

数据结构与算法

控制科学与工程学院 李丹

E-Mail: ldan@dlut.edu.cn

Office: 北门创新园大厦A0716

课程设置

数据结构与算法
【48学时、3学分】



数据结构及算法
(用抽象数据类型
及C语言描述)

数据结构与算法 【百分制】



闭卷笔试

(100分*80%)



平日作业

(100分*20%)

作业及考试

不涉及程序编写



教材及参考书

 教材: 《数据结构(C语言版)》 严蔚敏

清华大学出版社

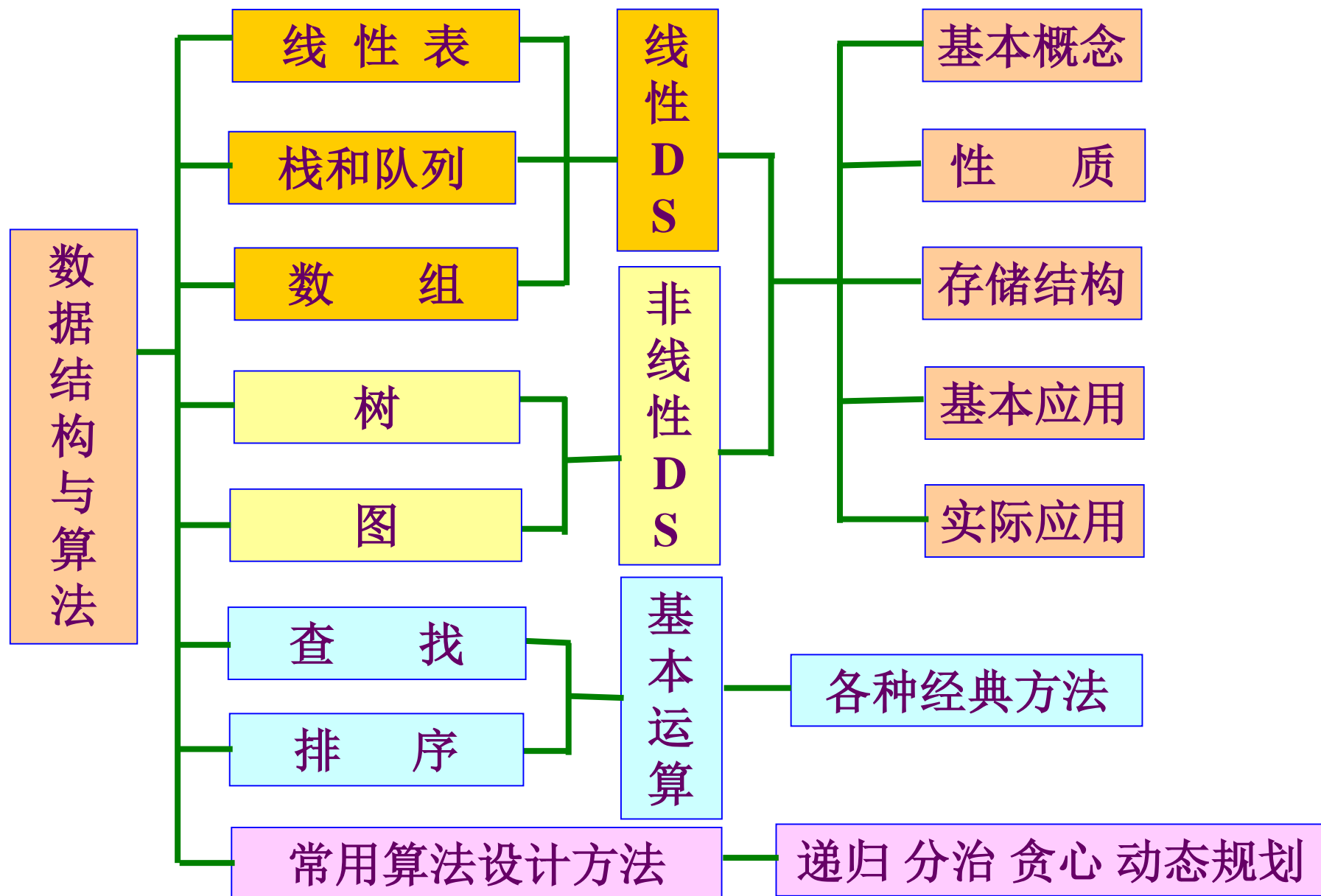
 课件\作业下载QQ群:

437242452

“大工数据结构与算法”

建议—打印PDF版课件,
用于课堂笔记





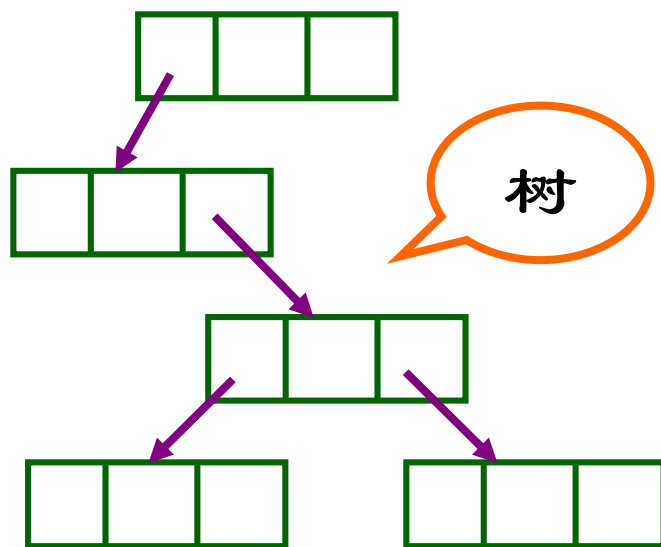
线性DS——数据之间是**一对一**的关系

001	高等数学	樊映川	S01
002	理论力学	罗远祥	L01
003	高等数学	华罗庚	S01
.....

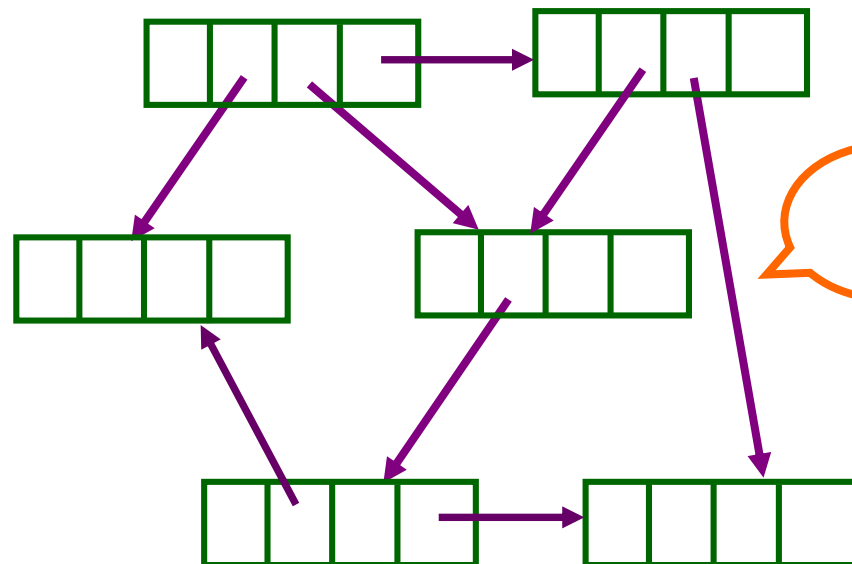


线性表

非线性DS——数据之间是**一对多**、**多对多**的关系



树



图



C语言基础复习

 结构体——把不同类型数据组合成一个整体

```
struct student
{
    long num;
    char name[20];
    int age;
};
```

