数据结构与算法

李锡祚

大连民族大学 计算机科学与工程学院

第 1 章 绪 论

本章的基本内容是：

1.1 什么是数据结构？

1.2 算法及算法分析

例1：书店往往是书的海洋，下图显示了巴西圣保罗Livraria da Vila书店一角。如果你是书店的主人，该如何摆放书店里的书，才能让读者很方便地找书？

分析

方法1：随便放。当有新书进来，哪有空就把书插到哪里。

查找比较痛苦，最不幸的是翻遍了也没找到！

方法2：按书名的拼音字母顺序排放

查找方便了，但插入一本新书比较困难！

方法3：划分几个区域，每个区域摆放某类图书；每类再按书名的拼音字母顺序排放

和方法2比较，查找、插入的工作量减少了，但区域大小不好事先确定。可能造成空间浪费！

你还有更好的解决方案吗？

结论1：查找图书的效率和数据组织方式有关系。

例2：编写程序实现一个函数PrintN，使得传入一个正整数参数N后，能顺序打印出从1到N的全部正整数。

代码2.1：

```
Void PrintN (int N)
{
    int i;
    for ( i=1; i<=N; i++)
        cout<<i<<endl;
}
```

代码2.2：

```
Void PrintN (int N)
{
    if (!N) return;
    printN (N-1);
    cout<<N<<endl;
    return;
}
```

对于充分大的N，代码2.2中的递归函数拒绝工作，而代码2.1仍然继续工作。为什么？

结论2：解决问题的效率与空间的利用率有关。

例3：多项式的标准表达式可以写为：$f(x)=a_0+a_1x+\cdots+a_{n-1}x^{n-1}+a_nx^n$。现给定一个多项式的阶数n，并将全体系数$a_i$存放在数组a[]里。请编写程序计算这个多项式在给定点x处的值。

代码3.1（直接算法）：

```
double f (int n, double a[], double x)
{
   int i;
   double p =a[0];
   for ( i=1; i<=n; i++)
      p += a[i]*pow(x, i);
   return p;
}
```

代码3.2（秦九韶算法）：

```
double f (int n, double a[], double x)
{
   int i;
   double p =a[n];
   for ( i=n; i>=0; i--)
      p = a[i-1]+x*p;
   return p;
}
```

这两个算法有什么不一样呢？

代码4 测试函数function()的运行时间

```
#include <stdio.h>
#include <time.h>

clock_t  start, stop; /* clock_t是clock()函数返回的变量类型 */
double  duration;  /* 记录被测函数运行时间，以秒为单位 */

int main ()
{ /* 不在测试范围内的准备工作写在clock()调用之前*/

  start = clock(); /* 开始计时 */
  function();      /* 把被测函数加在这里 */
  stop = clock();  /* 停止计时 */
  duration = ((double)(stop - start))/CLOCKS_PER_SEC;
     /* 计算运行时间 */

  /* 其他不在测试范围的处理写在后面，例如输出duration的值 */

  return 0;
}
```

测试结果：

秦九韶算法的计算速度明显比
直接法快了一个数量级！

为什么会这样？

结论3：秦九韶算法通过不断提取公因式x来减少乘法次数，把

多项式：$f(x)=a_0+a_1x+\cdots+a_{n-1}x^{n-1}+a_nx^n$

改为：$f(x)=a_0+x(a_1+x(\cdots(a_{n-1}+x(a_n))\cdots))$。

计算速度明显比直接算法快了一个数量级。

例1：图书摆放策略
例2：Print(N)的循环与递归实现
例3：多项式求值

解决问题的效率，跟数据的组织方式有关(如例1)，跟空间的利用效率有关（如例2），也跟算法的巧妙程度有关（如例3）。

那么，什么是数据结构？“数据结构是计算机中存储、组织数据的方式。精心选择的数据结构可以带来高效率的算法。”

数据结构的基本概念

- 数据(data)：所有能输入到计算机中并能被计算机程序识别和处理的符号集合。
  数值数据：整数、实数等
  非数值数据：图形、图像、声音、文字等
- 数据元素(data element)：数据的基本单位，在计算机程序中通常作为一个整体进行考虑和处理。
- 数据项(data item)：构成数据元素的不可分割的最小单位。
- 数据对象(data object)：具有相同性质的数据元素的集合。

数据、数据元素、数据项之间的关系

- 包含关系：数据是由数据元素组成，数据元素是由数据项组成。

- 数据元素是讨论数据结构时涉及的最小数据单位，其中的数据项一般不予考虑。

数据结构的定义

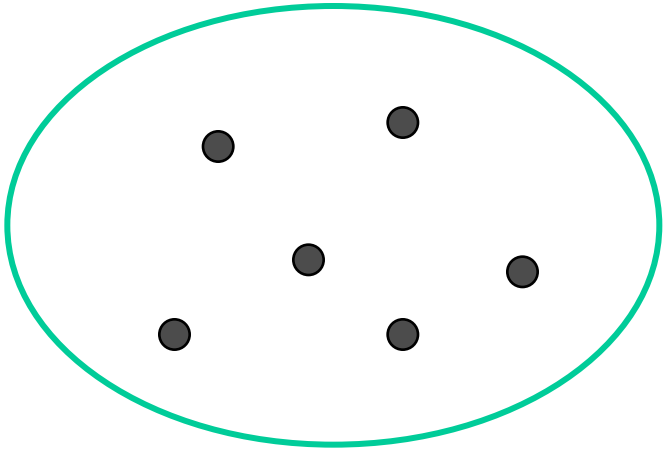
- 数据结构(data structure)：相互之间存在一定关系的数据元素的集合。按照视角的不同，数据结构分为逻辑结构和存储结构。
- 逻辑结构(logical structure):指数据元素之间逻辑关系的整体。

