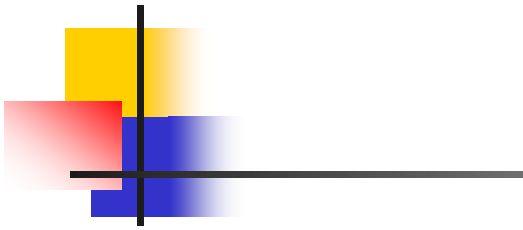


数据结构
Data Structure

姚宣霞
Email:kathy.yao@163.com

北京科技大学计算机科学与技术系

Department of Computer Science & Technology
University of Science & Technology Beijing
2018春季学期

总学时：64学时

讲授：48学时

实验： 16学时，13-16周

最终成绩的构成：

平时成绩40%（实验25%，考勤5%，作业10%）

考试成绩60%

课程邮箱：dsustb@163.com， pwd: ustb123456

教材与参考书

[1] 齐悦，夏克俭，姚琳．数据结构、算法与应用．北京：清华大学出版社，2015（教材）

[2] 夏克俭，王绍斌．数据结构．北京：国防工业出版社，2010

[3] Donald E. Knuth．The Art of Computer Programming. Vol. 1-3, Addison-Wesley （非常经典，已有译著，作者系图灵奖获得者）

[4] 严蔚敏，吴伟民．数据结构（C语言版）．北京：清华大学出版社，1997

[5] Mark Allen Weiss． Data Structure and Algorithm Analysis in C．2nd ed. 陈越，改编．北京：人民邮电出版社， 2005

[6] Clifford A. Shaffer．张铭，刘晓丹译． 数据结构与算法分析．北京：电子工业出版社，1998

[7] 冯玉琳，仲萃豪，陈友君．程序设计方法学．北京：北京科学技术出版社，1989

讲授的主要内容

第1章　绪论

第2章　线性表

第3章　栈和队列

第4章　串（自学）

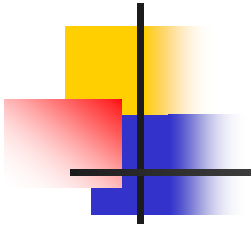
第5章　数组和广义表

第6章　树

第7章　图

第8章　查找

第9章　排序



第1章 绪论

1.1 认识数据结构与算法

1.2 抽象数据类型的表示与实现

1.3 算法分析

1.4 选择数据结构与算法时的若干考虑

1.5 学习数据结构课程要达到的目标

1.6 小结

1.1 认识数据结构与算法

程序（Program） + 文档（Document）= 软件（Software）

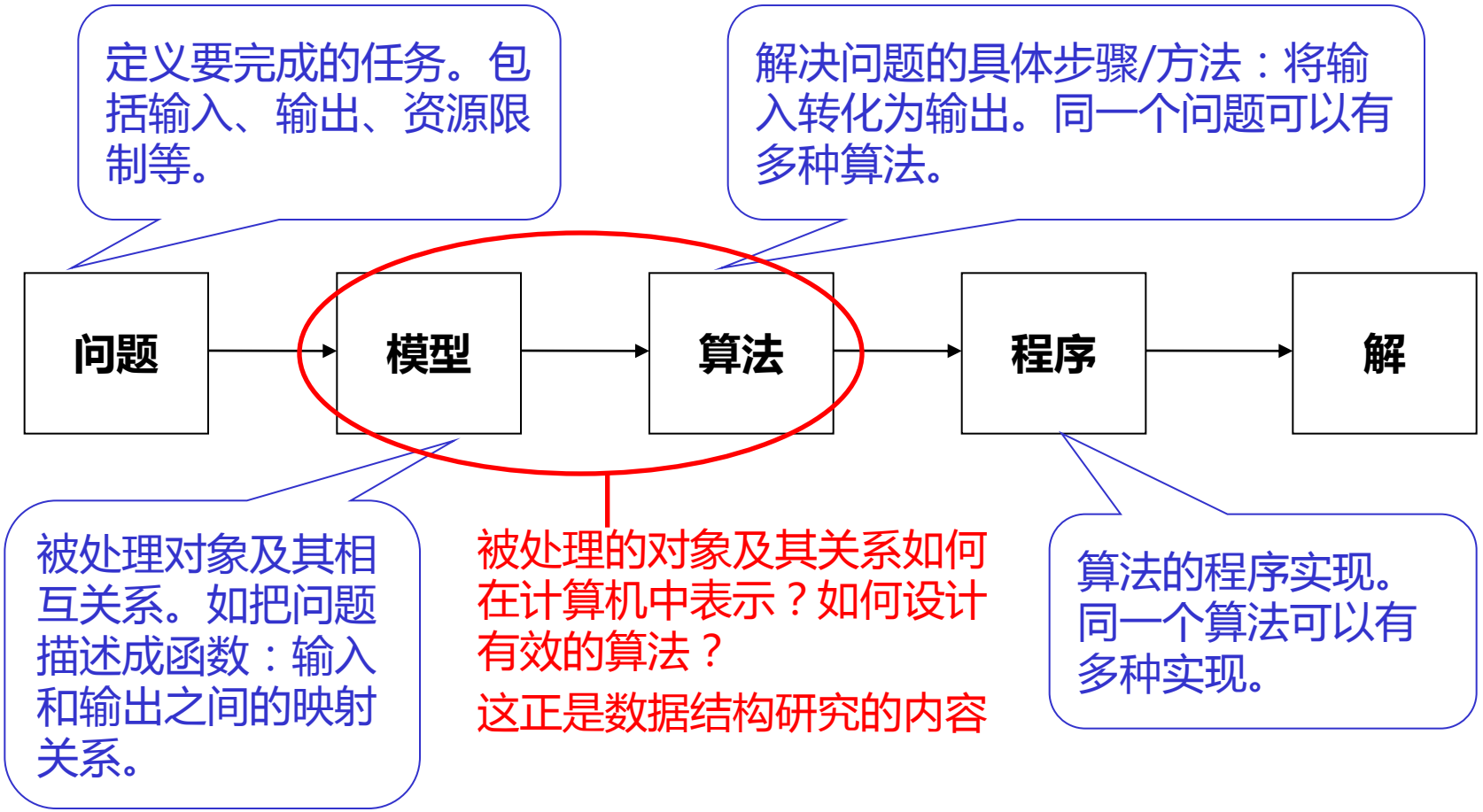
算法（Algorithm） + 数据结构（Data Structure）= 程序