结构体类型定义

完成描述，不分配内存

★ 变量定义：

分配内存

```
struct student s1,*p;
```


★ 成员访问：

```
s1.num=201682001;
```

```
p->age=19;
```

```
(*p).age=19;
```

C语言基础复习

 用户自定义类型——为已有数据类型重新命名

已有类型

新类型名

```
typedef int INT;
```

```
INT a, b;
int a, b;
```

等价

```
typedef struct student
```

```
{
```

```
    long num;
```

```
    char name[20];
```

```
    int age;
```

```
} ST;
```

```
ST stu1, *p;
```

```
stu1.num=201682001;
```

```
p->age=19;
```

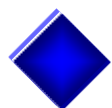
```
(*p).age=19;
```

已有类型

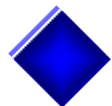
新类型名

End

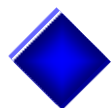
第一章 绪言



数据结构研究的问题



基本概念和术语

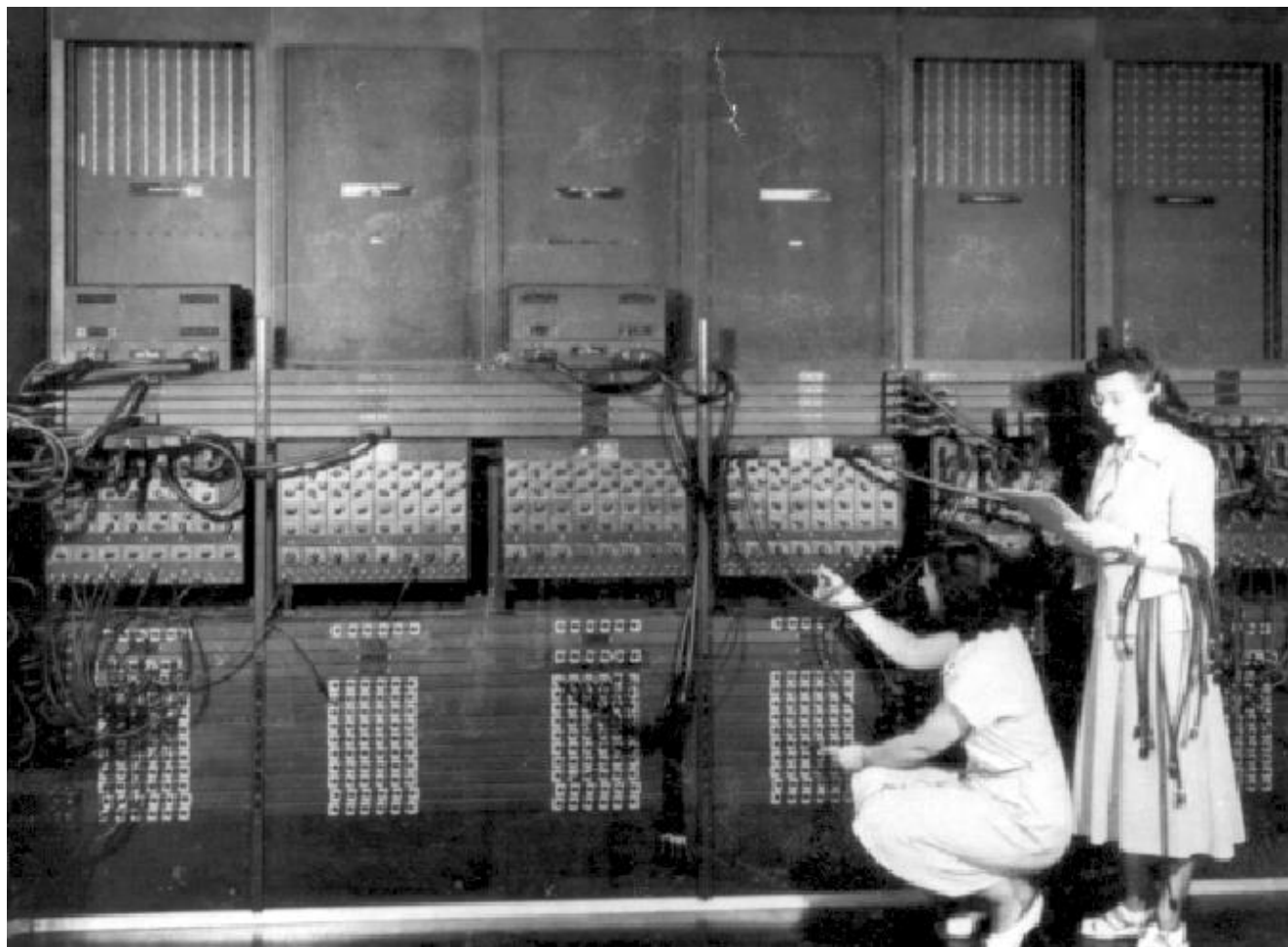


算法描述和算法分析



本章小结

数据结构研究的问题



1946年，第一台电子计算机**ENIAC**

Shown here are two women “programming” ENIAC. U.S. Army Photo.



数据结构研究的问题

数据结构的起源

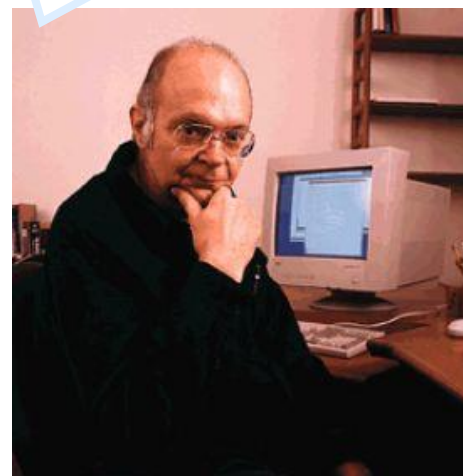
早期，人们都把计算机理解为数值计算工具，感觉计算机当然是用来计算的。

随着计算机所求解问题复杂性的增加，人们发现需要更科学有效的手段，才能更好的解决问题。

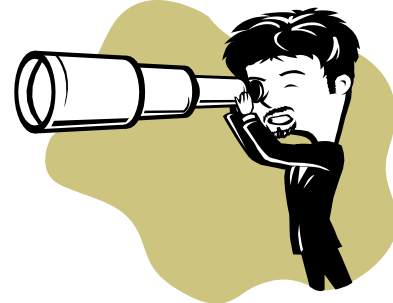
1968年，美国的高德纳教授在《计算机程序设计艺术》中开创了数据结构的课程体系。



