数据结构

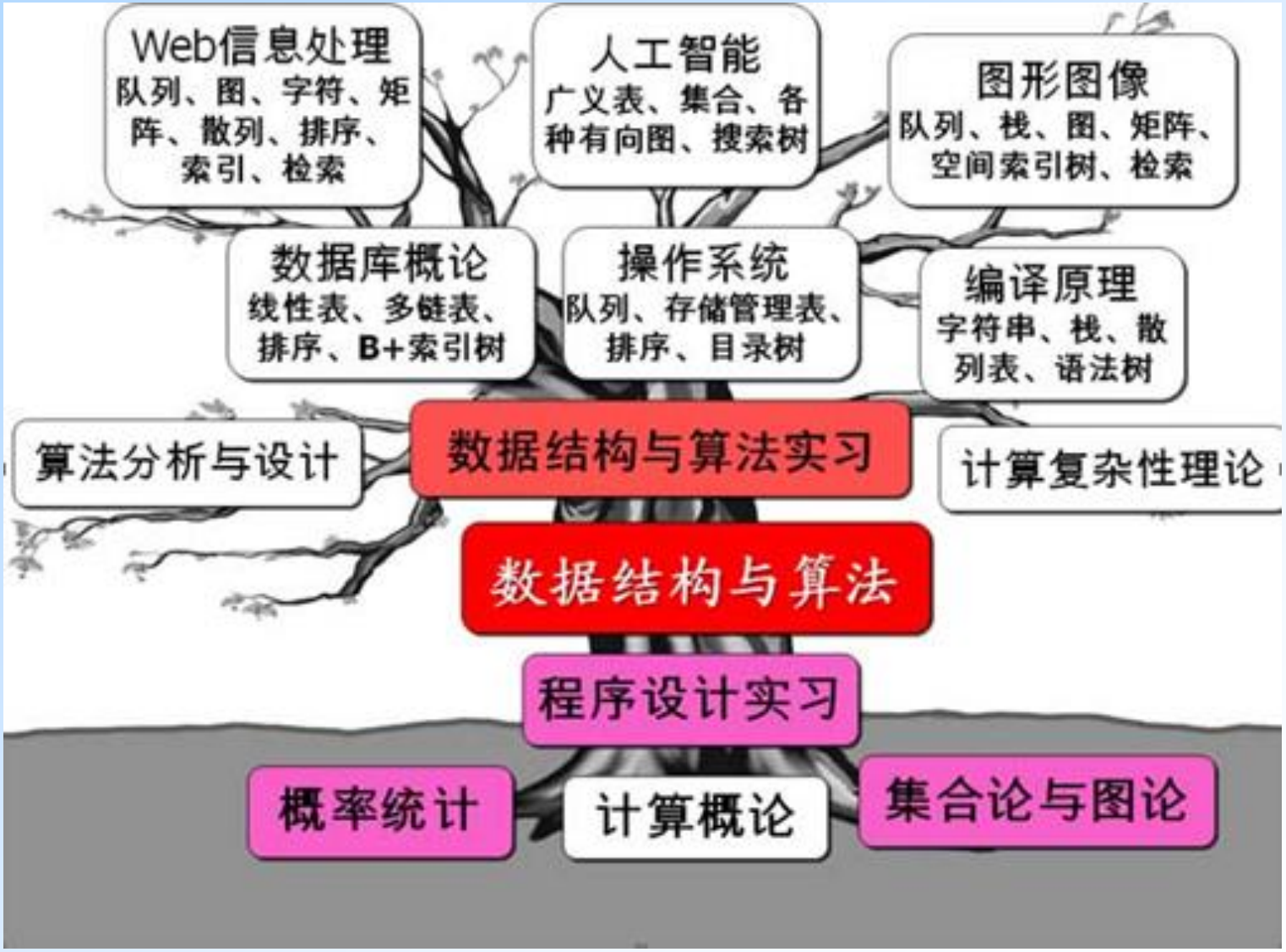
Data Structure

课程视频资源

- 中国大学MOOC(慕课)
- 智慧树
- 学堂在线

参考书目

(1) 教材《数据结构》c语言版
严蔚敏，清华大学出版社
(2) 《数据结构联考辅导教程》
李春葆等，清华大学出版社
(3) 《数据结构与算法》
张铭等，高等教育出版社
(4) 《Data Structures & Algorithm Analysis in C++》，MARK ALLEN WEISS，
优秀原版教材，清华大学出版社


Web信息处理
队列、图、字符、矩阵、散列、排序、索引、检索
人工智能
广义表、集合、各种有向图、搜索树
图形图像
队列、栈、图、矩阵、空间索引树、检索
数据库概论
线性表、多链表、排序、B+索引树
操作系统
队列、存储管理表、排序、目录树
编译原理
字符串、栈、散列表、语法树
算法分析与设计
数据结构与算法实习
计算复杂性理论
数据结构与算法
程序设计实习
概率统计
计算概论
集合论与图论

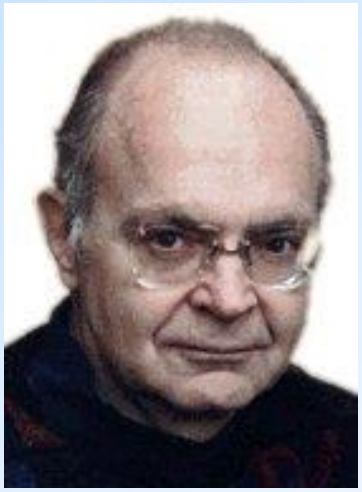
开设本课程的背景

“数据结构”在计算机科学中是一门综合性的**专业基础课**，数据结构是介于数学、计算机硬件和软件三者之间的一门核心课程。数据结构的内容不仅是一般程序设计（特别是非数值性程序设计）的基础，它是学习操作系统、编译原理、数据库原理、算法分析与设计、计算机图形学等计算机专业核心课程的基础。掌握好这门课程的内容，是学习计算机其它相关课程的必备条件。

它主要研究计算机加工对象的逻辑结构、在计算机中的存储表示形式以及实现各种基本操作的算法。

数据结构的创始人

《The art of computer programming》



- “数据结构”作为一门独立的课程在国外是从1968年才开始设立的。1968年美国唐·欧·克努特(Donald Ervin Knuth)教授开创了数据结构的最初体系，他所著的描述基本算法（Algorithm）与数据结构（Data Structures）的巨作《计算机程序设计的艺术》第一卷《基本算法》是第一本较系统地阐述数据的逻辑结构和存储结构及其操作的著作。

- 多卷本《计算机程序设计的艺术》堪称计算机科学理论与技术的经典巨著，被《美国科学家》杂志列为20世纪最重要的12本物理科学类专著之一，与爱因斯坦《相对论》、狄拉克《量子力学》、理查·费曼《量子电动力学》等经典比肩而立。本书作者高德纳因而荣获1974年度的图灵奖。

- 完成了对整个西文印刷行业带来了革命性变革的TEX排版软件和METAFONT字型设计软件。

Algorithm +Data Structures = Programs

1976 年，瑞士计算机科学家尼克劳斯·威茨（Niklaus Wirth）提出了程序设计的两大要素。因提出这一公式并以此作为其一本专著书名，于1984 年获得了图灵奖，是瑞士学者中唯一获此殊荣的人。

Pascal之父：在C语言问世以前， Pascal是风靡全球、最受欢迎的语言之一。首次提出了“结构化程序设计” 的概念，又称为“自顶向下”或“逐步求精”法，在程序设计领域引发了一场革命，成为程序开发的一个标准方法。

第一章 绪论

- 数据结构研究的主要内容
- 基本概念和术语
- 抽象数据类型
- 算法和算法分析

❖ 数据结构研究的主要内容

当今计算机应用的特点：

●所处理的数据量大且具有一定的关系；

●对其操作不再是单纯的数值计算，而更多地是需要对其进行组织、管理和检索。

- 一般来讲，用计算机解决一个具体问题时，大致需要下列几个步骤：
- 首先从具体问题抽象出一个数学模型
- 设计模拟数学模型的算法
- 编写程序、进行测试得到最终答案。

例如: 数值计算的程序设计问题

结构静力分析计算

– 线性代数方程组

全球天气预报

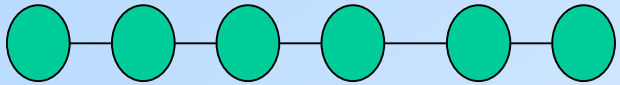
– 环流模式方程
(球面坐标系)

