§1. 绪论

1.1. 什么是数据结构

★ 数据的表现形式

- 数值数据：数字(用公式、方程等描述)
  用数学语言描述的数学模型
- 非数值数据：文字、图像、视频、声音(无法用公式、方程等描述)
  用数据结构描述的数学模型

★ 数据结构的引入：用来描述非数值计算问题的数学模型称为数据结构

★ 计算机程序的组成

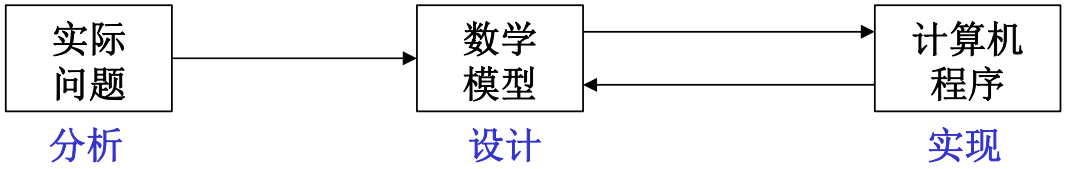
- 数据的描述：在程序中表示信息的方法（静态）（数据结构）
  在程序中要指定的数据的类型及数据的组织形式
- 操作的描述：在程序中处理信息的方法（动态）（算法）
  对动作的描述(操作步骤)

- 数据的描述+操作的描述=程序
- 数据结构　+算法　　=程序
- 数据结构+算法+程序设计方法+语言工具和环境=程序

§1. 绪论

1.1. 什么是数据结构

★ 程序设计的基本过程



模型的定义：若m能回答A的所有问题，则称m为A的模型

分析：实际问题->数学模型的过程(提取操作对象，找出对象间关系并用模型描述)

设计：解此数学模型的一个具体算法

实现：用某种具体的计算机程序设计语言来实现该算法

★ 数据结构的作用

数据结构研究的是数据元素之间抽象化的相互关系以及这种关系在计算机中的存储表示，对每种结构定义各自的运算，设计出相应的算法，并用某种具体的程序设计语言实现该算法

★ 数据结构课程的地位(P.4 图1.4)

介于数学、硬件、软件之间的核心专业基础课

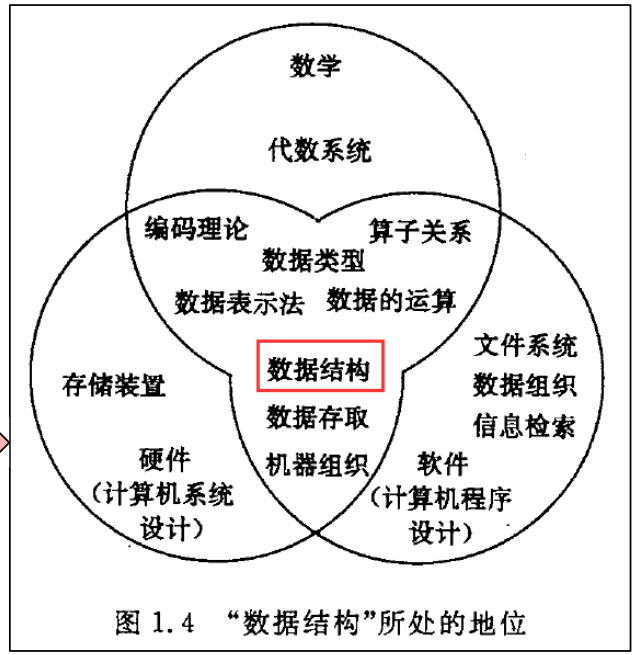


图 1.4 “数据结构”所处的地位

§1. 绪论

1.2. 基本概念和术语

1.2.1. 基本术语

★ 数据

广义定义：对客观事物的符号表示，即人们利用文字符号、数字符号以及其它规定的符号对现实世界事物及其活动所做的描述

计算机领域：所有能输入到计算机中并能被计算机程序所处理的各种符号的总称

- 数字/字符/声音/图形/图像 => 0/1
- 处理：输入、存储、加工、输出等

★ 数据元素与数据项

数据元素：数据的基本单位，在计算机程序中常常作为一个整体进行考虑和处理，由若干数据项组成

（例：struct student 结构体）

数据项：有意义的数据的不可分割的最小单位，一个数据元素通常包含若干数据项

（例：结构体中的成员，若char name[9]等）

- 两者之间是整体和局部的关系
- 最小单位是指有意义的最小单位，不是计算机表示数据的单位(bit/byte等)