关联方式或邻接关系

数据结构的定义

数据结构从逻辑上分为四类：

(1) 集合：数据元素之间就是“属于同一个集合”；



数据结构的定义

数据结构从逻辑上分为四类：

(1) 集合：数据元素之间就是“属于同一个集合”；

(2) 线性结构：数据元素之间存在着一对一的线性关系；

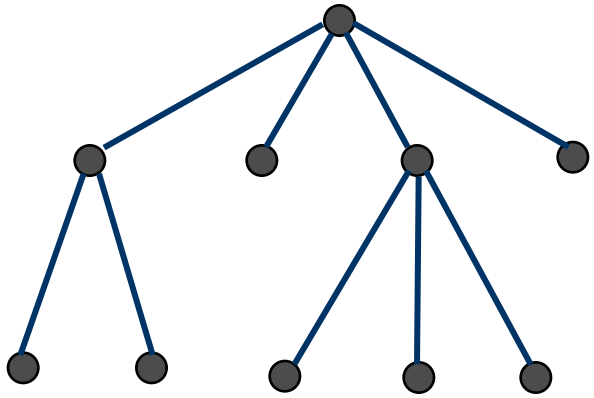
数据结构的定义

数据结构从逻辑上分为四类：

(1) 集合：数据元素之间就是“属于同一个集合”；

(2) 线性结构：数据元素之间存在着一对一的线性关系；

(3) 树结构：数据元素之间存在着一对多的层次关系；



数据结构的定义

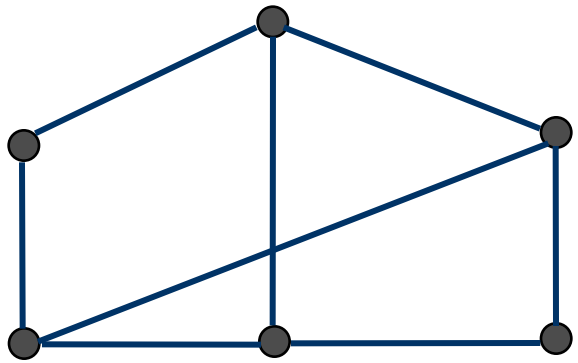
数据结构从逻辑上分为四类：

(1) 集合：数据元素之间就是“属于同一个集合”；

(2) 线性结构：数据元素之间存在着一对一的线性关系；

(3) 树结构：数据元素之间存在着一对多的层次关系；

(4) 图结构：数据元素之间存在着多对多的任意关系。



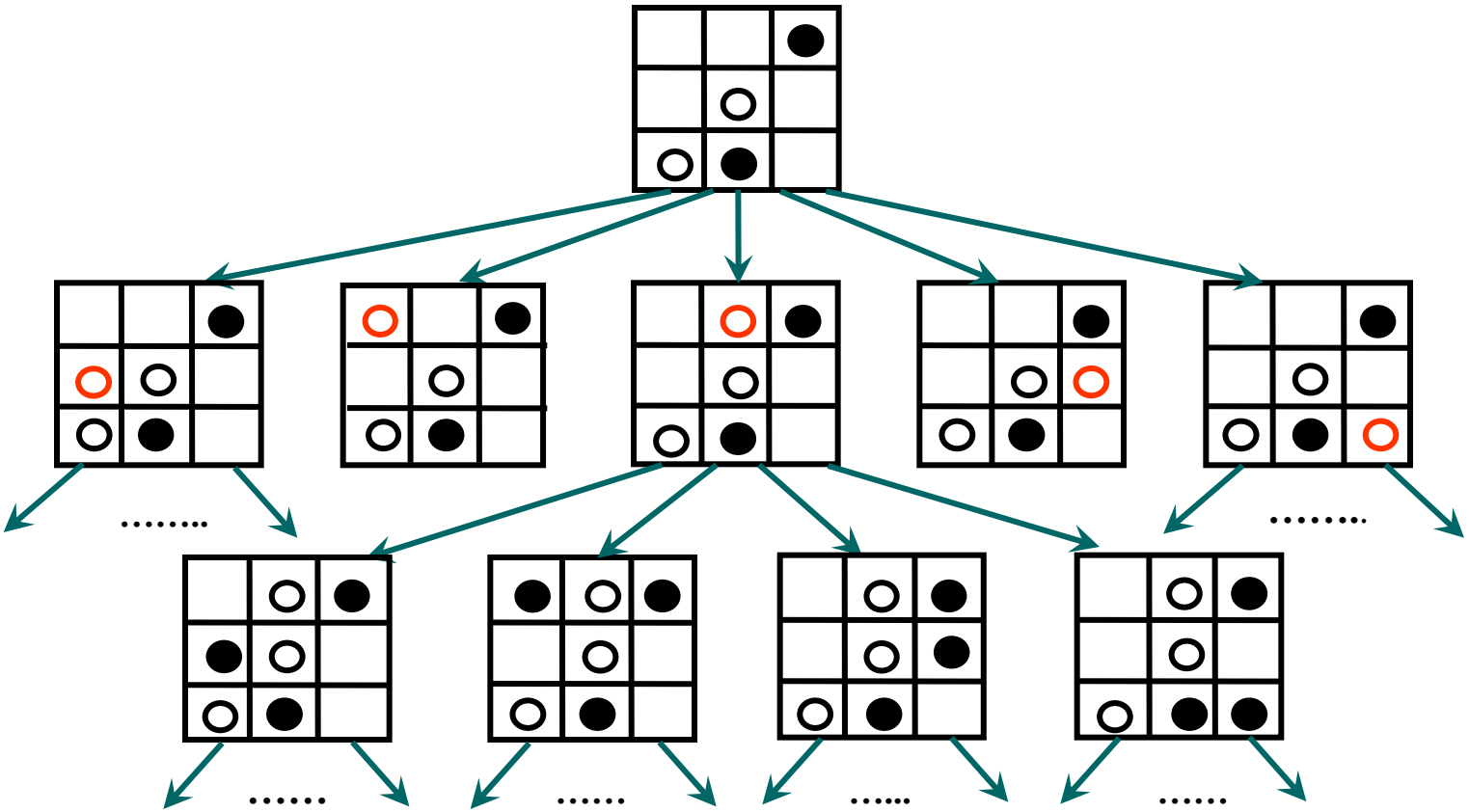
例1　学籍管理问题

各表项之间是什么关系？　抽象出的模型是什么？

学号	姓名	性别	出生日期	政治面貌
0001	王　军	男	**1983/09/02**	团员
0002	李　明	男	**1982/12/25**	党员
0003	汤晓影	女	**1984/03/26**	团员
…	…	…	…	…

