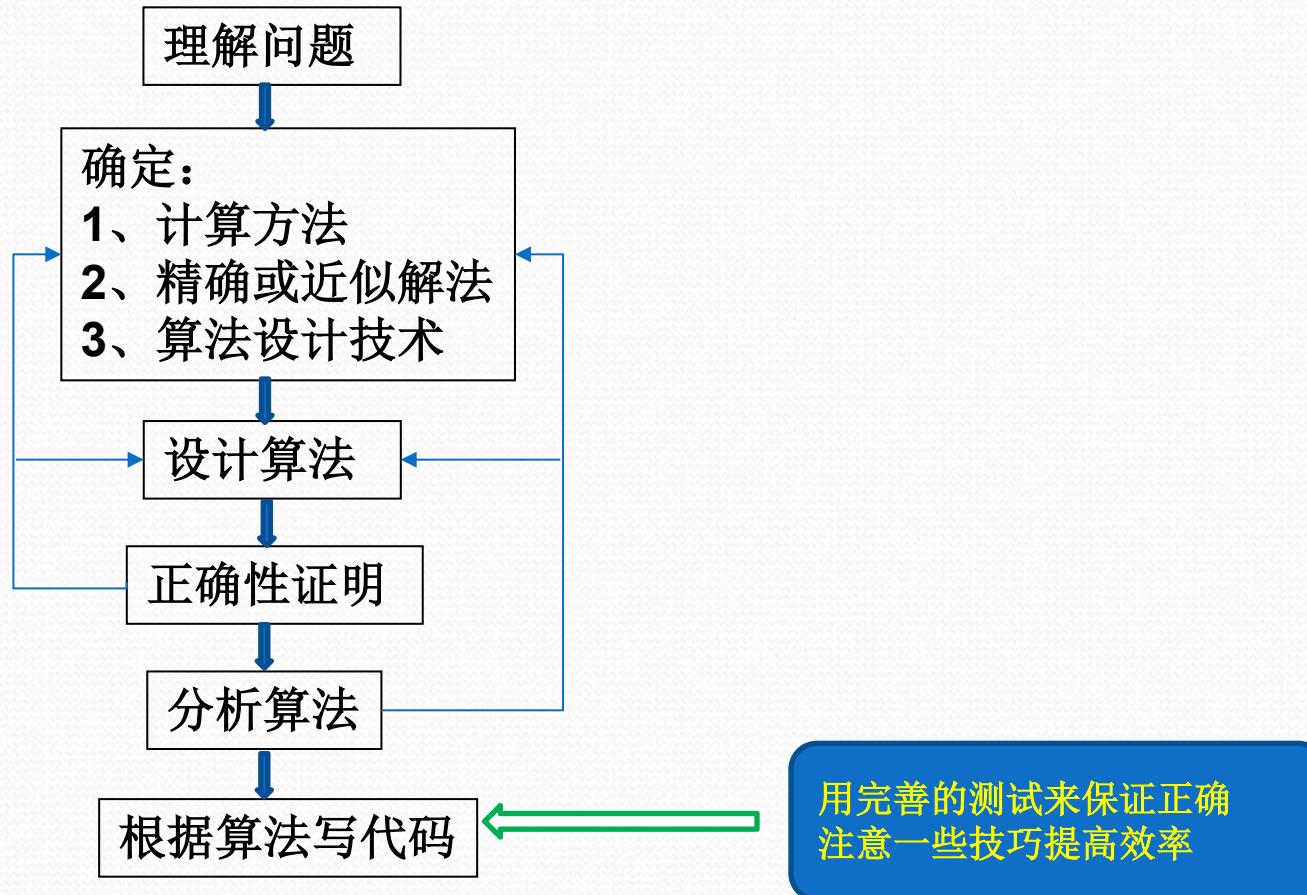
算法问题求解基础

- 算法是问题的程序化解决方案



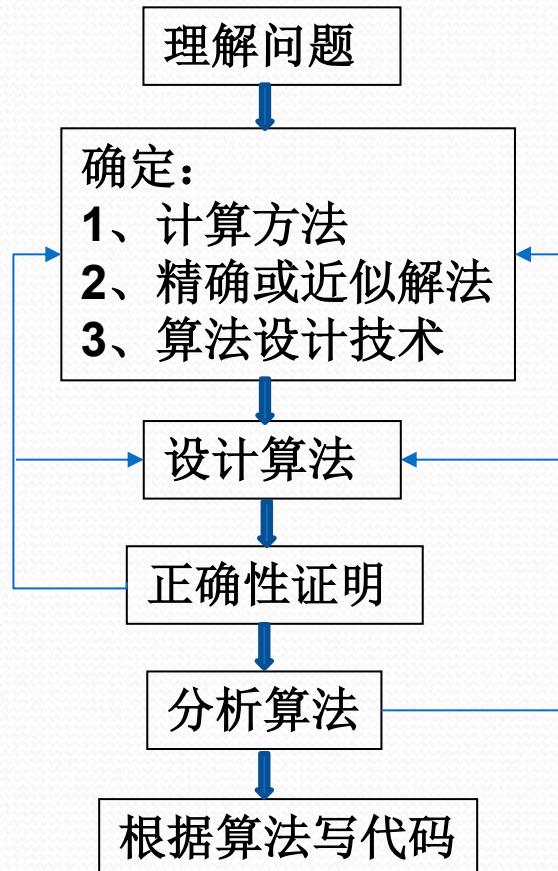
算法问题求解基础

- 算法是问题的程序化解决方案



算法问题求解基础

- 算法是问题的程序化解决方案



好的算法是不懈努力和反复修正的结果

第一章 緒論

- 什么是算法
- 算法问题求解基础
- 重要的问题类型
- 基本数据结构

重要的问题类型

- 排序
- 查找
- 字符串处理
- 图问题
- 组合问题
- 几何问题
- 数值问题

重要的问题类型

- 排序

按照升序或降序重新排列给定列表中的数据项

排序原因：

- 题目的输出要求。例如对学生成绩排序输出。
- 有利于问题的求解。例如查找学生姓名，学号。

排序算法特点：

- 执行速度。对不同输入数据，排序算法速度也有不同。
- 内存占用
- 稳定性

没有一种算法在任何情况下都是最优的。

重要的问题类型

- **查找**

在给定的集合找一个给定的值（查找键）

