



数据结构与算法

院 系：智能科学与技术学院

时 间：2024-2025 秋季

任课教师：王贝贝



授课时间

90311202-数据结构与算法02班 学分: 3 任务状态: 正常 Data Structure and Algorithms

学年学期: 2024-2025学年 第1学期(秋)

选课人数: 154

上课教师: 王贝贝

年级专业: 2023智能科学与技术

时间地点: 周四 2-4节 1-17周 南雍-西426

考试信息: 暂无

总容量/开放人数: 160/10

编辑教学班

编辑周历

学生名单

查看大纲

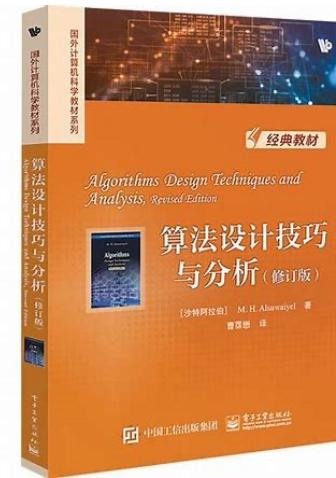
编辑教材

- 周四: 2-4节课 (理论课)
- 助教: 祝亦欣、广坤鑫
- 南雍楼东507, Beibei.wang@nju.edu.cn



教材

- 《数据结构（用面向对象方法与C++描述）》 / 殷人昆等 / 清华大学出版社
- 《算法设计技巧与分析》 M. H. Alsuwaiyel (沙特) 著, 电子工业出版社, 吴伟昶等译, 2016





课程QQ群

数据结构与算法课...

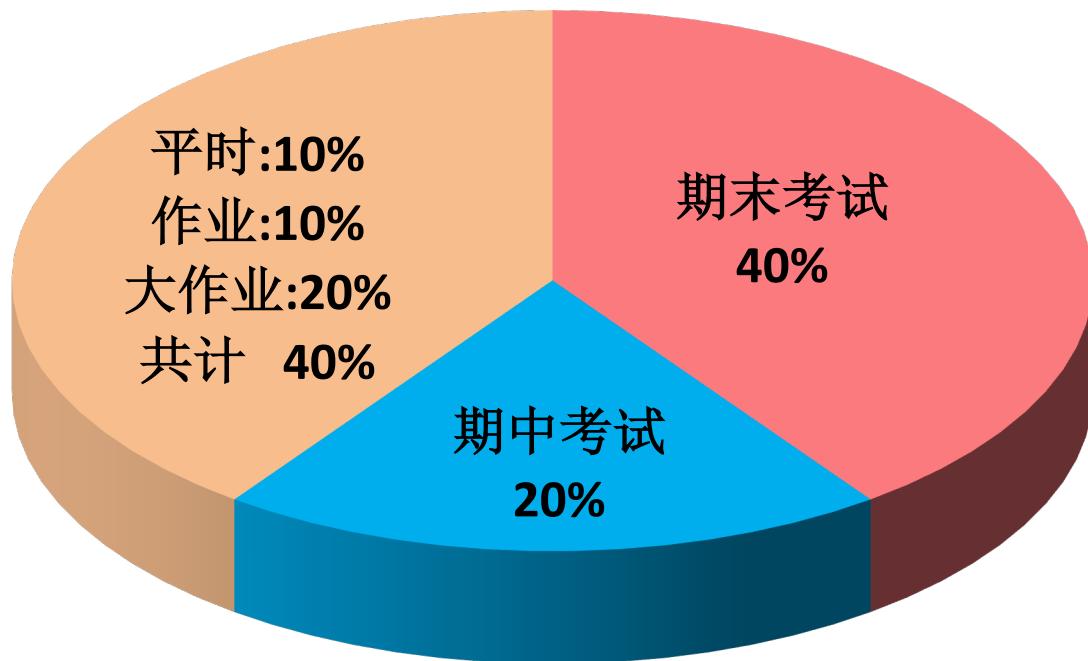
群号: 315156408



扫一扫二维码，加入群聊



课程成绩比例





几个问题

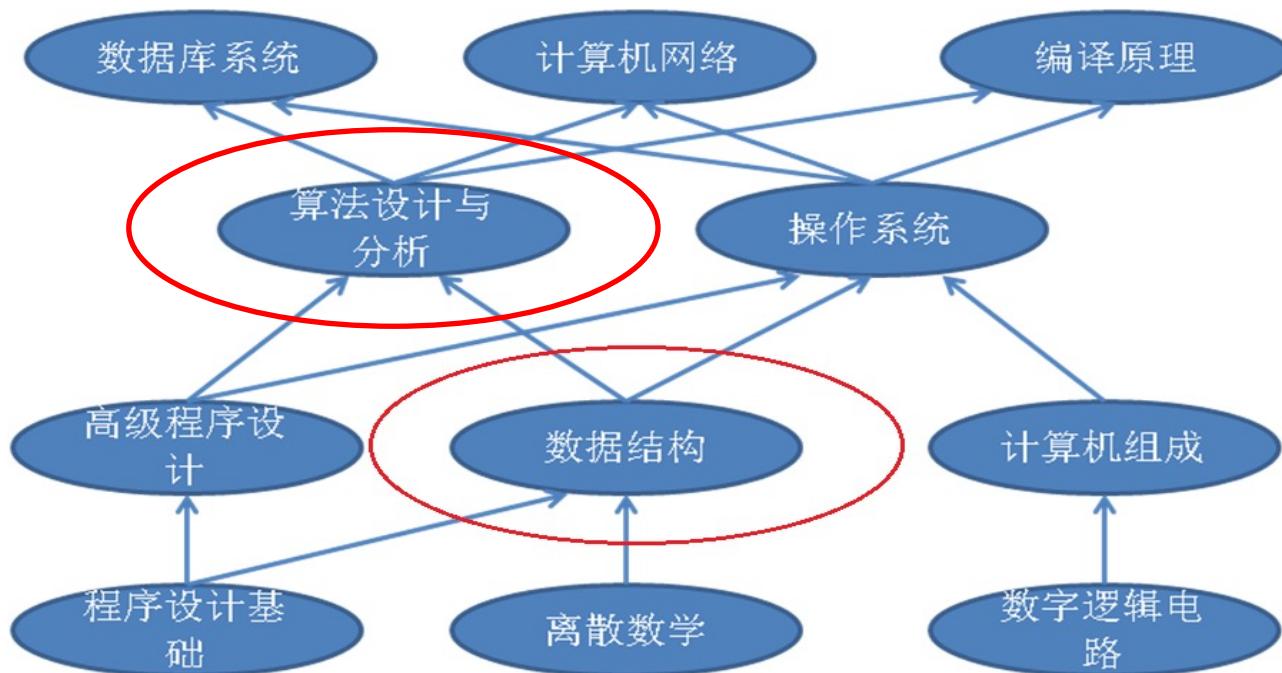
不学数据结构是否就不能编程
序?

数据结构的前导（先修）课程
是什么？

通过数据结构与算法课程的学
习提高自己什么方面的能力？



从专业知识结构来看





数据结构对相关课程的支撑

数据结构

计算机网络（图、最小生成树、散列表）

编译原理（字符串、栈、散列表）

数据库（线性表、多链表、B+树）

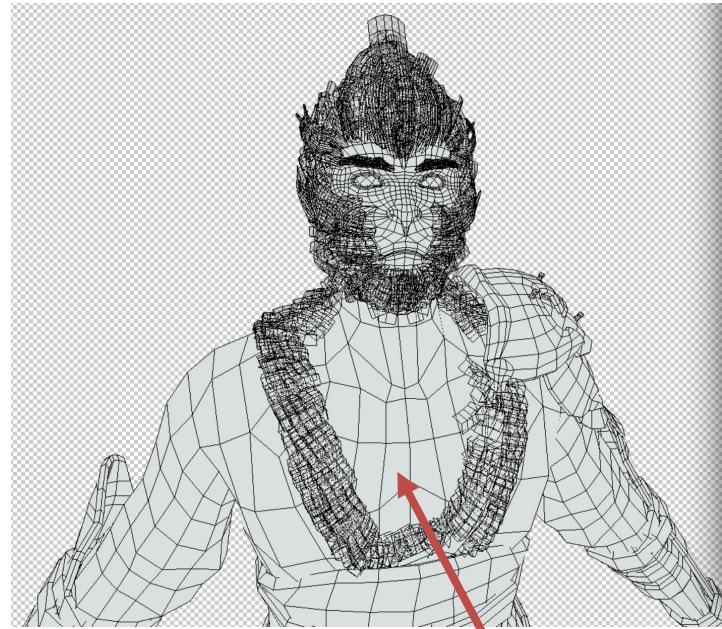
人工智能（广义表、集合、图、搜索树）

运筹学（图、关键路径）

图形图像（队、栈、图、检索）



数据结构与算法的应用



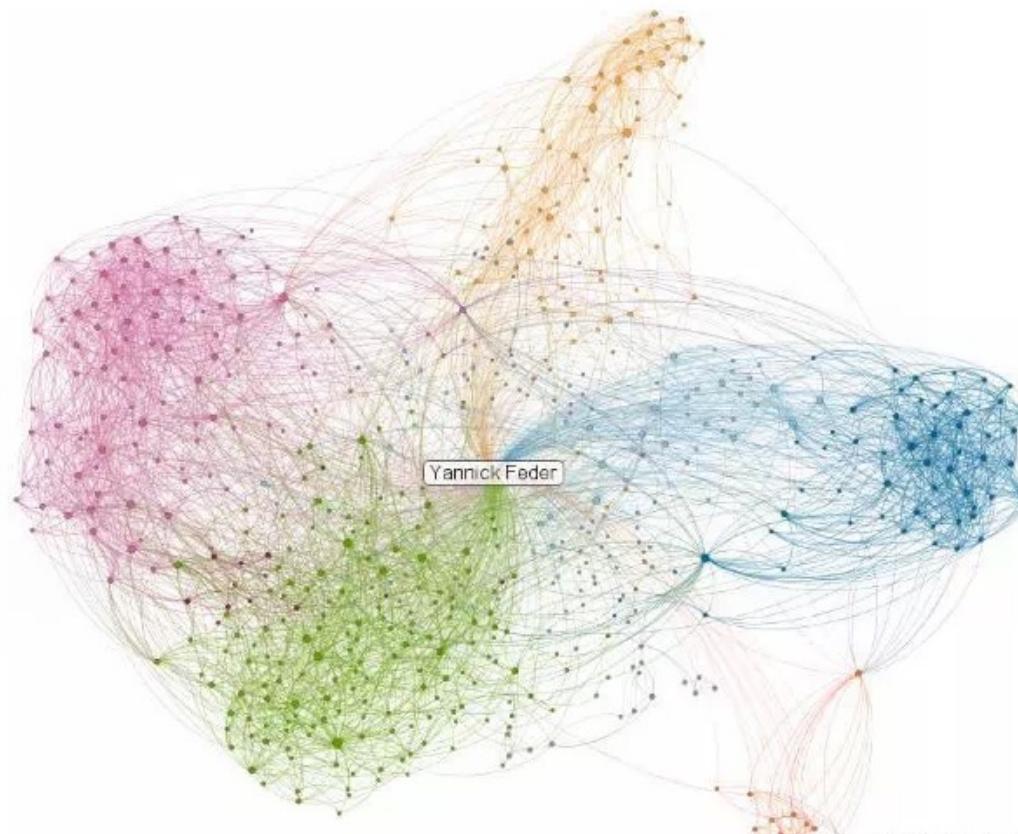
➤ 光线跟哪个四边形相交？

✓ 遍历所有的四边形？ RTX 4090也跑不动了。

模型数据: <https://sketchfab.com/3d-models/wu-kong-black-myth-09019ca079234fc19ef90499c967b60d>