例2　人机对弈问题

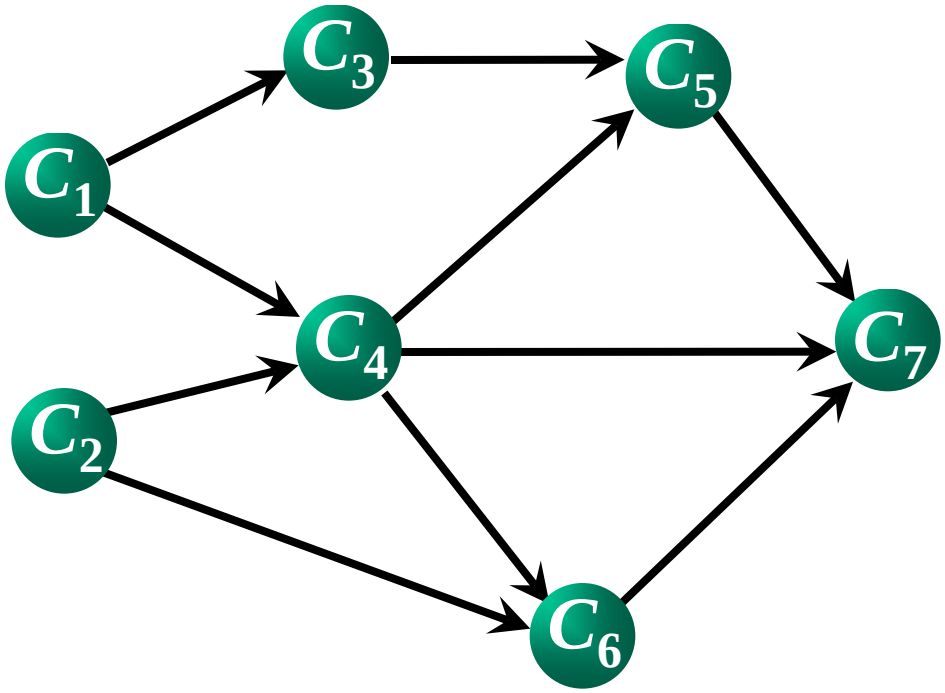
各格局之间是什么关系？ 抽象出的模型是什么？



例3 教学计划编排问题

如何表示课程之间的先修关系？抽象出的模型是什么？

编号	课程名称	先修课
C_1	高等数学	无
C_2	计算机导论	无
C_3	离散数学	C_1
C_4	程序设计	C_1, C_2
C_5	数据结构	C_3, C_4
C_6	计算机原理	C_2, C_4
C_7	数据库原理	C_4, C_5, C_6



数据结构的定义（续）

- 存储结构(storage structure)又称为物理结构，是数据及其逻辑结构在计算机中的表示。

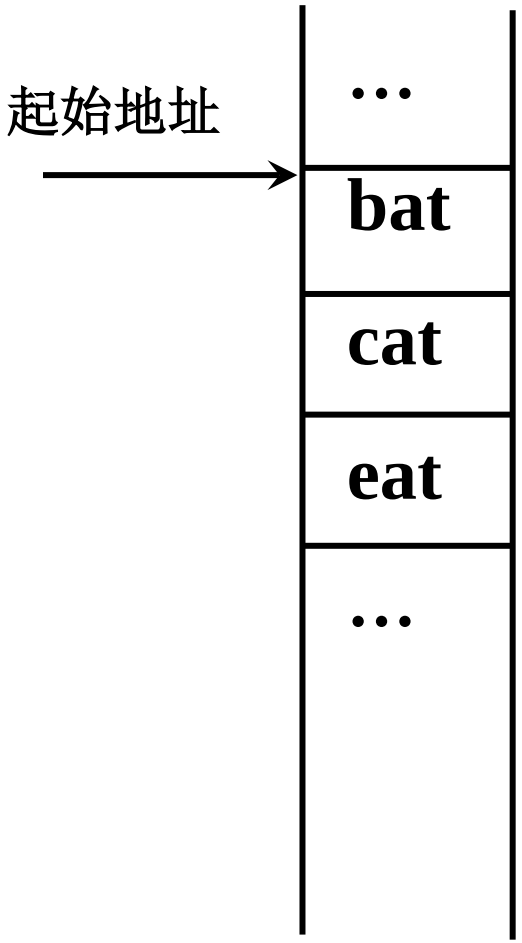
内存

存储结构实质上是内存分配，
在具体实现时，依赖于计算机语言。

两种存储结构

1. 顺序存储结构(sequential storage structure)： 用一组连续的存储单元依次存储数据元素，数据元素之间的逻辑关系由元素的存储位置来表示。