- 顺序查找
- 折半查找
- 查找树
- 哈希查找

考虑查找效率和构建结构效率的平衡。

重要的问题类型

- 字符串处理

字符串：字母表中符号所构成的序列

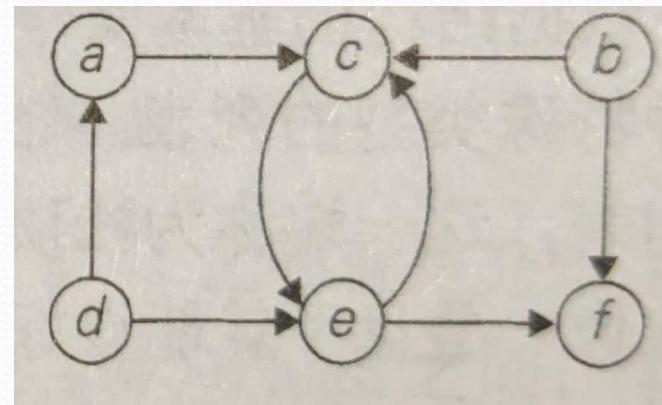
字符串匹配算法。

重要的问题类型

• 图问题

由称为顶点的点构成的集合，其中某些顶点由称为边的线段相连。 $G = \langle V, E \rangle$

- 图的遍历算法
- 最短路径算法
- 图拓扑排序算法



重要的问题类型

• 组合问题

寻找一个组合对象（排列/组合/子集），这些对象能够满足特定的条件并具有我们需要的特征，如价值最大化或成本最小化。

- 旅行商问题
 - 图填色问题
-
- ◆ 这类问题复杂性高，若规模大难以计算。
 - ◆ 没有好的通用算法。

重要的问题类型

- **数值问题**

处理具有连续性的数学问题。

- 解方程和方程组
- 计算定积分
- 计算函数值

大部分需要求近似解

第一章 緒論

- 什么是算法
- 算法问题求解基础
- 重要的问题类型
- 基本数据结构

基本数据结构