“数据结构与算法”是计算机科学研究中的基础课题。其理论基础是离散数学中的图论、集合论和关系理论等，**实践基础是程序设计**。

数据结构课程的内容来源于图论、操作系统、编译系统、编码理论以及检索与分类技术等相关领域。

1.1 认识数据结构与算法

1. 问题求解的过程



1.1 认识数据结构与算法

2. 什么是数据结构（Data Structure, DS）？

数据结构主要研究非数值型程序设计中数据元素间关系的表示及其处理的算法。

数据结构包括3个方面的内容：

- 数据的逻辑结构
- 数据的存储结构
- 数据的操作

C语言程序是如何构成的？设计程序时主要考虑什么？

狭义地讲，数据结构指的是

数据元素及数据元素之间的相互关系

组织数据的方式

数据的表示

数据结构同时涉及对数据的操作，这些操作通过算法来实现。

从广义上讲，数据结构描述现实世界实体的数据模型及其上的操作在计算机中的表示和实现。

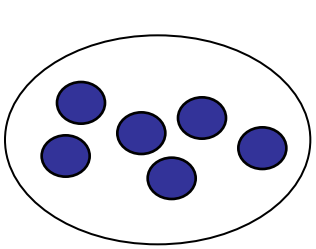
什么是数据结构?

（1）数据的逻辑结构（Logical Structure）

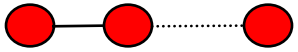
从用户的观点看，数据的组织形式，是独立于计算机存储器的（与机器无关）。

4种基本的逻辑结构：

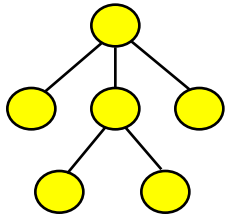
- ✓ 集合：数据元素除了“属于同一集合”的关系外，没有其他关系。
- ✓ 线性结构：数据元素之间存在一对一的关系。如：线性表，栈，队列
- ✓ 层次结构：数据元素之间存在一对多的关系。如：树
- ✓ 网状结构：数据元素之间存在若干个多对多关系。如：图



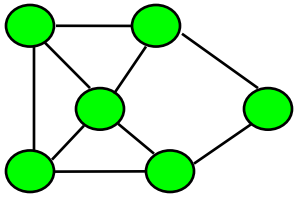
集合



线性结构



层次结构



网状结构

什么是数据结构?