数据结构与算法的应用



你跟Jensen Huang之间的最短距离是多少？

图片来源:

<https://blog.csdn.net/pmcuff2008/article/details/111471037>



第一章 绪论

本章内容：

1.1 什么是数据结构

1.2 基本概念和术语

1.3 抽象数据类型

1.4 算法和算法分析



例1 学生信息管理系统

- (1) 计算机处理的对象是各种表；
- (2) 元素之间的逻辑关系是线性关系；
- (3) 施加于对象上的操作有遍历、查找、插入、删除等。

学号	姓名	性别	专业	住址
04180101	侯亮平	男	计算机科学与技术	北京
04180102	高小琴	女	计算机科学与技术	深圳
04180103	陆亦可	女	计算机科学与技术	珠海
04180104	陈海	男	计算机科学与技术	上海
04180105	李达康	男	计算机科学与技术	杭州
04180106	高育良	男	计算机科学与技术	南京
04180107	赵东来	男	计算机科学与技术	武汉
04180108	陈岩石	男	计算机科学与技术	重庆
04180109	沙瑞金	男	计算机科学与技术	珠海



例2 人机博弈

- (1) 计算机处理的对象是树型结构；
- (2) 元素间的关系是一种一对多的层次关系；
- (3) 施加于对象上的操作有遍历、查找、插入、删除等。

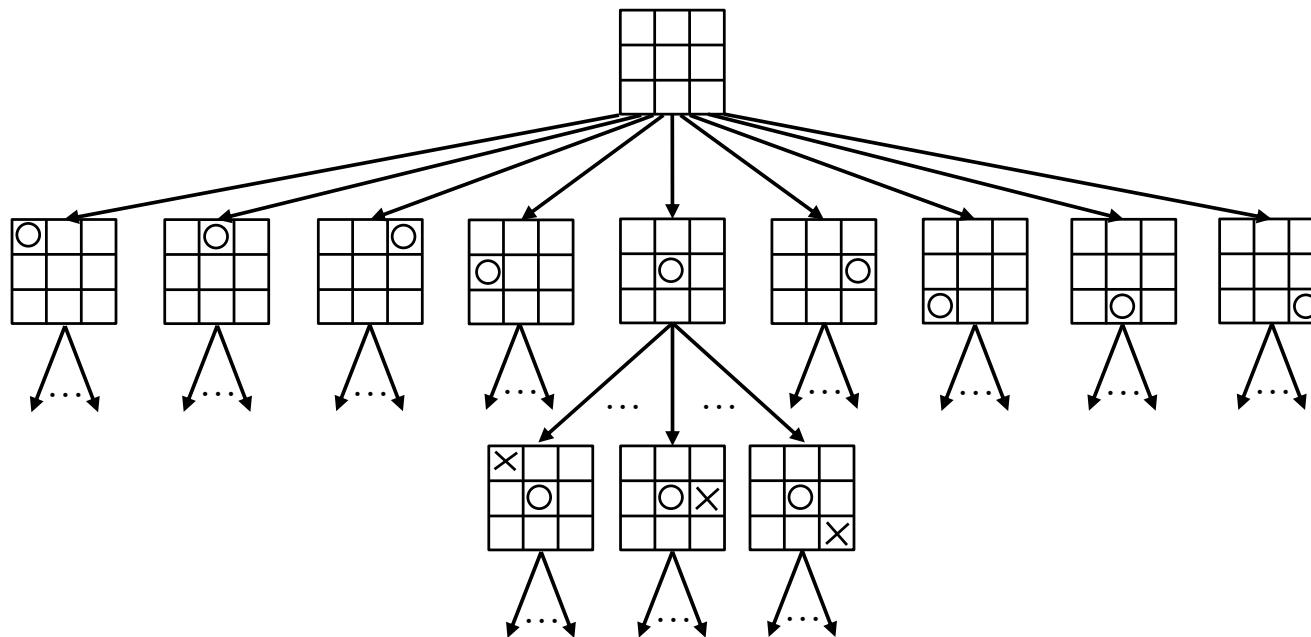
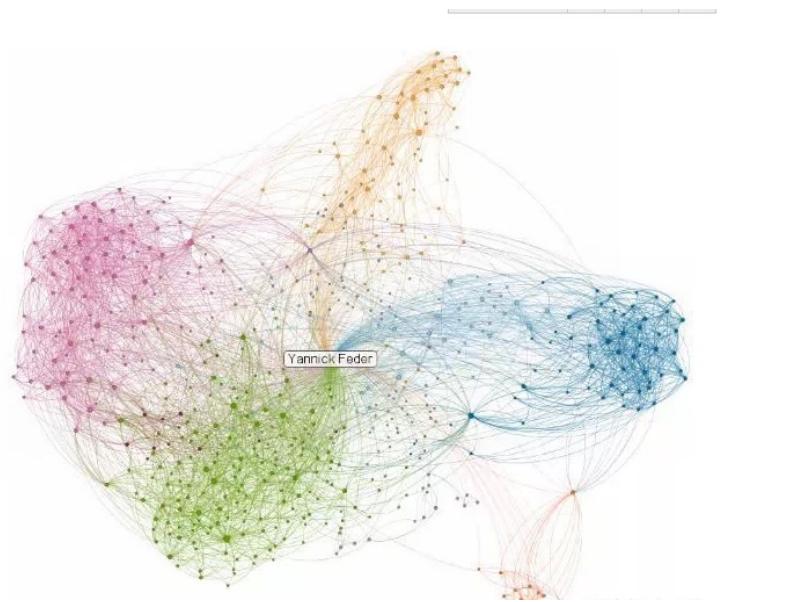


图 1.1 井字棋对弈树



例3 社交网络

- (1) 计算机处理的对象是各种图；
- (2) 元素间的关系是复杂的图形或网状关系，是一种多对多的关系；
- (3) 施加于对象上的操作有遍历、查找、插入、删除等。



图片来源: <https://blog.csdn.net/pmccaff2008/article/details/111471037>



什么是数据结构

- 描述这类**非数值**计算问题的数学模型不再是数学方程，而是诸如**表、树、图**之类的数据结构。
- 数据结构是一门关注（**非数值计算的**）程序设计问题中所出现的计算机**操作对象**以及它们之间的**关系**和**操作**的学科。



第一章 绪论

1.1 什么是数据结构

1.2 基本概念和术语

1.3 抽象数据类型

1.4 算法和算法分析



1.2 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。



1.2 基本概念和术语

➤ 数据元素（Data Element）是数据中的一个“个体”，是数据的基本单位。在有些情况下数据元素也称为元素、结点、顶点、记录等。数据元素用于完整地描述一个对象。
如：一个学生记录、棋盘中的一个格局、图中的一个顶点等。

学号	姓名	性别	专业	住址
04180101	侯亮平	男	计算机科学与技术	北京
04180102	高小琴	女	计算机科学与技术	深圳



1.2 基本概念和术语

➤ 数据项 (Data Item)是组成数据元素的有特定意义的不可分割的最小单位。如构成一个数据元素的字段、域、属性等都可称之为数据项。数据元素是数据项的集合。

如：学生信息表中的学号、姓名、性别、专业等。

学号	姓名	性别	专业	住址
04180101	侯亮平	男	计算机科学与技术	北京
04180102	高小琴	女	计算机科学与技术	深圳





1.2 基本概念和术语

➤ 数据对象 (Data Object) 是具有相同性质的数据元素的集合，是数据的一个子集。

如：

-
-
-

学号	姓名	性别	专业	住址
04180101	侯亮平	男	计算机科学与技术	北京
04180102	高小琴	女	计算机科学与技术	深圳
04180103	陆亦可	女	计算机科学与技术	珠海
04180104	陈海	男	计算机科学与技术	上海
04180105	李达康	男	计算机科学与技术	杭州
04180106	高育良	男	计算机科学与技术	南京
04180107	赵东来	男	计算机科学与技术	武汉
04180108	陈岩石	男	计算机科学与技术	重庆
04180109	沙瑞金	男	计算机科学与技术	珠海

• }
, ...,



1.2 基本概念和术语

- 表格表示了一个数据对象，每一行是一个**数据元素**，行中的每一列是一个**数据项**
- 数据元素是数据项的集合，数据对象是数据元素的集合

学号	姓名	性别	专业	住址
04180101	侯亮平	男	计算机科学与技术	北京
04180102	高小琴	女	计算机科学与技术	深圳
04180103	陆亦可	女	计算机科学与技术	珠海
04180104	陈海	男	计算机科学与技术	上海
04180105	李达康	男	计算机科学与技术	杭州
04180106	高育良	男	计算机科学与技术	南京
04180107	赵东来	男	计算机科学与技术	武汉

数据项 (指向学号、姓名、性别、专业、住址)

数据元素 (指向第5行整个行框)



数据结构的形式定义

数据结构由某一数据对象及该对象中所有数据成员之间的关系组成。记为：

$$\text{Data_Structure} = \{D, R\}$$

其中， D是某一数据对象， R是该对象中所有数据成员之间的关系的有限集合。



数据结构举例

例如：班级的数据结构

Class = (D, S)

