# 零基础学算法

## 第1章：基础算法思想

# 课程安排

- 编程的灵魂：数据结构+算法
- 算法的作用
- 递推算法
- 枚举(穷举)算法
- 递归算法
- 分治算法
- 贪婪算法
- 试探算法
- 模拟算法
- 算法的评价

# 1.1 编程的灵魂：数据结构+算法

程序= 数据结构+ 算法 + 程序设计语言

- 由上面的公式可以看出，程序设计中数据结构和算法是最重要的，是编程的灵魂。
- 数据结构是算法实现的基础，算法总是要依赖于某种数据结构来实现的。往往是在发展一种算法的时候，构建了适合于这种算法的数据结构。一种数据结构如果脱离了算法，也就没有存在的价值了。

# 1.2 算法的作用

- **1.2.1 算法的作用**

解决任何一个实际问题，都不可避免地涉及到算法的问题，例如本章开头提到的存钱问题，再如节假日公司值班人员的排班等，都需要通过一定的算法，得到一个最优（或较优）的方案。

- **1.2.2 实例：看商品猜价格**

首先出示一件价格在999元以内的商品，参与者要猜出这件商品的价格。在猜价格的过程中，主持人会根据参与者给出的价格，相应地给出“高了”或“低了”的提示。

表 1-1　二分法猜商品价格

| 次数 | 价格区间 | 中间值 |
|---|---|---|
| 第1次 | 0~999 | 500 |
| 第2次 | 500~999 | 750 |
| 第3次 | 500~750 | 620 |
| 第4次 | 620~750 | 680 |
| 第5次 | 620~680 | 650 |
| 第6次 | 620~650 | 630 |
| 第7次 | 630~650 | 640 |

# 1.3 递推算法

## 1.3.1 算法思路

递推算法使用“步步为营”的方法，不断利用已有的信息推导出新的东西。

- 顺推法：是指从已知条件出发，逐步推算出要解决问题的方法。例如：斐波拉契数列就可以通过顺推法不断递推算出新的数据。
- 逆推法：是从已知的结果出发，用迭代表达式逐步推算出问题开始的条件，即顺推法的逆过程。

# 1.3 递推算法

## 1.3.2 顺推实例

表 1-2　兔子的繁殖过程

| 月份 | 大兔数量 | 1月大的小兔数量 | 2月大的小兔数量 | 兔子总数 |
|---|---|---|---|---|
| 初始状态 | 0 | 1 | 0 | 1 |
| 1月 | 0 | 0 | 1 | 1 |
| 2月 | 1 | 1 | 0 | 2 |
| 3月 | 1 | 1 | 1 | 3 |
| 4月 | 2 | 2 | 1 | 5 |
| 5月 | 3 | 3 | 2 | 8 |
| 6月 | 5 | 5 | 3 | 13 |
| 7月 | 8 | 8 | 5 | 21 |
| 8月 | 13 | 13 | 8 | 34 |
| 9月 | 21 | 21 | 13 | 55 |
| 10月 | 34 | 34 | 21 | 89 |
| 11月 | 55 | 55 | 34 | 144 |
| 12月 | 89 | 89 | 55 | 233 |

# 1.3 递推算法

## 1.3.3 逆推实例

- 若在第48月小龙大学毕业时连本带息要取1000元，则要先求出第47个月时银行存款的钱数
- 第47月月末存款=1000/(1+0.0171/12)；
- 第46月月末存款=(第47月月末存款+1000)/(1+0.0171/12)
- 依次类推，可以求出第45月、第44月……的月末存款的数值
- 第45月月末存款=(第46月月末存款+1000)/(1+0.0171/12)
- 第44月月末存款=(第45月月末存款+1000)/(1+0.0171/12)
- ……
- 第2月月末存款=(第3月月末存款+1000)/(1+0.0171/12)
- 第1月月末存款=(第2月月末存款+1000)/(1+0.0171/12)

# 1.4 枚举(穷举)算法

## 1.4.1 算法思路

枚举法的本质就是从所有候选答案中去搜索正确的解，使用该算法需要满足两个条件：

（1）可预先确定候选答案的数量；

（2）候选答案的范围在求解之前必须有一个确定的集合。

# 1.4 枚举(穷举)算法

## 1.4.2 实例：填数游戏

| 算法描述题 |
| --- |
| × 算 |
| 题题题题题题 |

| 1 2 3 4 5 |
| --- |
| × 1 |
| 5 5 5 5 5 5 |

# 1.4 枚举(穷举)算法

## 1.4.3 实例：填运算符

5 5 5 5 5＝5

由于算术表达式的特殊性，在编程求解这个算式时，需要注意以下几点：

- 当填入除号时，要求右侧的数不能为0。
- 乘除的运算级别比加减高。

5＋5－5＊5 /5＝5

# 1.5 递归算法

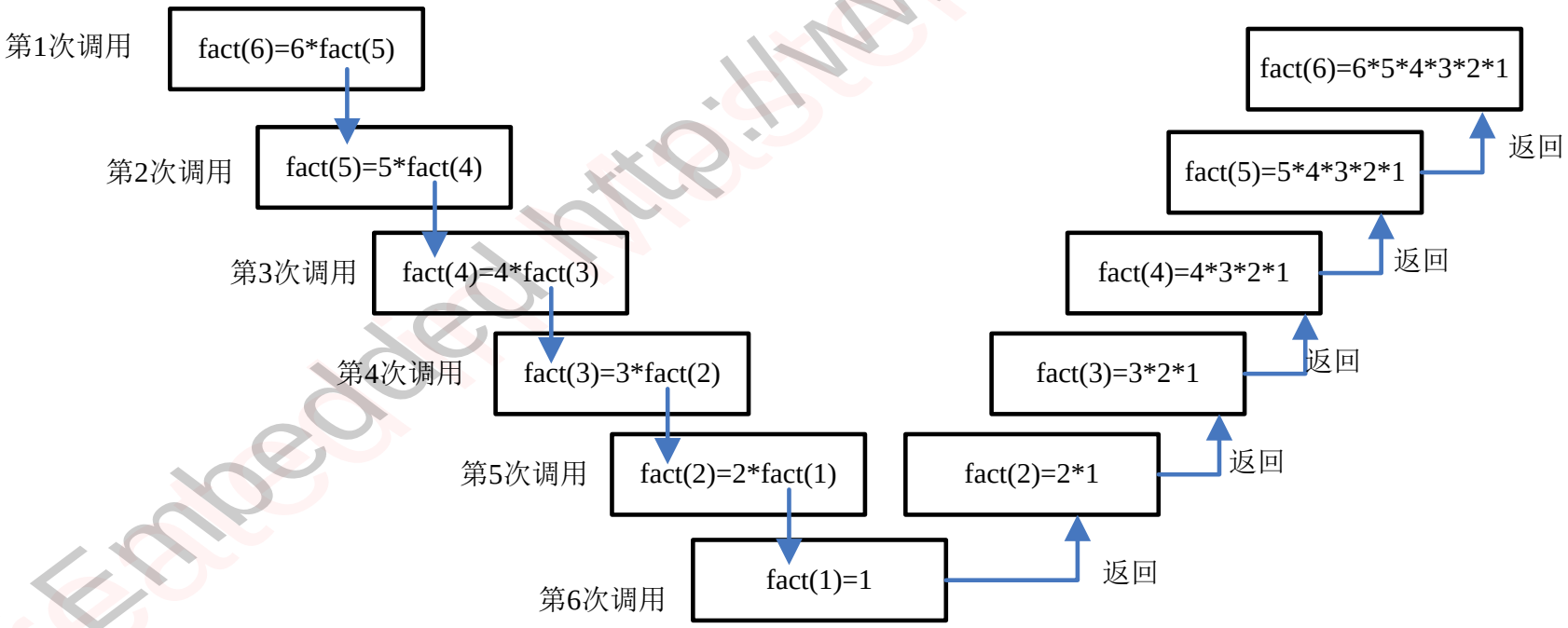
## 1.5.1 算法思路

递归算法，就是一种直接或者间接地调用自身的算法。递归算法的具体实现过程一般通过函数或子过程来完成，在函数或子过程的内部，编写代码直接或者间接地调用自己，即可完成递归操作。

# 1.5 递归算法

## 1.5.2 求阶乘

$$6!=6*5*4*3*2*1$$



# 1.5 递归算法

## 1.5.3 实例：数制转换


$(121)_{10}=(1111001)_2$

# 1.6 分治算法

- **1.6.1　算法思路**

  使用分治法设计程序时，一般可按以下步骤进行：

  （1）分解：将要求解的问题划分成若干规模较小的同类问题；

  （2）求解：当子问题划分得足够小时，用较简单的方法解决；

  （3）合并：按求解问题的要求，将子问题的解逐层合并，即可构成最终的解。

- **1.6.2　实例：乒乓球比赛赛程安排**

表 1-3　循环赛日程表（8 人）

| 选手＼时间 | 第 1 天 | 第 2 天 | 第 3 天 | 第 4 天 | 第 5 天 | 第 6 天 | 第 7 天 |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 2 | 1 | 4 | 3 | 6 | 6 | 8 | 7 |
| 3 | 4 | 1 | 2 | 7 | 8 | 5 | 6 |
| 4 | 3 | 2 | 1 | 8 | 7 | 6 | 5 |
| 5 | 6 | 7 | 8 | 1 | 2 | 3 | 4 |
| 6 | 5 | 8 | 7 | 2 | 1 | 4 | 3 |
| 7 | 8 | 5 | 6 | 3 | 4 | 1 | 2 |
| 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

表 1-4　循环赛日程表（4 人）

| 时间 / 选手 | 第 1 天 | 第 2 天 | 第 3 天 |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 2 | 1 | 4 | 3 |
| 3 | 4 | 1 | 2 |
| 4 | 3 | 2 | 1 |

表 1-5 循环赛日程表（2 人）

| 时间 / 选手 | 第 1 天 |
|---|---|
| 1 | 2 |
| 2 | 1 |

表 1-6　循环赛日程表（4 人）

| 时间 / 选手 | 第 1 天 | 第 2 天 | 第 3 天 |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 2 | 1 | 4 | 3 |
| 3 | 4 | 1 | 2 |
| 4 | 3 | 2 | 1 |

# 1.7 贪婪算法

- **1.7.1 算法思路**

贪婪算法基本思路：从问题的某一个初始解出发逐步逼近给定的目标，以尽可能快地求得更好的解。当达到算法中的某一步不能再继续前进时，就停止算法，给出近似解。

由贪婪算法的特点和思路可看出，该算法存在以下问题：

  - 不能保证最后的解是最优的；
  - 不能用来求最大或最小解问题；
  - 只能求满足某些约束条件的可行解的范围。

- **1.7.2 实例：换零钱**

人民币有100、50、10、5、2、1、0.5、0.2、0.1等多种面额（单位为元）。在补零钱时，可有多种方案，例如需补零钱68.90元，至少可有以下方案：

  - 1张50、1张10、1张5、3张1、1张0.5、2张0.2；
  - 1张50、1张10、1张5、3张1、1张0.5、4张0.1；
  - 6张10、1张5、3张1、1张0.5、2张0.2。

# 1.8 试探算法

- **1.8.1 算法思路**

  为了求得问题的解，先选择某一种可能情况进行试探，在试探过程中，一旦发现原来的选择的假设情况是错误的，就退回一步重新选择，继续向前试探，如此反复进行，直至得到解或证明无解。

- **1.8.2 实例：生成彩票号码组合**

  假设有一种彩票，每注由7个1~29的数字组成，且这7个数字不能相同，编写程序生成所有的号码组合。

# 1.9 模拟算法

- **1.9.1 算法思路**

  在程序设计语言中，可使用随机函数来模拟自然界中发生的不可预测情况。C语言中使用srand()和rand()函数可生成随机数。

- **1.9.2 实例：猜数游戏**

  使用模拟法编写一个猜数游戏，由计算机随机生成一个1~100之内的整数，然后由用户来猜这个数，根据用户猜测的次数分别给出不同的提示文字。

- **1.9.3 实例：模拟掷骰子游戏**

  由用户输入骰子数量和参赛人数，然后由计算机随机生成每一粒骰子的数量，再累加起来就得到每一个选手的总点数。

# 1.10 算法的评价

- **1.10.1 算法评价原则**
  - 正确性。
  - 高效性。
  - 空间性。
  - 可读性。
- **1.10.2 算法的效率**

通常认为，通过统计算法中基本操作重复执行的次数就可近似地得到算法的执行效率，用O(*n*)表示，称为时间复杂度。

性格决定命运，专注成就人生

# 零基础学算法

## 第2章：简单数据结构

# 课程安排

- **2.1 最简单的结构：线性表**
  - 什么叫线性表
  - 操作顺序表
  - 操作链表
  - 实例：用链表制作通信录
- **2. 2 先进先出结构：队列**
  - 什么是队列
  - 操作队列
  - 循环队列的操作
  - 实例：银行排号程序
- **2.3 后进先出结构：栈**
  - 什么是栈
  - 操作栈
  - 实例：算术表达式求值

# 2.1 最简单的结构：线性表

- 2.1.1 什么叫线性表
- 2.1.2 操作顺序表
- 2.1.3 操作链表
- 2.1.4 实例：用链表制作通信录

# 2.1 最简单的结构：线性表

## 2.1.1 什么叫线性表

线性表数据结构具有以下特征：

- 有且只有一个“首元素”；
- 有且只有一个“末元素”；
- 除末元素之外，其余元素均有惟一的后继元素；
- 除首元素之外，其余元素均有惟一的前驱元素。

对于线性表，主要可进行以下操作：

- 添加结点；
- 插入结点；
- 删除结点；
- 查找结点；
- 遍历结点；
- 统计结点数。

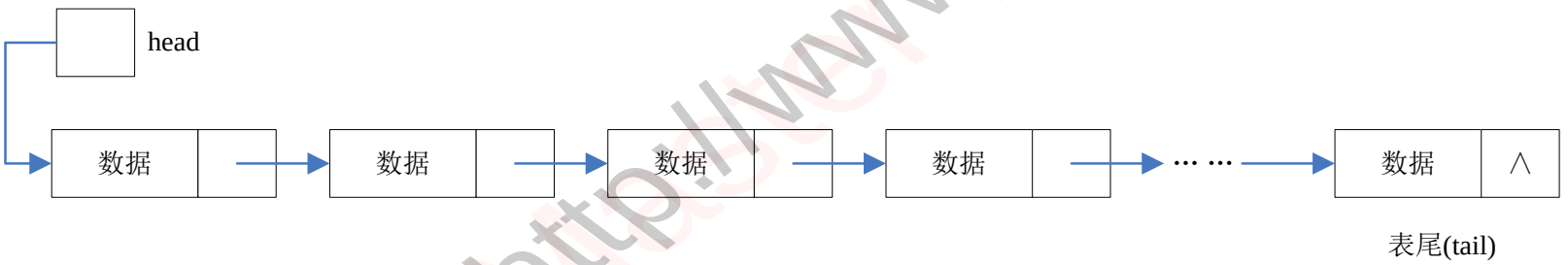
# 2.1 最简单的结构：线性表

## 2.1.2 操作顺序表

- 1. 定义顺序队列结构
- 2. 初始化队列
- 3. 获取队列状态
- 4. 入队操作
- 5. 出队操作
- 6. 获取队头元素

# 2.1 最简单的结构：线性表

## 2.1.3 操作链表

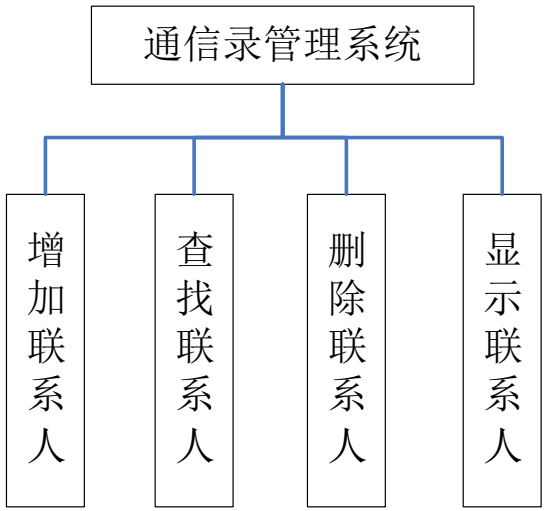



- 1. 定义链表的结构
- 2. 添加结点至尾部
- 3. 添加结点至首部
- 4. 插入结点
- 5. 查找结点
- 6. 删除结点
- 7. 链表的长度
- 8. 测试链表操作

# 2.1 最简单的结构：线性表

## 2.1.4 实例：用链表制作通信录

- 1. 定义通信录结构
- 2. 编写显示联系人信息模块
- 3. 编写添加联系人模块
- 4. 编写查找联系人模块
- 5. 编写删除联系人模块
- 6. 编写主模块



# 2.2 先进选出结构：队列

- 2.2.1 什么是队列
- 2.2.2 操作队列
- 2.2.3 循环队列的操作
- 2.2.4 实例：银行排号程序

# 2.2 先进选出结构：队列

## 2.2.1 什么是队列

队列是一种特殊的线性表，只允许在表的前端进行删除操作，而在表的后端进行插入操作。进行插入操作的端称为队尾，进行删除操作的端称为队头。当队列中没有元素时，称为空队列。

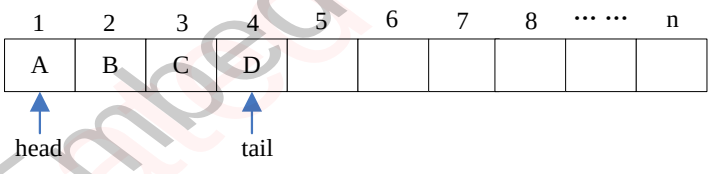
对于队列这种结构，其操作很简单，主要有以下几种：

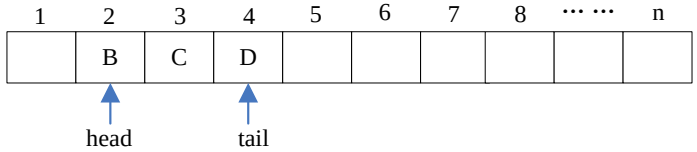
- 初始化队列：创建一个队列。
- 进队列：将一个元素添加到队尾（相当于到队列最后排队等候）。
- 出队列：将队头的元素取出，同时删除该元素，使后一个元素成为队头。
- 获取队列第1个元素：将队头的元素取出，不删除该元素（队头仍然是该元素）。
- 获取队列长度：根据队头、队尾计算出队列中元素的数量。

# 2.2 先进选出结构：队列

## 2.2.2 操作队列

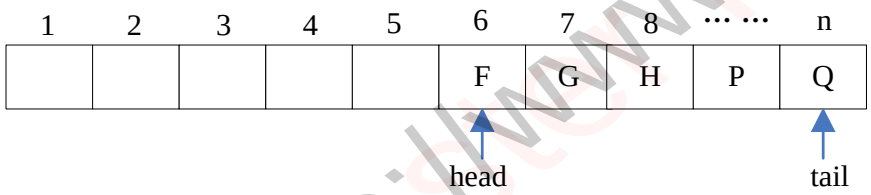
- 1. 定义顺序队列结构
- 2. 初始化队列
- 3. 获取队列状态
- 4. 入队操作
- 5. 出队操作
- 6. 获取队头元素