应用举例—— 学籍档案管理

假设一个学籍档案管理系统应包含如下表所示的学生信息。

学生基本情况

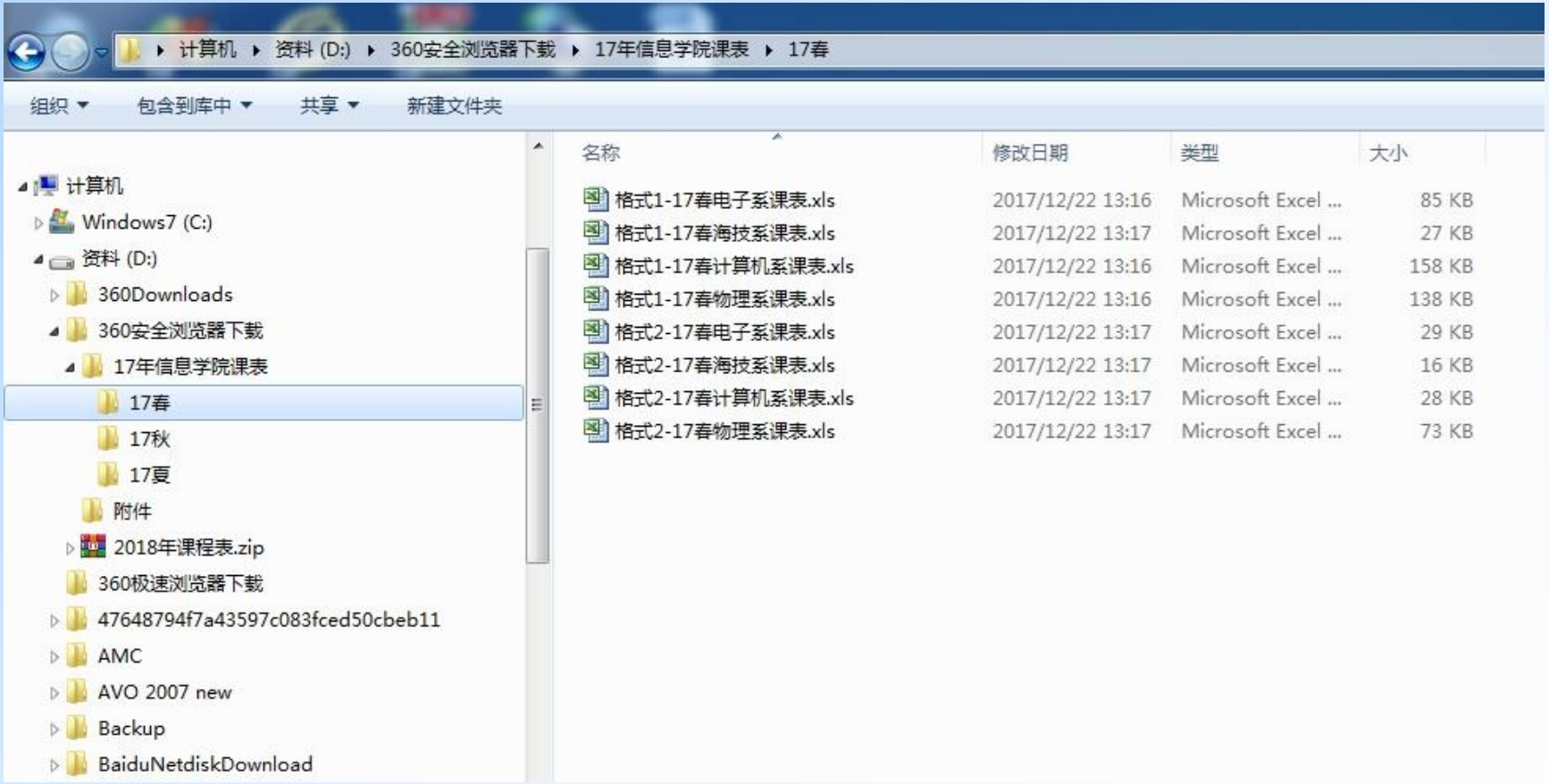
学　号	姓　名	性　别	出生年月	……
99070101	李　军	男	80．12	……
99070102	王颜霞	女	81．2	……
99070103	孙　涛	男	80．9	……
99070104	单晓宏	男	81．3	……
……	……	……	……	……



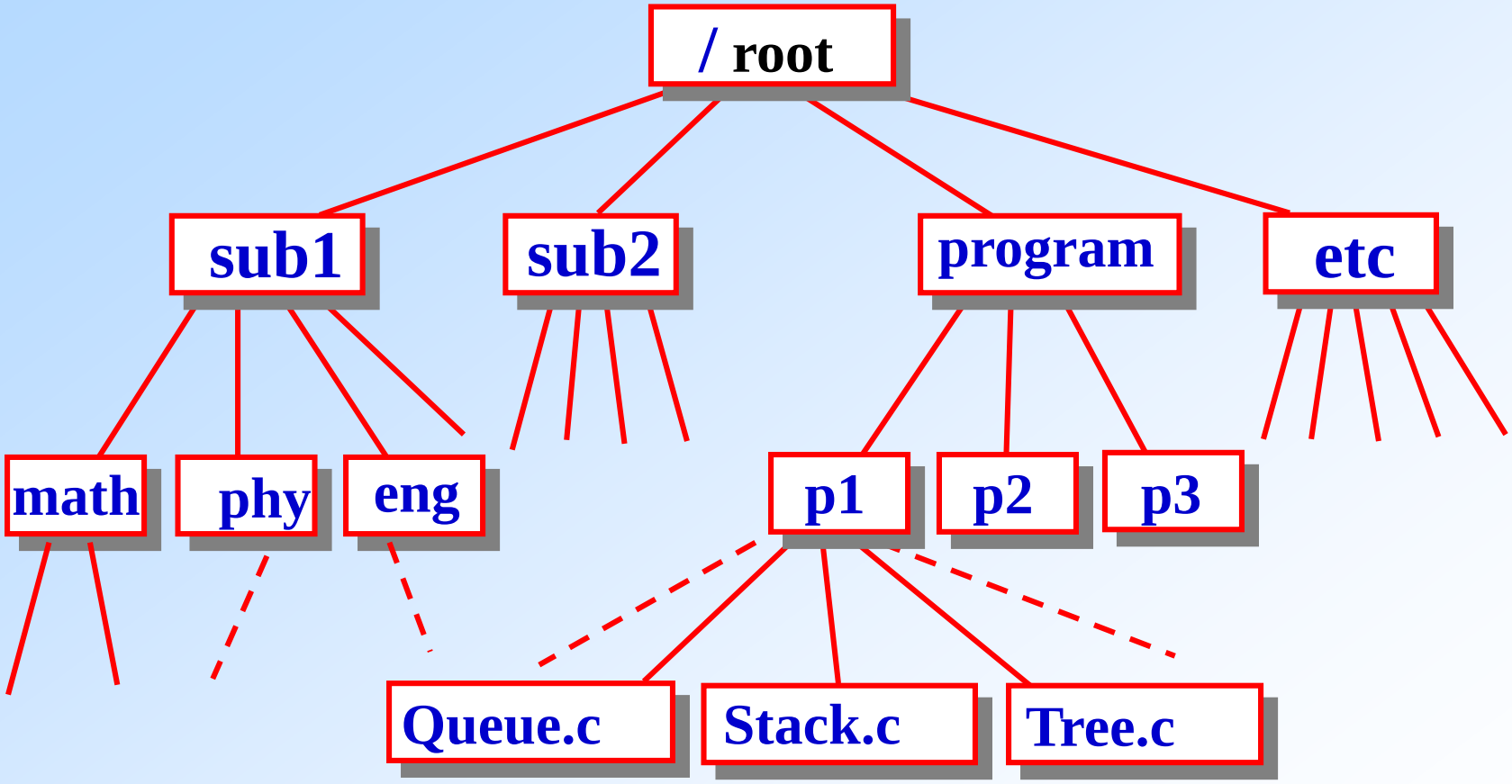
特点：

- 每个学生的信息占据一行，所有学生的信息按学号顺序依次排列构成一张表格；
- 表中每个学生的信息依据学号的大小存在着一种前后关系，这就是我们所说的线性结构；
- 对它的操作通常是插入某个学生的信息，删除某个学生的信息，更新某个学生的信息，按条件检索某个学生的信息等等。