（2）数据的物理结构（存储结构）（Physical Structure）

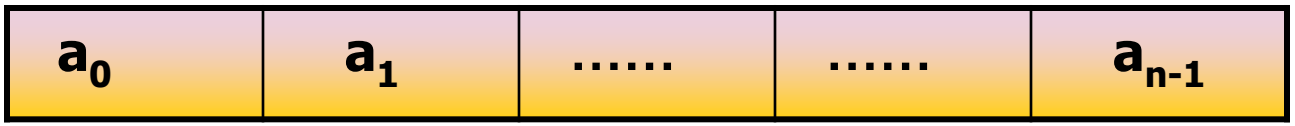
数据在存储器中的存放方式，包括数据元素的表示和关系的表示。

4种基本的存储结构：**顺序存储、链式存储、索引存储、散列存储**

1）顺序存储（Sequential Storage）

将数据结构中各元素按照其逻辑顺序存放于一片**连续**的存储空间中。

如C语言的一维数组：



有何优缺点？

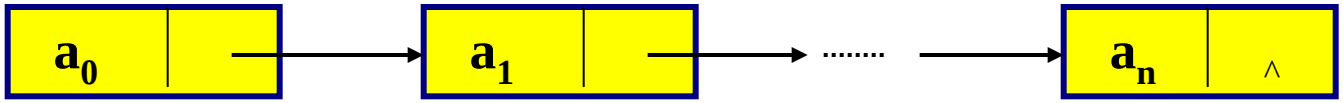
优点：随机访问快（可以直接计算数据的地址）

缺点：插入、删除效率低，不利于动态增长

数据的物理结构

2）链式存储（Linked Storage）

数据结构中各元素可以存放到存储器的不同地方，数据元素之间以指针（地址）链接。



有何优缺点？

缺点：随机访问慢

优点：便于插入、删除和动态增长

数据的物理结构

3）索引存储（Indexed Storage）

在存储数据的同时，建立一个附加的索引表，即

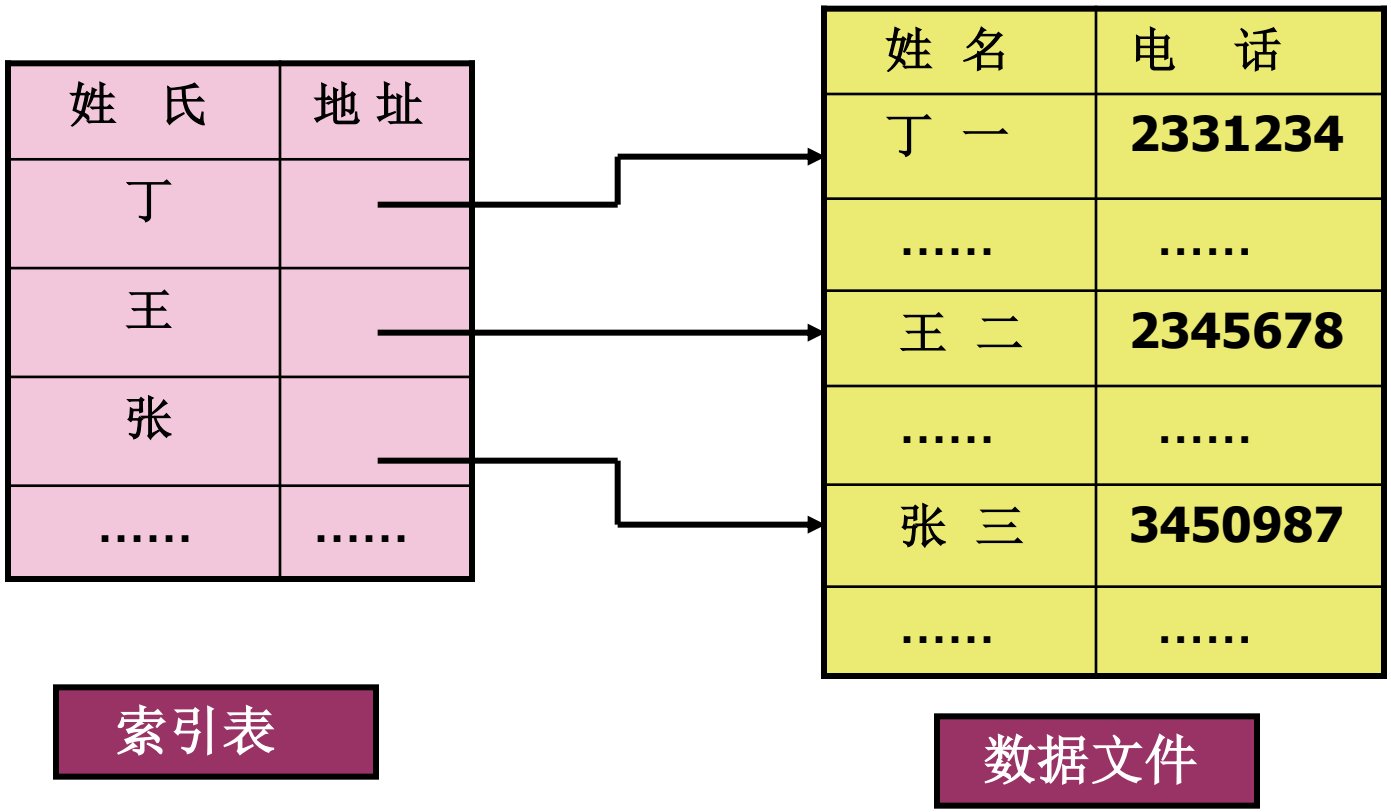
索引存储结构 = 数据文件 + 索引表

索引表存放各数据元素的地址

数据的物理结构

例1.1 电话号码查询问题。

在存储用户数据文件的同时，建立一张姓氏索引表。这样，查找一个电话就可以先查索引表，再查找相应的数据文件，加快了查询速度。



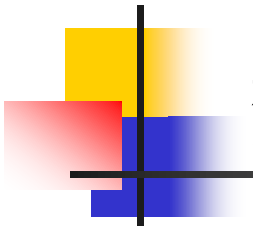
数据的物理结构

索引表的组织方式:

- ✓ 顺序存储：索引表占用连续空间
- ✓ 链式存储
- ✓ 索引存储：多级索引（多重索引）

例如，二级索引：将一个大文件的所有索引表（二级索引)的地址放在另一个索引表（一级索引)中

此外，还有三级索引等。



数据的物理结构

4）散列存储（Hash Structure）

根据数据元素的特殊字段（称为关键字key），计算数据元素的存放地址，然后数据元素按地址存放，所得到的存储结构为散列存储结构(或**Hash结构**)。

什么是数据结构?

