第一章 绪论

1

《数据结构与算法》是计算机科学中的一门综合性专业基础课。是介于数学、计算机硬件、计算机软件三者之间的一门核心课程，不仅是一般程序设计的基础，而且是设计和实现编译程序、操作系统、数据库系统及其他系统程序和大型应用程序的重要基础。

- 编译技术：栈、散列表及语法树
- 操作系统：队列、存储管理表及目录树
- 数据库系统：线性表、多链表、及索引树

学习目标

- 掌握数据结构的基本概念、研究对象，对数据结构与算法课程有一个宏观的认识。
- 掌握抽象数据型的概念，包括其定义和实现方法，初步掌握抽象技术方法。
- 掌握算法的概念、算法复杂性和算法性能的评价方法。
- 了解解决问题的一般过程和算法的逐步求精方法，掌握问题求解的基本过程和方法。
- 增强*求解复杂问题*的能力 。

本章主要内容

- 学习数据结构的意义
- 数据结构的基本概念
- 抽象数据类型
- 算法的概念
- 逐步求精的问题求解

1.1 学习数据结构课程的目的

前序课程:

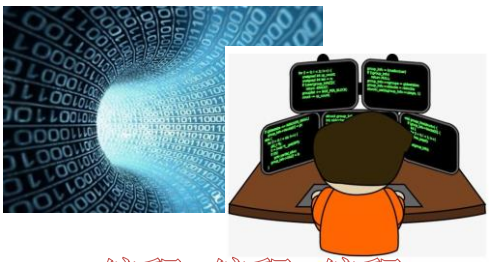
计算机导论

程序设计

数学

计算机原理……

要做一个优秀的程序员！！！



编程+编程+编程

Case 1: 一个简单的例子

$$x^2+4x-8=0$$

$$x1=\frac{-b+\sqrt{b^2-4ac}}{2a}$$

$$x2=\frac{-b-\sqrt{b^2-4ac}}{2a}$$

```
Main()
{ int a,b,c;
  float x1,x2;
  a=1;
  b=4;
  c=-8;
  x1=(-b+sqrt(b*b-4*a*c)/(2*a);
  x2=(-b-sqrt(b*b-4*a*c)/(2*a);
}
```

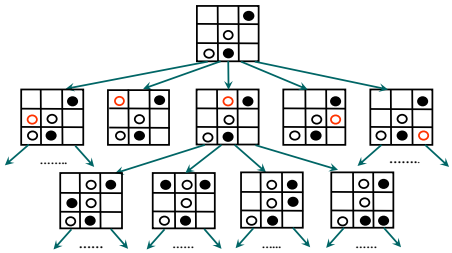
问题：是不是一个好的程序？

Case 2: 表的查找问题

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
…	…	…	…	…

问题：如何存储表？用什么算法实现查找？

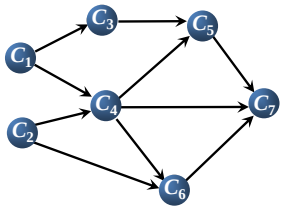
Case 3: 皇后问题



问题：怎样在计算机中描述该问题？
用什么算法实现？

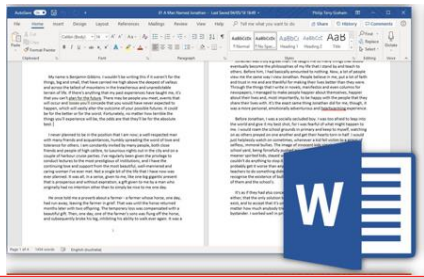
Case 4: 图的应用

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



问题：怎样在计算机中描述该问题？（结点，关系）
用什么算法实现？

Case 5: 文本编辑



问题：怎样在计算机中描述该问题？（文字，符号）
用什么算法实现？

Case 6: 图形图像处理



问题：怎样在计算机中描述该问题？用什么算法实现？

面对的问题:

- 求解问题涉及到数值以及非数值求解
- 数据之间的关系更为复杂，不再是数学方程，而是表、树和图等
- 需要考虑不同算法的选择和实现

1.2 如何处理复杂的问题?

计算机是一门研究用计算机进行信息表示和处理的科学。这里面涉及到两个问题：数据信息的*表示和处理*。

程序设计的本质：

- ✓ **数据表示：**将数据存储在计算机中（存储设计）
- ✓ **数据处理：**处理数据，求解问题（算法设计）

信息的表示和组织又直接关系到处理信息的程序的*效率*。随着应用问题的不断复杂，导致信息量剧增与信息范围的拓宽，使许多系统程序和应用程序的规模很大，结构又相当复杂。因此，必须分析待处理问题中的对象的特征及各对象之间存在的关系，这就是数据结构这门课所要研究的问题。

编写解决问题的程序需要考虑以下几点：

- 如何确定问题所涉及的数据及数据之间的关系;
- 如何在计算机中存储数据及体现数据之间的关系?
- 处理问题时需要对数据作何种运算?
- 所编写的程序的性能是否良好?

上面所列举的问题基本上由数据结构这门课程来回答。

1.3 名词术语

1) **数据:** 数据是指对客观事件进行记录并可以鉴别的符号，是对客观事物的性质、状态以及相互关系等进行记载的物理符号或这些物理符号的组合。它是可识别的、抽象的符号。

数据可以是连续的值，比如声音、图像，称为模拟数据；也可以是离散的，如符号、文字，称为数字数据。

在计算机科学中，数据是指一切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。

3) 数据元素：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。

4) 数据项: 构成数据元素的最小单位。

5) 数据对象: 具有相同性质的数据元素的集合。

数据元素

数据项

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
…	…	…	…	…

方程求解

$$x^2+4x-8=0$$

$$x1=\frac{-b+\sqrt{b^2-4ac}}{2a}$$

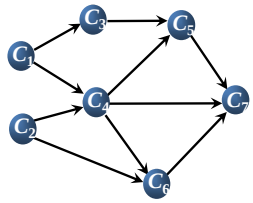
$$x2=\frac{-b-\sqrt{b^2-4ac}}{2a}$$

```
Main()
{ int a,b,c;
  float x1,x2;
  a=1;
  b=4;
  c=-8;
  x1=(-b+sqrt(b*b-4*a*c)/(2*a);
  x2=(-b-sqrt(b*b-4*a*c)/(2*a);
}
```