应用举例—— 文件目录系统结构图



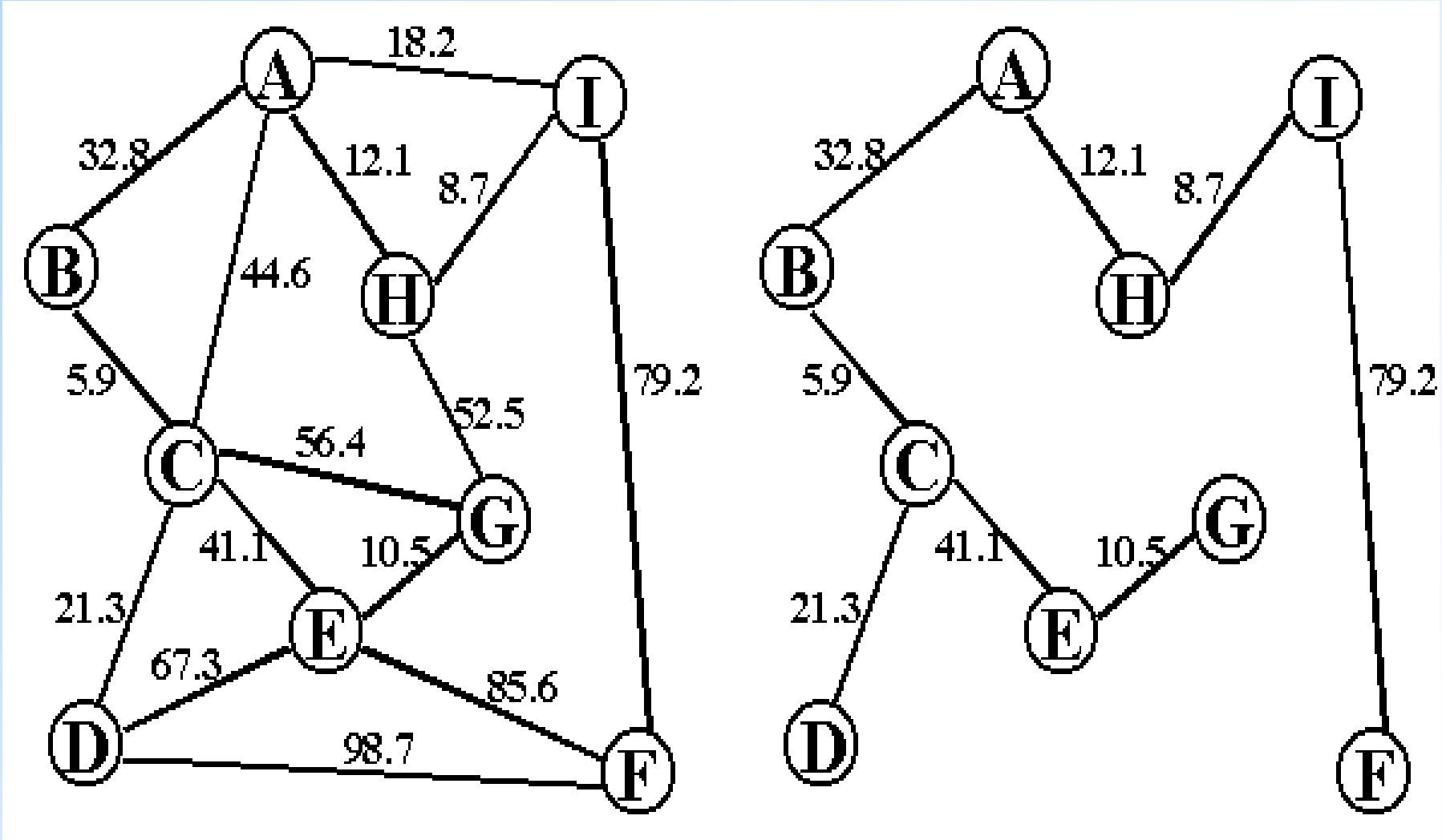
应用举例—— 文件系统结构图



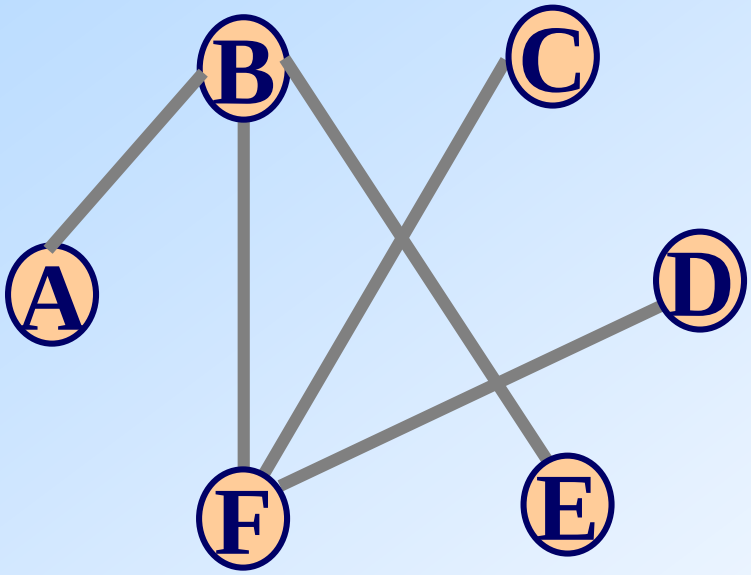
应用举例——铺设城市的煤气管道

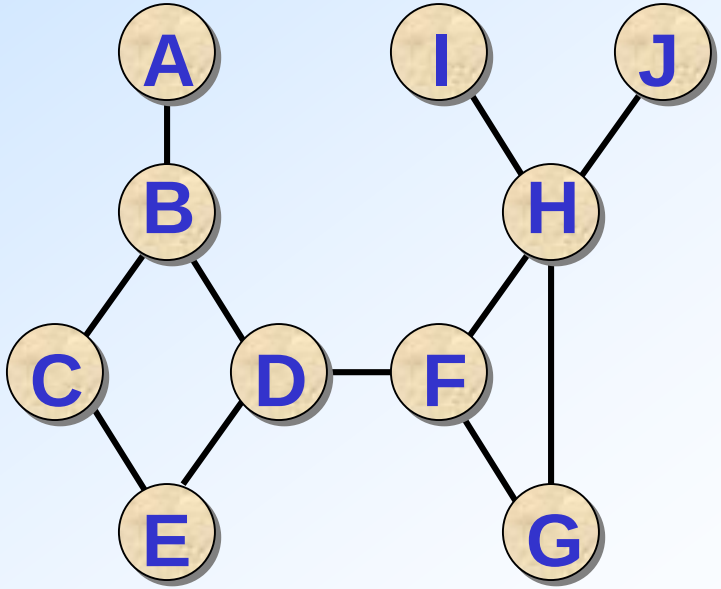
算法：？　如何规划使得总投资花费最少?

模型：？　图

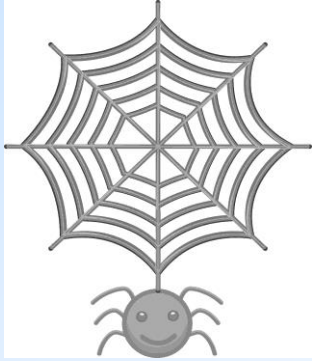

A
I
B
H
C
G
E
D
F
18.2
32.8
12.1
8.7
44.6
5.9
79.2
52.5
56.4
41.1
10.5
21.3
67.3
85.6
98.7
A
I
B
H
C
G
E
D
F
32.8
12.1
8.7
5.9
79.2
41.1
10.5
21.3