说明：

同一逻辑结构可映射为不同的物理结构。

例如：线性队列可映射为数组或链表。

多维数组的2种不同的存储方式，如二维数组：

- ✓ 行主次序（row major）：按行优先，C语言如此
- ✓ 列主次序（column major）：按列优先

什么是数据结构?

（3）数据的操作

基本的数据操作：增删改查

- ✓ **查找**：在数据结构中寻找满足某条件的数据元素的位置或值。
- ✓ **插入**：在数据结构中添加新的数据元素。
- ✓ **删除**：删去数据结构中指定的数据元素。
- ✓ **更新**：改变数据结构中某个数据元素一个或多个数据项的值。
- ✓ **排序**：重新安排数据元素之间的逻辑顺序，使之按（某个数据项的）值由小（大）到大（小）的次序排列。

1.1 认识数据结构与算法

3. 什么是算法（Algorithm）？

算法是一个有穷规则（或语句、指令）的有序集合。它确定了解决某一问题的一个操作序列。对于问题的初始输入，通过算法有限步的运行，产生一个或多个输出。

简单地说，**算法是指解决问题的、由一系列具体步骤组成的方法或过程。**

- ✓ **对特定问题求解步骤的一种描述**
- ✓ **是一有限长的操作序列**

什么是算法?

例1.2 求两个正整数m、n的最大公因子的算法如下：

① 输入m, n；

② m/n（整除），余数→r (0≤r≤n)；

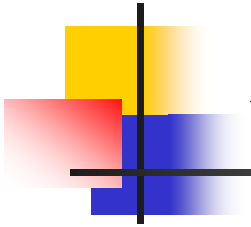
③ 若r=0，则当前n=结果，输出n，算法停止；否则，转④；

④ n→m, r→n； 转②。

如初始输入 m=10, n=4, 则m, n, r 在算法中的变化如下：

m	n	r
10	4	2
4	2	0(停止)

即10和4 的最大公因子为2。



什么是算法?

算法的性质：

1) **有穷性** —— 算法执行的步骤是有限的，就是说是可终止的；
2) **确定性** —— 每个计算步骤无二义性；
3) **可行性** —— 每个计算步骤能够在有限的时间内完成；
4) **输入** —— 算法有一个或多个外部输入；
5) **输出** —— 算法有一个或多个输出。

什么是算法?

算法与程序的联系与区别：

共同点：二者都是为完成某个任务，或解决某个问题而编写的规则（或语句）的有序集合。

区别：

1）算法与程序设计语言无关。

2）算法必须是有穷尽的，但程序可能是无穷尽的。

3）算法可忽略一些语法细节问题，**重点放在解决问题的思路上**，但程序必须严格遵循相应语言工具的语法。算法转换成程序后才能在计算机上运行。

不严格地说，程序是算法的具体实现，故有时会混用算法和程序。

1.2 抽象数据类型的表示与实现

1. 数据类型（Data Type）

值的集合和定义在这个值集上的一组操作的总称

如C语言中的整型变量(int)，

其值集为某个区间上的整数，

定义在其上的操作为+, -, x, /等

1.2 抽象数据类型的表示与实现

（1） 原子数据类型

- ✓ 是不可分解的数据类型
- ✓ 如C语言中的整型(int)，实型(float)，字符型(char)，指针类型(*)和空类型(void)等

（2） 结构数据类型

- ✓ 由若干成分（原子类型或结构类型）按某种结构组成的数据类型
- ✓ 结构数据类型可以看作是一种数据结构和定义在其上的一组操作组成的整体
- ✓ 如数组，由若干元素组成，每个元素可以是整数，也可以是数组（如int a[10], b[5][6]）

1.2 抽象数据类型的表示与实现

2. 抽象数据类型（Abstract Data Type, ADT）

一个数据结构以及定义在其上的一组操作

ADT的定义仅依赖于其逻辑特性，独立于具体的计算机实现。

- ✓ 由用户定义，用来表示应用问题的数据模型
- ✓ 由固有的数据类型组成，并包括一组相关的操作
- ✓ 信息隐蔽和数据封装，使用与实现相分离

1.2 抽象数据类型的表示与实现

ADT的表示：

ADT =（D，R，P）

其中，D是数据元素的有限集；R是D上关系的有限集，（D，R）构成了一个数据结构；P是对该数据结构的基本操作集。

用以下格式定义抽象数据类型：

ADT 抽象数据类型名{

数据元素集：< 数据元素集的定义 >

数据关系集：< 数据关系集的定义 >

基本操作集：< 各种基本操作的定义 >

} ADT 抽象数据类型名；

1.2 抽象数据类型的表示与实现

基本操作的定义格式是：

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 >

“初始条件”描述了操作执行之前的数据结构及参数应满足的条件；

“操作结果”说明了操作正常完成之后，数据结构的变化和应返回的结果。

1.2 抽象数据类型的表示与实现

ADT定义例：

ADT Triplet {

数据元素集：D = {e1,e2,e3 | e1,e2,e3∈ElemSet}

数据关系集：R = {<e1,e2>, <e2,e3>}

基本操作集：P

Max(T, &e)