★ 数据对象

性质相同的数据元素的集合，是数据的一个子集

- 对象内的元素可视不同情况而不同　　（例1：字母对象、大写字母对象）
  （例2：同济选课制度下每门课程的同学）

★ 数据处理：指对数据进行查找、插入、删除、合并、统计、简单计算、输入、输出等的操作过程，又称信息管理

§1. 绪论

1.2. 基本概念和术语

1.2.2. 数据结构

含义：指相互之间存在一种或多种特定关系的数据元素的集合

结构：数据元素间不是相互独立的，而是存在着某种特定关系，数据元素之间的关系，称为"结构"

- 结构 = 实体 + 关系
  元素 + 元素间特定关系

数据结构的另一种定义： 按照逻辑关系组织起来的一批数据，按一定的存储方法存储在计算机中，并在这些数据上定义了一组运算的集合

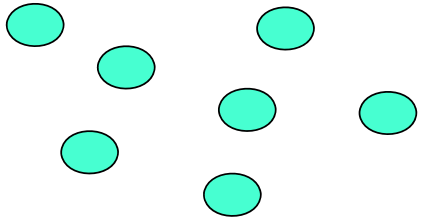
§1. 绪论

1.2. 基本概念和术语

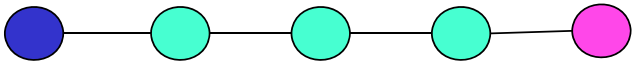
1.2.2. 数据结构

数据结构的基本结构及图示表示：

★ 集合：元素间仅存在有"同属于一个集合的关系"

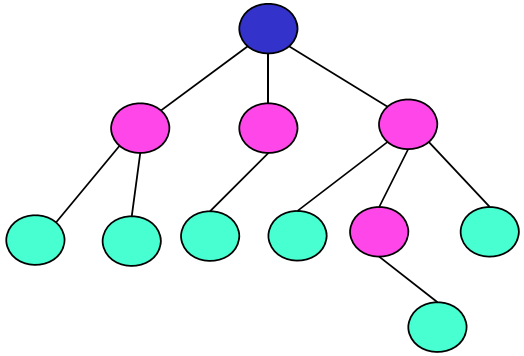


★ 线性结构：元素间存在一对一的关系



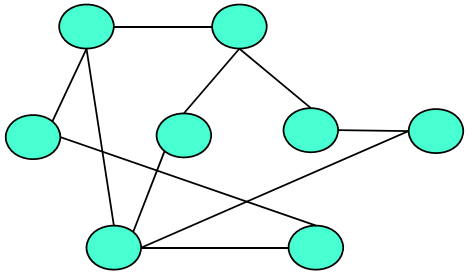
- 存在唯一一个称为"第一"的数据元素
- 存在唯一一个称为"最后"的数据元素
- 除最后一个外，每个元素仅有一个后继
- 除第一个外，每个元素仅有一个前驱

★ 树形结构：元素间存在一对多的关系



- 有唯一的一个点没有前驱，仅有若干后继(1-n)，称为"树根"
- 有若干点仅有一个前驱，没有后继，称为"树叶"
- 有若干点仅有一个前驱，若干后继(1-n)，称为"树枝"

★ 图(网)状结构：元素间存在多对多的关系



- 每个节点都可有任意前驱(1-m)和任意后继(1-n)