数据，数据元素（数据项），数据对象
这是一个数值计算问题，特点？

图的应用

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，数据元素（数据项），数据对象
这是一个非数值计算问题，特点？

发现的问题

在计算机求解问题的过程中，数据元素不是孤立存在的，他们之间存在着某种关系，这种关系称之为数据元素之间的逻辑关系。

1.4 逻辑结构

数据的逻辑结构是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，与数据的存储方式也无关。反映的是我们对问题中涉及到的数据元素的解释，是数据元素之间内在的关系表示。

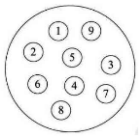
面对需要解决的问题，我们要进行分析，找出数据及其内在的逻辑结构。常用的有四种逻辑结构的表示。

1）集合

问题表现：

- ✓ **1-20以内所有的质数；**
- ✓ **我国近15年发射的所有人造卫星；**
- ✓ **中大2018年所有的新生…**

在数据元素之间“同属一个集合”，别无其他关系。



确定性
互异性
无序性

2）线性结构

问题表现：

- ✓ **排队的人群；**
- ✓ **一句话“我国近15年发射的所有人造卫星”；**
- ✓ **学生名单；**
- ✓ **待处理的事情（一件件的处理）…**

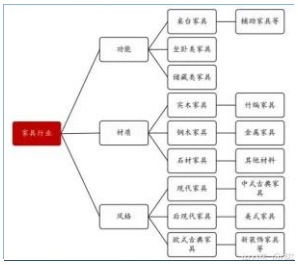
姓名	电话号码
陈海	13612345588
李四锋	13056112345
…	…

数据元素之间存在着一个接一个的线性关系。

3）树结构

问题表现：

- ✓ **公司组织框架；**
- ✓ **家具行业分类；**
- ✓ **企业组织架构图；**
- ✓ **比赛赛程…**

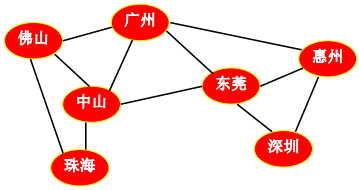


数据元素之间存在 着一对多的层次关系。

4）图结构

问题表现：

- ✓ **公路交通图；**
- ✓ **航线图；**
- ✓ **城市规划图…**



结点之间的关系可以是任意的，图中任意两个数据元素之间都可能相关。**数据元素之间存在着多对多的任意关系。**

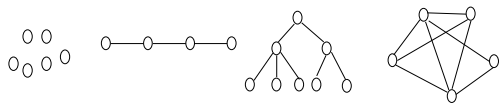
四种逻辑结构的主要特征：

① **集合**：结构中的数据元素除了“同属于一个集合”外，没有其它关系。

② **线性结构**：结构中的数据元素之间存在一对一的关系。

③ **树型结构**：结构中的数据元素之间存在一对多的关系。

④ **图状结构或网状结构**：结构中的数据元素之间存在多对多的关系。

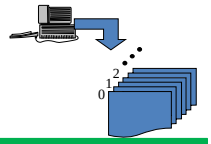


1.5 物理结构

确定了求解问题中数据的逻辑结构后，需要进行物理结构的设计，即如何在计算机中表示数据元素及其内在的逻辑关系，其要点是不仅要保存数据本身的值，还要保存数据元素之间的关系 。

数据元素的值 + 逻辑结构

→存储在计算机中



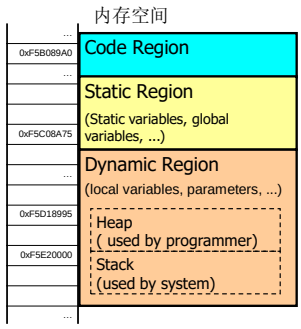
针对求解的问题：

- **在处理数据元素的同时，还要受到其逻辑结构的约束。无论在逻辑结构的层面还是在计算机中物理结构存储下，都需要考虑数据元素及其逻辑关系。**
- **一种数据的逻辑结构可以用多种存储结构来存储，不同的存储结构，数据处理的效率往往是不同的。**

1.5.1 计算机内存示意图

- 内存空间
  - A “continuous” space
  - Three regions
- 内存地址
  - Each memory unit has an address
  - 内存区的每一个字节有一个编号，这就是“地址”。

地址及其对应的单元
如何对地址及其对应的单元进行操作



变量：程序设计中离不开变量，变量是存放数据的“容器”，每一个变量都有一个名字（变量名），该名字指定了程序中使用的一个存储区域，数据称为变量的值。

每个变量都有涉及到以下四个方面的内容：

- ✓ 一个用于存放数据的空间
- ✓ 空间的位置（地址）
- ✓ 一个确定的类型
- ✓ 存放的数据的值

数据通过变量名与内存进行关联

如果在程序中定义了一个变量，在对程序进行编译时，系统就会根据它的类型给这个变量分配内存单元。
该变量就有了一个内存的地址。

int i,j,k　为三个整型变量分配内存单元

float s,t　为二个实型变量分配内存单元

i=50　为这个变量赋值

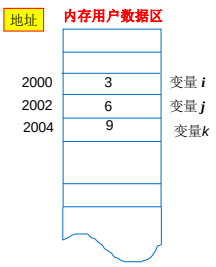
变量的地址可以得到（用取地址运算&）

&i=2000

1.5.2 对变量进行访问和操作的方式：直接访问

变量名表示了内存地址，该地址对应的内存单元存储了变量要表示的数据，不需要关注变量的地址，直接通过变量名进行对内存进行操作的方式称为“直接访问”方式。

```
int i,j,k;
i=3;
j=6;
k = i + j ;
```