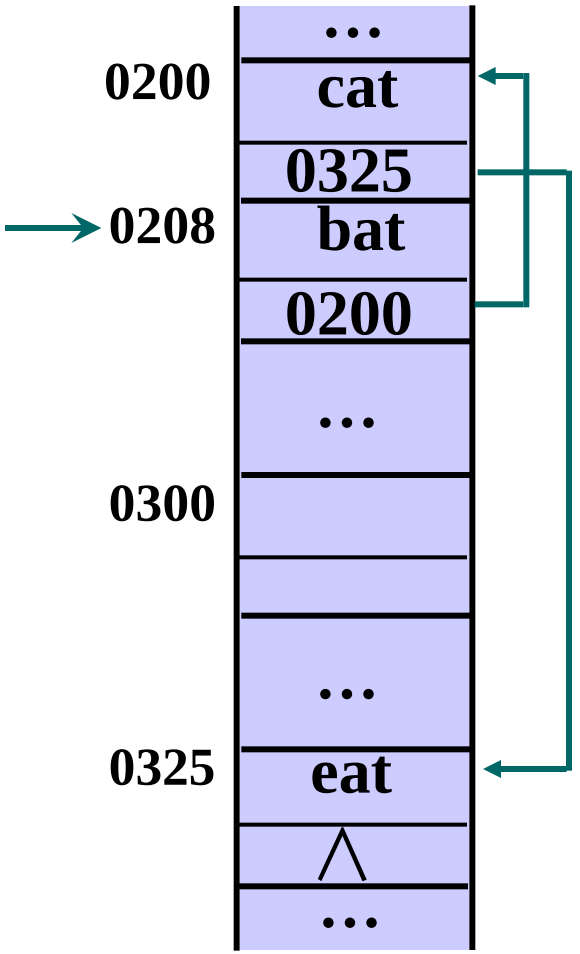
例：（bat, cat, eat）



两种存储结构

2. 链接存储结构(linked storage structure)：用一组任意的存储单元存储数据元素，数据元素之间的逻辑关系用指针来表示。

例：（bat, cat, eat）



逻辑结构和存储结构之间的关系

- 数据的逻辑结构属于用户视图，是面向问题的，反映了数据内部的构成方式；数据的存储结构属于具体实现的视图，是面向计算机的。

- 一种数据的逻辑结构可以用多种存储结构来存储，而采用不同的存储结构，其数据处理的效率往往是不同的。

抽象数据类型

1. 数据类型（Data Type）： 一组值的集合以及定义于这个值集上的一组操作的总称。

例如：C++中的整型变量

2. 抽象（Abstract）: 抽出问题本质的特征而忽略非本质的细节。

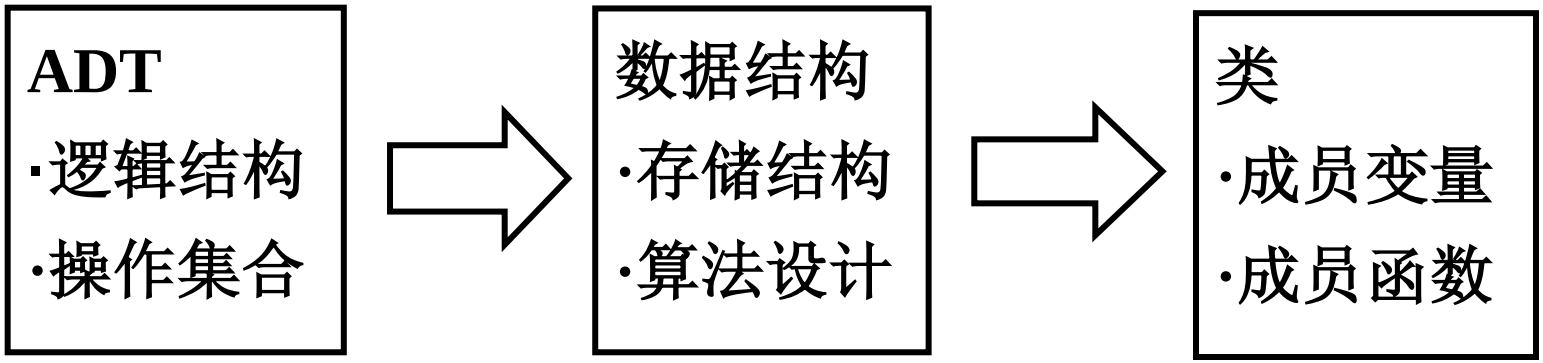
例如： 表结构（线性表）、树结构、图结构等

3. 抽象数据类型（Abstract Data Type，ADT）: 一个数据结构以及定义在该结构上的一组操作的总称。

ADT的描述方法与面向对象的思想是一致的，它把数据对象和相关操作封装在一起。

ADT是对数据类型的进一步抽象

ADT的不同视图



(a)使用视图
ADT的定义

(b) 设计视图
ADT的设计

(c) 实现视图
ADT的实现

ADT的定义形式

ADT 抽象数据类型名

Data

数据元素之间逻辑关系的定义

Operation

操作1

前置条件：执行此操作前数据所必须的状态

输　　入：执行此操作所需要的输入

功　　能：该操作将完成的功能

输　　出：执行该操作后产生的输出

后置条件：执行该操作后数据的状态

操作2

……

……

操作n

……

endADT

例如：“表”的抽象数据类型定义

ADT List

Data

线性表中的数据元素具有相同类型， 相邻元素具有前驱和后继关系

Operation

InitList

前置条件：表不存在
输入：无
功能：表的初始化
输出： 无
后置条件：建一个空表

Locate

前置条件：表已存在
输入：数据元素x
功能：在线性表中查找值等于x的元素
输出：若查找成功，返回x在表中的序号，否则返回0
后置条件：表不变

Insert

前置条件：表已存在
输入：插入i；待插x
功能：在表的第i个位置处插入一个新元素x
输出：若插入不成功，抛出异常
后置条件：若插入成功，表中增加一个新元素

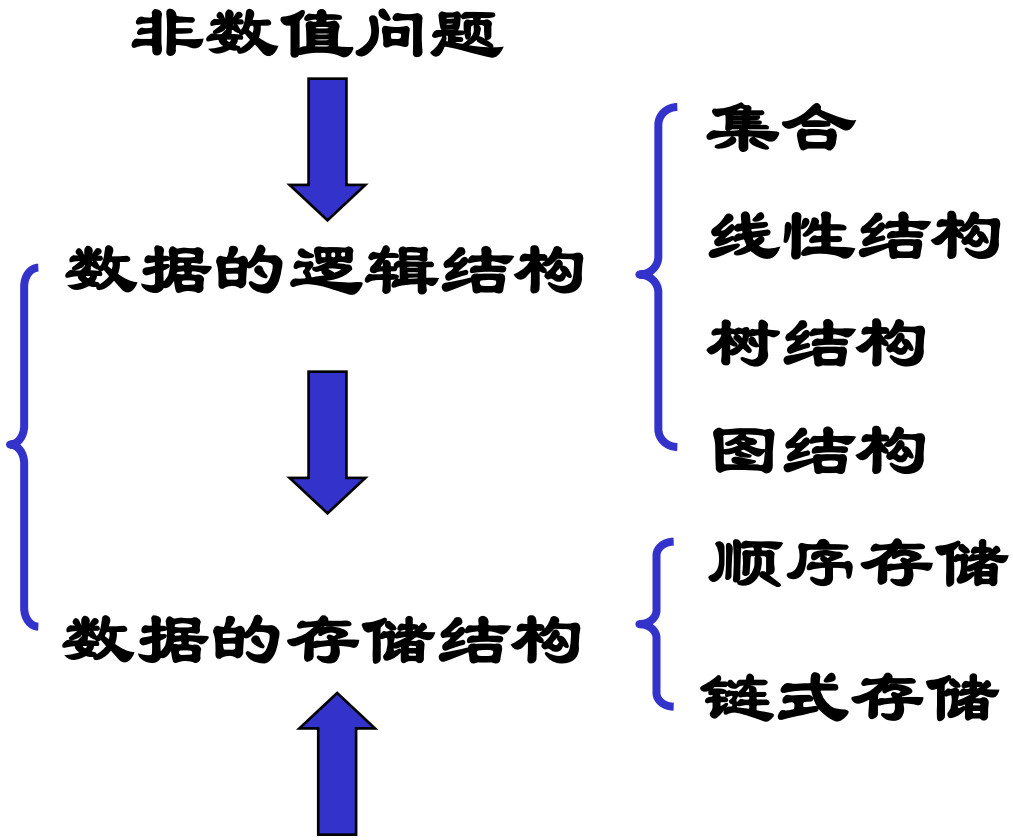
Delete

前置条件：表已存在
输入：删除位置i
功能：删除表中的第i个元素
输出：若删除成功，返回被删元素，否则抛出异常
后置条件：若删除成功，表中减少一个元素

……

endADT

数据结构基本概念小结



数据的操作：插入、删除、修改、检索、排序等

1.2 算法及算法分析

1.算法（Algorithm）:为了解决某类问题而规定的一个有限长的操作序列。简单地说，一个算法就是一步一步地解决某种问题的方法。

算法举例

比如：在三个数中寻找最大的一个。

考虑如下算法，从三个数a, b, c中寻找最大值：

①输入a, b, c ;