数据集合 : D = { a, b₁, ..., b_n, c₁, ..., c_n, d₁, ..., d_n }

关系集合 : S = { R₁, R₂ }

R₁ = { <a, b₁>, <a, c₁>, <a, d₁> }

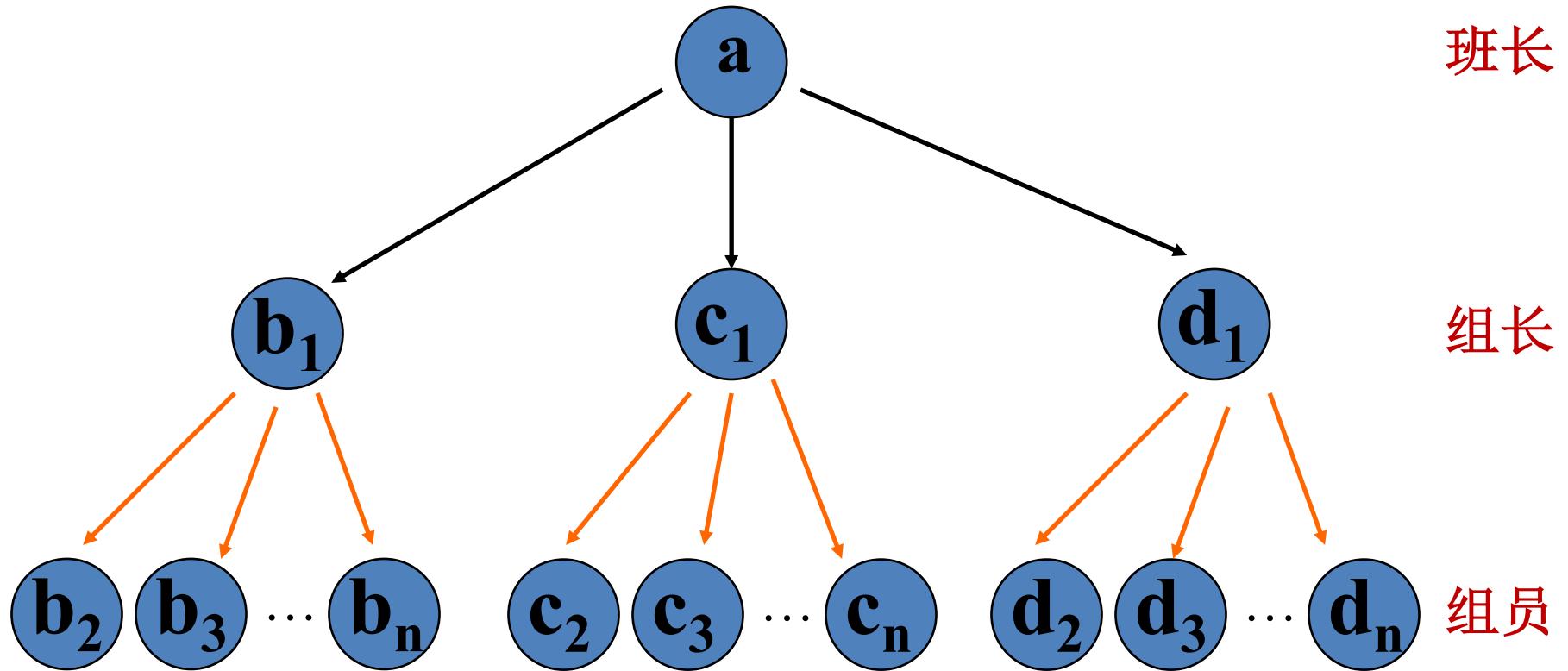
//班长-组长

R₂ = { <b_i, b_i>, <c_i, c_i>, <d_i, d_i> | i = 2, 3, ..., n }

//组长-组员



数据结构举例



班级 Class 的逻辑结构的图示



数据结构关注的三个方面

- 1. 逻辑结构：**指从解决问题的需要出发，为实现必要的功能所建立的数据结构，它属于用户的视图，是面向问题的，表示数据元素之间的逻辑关系；
- 2. 存储结构：**是指数据如何在计算机中存放，是数据逻辑结构的物理存储映象，是属于具体**实现的视图**，是面向计算机的。



数据结构关注的三个方面

3. **运算**: 作用于数据结构上的运算，及实现方法（查找、插入、删除、更新等）

例:

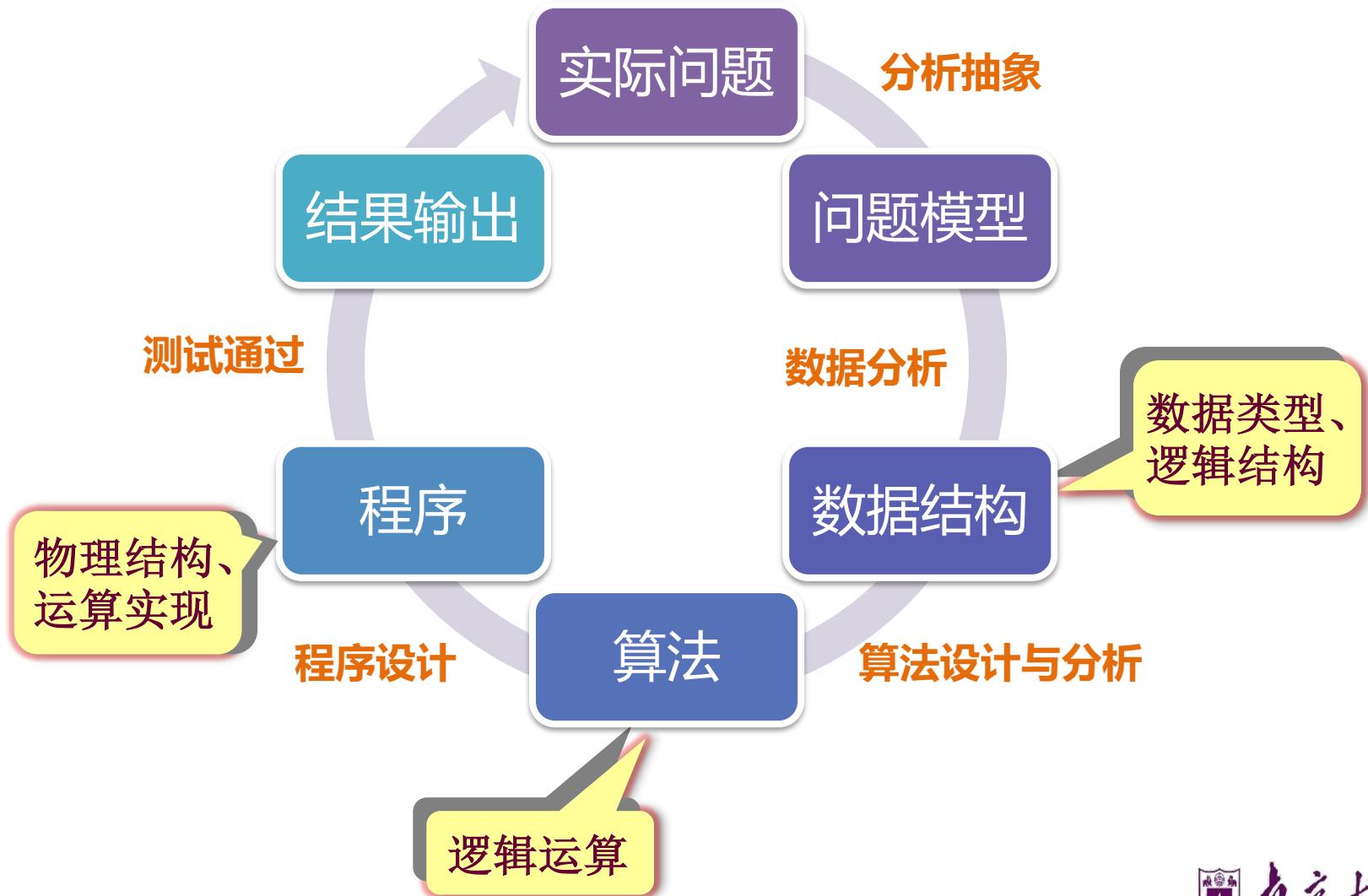
学生表：逻辑结构——线性表

物理结构——数组，链表

操作——插入，删除，查找



求解问题的过程





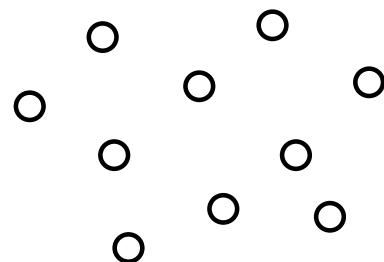
数据结构的逻辑结构

- 逻辑结构是对数据成员之间的逻辑关系的描述，它可以用一个数据成员的集合和定义在此集合上的若干关系来表示。
- 线性结构：表、栈、队列
- 非线性结构
 - 层次结构：树，二叉树
 - 网状结构：图
 - 其它：集合

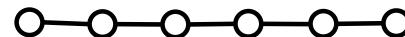


数据的逻辑结构

- **集合结构：**元素间的次序是任意的。元素之间除了“属于同一集合”的联系外没有其他的关系。
- **线性结构：**数据元素的有序序列。除了第一个和最后一个元素外，其余元素都有一个前趋和一个后继，1对1。