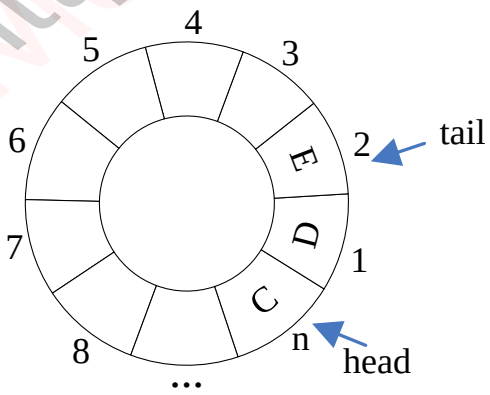


# 2.2 先进选出结构：队列

## 2.2.3 循环队列



# 2.3 后进先出结构：栈

- 2.3.1 什么是栈
- 2.3.2 操作栈
- 2.3.3 实例：算术表达式求值

# 2.3 后进先出结构：栈

## 2.3.1 什么是栈

栈是一种线性表的特殊表现形式，与队列的“先进先出”不同，栈是按照“后进先出”（Last In Firt Out，LIFO）的原则处理数据。

- 栈的基本操作只有两个：
  - 入栈（Push）：即将数据保存到栈顶。进行该操作前，先修改栈顶指针，使其向上移一个元素位置，然后将数据保存到栈顶指针所指的位置。
  - 出栈（Pop）：即将栈顶的数据弹出，然后修改栈顶指针，使其指向栈中的下一个元素。

# 2.3 后进先出结构：栈

## 2.3.2 操作栈

- 1. 定义顺序栈的结构
- 2. 初始化栈
- 3. 判断栈的状态
- 4. 入栈操作
- 5. 出栈操作
- 6. 获取栈顶元素
- 7. 测试栈的操作

# 2.3 后进先出结构：栈

## 2.3.3 实例：算术表达式求值

对于算术表达式的求值，主要就是解决算术运算符的优先级问题，有以下规则：

- 先进行乘除运算，再进行加减运算（乘除优先级大于加减）；
- 对于相同优先级的运算符，从左向右计算；
- 若要改变优先级，可使用括号。对有括号的表达式，先计算括号内，再计算括号外。

在表达式的计算过程中，既要保存操作数，又要保存运算符。这时，可定义两个栈，一个用来保存操作数，一个用来保存运算符。

性格决定命运，专注成就人生

# 零基础学算法

## 第3章：复杂数据结构

# 课程安排

- **3.1** 层次关系结构：树
- **3.2** 网状关系：图

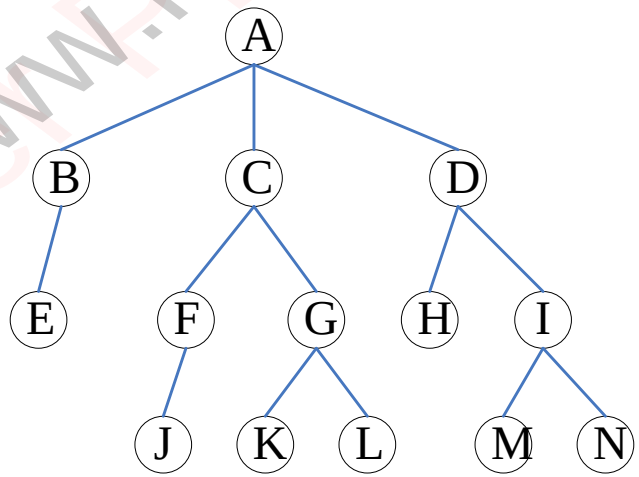
# 3.1 层次关系结构：树

- 3.1.1 树的概念
- 3.1.2 二叉树的概念
- 3.1.3 二叉树的存储
- 3.1.4 操作二叉树
- 3.1.5 遍历二叉树
- 3.1.6 测试二叉树
- 3.1.7 线索二叉树
- 3.1.8 最优二叉树（赫夫曼树）

# 3.1 层次关系结构：树

## 3.1.1 树的概念

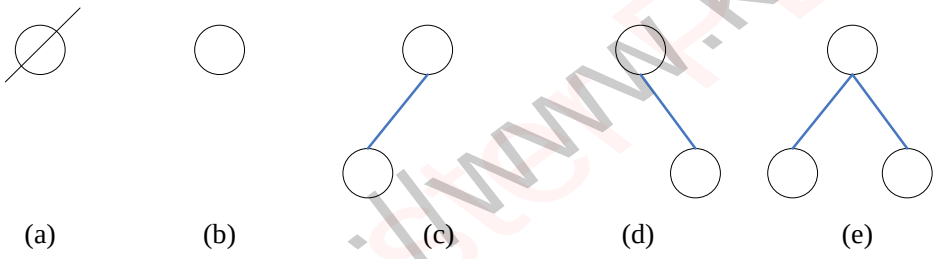
- 1. 树的定义
- 2. 树的相关术语
- 3. 树的表示



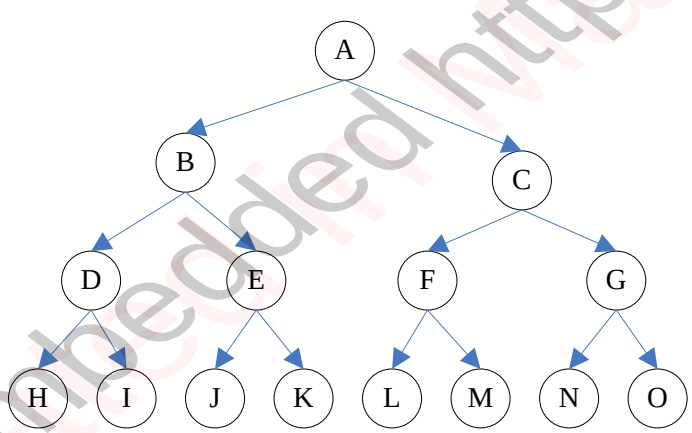

- (A(B(E)),(C(F(J)),(G(K,L))),(D(H),(I(M,N))))

# 3.1 层次关系结构：树

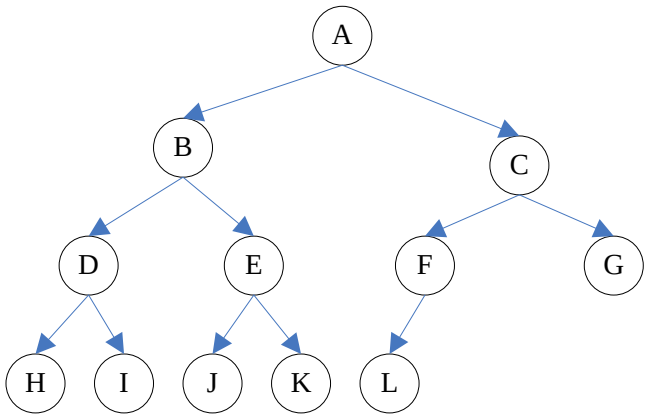
## 3.1.2 二叉树的概念



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




(a)满二叉树




(b)完全二叉树

# 3.1 层次关系结构：树

## 3.1.2 二叉树的概念

- 根据二叉树的定义，可得知其具有以下性质：
  - （1）在二叉树中，第i层的结点总数最多有2i-1个结点。
  - （2）深度为k的二叉树最多有2k-1个结点（k>=1），最少有k个结点。
  - （3）对于一棵二叉树，如果其叶结点数为n0，而度为2的结点总数为n2，则n0=n2+1。
  - （4）具有n个结点的完全二叉树的深度k为：k=[log2n]+1。
  - （5）有n个结点的完全二叉树各结点如果用顺序方式存储，对任意结点i，有如下关系：

    如果i != 1，则其父结点的编号为i/2；

    如果2*i<=n，则其左子树根结点的编号为2*i；若2*i>n，则无左子树；

    如果2*i+1<=n，则其右子树根结点编号为2*i+1；若2*i+1>n，则无右子树。

# 3.1 层次关系结构：树

## 3.1.3 二叉树的存储

- **1. 顺序存储结构**

二叉树顺序存储结构的数据定义如下：

```
#define MAXSIZE 100              //最大结点数
typedef int DATA;                //元素类型
typedef DATA SeqBinTree[MAXSIZE];
SeqBinTree SBT;                  //定义保存二叉树数组
```

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G | H | I | J | K | L |

# 3.1 层次关系结构：树

## 3.1.3 二叉树的存储

- **2. 链式存储结构**

二叉链式结构

```
typedef struct ChainTree
{
    DATA data;
    struct ChainTree *left;
    struct ChainTree *right;
}ChainTreeType;
ChainTreeType *root=NULL;
```


(a)二叉链式结构

三叉链式结构

```
typedef struct ChainTree
{
    DATA data;
    struct ChainTree *left;
    struct ChainTree *right;
    struct ChainTree *parent;
}ChainTreeType;
ChainTreeType *root=NULL;
```


(b)三叉链式结构

# 3.1 层次关系结构：树

## 3.1.4 操作二叉树

- 1. 定义二叉链式结构
- 2. 初始化二叉树
- 3. 添加结点到二叉树
- 4. 获取二叉树左右子树
- 5. 获取二叉树状态
- 6. 在二叉树中查找
- 7. 清空二叉树

# 3.1 层次关系结构：树

## 3.1.5 遍历二叉树

- 先序遍历（DLR）：称为先根次序遍历，即先访问根结点，再按先序遍历左子树，最后按先序遍历右子树。
- 中序遍历（LDR）：称为中根次序遍历，即先按中序遍历左子树，再访问根结点，最后按中序遍历右子树。
- 后序遍历（LRD）：称为后根次数遍历，即先按后序遍历左子树，再按后序遍历右子树，最后访问根结点。
- 按层遍历：按二叉树的层进行遍历，可更直观地从图中得出遍历的结果。

# 3.1 层次关系结构：树

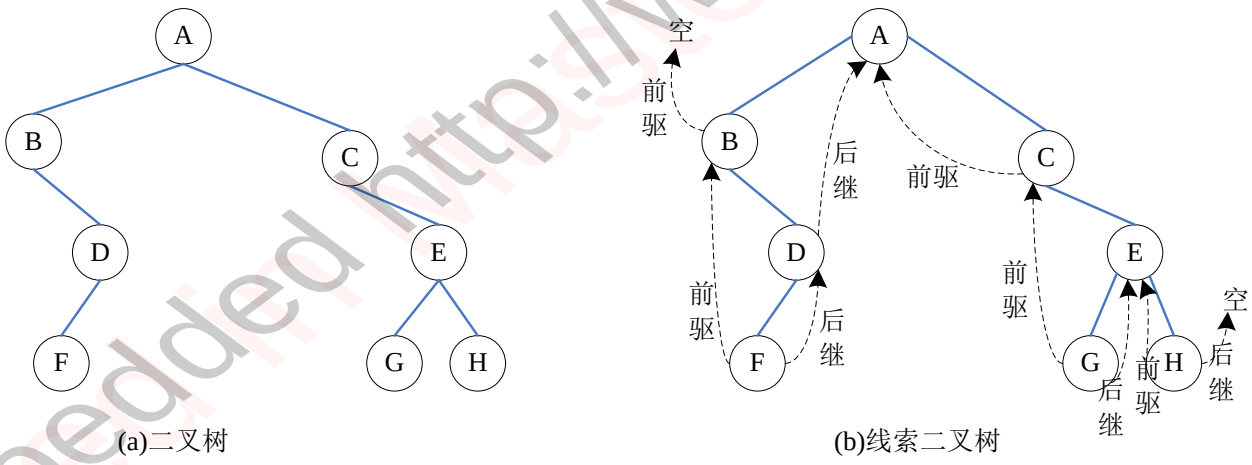
## 3.1.6 测试二叉树

编写程序测试二叉树，在该测试程序中，将创建二叉树，并向二叉树中添加结点，然后能按4种方法进行遍历。

# 3.1 层次关系结构：树

## 3.1.7 线索二叉树

- **1. 线索二叉树的表示**




(a)二叉树　　(b)线索二叉树

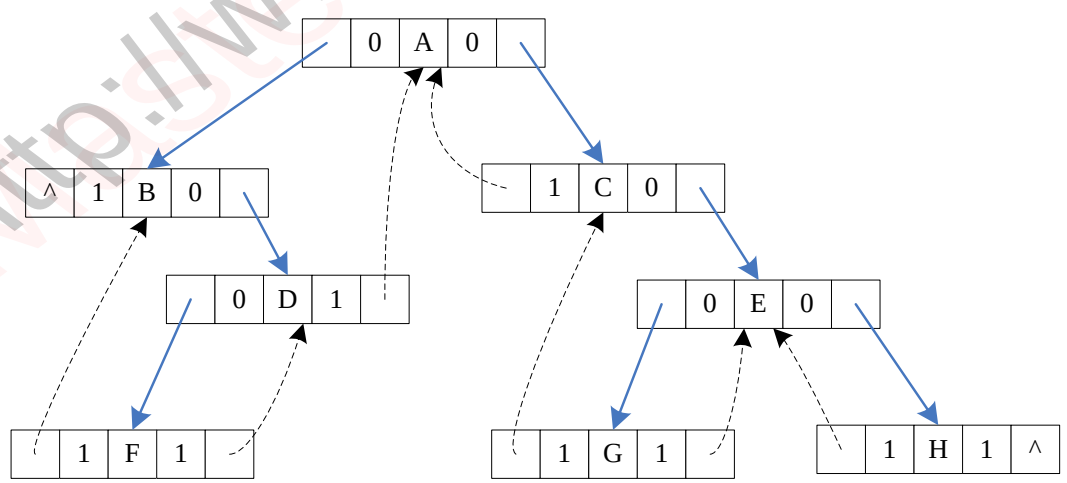
**中序:B F D A C G E H**

# 3.1 层次关系结构：树

## 3.1.7 线索二叉树

- **1. 线索二叉树的表示**

```
typedef struct ThreadTree
{
    DATA data;
    NodeFlag lflag;
    NodeFlag rflag;
    struct ThreadTree *left;
    struct ThreadTree *right;
}ThreadBinTree;
```



# 3.1 层次关系结构：树

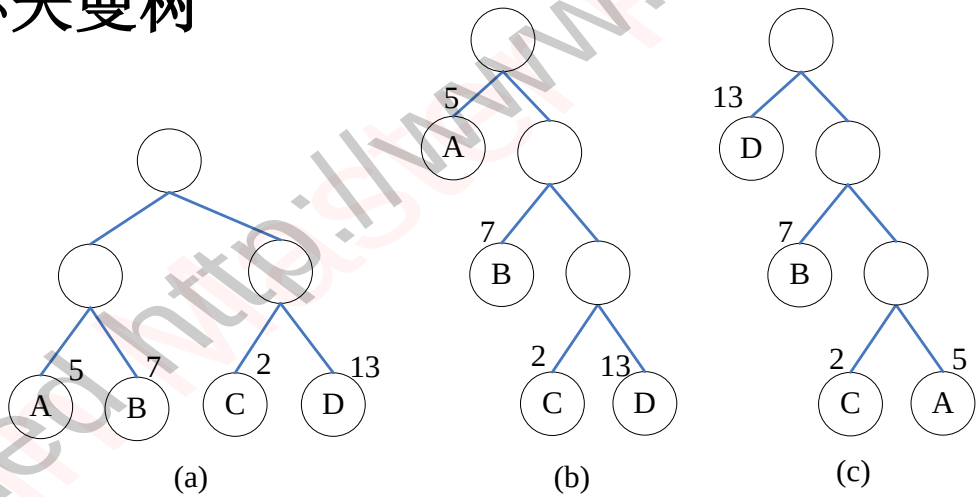
## 3.1.7 线索二叉树

- 创建线索二叉树
- 操作线索二叉树
- 遍历线索二叉树
- 测试线索二叉树

# 3.1 层次关系结构：树

## 3.1.8 最优二叉树（赫夫曼树）

- **1. 创建赫夫曼树**




(a) WPL=5×2+7×2+2×2+13×2=54

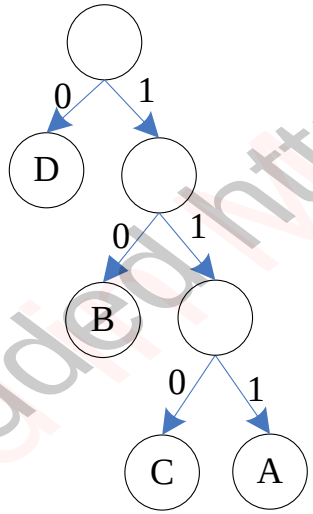
(b) WPL=5×1+7×2+2×3+13×3=64

(c) WPL=1×13+2×7+3×2+5×3=48

# 3.1 层次关系结构：树

## 3.1.8 最优二叉树（赫夫曼树）

- 2．赫夫曼编码




A：111
B：10
C：110
D：0

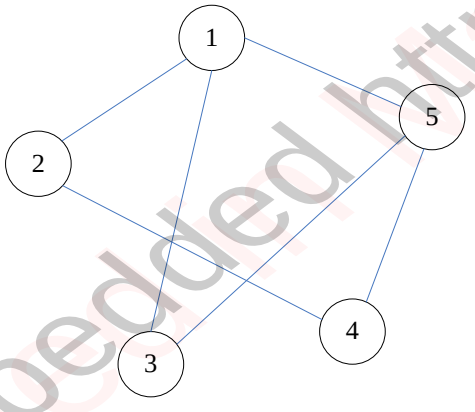
# 3.2 网状关系：图

- 3.2.1 图的定义和基本术语
- 3.2.2 图的存储
- 3.2.3 创建图
- 3.2.4 图的遍历
- 3.2.5 最小生成树
- 3.2.6 最短路径

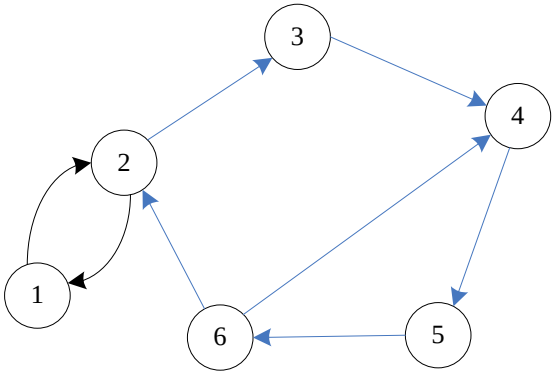
# 3.2 网状关系：图

## 3.2.1 图的定义和基本术语

- 无向图
- 有向图




(a)无向图



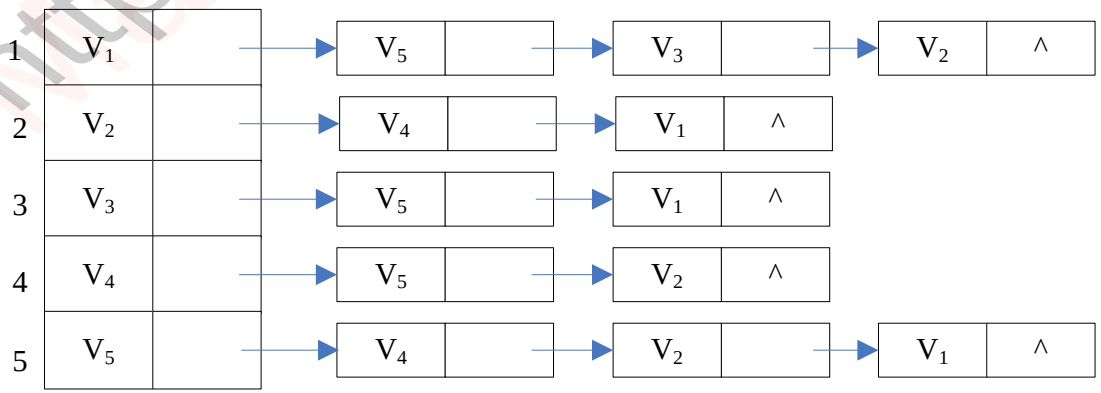

(b)有向图

# 3.2 网状关系：图

## 3.2.2 图的存储

- 邻接矩阵
- 邻接表

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 1 |
| 2 | 1 | 0 | 0 | 1 | 0 |
| 3 | 1 | 0 | 0 | 0 | 1 |
| 4 | 0 | 1 | 0 | 0 | 1 |
| 5 | 1 | 0 | 1 | 1 | 0 |