数据结构：对相关的**数据项进行组织**的特殊架构

- 线性数据结构
 - 数组
 - 链表
- 图
- 树 连通无回路图
- 集合与字典

基本数据结构

- 线性数据结构

- 数组

n 个相同数据类型的元素构成的序列，连续存储在存储器中，通过数组下标访问。

数据项[0]	数据项[1]	...	数据项[n-1]
--------	--------	-----	----------

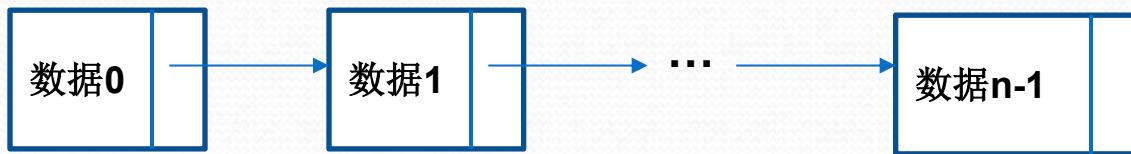
- 数组可构成多种数据结构
 - 字符串
 - 二进制串/位串

基本数据结构

- 线性数据结构

- 链表

由0个或多个节点元素构成的序列，每个节点包括数据和指针两部分，指针可指向其他节点。



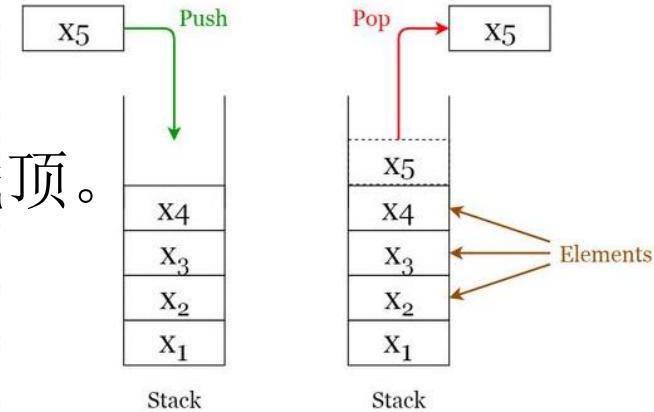
- 表头
 - 双链表

基本数据结构

- 线性数据结构

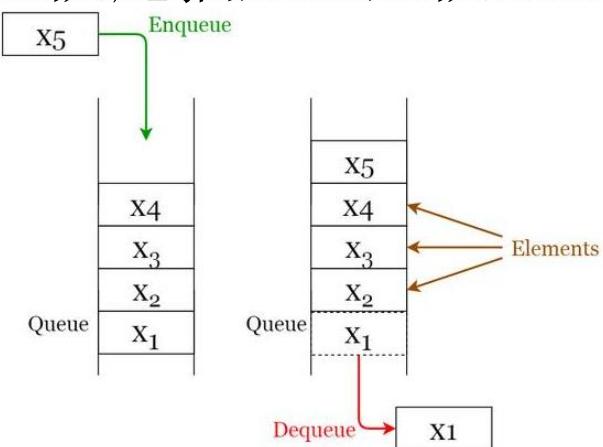
- 栈

- 插入删除都在顶端进行， 栈顶。
 - 后进先出**LIFO**