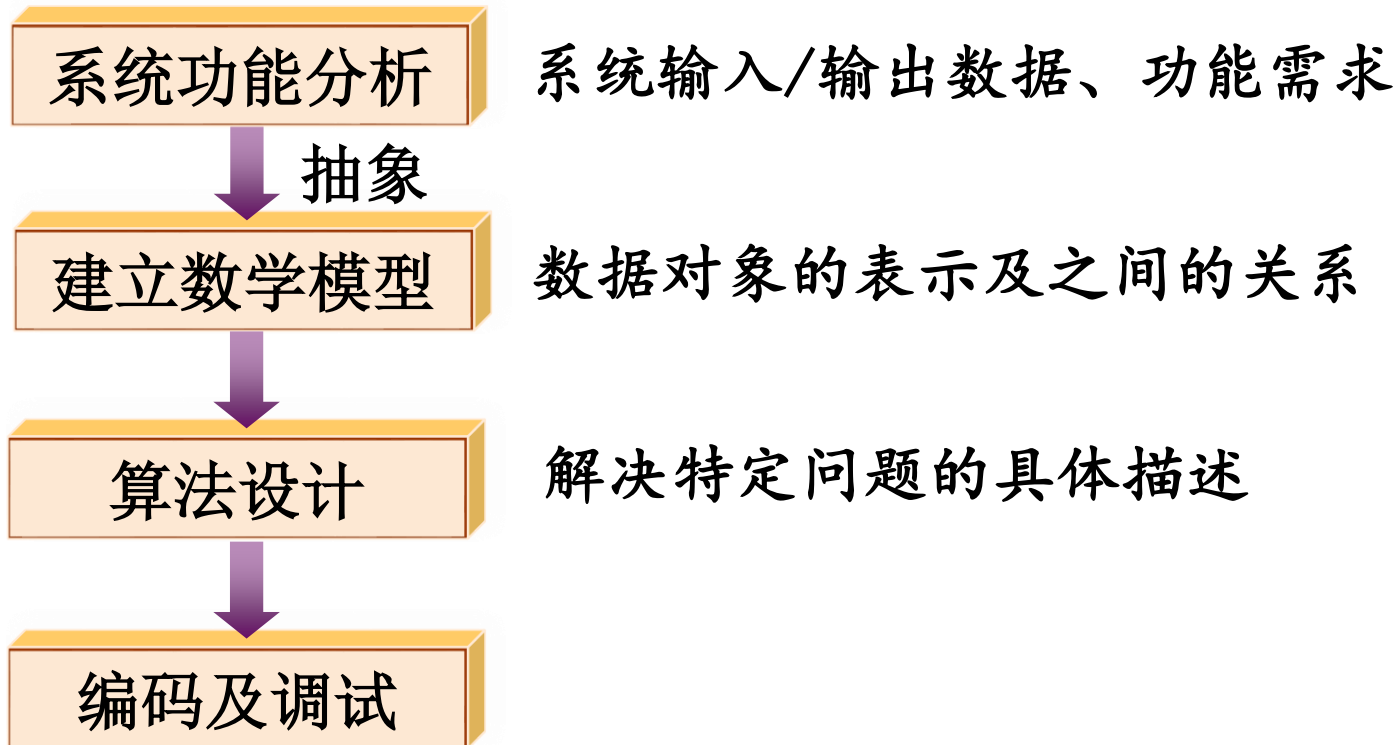
niph.com/505



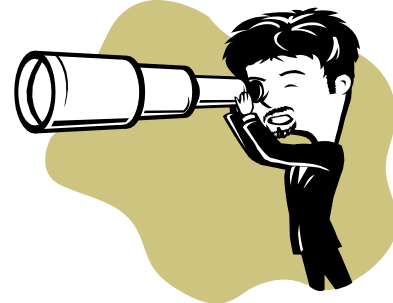
数据结构研究的问题



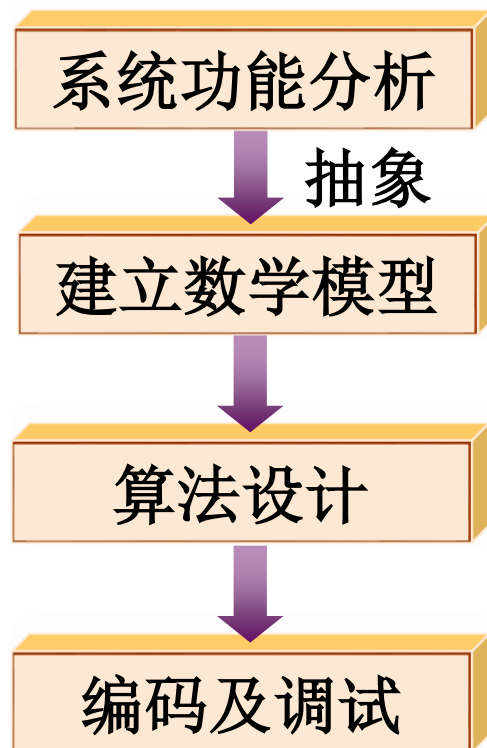
程序设计一般流程



数据结构研究的问题



程序设计一般流程



数值计算问题

非数值计算问题

数学方程

✗ 数学方程

✓ 数据结构

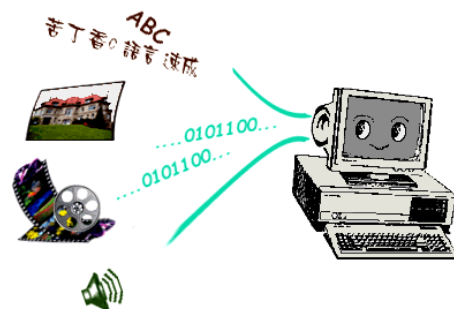
例：已知长方形泳池的长和宽，求面积。

① 数学模型：

$$\text{area} = \text{len} \times \text{wide}$$

② 编程

```
main( )
{ int len=3,wide=5,area;
  area=len*wide;
  printf ("%d",area);
}
```



数据结构研究的问题

线性表

例1 图书馆的书目自动检索系统

书目卡片

登录号：

书名：

作者：

分类号：

.....

001	高等数学	樊映川	S01
002	理论力学	罗远祥	L01
003	高等数学	华罗庚	S01
004	线性代数	栾汝书	S02
.....

书目文件

索引表

高等数学	001, 003.....
理论力学	002,
线性代数	004,
.....

樊映川	001, ...
华罗庚	002,
栾汝书	004,
.....




电话号码查询系统、学生成绩
管理系统、职工信息系统等**文**
档管理的数学模型

数据结构研究的问题

例2 人机对弈问题

 1997年，加里·卡斯帕罗夫与IBM “Deep Blue”对弈，国际象棋AI(Artificial Intelligence)第一次打败顶尖人类棋手。



 2016年3月，李世石与Google的“AlphaGo”对弈，在五番棋中以1-4被围棋AI击败。



2017年，李世石被称为“最后一个战胜AlphaGo的人类棋手”。



数据结构研究的问题

例2 人机对弈问题

🏠 2017年1月，一个叫“**Master**”的神秘ID在网络对战平台上与全球最强人类棋手对弈。在不到三天时间里，**Master**大开杀戒，迅速到达围棋九段，获得了**60胜1平0负**的不败战绩。

🏠 Master [9段]: 現在我將與古力老師下最後一局棋

🏠 Master [9段]: 希望大家這陣子，都享受master的棋局 😊

🏠 Master [9段]: 我是AlphaGo的黃博士

随后，Google官方发文称“Now that our unofficial testing is completed, we're looking forward to playing some official, full-time games later this year.....”