# 3.2 网状关系：图

## 3.2.3 创建图

- 用邻接矩阵保存图
- 用邻接表保存图

# 3.2 网状关系：图

## 3.2.4 图的遍历

- **广度优先遍历**

广度优先遍历类似于树的按层次遍历，具体过程如下：

- （1）从isTrav数组中选择一个未被访问的顶点Vi，将其标记为已访问。
- （2）接着依次访问Vi的所有未被访问的邻接点，并标记为已被访问过。
- （3）从这些邻接点出发进行广度优先遍历，直至图中所有和Vi有路径相通的顶点都被访问过。
- （4）重复步骤（1）至步骤（3）的操作，直至所有顶点都被访问过。

# 3.2 网状关系：图

## 3.2.4 图的遍历

- **深度优先遍历**

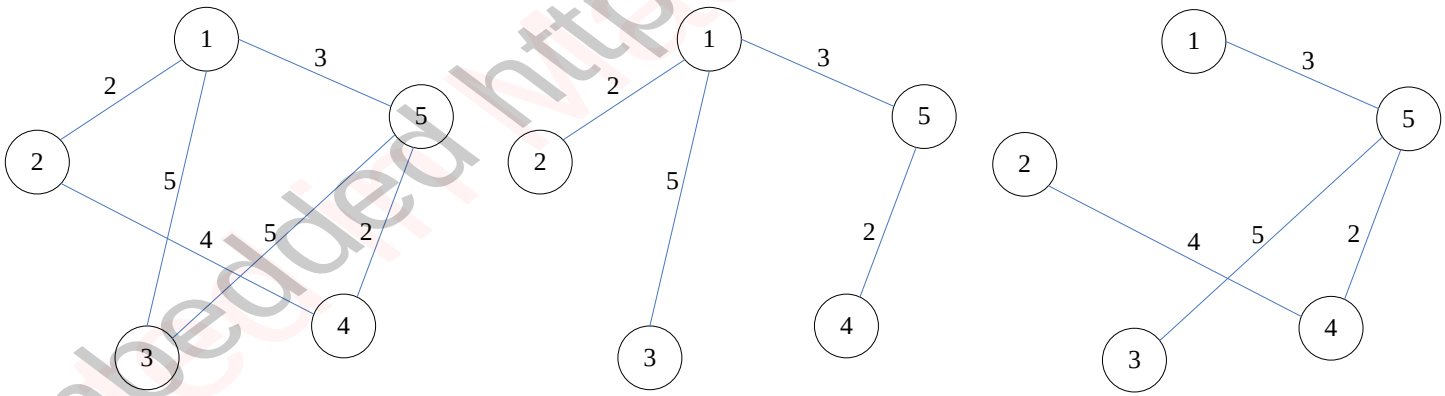
深度优先遍历方法类似于树的先序遍历，具体过程如下：

- （1）从isTrav数组中选择一个未被访的顶点Vi，将其标记为已访问。
- （2）接着从Vi的一个未被访问过的邻接点出发进行深度优先遍历。
- （3）重复步骤（2），直至图中所有和Vi有路径相通的顶点都被访问过。
- （4）重复步骤（1）至步骤（3）的操作，直至所有顶点都被访问过。

# 3.2 网状关系：图

## 3.2.5 最小生成树

对于一个带权连通图，生成树不同，树中各边上的权值总和也不同，权值最小的生成树称为图的最小生成树。



# 3.2 网状关系：图

## 3.2.6 最短路径

对于一个带权连通图，称路径的起始顶点为源点，最后一个顶点为终点。下面介绍从一个顶点到图中其他各顶点的最短路径。

性格决定命运，专注成就人生

# 零基础学算法

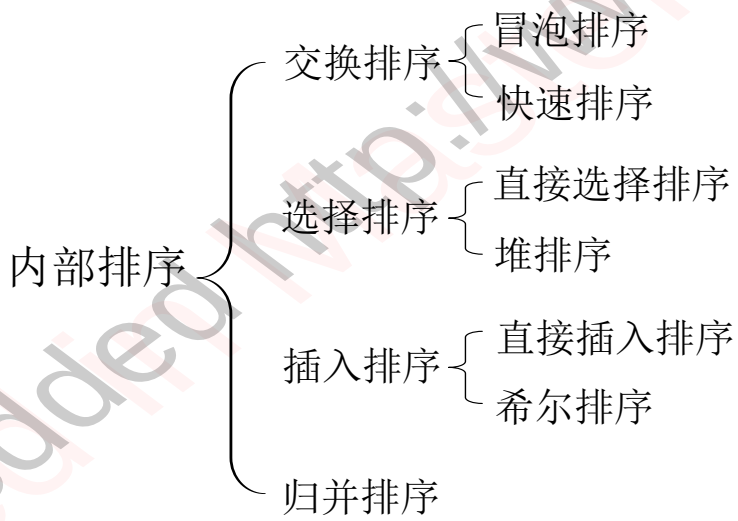
## 第4章：常用算法——排序

# 课程安排

- **4.1** 排序概述
- **4.2** 冒泡排序法
- **4.3** 快速排序法
- **4.4** 简单选择排序法
- **4.5** 堆排序法
- **4.6** 直接插入排序法
- **4.7** 希尔**(shell)**排序法
- **4.8** 合并排序法
- **4.9** 排序算法的选择

# 4.1 排序概述

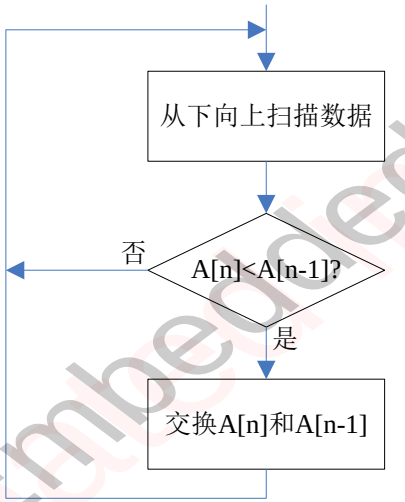
- 内部排序
- 外部排序

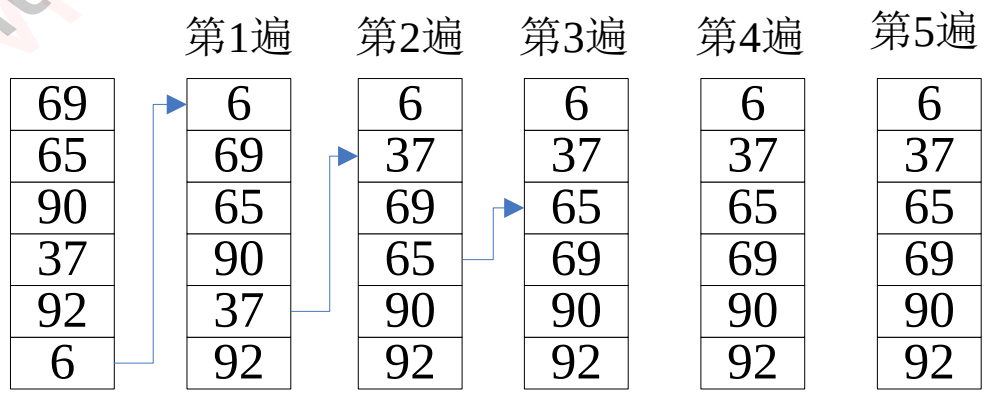


# 4.2 冒泡排序法

## 4.2.1 冒泡排序法

冒泡排序法的基本思想是：对待排序记录关键字从后往前（逆序）进行多遍扫描，当发现相邻两个关键字的次序与排序要求的规则不符时，就将这两个记录进行交换。这样，关键字较小的记录将逐渐从后面向前面移动，就象气泡在水中向上浮一样，所以该算法也称为气泡排序法。



# 4.2 冒泡排序法

## 4.2.2 改进的冒泡排序法

为了提升冒泡排序法的效率，可对BubbleSort函数进行改进，当在某一遍扫描时，发现数据都已经按顺序排列了，就不再进行后继的扫描，而结束排序过程。

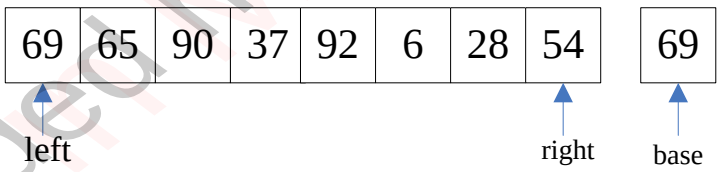
# 4.3 快速排序法

快速排序使用分治策略来把待排序数据序列分为两个子序列，具体步骤为：

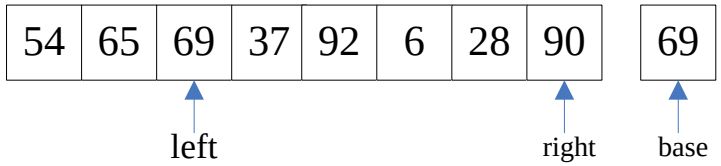
（1）从数列中挑出一个元素，称该元素为“基准”。

（2）扫描一遍数列，将所有比“基准”小的元素排在基准前面，所有比“基准”大的元素排在基准后面。

（3）通过递归，将各子序列划分为更小的序列，直到把小于基准值元素的子数列和大于基准值元素的子数列排序。




(a)




(b)

# 4.4 简单选择排序法

选择排序（Selection Sort）的基本思想：对n个记录进行扫描，选择最小的记录，将其输出，接着在剩下的n-1个记录中扫描，选择最小的记录将其输出，……不断重复这个过程，直到只剩一个记录为止。

# 4.5 堆排序法

堆是一个完全二叉树，树中每个结点对应于原始数据的一个记录，并且每个结点应满足以下条件：非叶结点的数据大于或等于其左、右孩子结点的数据（若是按从大到小的顺序排序，则要求非叶结点的数据小于或等于其左、右孩子结点的数据）。

由堆的定义可看出，其根结点为最大值，堆排序就是利用这一特点进行的。堆排序过程包括两个阶段：

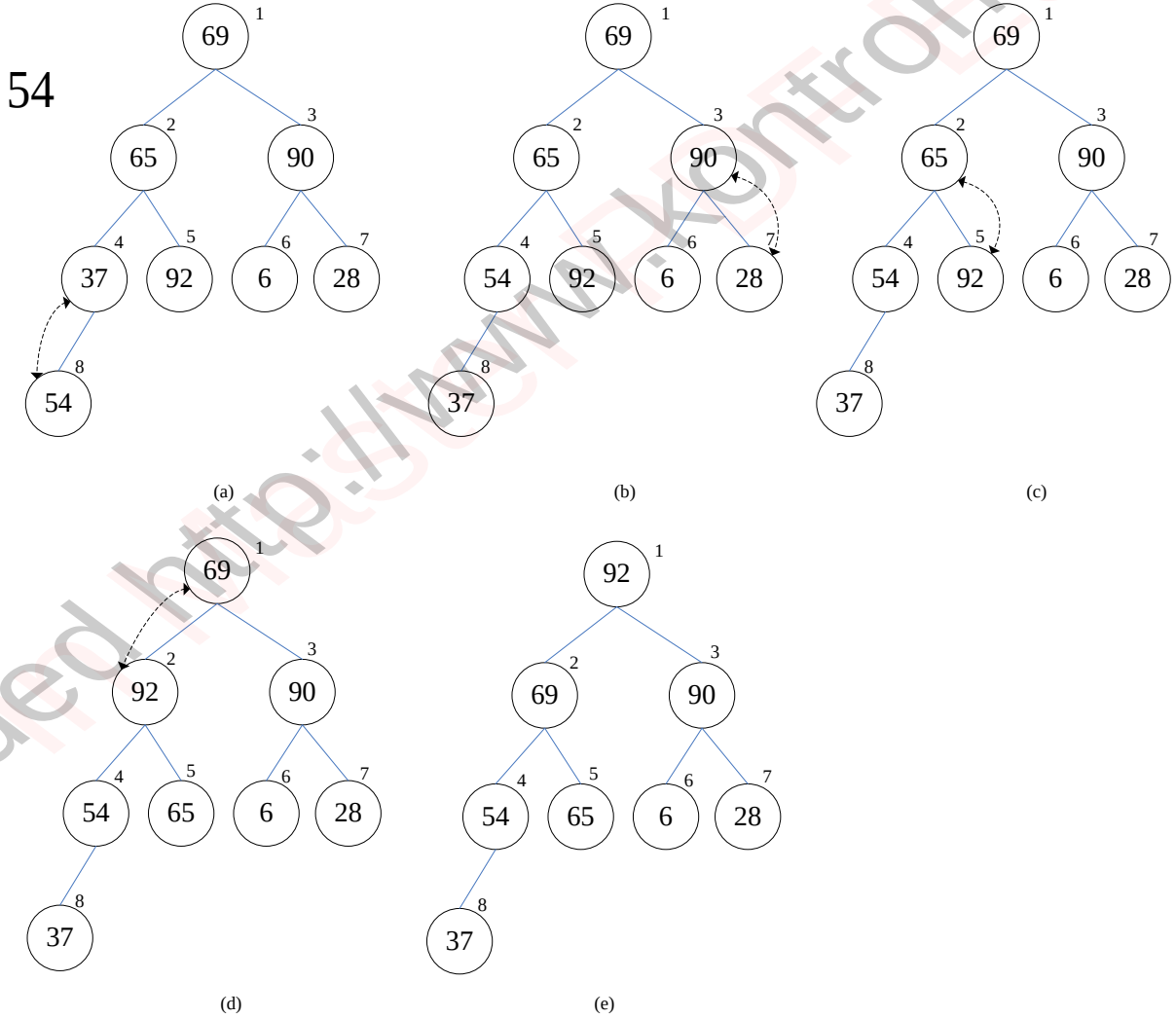
（1）将无序的数据构成堆（即用无序数据生成满足堆定义的完全二叉树）。

（2）利用堆排序（即用上一步生成的堆输出有序的数据）。

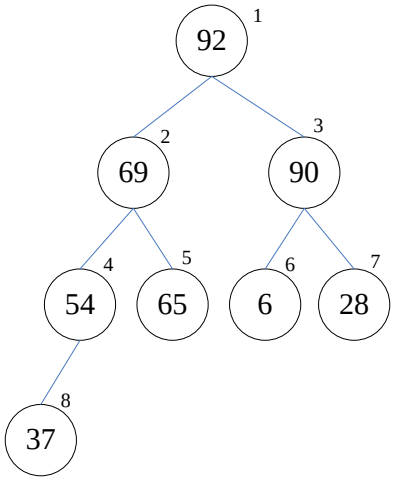
69, 65, 90, 37, 92, 6, 28, 54

# 4.5 堆排序法

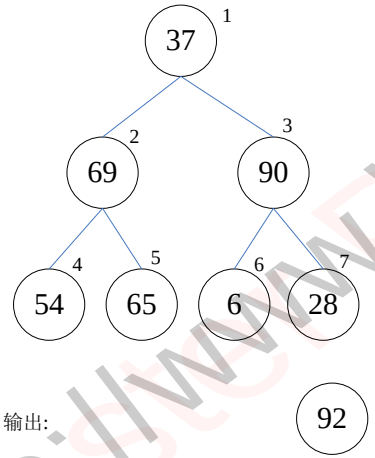
69, 65, 90, 37, 92, 6, 28, 54



# 4.5 堆排序法

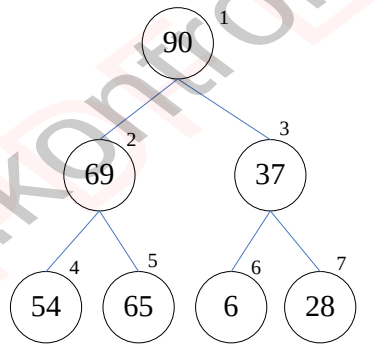



(a)



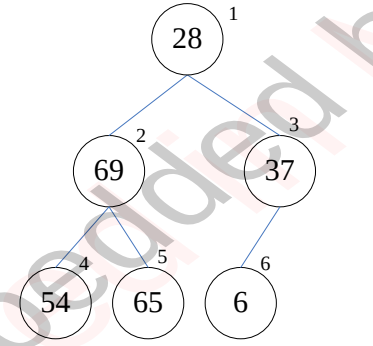

输出: 92

(b)



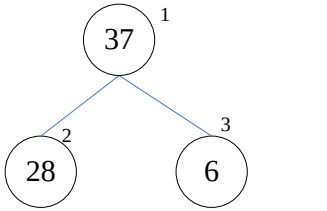

输出: 92

(c)



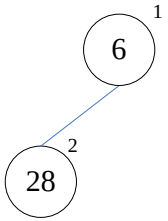

输出: 90 92

(d)




输出: 54 65 69 90 92

(e)

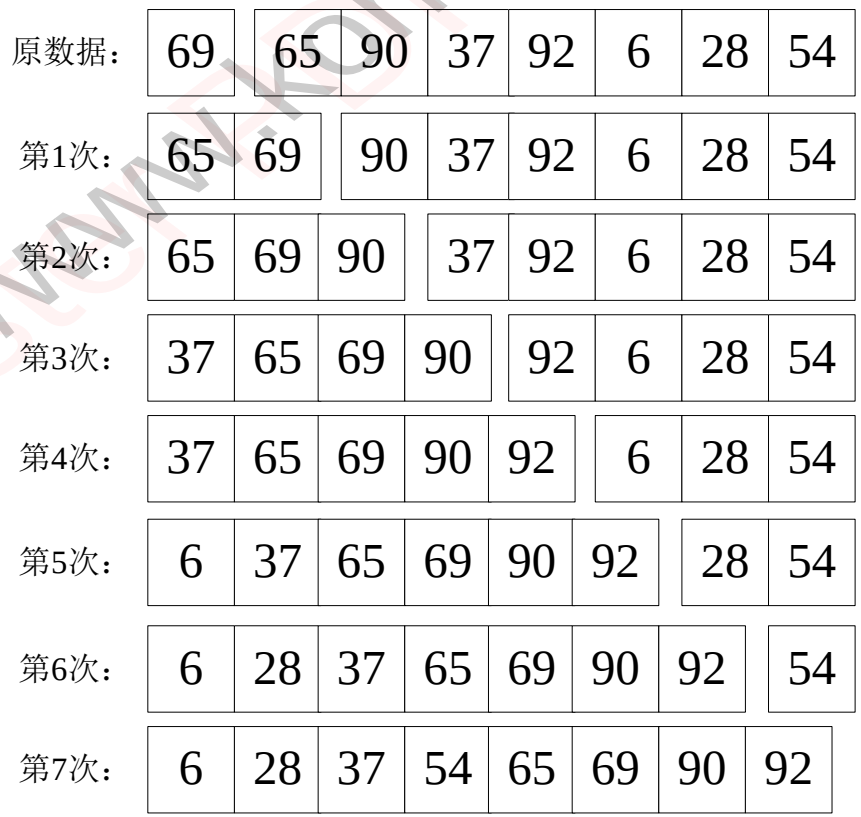



输出: 37 54 65 69 90 92

(f)