 - 递归算法中使用保存变量



- 队列

- 数据在对头删除（出队）， 在队尾插入（入队）
 - 先进先出**FIFO**
 - 广度优先搜索



基本数据结构

- 图

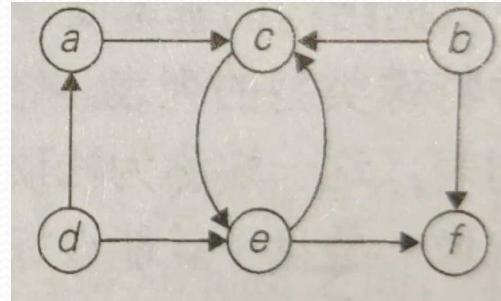
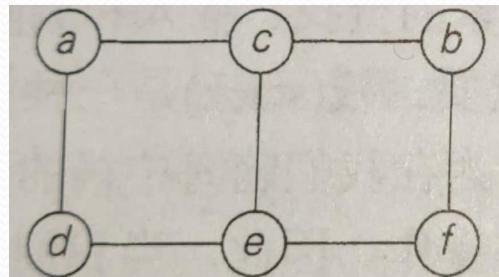
- $G = \langle V, E \rangle$
- 有限集合 V , 元素为顶点
- 有限集合 E , 元素为一对顶点, 称为边

概念:

- 无向图。有边 (u, v) , 则 u, v 相互邻接
- 有向图。有边 (u, v) , 则 u 是尾, v 是头
- 完全图、稠密图、稀疏图

基本数据结构

- 图



- 邻接矩阵，邻接链表

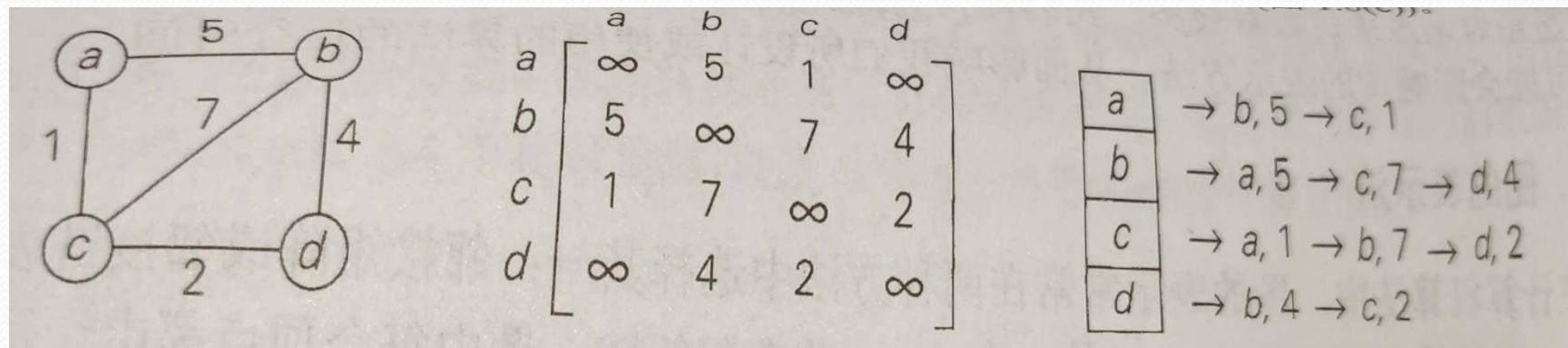
	a	b	c	d	e	f
a	0	0	1	1	0	0
b	0	0	1	0	0	1
c	1	1	0	0	1	0
d	1	0	0	0	1	0
e	0	0	1	1	0	1
f	0	1	0	0	1	0