(a) 集合结构

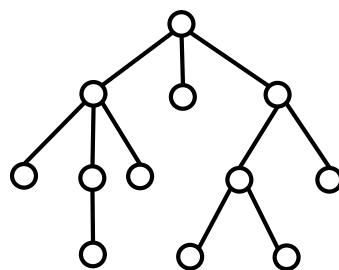


(b) 线性结构

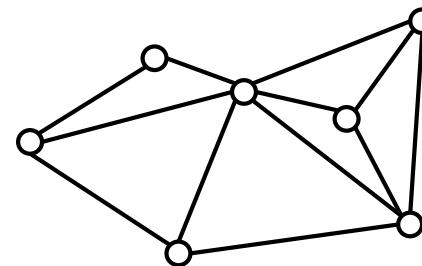


数据的逻辑结构

- **树型结构：**除了根元素外，每个节点有且仅有一个前趋，后继数目不限，1对多。
- **图型结构：**每个元素的前趋和后继数目都不限，多对多。



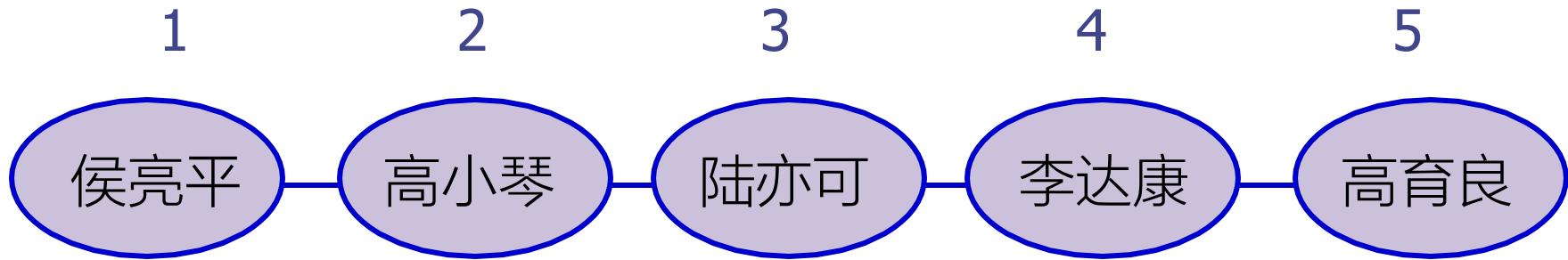
(c) 树型结构



(d) 图型结构



线性结构

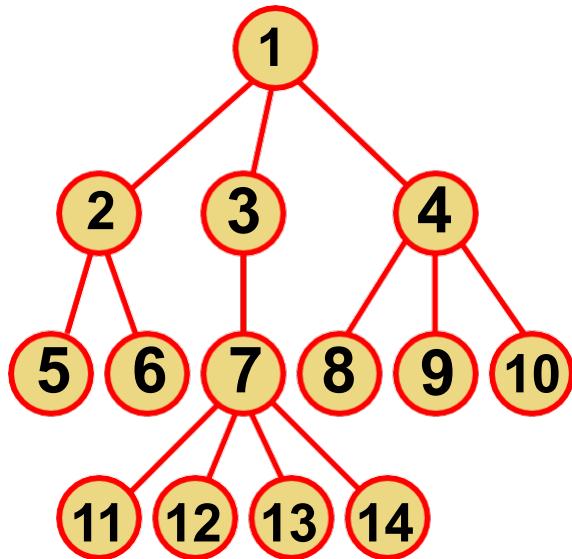


元素之间为**一对一的线性关系**，第一个元素无直接前驱，最后一个元素无直接后继，其余元素都有一个直接前驱和直接后继。

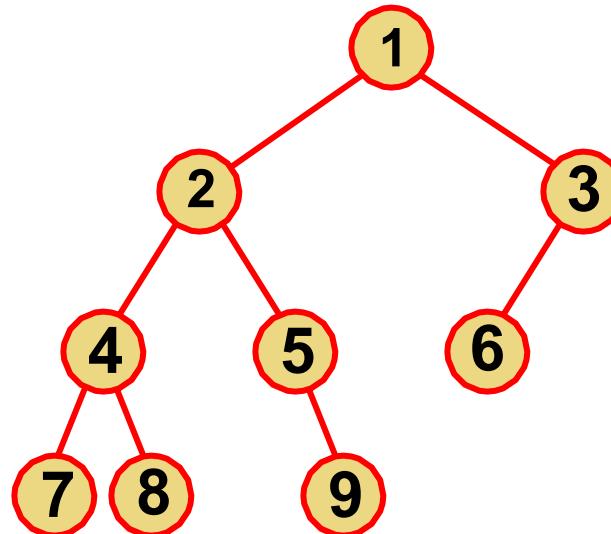


树形结构

树



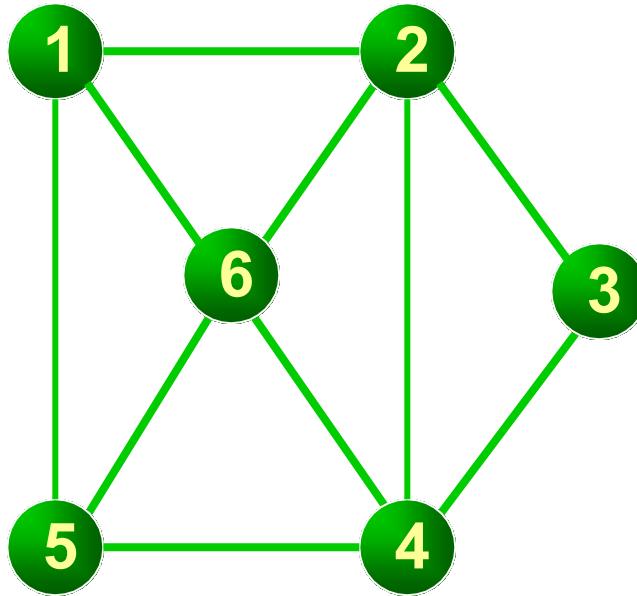
二叉树



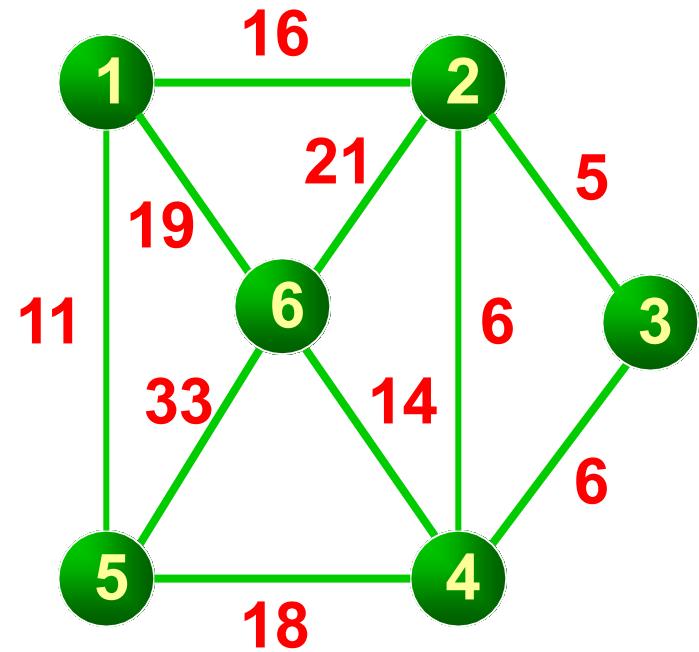
元素之间为一对多非线性关系的非线性结构称为**树结构**，除根结点无直接前驱、有多个直接后继外，其余元素均有一个直接前驱或多个直接后继。



图结构



图结构



加权图结构

元素之间为**多对多非线性关系**的非线性结构，每个元素均
有多个直接前驱或多个直接后继。



数据结构的物理结构

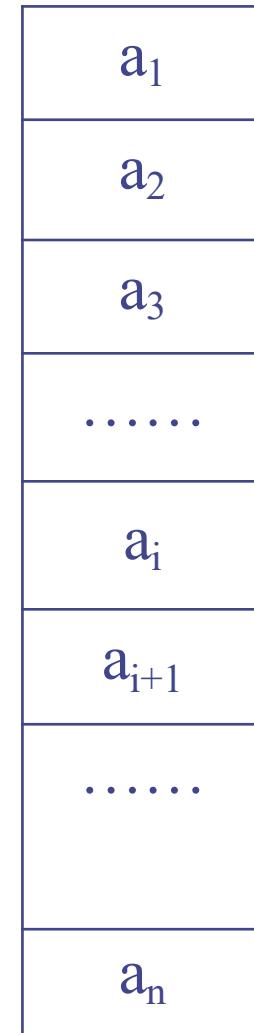
• 物理结构是逻辑结构在计算机中的表示和实现，故又称“存储结构”。

- 顺序存储结构
- 链接存储结构
- 索引存储方式
- 哈希（散列）存储方式



数据的存储结构

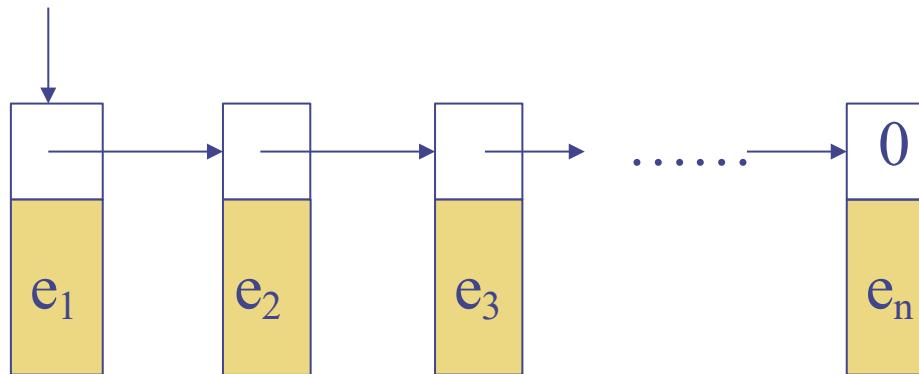
- **顺序存储：**所有元素存放在一片连续的存储单元中，逻辑上相邻的元素存放到计算机内存仍然相邻。





数据的存储结构

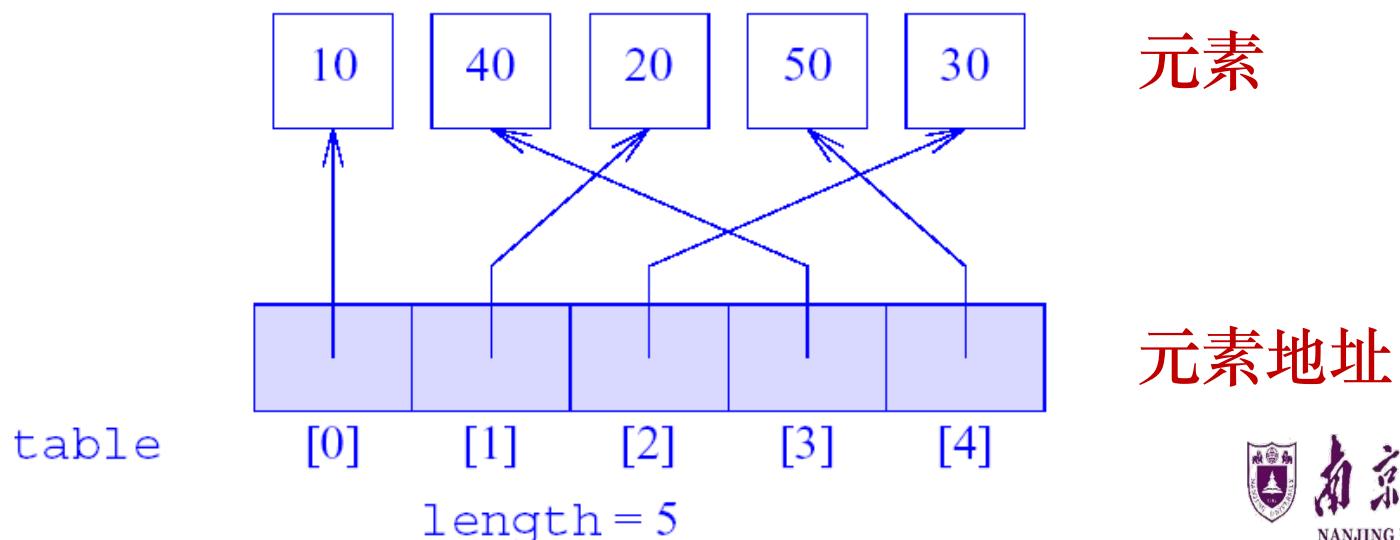
- **链式存储：**所有元素依次存放在可以不连续的存储单元中，元素之间的关系可以通过指针(地址)表示。
- 注：逻辑上相邻的元素其物理位置不要求相邻。