# 4.6 直接插入排序法

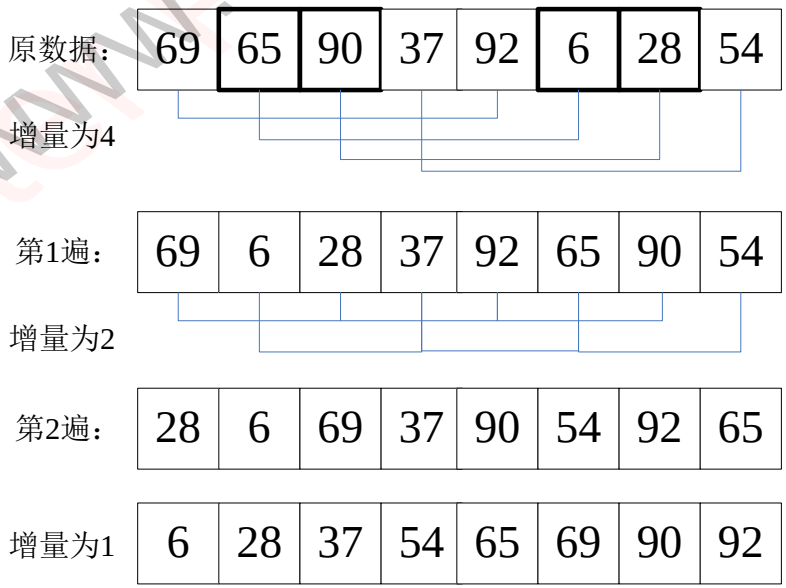
插入排序（Insertion Sort）的算法描述是一种简单直观的排序算法。它的工作原理是通过构建有序序列，对于未排序数据，在已排序序列中从后向前扫描，找到相应位置并插入。插入排序在实现上，在从后向前扫描过程中，需要反复把已排序元素逐步向后移动，为最新元素提供插入空间。



# 4.7 希尔排序法

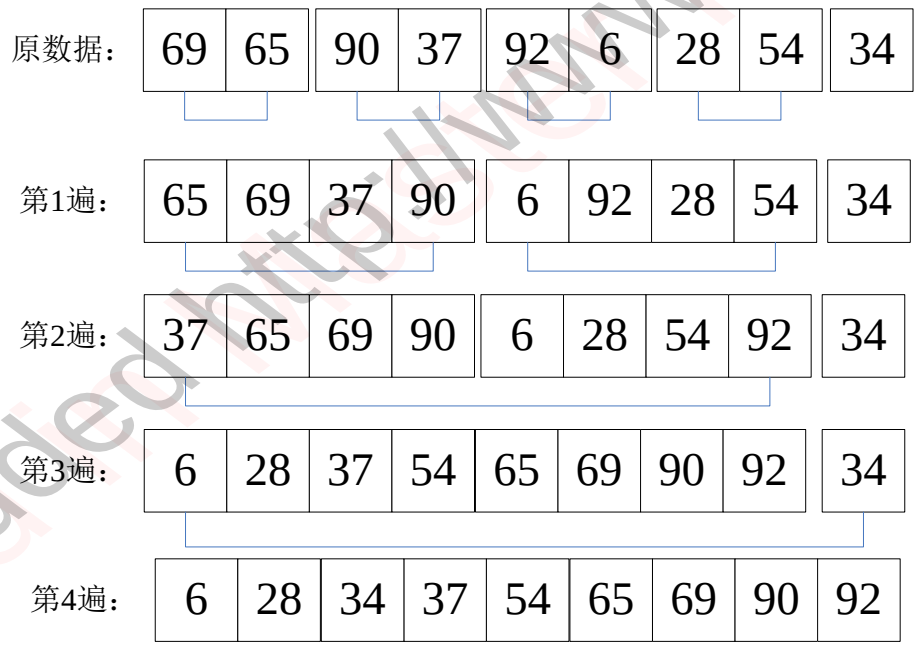
希尔排序又称为缩小增量排序，也属于插入排序类的算法，是对直接插入排序的一种改进。

基本思想就是：将需要排序的序列划分为若干个较小的序列，对这些序列进行直接插入排序，通过这样的操作可使用需要排序的数列基本有序，最后再使用一次直接插入排序。这样，首先对数量较小的序列进行直接插入排序可提高效率，最后对基本有序的序列进行直拦插入排序，也可提高效率，从而使整个排序过程的效率得到提升。



# 4.8 合并排序法

合并排序（Merge Sort）就是将两个或多个有序表合并成一个有序表。



# 4.9 排序算法的选择

- **1. 选择基准**
  - 计算的复杂度
  - 系统资源的使用
  - 稳定度
- **2. 各种排序算法的优缺点**

性格决定命运，专注成就人生

# 零基础学算法

## 第5章: 常用算法——查找

# 课程安排

- **5.1** 查找的基本概念
- **5.2** 简单查找
- **5.3** 二叉排序树
- **5.4** 索引查找
- **5.5** 哈希表

# 5.1 查找的基本概念

- 主关键字和次关键字
- 查找结果
- 静态查找表和动态查找表

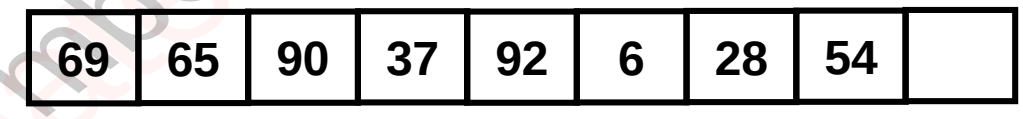
# 5.2 简单查找

- **顺序查找**

  从线性表的一端开始，依次将每个记录的关键字与给定值进行比较，若某个记录的关键字等于给定值，表示查找成功，返回记录序号；若将线性表中所有记录都比较完，仍未找到关键字与给定值相等的记录，则表示查找失败，返回一个失败值。