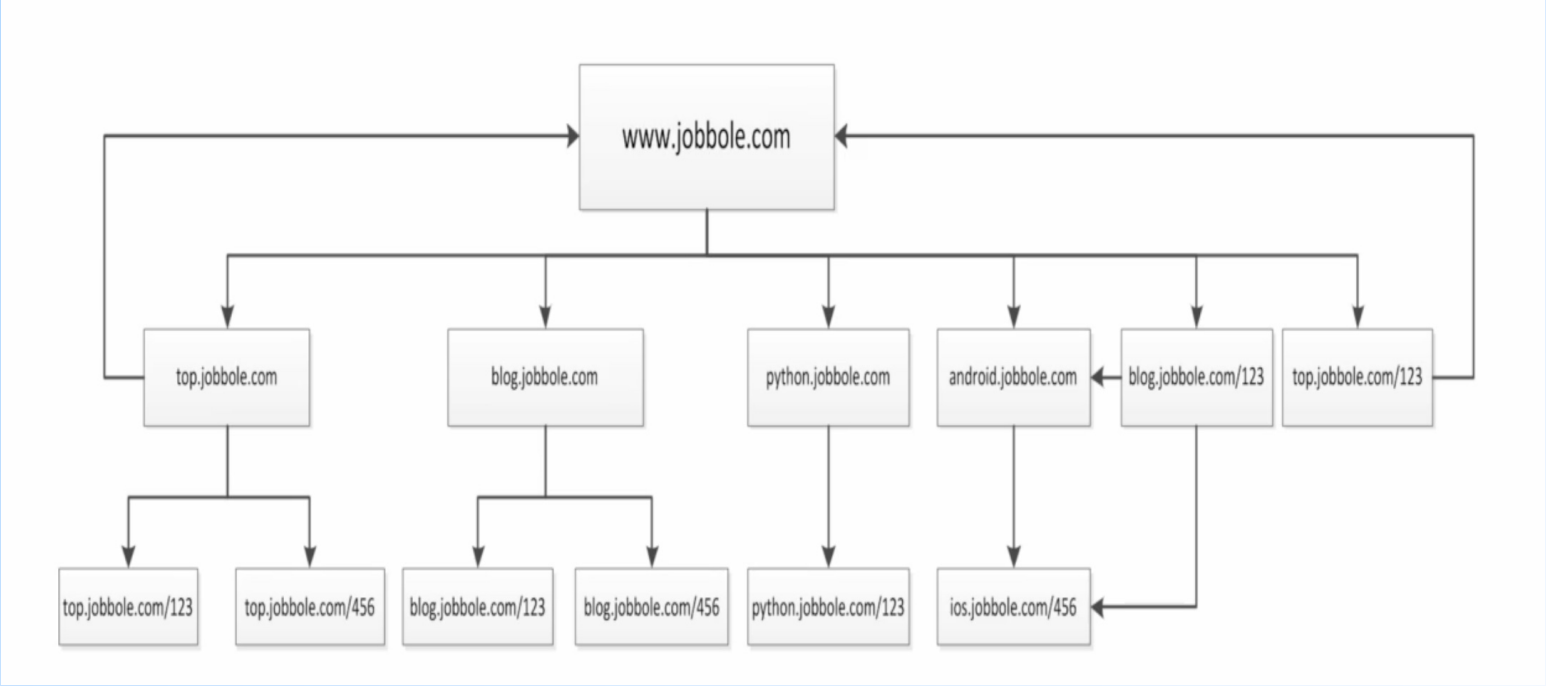
应用举例——通信网络中的应用





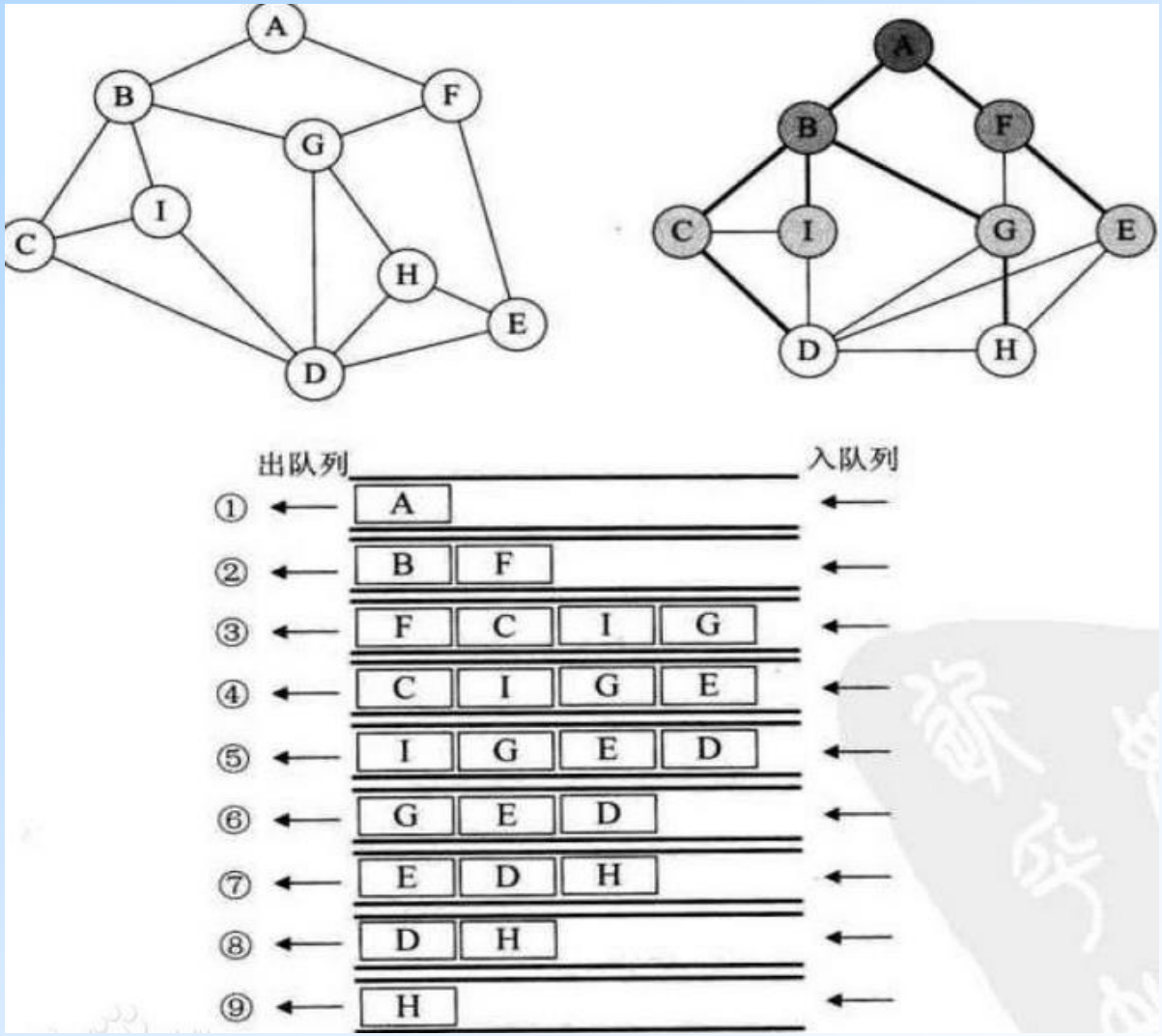
应用举例——搜索引擎技术之网络爬虫





从任何一个子页面其实都是可以返回到首页(伯乐在线)，所以当爬取页面数据的时候就会涉及到去重的问题。

网页的抓取策略可以分为深度优先、广度优先和最佳优先三种。深度优先在很多情况下会导致爬虫的陷入(trapped)问题，目前常见的是广度优先和最佳优先方法。



需要将爬过的url记录下来广度优先(宽度优先)，是指将新下载网页发现的链接直接插入到待抓取URL队列的末尾，也就是指网络爬虫会先抓取起始页中的所有网页，然后在选择其中的一个连接网页，继续抓取在此网页中链接的所有网页 。

- 计算机的操作对象的关系更加复杂，操作形式不再是单纯的数值计算，而更多地是对这些具有一定关系的数据进行组织管理，我们将此称为非数值性处理。要使计算机能够更有效地进行这些非数值性处理，就必须弄清楚这些操作对象的特点，在计算机中的表示方式以及各个操作的具体实现手段。
- **因此从广义上讲，数据结构描述现实世界实体的数学模型及其上的操作在计算机中的表示和实现.**

机器学习之深度学习

- “西方国家点燃了深度学习的火炬，但最大的受益者将会是中国。这种全球性的变化是由两方面转变引起的：从发明的年代转变为实干的年代；从专家的年代转变为数据的年代。”
- 规则式方法（专家系统）用一系列写好的逻辑规则教导计算机如何思考，例如“若X，则Y”。
- 神经网络方法只是把某一现象（图片、国际象棋、声音）等大量例子输入人工神经网络，让网络从数据中学习、识别规律。人的干预越少越好。训练成功的深度学习算法需要运算力、工程能力及大量的数据。

❖基本概念和术语

数据(Data)

- 是信息的载体，是描述客观事物的数、字符、以及所有能输入到计算机中，被计算机程序识别和处理的符号的集合。
  - 数值性数据
  - 非数值性数据

数据元素（Data Element）

- 数据的基本单位。在计算机程序中常作为一个整体进行考虑和处理。
- 有时一个数据元素可以由若干数据项(Data Item)组成。数据项是具有独立含义的最小标识单位。
- 数据元素又称为元素、结点、记录

数据项(Data Item)

姓名	俱乐部名称	出生日期			入队日期	职位	业绩
		年	月	日			

数据对象 (data object)