a	→	c	→	d		
b	→	c	→	f		
c	→	a	→	b	→	e
d	→	a	→	e		
e	→	c	→	d	→	f
f	→	b	→	e		

基本数据结构

- 图

- 加权图，给边赋予值的图。



- 路径和环

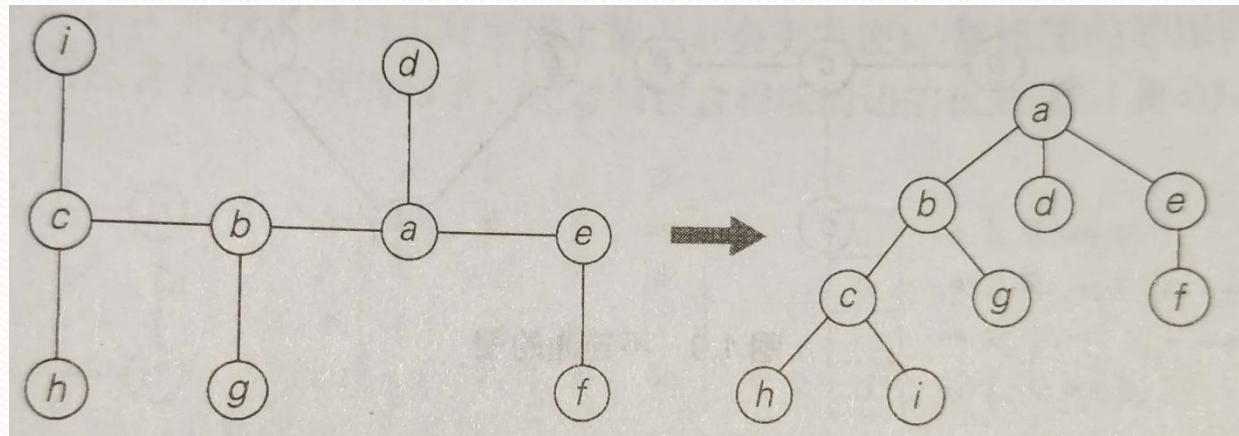
- 路径：始于 u 止于 v 的邻接顶点序列。
- 连通：每对顶点都存在路径
- 回路/环：始点和终点是同一顶点，长度大于0
- 图的连通性、无环性