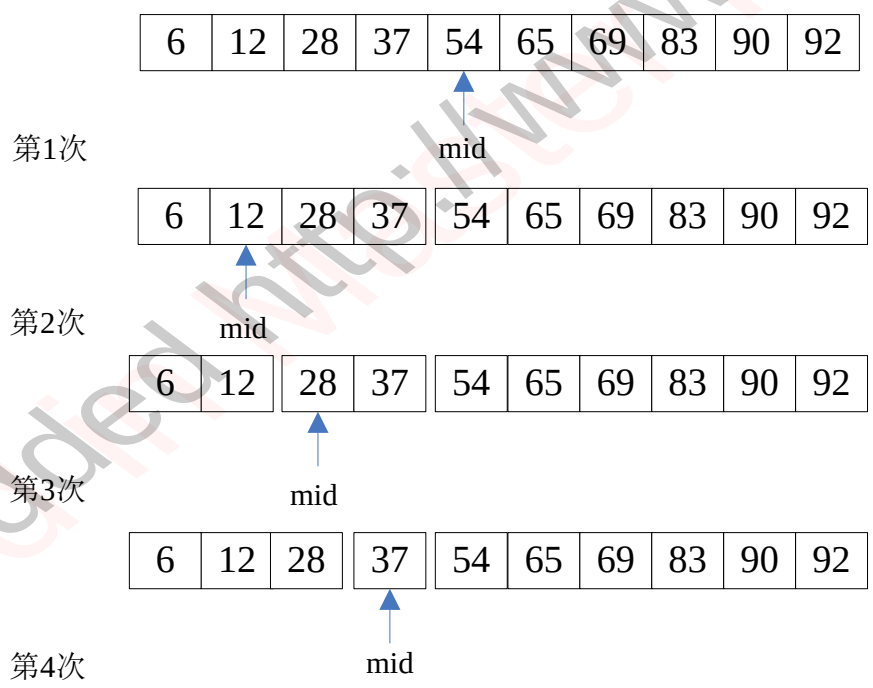
- **折半查找**

  又称为二分查找。这种查找方法要求查找表的数据是线性结构保存，并且还要求查找表中的数据是按关键字由小到大有序排列。



# 5.2 简单查找

- 折半查找



# 5.3 二叉排序树

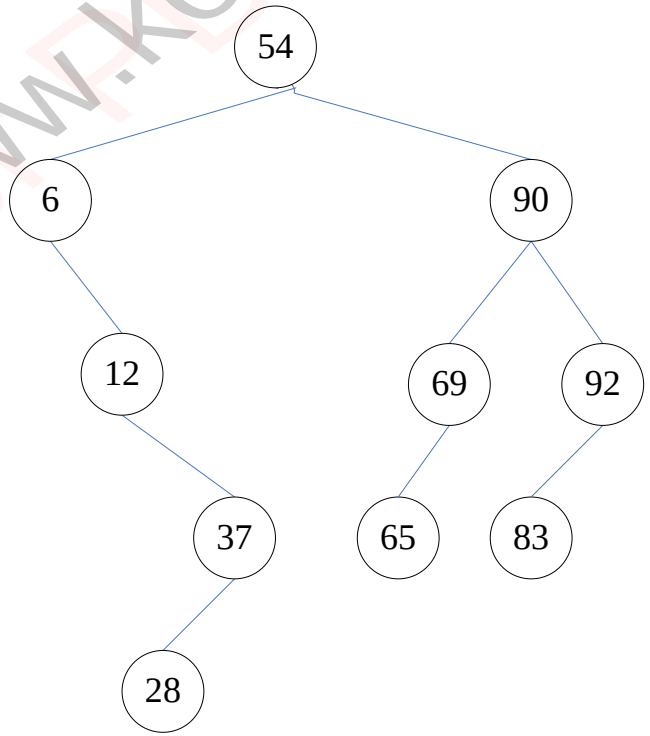
## 5.3.1 二叉排序树的定义

二叉排序数或者是一棵空树，或者是一棵具有以下性质的二叉树：

（1）若它有左子树，则左子树上所有结点的数据均小于根结点的数据。

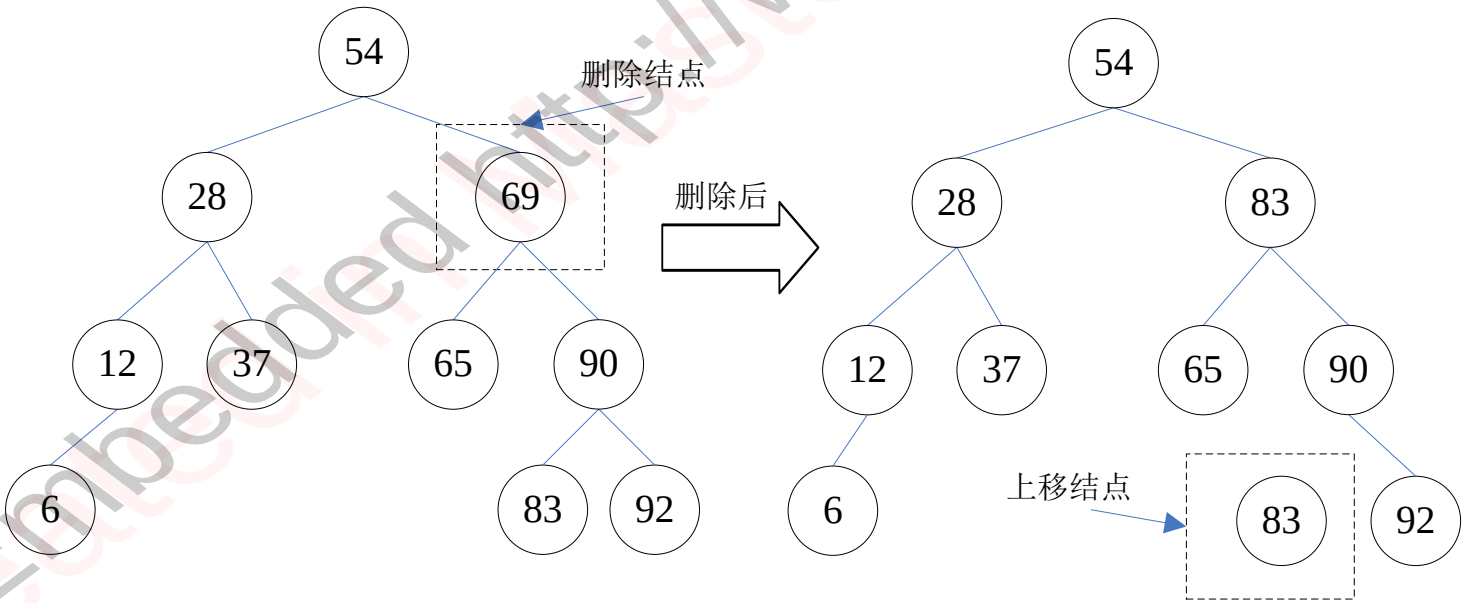
（2）若它有右子树，则右子树上所有结点的数据均大于根结点的数据。

（3）左、右子树本身又各是一棵二叉排序树。



# 5.3 二叉排序树

- 插入结点
- 查找结点
- 删除结点

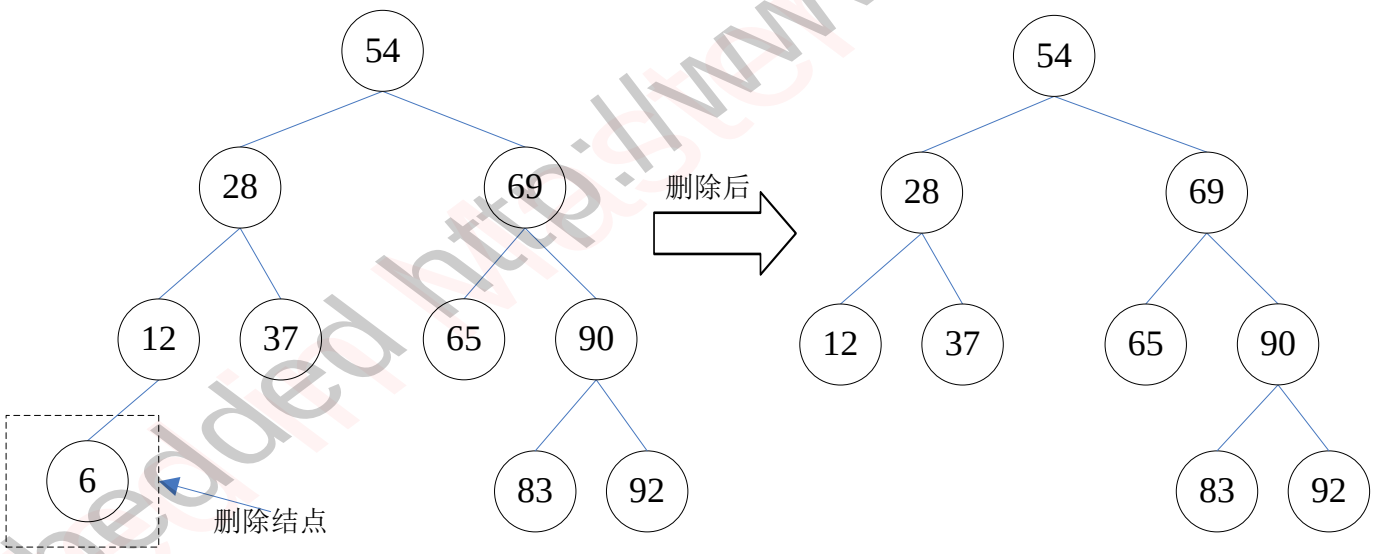


# 5.3 二叉排序树

- 插入结点
- 查找结点
- 删除结点

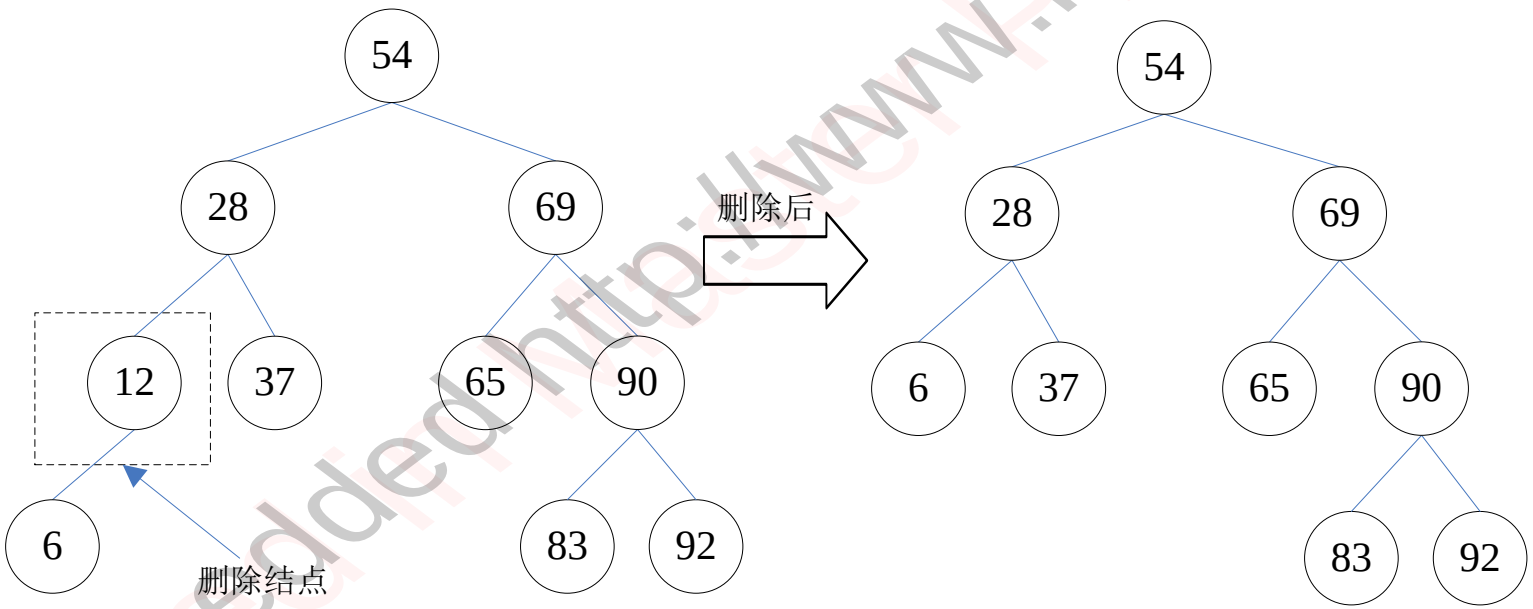
# 5.3 二叉排序树

- 删除叶结点



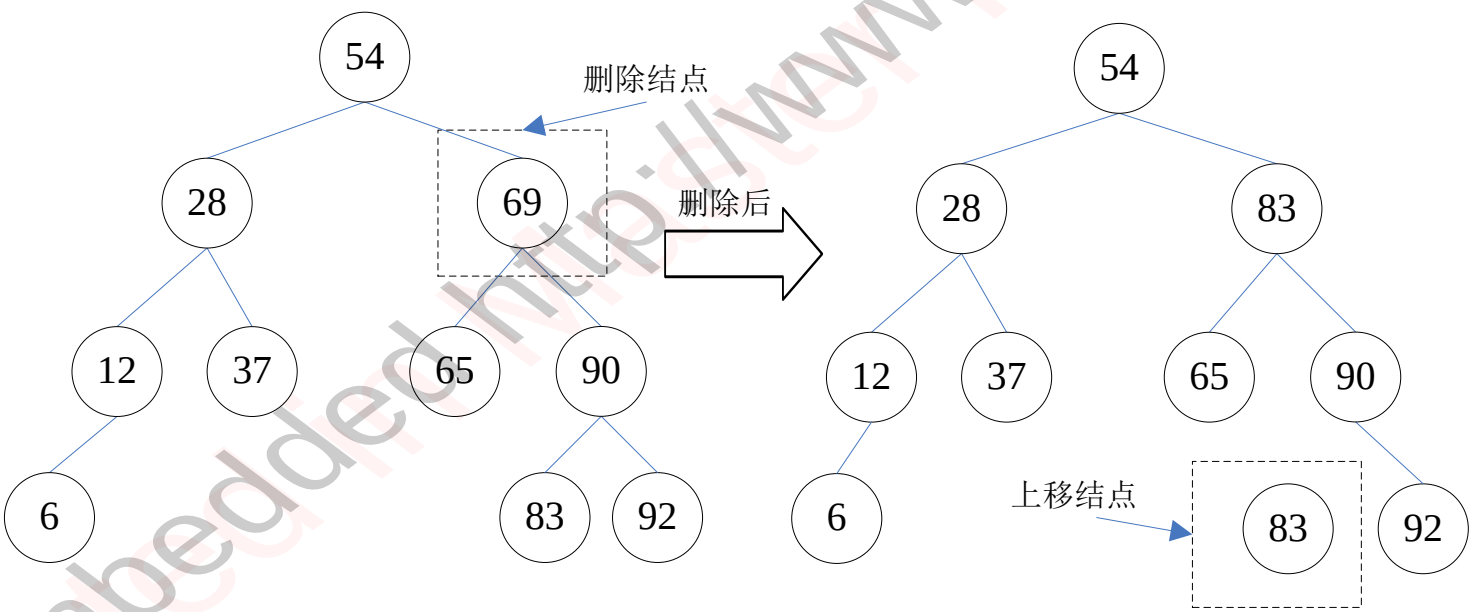
# 5.3 二叉排序树

- 删除无右子树结点



# 5.3 二叉排序树

- 删除有左右子树结点



# 5.4 索引查找

## 5.4.1 索引的概念

- 主表和索引表
- 创建索引的一般过程
- 索引的特点

# 5.4 索引查找

## 5.4.1 索引查找算法

- **在索引表中进行查找**

  引查找的过程是：

  （1）首先根据给定的关键字key，按定义的函数计算出索引值index1，在索引表上查找出索引值等于index1的索引项，以确定对应子表在主表中的开始位置和长度，

  （2）接着根据从索引表中获取的开始序号start，在主表指定位置（即子表的开始处）顺序查找关键字key。

- **向主表中插入数据**

  在线性表的索引存储结构上进行插入和删除运算的算法，与查找算法类似，其具体过程如下：

  （1）根据待插入元素的值查找索引表，确定出对应的子表。

  （2）接着，根据待插入元素的关键字，在该子表中做插入元素的操作。

  （3）插入完成后，修改索引表中的相应子表的长度。

# 5.5 哈希表

## 5.5.1 哈希表概述

哈希表的基本思想是：以线性表中每个元素的关键字key为自变量，通过一定的函数关系h(key)计算出函数的值，把这个值作为数组的下标，将元素存入对应的数组元素中。

函数h(key)称为哈希函数，函数的值称为哈希地址。

**线性表：** 69, 65, 90, 37, 92, 6, 28, 54

**哈希函数：** h(key)=key % 13

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 65 | 92 | 28 | | 69 | | 6 | | | | | 37 | 90 |

# 5.5 哈希表

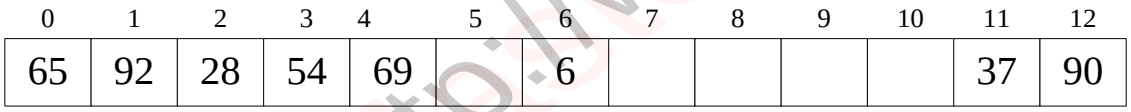
## 5.5.2 构造哈希函数

- 直接定址法
- 除法取余法
- 数字分析法
- 平方取中法
- 折叠法

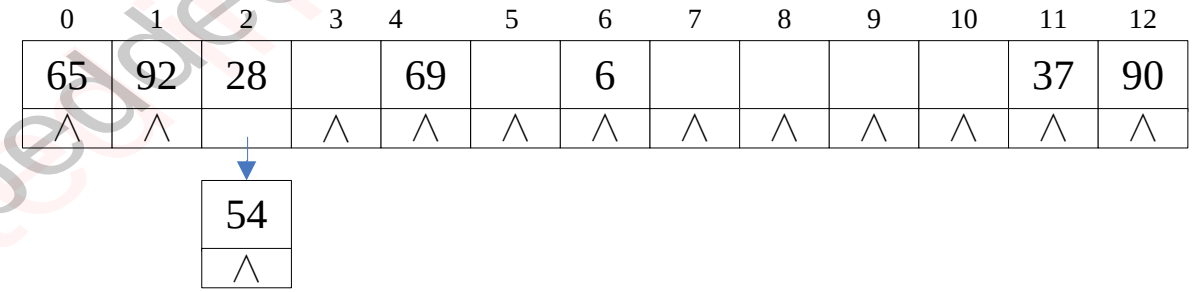
# 5.5 哈希表

## 5.5.3 处理冲突

- 开放地址法
  - 线性探测法
  - 双哈希函数探测法




- 链接法



性格决定命运，专注成就人生

# 零基础学算法

## 第6章：数学问题

# 课程安排

- 6.1 有趣的整数
- 6.2 素数
- 6.3 阶乘
- 6.4 求π的近似值
- 6.5 方程求解
- 6.6 矩阵的运算
- 6.7 一元多项式的运算

# 6.1 有趣的整数

## 6.1.1 完数

如果一个数恰好等于其因子之和，这个数就称为完数。

**6=1＋2＋3**

**28=1＋2＋4＋7＋14**

求10000以内的所有完数的过程：

（1）则用n去除以1~n之间的所有整数，将能整除的被除数保存到一个数组中，作为n的一个因子。

（2）用数n减去该因子，以方便计算各因子之和是否正好等于n。

（3）继续重复步骤1和步骤2，直至将所有整数除完为止。

（4）最后判断各因子之和是否等于数n，若相等，则数n为完数，输出该数和各因子。

# 6.1 有趣的整数

## 6.1.2 亲密数

假设有a、b两个数，若a的所有因子之和等于b的所有因子之和，并且a不等于b，则称a和b是一对亲密数。如284和220就是一对亲密数。

若要找出10000以内的亲密数，可使用以下算法：

（1）对每一个数a，将其因子分解出来，并将因子保存到一个数组中，再将因子之和保存到变量b1。

（2）将因子之和b1再进行因子分解，并将因子保存到一个数组中，将因子之和保存到变量b2中。

（3）若b2等于a，并且b1不等于b2，则找到一对亲密数为a和b1，可将其输出。

（4）重复步骤（1）~（3），即可找出指定范围的亲密数。

# 6.1 有趣的整数

## 6.1.3 水仙花数

一个三位数，若数值等于各位数字的三次幂之和，就称为“水仙花数”。

$$153=1^3+5^3+3^3$$

$$371=3^3+7^3+1^3$$

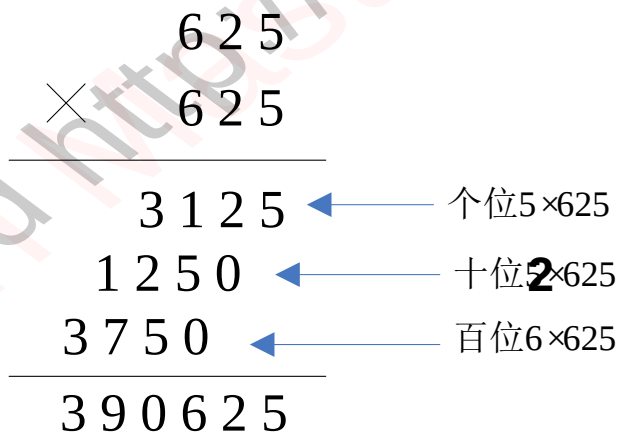
$$370=3^3+7^3+0^3$$

$$407=4^3+0^3+7^3$$

# 6.1 有趣的整数

## 6.1.4 自守数

所谓自守数，是指一个数的平方的尾数等于该数自身的自然数。例如：6的平方等于36，尾数是6，所以6是自守数；25的平方等于625，尾数是25，所以25是自守数。



# 6.1 有趣的整数

## 6.1.5 最大公约数最小公倍数

- **欧几里德算法**

欧几里德算法采用辗转相除的方法来求最大公约数，这是计算两个数最大公约数的传统算法

其算法思路为：

（1）对于已知两数m、n，使m>n；

（2）m除以n得余数r；

（3）若r=0，则n为求得的最大公约数，跳至第（5）求最小公倍数；否则执行第4步；

（4）将n的值保存到m中，将r的值保存到n中，重复执行步骤（2）和（3）。

（5）有了两数的最大公约数，则最小公倍数就很简单了，将两数相乘的积除以最大公约数即可。

# 6.1 有趣的整数

## 6.1.5 最大公约数最小公倍数

- **Stein算法**

Stein算法只有整数的移位和加减法，而不需要进行除法和取模运算，这将提高算法的执行效率。

Stein算法如下（求a、b两数的最大公约数）：

（1）首先判断a或b是否为0，若a=0，b就是最大公约数；若b=0，a就是最大公约数，完成计算操作。

（2）设$a_1$=a、$b_1$=b和$c_1$=1。

（3）判断$a_n$和$b_n$是否为偶数，若都是偶数，则使$a_{n+1}=a_n/2$，$b_{n+1}=b_n/2$，$c_{n+1}=c_n*2$。

（4）若$a_n$是偶数，$b_n$是奇数，则使$a_{n+1}=a_n/2$，$b_{n+1}=b_n$，$c_{n+1}=c_n$。

（5）若$b_n$是偶数，$a_n$是奇数，则使$b_{n+1}=b_n/2$，$a_{n+1}=a_n$，$c_{n+1}=c_n$。

（6）若$a_n$和$b_n$都是奇数，则使$a_{n+1}=|a_n-b_n|$，$b_{n+1}=\min(a_n,b_n)$，$c_{n+1}=c_n$。

（7）n累加1，跳转到第3步进行下一轮运算。

# 6.2 素数

## 6.2.1 求素数

所谓素数，是指除了1和自身之外，没有别的因数的数。除了1和自身外，还有别的因数的数是合数。1既不是素数也不是合数。素数的分布是没有规律的。如：101、401、601、701都是素数，但上下面的301和901却是合数。

要求N是不是素数，可用N逐个除以2~N-1之间的数，若某个数能被整除，则表示该数不是素数

# 6.2 素数

## 6.2.2 回文素数

所谓回文数，是指一个多位数在按位读时，无论从左向右还是从右向左倒序读取，其结果都是一样的特征。例如：11、22、101、111、818、12321等。

- 回文数素数
- 平方回文数

# 6.2 素数

## 6.2.3 哥德巴赫猜想

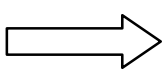
所谓哥德巴赫猜想，是指哥德巴赫在教学中发现，每个不小于6的偶数都是两个素数之和。大家都相信这个猜想是正确的，但不能证明。

对于哥德巴赫猜想的验证，算法很简单，其基本思路是：设n为大于等于6的一个偶数，可将其分解为n1和n2两个数，分别检查n1和n2是否为素数，如都是，则在该数得到验证。若n1不是素数，就不必再检查n2是否素数。先从n1=2开始，检验n1和n2（n2=n-n1）是否素数。然后使n1+2再检验n1、n2是否素数……，直到n1=n/2为止。

# 6.3 阶乘

- **6.3.1 用递归计算阶乘**
- **6.3.2 大数阶乘**

```
  39916800
×        12
----------
  79833600
  39916800
----------
 479001600
```

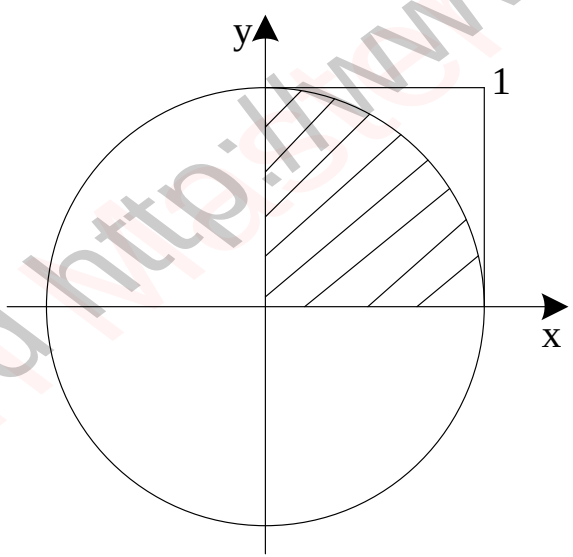


```
          12
×39916800
----------
          00
         00
        96
      72
     12
   108
 108
36
----------
479001600
```

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 11的阶乘结果 | 0 | 3 | 9 | 9 | 1 | 6 | 8 | 0 | 0 |
| 乘以12的结果 | 0 | 36 | 108 | 108 | 12 | 72 | 96 | 0 | 0 |
| 进位后的结果 | 4 | 7 | 9 | 0 | 0 | 1 | 6 | 0 | 0 |

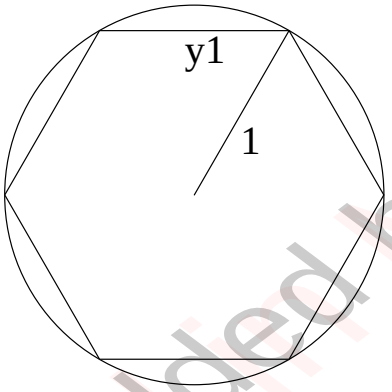
# 6.4 求π的近似值

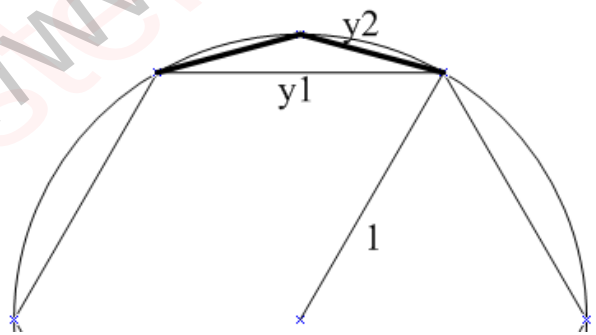
## 6.4.1 概率法



# 6.4 求π的近似值

## 6.4.2 割圆法



# 6.4 求π的近似值

## 6.4.3 公式法

公式 1：$$\pi = 2 \times \frac{2}{\sqrt{2}} \times \frac{2}{\sqrt{2+\sqrt{2}}} \times \frac{2}{\sqrt{2+\sqrt{2+\sqrt{2}}}} \times \cdots$$

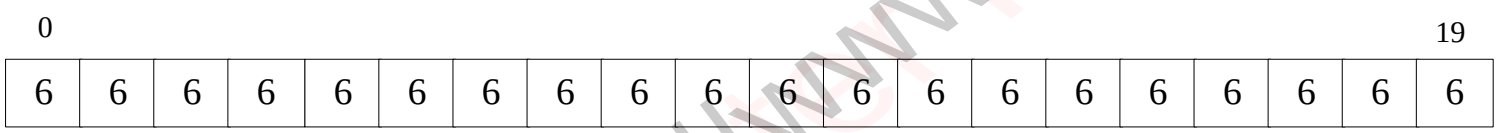
公式 2：$$\frac{\pi}{2} = 1 + \frac{1}{3} + \frac{1}{3} \times \frac{2}{5} + \frac{1}{3} \times \frac{2}{5} \times \frac{3}{7} + \frac{1}{3} \times \frac{2}{5} \times \frac{3}{7} \times \frac{4}{9} + \cdots$$

公式 3：$$\frac{\pi}{2} = \frac{2}{1} \times \frac{2}{3} \times \frac{4}{3} \times \frac{4}{5} \times \frac{6}{5} \times \frac{6}{6} \times \cdots$$

$$\pi = 2 + \frac{2}{3} + \frac{2}{3} \times \frac{2}{5} + \frac{2}{3} \times \frac{2}{5} \times \frac{3}{7} + \frac{2}{3} \times \frac{2}{5} \times \frac{3}{7} \times \frac{4}{9} + \cdots$$

# 6.4 求π的近似值

## 6.4.4 计算任意位数的π




$$\frac{2}{3}\times\frac{2}{5}$$

# 6.5 方程求解

## 6.5.1 高斯消元法解线性方程组

$$\begin{cases} 5x_1 + 3x_2 - 4x_3 = 0 \\ 2x_1 + 7x_2 + 6x_3 = -4 \\ -x_1 + 4x_2 + 5x_3 = -5 \end{cases}$$

$$A = \begin{bmatrix} 5 & 3 & -4 & 0 \\ 2 & 7 & 6 & -4 \\ -1 & 4 & 5 & -5 \end{bmatrix}$$

$$a_{ij} = a_{ij} - \frac{a_{ik}}{a_{kk}} a_{kj}$$

$$x_n = a_{nn+1} / a_{nn}$$

$$x_k = (a_{kn+1} - \sum_{j=k+1}^{n} a_{kj} x_j) / a_{kk}$$

# 6.5 方程求解

## 6.5.2 二分法解非线性方程

二分法又称对分法，是最简单的求解一元非线性方程根的算法之一。其基本思想是：将含根区间*I*逐次分半缩小，得到一个区间长度以1/2的比例减小的含根区间序列$I_1$、$I_2$，……，在给定根的误差界时，利用$\{I_k\}$的长度趋于0的特点，可得到在某个区间*I*中求得满足要求的近似根。

$$2x^3 - 5x - 1 = 0$$

# 6.5 方程求解

## 6.5.3 牛顿迭代法解非线性方程

牛顿迭代法是牛顿在17世纪提出的一种在实数域和复数域上近似求解方程的方法。迭代法的基本思想就是构造一串收敛到解的序列，即建立一种从已有近似解来计算新的近似解的迭代式，然后选取方程的某个初始近似值$x_0$代入迭代式，反复这个过程使得到的根逐渐逼近于真实根，直到满足精度为止。

$$x^4 - 3x^3 + 1.5x^2 - 4 = 0$$

导函数：$$4x^3 - 9x^2 + 3x = 0$$

# 6.6 矩阵运算

## 6.6.1 矩阵加法和乘法运算

- **1. 矩阵加法运算**

$$\begin{bmatrix} 5 & 8 & 3 \\ 11 & 0 & 5 \\ 12 & 2 & 7 \end{bmatrix} + \begin{bmatrix} 1 & 18 & 7 \\ 2 & 11 & 15 \\ 10 & 3 & 4 \end{bmatrix} = \begin{bmatrix} 6 & 26 & 10 \\ 13 & 11 & 20 \\ 22 & 5 & 11 \end{bmatrix}$$

- **2. 矩阵乘法运算**

$$\begin{bmatrix} 5 & 8 & 3 \\ 11 & 0 & 5 \end{bmatrix} \times \begin{bmatrix} 1 & 18 \\ 2 & 11 \\ 10 & 3 \end{bmatrix} = \begin{bmatrix} 5\times1+8\times2+3\times10 & 5\times18+8\times11+3\times3 \\ 11\times1+0\times2+5\times10 & 11\times18+0\times11+5\times3 \end{bmatrix} = \begin{bmatrix} 51 & 187 \\ 61 & 213 \end{bmatrix}$$