- 具有相同性质的数据元素的集合。
  - 整数数据对象

    $N = \{ 0, \pm1, \pm2, \ldots \}$
  - 字母字符数据对象

    $C = \{ \text{‘A’}, \text{‘B’}, \text{‘C’}, \ldots \text{‘F’} \}$

数据结构（Data Structure）

- 形式定义：

  某一数据对象的所有数据成员之间的关系。

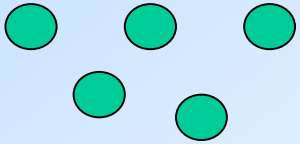
  记为：

  Data_Structure = {D, S}

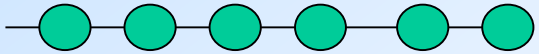
  其中，D 是某一数据对象， S 是该对象中所有数据成员之间关系的有限集合。

四个基本结构

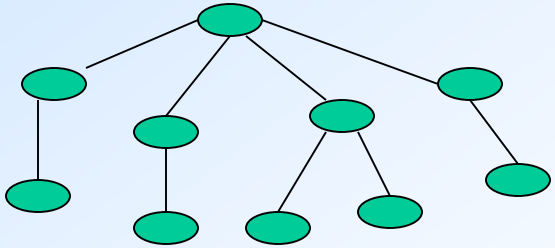
- 集合



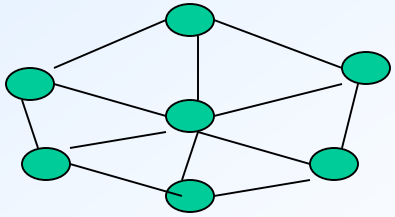
- 线性结构



- 树形结构



- 网状结构



数据的逻辑结构

- 从逻辑关系上描述数据，与数据的存储无关；
- 从具体问题抽象出来的数据模型；
- 与数据元素本身的形式、内容无关；
- 与数据元素的相对位置无关。

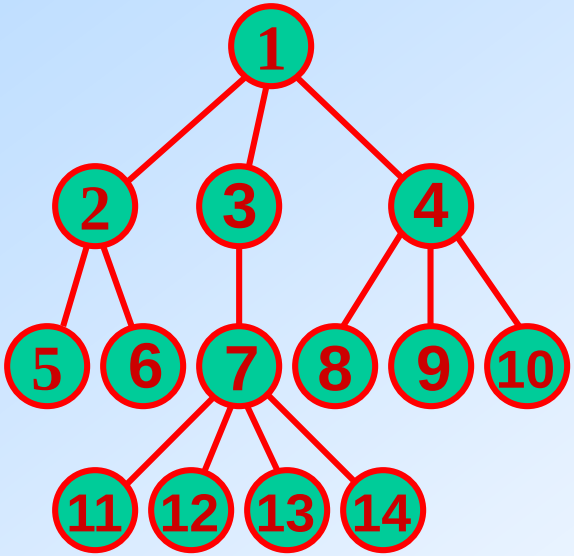
数据的逻辑结构分类

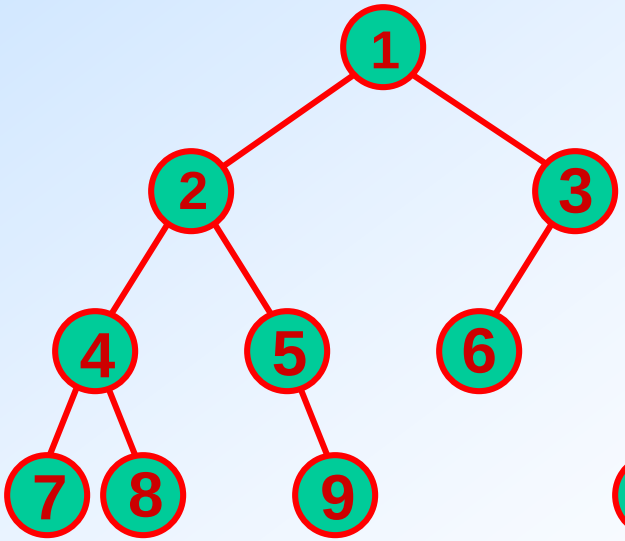
- 线性结构
  - 线性表
- 非线性结构
  - 树
  - 图（或网络）

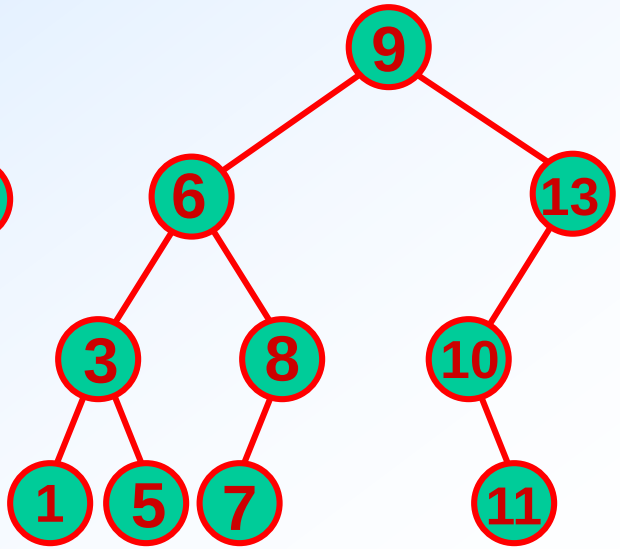
线性结构

树形结构

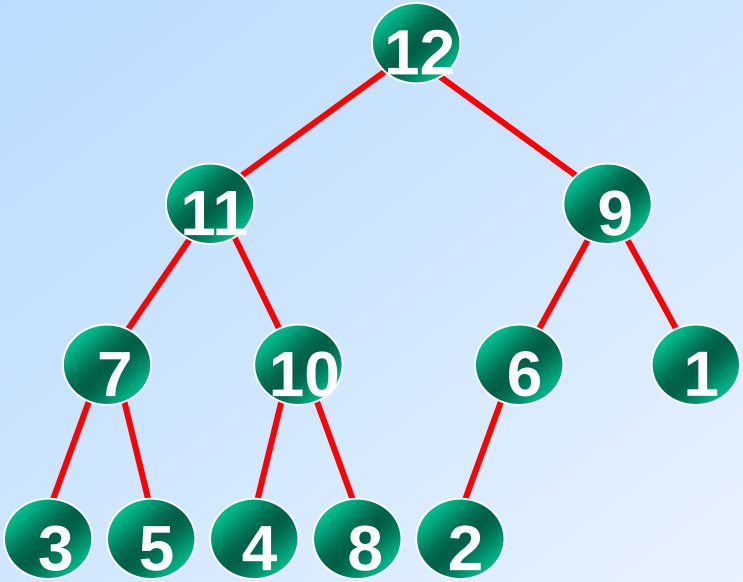
树　　二叉树　　二叉排序树



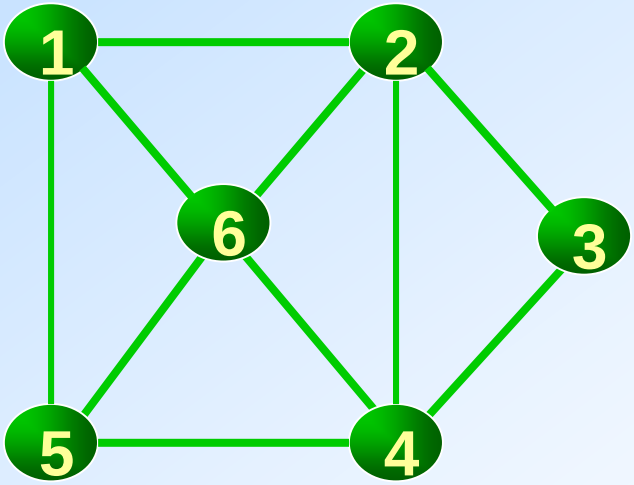




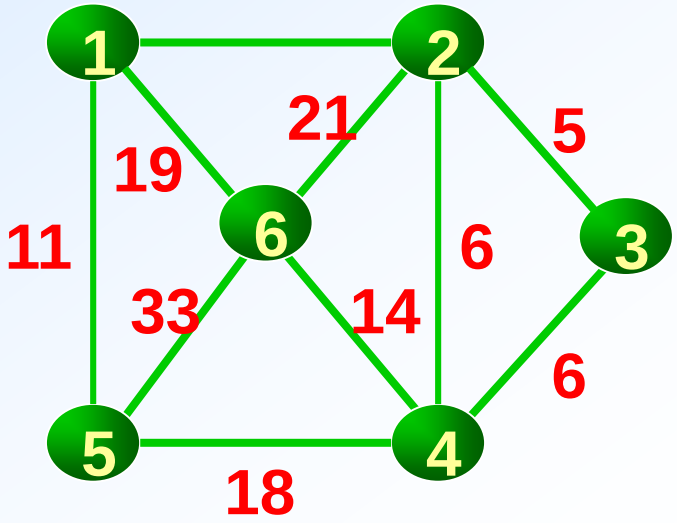
堆结构



图结构

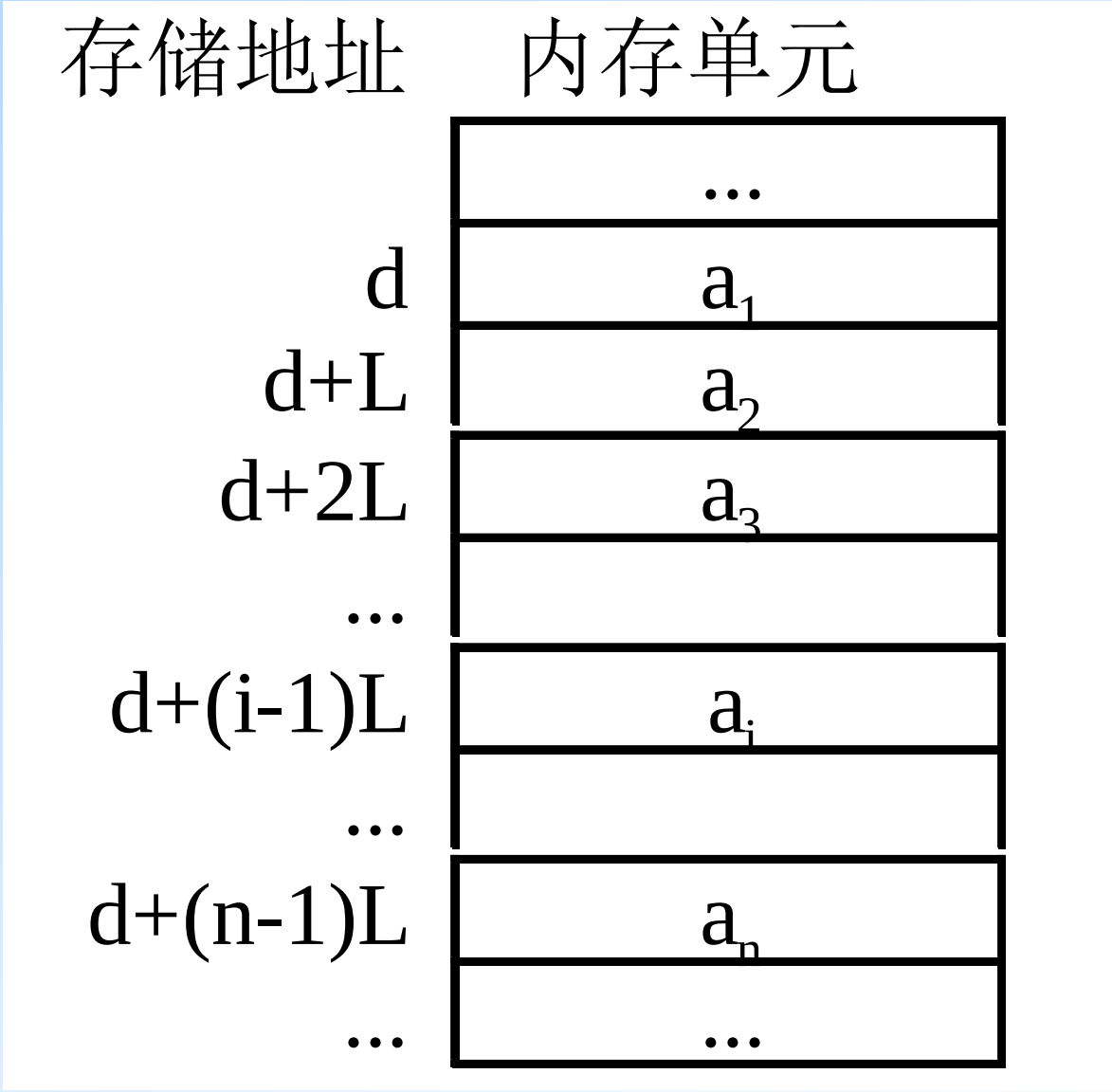


网络结构



数据的存储结构（物理结构）

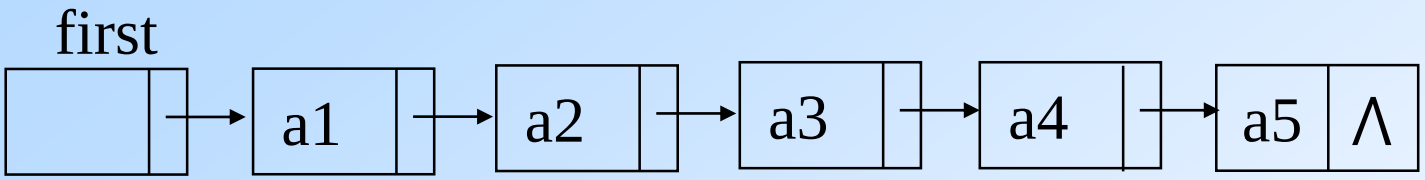
- 数据结构在计算机中的表示。
- 数据的存储结构依赖于计算机语言。
  - 顺序存储表示
  - 链式存储表示
  - 索引存储表示
  - 散列存储表示