直接从 *i* 和 *j* 对应的地址2000和2002取出内容3和6然后相加将结果9存入变量 *k* 对应的地址2004的单元中。

不需要知道具体的地址信息

1.5.3 对变量进行访问和操作的方式：间接访问

指针变量

内存的地址的值是无符号整型，因此也可以放在变量中。**存放内存地址的变量称为指针变量。**

与普通变量一样，在使用指针变量前，也需要进行定义，做如下几方面的说明：

① 指针变量的名 （*p）

② 指针变量的类型：指针所指向地址单元中数值的类型 （int *p）（float *p）

③ 初值

Float f=88.88;

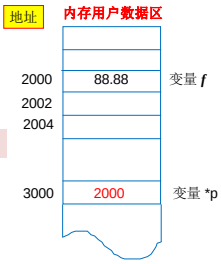
Float *p;

p=&f;

指针不能指向类型不符的变量

int a;
Float *p;

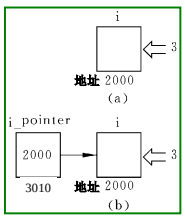
p=&a;

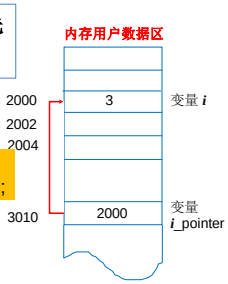


1.5.3 对变量进行访问和操作的方式：间接访问

通过指针变量对变量所在的存储单元进行访问的方式称为“间接访问”。

int i=3;
int *i_point=&i;





访问地址为2000的单元，可以通过变量名 *i* 或在变量名*i_point* 中存放的地址来访问。

指针变量的运算

1. &: 取地址运算符, 取右边变量的地址

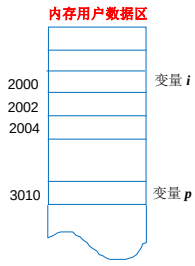
   &a是变量a的地址

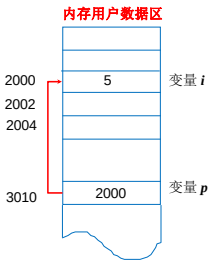
2. * : 指向运算符(间接访问运算符),
   访问指针变量右边所指向的变量。

   *p 表示的是指针变量p指向的变量。

通过指向变量*i*的地址的指针变量 *p* 进行访问

```
int i, *p;
p=&i;
*p=5;
```





i=10;
*p=?


```
void main()
{  int *i_pointer;      //声明int型指针变量i_pointer
   int i;//声明int型数变量i
   i_pointer=&i;        //取i的地址赋给i_pointer
   i=10;          //int型数赋初值
   cout<<"Output int i="<< i <<endl; //输出int型数的值
   cout<<"Output int pointer i="<<*i_pointer<<endl;
                  //输出int型指针所指地址的内容
}
```

程序运行的结果是：
Output int i=10
Output int pointer i=10

- 因为指针也是个变量，程序中将哪个变量的地址赋给指针，指针就指向哪个变量。例如：

```
int a, b;
int *p1;
p1=&a;          //p1指向变量a
*p1=100;        //将100存入a
p1=&b;          //p1指向变量 b
*p1=200;        //将200存入b
```

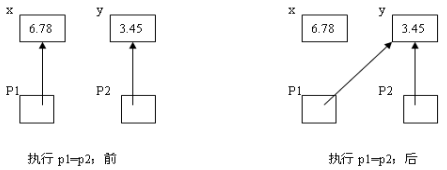
- 注意：指针只能接受地址量。如果将一个任意常数赋予指针，会产生语法错误。

```
int i=300;
int *p;
  p=300;   //错误，编译器不会将一般整数常量理解为地址。
  p=i;     //错误
```


- 同类型的指针之间可以互相赋值，例如：

```
double x=6.78, y=3.45;
double *p1=&x, *p2=&y;
p1=p2;              //执行后p1和p2都指向变量 y 。
```

执行p1=p2前后见示意图如下:



注意

(1) 指针变量定义和引用指向变量的“*”含义有差别。

(2) 不能引用没有赋值的指针变量,不要误认为p定义后变量*p就已存在,必须给p正确赋值后,变量*p才存在。

(3) p=&a;是给指针变量p 赋值, *p=3; 是给p指向的变量赋值。两者含义完全不同。

(4) 给指针变量赋值必须用同类型的指针。

(5) 指针变量只存放地址, 地址值是无符号整数, 但不能直接用整型量(或其它非地址量)赋值给指针变量。

通过指针变量访问整型变量

指向哪里？

```
#include <stdio.h>
void  main ( )
{ int a , b ;
  int *pointer_1 , *pointer_2 ;
  a = 1 0 0 ; b = 1 0 ;
  pointer_1 = & a ;  /*把变量 a 的地址赋 pointer_1 */
  pointer_2 = & b ;  /*把变量 b 的地址赋给 pointer_2 */

  printf ("%d , %d\n",a,b) ;
  printf ("%d , %d\n",*pointer_1,*pointer_2) ;
 }
```

为a 和b 分配了内存单元

为两个指针变量分配了内存单元

输入a和b两个参数，按先大后小的顺序输出a和b。

```
#include <stdio.h>
void  main ( )
{ int *p1, *p2, *p, a, b;
  scanf ("%d , %d", &a , &b ) ;
  p1=&a ; p2=&b ;
  if ( a < b )
  { p = p1 ; p1 = p2 ; p2 = p ; }
  printf ("a =%d,b =%d\n\n",a,b ) ;
  printf ("max=%d,min=%d\n",*p1,*p2) ;
}
```

运行情况如下：

```
5, 9↙
a = 5 , b = 9
m a x = 9 , m i n = 5
```


Examples

```
void main()
{
    int a =10, b = 20;
    int *pa, *pb;
    pa = &a;
    pb = pa+2;

    cout <<"a in: "<< &a <<endl;
    cout <<"b in: "<< &b <<endl;

    cout <<"pa is: "<< pa <<endl;
    cout <<"pa points to: "<< *pa <<endl;
    cout <<"pb is: "<< pb <<endl;
    cout <<"pb points to: "<< *pb <<endl;
    if(pa != pb)
        cout <<"pa and pb are not equal!"<<endl;
    cout<<"pb - pa is "<<pb-pa<<endl;
}
```

```
a in: 0012FF34
b in: 0012FF2C
pa is: 0012FF34
pa points to: 10
pb is: 0012FF3C
pb points to: 4227998
pa and pb are not equal!
pb - pa is 2
```