# 6.6 矩阵运算

## 6.6.2 多维矩阵转一维矩阵

二维矩阵转一维矩阵下标公式（以行为主）：

loc=row*每行元素数量+column

二维矩阵转一维矩阵下标公式（以列为主）：

loc= column*每列元素数量+row

$$\begin{bmatrix} 5 & 8 & 3 \\ 11 & 0 & 5 \\ 12 & 2 & 7 \end{bmatrix} \Rightarrow \begin{bmatrix} 5 & 8 & 3 & 11 & 0 & 5 & 12 & 2 & 7 \end{bmatrix}$$

(a)以行为主

$$\begin{bmatrix} 5 & 8 & 3 \\ 11 & 0 & 5 \\ 12 & 2 & 7 \end{bmatrix} \Rightarrow \begin{bmatrix} 5 & 11 & 12 & 8 & 0 & 2 & 3 & 5 & 7 \end{bmatrix}$$

(b)以列为主

# 6.6 矩阵运算

## 6.6.3 逆矩阵

对于n阶方阵$A$，如果存在一个n阶方阵$B$，使得$AB=BA=C$（$C$为n阶单位矩阵），则把方阵$B$称为$A$的逆矩阵（简称逆阵）记作$A^{-1}$，即$B=A^{-1}$

$$A=\begin{bmatrix}1 & 3\\ 2 & 5\end{bmatrix} \quad B=\begin{bmatrix}-5 & 3\\ 2 & -1\end{bmatrix}$$

$$AB=\begin{bmatrix}1 & 3\\ 2 & 5\end{bmatrix}\times\begin{bmatrix}-5 & 3\\ 2 & -1\end{bmatrix}=\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} \quad BA=\begin{bmatrix}-5 & 3\\ 2 & -\end{bmatrix}\times\begin{bmatrix}1 & 3\\ 2 & 5\end{bmatrix}=\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$$

# 6.6 矩阵运算

## 6.6.4 稀疏矩阵

将表示稀疏矩阵的非0元素的三元组按行优先（或列优先）的顺序排列，并依次存放在数组中，这种稀疏矩阵的顺序存储结构称为三元组表。

$$A=\begin{bmatrix} 0 & 5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 \\ 0 & 0 & 9 & 7 & 0 & 0 \\ 0 & 6 & 0 & 0 & 0 & 0 \\ 8 & 0 & 0 & 0 & 0 & 8 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 9 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(a)稀疏矩阵

| | | |
|---|---|---|
| 6 | 5 | 8 |
| 0 | 1 | 5 |
| 1 | 5 | 3 |
| 2 | 2 | 9 |
| 2 | 3 | 7 |
| 3 | 1 | 6 |
| 4 | 0 | 8 |
| 4 | 5 | 8 |
| 6 | 0 | 9 |

(b)三元组

# 6.7 一元多项式的运算

## 6.7.1 多项式加法

一元多项式加法运算的规则非常简单，就是将具有相等幂项的系数相加即可得到合并后的多项式。若某个幂只存在于一个多项式中，则直接合并到结果中。

$$P_1(x) = 3 + 5x - 6x^3 + 2x^5$$

$$P_2(x) = 7x^2 + x^3 - x^4$$

**结果：**

$$P_3(x) = P_1(x) + P_2(x) = 3 + 5x + 7x^2 - 5x^3 - x^4 + 2x^5$$

# 6.7 一元多项式的运算

## 6.7.2 多项式减法

与多项式加法类似，两个一元多项式相减也只是将具有相等幂项的系数相减即可，若某个幂只存在于被减项，则直接合并到结果中，若某个幂只存在于减项，则将其系数符号取反（乘以-1），再合并到结果中。

性格决定命运，专注成就人生

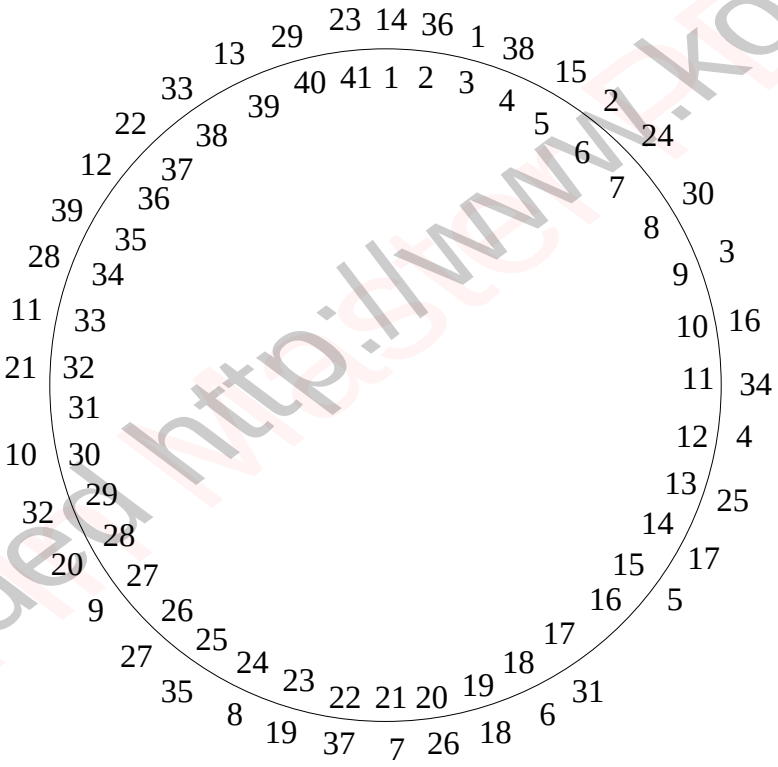
# 零基础学算法

## 第7章：数据结构问题

# 课程安排

- 7.1 约瑟夫环
- 7.2 大整数四则运算
- 7.3 进制转换
- 7.4 括号匹配
- 7.5 中序式转后序式
- 7.6 停车场管理
- 7.7 迷宫求解
- 7.8 LZW压缩的实现

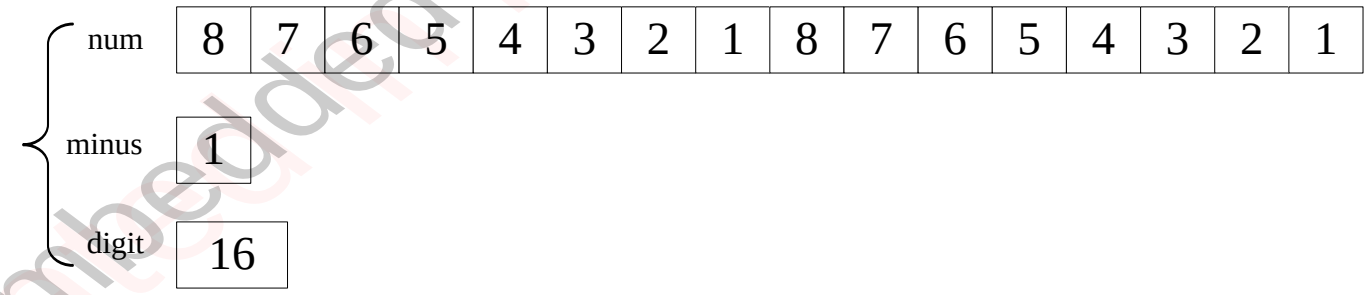
# 7.1 约瑟夫环



# 7.2 大整数四则运算

## 7.2.1 使用数组进行大整数运算

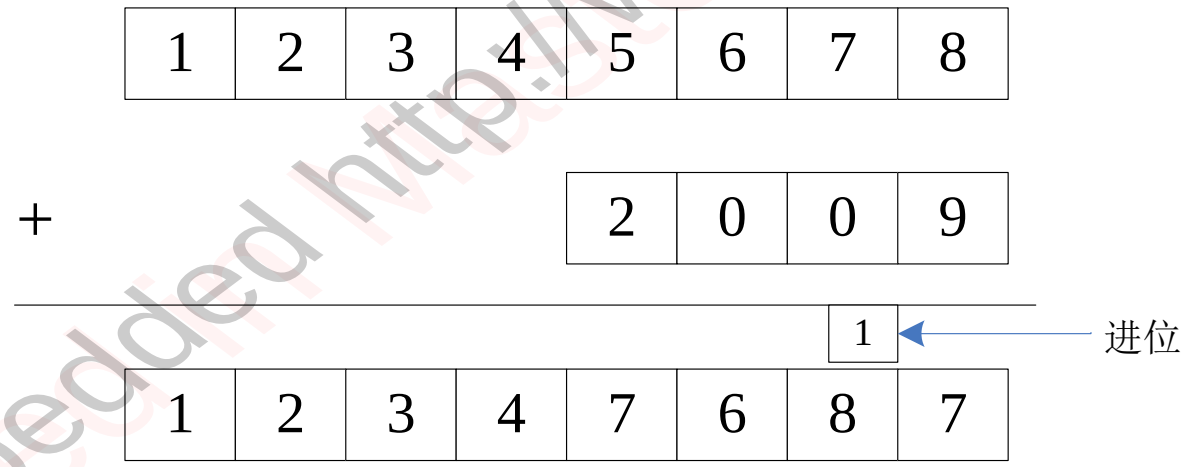
- 设计大整数的存储结构
- 输入/输出大整数
- 比较大整数的大小
- 进行加减乘除运算



# 7.2 大整数四则运算

## 7.2.1 使用数组进行大整数运算

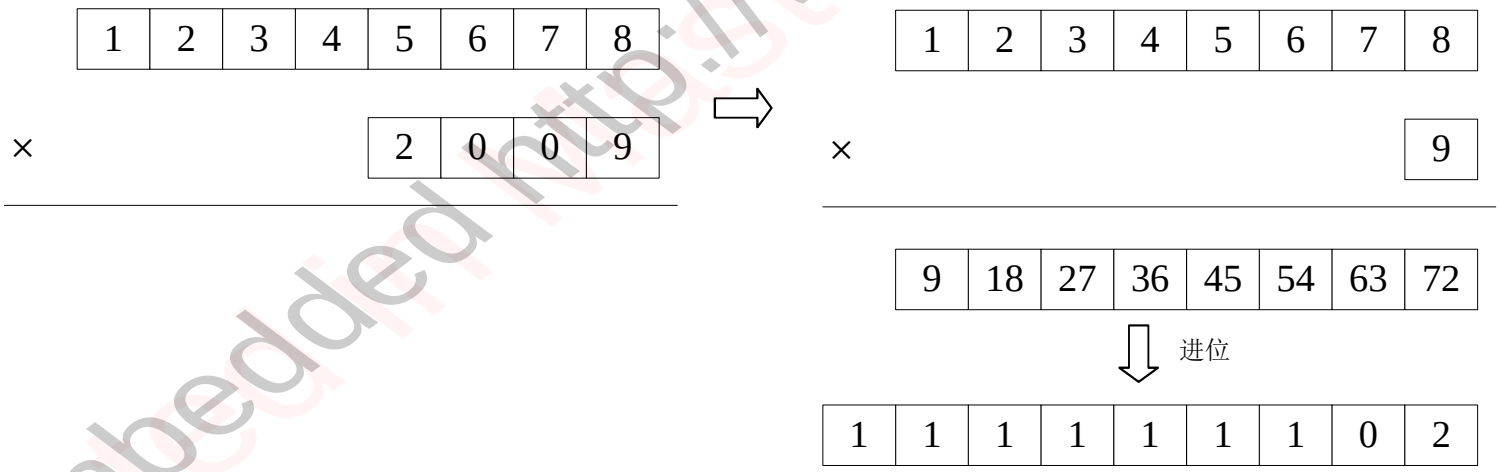
- 加法运算



# 7.2 大整数四则运算

## 7.2.1 使用数组进行大整数运算

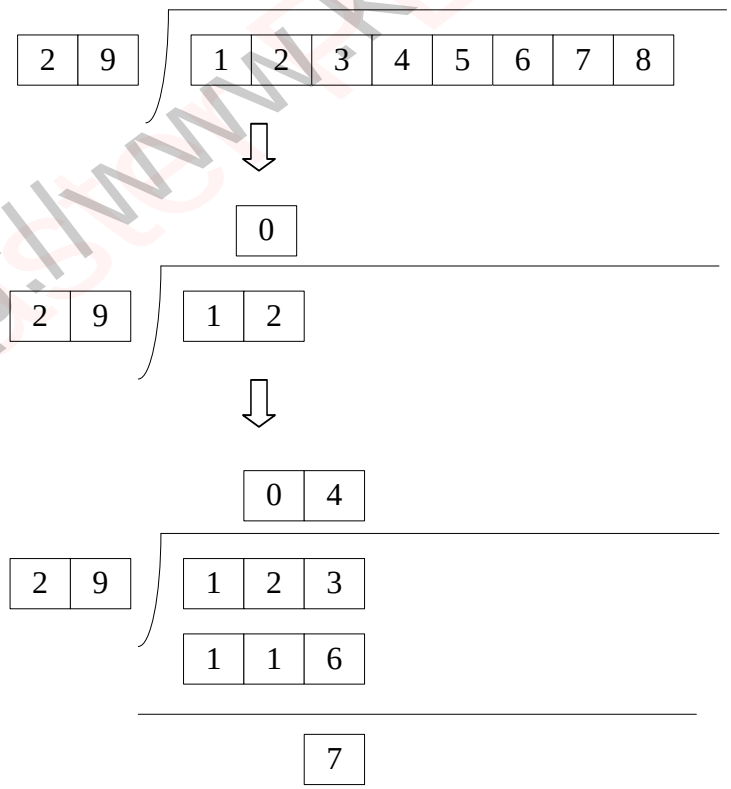
- 乘法运算



# 7.2 大整数四则运算

## 7.2.1 使用数组进行大整数运算

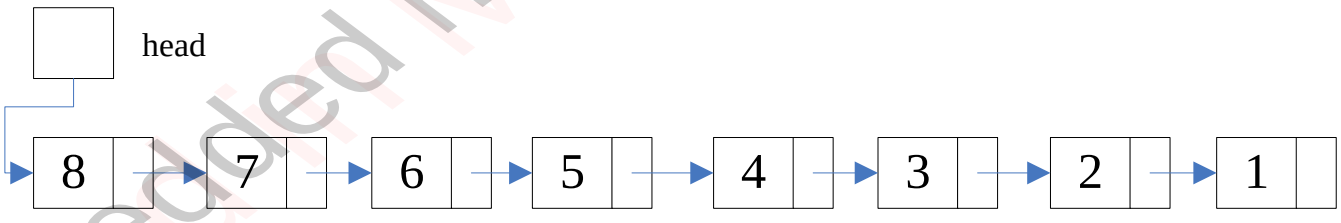
- 除法运算



# 7.2 大整数四则运算

## 7.2.2 使用链表进行大整数运算

- 设计大整数的链表结构
- 输入/输出大整数
- 进行加减运算



# 7.3 进制转换

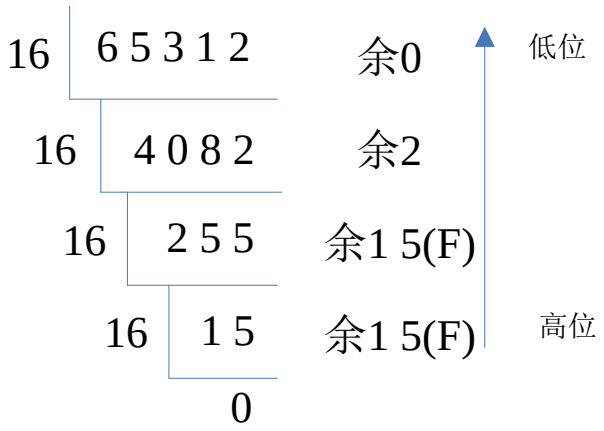
- **任意进制转换为十进制**

对于任意进制转换为十进制的操作，只需要将该进制的数据按权展开，然后相加即可。

$$
\begin{aligned}
(101101)_2 &= 1\times 2^5 + 0\times 2^4 + 1\times 2^3 + 1\times 2^2 + 0\times 2^1 + 1\times 2^0 \\
&= 32 + 8 + 4 + 1 \\
&= 45
\end{aligned}
$$

- **十进制转换为任意进制**

十进制数转换为其他任意进制时，采用反复除以某进制的基数，取其余数作为对应进制的数据，并且最先得到的是该进制的低位，最后得到的才是该进制的高位。



# 7.4 括号匹配

要检查某一表达式的括号是否匹配，可从左向右扫描表达式中的每一个字符，若字符为左、右括号，则进行匹配操作，可分两种情况：

若是左括号，则将其位置序号进入栈中。

若是右括号，则从栈中弹出一个左括号与之匹配。如果栈已为空，表示多了一个右括号。

# 7.5 中序式转后序多

- **1. 后序表达式**

中序表达式：(a+b)*(c+d)　　转为后序表达式：ab+cd+*

- **2.中序式转 后序式的过程**

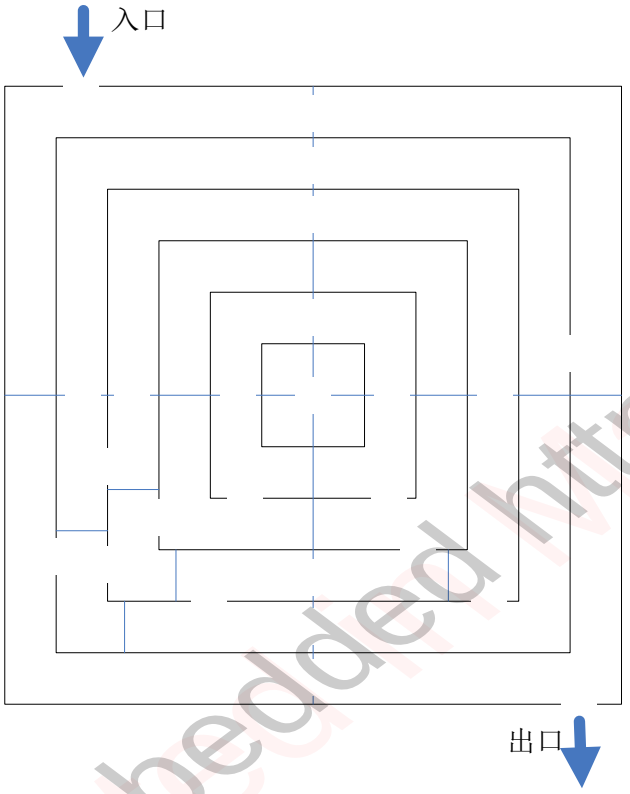
使用栈来进行转换，逐个取出中序表达式中的字符，若是运算数，则直接将其输出，若是运算符，则需根据运算符的优先级进行判断：

