Chapter 1 Introduction

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。

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. Why to study Data Structure?

Pre-courses:

Introduction of Computer

Programming Language

Mathematical course

Principle of Computer

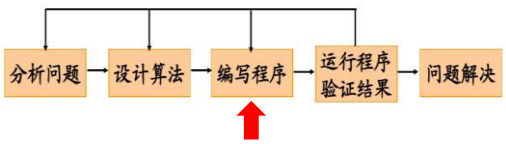
……

用计算机求解问题

计算机科学是研究信息表示和信息处理的科学。

信息在计算机内是用数据表示的。

用计算机解决实际问题的实质可以用下图表示：



程序设计的实质是什么?

✓ **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）

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：信息的*表示*，信息的*处理*。

Case 1: Simple case

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

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

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

```
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);
}
```

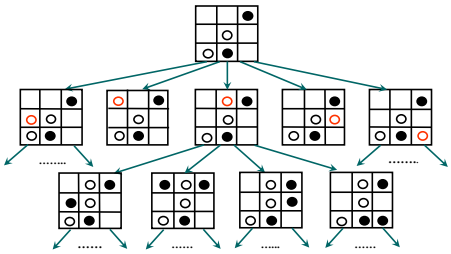
Analysis……(Correct? Efficient?)

Case 2: List(Search)

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
…	…	…	…	…

Analysis……(Correct? Efficient?)

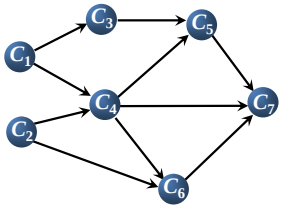
Case 3: Queens Puzzle



Analysis……(Correct? Efficient?)

Case 4: Graph

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



Analysis……(Correct? Efficient?)

Characteristic of computer application:

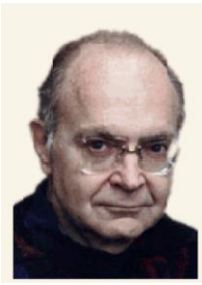
- **Data with complex relationship**
- **Numerical and nonnumeric**

- 数据元素之间的关系
- 数据元素的值

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



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。

编写解决实际问题的程序的一般过程：

- 如何用数据形式描述问题?—即由问题抽象出一个适当的数学模型;
- 问题所涉及的数据量大小及数据之间的关系;
- 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。

Course purposes:

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++.

数据结构与算法是研究数值与非数值计算问题中计算机的操作对象以及它们之间的关系和操作的学科

2. Definitions/terms

1) **Data :** to be used to describe the object.

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。（数值数据，非数值数据）

2) **Data type:** Value range + Operations.

Case 1:

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

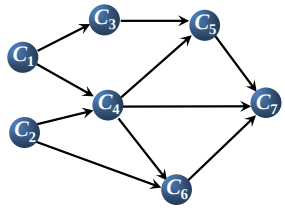
$$x1=\frac{-b+\sqrt{b^2-4ac}}{2b}$$

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

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

Case 2: Graph

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



3) **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.

4) **Data item**: 构成数据元素的最小单位。

5) **Data object**: 具有相同性质的数据元素的集合。

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

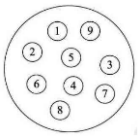
6) Logical structure

- The inherent relationship between data in the problem.
- 现实中的数据元素之间表现出来的依赖关系。用数学模型描述这种依赖关系，并通过研究模型的性质了解他们的变化规律。

(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；
- ✓ 方程$x^2+3x+2=0$的所有实数解；
- ✓ 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。

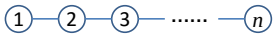


确定性
互异性
无序性

(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）…

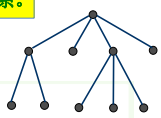
数据元素之间存在着一个接一个的线性关系。

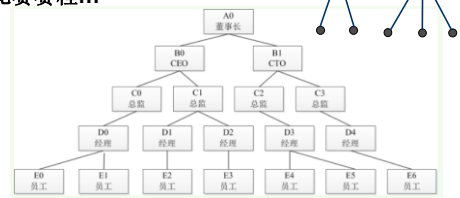


(3) *树结构*

数据元素之间存在着一对多的层次关系。

问题表现：
- ✓ 公司组织框架；
- ✓ 家谱；
- ✓ 比赛赛程…





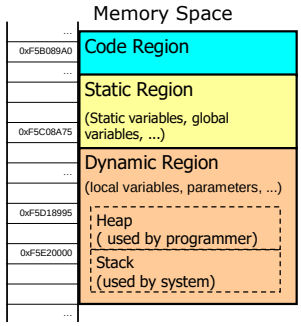
(4) *图结构*：数据元素之间存在着多对多的任意关系。

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。

7) Memory Space

- Memory space
  - A "continuous" space
  - Three regions
- Memory address
  - 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"。