§1. 绪论

1.2. 基本概念和术语

1.2.2. 数据结构

数据结构的基本结构及图示表示：

★ 集合：极为松散的结构，暂不讨论

★ 线性结构：线性关系

★ 树形结构：层次关系 } 非线性关系

★ 图(网)状结构： }

树的特例

图的特例

图

数据结构的形式定义（数学表示）：

数据结构是一个二元组，表示为：Data_structure = (D, S)

其中：D：数据元素的有限集

S：D上关系的有限集

例：有表格如右，请问下列三种分别是什么类型的数据结构？

编号	姓名	性别	出生年月	职务	部门
01			1966.08	处长	
02			1964.02	科长	教务科
03			1983.01	科员	...
04			1975.10	科员	...
05			1978.05	科长	选课中心
06			1967.03	科员	...
07			1956.12	科长	实践教学科
08			1974.02	科员	...
09			1983.04	科员	...
10			1972.03	科员	...

(1) 有一种结构 L=(D,S)，其中

D={01,02,03,04,05,06,07,08,09,10}

S={<07,02>,<02,01>,<01,06>,<06,10>,<10,08>,<08,04>,<04,05>,<05,03>,<03,09>}

(2) 有一种结构 T=(D,S)，其中

D={01,02,03,04,05,06,07,08,09,10}

S={<01,02>,<02,03>,<02,04>,<01,05>,<05,06>,<01,07>,<07,08>,<07,09>,<07,10>}

(3) 有一种结构 G=(D,S)，其中

D={01,02,03,04,05,06,07,08,09,10}

S={<01,02>,<02,01>,<01,04>,<01,05>,<02,08>,09,04>,<05,09>,<08,03>,<03,07>,
<06,07>,<09,10>,<08,10>,<10,06>,<08,06>}

§1. 绪论

1.2. 基本概念和术语

1.2.3. 逻辑结构和物理结构

逻辑结构：数据结构的定义仅仅是数学描述，描述了数据元素之间的逻辑关系，也称为逻辑结构

物理结构：指数据结构(逻辑结构)在计算机内部的表示，包括数据元素和数据元素之间关系的表示，也称为存储结构

- 存储器模型：一个存储器M是一系列固定大小的存储单元，每个单元有一个唯一的地址，该地址被连续编码，每个单元有一个唯一的后继单元(一维线性结构)
- 物理结构就是逻辑结构到存储器的一个映射

数据元素的物理表示：

- 位：1 bit
- 位串：表示一个数据元素的若干bit(多个byte)，称为元素/结点
- 子位串：位串中各个数据项的位置，称为数据域

数据元素的物理存储：

- 顺序存储结构：借助元素在存储器的相对位置来表示数据元素之间的逻辑关系
  - 优点：查找效率高，可随机访问
  - 缺点：插入/删除时需要移动大量数据
- 链式存储结构：借助指示元素存储地址的指针表示数据元素之间的逻辑关系
  - 优点：插入/删除方便，不需要移动数据
  - 缺点：查找效率低，只能顺序访问
- 索引存储方法：在存储结点的同时，还建立附加的索引表，索引表中的每一项称为索引项
- 散列存储方法：根据结点的关键字直接计算出该结点的存储地址

§1. 绪论

1.2. 基本概念和术语

1.2.4. 数据类型

定义：一个值的集合以及定义在这个值集上的一组操作的总称

例：short型数据

值集：-32768 ~ +32767

操作：+、-、*、/、%、>>、<<等

分类：

- 原子类型：值不可再分解的类型
  (int，float等基本类型)
- 结构类型：由若干原子类型或结构类型按某种方式组合而成的类型，值可再分解
  (数组、struct、union等组合类型)