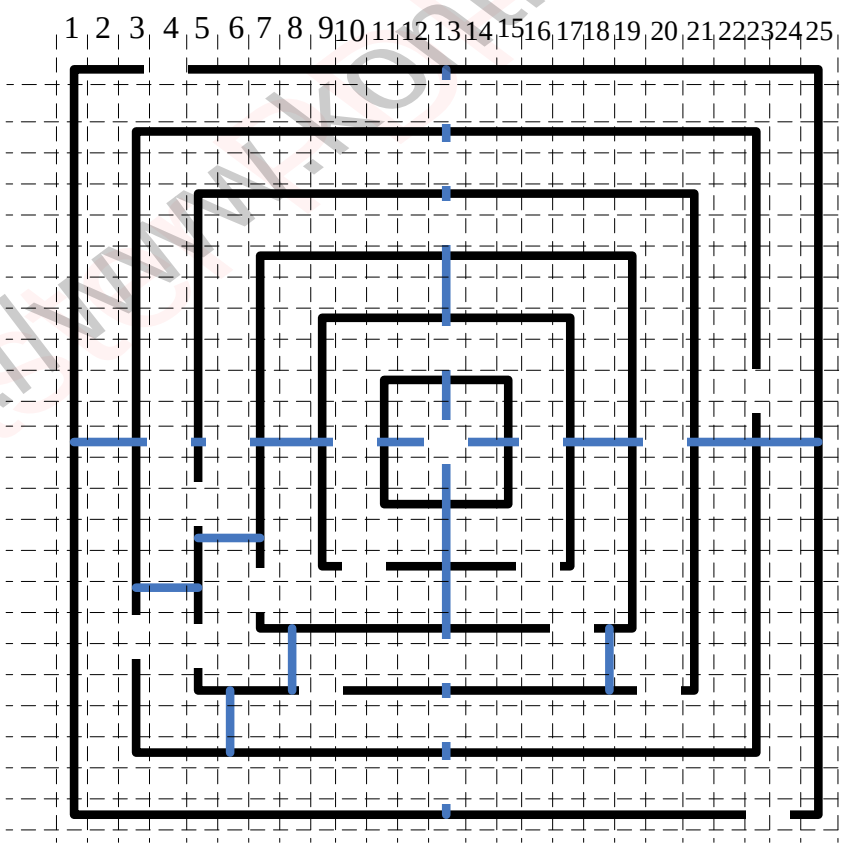
  - 若是左括号，将其入栈；
  - 若是“+、-、*、/”运算符，用当前运算符与栈顶运算符比较，若栈顶运算符优先级大，则弹出栈顶运算符。接着再将当前运算符与栈顶运算符进行比较，这样不断循环，直到栈顶运算符的优先级比当前运算符相等或低时为止。接着将当前运算符入栈。
  - 若是右括号，则查看栈顶是否为左括号

# 7.6 停车场管理

该题的具体描述是：设停车场内只有一个可停放n辆汽车的狭长通道，且只有一个大门可供汽车进出。汽车在停车场内按车辆到达时的先后顺序，依次由北向南排列（大门在最南端，最先到达的第一辆车停放在车场的最北端），若停车场内已停满n辆汽车（即车位已满），则后来的汽车只能停在门外的过道上等候，一旦停车场内有车开走，则排在过道上的第一辆车即可开入；当停车场内某辆车要离开时，由于停车场是狭长的通道，在它之后开入车场的车辆必须先退出车场为它让路，待该辆车开出大门外后，为它让路的车辆再按原次序进入车场（在这里假设汽车不能从便道上开走）。每辆车按其在停车场停留的时间付费，车辆停在停车场内时需要计时收费，而停在过道上不收费。

# 7.7 迷宫求解



# 7.8 LZW压缩的实现

压缩字符串“One One One ”的过程：

表 7-1　LZW算法压缩数据过程

| 执行步骤 | 标号 | 前缀 | 后缀 | 输出 |
|---|---|---|---|---|
| 1 | | O | | |
| 2 | 258 | O | n | O |
| 3 | 259 | n | e | n |
| 4 | 260 | e | □ | e |
| 5 | 261 | □ | O | □ |
| 6 | | O | n | |
| 7 | 262 | 258 | e | 258 |
| 8 | | e | □ | |
| 9 | 263 | 260 | O | 260 |
| 10 | | O | n | |
| 11 | | 258 | e | |
| 12 | | 262 | | 262 |

性格决定命运，专注成就人生

# 零基础学算法

## 第8章：算法经典问题

# 课程安排

- **8.1** 不定方程问题
- **8.2** 推算问题
- **8.3** 魔术方阵
- **8.4** 智力趣题
- **8.5** 趣味游戏

# 8.1 不定方程问题

## 8.1.1 百钱买百鸡

公鸡5文钱1只，母鸡3文钱1只，小鸡3只1文钱，要求用100文钱买100只鸡，求公鸡、母鸡和小鸡各应该买多少只？

x+y+z=100

5x+3y+z/3=100

设一个整数参数k，就有：

x=4k

y=25 - 7k

z=75 + 3k

# 8.1 不定方程问题

## 8.1.2 存钱利息最大化

表 8-1 存款基准利率

| | |
|---|---|
| 活期存款 | 0.36 |
| 三个月 | 1.71 |
| 半年 | 1.98 |
| 一年 | 2.25 |
| 二年 | 2.79 |
| 三年 | 3.33 |
| 五年 | 3.60 |

$$\text{第1期}: 10000\times(1+5\times3.6/100)$$

$$\text{第2期}: 10000\times(1+5\times3.6/100)\times(1+5\times3.6/100)$$

$$\text{第3期}: 10000\times(1+5\times3.6/100)\times(1+5\times3.6/100)\times(1+5\times3.6/100)$$

$$\text{第4期}: 10000\times(1+5\times3.6/100)\times(1+5\times3.6/100)\times(1+5\times3.6/100)\times(1+5\times3.6/100)$$

$$=10000\times(1+5\times3.6/100)^4$$

# 8.1 不定方程问题

## 8.1.3 求阶梯数

有一次，大科学家爱因斯坦给他的朋友出了这样一道数学题：在你面前有一条长长的阶梯。如果你每步跨2阶，那么最后剩一阶。如果你每步跨3阶，那么最后剩2阶。如果你每步跨5阶，最后剩4阶，如果你每步跨6阶，最后剩5阶。只有当你能够每步跨7阶时，才正好到头，一阶也不剩。你想一想，这阶梯到底有多少阶？

# 8.2 推算问题

## 8.2.1 猴子吃桃

一只猴子摘了一堆桃子，它每天吃了其中的一半然后再多吃了一个，直到第10天，它发现只有1个桃子了，问它第一天摘了多少个桃子？

$a_1=(a_2+1)\times 2;$

$a_2=(a_3+1)\times 2;$

……

$a_9=(a_{10}+1)\times 2;$

$a_{10}=1;$

# 8.2 推算问题

## 8.2.2 舍罕王的赏赐

这是一个典型的等比数列求和的问题。

第1格：1粒；

第2格：1×2=2粒；

第3格：1×2×2=4粒；

第4格：1×2×2×2=8粒；

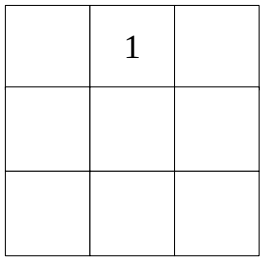
……

将每一格的麦子粒数加起来：

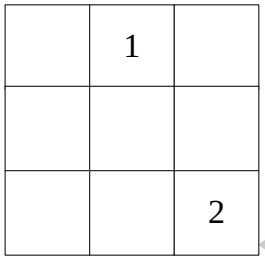
sum=1+2+4+8+……

# 8.3 魔术方阵

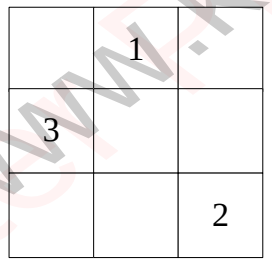
## 8.3.1 简捷连续填数法



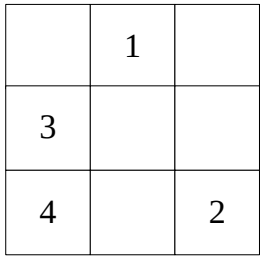

(a)



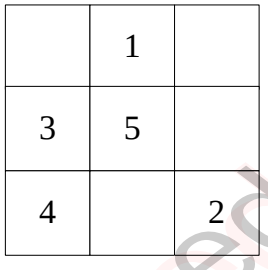

(b)



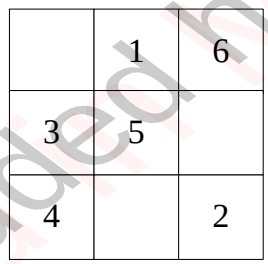

(c)



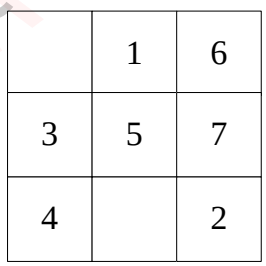

(d)



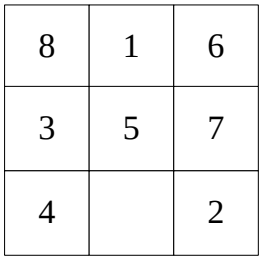

(e)



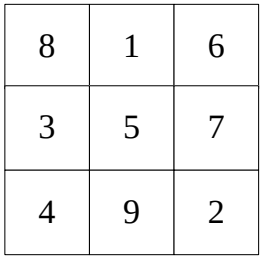

(f)




(g)




(h)




(i)

# 8.3 魔术方阵

## 8.3.2 双向翻转法

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 |

(a)

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 8 | 7 | 6 | 5 |
| 12 | 11 | 10 | 9 |
| 13 | 14 | 15 | 16 |

(b)

| | | | |
|---|---|---|---|
| 1 | 14 | 15 | 4 |
| 8 | 11 | 10 | 5 |
| 12 | 7 | 6 | 9 |
| 13 | 2 | 3 | 16 |

(c)

# 8.3 魔术方阵

## 8.3.3 井字调整法

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 |

(a)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 33 | 34 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 |
| 18 | 14 | 22 | 21 | 17 | 13 |
| 24 | 20 | 16 | 15 | 23 | 19 |
| 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 3 | 4 | 35 | 36 |

(b)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 32 | 33 | 34 | 5 | 6 |
| 12 | 8 | 27 | 28 | 11 | 7 |
| 13 | 20 | 22 | 21 | 17 | 18 |
| 24 | 14 | 16 | 15 | 23 | 19 |
| 30 | 26 | 9 | 10 | 29 | 25 |
| 31 | 2 | 3 | 4 | 35 | 36 |

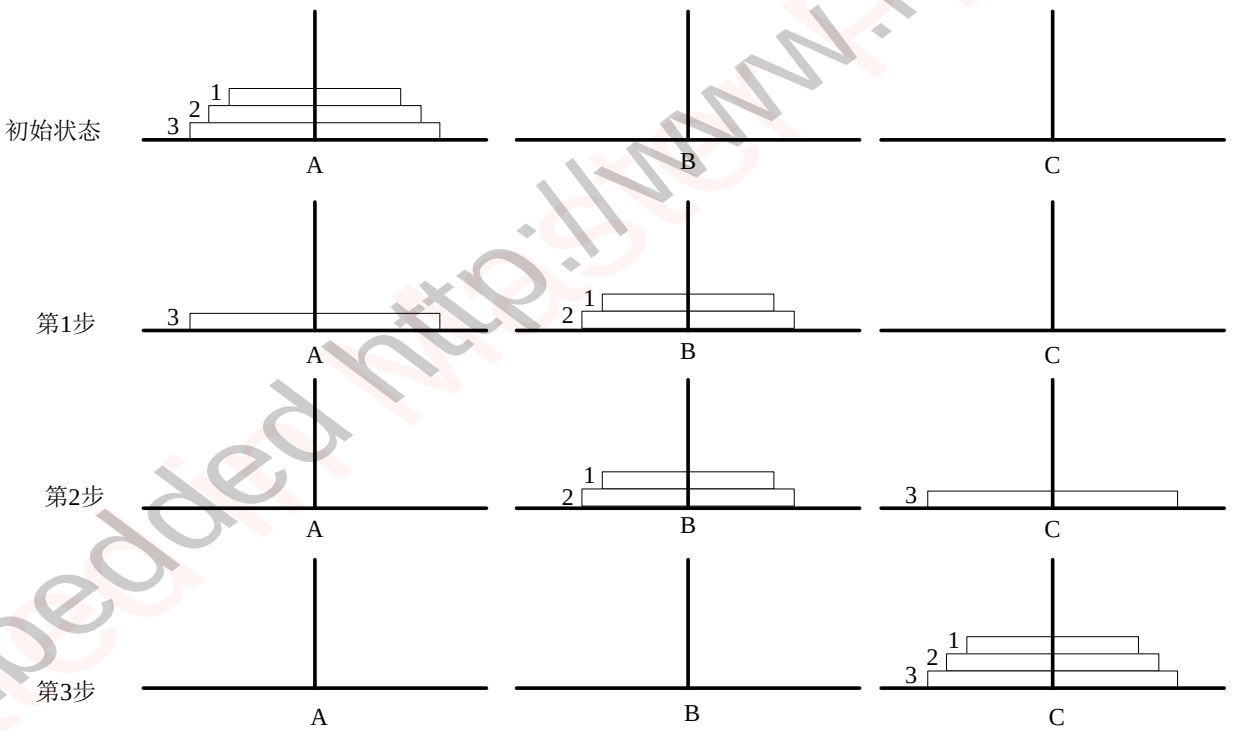
(c)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 32 | 33 | 34 | 5 | 6 |
| 30 | 8 | 28 | 27 | 11 | 7 |
| 13 | 20 | 22 | 21 | 17 | 18 |
| 24 | 23 | 16 | 15 | 14 | 19 |
| 12 | 26 | 9 | 10 | 29 | 25 |
| 31 | 2 | 3 | 4 | 35 | 36 |

(d)

# 8.4 智力趣题

## 8.4.1 汉诺塔



# 8.4 智力趣题

## 8.4.2 背包问题

有一个背包最多可装重量8公斤的物品，假设要用该背包装如下水果，要求使背包中装的物品的价值最大，应该装下列哪些物品才能达到要求？

各水果的重量和价值：

- 苹果：5公斤，40元；
- 梨：2公斤，12元；
- 桃：1公斤，7元；
- 葡萄：1公斤，8元；
- 香蕉：6公斤，48元。

# 8.4 智力趣题

## 8.4.3 马踏棋盘

国际象棋共有8行8列，共64个单元格，无论将马放于棋盘的哪个单元格，都可让马踏遍棋盘的每个单元格。要求编写程序实现马踏棋盘的过程，输出马走过的次序。

马只能走日字，当马位于棋盘中间位置时，马可以向8个方向跳动，当马位于棋盘的边或角时，马可以跳动的方向将少于8个。另外，当马所跳向的8个方向中的某一个或几个方向若已被马走过，也将跳至马下一步要走的位置。

# 8.4 智力趣题

## 8.4.4 八皇后问题

八皇后问题是一个古老而著名的问题，是回溯算法的典型例题。该问题是十九世纪著名的数学家高斯1850年提出：在国际象棋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，求有多少种摆放方法。

# 8.5 趣味游戏

## 8.5.1 取石子游戏

有n堆石子，每堆有若干石子，数量不一定相同，两人（游戏者与计算机）轮流从任一堆中拿走任意数量的石子，最后把石子全部拿走者为胜利方。

所谓“必负局”，是指把剩余的每一堆的数目都转化成二进制的数，然后把它们相加，进行不进位的加法（也就是异或运算），即0+0＝0、1+0＝1、0+1＝1、1+1＝0（不进位），如果所得和是0（多个0），那么此种局势称为“必负局”。

# 8.5 趣味游戏

## 8.5.2 生命游戏

生命游戏的规则很简单：假设平面上画许多方形网格，每个方格中放置一个生命细胞，生命细胞只有两种状态：“生”或“死”。对其中一个网格有上、下、左、右、左上、左下、右上、右下共8个相邻网格，根据这些网络中的细胞数量决定当前网格细胞的存活，游戏规则如下：

- 孤单死亡：若细胞的相邻网格中没有细胞存在，则该细胞在下一次状态中将死亡；
- 拥挤死亡：若细胞的相邻网格中细胞数量在4个（含4个）以上，则该细胞在下一次状态中将死亡；
- 复活：若细胞的相邻网格中细胞数量为3个，则将当前位置的细胞复活；
- 稳定：若细胞的相邻网格中细胞数量为2个，则将当前位置的细胞保持原状。

# 8.5 趣味游戏

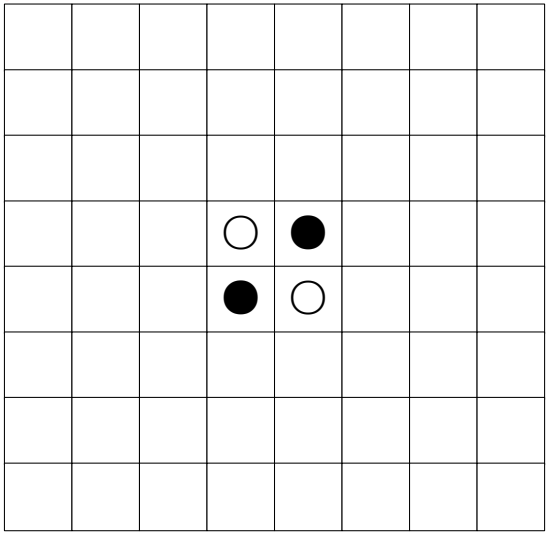
## 8.5.3 洗扑克牌

洗扑克牌的原理其实与随机数排列是相同的，都是将一组数字打乱重新排列。扑克牌共有52张，4种花色。因此，在生成随机数时，除了生成数之外，还需要生成花色代号。

# 8.5 趣味游戏

## 8.5.4 黑白棋

黑白棋，又叫翻转棋（Reversi）、苹果棋或奥赛罗棋（Othello）。棋盘共有8行8列共64格。开局时，棋盘正中央的4格先置放黑白相隔的4枚棋子，如图8-32所示。通常黑子先行。双方轮流落子。只要落子和棋盘上任一枚己方的棋子在一条线上（横、直、斜线皆可）夹着对方棋子，就能将对方的这些棋子转变为我己方（翻面即可）。如果在任一位置落子都不能夹住对手的任一颗棋，就要让对手下子棋。当双方皆不能下子时，或者64个方格全部占满后，游戏就结束，子多的一方胜。



性格决定命运，专注成就人生

# 零基础学算法

## 第9章：信息学奥赛试题精解

# 课程安排

- **9.1 NOIP普及组试题精解**
- **9.2 NOIP提高组试题精解**

# 9.1 NOIP普及组试题精解

## 9.1.1 求级数之和

$$s_n = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$$

# 9.1 NOIP普及组试题精解

## 9.1.2 求素数组合

在输入文件中有n个整数，每个数都在1~500万之间，从这些数中选出k个整数进行相加，可得到不同的组合。

例如：有4个整数8、11、12、13，用其中的3个数进行相加，可得到如下所示的4种组合：

- 8+11+12=31
- 8+11+13=32
- 8+12+13=34
- 11+12+13=36

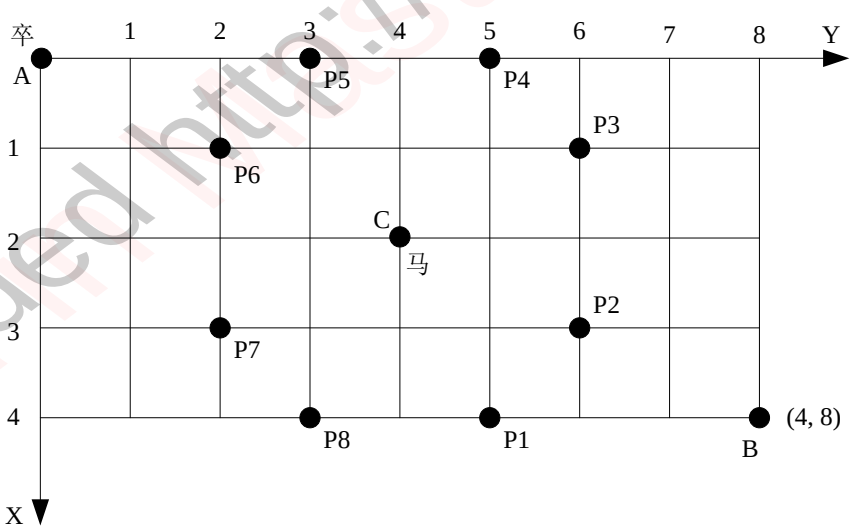
在以上的4个和值中，只有8+11+12=31的结果为素数。

现在要求你编写程序，计算给定n个整数，从中选中k个整数进行组合相加，其和值为素数的情况有多少种？

# 9.1 NOIP普及组试题精解

## 9.1.3 计算卒的路线

在中国象棋中，卒过了界河以后不仅可以向前移动，也可以横向移动。现在假设有一个过河的卒位地图9-3所示的A点，要使该卒从A点移到B点（目标位置）可有多少种不同的路线？