取地址运算符&和指向运算符*的应用

```
main( )
{ int m, n;
  int *p=&m,*q=&n;
  printf("Input m,n:");
  scanf("%d %d",p,&n);
  printf("m=%d  &m=%X\n",m,&m);
  printf("*p=%d  p=%X\n",*p,p);
  printf("n=%d  &n=%X\n",n,&n);
  printf("*q=%d  q=%X\n",*q,q);
}
```

运行结果:

```
Input m,n: 123  456↵
m=123     &m=FFD6
*p=123    p=FFD6
n=456     &n=FFD8
*q=456    q= FFD8
```


指向结构体

```
struct Student{
  int std_no;
  char name[10];
  int sex;
  int age;
  char dept[10];
};
struct Student *p;
```

```
struct Student{
  int std_no;
  char name[10];
  int sex;
  int age;
  char dept[10];
  struct Student *next;
};
struct Student *p;
```

p → □　　p → □ →

1.5.4 两种物理结构的设计方式

（1）连续设计（顺序映像）

- ✓ 一系列地址连续的存储单元
- ✓ 通过数据元素在存储单元中的相对位置来表示 数据元素之间的逻辑关系

本课程中用一维数组来描述连续设计方式

（2） 链接设计

- ✓ 任意存储单元
- ✓ 借助指示数据元素存储地址的指针来表示数据元素之间的逻辑关系，即在每一个数据元素中增加一个存放另一个元素地址的指针(pointer)，用该指针来表示数据元素之间的逻辑结构(关系)。

计算机中的实现方式：指针类型

逻辑结构和物理结构之间的关系

- ➢ 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，数据的逻辑结构独立于计算机；数据的存储结构属于具体实现的视图，依赖于具体的计算机语言的，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。

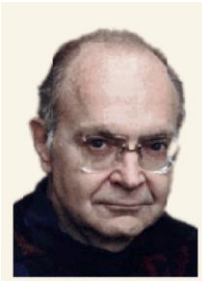
1.6 数据结构

数据结构问题起源于程序设计，随着程序设计的发展而发展。

- ➢ 无结构阶段：在简单数据上作复杂运算
- ➢ 结构化阶段：数据结构＋算法＝程序
- ➢ 面向对象阶段：（对象＋行为）＝程序

数据结构的创始人：Donald. E. Knuth

唐纳德·克努斯



1938年出生，25岁毕业于加州理工学院数学系，博士毕业后留校任教，28岁任副教授。30岁时，加盟斯坦福大学计算机系，任教授。从31岁起，开始出版他的历史性经典巨著：

The Art of Computer Programming

他计划共写7卷，然而出版三卷之后，已震惊世界，使他获得计算机科学界的最高荣誉图灵奖，年仅36岁。

数据结构与算法是研究计算机求解问题过程中数据的逻辑结构、存储结构以及相关算法的学科。

数据结构的三个组成部分：

逻辑结构： 数据元素之间逻辑关系的描述

D_S=（D，S）

存储结构： 数据元素在计算机中的存储及其逻辑关系的表现称为数据的存储结构或物理结构。

数据操作： 对数据要进行的运算。

数据结构的形式定义是一个二元组：

Data-Structure=(D，S)

其中：D是数据元素的有限集，S是D上关系的有限集。

例2：设数据逻辑结构B=（K，R）

K={k_1, k_2, …, k_9}

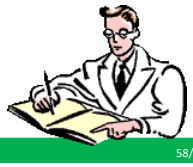
R={ <k_1, k_3>，<k_1, k_8>，<k_2, k_3>，<k_2, k_4>，<k_2, k_5>，<k_3, k_9>，<k_5, k_6>，<k_8, k_9>，<k_9, k_7>，<k_4, k_7>，<k_4, k_6> }

数据结构的三个主要环节

Abstract: To analysis the requirement of problem and data's logical structure;

Design: To design the data's physical structure and algorithms, to analysis the algorithm's time complexity;

Implementation: to practice the algorithm in computer language C++.



数据结构的运算

(1) 建立(Create)一个数据结构；
(2) 消除(Destroy)一个数据结构；
(3) 从一个数据结构中删除(Delete)一个数据元素；
(4) 把一个数据元素插入(Insert)到一个数据结构中；
(5) 对一个数据结构进行访问(Access)；
(6) 对一个数据结构(中的数据元素)进行修改(Modify)；
(7) 对一个数据结构进行排序(Sort)；
(8) 对一个数据结构进行查找(Search)。

1.7 数据类型与抽象数据类型

数据类型: 数据的取值（范围）+一组操作。

例如： short int 取值范围：-32768~32767

一组操作：加，减，乘，除，开方，.....

一个简单的例子

$$x^2+4x-8=0$$

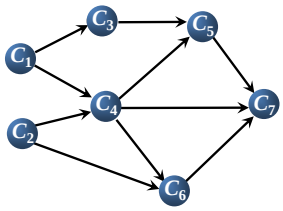
$$x1=\frac{-b+\sqrt{b^2-4ac}}{2a}$$

$$x2=\frac{-b-\sqrt{b^2-4ac}}{2a}$$

```
Main()
{ int a,b,c;
  float x1,x2;
  a=1;
  b=4;
  c=-8;
  x1=(-b+sqrt(b*b-4*a*c)/(2*a);
  x2=(-b-sqrt(b*b-4*a*c)/(2*a);
}
```


图的应用

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



- **ADT（Abstract Data Type）：为求解问题而定义的数据类型。**

ADT的形式化定义是三元组：ADT=(D，S，P)
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的定义仅是一组逻辑特性描述， 与其在计算机内的表示和实现无关。因此，不论ADT的内部结构如何变化，只要其数学特性不变，都不影响其外部使用。