★ 数据结构可看做一组具有相同结构的值

★ 结构类型可看做一种数据结构及定义在其上的一组操作组成

★ 作用是实现数据封装及低层次的隐蔽

§1. 绪论

1.2. 基本概念和术语

1.2.5. 抽象数据类型

定义：一个数学模型以及定义在该模型上的一组操作

★ 仅体现逻辑特性

★ 仅研究逻辑特性而不关心具体的表示和实现

★ 可以理解为面向对象中的信息的封装与隐蔽

形式：三元组（D,S,P）

D：数据对象

S：D上的关系集

P：对D的基本操作集

分类：

- 原子类型：值不可再分解
- 固定聚合类型：值由数量、类型固定的成分按某种方式组合，值可再分解（结构体）
- 可变聚合类型：值由数量、类型不固定的成分按某种方式组合，值可再分解（数组）

§1. 绪论

1.2. 基本概念和术语

1.2.6. 多形数据类型

定义：值的成分不确定的数据类型

★ 某些抽象特征的讨论与元素的具体数据类型无关，则可以使用多形数据类型

例：加法可适用多种数据类型

```
int + int
double + double
Complex + Complex    (Complex类重载+)
```

=> 讨论加法操作是不必考虑具体数据类型

★ 相当于C++中的函数模板和类模板

★ 本书绝大多数数据类型均是多形数据类型

1.3. 抽象数据类型的表示与实现

P.10-11　C基础上抽取的类C子集

P.12-13　例1-7

§1. 绪论

1.4. 算法和算法分析

算法的含义：算法是指对特定问题求解步骤的一种描述是指令的有限序列，每条指令表示一个或几个基本操作

- 数值算法：求解数值问题
- 非数值算法：求解非数值问题

算法的五个基本特征：

★ 有穷性：每个算法在有穷步骤内完成；每个步骤都在有穷时间内完成(要考虑合理性)

★ 确定性：每条指令有确切含义，不产生二义性；对相同输入只能得到相同输出

★ 可行性：算法中所有操作都可以通过已实现的基本运算执行有限次数实现(操作足够基本)

★ 输入：有0-n个输入

★ 输出：有1-n个输出

算法的描述方法：文字、流程图、伪码、程序等

§1. 绪论

1.4. 算法和算法分析

算法设计的要求：

★ 正确性：功能正确，描述不含二义性，在有限的步骤内获得结果

- 不含有语法错误
- 对常规的合法输入能得到正确结果
- 对边界数据，能得到正确的结果
- 不含逻辑错误，对所有合法输入能得到正确结果

（非常困难，甚至不可能）

★ 可读性：方便程序员理解/编码/阅读/调试/修改

（先对人，后对机器）

★ 健壮性：对非法输入等意外情况所做出的正确反映

- 发现错误并用预置的正确方法去应对
- 发现错误后报告错误并返回，由上层处理
- 发现错误后直接中止程序的执行

```
//常量被0除
int main()
{
   cout << 10/0 << endl;
   return 0;
}  //编译报错
```

```
error C2124: 被零除或对零求模
```

```
//变量被0除
int main()
{  int i=0;
   cout << 10/i << endl;
   return 0;
}  //编译不错运行错
```

```
demo.exe (进程 6384)已退出，代码为 -1073741676。
```

★ 效率与低存储量需求：效率是指算法的执行时间，存储量需求是指算法执行过程中所需要的最大存储空间，

尽量做到执行快，空间少（相同问题，相同环境）

§1. 绪论

1.4. 算法和算法分析

算法效率的度量：

★ 效率：算法的执行时间

★ 度量方法：

事后统计：不同算法的程序通过一组/多组相同数据来区分优劣（数据量大且跨数量级）

- 在无法确定算法优劣前，先要实现算法
- 对计算机的软硬件环境依赖性较大

事前估算：用数学方法确定算法运行工作量的大小

- 算法选用何种策略
- 问题的规模
- 所用的计算机程序设计语言（汇编 > C语言 > 其它高级程序设计语言）
- 编译器的质量，编译优化的质量
- 机器指令的执行速度

① 依据的算法选用何种策略；
② 问题的规模，例如求 100 以内还是 1000 以内的素数；
③ 书写程序的语言，对于同一个算法，实现语言的级别越高，执行效率就越低；
④ 编译程序所产生的机器代码的质量；
⑤ 机器执行指令的速度。