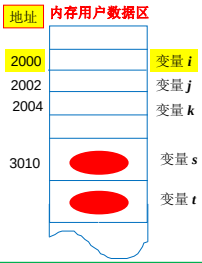
如果在程序中定义了一个变量，在对程序进行编译时，系统就会根据它的类型给这个变量分配内存单元。

内存单元的地址即为变量的地址。

内存单元的内容就是变量的值。

int i,j,k 为这三个变量分配内存单元

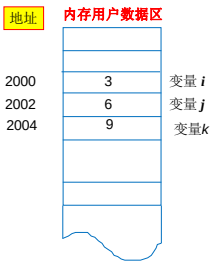
float s,t



接下来可以对这些变量进行访问和操作。

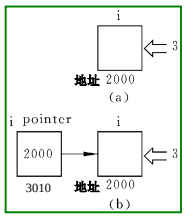
按变量地址存取变量值的方式称为"直接访问"方式，变量名表示了地址

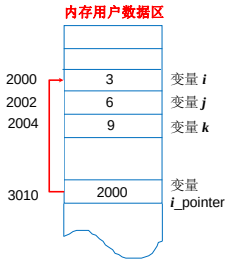
```
int i,j,k;
i=3;
j=6;
k = i + j ;
```



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。

另一种存取变量值的方式称为"间接访问"的方式。即，将变量 *i* 的地址存放在另一个变量中。



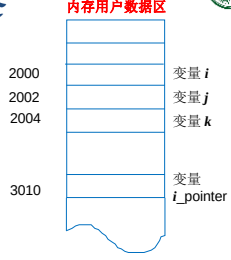


访问地址为2000的单元，可以通过变量名 *i* 或在变量名*i_point* 中存放的地址来访问。

指针变量的概念

指针： 内存地址，用于间接访问内存单元

指针变量： 用于存放地址的变量



声明

例：int i,j,k;
int *i_pointer;

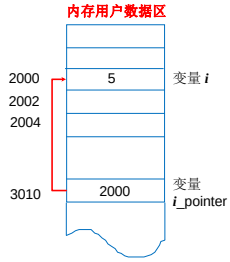
为变量 *i* 分配了一个单元2000
为指针变量 *i_pointer*分配了一个单元3010。

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。

间接访问即通过指向
变量*i*的地址的指针变量 *p* 进行访问

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




```
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
```

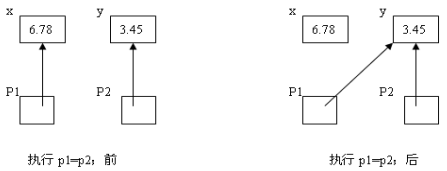
- **注意：指针只能接受地址量。如果将一个任意常数赋予指针，会产生语法错误。**

```
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＝100；b＝10；
  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",a,b);
  printf ("max=%d,min=%d\n",*p1,*p2);
}
```