初始条件：三元组T已存在。

操作结果：用e返回T的3个元素中的最大值。

Min(T, &e)

初始条件：三元组T已存在。

操作结果：用e返回T的3个元素中的最小值。

} ADT Triplet;

1.2 抽象数据类型的表示与实现

ADT的实现：

可以通过固有数据类型(如C语言中已有的数据类型)来实现

本课程采用的数据结构和算法描述：

- ✓ 以介于伪码和C语言之间的**类C语言**作为描述工具，不拘泥于C语言的细节，又容易转换为C或C++程序
- ✓ 有时采用伪码描述一些只含抽象操作的抽象算法

C语言程序是如何构成的？设计程序时主要考虑什么？

- ✓ 说明性语句：int a[10]； typedef struct { }
- ✓ 执行类语句：赋值、循环、输入/出等
- ✓ 不同的数据结构适用于不同算法，用C实现算法时注意，避免二义性，goto语句、预处理和函数返回值的隐含说明等问题

1.3 算法分析

算法分析是指算法在正确的情况下，对其优劣的分析。

1. 算法评价

一个好算法的标准：

（1）算法结构良好，容易理解、编码、移植和调试等等。

（2）高时空效率。

- ✓ 时间（执行时间，开发时间）短
- ✓ 空间少

2个度量标准：

- ✓ **时间复杂度（重点）**
- ✓ 空间复杂度

1.3 算法分析

2. 时间复杂度的估算

（1）语句的频度(Frequency Count)

可执行语句在算法（或程序）中重复执行的次数

若某语句执行一次的时间为t，执行次数为f，则该语句所耗时间为 t*f。

1.3 算法分析

例1.3 求两个n 阶方阵乘积

$$C_{n\cdot n}=A_{n\cdot n}\times B_{n\cdot n}=\begin{pmatrix} c[0][0] & c[0][1] & \cdots\cdots & c[0][j] & \cdots\cdots & c[0][n-1] \\ c[1][0] & c[1][1] & \cdots\cdots & c[1][j] & \cdots\cdots & c[1][n-1] \\ & & & & & \cdots\cdots \\ & & & \cdots\cdots & & \\ c[i][0] & c[i][1] & \cdots\cdots & c[i][j] & \cdots\cdots & c[i][n-1] \\ & & & & & \cdots\cdots \\ c[n-1][0] & c[n-1][1] & \cdots\cdots & c[n-1][j] & \cdots\cdots & c[n-1][n-1] \end{pmatrix}$$

其中：

$$c[i][j] = \sum_{k=0}^{n-1} a[i][k]\times b[k][j]$$

1.3 算法分析

求两个n 阶方阵乘积的算法：

```
void  matrixm(float a[n][n], float b[n][n], float c[n][n])
{
    int i, j, k;                                          语句频度
    for (i = 0; i < n; i++) {                             n+1
        for (j = 0; j < n; j++) {                         n(n+1)
            c[i][j] = 0;                                  n^2
            for (k = 0; k < n; k++)                       n^2(n+1)
                c[i][j] = c[i][j] + a[i][k] * b[k][j];    n^3

        }
    }
}
```

1.3 算法分析

（2）算法的时间复杂度(Time Complexity)

算法中可执行语句的频度之和，记为T(n)。

T(n)是算法所需时间的一种估计，其中n为问题的规模（或大小、体积）。

例 1.3中，问题的规模n为矩阵的阶，算法的时间复杂度为：

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

度量标准：以量级来估算当问题规模增大时算法执行时间的增加速度（增长率），以**O()**表示。

T(n)的量级为n^3，记为：$T(n) = O(n^3)$

1.3 算法分析

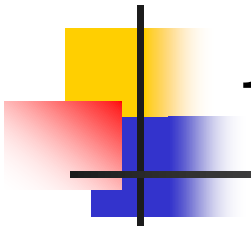
例 1.4 在数组（a[0], a[1], ..., a[n-1]）中查找首个与给定值k相等的元素序号。

```
int search(datatype a[n], datatype k) //在a[n] 中查找k的算法
{
    int i = 0;
    while (i < n && a[i] != k)   i++;
    if (i < n) return i;
    else return (-1);
}
```

以**平均查找次数**作为算法的T(n)。设查找每个元素的概率**P_i(0≤i≤n-1)**相等，即**P_i=1/n**，查找元素k时，k与a[i]的比较次数（即执行while循环语句的次数）**C_i=i+1**，则查找次数的平均值（或期望值）：

$$\mathbf{T(n)} = \sum_{i=0}^{n-1} P_i C_i = \frac{1}{n}\sum_{i=0}^{n-1}(i+1) = \frac{n+1}{2}$$