★ 度量标准的选择：

从算法中选取一种对于所研究问题来说是基本运算的原操作，以原操作被重复执行的次数来做为算法的时间量度

- 不计软硬件环境的影响，只考虑问题规模(n)对效率的影响
- 原操作应是其重复执行次数和算法的执行时间成正比的
- 原操作的执行次数，与包含它的语句的频度相同（频度：该语句重复执行的次数）

§1. 绪论

1.4. 算法和算法分析

★ 度量标准的选择：

从算法中选取一种对于所研究问题来说是基本运算的原操作，以原操作被重复执行的次数来做为算法的时间量度

- 不计软硬件环境的影响，只考虑问题规模(n)对效率的影响
- 原操作应是其重复执行次数和算法的执行时间成正比的
- 原操作的执行次数，与包含它的语句的频度相同（频度：该语句重复执行的次数）

例1：N x N的矩阵相加

```
for (i=0; i<n; i++)
    for (j=0; j<n; j++)
        c[i][j] = a[i][j] + b[i][j];      //此句原操作
```

例2：N x N的矩阵相乘

```
for (i=0; i<n; i++)
    for (j=0; j<n; j++) {
        c[i][j] = 0;
        for (k=0; k<n; k++)
            c[i][j] += a[i][k] * b[k][j]; //此句原操作，不需要再考虑是复合语句
    }
```

例3：累加求和

```
s = 0;
for (i=1; i<=n; i++)
    s += i;     //此句原操作
```

§1. 绪论

1.4. 算法和算法分析

算法效率的度量：

★ 时间复杂度的形式定义：

若原操作的执行次数是问题规模n的函数（即f(n)），则算法的时间量度记作：T(n)=O(f(n))，表示n增大时，执行次数的增长率与f(n)的增长率相同，称作算法的渐进时间复杂度

★ O的形式定义：

若f(n)是正整数n的函数，则T(n)=O(f(n))，若$n \geqslant n_0$(n_0足够大)时，存在两个正常数a，b，使 $a < T(n)/f(n) < b$ 均成立，则称f(n)是T(n)的同数量级函数（P.15 附注①另一种方法）

① “O”的形式定义为[2]：若 $f(n)$是正整数 n 的一个函数，则 $x_n=O(f(n))$表示存在一个正的常数 M，使得当 $n \geqslant n_0$ 时都满足 $|x_n| \leqslant M|f(n)|$。

★ 大O的基本运算规则：

规则1：kf(n)=O(f(n)) 忽略常数因子

规则2：若 f(n)=O(g(n)) 并且 g(n)=O(h(n))，则 f(n)=O(h(n)) 传递性

规则3：f(n)+g(n)=O(max{f(n),g(n)}) 加法

规则4：若f1(n)=O(g1(n)) 并且 f2(n)=O(g2(n))，则 f1(n)*f2(n)=O(g1(n)*g2(n)) 乘法

§1. 绪论

1.4. 算法和算法分析

算法效率的度量：

★ 时间复杂度的形式定义：

若原操作的执行次数是问题规模n的函数（即f(n)），则算法的时间量度记作：T(n)=O(f(n))，表示n增大时，执行次数的增长率与f(n)的增长率相同，称作算法的渐进时间复杂度，简称时间复杂度

例1：N x N的矩阵相加

```
for (i=0; i<n; i++)
    for (j=0; j<n; j++)
        c[i][j] = a[i][j] + b[i][j];
```

$O(n^2)$

例2：N x N的矩阵相乘

```
for (i=0; i<n; i++)
    for (j=0; j<n; j++) {
        c[i][j] = 0;
        for (k=0; k<n; k++)
            c[i][j] += a[i][k] * b[k][j];
    }
```