04/01/17

We've been hard at work improving AlphaGo, and over the past few days we've played some unofficial online games at fast time controls with our new prototype version, to check that it's working as well as we hoped. We thank everyone who played our accounts Magister(P) and Master(P) on the Tygem and FoxGo servers, and everyone who enjoyed watching the games too! We're excited by the results and also by what we and the Go community can learn from some of the innovative and successful moves played by the new version of AlphaGo.

Having played with AlphaGo, the great grandmaster Gu Li posted that, "Together, humans and AI will soon uncover the deeper mysteries of Go". Now that our unofficial testing is complete, we're looking forward to playing some official, full-length games later this year in collaboration with Go organisations and experts, to explore the profound mysteries of the game further in this spirit of mutual enlightenment. We hope to make further announcements soon!



数据结构研究的问题

例2 人机对弈问题

📺 2017年5月，“当今围棋第一人”柯洁与AlphaGo之间的三番棋最终结局锁定在0:3，人类最终全负于AlphaGo。



📺 2017年11月，Google Deepmind的新版AlphaGo Zero以100:0打败了前辈AlphaGo。



棋士柯洁

今天 02:22

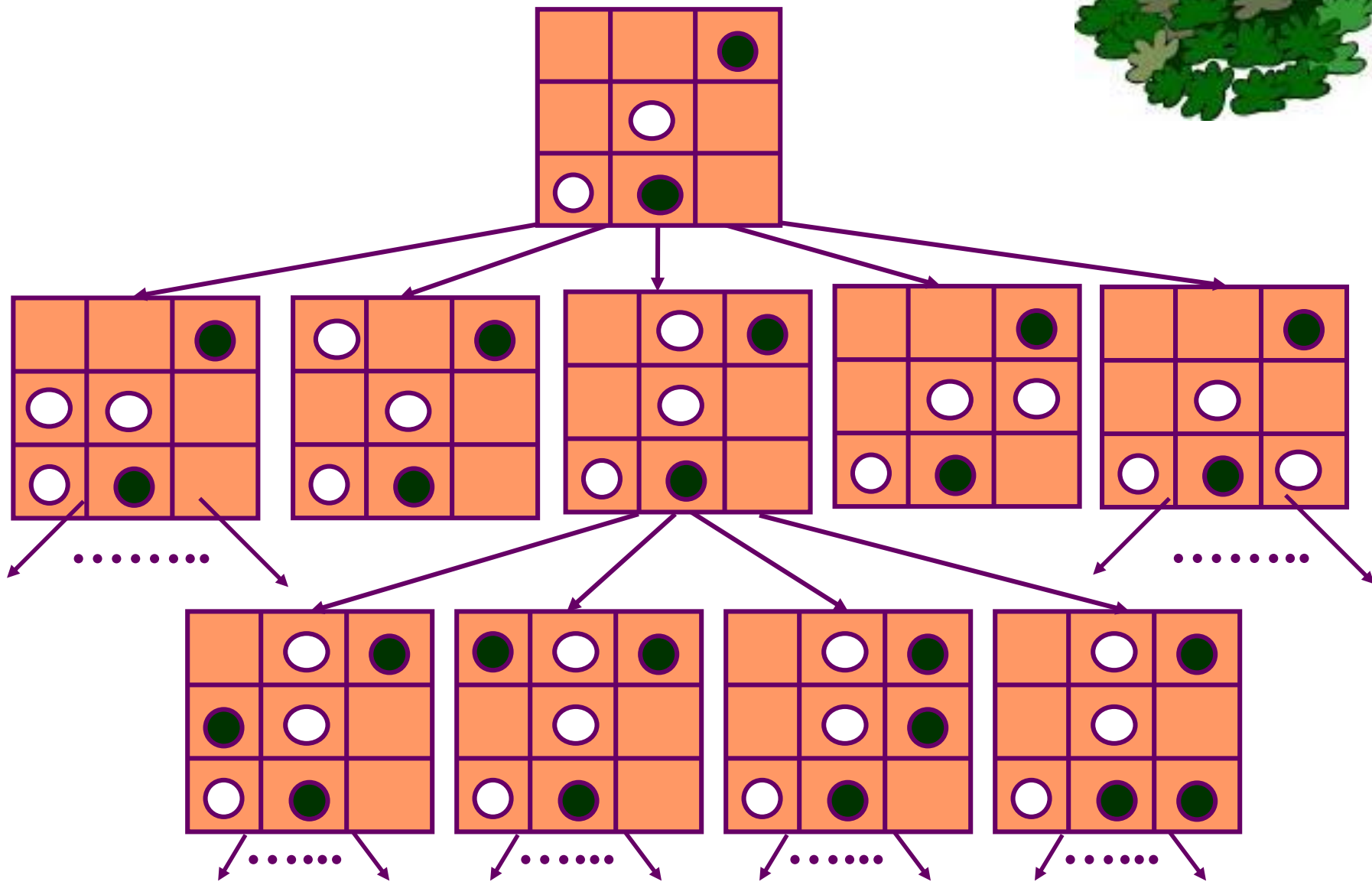
一个纯净、纯粹自我学习的alphago是最强的...对于alphago的自我进步来讲...人类大多余了🙄