运行情况如下：

```
5, 9↙
a=5, b=9
max=9, min=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 → □ →

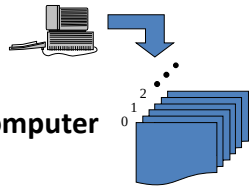
8) Physical structure

Data value

+

logical structure

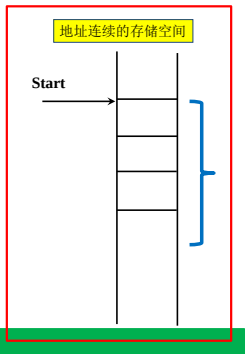
→Stored in the computer



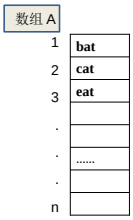
The storage method in the computer

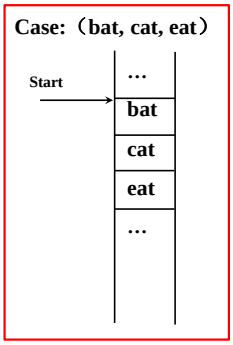
（1）Sequence：连续设计

- ✓ **Storage cell with continuous location address**
- ✓ **Logical relationship is described by the function of storage location**



计算机中的实现方式：
数组类型，单个变量

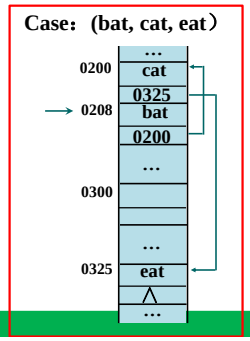




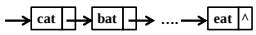
（2） Linked：链接设计

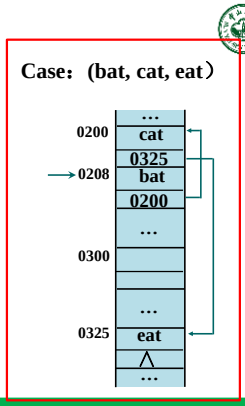
- ✓ Storage cell
- ✓ Logical relationship is described by point

带有指针的单元设计



计算机中的实现方式：指针类型





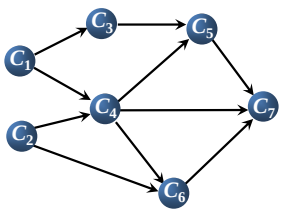
Relationship between logical structure and physical structure

- ➢ 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。

9) Data structures

- ◆ Logical structure
- ◆ Physical structure
- ◆ Set of operations

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



3. Definition of ADT

- ADT（Abstract Data Type）：Data type definition used to solve problem.

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。

数据类型可以看作是已经实现了的抽象数据类型。

ADT int=($\{x \mid x \in Z\}$， {+,-,*,/,%, ≦,==})

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>

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);

Outpit: (y: Bigint);

Pre-condition：n*x.n<2^{512};

Post-condition：y.n=n*x.n;

End ADT

4. Algorithms

4.1 Definition

Informally, an algorithm is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output.

We can also view an algorithm as a tool for solving a well-specified computational problem. The statement of the problem specifies in general terms the desired input/output relationship.

The algorithm describes a specific computational procedure for achieving that input/output relationship.

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”。**

4.2 Description of algorithm

1). 算法的描述方法——*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). 算法的描述方法——*流程图*

优点：处理流程直观

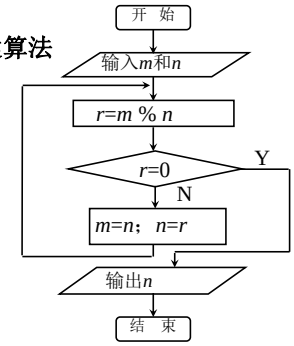
缺点：缺少严密性、灵活性

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). 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。

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). 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。

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;
}
```

4.3 Evaluation:

A Good algorithm:

(1) 正确性(Correctness)
(2) 可读性(Readability)
(3) 健壮性(Robustness)
(4) 效率(Efficiency)

4.4 Analyzing algorithms

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 Compl

- 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。

4.4.1 算法的执行时间，是基本（操作）语句重复执行的次数，它是问题规模的一个函数。我们把这个函数的渐近阶称为该算法的时间复杂度。

- 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

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

Case 1：

```
for(I=1，I<=n;++I)                    n+1
  for(j=1;j<=n;++j)                   n*(n+1)
 {
   c[I][j]=0;                         n*n
    for(k=1;k<=n;++k)                 n*n*(n+1)
        c[I][j]+=a[I][k]*b[k][j];     n*n*n
 }
```

Total time: $2n^3+3n^2+2n+1$

Insert-Sort(A)	Cost	times
1. for j<-2 to length[A]	c_1	n
2. do key<- A[j]	c_2	n-1
3. *Insert A[j] into the sorted sequence A[1..j]	c_3	
4. i<-j-1	c_4	n-1
5. While i>0 and A[i] >key	c_5	$\sum_{j=2}^{n} t_j$
6. do A[i+1]<-A[i]	c_6	$\sum_{j=2}^{n} (t_j-1)$
7. i<-i-1	c_7	$\sum_{j=2}^{n} (t_j-1)$
8. A[i+1]<-key	c_8	n-1

Order of growth

- In order to describe the computational times, order of growth of the running time of an algorithm is used.
- When we look at input sizes large enough to make only the order of growth of the running time relevant, we are studying the asymptotic efficiency of algorithms.
- That is, we are concerned with how the running time of an algorithm increases with the size of the input in the limit, as the size of the input increases without bound.