故T(n) = **O(n)**



1.3 算法分析

分析时间复杂度时可能要考虑的几种情况：

1）平均情况

一般作为考虑的重点。当然要看输入数据的分布概率。

2）最差情况

有时会关注。例如，实时系统。

3）最佳情况

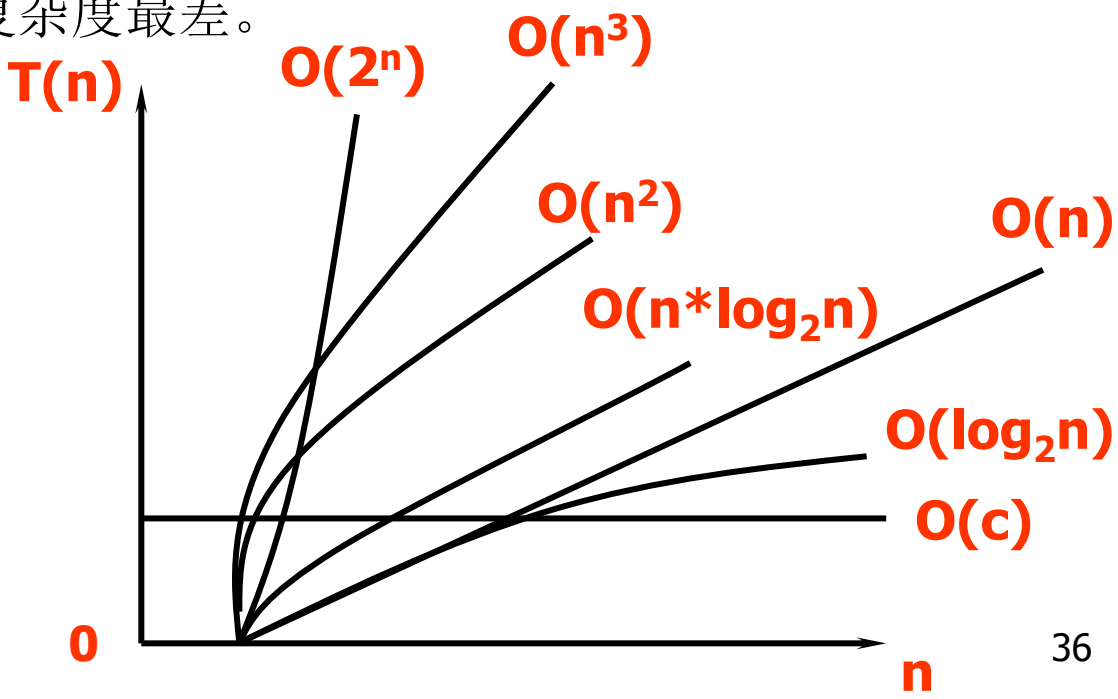
一般不关注。只有发生概率较大时才考虑。

1.3 算法分析

T(n)常见的量级：

- O(c) ——常数级，与问题规模无关，是最理想的量级。有时记作O(1)。
- O(n) —— 线性级；
- $O(n^2)$, $O(n^3)$ —— 平方、立方级；
- $O(\log_2 n)$, $O(n*\log_2 n)$ ——对数级、线性对数级；
- $O(2^n)$ —— 指数级，时间复杂度最差。