数据的存储结构

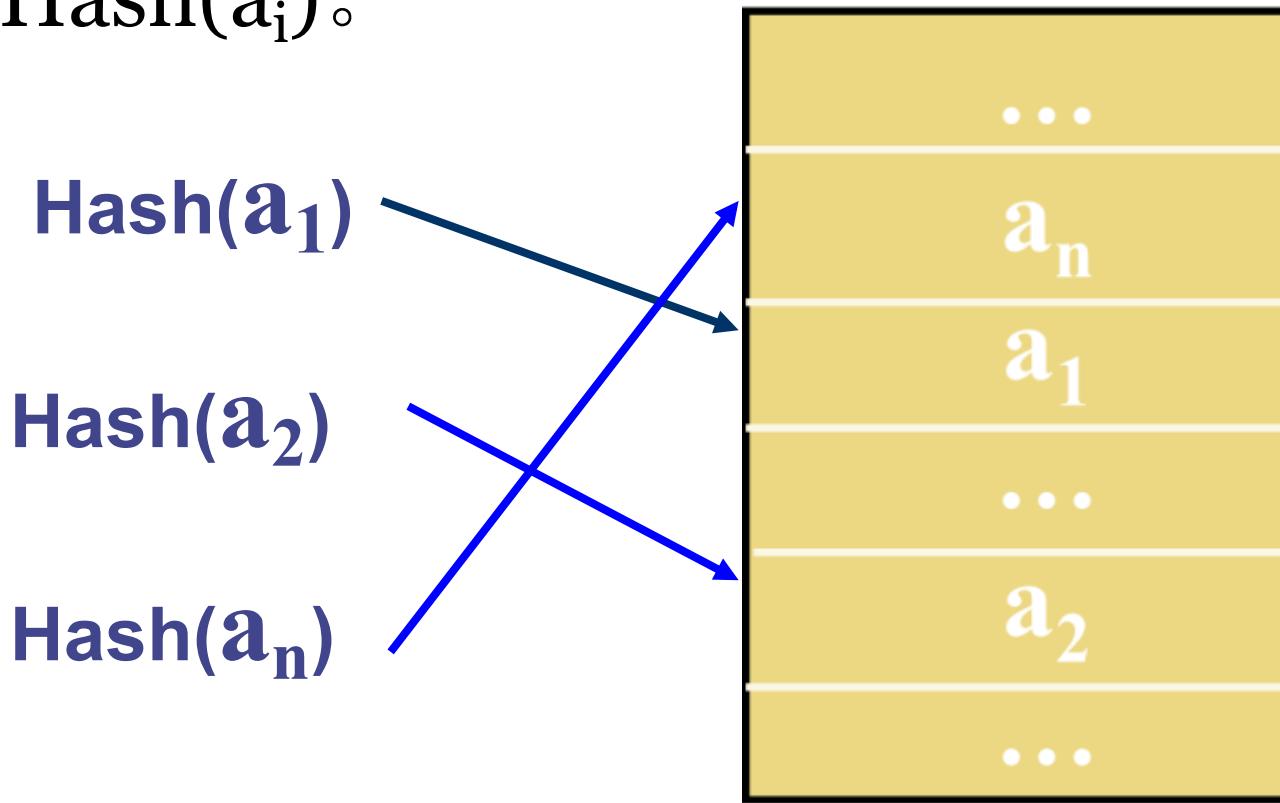
• 索引存储（间接寻址）：元素表中的每个元素也可以存储在存储器的不同区域中；元素地址则被收集在一张表中，该表的第 i 项指向元素表中的第 i 个元素，这张表是一个用来存储元素地址的表。





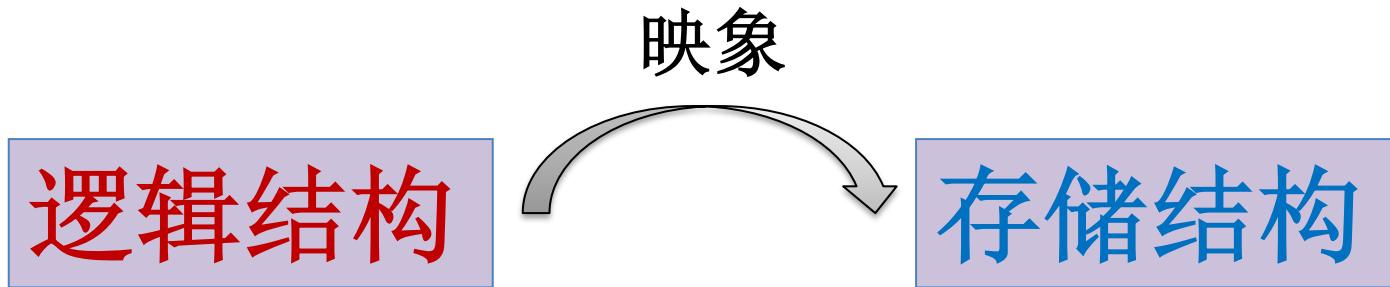
数据的存储结构

- **散列存储：**通过构造散列函数，用函数的值来确定元素存放的地址。即：元素 a_i 的地址= $\text{Hash}(a_i)$ 。





逻辑结构和存储结构的关系



- 任何一个**算法**的设计取决于选定的**逻辑结构**；
- 算法的最终实现依赖于采用的**存储结构**。



数据的运算

- (1) **创建**: 创建某种数据结构;
- (2) **清除**: 删掉数据结构;
- (3) **插入**: 在数据结构指定的位置上插入一个新元素;
- (4) **删除**: 将数据结构中的某个元素删去;

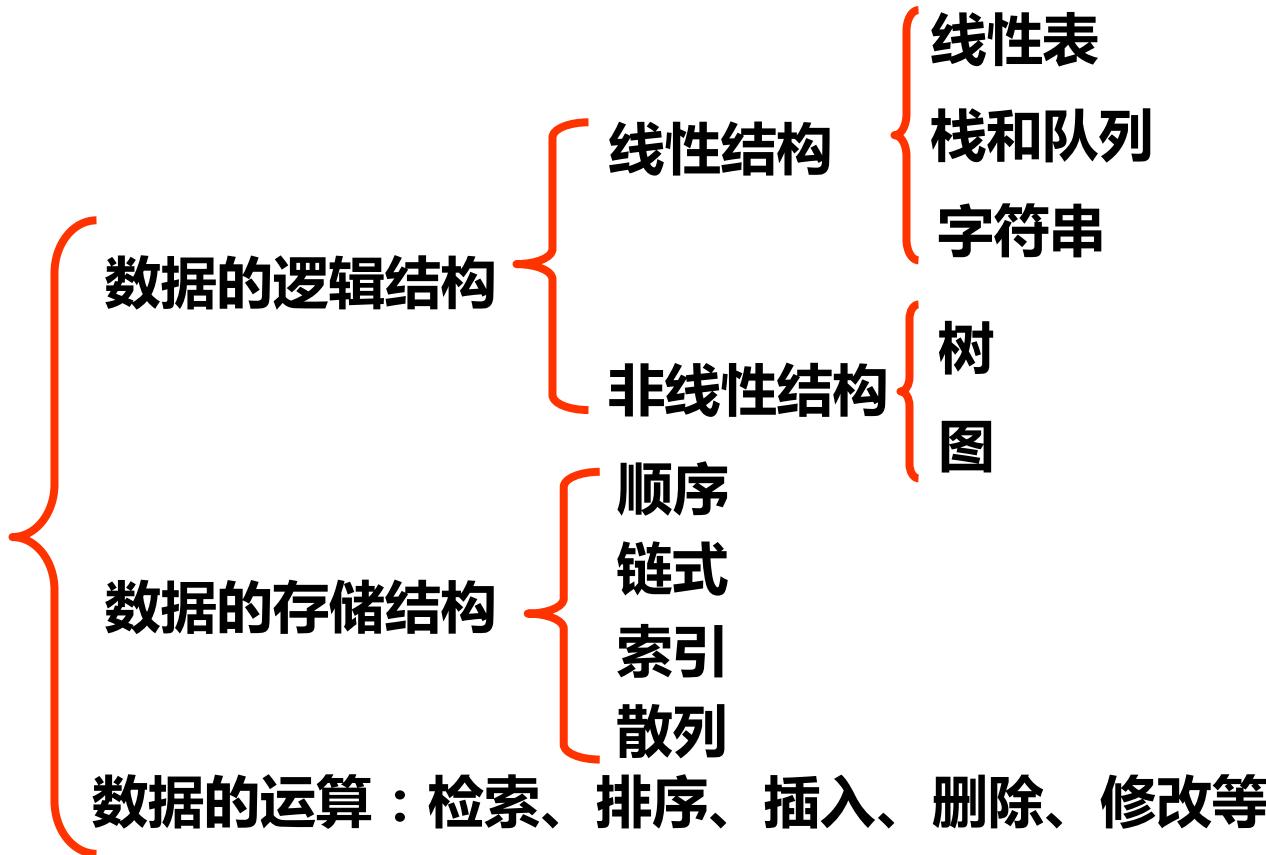


数据的运算

- (5) **搜索**: 在数据结构中搜索满足特定条件的元素;
- (6) **更新**: 修改数据结构中的某个元素的值;
- (7) **访问**: 访问数据结构中的某个元素;
- (8) **遍历**: 按照某种次序访问数据结构中的每一元素, 使每个元素恰好被访问一次;