②x=a;

③若b>x,则x=b;

④若c>x,则x=c;

⑤输出x。

算法的几个特性

① 输入：一个算法有零个或多个输入。

② 输出：一个算法有一个或多个输出。

③ 确定性：算法中的每一条指令必须有确切的含义，不可以有歧义。对于相同的输入只能得到相同的输出。

④ 有穷性：一个算法必须总是在执行有穷步之后结束，且每一步都在有限时间内完成。

⑤ 可行性：算法中的所有操作可以通过已经实现的基本操作执行有限次来实现。换句话说，每一步必须在计算机能处理的范围之内。

算法的描述方法

例：求两个自然数 m 和 n 的最大公约数

（欧几里德算法，Euclidean algorithm，也叫辗转相除法）：

例：欧几里德算法

自然语言

①输入m 和n；

②求m除以n的余数r；

③若r等于0，则n为最大公约数，算法结束；否则执行第④步；

④将n的值放在m中，将r的值放在n中；

⑤重新执行第②步。

算法的描述方法——自然语言

优点：容易理解

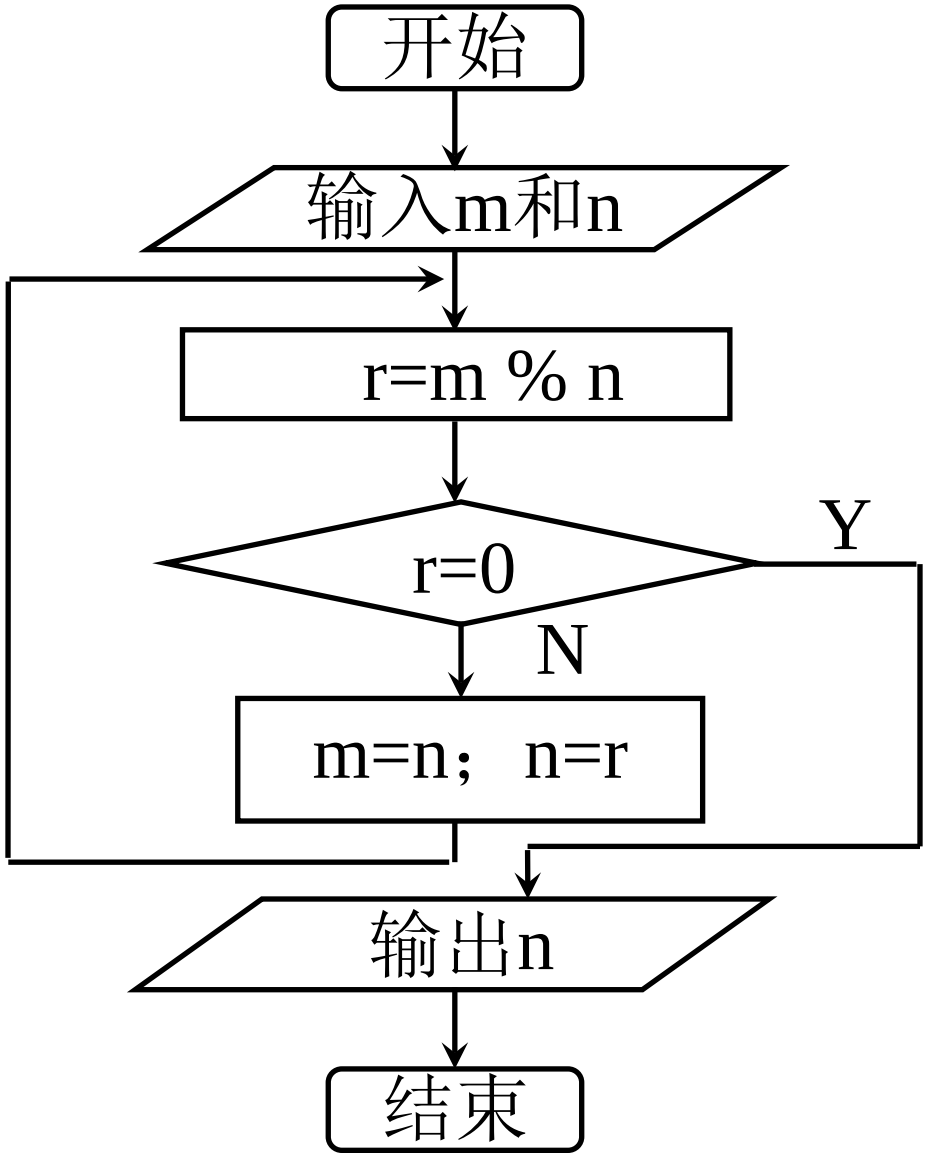
缺点：冗长、二义性

适用：粗线条描述算法思想

注意事项：避免写成自然段

例：欧几里德算法

流程图



算法的描述方法——程序流程图

优点：流程直观、控制灵活

缺点：缺少严密性、篇幅过长

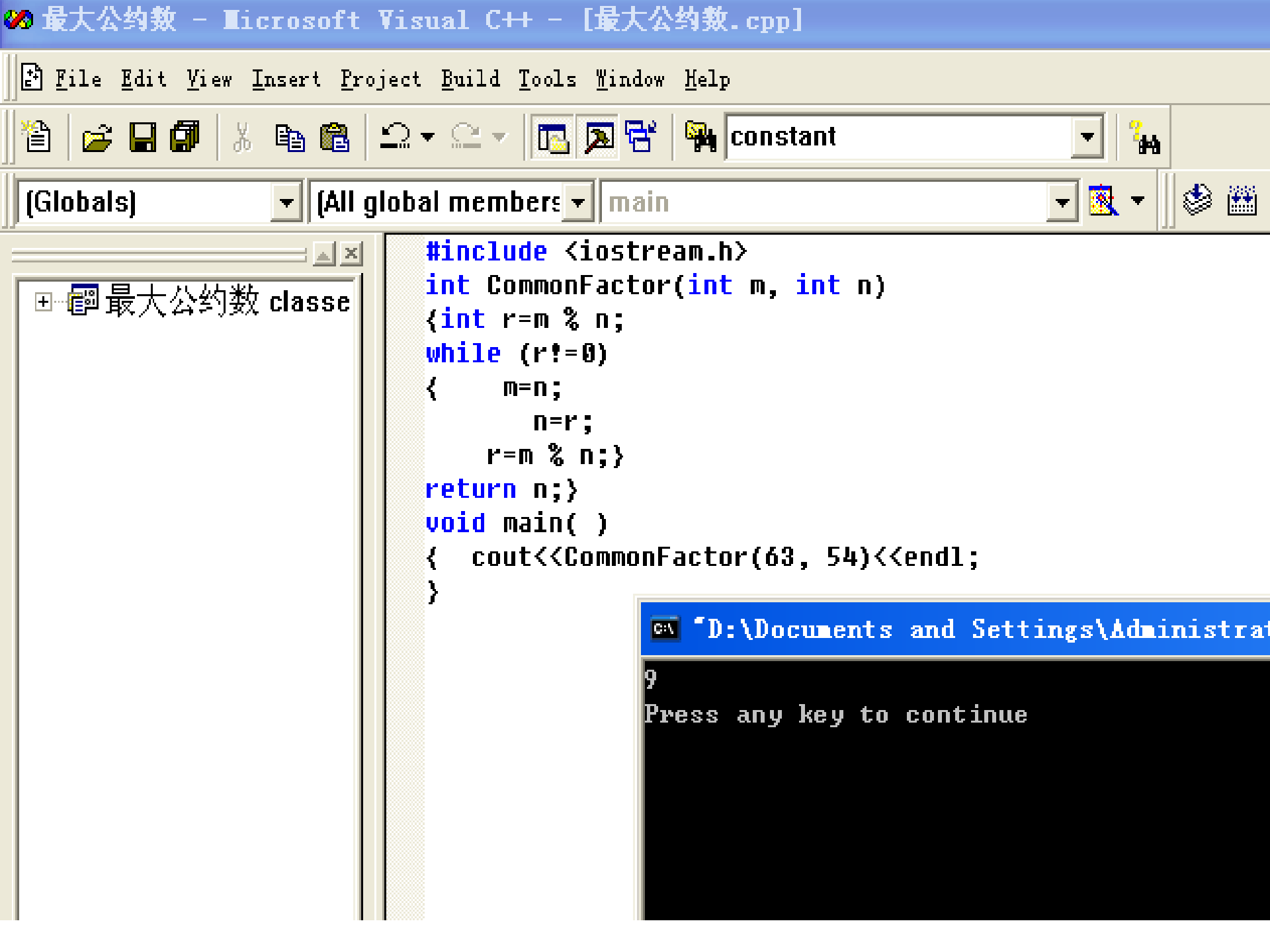
适用：描述简单算法

注意事项：注意抽象层次

例：欧几里德算法

程序设计语言

```
#include <iostream.h>
int CommonFactor(int m, int n)
{
    int r=m % n;
    while (r!=0)
    {
       m=n;
       n=r;
       r=m % n;
    }
    return n;
}
void main( )
{
   cout<<CommonFactor(63, 54)<<endl;
}
```


最大公约数 - Microsoft Visual C++ - [最大公约数.cpp]
File Edit View Insert Project Build Tools Window Help
constant
(Globals)
(All global members
main
最大公约数 classe
#include <iostream.h>
int CommonFactor(int m, int n)
{int r=m % n;
while (r!=0)
{ m=n;
n=r;
r=m % n;}
return n;}
void main( )
{ cout<<CommonFactor(63, 54)<<endl;
}
"D:\Documents and Settings\Administrat
9
Press any key to continue