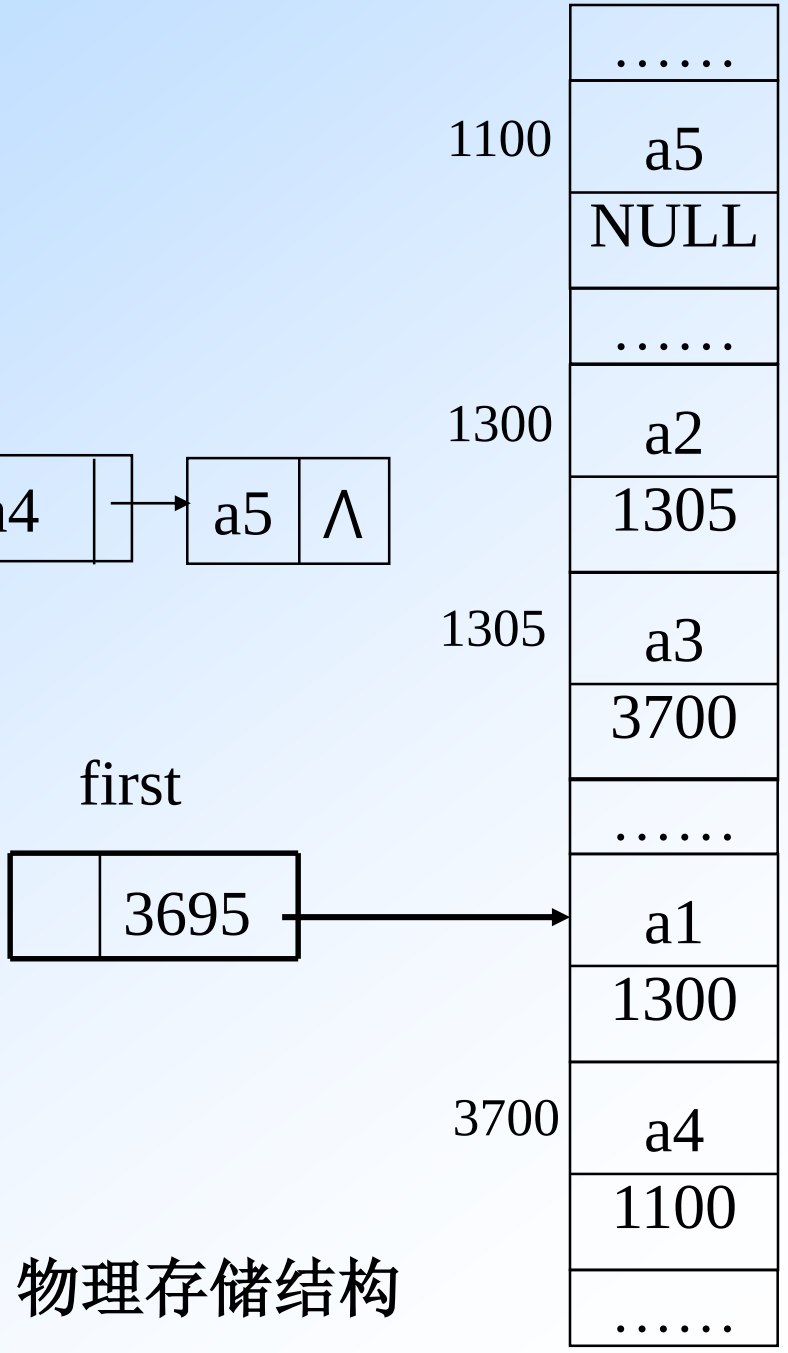


线性表顺序存储结构示意图

线性表L=(a1,a2,a3, a4,a5)

例： 逻辑结构





物理存储结构

❖抽象数据类型（Abstract Data Type）

- 数据类型
  定义：一个值的集合和定义在这个值集上的一组操作的总称。
- C语言中分为原子类型和结构类型

例1、 在FORTRAN语言中，变量的数据类型有整型、实型、和复数型

例2、在C语言中数据类型：基本类型和构造类型

基本类型：整型、浮点型、字符型

构造类型：数组、结构、联合、指针、枚举型、自定义

- 整数类型：2~4个字节
- 实数类型： 4~8个字节
- 指针类型：4~8个字节（32bits或64bits地址长度）
- 字符类型：单个字节表示ASCII，汉字需要2个字节
- 布尔类型：整数0表示false，非0表示true

抽象数据类型

- 是指一个数学模型以及定义在此数学模型上的一组操作，人们将用户自定义的类型称作抽象数据类型。
- 数据结构+定义在此数据结构上的一组操作 = 抽象数据类型
- 例如：矩阵 + （求转置、加、乘、
  求逆、求特征值）
  构成一个矩阵的抽象数据类型

抽象数据类型的描述

- 抽象数据类型可用（D，S，P）三元组表示

  其中，D是数据对象，S是D上的关系集，P是对D的基本操作集。

  ADT 抽象数据类型名 {

  数据对象：〈数据对象的定义〉

  数据关系：〈数据关系的定义〉

  基本操作：〈基本操作的定义〉

  } ADT 抽象数据类型名

其中,数据对象、数据关系用伪码描述；基本操作定义格式为

基本操作名（参数表）

初始条件：〈初始条件描述〉

操作结果：〈操作结果描述〉

- 基本操作有两种参数：赋值参数只为操作提供输入值；引用参数以&打头， 除可提供输入值外，还将返回操作结果。
- “初始条件”描述了操作执行之前数据结构和参数应满足的条件，若不满足，则操作失败，并返回相应出错信息。
- “操作结果”说明了操作正常完成之后，数据结构的变化状况和应返回的结果。若初始条件为空，则省略之。

20世纪60年代后期，为了实现算法细节和数据内部结构的隐蔽，在simula67语言中引入了类，随后出现了模块的概念。

模块分成模块式和模块体。模块式定义外部可见的运算接口，模块体定义对外不可见的私有数据和运算。通过接口和实现的分离，模块提供了用户自定义类型的手段，达到数据抽象、信息隐蔽的目的。

抽象数据类型的表示和实现

- 抽象数据类型可以通过固有数据类型(高级编程语言中已实现的数据类型)来实现

- C++用类的说明来表示ADT，用类的实现来实现ADT。C++中实现的类相当于数据的存储结构及其在存储结构上实现的对数据的操作。

“能行可计算”的概念

20世纪20年代，为了解决数学本身的可检验性问题，大数学家希尔伯特提出抽象的“计算”概念——“能否找到一种 基于有穷观点的能行方法，来判定任何一个数学命题的真假”。

- 由有限数量的明确有限指令构成；
- 指令执行在有限步骤后终止；
- 指令每次执行都总能得到正确解；
- 原则上可以由人单独采用纸笔完成，而不依靠其它辅助；
- 每条指令可以机械地被精确执行，而不需要智慧和灵感。

❖算法和算法分析

- 算法定义：为了解决某类问题而规定的一个有限长的操作序列 。
- 特性：
  - 有穷性 算法在执行有穷步后能结束
  - 确定性 每步定义都是确切、无歧义
  - 可行性 每一条运算应足够基本
  - 输入 有0个或多个输入
  - 输出 有一个或多个输出

算法的描述

- 本书将采用类C语言描述算法
- 类C语言是标准C语言的简化，与标准C语言的主要区别：

  1. 所有算法都以如下所示的函数形式表示：

     函数类型　函数名（参数表）

     {

     　　语句序列

     }

     类C语言的形参书写比标准C语言简单，如，int xyz(int a,int b,int c)可以简单写成int xyz (int a,b,c)

类C与标准C的主要区别(续)

2. 局部量的说明可以省略，必要时对其作用给予注释 。

3. 不含go to语句，增加一个出错处理语句error(字符串)，其功能是终止算法的执行并给出表示出错信息的字符串。

4. 输入/输出语句有:

输入语句 scanf([格式串]),变量1，…，变量N）；

输出语句 printf([格式串]),变量1，…，变量N）；

通常省略格式串 。

返回

算法设计

- 例子： 选择排序
- 问题：递增排序
- 解决方案：逐个选择最小数据
- 算法框架：

```
for ( int i = 0; i < n-1; i++ ) {
//n-1趟， 从a[i]检查到a[n-1];
    若最小整数在a[k], 交换a[i]与a[k];
}
```