ADT的一般定义形式是：
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>

– 其中数据对象和数据关系的定义用伪码描述。
– 基本操作的定义是：
<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，返回相应的出错信息。

– 操作结果：描述操作正常完成之后，数据结构的变化状况和 应返回的结果。

其他的说明方法

```
Operation description:
  Name (parameters list)
    Input:     data used to input;
    Output:    data used to output;
    Pre-condition: if the condition can not be
                   satisfied, the operation
                   may be not correct.
    Post-condition: The status after the
                   operation be executed 。
```

Case 1 Big integer

ADT Bigint {
DATA
n: 0..2^{512}-1;
Operations
Addone
Pre-condition： n+1<2^{512};
Post-condition： n=n+1;

Subone
Pre-condition： n>0;
Post-condition： n=n-1;

Mult(x,y)
Input：(x:Bigint)；
Output: (y: Bigint)；
Pre-condition： n*x.n<2^{512};
Post-condition： y.n=n*x.n;

} ADT Bigint

抽象数据类型是为求解问题而定义的数据类型。

数据类型可以看作是已经实现了的抽象数据类型。

1.8 算法

算法(Algorithm)：是对特定问题求解方法(步骤)的一种描述，是指令的有限序列，其中每一条指令表示一个或多个操作。

算法的五个重要特性

确定性、能行性、输入、输出、有穷性/有限性

1）确定性

算法每种运算必须有确切定义，不能有二义性。

例：不符合确定性的运算：

①5/0

②将6或7与x相加

③未赋值变量参与运算

2）能行性

算法中有待实现的运算都是基本的运算，可以通过已经实现的基本操作执行有限次来实现。

3）输入

每个算法有0个或多个输入。这些输入是在算法开始之前给出的量，取自于特定的对象集合——定义域。

4）输出

一个算法产生一个或多个输出，这些输出是同输入有某种特定关系的量。

5）有穷性/有限性

一个算法总是在执行了有穷步的运算之后终止。

计算过程：只满足确定性、能行性、输入、输出四个特性但不一定能终止的一组规则。

准确理解算法和计算过程的区别：

- 不能终止的计算过程：操作系统
- 算法是“可以终止的计算过程”。

1.8.2 算法的描述方法

1) *自然语言*

优点：容易理解

缺点：冗长、二义性

使用方法：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—自然语言描述算法

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。

2) *流程图*

优点：处理流程直观

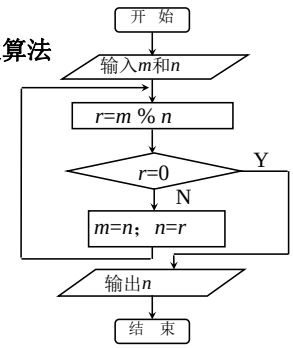
缺点：缺少严密性、灵活性

使用方法：描述简单算法

注意事项：注意抽象层次

例：流程图描述算法

流程图



3) *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。

例：欧几里德算法

伪代码

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

上述伪代码再具体一些，用C++的函数来描述。

4) *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。

使用方法：算法需要验证

注意事项：将算法写成子函数

例：欧几里德算法—程序设计语言描述算法

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;
}
```


1.8.3 评价算法的标准

① 正确性(Correctness)：算法应满足具体问题的需求。

② 可读性(Readability)：算法应容易供人阅读和交流。可读性好的算法有助于对算法的理解和修改。

③ 健壮性(Robustness)：算法应具有容错处理。当输入非法或错误数据时，算法应能适当地作出反应或进行处理，而不会产生莫名其妙的输出结果。

④ 通用性(Generality)：算法应具有一般性，即算法的处理结果对于一般的数据集合都成立。

⑤ 效率与存储量需求：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。

1.8.4 算法分析

对算法所需要的计算机资源----时间和空间进行估算。

时间复杂度（Time Complexity）

空间复杂度（Space Complexity）

与此相关的因素有：

- 依据算法选用何种策略；
- 问题的规模；
- 程序设计的语言；
- 编译程序所产生的机器代码的质量；
- 机器执行指令的速度；

撇开软硬件等有关因素，可以认为一个特定算法“运行工作量”的大小，只依赖于问题的规模（通常用n表示），或者说，它是问题规模的函数。

- 评价算法效率的方法
  - 事后统计：将算法实现，测算其时间和空间开销。
  - 优点：准确的实测值
  - 缺点：
    - 编写程序实现算法将花费较多的时间和精力；
    - 所得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。

- 算法分析----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

单位时间

执行次数×执行一次的时间

指令系统、编译的代码质量

- Sum 1+2+…+100?

第一种算法：

```
int i, sum = 0,n = 100;          /* 执行 1 次 */
for(i = 1; i < = n; i++)         /* 执行了 n+1 次 */
{
    sum = sum + i;               /* 执行 n 次 */
}
printf("%d", sum);               /* 执行 1 次 */
```

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

第二种算法：

```
int sum = 0,n = 100;             /* 执行一次 */
sum = (1 + n) * n/2;             /* 执行一次 */
printf("%d", sum);               /* 执行一次 */
```

1+1+1 = 3

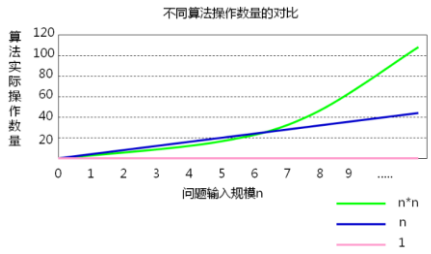
- 第三种算法:

```
int i, j, x = 0,sum = 0,n = 100;    /* 执行一次 */
for(i = 1; i < = n; i++)
{
    for (j = 1; j < = n; j++)
    {
        x++;                        /* 执行 n×n 次 */
        sum = sum + x;
    }
}
printf("%d", sum);                  /* 执行一次 */
```

n^2 iterations

算法的执行时间随着n的增加也将远远多于前面2个算法。

- 由前述例子可知，同样的输入规模是 n：
- 第一种方法运行代码n次，则操作数量$f(n)=n$，显然运行100次是运行10次的10倍时间
- 第二种方法则无论n为多少，运行次数都为1，即$f(n)=1$
- 第三种方法，由于$f(n)=n^2$，即运算100次是运算10次的100倍。



➢ 算法时间复杂度分析

- 函数的渐进增长
- 算法A做$2n$+3次操作，算法B做$3n$+1次操作，谁更快？

次数	算法A (2n+3)	算法A' (2n)	算法B (3n+1)	算法B' (3n)
N=1	5	2	4	3
N=2	7	4	7	6
N=3	9	6	10	9
N=10	23	20	31	30
N=100	203	200	301	300

- 在输入规模n没有限制的情况下，只要超过一个数值N，这个函数就总是大于另一个函数，则称函数是渐进增长的。
- 注：忽略加法常数，不影响算法变化。

- 算法C是$4n$+8，算法D是$2n^2$+1

次数	算法C (4n+8)	算法C' (n)	算法D ($2n^2+1$)	算法D' (n^2)
N=1	12	1	3	1
N=2	16	2	9	4
N=3	20	3	19	9
N=10	48	10	201	100
N=100	408	100	20 001	10 000
N=1000	4 008	1 000	2 000 001	1 000 000

- 注：与最高次项相乘的常数并不重要。

- 算法E是$2n^2+3n+1$，算法F是$2n^3+3n+1$

次数	算法E ($2n^2+3n+1$)	算法E' (n^2)	算法F ($2n^3+3n+1$)	算法F' (n^3)
N=1	6	1	6	1
N=2	15	4	23	8
N=3	28	9	64	27
N=10	231	100	2 031	1 000
N=100	20 301	10 000	2 000 301	1 000 000

- 注：最高次项的指数大的，函数随着n的增长，结果也会变得增长特别快。

- 算法G是$2n^2$，算法H是$3n+1$，算法I是$2n^2+3n+1$

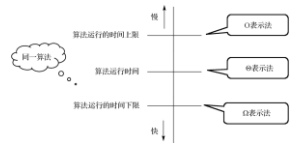
次数	算法G ($2n^2$)	算法H ($3n+1$)	算法I ($2n^2+3n+1$)
N=1	2	4	6
N=2	8	7	15
N=5	50	16	66
N=10	200	31	231
N=100	20 000	301	20 301
N=1,000	2 000 000	3 001	2 003 001
N=10,000	200 000 000	30 001	200 030 001
N=100,000	20 000 000 000	300 001	20 000 300 001
N=1,000,000	2 000 000 000 000	3 000 001	2 000 003 000 001

- 结论：判断一个算法的效率时，函数中的常数和其他次要项常常可以忽略，而更应该关注主项（最高阶项）的阶数。

算法复杂度的记法

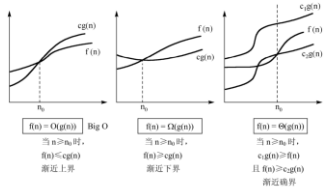
- 定义：
  - 在进行算法分析时，语句总的执行次数$T(n)$是关于问题规模n的函数，进而分析$T(n)$随n的变化情况，并确定$T(n)$的数量级。
  - 算法的时间复杂度，记作：$T(n)=O(f(n))$。表示随问题规模n的增大，算法执行时间的增长率和$f(n)$的增长率相同，称作算法的渐近时间复杂度，简称为时间复杂度。
  - 其中$f(n)$是问题规模n的某个函数。
- 用大写$O()$来体现算法时间复杂度的记法，称之为大O记法。
- $O(1)$为常数阶，$O(n)$为线性阶，$O(n^2)$为平方阶。

- 一个算法往往有最好情况、平均情况、最坏情况三种情形，通常最坏情况的效率是我们最关注的。
  - 最差效率是指在输入规模为n时，算法在最坏情况下的效率。
  - 最好效率是指在输入规模为n时，算法在最优情况下的效率。
  - 平均效率是指在“典型”或“随机”输入的情况下，算法具有的效率。
- 算法分析是为了得到近似的执行时间，一般考察的是当问题规模n增加时，运算所需时间频度$T(n)$的上界和下界，如图所示。
- 其中的O、Ω、Θ表示法，分别表示了算法的效率上限(上界)、下限(下界)和等限(确界)。



- 其中的O、Ω、Θ表示法，分别表示了算法的效率上限（上界）、下限（下界）和等限（确界），其数学上的具体定义见表，对应曲线如图所示。

记号	定义	含义
O	定义：f(n)=O(g(n)) 若存在两个正常数 c 和 n_0，使得当 $n \geq n_0$ 时，$f(n) \leq c*g(n)$	f(n)的渐近上限为 g(n)
Ω	定义：f(n)=Ω(g(n)) 若存在两个正常数 c 和 n_0，使得当 $n \geq n_0$ 时，$f(n) \geq c*g(n)$	f(n)的渐近下限为 g(n)
Θ	定义：f(n)=Θ(g(n)) 若存在正常数 c_1、c_2 和 n_0，使得当 $n \geq n_0$ 时，$c_1*g(n) \leq f(n) \leq c_2*g(n)$	f(n)的渐近确界为 g(n)



注：n_0、c、c_1、c_2均为正常数

计算算法的时间复杂度

✓ 首先求出程序中各语句、各模块的运行时间。

✓ 再求整个程序的运行时间。

各种语句和模块分析应遵循的规则

(1) 赋值语句或读/写语句

运行时间通常取$O(1)$。有函数调用的除外，此时要考虑函数的执行时间。

(2) 语句序列

运行时间由加法规则确定，即该序列中耗时最多的语句的运行时间。

(3) 分支语句

运行时间由条件测试（通常为$O(1)$）加上分支中运行时间最长的语句的运行时间。

(4)循环语句

➢运行时间是对输入数据重复执行n次循环体所耗时间的总和。

➢每次重复所耗时间包括两部分：一是循环体本身的运行时间；二是计算循环参数、测试循环终止条件和跳回循环头所耗时间。后一部分通常为$O(1)$。

➢当遇到多重循环时，要由内层循环向外层逐层分析。因此，当分析外层循环的运行时间是，内层循环的运行时间应该是已知的此时可以把内层循环看成是外层循环的循环体的一部分。

(5) 函数调用语句

➢若程序中只有非递归调用，则从没有函数调用的被调函数开始，计算所有这种函数的运行时间。然后考虑有函数调用的任意一个函数P，在P调用的全部函数的运行时间都计算完之后，即可开始计算P的运行时间。

➢若程序中有递归调用，则令每个递归函数对应于一个未知的时间开销函数$T(n)$，其中n是该函数参数的大小之后列出关于T的递归方程并求解之。

时间复杂度的分析方法

✓首先求出程序中各语句、各模块的运行时间。

✓再求整个程序的运行时间。

法则：

$O(f)+O(g)=O(\max(f,g))$

$O(f)O(g)=O(f*g)$

$O(cf(n))=O(f(n))$

算法的时间复杂度示例

- 常数阶