数据结构研究的问题

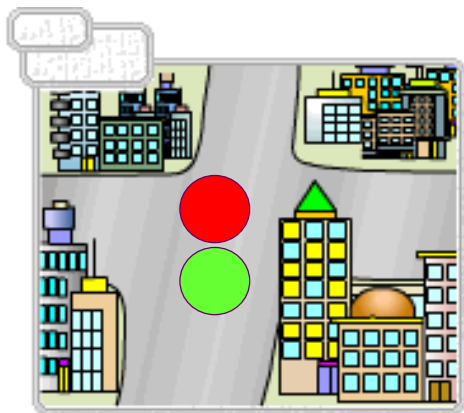
例2 人机对弈问题

树

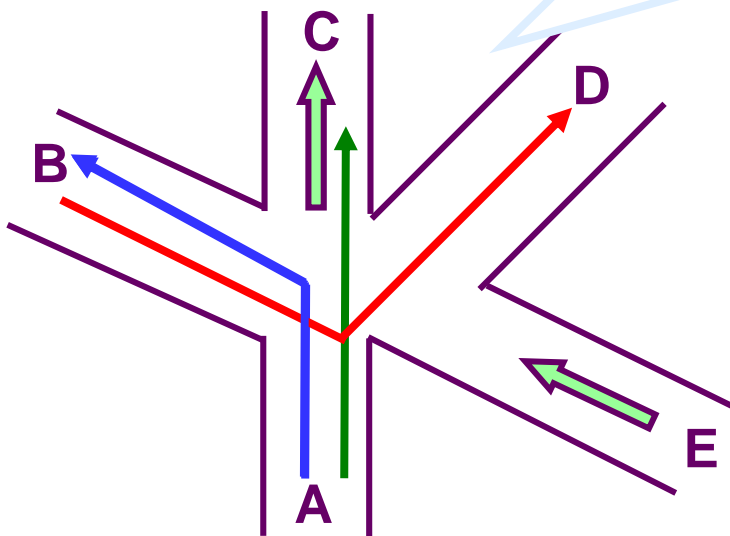


数据结构研究的问题

例3 多叉路口交通灯管理问题



问题的求解思路——
根据各条通路能否同时
通行设置交通灯



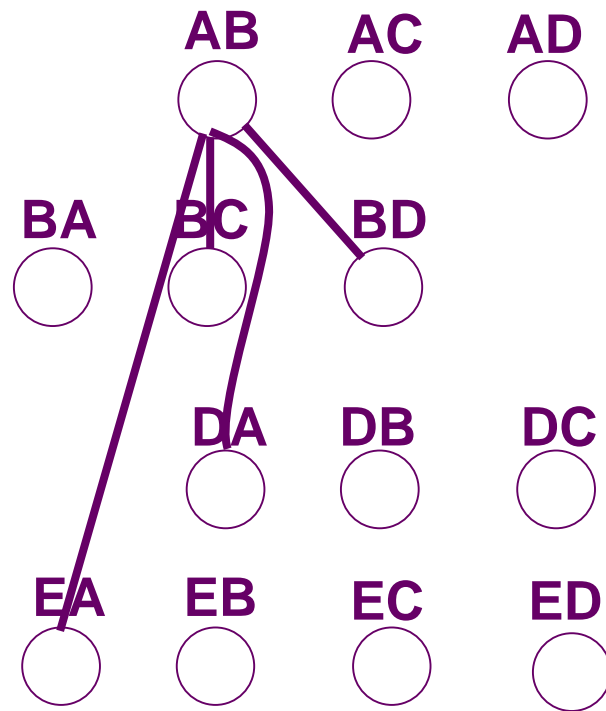
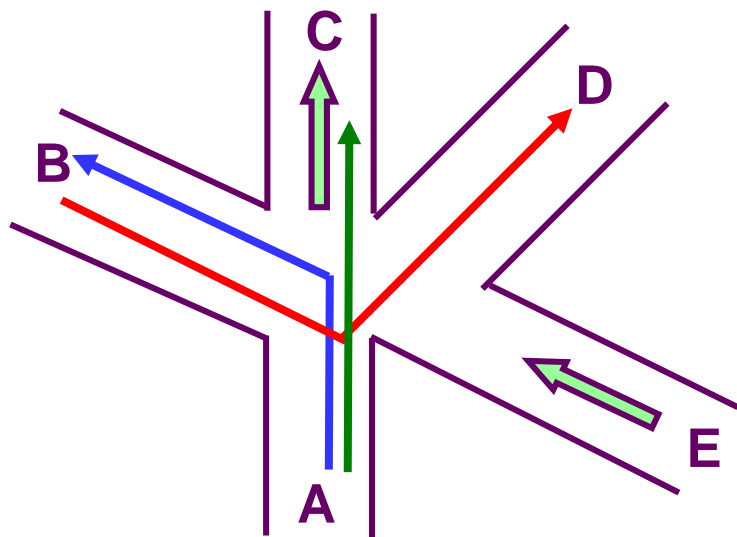
交通灯的设置规则

- ① 能够同时通行的通路，可用同一种颜色的交通灯控制，
如：通路**AB**与**AC**；
- ② 不能同时通行的通路，必须用不同颜色的交通灯控制，
如：通路**AB**与**BD**；
- ③ 交通灯的颜色尽可能少。

数据结构研究的问题

例3 多叉路口交通灯管理问题

问题的求解思路——
根据各条通路能否同时
通行设置交通灯



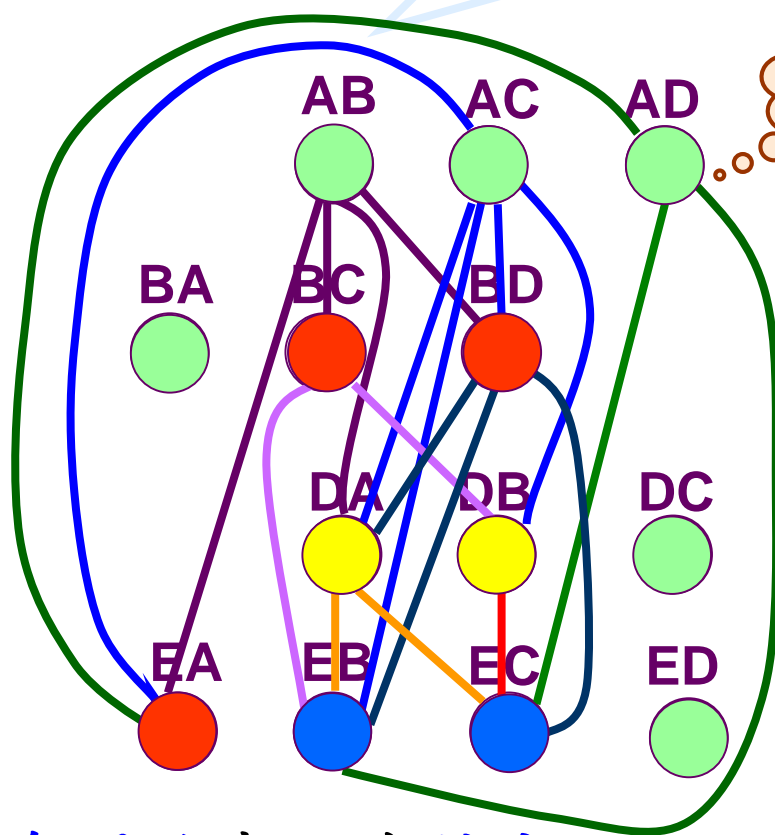
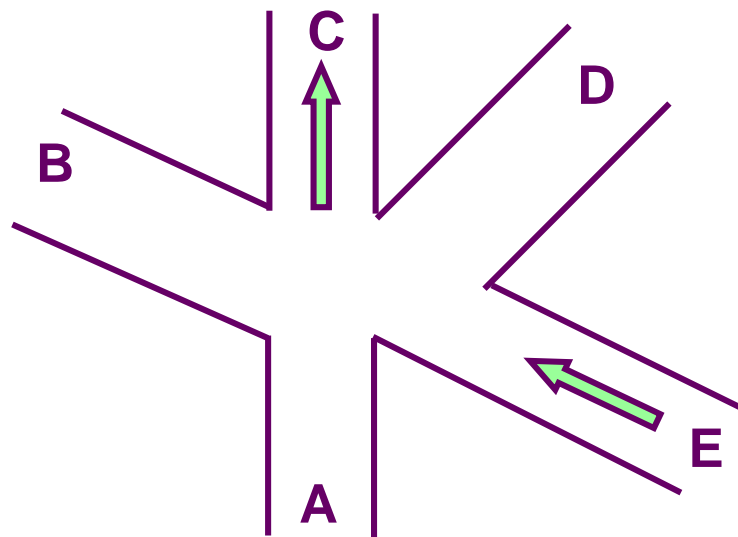
求解步骤