数据结构关注的三个方面





第一章 绪论

1.1 什么是数据结构

1.2 基本概念和术语

1.3 抽象数据类型

1.4 算法和算法分析



数据类型

- C语言中的数据类型

char int float double void

字符型 整型 浮点型 双精度型 无值

- 数据类型

定义：一组性质相同的值的集合，以及定义于这个值集合上的一组操作的总称。



数据类型

不同类型的变量，其所能取的值的范围不同，所能进行的操作不同。

例如：整型 (**int**)

值的范围是： -32768 ~ 32767 (16位)

操作是： +, -, *, /, %



抽象数据类型 (ADT: Abstract Data Type)

- 由用户定义，用以表示应用问题的数据模型
- 由基本的数据类型组成，并包括一组相关的服务（或称操作）
- 抽象：抽取反映问题本质的东西，不涉及具体实现细节
- 支持了逻辑设计和物理实现的分离，支持封装和信息隐蔽



抽象数据类型ADT

- 和数据结构的形式定义相对应，抽象数据类型可用以下三元组表示：（D， R， P）

D是数据对象，即具有相同特性的数据元素的集合。R是D上的关系集合，P是对D的基本操作集合。

- 抽象数据类型的伪代码定义格式：

```
ADT 抽象数据类型名 {  
    数据对象D: <数据对象的定义>  
    数据关系R: <数据关系的定义>  
    基本操作P: <基本操作的定义>  
} ADT 抽象数据类型名
```



抽象数据类型ADT

```
1 ADT NaturalNumber IS
2 /*objects: 一个整数的有序子集合,它开始于0,结束于机器能表示的最大整数(MaxInt)。*/
3 {
4     Function:对于所有的属于NaturalNumber x, y, +、-、<、==、=等可用。
5
6     Zero( ): NaturalNumber          return 0
7     IsZero(x): Boolean            if (x==0) return true else return false
8
9
10    Add (x, y): NaturalNumber      if (x+y<=MaxInt) return x+y else return MaxInt
11    Subtract(x, y): NaturalNumber if (x < y) return 0 else return x - y
12
13    Equal(x, y): Boolean          if (x==y) return true else return false
14
15    Successor(x): NaturalNumber   if (x==MaxInt) return x else return x+1
16 }//NaturalNumber
```



抽象数据类型的两种视图

- **设计者的角度:**

根据问题来定义抽象数据类型所包含的信息，给出其相关功能的实现，并提供公共界面的接口。

- **用户的角度:**

使用公共界面的接口对抽象数据类型进行操作，不需要考虑其物理实现。对于外部用户来说，抽象数据类型应该是一个黑盒子。



抽象数据类型ADT的特点

- 降低了软件设计的复杂性
- 提高了程序的可读性和可维护性
- 程序的正确性容易保证



作为ADT的C++类

- 面向对象的概念

面向对象 = 对象 + 类 + 继承 + 通信

面向对象方法中类的定义充分体现了抽象数据类型的思想



作为ADT的C++类（例）

```
2 #include "author.h"
3 #include "publisher.h"
4
5
6 class Book {
7 public:
8     typedef const Author* AuthorPosition;
9
10    Book (Author);           // for books with single authors
11    Book (const Author[], int nAuthors); // for books with multiple authors
12
13
14    /**/std::string getTitle() const;
15    void setTitle(std::string theTitle);
16
17    int getNumberOfAuthors() const;
18
19    std::string getISBN() const;
20    void setISBN(std::string id);
21
22    Publisher getPublisher() const;
23    void setPublisher(const Publisher& publ);/**/
24
25    AuthorPosition begin() const;
26    AuthorPosition end() const;
27
28    void addAuthor (AuthorPosition at, const Author& author);
29    void removeAuthor (AuthorPosition at);
30
31 private:
32
33     std::string title;
34     int numAuthors;
35     std::string isbn;
36     Publisher publisher;
37
38     static const int MAXAUTHORS = 12;
39     Author authors[MAXAUTHORS];
40
41 };
42
```

```
BookInSeries: public Book {
public:
    std::string getSeriesTitle() const;
    void putSeriesTitle(std::string theSeries);

    int getVolume() const;
    void putVolume(int);
private:
    std::string seriesTitle;
    int volume;
```

参考代码：

[https://www.cs.odu.edu/~zeil/cs330/late
st/Public/implementingADTS/index.html](https://www.cs.odu.edu/~zeil/cs330/latest/Public/implementingADTS/index.html)



用C++语言描述面向对象程序

- C++的函数特征
- C++的数据声明
- C++的作用域
- C++的类
- C++的对象
- C++的输入/输出
- C++的函数
- C++的参数传递
- C++的函数名重载和操作符重载
- C++的动态存储分配
- 友元(friend)函数
- 内联(inline)函数
- 结构(struct)与类
- 联合(Union)与类



第一章 绪论

1.1 什么是数据结构

1.2 基本概念和术语

1.3 抽象数据类型

1.4 算法和算法分析



算法的历史背景简介

阶段1：二十世纪30年代，能否使用一个有效的过程(相当于现在算法的概念)来求解一个给定的问题一直是人们所关注的。当时的焦点是将问题进行分类：**可解或是不可解**。关注问题是否可以求解的理论称为**可计算理论**(theory of computation)。出现了一系列的计算模型，例如：calculus of Church、Post machines of Post、Turing machines of Turing、RAM model of computation。



算法的历史背景简介

阶段2：慢慢地，人们需要考虑在**有限资源**的条件下高效地解决问题（如目标拦截）。这就导致了**计算复杂度**（computational complexity）这一新学科的诞生。重点是研究解决可求解问题时所需要的资源（主要是时间和空间复杂性）。有时候，其他的资源也需要考虑，例如，通信代价、需要使用的处理器的个数（使用并行计算模型）等等。



例：搜索

- 给定已经排好序(不妨假设为非降序)的n个元素A[1...n] ,现在要判定一个给定的元素x是否在此数组中出现。
 - 方法1：顺序搜索
 - 方法2：二分搜索



方法一：顺序搜索

输入： 非降序排列的数组A[1...n]和元素x

输出： 如果 $x=A[j]$, $1 \leq j \leq n$, 则输出j, 否则输出-1.

1. $j \leftarrow -1$

2. for $i = 1$ to n

3. if $x = A[j]$

4. $j \leftarrow i$

5. break;

6. end if

7. end for

8. return j



方法二：二分搜索

输入： 非降序排列的数组A[1...n]和元素x

输出： 如果 $x=A[j]$, $1 \leq j \leq n$, 则输出j, 否则输出-1.

1. $low \leftarrow 1; high \leftarrow n; j \leftarrow -1$
2. while($low \leq high$) and ($j = -1$)
 3. $mid \leftarrow \lfloor (low+high)/2 \rfloor$
 4. if $x = A[mid]$ then $j \leftarrow mid$
 5. else if $x < A[mid]$ then $high \leftarrow mid-1$
 6. else $low \leftarrow mid+1$
7. end while
8. return j



分析

- 最好情形： 比较1次
- 最坏情形： 比较 $\lfloor \log n \rfloor + 1$ 次
 - 每次循环都要抛弃一些元素，例如第二次循环时，剩余元素为 $A[1..mid-1]$ 或 $A[mid+1..n]$ ，不妨设为 $A[mid+1..n]$ ，则剩余的元素个数是 $\lfloor n/2 \rfloor$
 - 第 j 次while循环时，剩余元素的个数是 $\lfloor n/2^{j-1} \rfloor$
 - 或者找到 x ，或者程序在子序列长度达到1时终止搜索，此时 $\lfloor n/2^{j-1} \rfloor = 1 \Leftrightarrow 1 \leq n/2^{j-1} < 2 \Leftrightarrow 2^{j-1} \leq n < 2^j \Leftrightarrow \log n < j \leq \log n + 1 \Leftrightarrow j = \lfloor \log n \rfloor + 1$



数据结构、算法和程序设计

程序设计的实质是对确定的问题选择一种好的**结构**，加上设计一种好的**算法**。

数据结构 + 算法 = 程序设计

问题的
数学模
型

处理问题
的策略

为计算机处理
问题编制一组
指令集



算法的定义

算法: 一个有穷的指令集，这些指令为解决某一特定任务规定了一个运算序列。

(算法是对特定问题求解步骤的一种描述)



算法的描述

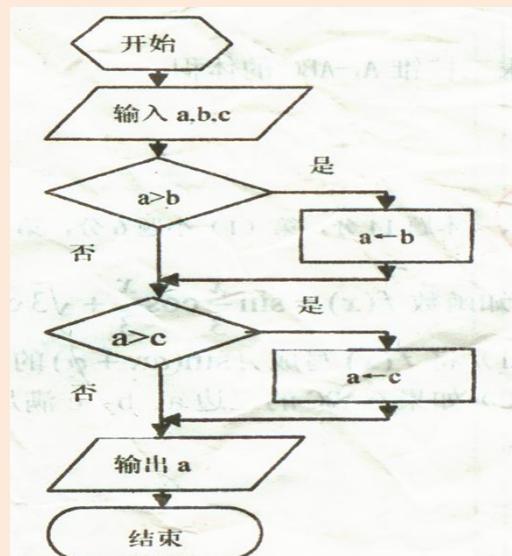
常用的算法的描述方法

- 自然语言
- 流程图
- 伪语言
- 类语言
- 类C语言

特定的
包括程序

求两个正整数的最大公约数算法(设给定的两个正整数为m和n):

1. 输入两个正整数m和n;
2. 若 $m < n$, 则交换m和n;
3. 以m减去n, 令所得的差为r;
4. 若 $r = 0$, 则输出结果n, 算法结束; 否则, 继续步骤3;
5. 若 $r < n$, 则令 $m = n, n = r$; 否则令 $m = r$, 并返回步骤3继续进行.



例如，类PASCAL



算法必须满足的五个重要特性

- **输入** 有0个或多个输入
- **输出** 有一个或多个输出(处理结果)
- **确定性¹** 每步定义都是确切、无歧义的
- **有穷性²** 算法应在执行有穷步后结束
- **有效性³** 每一条运算应足够基本

1: 一条语句遇到两次相同输入，应给出相同的结果。

3: 算法中所有操作都必须可以通过已经实现的基本操作及基本运算，并在有限次内实现。

2: 算法不能陷入无限循环。例如求 π 的精确解，由于是无穷位的数，所以可以称为计算方法，而不能是算法。



算法的例子

输入： 非降序排列的数组A[1...n]和元素x

输出： 如果 $x=A[j]$, $1 \leq j \leq n$, 则输出j, 否则输出-1.