```
int sum = 0,n = 100;      /* 执行一次 */
sum = (1+n)*n/2;          /* 执行一次 */
printf("%d", sum);        /* 执行一次 */
```

3次

```
int sum = 0, n = 100;     /* 执行1次 */
sum = (1+n)*n/2;          /* 执行第1次 */
sum = (1+n)*n/2;          /* 执行第2次 */
sum = (1+n)*n/2;          /* 执行第3次 */
sum = (1+n)*n/2;          /* 执行第4次 */
sum = (1+n)*n/2;          /* 执行第5次 */
sum = (1+n)*n/2;          /* 执行第6次 */
sum = (1+n)*n/2;          /* 执行第7次 */
sum = (1+n)*n/2;          /* 执行第8次 */
sum = (1+n)*n/2;          /* 执行第9次 */
sum = (1+n)*n/2;          /* 执行第10次 */
printf("%d",sum);         /* 执行1次 */
```

12次

只有执行次数的差异，跟问题规模n的取值无关，所以为$O(1)$时间复杂度。

- 线性阶

```
int i;
for(i = 0; i < n; i++)
{
   /* 时间复杂度为O(1)的程序步骤序列 */
}
```

循环n次，所以为$O(n)$时间复杂度。

- 对数阶

```
int count = 1;
while (count < n)
{
   count = count * 2;
   /* 时间复杂度为O(1)的程序步骤序列 */
}
```

由$2^x=n$，得$x=\log_2 n$，所以为$O(\log n)$时间复杂度。

- 平方阶

```
int i,j;
for(i = 0; i < n; i++)
{
   for (j = 0; j < n; j++)
   {
      /* 时间复杂度为O(1)的程序步骤序列 */
   }
}
```

内循环n次，再进行外循环n次，所以为$O(n^2)$时间复杂度。

```
int i,j;
for(i = 0; i < m; i++)
{
   for (j = 0; j < n; j++)
   {
      /* 时间复杂度为O(1)的程序步骤序列 */
   }
}
```

内循环n次，外循环m次，所以为$O(mn)$时间复杂度。

- 立方阶

两个n阶方阵的乘法

```
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] ; }
```

由于是一个三重循环，每个循环从1到n，则总次数为：$n \times n \times n = n^3$　时间复杂度为$T(n)=O(n^3)$

●求n!的递归算法

```
long fact ( int n)
{  if ( n==0 ) || ( n ==1 )
       return( 1 );
   else
       return( n * fact(n – 1));
}
```

●时间复杂性的递归方程

$$T(n)=\begin{cases} C & \text{当 } n=0,\ n=1 \\ G+T(n-1) & \text{当 } n>1 \end{cases}$$

●解递归方程：

$$\begin{aligned} T(n) &= G+T(n-1) \\ T(n-1) &= G+T(n-2) \\ T(n-2) &= G+T(n-3) \\ &\cdots\cdots \\ T(2) &= G+T(1) \\ +\ T(1) &= C \\ \hline T(n) &= G\ (n\text{-}1)\ +C \end{aligned}$$

取 $f(n)=n$

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

$=O(n)$

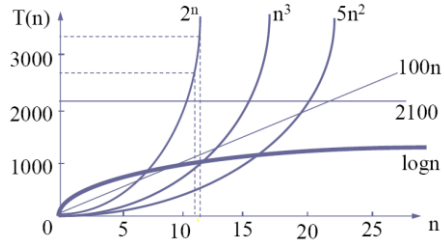
常见的算法时间复杂度

执行次数函数举例	阶	非正式术语	常见的算法
12	$O(1)$	常数阶	在已排序的数组中寻找中值
$2n+3$	$O(n)$	线性阶	在未排序的数组中寻找最小值、最大值
$3n^2+2n+1$	$O(n^2)$	平方阶	冒泡排序、插入排序
$5\log_2 n+20$	$O(\log n)$	对数阶	二叉树查找
$2n+3n\log_2 n+19$	$O(n\log n)$	$n\log n$阶	基于比较的排序算法上限、快速傅里叶变换
$6n^3+2n^2+3n+4$	$O(n^3)$	立方阶	普通的矩阵相乘算法
2^n	$O(2^n)$	指数阶	寻找所有子集

常用的复杂度：

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

$O(2^n) < O(n!) < O(n^n)$



重要的概念

① 算法分析：计数的对象是什么？

② 大O记法：关注了什么？忽略了什么？

③ 算法时间复杂度：常见的形式；大小关系

④ 算法空间复杂度

➢算法的空间复杂度

➢一个算法的存储量需求除了存放算法本身所有的指令、常数、变量和输入数据外，还包括对数据进行操作的工作单元和存储实现计算所需信息的辅助空间。

➢算法的存储量需求与输入的规模、表示方式、算法采用的数据结构、算法的设计以及输入数据的性质有关。

➢算法的执行的不同时刻，其空间需求可能是不同的。

➢算法的空间复杂度是指算法在执行过程中的最大存储量需求。

时间、空间的权衡

- 数据结构
  - ✓ 一定的空间来存储它的每一个数据项
  - ✓ 一定的时间来执行单个基本操作
- 代价和效益
  - ✓ 空间和时间的限制
  - ✓ 程序设计工作量

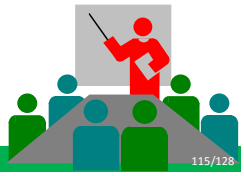
1.9 C++ for course practice

- C++ has emerged as the leading systems programming language. In addition to fixing many of the syntactic flaws of C, C++ ***provides direct constructs (the class and template) to implement generic data structures as abstract data types.***

1.10 问题求解示例：逐步求精的设计过程

- 1.模型化（建模）
  - 对实际问题进行分析，选择适当的模型来描述问题，即建模；
- 2.确定算法
  - 根据模型，找出解决问题的方法，即算法；
- 3.逐步求精
  - 对用自然语言等描述的算法逐步细致化、精确化和形式化，这一阶段可能包括多步求精。
  - 当逐步求精到某一步时，根据程序中所使用的数据形式，定义若干ADT，并且用ADT中的操作代替对应的非形式语句。
- 4.ADT的实现
  - 对每个ADT，选择适当的数据结构表示数学模型，并用相应的函数实现每个操作。

例题：设学校有足够的课室供学生考试使用。学校开设的课程名单和学生选读课程的名单都已经分别存放在名为csfile和scsfile的文件中。规定每一个学生同一天至多参加一门课程的考试，希望编排一个考试时间表，尽量缩短考试的周期。编写程序，完成考试的分组情况。（哪一科目在哪一天、哪个课室考试由人工安排）。



问题分析

Input data：

csfile 文件中一个数据表示一门功课，每个数据由五个字符构成，前两位是字母，第三位是数字 1..4，第四位是数据0或1，第五位是数字0..9。 文件中任何数据都不相同，数据之间以空白字符分隔。

scsfile文件中每个记录表示一个学生选学的课程，每个记录由一个学号和所选的课程名构成，中间以空白分隔。

Samples(it is important!)

Data in csfile

cs301 cs401 cs110 ma411 cs120

Data in scsfile

31027 cs401 ma411 cs110
43816 ma411 cs120 xs301
17390 cs401 cs301

Output：

将可以同时进行考试的科目分组输出，使得同一组里的任何两门功课都没有同一位学生同时选读。每组之间以一行“*“分隔。