- 细化：Select Sort

```
void selectSort ( int a[ ], int n ) {
    //对n个整数a[0],a[1],…,a[n-1]按递增顺序排序
    for ( int i = 0; i < n−1; i++ ) {
        int k = i;

        //从a[i]查到a[n-1], 找最小整数,保存在a[k]
        for ( int j = i+1; j < n; j++ )
            if ( a[j] < a[k] )    k = j ;

        if ( k != i ) {int temp = a[i];  a[i] = a[k];   a[k] = temp;}
    }
}
```

性能分析与度量

算法的性能标准

- **正确性**

算法应满足具体问题的需求,对算法是否“正确”的理解可以有四个层次:

1)程序中不含语法错误;

2)程序对于几组输入数据能够得出满足要求的结果;

3)程序对于精心选择的、典型、苛刻且带有刁难性的几组输入数据 能够得出满足要求的结果;

4)程序对于一切合法的输入数据都能得出满足要求的结果;

通常以第 3层意义的正确性作为衡量一个算法是否合格的标准。

性能分析与度量

算法的性能标准

- **可读性**算法应该好读,有利于阅读者对程序的理解。

  算法主要是为了人的阅读与交流，其次才是为计算机执行，因此算法应该易于人的理解；另一方面，晦涩难读的程序易于隐藏较多错误而难以调试。

- **健壮性**算法应具有容错处理。当输入非法数据时，算法应其作出反应，而不是产生莫名其妙的输出结果.并且处理出错的方法不应是中断程序的执行，而应是返回一个表示错误或错误性质的值，以便在更高的抽象层次上进行处理。

性能分析与度量

- **效率**效率指的是算法执行的时间；存储量需求指算法执行过程中所需要的最大存储空间。一般，这两者与问题的规模有关。

Kent Beck： Make It Work
Make It Right
Make It Fast

算法设计

1、**穷举法**

2、**回溯法** （深度优先搜索）

3、**分治法和递归法**

（快速排序、归并排序、二分检索）

4、**贪心法和动态规划法**

（prim算法和Kruskal算法、Dijkstra算法、Floyd算法）

算法复杂性的分析

- 算法的复杂性包括时间复杂性（所需运算时间）和空间复杂性（所占存储空间），重点是时间复杂性 。

- 1965年计算机科学家Juris Hartmains 和Richard Stearns在《论算法的计算复杂度》一文提出了这个概念，并因此获得图灵奖。

- 最早将计算复杂度量化衡量的是算法分析之父高德纳(Donald Ervin Knuth)，使得一个算法好坏的度量和问题的大小不再有关，这是一个了不起的贡献。

算法的后期测试

在算法中的某些部位插装时间函数time ( ),

测定算法完成某一功能所花费时间

```
double start, stop;
time (&start);
int k = seqsearch (a, n, x);
time (&stop);
double runTime = stop - start;
printf ( ”%d%d\n " , n, runTime );
```

```
int seqsearch ( int a[ ], int n, int x ) {
//在a[0],…,a[n-1]中搜索x
    int i = 0;
    while ( i < n && a[i] != x )
          i++;
    if ( i == n ) return −1;
    return i;
}
```

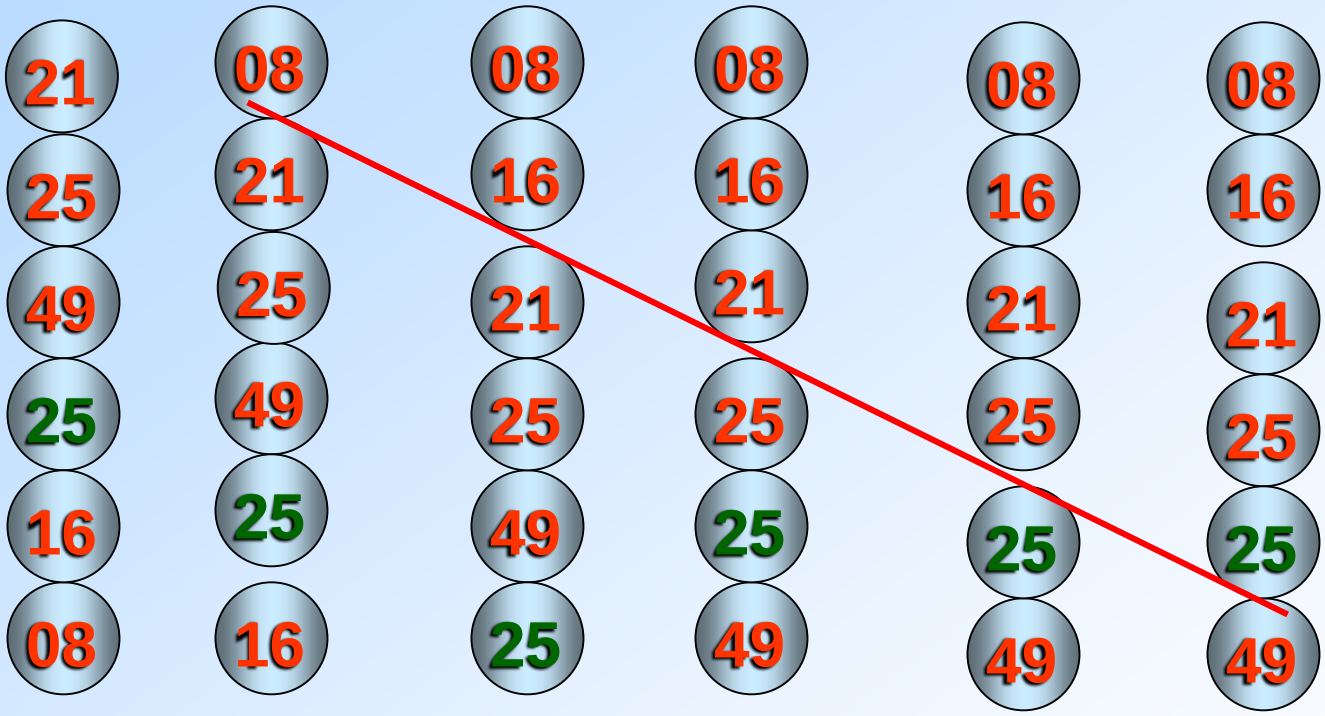
算法的事前估计

- **时间复杂度度量**
  - 运行时间 = 算法中每条语句执行时间之和。
  - 每条语句执行时间 = 该语句的执行次数（频度）* 语句执行一次所需时间。
  - 语句执行一次所需时间取决于机器的指令性能和速度和编译所产生的代码质量，很难确定。
  - 设每条语句执行一次所需时间为单位时间，则一个算法的运行时间就是该算法中所有语句的频度之和。

- 算法执行时间与原操作执行次数之和成正比（原操作执行时间固定）

- 从算法中选取一种对于所研究问题为基本操作的原操作，以该操作在算法中重复执行的次数作为算法运行时间的衡量准则。

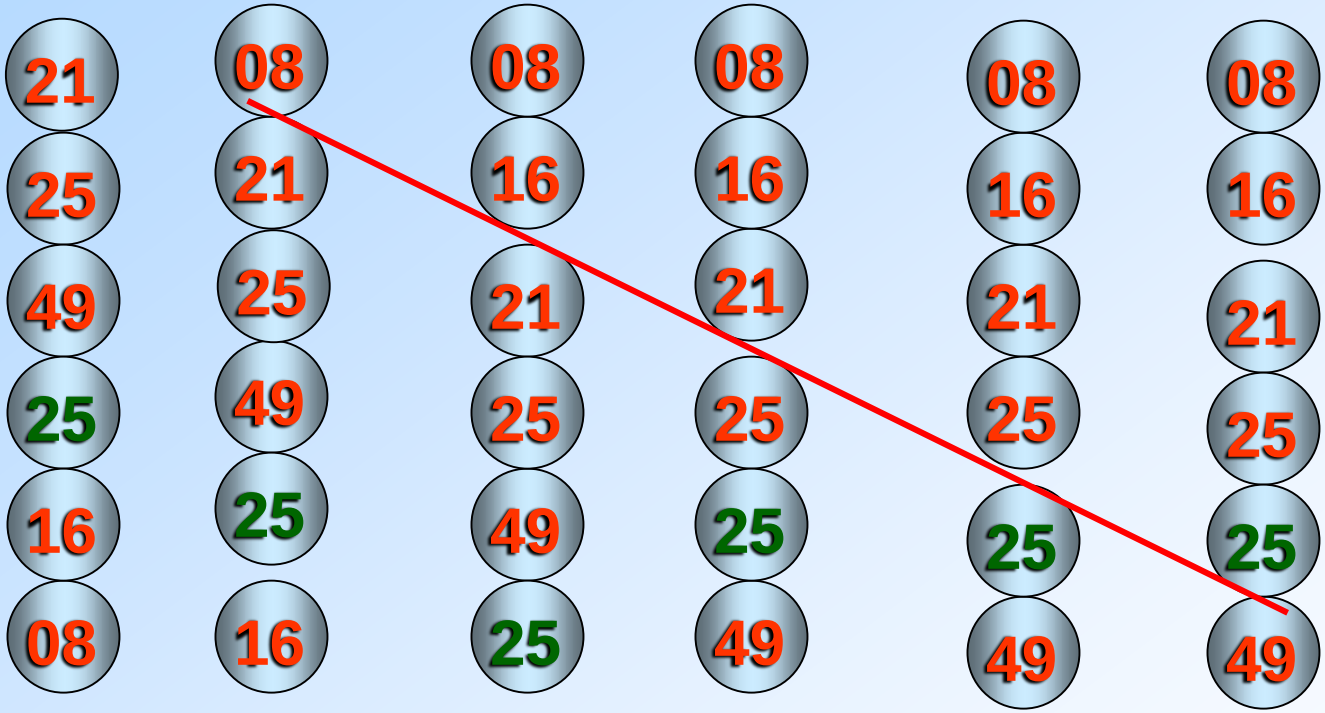
上起泡排序的过程


21
25
49
25
16
08
08
21
25
49
25
16
08
16
21
25
49
25
08
16
21
25
25
49
08
16
21
25
25
49
08
16
21
25
25
49