1. $j \leftarrow -1$
2. for $i = 1$ to n
3. if $x = A[j]$
4. $j \leftarrow i$
5. break;
6. end if
7. end for
8. return j

输入
输出
确定性
有穷性
有效性



程序和算法的区别

程序 = 算法 ?



- ✓ 程序是算法用某种程序设计语言的具体实现。
- ✓ 程序不一定满足有穷性，即不一定是算法。



算法性能分析与度量

算法的性能标准：

- **正确性**: 算法应满足具体问题的需求。
- **可使用性**: 或者用户友好性，算法能够很方便地使用。
- **可读性**: 算法应该好读。有利于阅读者对程序的理解。
- **效率**
- **健壮性**: 算法应具有容错处理。输入非法数据时，算法应对其作出反应。
- **简单性**: 算法所采用的数据结构和方法的简单程度。



算法的效率

算法的效率包括：

- **时间代价**：算法执行时间；
- **空间代价**：算法执行过程中**所需的最大存储空间**。

两者都与**问题的规模**有关。



算法效率的度量

- 后期测试：收集此算法的执行时间和实际占用空间的统计资料。
- 事前估计：求出该算法的一个时间界限函数。



算法效率的度量

后期测试：

- 在算法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的具体时间。
- ✓ 一是必须先运行依据算法编制的程序；
- ✓ 二是所得时间的统计量依赖于计算机的硬件、软件等环境因素，有时容易掩盖算法本身的优劣。



算法效率的度量

事前估计：

算法的复杂性度量属于事前估计

- 空间复杂度：算法执行时所占用存储空间
- 时间复杂度：算法执行时所耗费的时间



绝对运行时间来度量时间复杂度?

- 一个编程实现了的算法的绝对运行时间，不仅和算法本身相关，还和其他因素密切相关：机器性能、编程语言、编译器、编程技巧等。
- 在分析一个算法的运行时间时，通常将该算法和其他算法在同一问题、甚至是**不同的问题**上进行比较，因而运行时间只能是相对的，而不是绝对的。



绝对运行时间来度量时间复杂度?

- 希望算法的描述不仅独立于机器，并且可以以任何语言来加以描述，包括自然语言。
- 希望使用度量算法运行时间的准则不依赖于软硬技术的进步。
- 不仅仅关注小规模输入下的，而且还关注在大规模输入下的情形。



时间复杂性度量

- 用程序步来衡量一个程序的执行时间
 - 程序步：语法（义）上有意义的一段指令
-
- 注释\声明语句：程序步数为0
 - 赋值语句\表达式计算：程序步为1
 - 循环控制语句：每次执行的程序步数为1



程序中加入计数全局变量语句

```
float sum ( float a[ ], const int n ) {  
    float s = 0.0;  
    count++;           //count统计执行语句条数  
    for ( int i = 0; i < n; i++ )  
    {  
        count+=2;     //针对for语句  
        s += a[i];  
        count++;      //针对赋值语句  
    }  
    count+=2;          //针对for的最后一次  
    count++;          //针对return语句  
    return s;  
}    执行结束得程序步数 count = 3 * n + 4
```



程序中加入计数全局变量语句

注意：一个语句本身的程序步数可能不等于该语句一次执行所具有的程序步数。

例如：赋值语句 $x = \text{sum}(R, n);$

- 本身的程序步数为1；
- 一次执行对函数 $\text{sum}(R, n)$ 的调用需要的程序步数为 $3*n+4$ ；
- 一次执行的程序步数为 $1+3*n+4 = 3*n+5$



程序中加入计数全局变量语句

- 实际场景中，确定程序的**准确**程序步数十分困难。程序步数本身也不是一个准确的时间概念，不能确切反映运行时间。



算法的渐进分析

渐进的时间复杂度

- 一个算法的“运行时间”通常是**随问题规模的增长而增长**, 它是问题规模(用正整数n表示)的函数。
- 统计算法中的关键操作, 以**关键操作**重复执行的次数作为算法的时间度量。
- 算法中关键操作重复执行的次数是规模n的某个函数 $T(n)$, 称 $T(n)$ 为算法的(**渐近**)时间复杂度。



算法的渐进时间复杂度

- 要精确地计算 $T(n)$ 通常较困难。
- 渐进时间复杂度——算法中基本操作重复执行的次数是问题规模 n 的某个函数 $f(n)$, 算法的时间度量记作:
(大O记号): $T(n) = O(f(n))$
- 随着问题规模 n 的增长, 算法执行时间的增长率和 $f(n)$ 的增长率相同。

例: 一个程序的实际执行时间为

$$T(n) = 2.7n^3 + 3.8n^2 + 5 \cdot 3, \text{ 则 } T(n) = O(n^3)$$



例 矩阵相乘

例：两个方阵相乘 $C_{n \times n} = A_{n \times n} \times B_{n \times n}$ 的算法如下，分析该算法的时间复杂度。

```
for(i=0;i<n;i++)          //n+1 注意：跳出循环那次也算
    for(j=0;j<n;j++) {      //n(n+1) 前面的n为外循环
        c[i][j]=0.;          //n2
        for(k=0;k<n;k++)     //n2(n+1)
            c[i][j]=c[i][j]+a[i][k]*b[k][j];           //n3
    }
```

$$T(n)=(n+1)+n(n+1)+n^2+n^2(n+1)+n^3=2n^3+3n^2+2n+1$$

- 问题的规模 (Size) 用整数 n 表示。
- 时间复杂度是规模 n 的函数，随问题规模增长而增长。



算法的渐进时间复杂度

当问题的规模n趋向无穷大时，时间复杂度T(n)的数量级(阶)称为算法的渐近时间复杂度。

$$T(n)=2n^3+3n^2+2n+1=O(n^3)$$

时间复杂度的解析形式常常难求，或非常复杂。但问题规模较大时，复杂度表示式中实际上只有一些占主导地位的项有意义，其它低次项可忽略不计。所以通常用某些简单函数来近似表示其大致性能。



算法的渐进时间复杂度

- 若语句很少执行，且与规模无关，则可忽略不计。
- 若所有语句都与规模无关，即使有上千条语句，其执行时间也不过是一个较大的常数，时间复杂度也只是 $O(n^0)=O(1)$ 。
- 一般可只考虑与程序规模有关的频度最大的语句，如循环语句的循环体，多重循环的内循环等。

通常用 $O(1)$ 表示常数计算时间。常见的渐进时间复杂度有：

$$\begin{aligned} O(1) &< O(\log_2 n) < O(n) < O(n \log_2 n) < O(n^2) < O(n^3) \\ &< O(2^n) < O(3^n) < O(n!) \end{aligned}$$



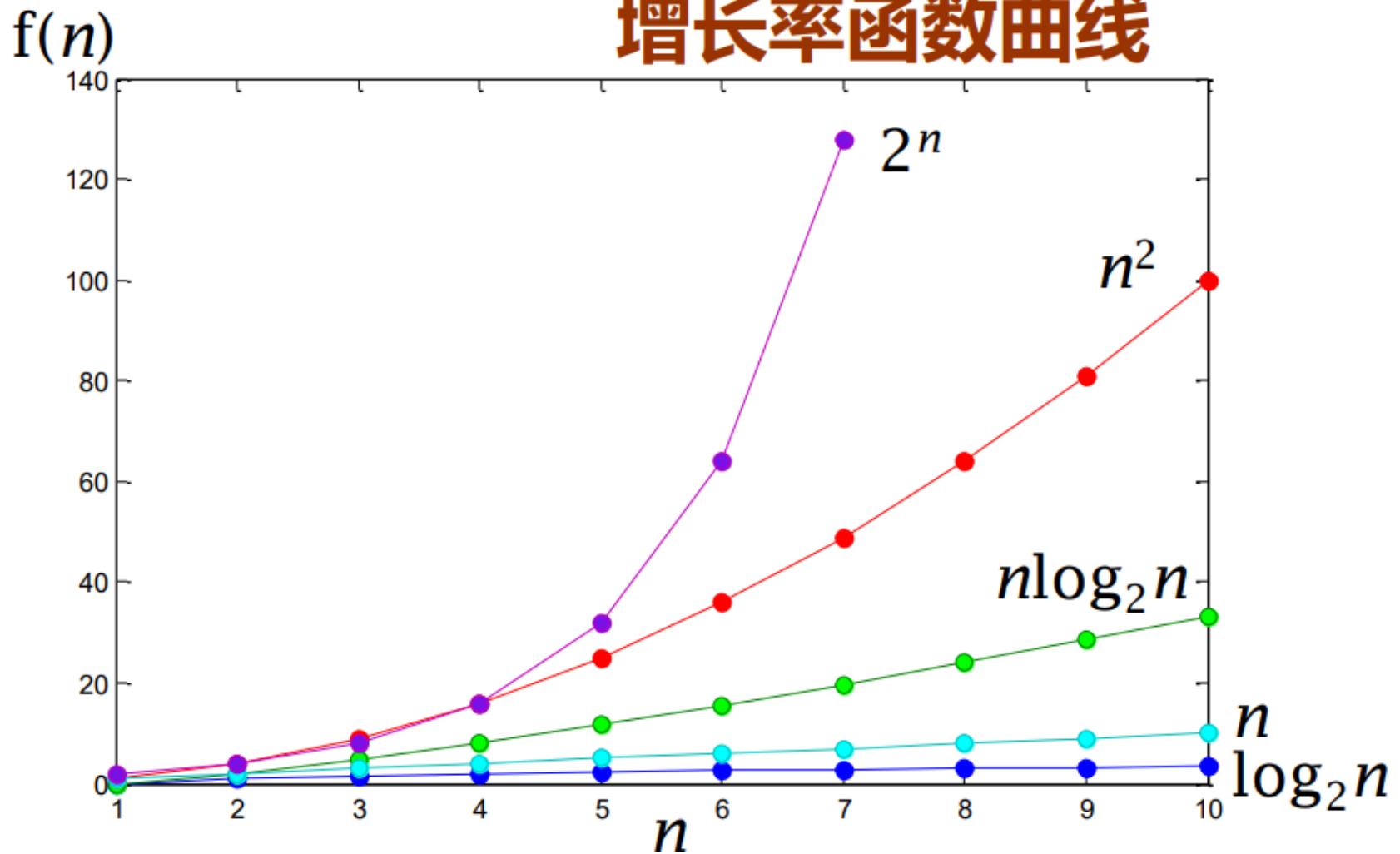
算法复杂度的阶对耗时的影响

假设计算机每秒可以进行 10^6 次运算

Algorithm	1	2	3	4	5
$T(n)$: Time function(ms)	$33n$	$46n \lg n$	$13n^2$	$3.4n^3$	2^n
Input size(n)	Solution time				
10	0.00033 sec.	0.0015 sec.	0.0013 sec.	0.0034 sec.	0.001 sec.
100	0.0033 sec.	0.03 sec.	0.13 sec.	3.4 sec.	4×10^{16} yr.
1,000	0.033 sec.	0.45 sec.	13 sec.	0.94 hr.	
10,000	0.33 sec.	6.1 sec.	22 min.	39 days	
100,000	3.3 sec.	1.3 min.	1.5 days	108 yr.	
Time allowed	Maximum solvable input size (approx.)				
1 second	30,000	2,000	280	67	20
1 minute	1,800,000	82,000	2,200	260	26