Definition：

$f(n)$，$g(n)$ are positive function about natural number. If there is a constant $c>0$，and natural number n_0，When $n>n_0$，there is $f(n)<c\,g(n)$, so

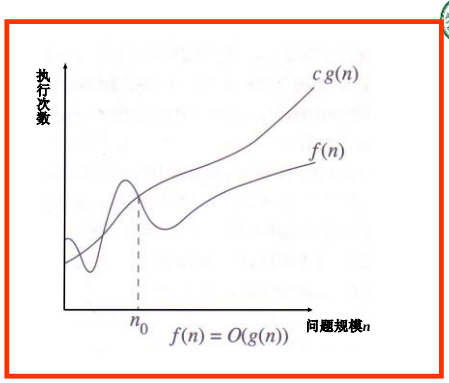
$f(n)=O(g(n))$

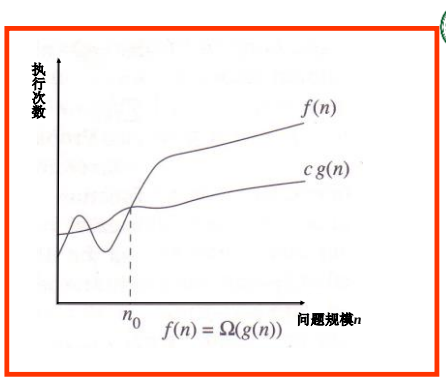
If there is a constant $c>0$，and natural number n_0，When $n>n_0$，there is $f(n)>c\,g(n)$, so

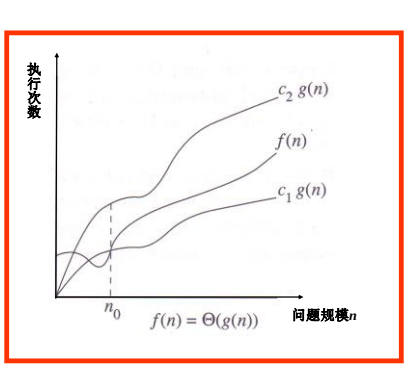
$f(n)=\Omega(g(n))$

if $f(n)=O(g(n))$, $f(n)=\Omega(g(n))$, then

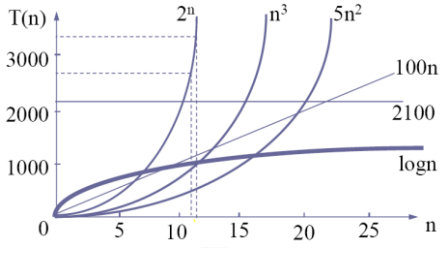
$f(n)=\Theta(g(n))$







常见的时间复杂性



Normal：

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

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

Rules：

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

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

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

时间复杂性的分析方法

✓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。

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。

●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：

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

取 $f(n)=n$

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

$=O(n)$

4.4.2. 算法的空间复杂性：是指算法在执行过程中的存储量需求。

➢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。

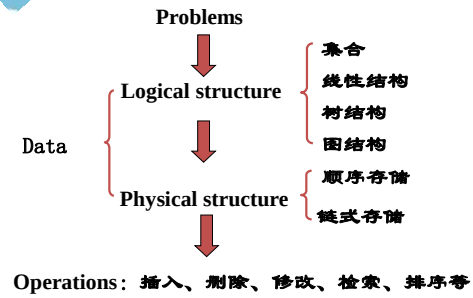
➢算法的执行的不同时刻，其空间需求可能是不同的。

➢算法的空间复杂性是指算法在执行过程中的最大存储量需求。

5. Data structure and Algorithm

Program=Algorithm + Data Structure

Summary of Data structure and algorithm



6. 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.***

7. A case：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）。

Analysis

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

Design

对于课程文件中表示的开设课程的名称：

数据课程名称之间没有内在联系，可以考虑是集合类型。

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
Data
cstype V; //cstype 为课程类型的集合，集合中的元素
//为cstype
Operations
Makenull
后件：V被置成空集合；
Insert(cs)
输入：cs是课程名，为cstype类型；
后件：将cs加入到集合的最后位置上。
Delete(cs)
输入：cs是课程名，为cstype类型；
前件：cs是 V中的一门课程；
后件：从V中删除 cs 这门课 。

Retrieve(p，cs)
输入：p为整数，表示集合中元素的位置；
输出：cs为cstype类型，表示课程名；
前件：P是集合V 中一个有效位置；
后件：返回位置P所对应的课程名。

……..

End ADT

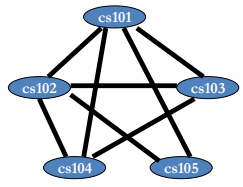
对于学生选课程的文件：

课程同时被一位学生选，建立课程被学生选学的关系。（无向图表示）

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



图的染色问题

ADT definition：

ADT Graph
Data
Csset V;
Edgeset E;
Operations
Initial
后件：边集合为空；
Addedge(u,v)
输入：u,v是cstype类型；
前件：(u,v) ∈E, 且u≠v，u，v ∈V
后件：E中添加了边（u,v）。

Isedge(u,v)
输入：u, v 是 cstype 类型；
前件：u，v ∈V
后件：若边（u, v） ∈ E 则返回真，否则返回假。

End ADT

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；
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
```