$O(n^3)$

例3：累加求和

```
s = 0;
for (i=1; i<=n; i++)
    s += i;
```

$O(n)$

例4：内循环取决于外循环

```
for (i=0; i<n; i++)
    for(j=0; j<i; j++)
        原操作语句;
```

$$\sum_{k=0}^{n-1} k = \frac{n(n-1)}{2}$$

$O(n^2)$

推论：

当T(n)是n的多项式时，f(n)是T(n)的最高次幂

§1. 绪论

★ 各种不同的时间复杂度的比较：

假设某原操作每执行一次用时1us(10^{-6}秒)，则算法运行时间与T(n)的关系如下表所示：

数据单位：无 - us　　ms - 毫秒　　s - 秒　　min - 分钟　d - 天　　c - 世纪

有倒挂

T(n)	$\log_2 n$	n	$n\log_2 n$	n^2	n^3	n^5	2^n	n!
n=20	4.3	20	86.4	400	8ms	3.2s	1.05s	771c
n=40	5.3	40	213	1.6ms	64ms	1.7min	12.7d	2.59×10^{32}c
n=60	5.9	60	354	3.6ms	216ms	13min	366c	2.64×10^{66}c

以n=20，O(n!)为例：

20! = 2432902008176640000 微秒 (/1000000)
= 2432902008176.64 秒 (/86400)
= 28158588.0576 天 (/365)
= 77146.81 年 (/100)
= 771 世纪

假设计算机运行速度快了1000倍(1ns=10^{-9}秒)
本算法仍需要77年

结论：

(1) 当T(n)是对数函数、幂函数或它们的乘积时，运行时间是可以接受的，称为有效算法

(2) 当T(n)为指数函数、阶乘时，算法的运行时间是不可接受的，称为"坏"算法/无效算法

(3) 当n增长时，各种T(n)的增长速度不同，当n足够大时，T(n)存在如下关系：
$O(\log_2 n)<O(n)<O(n\log_2 n)<\ldots<O(2^n)<o(n!)$

(4) 如果同一问题的两种不同的算法大O相同，则还需要考虑常数优化

§1. 绪论

1.4. 算法和算法分析

★ 平均/最坏/最好时间复杂度：

- 平均时间复杂度：在某些情况下，原操作可能执行，也可能不执行，取其平均值，称平均时间复杂度
- 最坏时间复杂度：在某些情况下，原操作可能执行，也可能不执行，假设每次均执行，称最坏时间复杂度
- 最好时间复杂度：在某些情况下，原操作可能执行，也可能不执行，假设每次均不执行，称最好时间复杂度

★ 除特别声明外，均指最坏时间复杂度

例：n个数进行冒泡排序	左例分析结论：

for(j=0; j<n-1; j++)
for(i=0; i<n-1-j;i++)
if (a[i] > a[i+1]) {
t=a[i];
a[i]=a[i+1];
a[i+1]=t;
}
	1、若n个数本身有序，则一次交换也不需要做
	2、若n个数逆序，则达到最大交换次数
	3、数字随机排列，则交换次数介于两者之间

§1. 绪论

1.4. 算法和算法分析

★ 算法的空间复杂度：

算法的存储空间时指算法本身的指令、常量、变量、输入数据所占的计算机内部存储空间以及用做存放数据操作过程中的临时变量等工作单元的辅助空间(若辅助空间相对于输入数据量是常数，则称此算法是原地工作)

★ 空间复杂度的形式定义

若算法的存储空间是问题规模n的函数（即f(n)），则算法的空间复杂度记作：S(n) = O(f(n))

★ 平均/最坏/最好空间复杂度

同时间复杂度

1.5. 算法的评价与比较

★ "好"算法的定义

一个算法如果能在所要求的资源限制(空间+时间)内将问题解决好，则这个算法是"好"算法

★ 算法的评价标准

- 如果在时间和空间的使用上无法同时兼顾，则可以"以空间换时间"或"以时间换空间"
- 从未来的发展趋势看，倾向于"空间换时间"
- 在时间/空间复杂度相同的情况下，换成效率更高的原操作也可以提高算法的效率(但这种方法不会改变大O)
- 在提高算法效率的同时，也要兼顾可读性