- ① 将问题中的数据——各条通路表示为结点；
- ② 连线：将不能同时通行的通路用线连接；

数据结构研究的问题

例3 多叉路口交通灯管理问题

问题的求解思路——
根据各条通路能否同时
通行设置交通灯



求解步骤

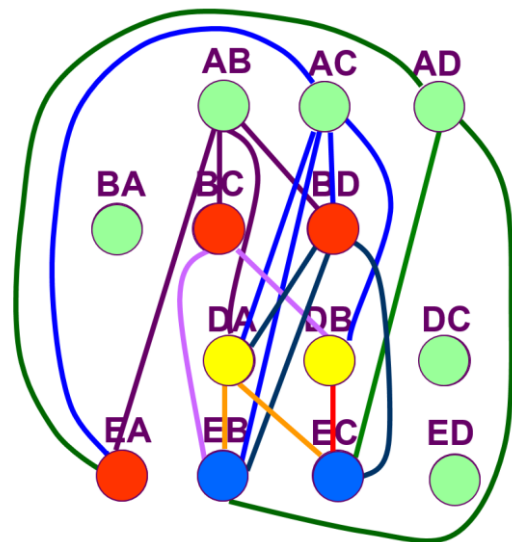
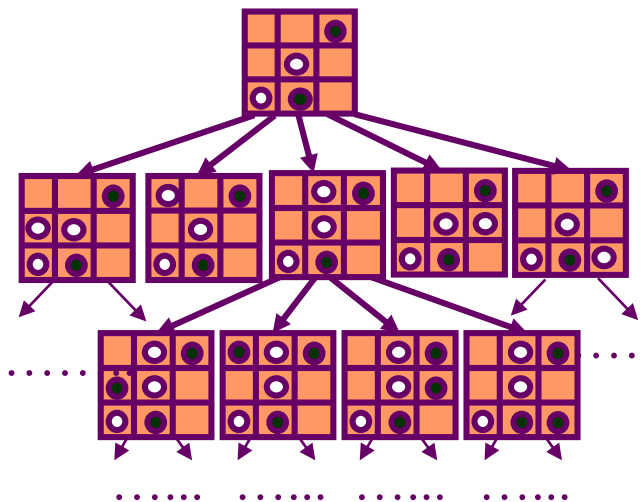
- ① 将问题中的数据——各条通路表示为结点；
- ② 连线：将不能同时通行的通路用线连接；
- ③ 填色：有线连接的结点用不同颜色填充。

数据结构研究的问题

☺数据结构是一门研究**非数值计算问题**中，
计算机的**操作对象**及对象之间**关系**和**运算**的
学科。

001	高等数学	樊映川	S01
002	理论力学	罗远祥	L01
003	高等数学	华罗庚	S01
004	线性代数	栾汝书	S02
.....

非数值计算问题

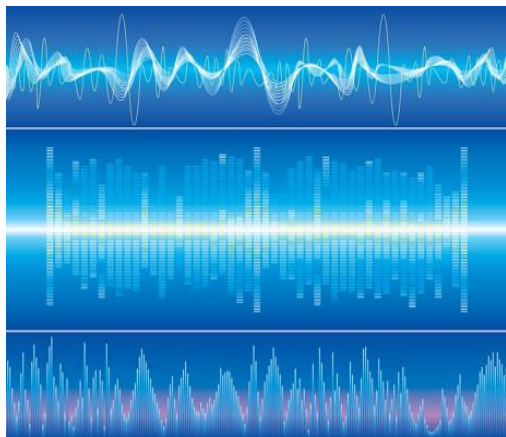


Back

基本概念和术语

★ 数据(data)——所有能输入到计算机中、用于描述客观事物的符号。

数据不仅包括整型、实型等数值类型，还包括非数值类型，如



声音



图像



视频

.....

可以通过编码等手段
变成字符数据来处理

基本概念和术语

★数据元素(data element)——组成数据的、有一定意义的基本单位，也称结点或记录。

★数据项(data item)——有独立含义的数据最小单位，也称域。

一个数据项

学号	姓名	语文	数学	C语言
201082001	张三	85	54	92
201082002	李四	92	86	67
201082003	王五	87	76	73
201082004				
.....				

一个数据元素

基本概念和术语

面向问题的，从问题中抽象出的逻辑关系

★数据的逻辑结构——数据元素间的逻辑关系

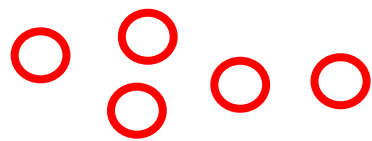
数据逻辑结构分为：

集合——数据元素间除“同属于一个集合”外，无其它关系

线性结构——数据元素是一一对应的关系，如线性表、栈、队列

树形结构——数据元素是一对多的关系

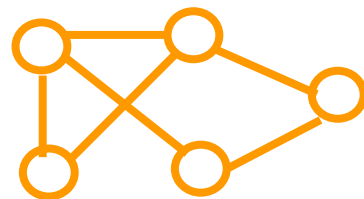
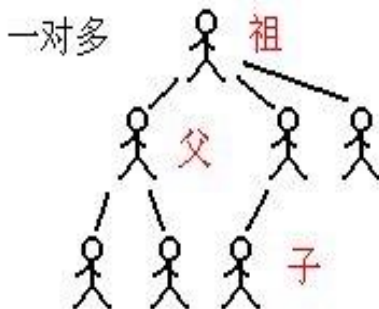
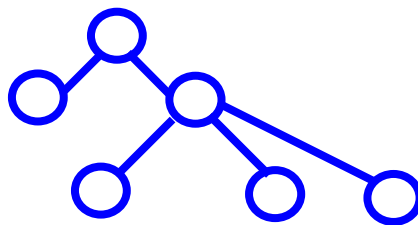
图状结构——数据元素是多对多的关系



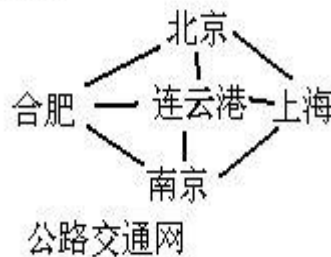
同属色彩集合



学号	姓名	语文	数学	C语言
201082001	张三	85	54	92
201082002	李四	92	86	67
201082003	王五	87	76	73
201082004				
.....				