# 9.1 NOIP普及组试题精解

## 9.1.4 检查校验码

对于正式出版的图书，都必须有一个ISBN码。ISBN码一共有10位数字，前9位分别表示国家、出版者、书名号，最后一位是根据前9位数字计算得来的，作为校验码使用。一般ISBN的格式如下：

7-118-01984-4

以上ISBN码中，7表示中国，118表示国防工业出版社，01984代表书名号，最后一位4表示校验码。

校验码是其他9位数字的求余函数。计算方法是：用10～2这九个数分别顺序乘以ISBN的前九位数字，所得乘积之和被模数11除，其余数与11的差，即是校验位的数值。所以，校验位的数值可能是1～11中的任何一个整数，当校验位为10时，用大写字母“X”表示；当校验位为11时，用“0”表示。由此可见校验位只能是0、1、2、3、4、5、6、7、8、9、X，恒为一位数。

# 9.1 NOIP普及组试题精解

## 9.1.5 排座位

六（一）班有几个调皮的同学，当这些同学坐在相邻位置（前后或左右相邻）时，在上课的时候总是会交头接耳，影响其他同学上课。这让班主任李老师非常头疼。

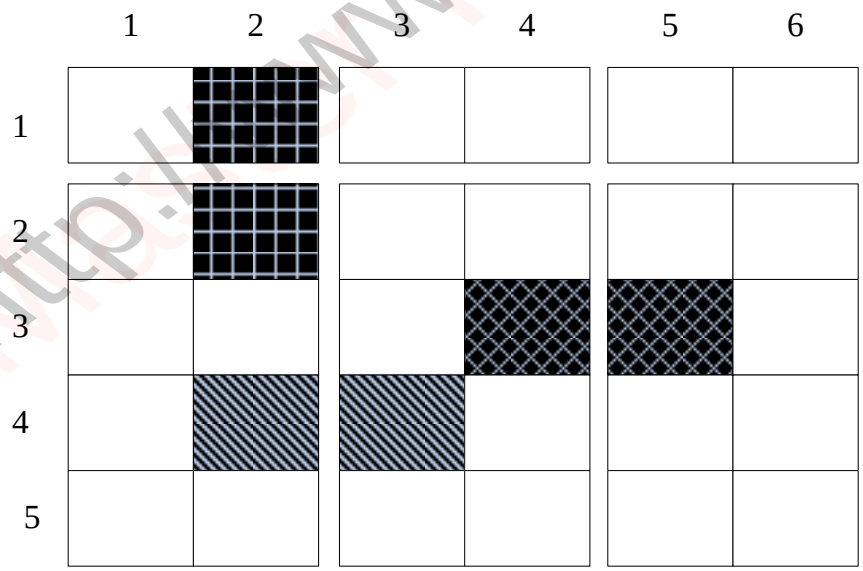
为了尽量减少这种现象，李老师想到一个办法，将这些上课喜欢说话的同学用过道分开。即某两个左右相邻的同学经常上课说话，则从这两个同学之间增加一条纵向过道，让他们的距离拉开，即可杜绝他们两个上课说话了。同样，若前后相邻同学喜欢上课说话，也在他们之间增加一条横向过道，将他们的距离拉开。

李老师决定按这种办法重新安排教室中的过道，以尽量减少上课时私下说话的同学的数量。如果教室中共有M行N列桌椅，需设置K条横向过道和L条纵向地道。请你编写程序，从输入文件中读入上课时喜欢私下说话的D对同学的坐标位置，然后根据这些已知条件，设计出在什么位置设置横向和纵向过道可使上课时私下说话的同学的数量最少。

# 9.1 NOIP普及组试题精解

## 9.1.5 排座位

- 输入数据：
  - 5 6 1 2 3
  - 4 2 4 3
  - 2 2 1 2
  - 3 4 3 5
- 输出数据
  - 1
  - 2 4



# 9.2 NOIP提高组试题精解

## 9.2.1 码码称重

一个天平秤配有若干砝码，这些砝码的重量分别为1g、2g、3g、5g、10g、20g等6种，所有砝码的总重量不超过1000g，各种砝码的数量在输入文件中给出，要求根据输入文件中给出的各种重量的砝码数量，计算出该天平能称出的重量的种类数。

# 9.2 NOIP提高组试题精解

## 9.2.2 阿明的零花钱

爸爸每月给阿明300元零花钱，由阿明自己管理使用。阿明每个月会自己做一个预算，计算本月需要用多少钱，并且总能严格地执行预算，即每月花的钱与预算相同。

为了让阿明从小养成储蓄的习惯，爸爸建议阿明可以将没用完的钱（整百的钱）存在他那里，到年底将存的钱再加上20%的奖励还给阿明。

为此，阿明制定了一个储蓄计划：在每个月的月初，爸爸将零花钱给他后，根据这个月的预算，到月末手中还会有多于100元或恰好100元的钱，就将余下的钱中的整百部分交给爸爸存起来，剩余的钱留在自己手中。

# 9.2 NOIP提高组试题精解

## 9.2.3 购买年货

马上要过春节了，妈妈到超市进行大采购，一共购买n种年货商品。年货采购齐以后，妈妈决定将所有年货商品都合并到一起，好搬回家里。

每一次合并时，可以把两种商品合并到一起，消耗的体力等于两种商品的重量之和。可以看出，所有的商品经过n-1次合并之后，就只剩下一大包了。在合并这些商品时总共消耗的体力等于每次合并时所耗体力之和。

因为还要花大力气把这些年货商品搬回家，所以在合并这些商品时，要尽可能地节省体力。现在要求你给妈妈设计一种方案，让妈妈在合并这些商品时耗费的体力最少，并输出这个最小的体力耗费值。

# 9.2 NOIP提高组试题精解

## 9.2.4 调整队形

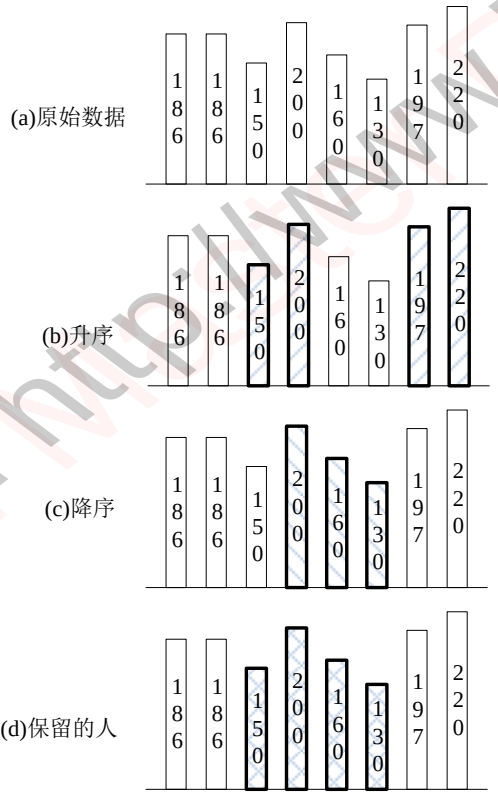
合唱队的队员在演出时一般是按这种形式排列队伍：最高的队员排在中间，然后各队员按身高降序向两侧排列。设有K位合唱队员，从左到右依次编号为1、2、…、K，他们的身高分别为T1、T2、…、TK，则他们的身高满足$T_1<T_2...<T_i>T_{i+1}>...T_{K-1}>T_K$（1<=i<=K）。

现在合唱队有N个队员随机地站成一排（并未按高矮次序排列），要想构成演出时的中间高两边矮的队形，则需要请其中的(N-K)位队员出列，使得剩下的K位队员正好排成合唱队形。

请你根据数据输入文件中给出的各队员的身高数据，计算最少需要几位队员出列，才能使使剩下的队员正好组成合唱队形。

# 9.2 NOIP提高组试题精解

## 9.2.4 调整队形



# 9.2 NOIP提高组试题精解

## 9.2.5 均分纸牌

将一副牌随机分成了N堆，编号分别为1、2、…、N。每堆上有若干张，但纸牌总数一定是牌堆数N的倍数。为了使各堆牌的张数相同，可以在任一堆上取若干张纸牌，然后移动其附近的两堆牌中。具体的移牌规则为：

（1）编号为1的牌堆上的牌，只能移到编号为2的牌堆上（因为编号为1的牌堆左侧没有牌堆）；

（2）编号为N的牌堆上的牌，只能移到编号为N-1的牌堆上（因为其右侧没有牌堆）；

（3）其他牌堆上的牌（例如编号为i的牌堆），可以向左移（编号为i-1的牌堆）或向右移（编号为i+1的牌堆）。

现在要你设计一个程序，用最少的移动次数使每堆上纸牌的数量相同。

性格决定命运，专注成就人生

# 嵌入式资源免费下载

## 总线协议：

1. 基于 PCIe 驱动程序的数据传输卡 DMA 传输
2. 基于 PCIe 总线协议的设备驱动开发
3. CANopen 协议介绍
4. 基于 PXI 总线 RS422 数据通信卡 WDM 驱动程序设计
5. FPGA 实现 PCIe 总线 DMA 设计
6. PCI Express 协议实现与验证
7. VPX 总线技术及其实现
8. 基于 Xilinx FPGA 的 PCIE 接口实现
9. 基于 PCI 总线的 GPS 授时卡设计
10. 基于 CPCI 标准的 6U 信号处理平台的设计
11. USB30 电路保护
12. USB30 协议分析与框架设计
13. USB 30 中的 CRC 校验原理及实现
14. 基于 CPLD 的 UART 设计
15. IPMI 在 VPX 系统中的应用与设计
16. 基于 CPCI 总线的 PMC 载板设计
17. 基于 VPX 总线的工件台运动控制系统研究与开发
18. PCI Express 流控机制的研究与实现
19. UART16C554 的设计
20. 基于 VPX 的高性能计算机设计
21. 基于 CAN 总线技术的嵌入式网关设计
22. Visual C 串行通讯控件使用方法与技巧的研究
23. IEEE1588 精密时钟同步关键技术研究
24. GPS 信号发生器射频模块的一种实现方案
25. 基于 CPCI 接口的视频采集卡的设计
26. 基于 VPX 的 3U 信号处理平台的设计
27. 基于 PCI Express 总线 1394b 网络传输系统 WDM 驱动设计
28. AT89C52 单片机与 ARINC429 航空总线接口设计
29. 基于 CPCI 总线多 DSP 系统的高速主机接口设计
30. 总线协议中的 CRC 及其在 SATA 通信技术中的应用
31. 基于 FPGA 的 SATA 硬盘加解密控制器设计
32. Modbus 协议在串口通讯中的研究及应用
33. 高可用的磁盘阵列 Cache 的设计和实现
34. RAID 阵列中高速 Cache 管理的优化

35. 一种新的基于 RAID 的 CACHE 技术研究与实现
36. 基于 PCIE-104 总线的高速数据接口设计
37. 基于 VPX 标准的 RapidIO 交换和 Flash 存储模块设计
38. 北斗卫星系统在海洋工程中的应用
39. 北斗卫星系统在远洋船舶上应用的研究
40. 基于 CPCI 总线的红外实时信号处理系统
41. 硬件实现 RAID 与软件实现 RAID 的比较
42. 基于 PCI Express 总线系统的热插拔设计
43. 基于 RAID5 的磁盘阵列 Cache 的研究与实现

# VxWorks：

1. 基于 VxWorks 的多任务程序设计
2. 基于 VxWorks 的数据采集存储装置设计
3. Flash 文件系统分析及其在 VxWorks 中的实现
4. VxWorks 多任务编程中的异常研究
5. VxWorks 应用技巧两例
6. 一种基于 VxWorks 的飞行仿真实时管理系统
7. 在 VxWorks 系统中使用 TrueType 字库
8. 基于 FreeType 的 VxWorks 中文显示方案
9. 基于 Tilcon 的 VxWorks 简单动画开发
10. 基于 Tilcon 的某武器显控系统界面设计
11. 基于 Tilcon 的综合导航信息处理装置界面设计
12. VxWorks 的内存配置和管理
13. 基于 VxWorks 系统的 PCI 配置与应用
14. 基于 MPC8270 的 VxWorks BSP 的移植
15. Bootrom 功能改进经验谈
16. 基于 VxWorks 嵌入式系统的中文平台研究与实现
17. VxBus 的 A429 接口驱动
18. 基于 VxBus 和 MPC8569E 千兆网驱动开发和实现
19. 一种基于 vxBus 的 PPC 与 FPGA 高速互联的驱动设计方法
20. 基于 VxBus 的设备驱动开发
21. 基于 VxBus 的驱动程序架构分析
22. 基于 VxBus 的高速数据采集卡驱动程序开发
23. Vxworks 下的冗余 CAN 通讯模块设计
24. WindML 工业平台下开发 S1d13506 驱动及显示功能的实现

# Linux：

1. Linux 程序设计第三版及源代码
2. NAND FLASH 文件系统的设计与实现
3. 多通道串行通信设备的 Linux 驱动程序实现
4. Zsh 开发指南-数组
5. 常用 GDB 命令中文速览
6. 嵌入式 C 进阶之道
7. Linux 串口编程实例
8. 基于 Yocto Project 的嵌入式应用设计
9. Android 应用的反编译
10. 基于 Android 行为的加密应用系统研究
11. 嵌入式 Linux 系统移植步步通
12. 嵌入式 CC++语言精华文章集锦
13. 基于 Linux 的高性能服务器端的设计与研究
14. S3C6410 移植 Android 内核
15. Android 开发指南中文版
16. 图解 Linux 操作系统架构设计与实现原理（第二版）
17. 如何在 Ubuntu 和 Linux Mint 下轻松升级 Linux 内核
18. Android 简单 mp3 播放器源码
19. 嵌入式 Linux 系统实时性的研究
20. Android 嵌入式系统架构及内核浅析
21. 基于嵌入式 Linux 操作系统内核实时性的改进方法研究
22. Linux TCP IP 协议详解
23. Linux 桌面环境下内存去重技术的研究与实现
24. 掌握 Android 7.0 新增特性 Quick Settings
25. Android 应用逆向分析方法研究
26. Android 操作系统的课程教学
27. Android 智能手机操作系统的研究
28. Android 英文朗读功能的实现
29. 基于 Yocto 订制嵌入式 Linux 发行版
30. 基于嵌入式 Linux 的网络设备驱动设计与实现
31. 如何高效学习嵌入式
32. 基于 Android 平台的 GPS 定位系统的设计与实现

## Windows CE:

1. Windows CE.NET 下 YAFFS 文件系统 NAND Flash 驱动程序设计
2. Windows CE 的 CAN 总线驱动程序设计
3. 基于 Windows CE.NET 的 ADC 驱动程序实现与应用的研究

4. 基于 Windows CE.NET 平台的串行通信实现
5. 基于 Windows CE.NET 下的 GPRS 模块的研究与开发
6. win2k 下 NTFS 分区用 ntldr 加载进 dos 源代码
7. Windows 下的 USB 设备驱动程序开发
8. WinCE 的大容量程控数据传输解决方案设计
9. WinCE6.0 安装开发详解
10. DOS 下仿 Windows 的自带计算器程序 C 源码
11. G726 局域网语音通话程序和源代码
12. WinCE 主板加载第三方驱动程序的方法
13. WinCE 下的注册表编辑程序和源代码
14. WinCE 串口通信源代码
15. WINCE 的 SD 卡程序[可实现读写的源码]
16. 基于 WinCE 的 BootLoader 研究
17. Windows CE 环境下无线网卡的自动安装
18. 基于 Windows CE 的可视电话的研究与实现

## PowerPC：

1. Freescale MPC8536 开发板原理图
2. 基于 MPC8548E 的固件设计
3. 基于 MPC8548E 的嵌入式数据处理系统设计
4. 基于 PowerPC 嵌入式网络通信平台的实现
5. PowerPC 在车辆显控系统中的应用
6. 基于 PowerPC 的单板计算机的设计
7. 用 PowerPC860 实现 FPGA 配置
8. 基于 MPC8247 嵌入式电力交换系统的设计与实现
9. 基于设备树的 MPC8247 嵌入式 Linux 系统开发
10. 基于 MPC8313E 嵌入式系统 UBoot 的移植
11. 基于 PowerPC 处理器 SMP 系统的 UBoot 移植
12. 基于 PowerPC 双核处理器嵌入式系统 UBoot 移植
13. 基于 PowerPC 的雷达通用处理机设计
14. PowerPC 平台引导加载程序的移植

## ARM：

1. 基于 DiskOnChip 2000 的驱动程序设计及应用
2. 基于 ARM 体系的 PC-104 总线设计
3. 基于 ARM 的嵌入式系统中断处理机制研究
4. 设计 ARM 的中断处理
5. 基于 ARM 的数据采集系统并行总线的驱动设计
6. S3C2410 下的 TFT LCD 驱动源码
7. STM32 SD 卡移植 FATFS 文件系统源码
8. STM32 ADC 多通道源码
9. ARM Linux 在 EP7312 上的移植
10. ARM 经典 300 问
11. 基于 S5PV210 的频谱监测设备嵌入式系统设计与实现
12. Uboot 中 start.S 源码的指令级的详尽解析
13. 基于 ARM9 的嵌入式 Zigbee 网关设计与实现
14. 基于 S3C6410 处理器的嵌入式 Linux 系统移植
15. CortexA8 平台的μC-OSⅡ及 LwIP 协议栈的移植与实现
16. 基于 ARM 的嵌入式 Linux 无线网卡设备驱动设计
17. ARM S3C2440 Linux ADC 驱动
18. ARM S3C2440 Linux 触摸屏驱动
19. Linux 和 Cortex-A8 的视频处理及数字微波传输系统设计
20. Nand Flash 启动模式下的 Uboot 移植
21. 基于 ARM 处理器的 UART 设计

# Hardware:

1. DSP 电源的典型设计
2. 高频脉冲电源设计
3. 电源的综合保护设计
4. 任意波形电源的设计
5. 高速 PCB 信号完整性分析及应用
6. DM642 高速图像采集系统的电磁干扰设计
7. 使用 COMExpress Nano 工控板实现 IP 调度设备
8. 基于 COM Express 架构的数据记录仪的设计与实现
9. 基于 COM Express 的信号系统逻辑运算单元设计
10. 基于 COM Express 的回波预处理模块设计
11. 基于 X86 平台的简单多任务内核的分析与实现
12. 基于 UEFI Shell 的 PreOS Application 的开发与研究
13. 基于 UEFI 固件的恶意代码防范技术研究
14. MIPS 架构计算机平台的支持固件研究
15. 基于 UEFI 固件的攻击验证技术研究

16. 基于 UEFI 的 Application 和 Driver 的分析与开发

# Programming:

1. 计算机软件基础数据结构 - 算法