算法的描述方法——程序设计语言

优点：能由计算机执行

缺点：抽象性差，对语言要求高

适用：算法需要验证

注意事项：将算法写成函数

例：欧几里德算法

伪代码

1. 输入m和n;
2. r = m % n;
3. 循环直到 r 等于0
   2.1 m = n;
   2.2 n = r;
   2.3 r = m % n;
4. 输出 n ;

上述伪代码再具体一些，用C++的函数来描述。请同学们自行完成！

算法的描述方法——伪代码

伪代码（Pseudocode）：介于自然语言和程序设计语言之间的方法，它采用某一程序设计语言的基本语法，操作指令可以结合自然语言来设计。

优点：表达能力强，抽象性强，容易理解

什么是“好”的算法？

具体衡量、比较算法优劣的指标主要有两个：

空间复杂度S(n)：根据算法写成的程序在执行时占用存储单元的长度。这个长度往往与输入数据的规模有关。空间复杂度过高的算法可能导致使用的内存超限，造成程序非正常中断。

例1.2 的实现函数PrintN的递归算法S(n)太大。

时间复杂度 T(n)：根据算法写成的程序在执行时耗费时间的长度。这个长度往往也与输入数据的规模有关。时间复杂度过高的低效算法可能导致我们在有生之年都等不到运行结果。

例1.3 的秦九韶算法的T(n)比较小。

算法分析

算法分析（Algorithm Analysis）：对算法所需要的计算机资源——时间和空间进行估算。

- 时间复杂性（Time Complexity）
- 空间复杂性（Space Complexity）

算法分析的主要目的是什么？

度量算法效率的方法

- 事后统计：将算法实现之后，再测算其时间和空间开销。

  缺点：(1) 编写程序实现算法将花费较多的时间和精力；

  (2) 实验结果依赖于计算机的软硬件等环境因素。

- 事前分析：对算法所消耗资源的一种估算方法。

和算法执行时间相关的因素

1. 算法选用的策略

2. 问题的规模

3. 编写程序的语言

4. 编译程序产生的机器代码的质量

5. 计算机执行指令的速度

算法的时间复杂度分析

算法的执行时间＝每条语句执行时间之和

每条语句执行次数之和

基本语句的执行次数

执行次数×执行一次的时间

单位时间

指令系统、编译的代码质量

算法的时间复杂度分析

和算法执行时间相关的因素：

1. 算法选用的策略
2. √ 问题的规模
3. 编写程序的语言
4. 编译程序产生的机器代码的质量
5. 计算机执行指令的速度

因此，一个特定算法的“运行工作量”的大小，只依赖于问题的规模（通常用整数量n表示），或者说，它是问题规模的函数。

算法的时间复杂度分析

- 问题规模（problem scope）：输入量的多少。
- 基本语句(basic statement)：是执行次数与整个算法的执行次数成正比的操作指令。
- 基本语句的执行次数（或基本语句频度）记作f（n）。