常见的T(n)随n变化的增长率：



1.3 算法分析

对较为复杂的算法，可分段分析其时间复杂度。如某算法可分为两部分，其时间复杂度分别为：

$$T_1(n)=O(f(n)),\ T_2(n)=O(g(n))$$

若两部分问题的规模一致，则总的$T(n)=T_1(n)+T_2(n)=O(max(f(n),g(n))$

但若$T_1(m)=O(f(m))$, $T_2(n)=O(g(n))$，两部分的规模不一致，则：

$$T(m,n)= T_1(m)+ T_2(n)=O(f(m)+g(n))。$$

空间复杂度的定义：

设算法对应问题的规模为n，执行算法所占存储空间的量级为D(n)，则D(n)为算法的空间复杂度(Space Complexity)。

1.4 选择数据结构与算法时的若干考虑

1. 选择数据结构时的考虑

（1）与处理方式密切相关。

例如，设有一组数据，选择顺序结构还是链式结构好？

若建立1次，频繁查找、修改，很少增加/删除：顺序结构较好

若经常要增加/删除：链式结构较好

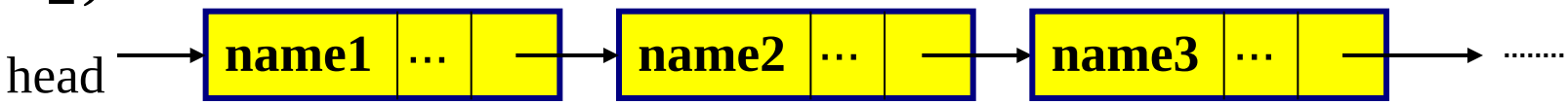
（2）数据存储在内存/磁盘要有不同的考虑

原因：磁盘读写以磁盘块为单位，且读写速度比在内存慢得多。

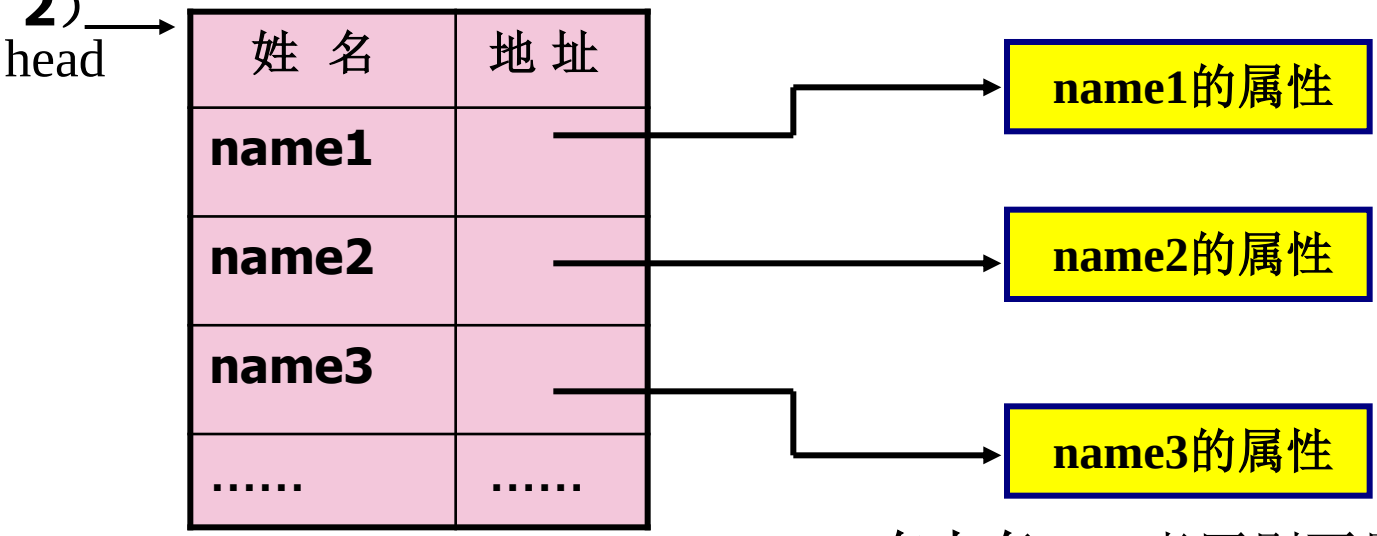
1.4 选择数据结构与算法时的若干考虑

例1.4 有一组学生数据，以非顺序方式存放，且无特定规律。要通过姓名查找某学生的其他属性。采用下列2种存储方式，那种效率高？

1）



2）



在内存：二者区别不显著。

在磁盘：2）要好得多。

1.4 选择数据结构与算法时的若干考虑

2. 选择算法时的考虑

（1）程序的使用次数

若使用次数很少，应选用容易正确实现的算法，以减少开发时间

（2）问题的规模（输入数据的规模）

规模很大：**一般不能用$O(2^n)$的算法**

规模较小：有时要考虑常数因子

例如，设$n < 20$，则$5n^3 < 100n^2$

（3）对于数值计算问题，除了时间效率外，还要考虑精确性与稳定性。

数值计算一般不是数据结构课程所关注的。

1.4 选择数据结构与算法时的若干考虑

（4）时空权衡（space/time trade-off）

一般原则：牺牲空间可换取时间，反之亦然。

例如：

1）信息压缩。空间减少，时间代价增大。

2）查找表（lookup table）。存放常用函数的值，以节省计算时间。在实时系统中常用。

例外：在磁盘上的存储开销越小，可能程序运行得越快。

为什么？

因为读写磁盘的时间开销远大于计算的时间开销，尤其是频繁读写时。

（5）算法的不同程序实现会导致效率的差异

同样的算法，通过优化代码可以改善效率。

1.5 学习数据结构课程要达到的目标

1. 为什么要学习数据结构？

- ✓ 在分析或开发计算机软件时需要数据结构（和算法）的知识
- ✓ 学习数据结构（和算法）有助于提高程序设计水平
- ✓ 针对具体问题，选择/设计合适的数据结构与算法
- ✓ 各个领域的思想、方法有很多是相通的

1.5 学习数据结构课程要达到的目标

2. 学习数据结构要达到的目标

1）理解常用的数据结构和算法的基本思路、思考方法及其适用场合，能对给定算法进行复杂度分析

- ✓ 数据结构的3个方面：逻辑结构、物理结构和操作
- ✓ 算法的复杂度分析
- ✓ 理解（**注意：不是简单的记忆**）每种数据结构和算法的原理：是什么？为什么？有何特点？如何实现？适合于解决什么问题？
- ✓ 不要简单记忆公式、定理/性质等，要会推导
- ✓ 对数据结构课程内容的**整体把握**