增长率函数曲线





$T(n) = O(f(n))$ 的几条规则

- 在进行阶的运算时，常系数、低的阶以及常数项可以忽略。
- 根据O的定义，得到的是在问题规模充分大时，算法复杂度的一个上界。上界的阶越低则评估越有价值。



$T(n) = O(f(n))$ 的几条规则

- 加法规则 针对并列程序段

$$\begin{aligned} T(n, m) &= T_1(n) + T_2(m) \\ &= O(\max(f(n), g(m))) \end{aligned}$$

- 乘法规则 针对嵌套程序段

$$\begin{aligned} T(n, m) &= T_1(n) * T_2(m) \\ &= O(f(n)*g(m)) \end{aligned}$$

- 乘法规则中的常数无关项 $O(C) \rightarrow O(1)$

$$T(n) = O(c * f(n)) = O(f(n))$$

- $O(1)$ 表示常数计算时间



例1：两个并列循环

```
void example (float x[ ][ ], int m, int n, int k) {  
    float sum [ ];  
    for ( int i = 0; i < m; i++ ) {      //x[ ][ ]中各行  
        sum[i] = 0.0;                      //数据累加  
        for ( int j=0; j<n; j++ )  
            sum[i] += x[i][j];  
    }  
    for ( i = 0; i < m; i++ ) //打印各行数据和  
        cout << "Line" << i <<  
            " :" <<sum [i] << endl;  
} 渐进时间复杂度为 O(max (m*n, m))
```



几个例子

【例1】分析下述程序段的时间复杂度。

```
{ ++x;
```

```
s=0; }
```

选取“`++x;`”为基本操作，语句频度为1，
则时间复杂度为 $O(1)$ ，即常量阶。



几个例子

【例2】分析下述程序段的时间复杂度。

```
(j=1; j<=10000; ++j) {  
    ++x;    s+=x;  
}
```

选取“`++x;`”为基本操作，语句频度为
10000，则时间复杂度为 $O(1)$ ，即常量阶。



几个例子

【例3】分析下述程序段的时间复杂度。

S=0 ;

for (j=1 ; j<=n ; j*=2)

++x ;

选取“`++x;`”为基本操作，语句频度为 $\log_2 n$ ，
则时间复杂度为 $O(\log_2 n)$ ，即**对数阶**。



几个例子

【例4】分析下述程序段的时间复杂度。

```
for (i=1; i<=2*n; ++i) {  
    ++x; s+=x;  
}
```

选取“`++x;`”为基本操作，语句频度为 $2 \times n$ ，
则时间复杂度为 $O(n)$ ，即线性阶。



几个例子

【例5】分析下述程序段的时间复杂度。

```
for (j=1; j<=n; ++j) {  
    for (k=1; k<=n/4; ++k) {  
        ++x; s+=x;  
    }  
}
```

选取“`++x;`”为基本操作，语句频度为 $n \times n/4$ ，
则时间复杂度为 $O(n^2)$ ，即平方阶。



几个例子

【例6】分析下述程序段的时间复杂度。

```
for (j=1; j<n; j*=2) {  
    for (k=1; k<=n; ++k) {  
        ++x; s+=x;  
    }  
}
```

时间复杂度为 $O(n \log_2 n)$ ，即线性对数阶。



最坏、最好和平均情况

(1) 大 **O** 表示法 $T(n)=O(f(n))$

即给出了时间复杂度的上界，即**最坏情况**。

(2) 大 **Ω** 表示法 $T(n)=\Omega(f(g))$

给出了时间复杂度的下界，即**最好情况**。

(3) 大 **Θ** 表示法 $T(n)=\Theta(f(N))$

即 $T(n)=O(f(n))$ 与 $T(n)=\Omega(f(n))$ 都成立，给出了时间复杂度的上界和下界。

经常将渐进时间复杂度大 **O** 表示法 $T(n) = O(f(n))$ 简称为**时间复杂度**。



渐进的空间复杂度

$S(n)$ 为算法的(渐近)空间复杂度

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

- 当实例特性n充分大时，需要的存储空间将如何随之变化。
- 这里所说得是为解决问题所需要的辅助存储空间。



算法复杂度分析的意义

- 已知待求解问题的多种算法时，挑选复杂度尽可能低的算法进行应用。
- 给定待求解的问题，设计复杂度尽可能低的算法。
- 设计出算法后，不要急于实现，而是先进行复杂度分析后；若该算法确实可行，才有实现的价值与必要。



本章小节

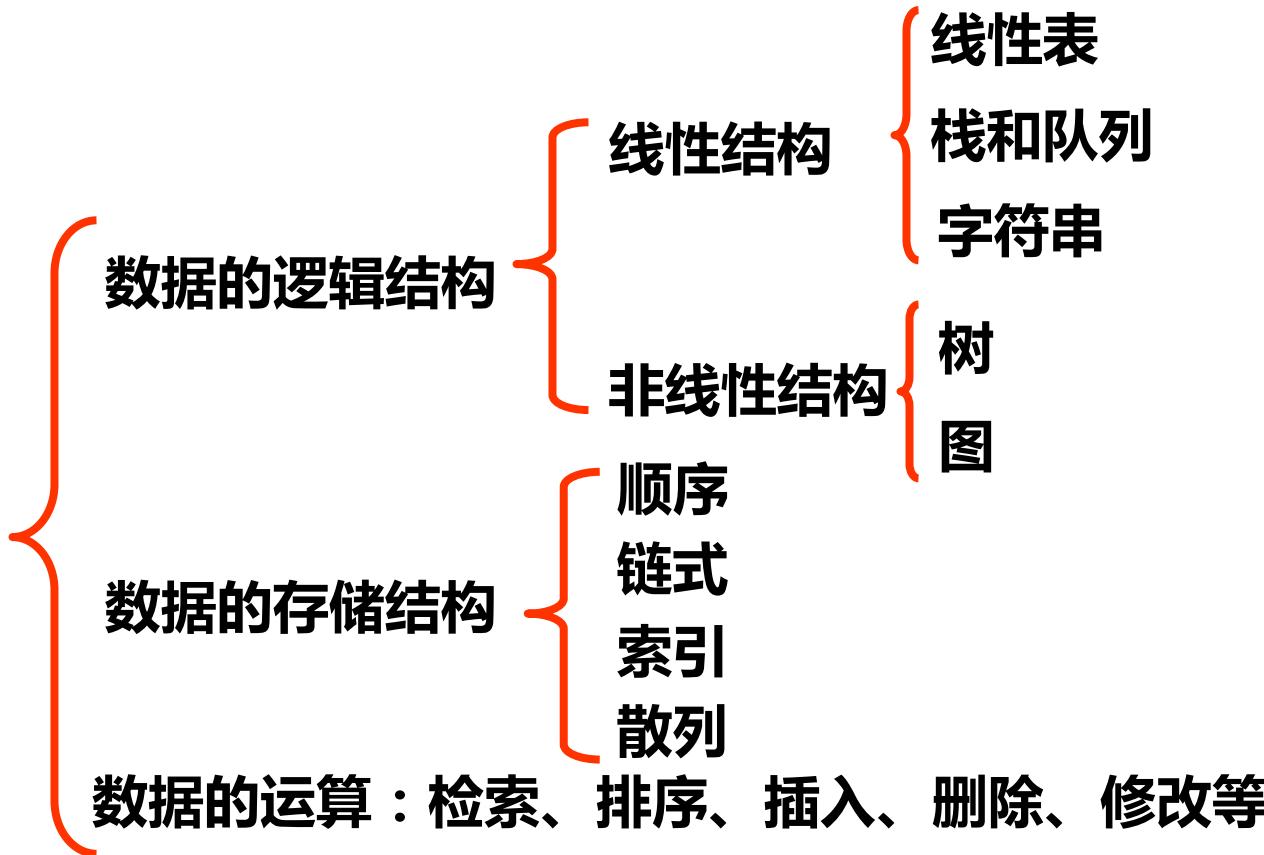
- 数据结构的基本含义 $DS = \{ D, R \}$
- 逻辑结构、存储结构

- 数据、数据元素、数据项
- 数据类型、ADT（抽象数据类型）

- 程序=数据结构+算法
- 算法的定义、算法的基本特征
- 算法性能（时空渐进复杂性）分析——大O表示法

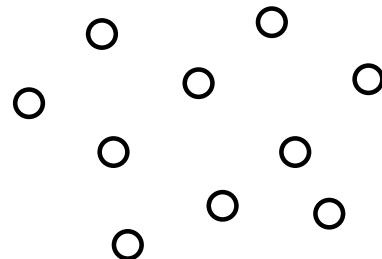


数据结构关注的三个方面





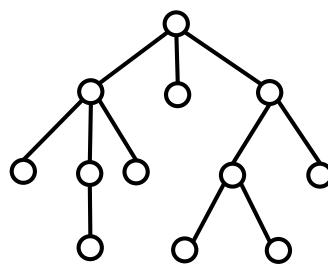
数据的逻辑结构



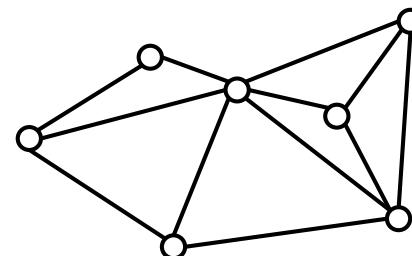
(a) 集合结构



(b) 线性结构



(c) 树型结构



(d) 图型结构