例如：

```
for (i=1; i<=n; i++)
    for (j=1; j<=n; j++)
        x++;
```

问题规模：n

基本语句：x++

基本语句的执行次数:$f(n)=n^2$

算法的时间复杂度分析

假如，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，则可记作：

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

称T(n)为算法的渐近时间复杂度。

算法分析举例

例1-5　++x;

$O(1)$ 常数阶

例1-6　for（i=1; i<=n; ++i）

　　++x;

$O(n)$ 线性阶

例1-7　for (i=1; i<=n; ++i)

　　for (j=1; j<=n; ++j)

　　++x;

$O(n^2)$ 平方阶

算法分析举例

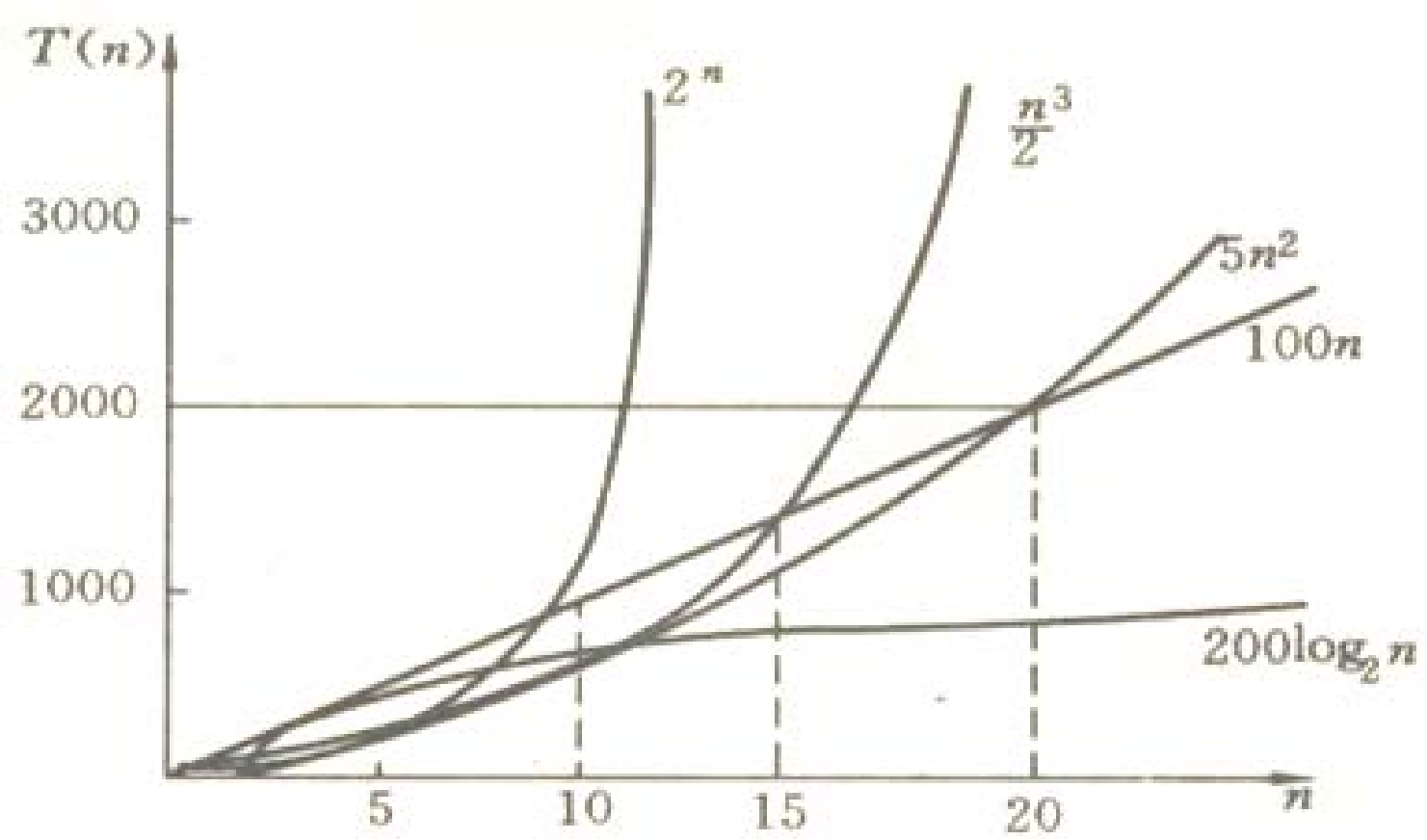
例1-8
```
for (i=1; i<=n; ++i)
    for (j=1; j<=n; ++j)
    {
      c[i][j]=0;
      for (k=1; k<=n; ++k)
        c[i][j]+=a[i][k]*b[k][j];
    }
```
$O(n^3)$

例1-9
```
for (i=1; i<=n; i=2*i)
    ++x;
```
$O(\log_2 n)$

$n=2^{f(n)}$

$$O(1)<O(\log_2 n)<O(n)<O(n\log_2 n)<O(n^2)<O(n^3)<\ldots<O(2^n)<O(n!)$$

部分函数的渐进增长趋势



一般来讲，具有多项式时间复杂度的算法是可接受的、可使用的算法，而具有指数时间复杂度的算法，只有当问题规模足够小时才是可使用的算法。

时间复杂度随n变化情况的比较

时间复杂度	n=10	n=100	n=1000
$O(1)$	1	1	1
$O(\log_2 n)$	3.322	6.644	9.966
$O(n)$	10	100	1000
$O(n\log_2 n)$	33.22	664.4	9966
$O(n^2)$	100	10000	10^6
$O(2^n)$	1024	约10^{30}	约10^{300}

最好、最坏、平均

例：在一维整型数组A[n]中顺序查找与给定值k相等的元素（假设该数组中有且仅有一个元素值为k）。

```
int Find(int A[ ], int n)
{
    for (i=0; i<n; i++)
        if (A[i]= =k) break;
    return i;
}
```

基本语句的执行次数是否只和问题规模有关？

最好、最坏、平均

结论：如果问题规模相同，时间代价与输入数据有关，则需要分析最好情况、最坏情况、平均情况。

- 最好情况：出现概率较大时分析
- 最差情况：实时系统
- 平均情况：已知输入数据是如何分布的，通常假设等概率分布

空间复杂度度量(Space complexity)