学习数据结构要达到的目标

2）深入体会经典算法的思想和分析解决问题的方法，养成从“问题解决者的角度”学习方法

- ✓ 从问题解决者（数据结构和算法设计者）的角度去思考、学习
- ✓ 注重系统，训练思维，领悟思想，理解方法：目的是应用以解决实际问题
- ✓ 系统观和方法学

学习数据结构要达到的目标

3）针对实际问题能选择应用合适的数据结构和算法，提高程序设计能力

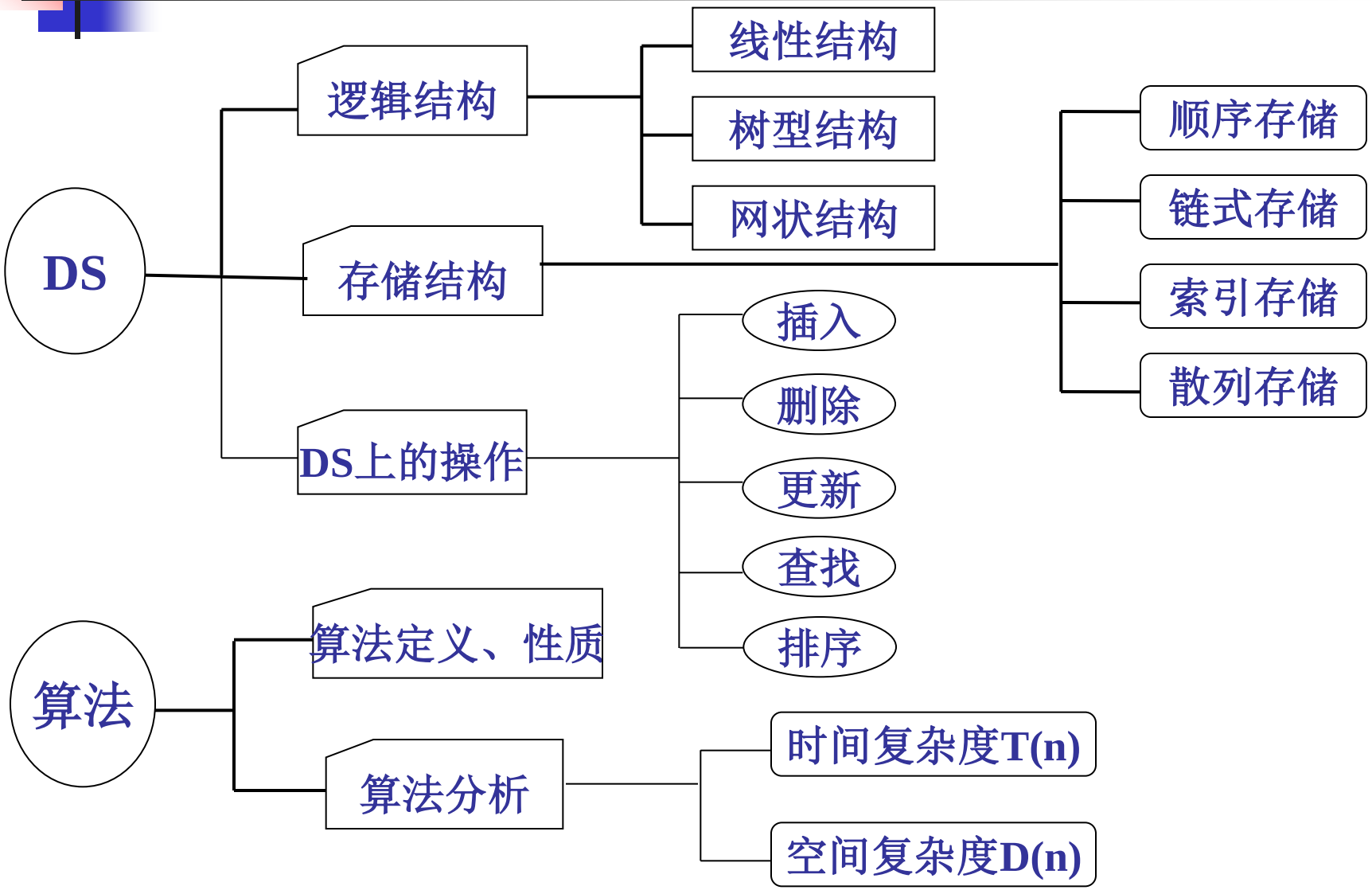
- ✓ 不存在绝对意义上最好的数据结构和算法
- ✓ 考虑问题的特点：资源限制、时间要求、常进行的操作、应用场景、用户特点等
- ✓ 权衡的思想

学习数据结构要达到的目标

4）针对新问题设计出有效的数据结构和算法

- ✓ 将复杂问题抽象为计算模型
- ✓ 分而治之（Divide and Conquer）
- ✓ 借鉴前人的思想、方法
- ✓ 需要的理论、技术涉及程序设计方法学、数学基础（包括离散数学）、问题分析的方法等

1.6 小结


DS
逻辑结构
线性结构
树型结构
网状结构
存储结构
顺序存储
链式存储
索引存储
散列存储
DS上的操作
插入
删除
更新
查找
排序
算法
算法定义、性质
算法分析
时间复杂度T(n)
空间复杂度D(n)