有的情况下，算法中基本操作重复执行的次数还随问题的输入数据集不同而不同。例如：

```
Void bubblesort(int a[]，int n)
  for(i=n-1;change=TURE;i>1 && change;--i)
    {
        change=false;
        for(j=0;j<i;++j)
            if (a[j]>a[j+1]) {
                a[j] ←→a[j+1];
                change=TURE}
      }
```

上起泡排序的过程



最好情况：交换0次

最坏情况：1+2+3+…+n-1=n(n-1)/2

n!种输入情况的平均时间复杂度为:$O(n^2)$

- 算法的运行时间往往还与具体输入的数据有关，通常用以下两种方法来确定一个算法的运算时间：
- 1. 平均时间复杂性：研究同样的n值时各种可能的输入，取它们运算时间的平均值。
- 2. 最坏时间复杂性：研究各种输入中运算最慢的一种情况下的运算时间。

返回

- 从保守估计的角度出发，在规模为n的所有输入中选择执行时间最长者作为T(n)，并以T(n)度量该算法的时间复杂度。

数量级函数（Order of Magnitude function）

- 基本操作数量函数T(n)的精确值并不是特别重要，重要的是T(n)中起决定性因素的主导部分。用动态的眼光看，就是当问题规模增大的时候，T(n)中的一些部分会盖过其它部分的贡献. 数量级函数描述了T(n)中随着n增加而增加速度最快的部分 。

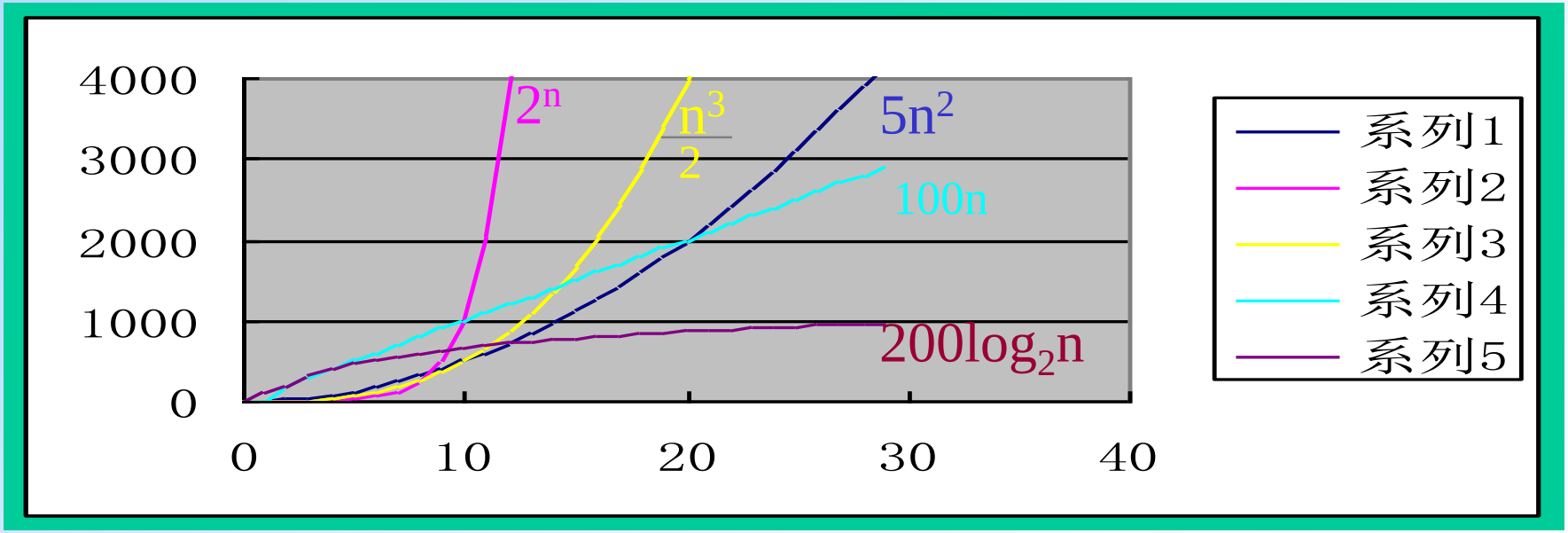
- 大O表示法：时间复杂度常用数量级的形式来，记作T(n)=O(f(n))。其中f(n)表示T(n)中的主导部分。当T(n)为多项式时,可只取其最高次幂项,且它的系数也可略去不写。其中O(1)为常数数量级。

- 定理：若$T(n)=a_m n^m+a_{m-1}n^{m-1}+\ldots+a_1 n+a_0$是一个m次多项式，则$T(n)=O(n^m)$

- 大Ω：对算法复杂度在最好情况下做出估计
- 大Θ：对算法复杂度的准确估计

- 以下是最常用的计算算法时间的多项式，其关系为：

  $O(1)<O(\log n)<O(n)<O(n\log n)<O(n^2)<O(n^3)<O(2^n)<O(n!)<O(n^n)$



当n取得很大时，指数时间算法和多项式时间算法在所需时间上非常悬殊。因此，只要有人能将现有指数时间算法中的任何一个算法化简为多项式时间算法，那就取得了一个伟大的成就。

- 有序数组二分查找 $O(\log_2 n)$
- 无序数组任意元素查找 $O(n)$

N	≈ 1 000	≈ 1 000 000	≈ 1 000 000 000
$\log_2$ **N**	10	Only 20	Only 30

例1: {++x; s=0;}

将x自增看成是基本操作，则语句频度为１，即时间复杂度为O(1)

如果将s=0也看成是基本操作，则语句频度为２，其时间复杂度仍为O(1)，即常量阶。

例2、for(i=1;i<=n;++i)

{++x;s+=x;}

语句频度为：2n　其时间复杂度为：O(n)

即时间复杂度为线性阶。

例3、for(i=1;i<=n;++i)

for(j=1;j<=n;++j)

{++x;s+=x;}

语句频度为：$2n^2$

其时间复杂度为：$O(n^2)$

即时间复杂度为平方阶。

例4

- 计算下面交换i和j内容程序段的时间复杂性。

```
temp=i;
i=j;
j=temp;
```

- 解：以上三条单个语句均执行1次，该程序段的执行时间是一个与问题n无关的常数，因此，算法的时间复杂度为常数阶，记作T(n)=O(1).

例5

- 计算下面求累加和程序段的时间复杂性

（1） sum=0；　　（一次）

（2） for(i=1;i<=n;i++)　（n次）

（3） for(j=1;j<=n;j++)　（n^2次）

（4） sum++；　　（n^2次）

- 解：$T(n)=2n^2+n+1=O(n^2)$

返回

时间复杂度分析方法

- 迭代：级数求和
- 递归：递归跟踪+递推方程
- 猜测+验证

算法的存储量包括:

1. **输入数据**所占空间
2. **程序本身**所占空间;
3. **辅助变量**所占空间。

- **空间复杂度度量**

一个算法的空间效率是指在算法的执行过程中，占据的辅助空间数量。辅助空间就是除算法代码本身和输入输出数据所占据的空间外，算法临时开辟的存储空间单元。在有些算法中，占据辅助空间的数量与所处理的数据量有关，而有些却无关。后一种是较理想的情况。在设计算法时，应该注意空间效率。

记作:

$$S(n)=O(f(n))$$

其中n为问题的规模(或大小)

对于空间复杂度为O(1)的算法称为原地工作或就地工作算法.

原地工作的含义是指不需要额外的辅助空间吗?

- 在早期计算机的内存容量很小、磁盘容量也很小的时代，降低空间复杂度是首要问题。

- 随着计算机的硬件规模越来越大，存储容量不是问题之后，时间复杂度变成首要矛盾。

练习1：求以下算法的时间复杂度

- Void fun(int n)
- { int i,j,x=0;
- for (i=1;i<n;i++)
- for(j=n;j>=i+1;j--)
- x++;
- }

$$T(n)=\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}1=\sum_{i=1}^{n-1}(n-i)=n(n-1)/2=O(n^2)$$

练习2：求以下算法的时间复杂度

- Void fun(int n)
- { int x=2;
- while (x<n/2)
- x=x*2;
- }

$2^{T(n)} < n/2$ $\quad\quad$ $T(n) < \log_2 n/2 = O(\log_2 n)$

练习3：求以下算法的时间复杂度 $\sqrt[3]{n}$

```
Void fun(int n)
{ int i=0;
      i++;
}
   while (i*i*i<=n)
```