```
cs401 cs301
**********
cs110 cs120
**********
ma411
```


抽象设计过程

如何在计算机中表示问题：

有多少课程？**单纯的课程信息。**

学生是怎么选课的？**由于学生选课，使得课程之间建立了是否能安排在同一天进行考试的关系。即一个学生所选的全部课，是不能安排在一天进行考试的。**

问题求解：编排一个考试时间表，即建立一个结点为课程的无向图，被一个学生同时选过的课之间有边的关系。这个问题的求解就是图的染色问题。

第一，对于课程文件中开设的课程：

课程名是需要处理的数据，课程名之间没有内在联系，可以考虑是集合类型。

因此，为课程名数据建立一个抽象数据类型：集合

集合上的基本操作：

置空，判空，插入，删除，按位置读元素，求元素数目。

ADT Csset {

数据对象：

$D=\{a_i,a_i \in \text{cstype}, i=1,2,\ldots,n,n\geq 0\}$

基本操作：

Makenull

操作结果：构造一个空的课程集合V；

Insert(V，cs)

输入：cs是cstype类型的课程名；

操作结果：将cs加入到集合V的最后位置上。

Delete(V，cs)

输入： cs是cstype类型的课程名；

初始条件：cs是 V中的一门课程；

操作结果：从V中删除 cs 这门课 。

Retrieve(V，p，cs)

输入：p为整数，表示集合V中元素的位置；

输出： cs是cstype类型的课程名；

初始条件：P是集合V 中一个有效位置；

操作结果：返回位置P所对应的课程名。

……..

} ADT Csset

通过读取课程csfile 文件来存储所开始的全部课程。

第二，两门课程同时被一位学生选课，则该两门课程之间有逻辑关系。建立一个无向图的抽象数据类型，表示学生选课后课程之间的逻辑关系。

课程集合：{ cs101,cs102,cs103,cs104,cs105 }

选课文件：

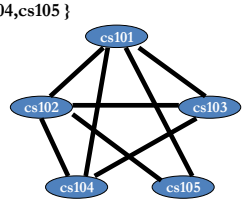
00001 cs101 cs102 cs103

00002 cs102 cs103 cs104

00003 cs101 cs102 cs105

00004 cs101 cs102 cs104

00005 cs101 cs105



ADT Graph{

数据对象：

Csset V; \顶点集合

Edgeset E{(v,w), v,w ∈ V}；

\边的集合，表示从v到w的边

基本操作：

Create_Graph(G);

初始条件： V为课程集合；

操作结果： E为空。

Addedge(G,u,v)

输入：u,v是cstype;

初始条件：(u,v) ∉ E，且u≠v， u，v ∈V；

操作结果：E中添加了边(u,v)。

Isedge(u,v)

输入：u, v 是 cstype 类型；

初始条件：u，v ∈V

操作结果：若边（u, v） ∈ E 则返回真，否则返回假。

} ADT Graph

通过读取课程csfile 文件和scsfile文件建立无向图。

算法设计：通过文件csfile和scsfile构造图。

输入：文件 csfile和scsfile;
输出：图G（V,E);
算法：

```
    G.V为空；
    G.E为空；
    for csfile 中每一门功课v
      将v加入到G.V中；（建立图的顶点）
    for scsfile中每个学生std
      设std所选的课程集合为S；
      for S中每两门不同的课程u,v
        if (u,v) 不在G.E中，将(u,v)加入G.E中；
end
```


本章小结

- 数据结构基本概念（名词术语）
- ADT的表示方式
- 算法的概念、性能分析和评价方法
- 算法的几种描述方法