基本数据结构

- 树

连通无回路的图，自由树。

- 有根树



- 根节点、父节点、兄弟节点、孩子节点、叶节点
- 顶点v的深度、树的高度。

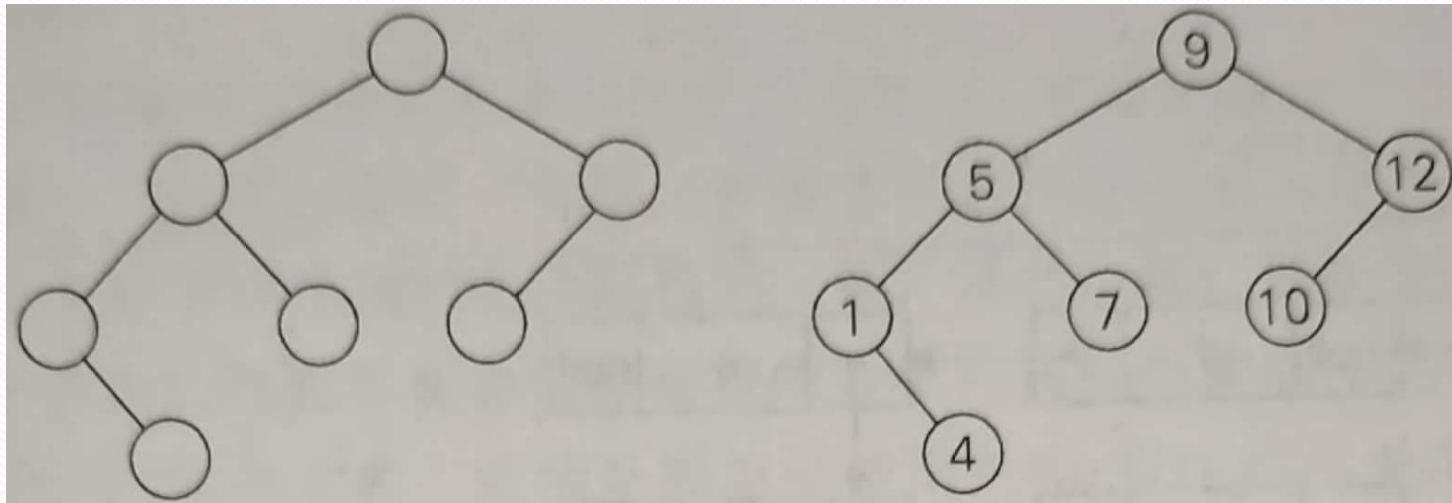
基本数据结构

- 树

- 有序树

有根树中每个顶点所有孩子都是有序的。

- 二叉树。最多两个孩子：左孩子，右孩子
 - 二叉查找树。顶点分配数字，大于左子树，小于右子树。



基本数据结构

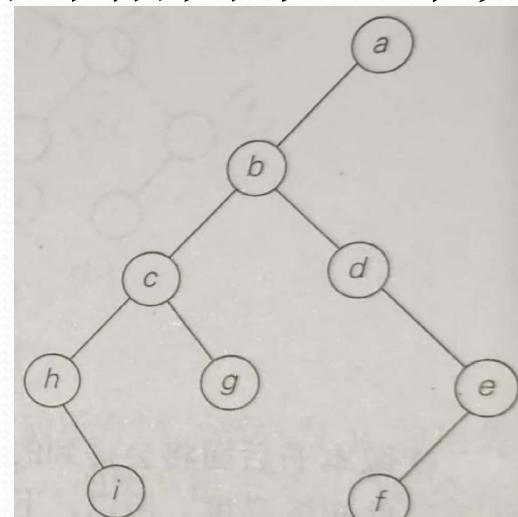
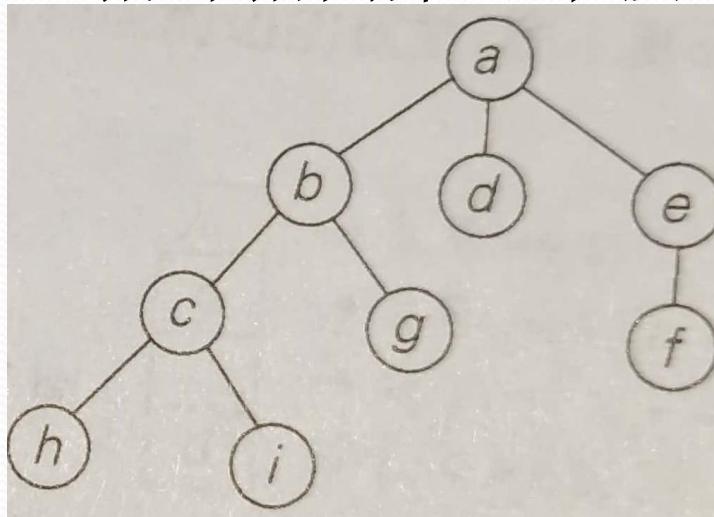
- 树

树的存储：

- 二叉树：父节点加入2个指针
- 有序树：父节点加孩子个数的指针，不方便。

转换为二叉树（孩子兄弟表示法）

左指针指向第一个孩子，右指针指向下一个兄弟。



基本数据结构

- 集合与字典

- 集合：互不相同的无序组合
- 运算：查找，求并，求交
- 集合表示方法
 - 1、通用大集合的子集。

若全集 U 有 n 个元素,可用长度为 n 的位串表示任意子集。

$U=\{1,2,3,4,5,6,7,8,9\}, S=\{2,3,5,7\}$ 表示为011010100

速度快，空间占用多(U 大小决定空间)

- 2、线性列表表示集合元素。
- 字典：能够实现集合元素查找、增加、删除的数据结构