- **存储空间的固定部分**

  程序指令代码的空间，常数、简单变量、定长成分(如数组元素、结构成分、对象的数据成员等)变量所占空间。

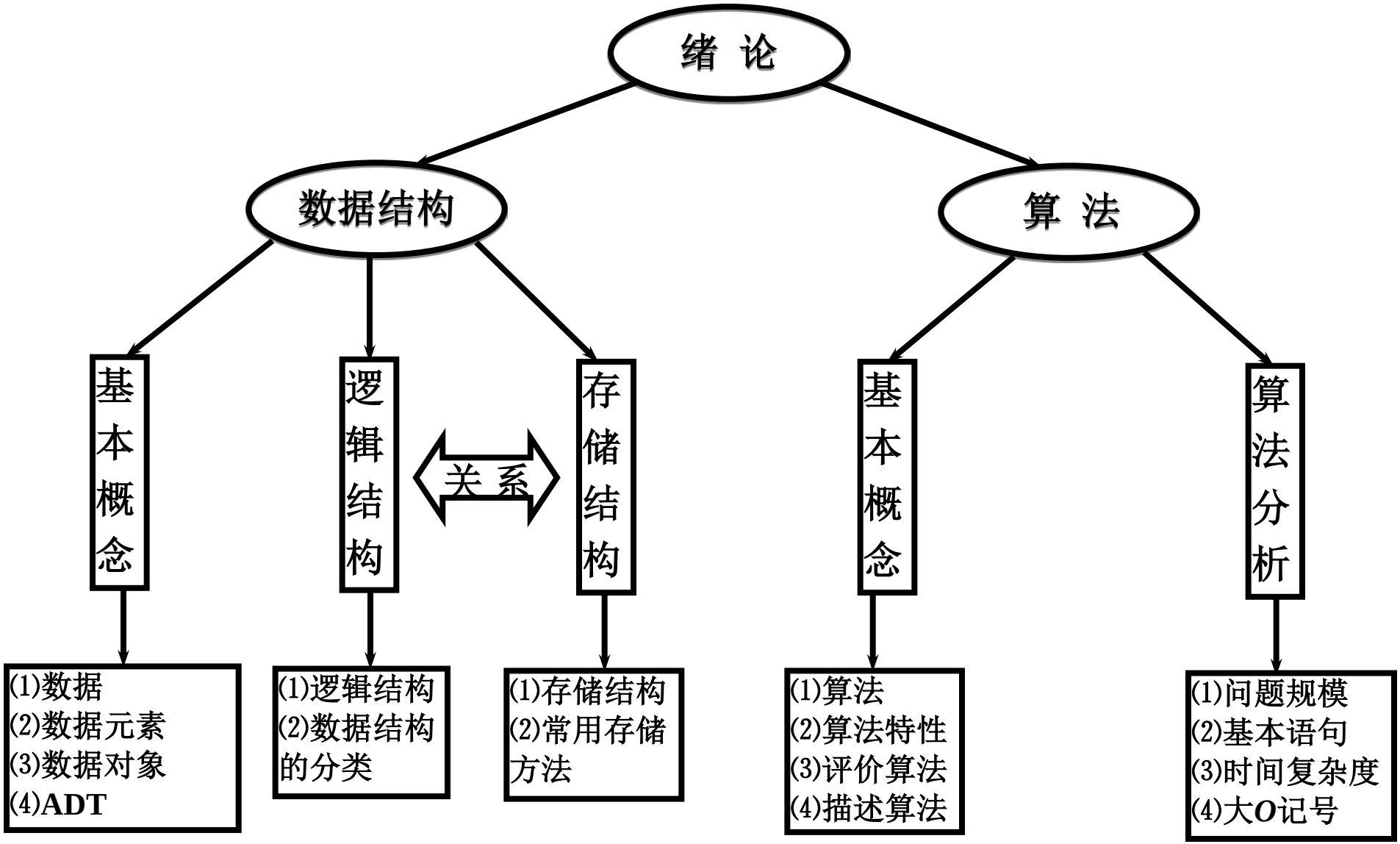
- **可变部分**

  尺寸与实例特性有关的成分变量所占空间、引用变量所占空间、递归栈所用空间、通过new和delete命令动态使用空间。

时空资源的折中原理(Balancing)

- 对于同一个问题求解，一般会存在多种算法。而这些算法在时空开销上的优劣往往表现出“时空折中”的性质。
- “时空折中”是指为了改善一个算法的时间开销，往往可以通过增大空间开销为代价，而设计出一个新算法来。
- 有时也可以为了缩小算法的空间开销，而牺牲计算机的运行时间，通过增大时间开销来换取存储空间的节省。

本章小结——知识结构图


绪 论
数据结构
算 法
基本概念
逻辑结构
关 系
存储结构
基本概念
算法分析
(1)数据
(2)数据元素
(3)数据对象
(4)ADT
(1)逻辑结构
(2)数据结构的分类
(1)存储结构
(2)常用存储方法
(1)算法
(2)算法特性
(3)评价算法
(4)描述算法
(1)问题规模
(2)基本语句
(3)时间复杂度
(4)大O记号

本章作业

习题1（P19）：4，5，6

常见的算法时间复杂度的分析举例

- 一个简单语句的时间复杂度为O(1)。

```
int count=0;
```

- 时间复杂度为O(*n*)的循环语句。

```
it n=8,count=0,i;
for(i=1;i<=n;i++)
   count++;                          //f(n)<=n
```

- 时间复杂度为$O(\log_2 n)$的循环语句。

```
int n=8,count=0,i;
for(i=1;i<=n;i= i* 2)
   count++;                          //  i=2^f(n)<=n,即f(n)<=log2n
```

- 时间复杂度为$O(n^2)$的二重循环。

```
count=0;
for(i=1;i<=n;i++)      //n
   for(j=1;j<=n;j++)   //n^2
      count++;                       //f(n)<=n^2
```

- 时间复杂度为$O(n\log_2 n)$的二重循环。

```
count=0;
for(i=1;i<=n;i*=2)          //log2n
   for(j=1;j<=n;j++)        //n*log2n
      count++;
```

- 时间复杂度为$O(n)$的二重循环。

```
count=0;
for(i=1;i<=n;i*=2)          //log2n
   for(j=1;j<=i;j++)        //1+2+4+8+…+ log2n=2n+1
      count++;
```

实验1

- 652：在一个有序表中插入2个元素。
- 654：删除所有值为x和y的元素。
- 702：找一数组中的最大值。