多对多



基本概念和术语

面向计算机的，指数据及数据间逻辑关系的存储方式

★数据的存储结构(物理结构)——数据的逻辑结构在计算机存储实现。

存储结构分为：

{ 顺序存储结构——使用一段连续的存储空间，借助元素在内存中的相对位置来表示数据元素间的逻辑关系。

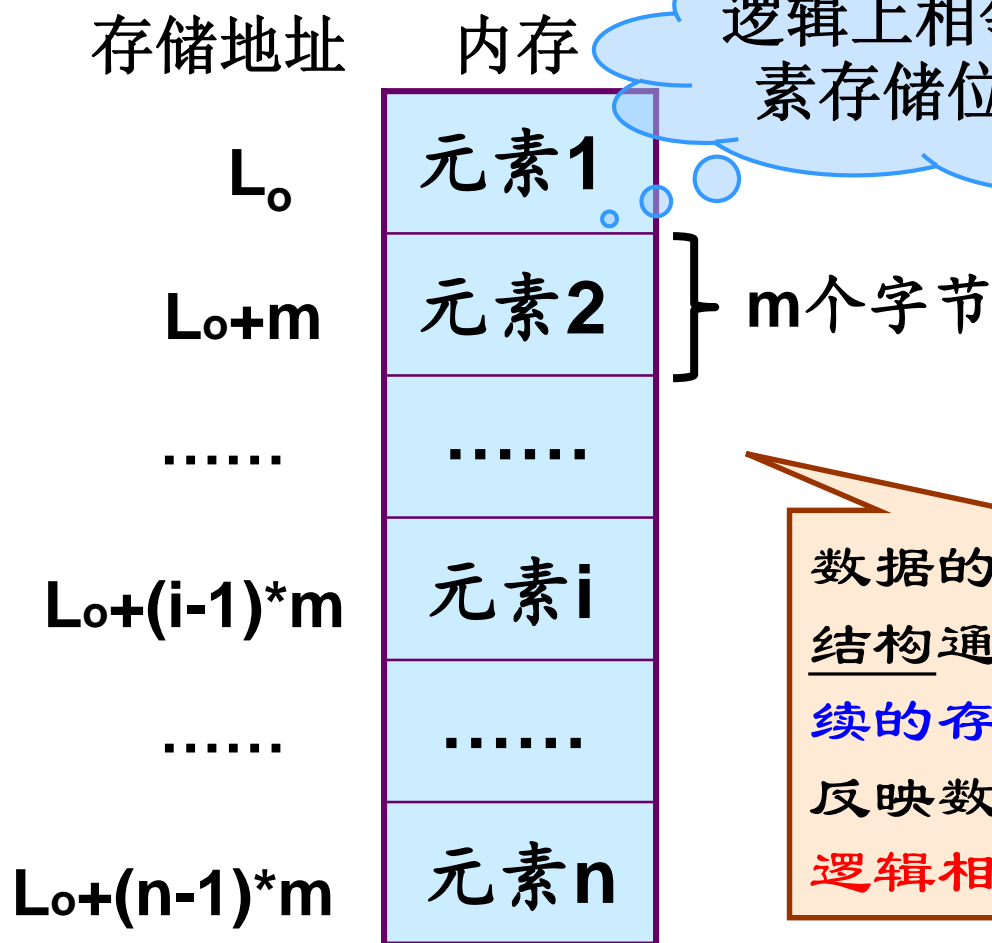
Click Here

{ 链式存储结构——无需使用连续的存储空间，借助指针来表示数据元素间的逻辑关系。

Click Here



顺序存储



逻辑上相邻的数据元素存储位置也相邻

数据的顺序存储结构通过分配连续的存储空间来反映数据元素的逻辑相邻关系

$$\text{Loc}(\text{元素}i) = L_0 + (i-1)*m$$

Back

链式存储

头指针

1346

“链”——指示逻辑上相邻元素的指针



内存

存储地址	元素值	指针
1346	元素1	1400
1350	元素4	^
...		
1400	元素2	1536
...		
1536	元素3	1350

一个数据元素

```
/*stdio.h中*/  
#define NULL 0
```

数据的链式存储结构通过指针来反映数据元素的逻辑相邻关系

Back

基本概念和术语

★ **数据类型**—— 一组**性质相同的值的集合**，以及定义在这个集合上的**操作**的总称。

例 C语言中有int、char、float、double等**基本数据类型**，还包括数组、结构体、共用体、枚举等**构造数据类型**，此外，可用typedef**自定义数据类型**。

```
typedef struct student
{
    long num;
    char name[20];
    int age;
} ST;
```

基本概念和术语

★抽象数据类型 (Abstract Data Type, ADT)

定义：指一个数学模型及定义在该模型上的一组操作

◆数据抽象：仅描述其数据对象、数据关系及基本操作，不考虑数据的存储形式和操作的实现算法

◆数据封装：将数据对象及的基本操作捆绑在一起，进行封装，对外部用户隐藏其内部实现细节

ADT 形式化定义：

功能——

完成描述

ADT 抽象数据类型名

{ 数据对象： ...

数据关系： ...

基本操作： ...

} ADT 抽象数据类型名

基本概念和术语

例 复数的ADT定义:

数据抽象: 不考虑数据的存储形式和操作的实现算法;

数据封装: 将数据与操作捆绑在一起

ADT Complex

{ 数据对象: $D = \{ e1, e2 \mid e1, e2 \text{ 均为实数} \}$
数据关系: $R = \{ \langle e1, e2 \rangle \mid e1 \text{ 是复数的实部, } e2 \text{ 是复数的虚部} \}$
基本操作:

AssignComplex(&Z, v1, v2); 构造复数Z, 其实部和虚部分别赋以参数v1和v2的值

GetReal(Z, &real); 用real返回复数Z的实部值

GetImag(Z, &Imag); 用Imag返回复数Z的虚部值

Add(z1, z2, &sum); 用sum返回两个复数z1, z2的和值

} ADT Complex

引用

✍ 教材中算法采用以抽象数据类型为基础的类C语言。

基本概念和术语

★ 引用(Reference)

*.cpp支持引用

- ✍ 为简化复杂的 **指针** 操作, **C++** 等面向对象程序设计语言提供了 **引用** 机制 (“&”), 即为变量起一个 **别名**。
- ✍ **别名** (引用) 与原变量享受同等的访问待遇。
- ✍ C语言 **不支持** 引用类型。 (***.c不支持引用**)

```
int a=4;
```

```
int &x=a; /*x是a的引用*/
```

```
x=8;
```

x与a占用相同的存储空间

变量a

8

引用x

变量及其引用
同步变化

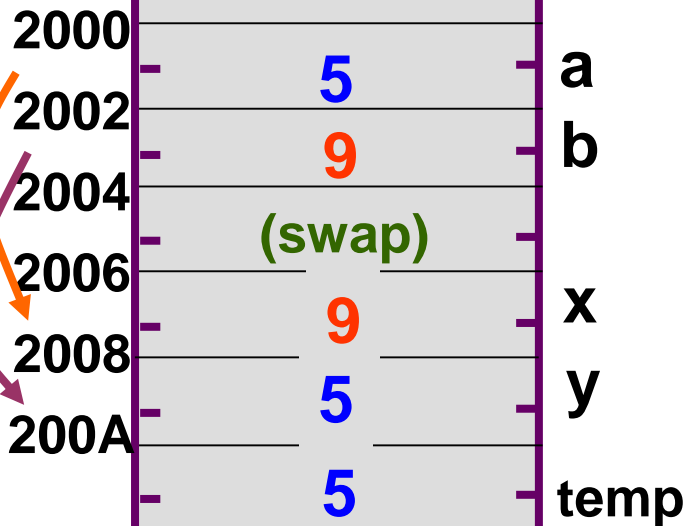
基本概念和术语

✍ 引用常用于函数形参，便于实现数据的**双向传递**。

```
void swap(int x, int y)
{
    int temp;
    temp=x;
    x=y;
    y=temp;
}
```

值传递——单向
实参=>形参，形参≠>实参

COPY



当用执行语句**swap(a,b)**时：
a 和 **b** 的值没有发生交换

基本概念和术语

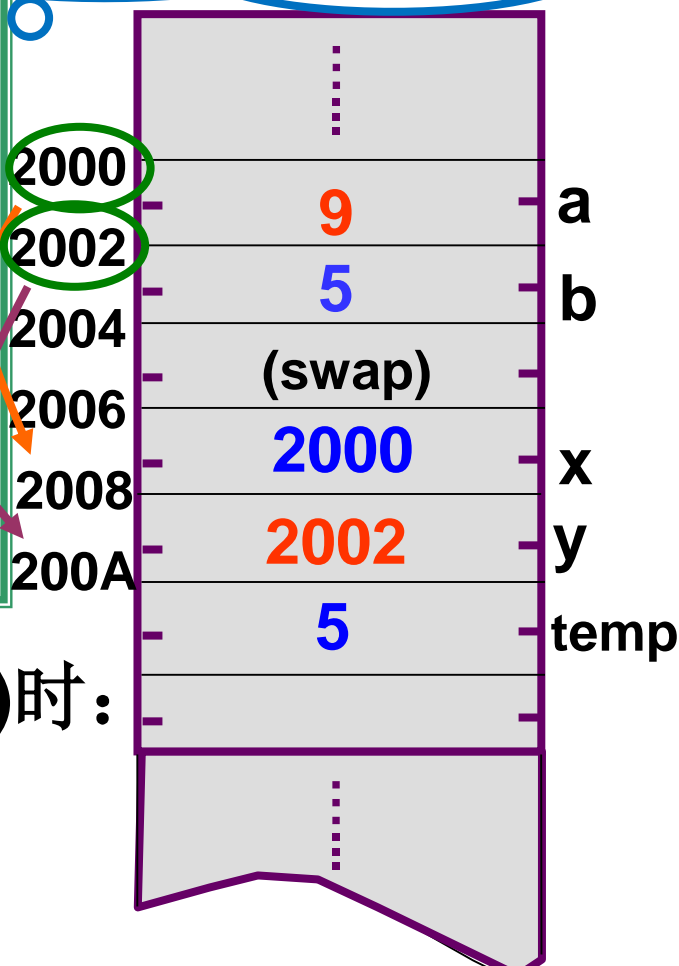
引用常用于函数形参，便于实现数据的**双向传递**。

地址传递——双向

实参、形参指针指向同一存储空间

```
void swap(int *x, int *y)
{
    int temp;
    temp=*x;
    *x=*y;
    *y=temp;
}
```

COPY



当用执行语句**swap(&a,&b)**时:

a和**b**的值发生了交换

基本概念和术语

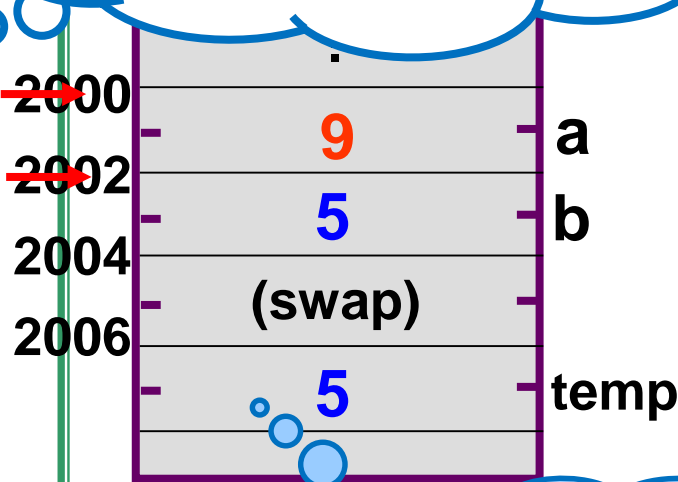
引用常用于函数形参，便于实现数据的双向传递。

```
void swap(int &x, int &y)
{
    int temp;
    temp=x;
    x=y;
    y=temp;
}
```

引用传递——双向
引用型形参与实参占
用同一存储空间

引用x → 2000

引用y → 2002



当用执行语句 `swap(a,b)` 时：
a和**b**的值发生了交换

*.c不支持引用
*.cpp支持引用

显然，采用引用方式更为简洁方便。

所以教材中很多算法都采用引用形式的形参。

基本概念和术语

引用常用于函数形参，便于实现数据的双向传递。

```
void swap(int &x, int &y)
{
    int temp;
    temp=x;
    x=y;
    y=temp;
}
```

引用



如何区分&是“引用”
还是“取变量地址”？

&仅在形参及定义部分表示引用

```
int a=4;
int &x=a;
```

引用

```
int a=5, b=9;
swap(&a,&b);
```

取变量地址

基本概念和术语

★抽象数据类型的表示和实现

——通过已实现的数据类型实现。

复数的定义

```
typedef struct
{   float realpart;
    float imagpart;
}complex;
```

//基本操作：用sum返回两个复数z1,z2的和值

```
void add(complex z1, complex z2, complex &sum)
{   sum.realpart = z1.realpart + z2.realpart;
    sum.imagpart = z1.imagpart + z2.imagpart;
}
```

基本概念和术语

★抽象数据类型的表示和实现

——通过**已实现的数据类型**实现。

//教材及课件常用宏定义及类型

```
#define TRUE      1
#define FALSE     0
#define OK        1
#define ERROR     0
#define INFEASIBLE -1
#define OVERFLOW  -2
```

```
typedef int Status;
```

算法描述和算法分析

★ 算法(algorithm)——解决特定问题的具体步骤描述，是指令的有限序列



★ 算法特性——

- ✓ **有穷性**——算法必须在执行有限步骤后结束
- ✓ **确定性**——算法每一步骤必须是确切定义的，不能产生二义性
- ✓ **可行性**——算法是能行的
- ✓ **输入**——算法有零个或多个输入
- ✓ **输出**——算法有一个或多个输出

算法描述和算法分析

★ 算法的评价——衡量算法优劣的标准

✓ 正确性(correctness)

✓ 可读性(readability)

✓ 健壮性(robustness)

✓ 效率

正确性：1. 程序中不含语法错误；
2. 程序对于精心选择的、典型、苛刻且带有刁难性的输入数据能够得出满足要求的结果。

可读性：1. 算法应该易于人的理解；
2. 晦涩难读的算法易于隐藏较多错误而难以调试。

健壮性：1. 当输入数据非法时，算法能做出恰当反应或进行相应处理，而不是产生莫名其妙的输出结果。
2. 有处理出错的方法，不是中断程序，而是返回一个表示错误或错误性质的值，以便进行处理。

算法描述和算法分析



★ 算法设计重要性

例 百钱买百鸡问题

用100元钱买100只鸡，母鸡5元/只，公鸡3元/只，小鸡3只1元，计算三种鸡可以各买几只？

解法：设母鸡、公鸡、小鸡各买 x , y , z 只。则有：

$$x + y + z = 100$$

$$5x + 3y + z/3 = 100$$

循环层数太多☹

方法1——用三重循环：

```
for(i=0; i<=100; i++)
```

```
    for(j=0; j<=100; j++)
```

```
        for(k=0; k<=100; k++)
```

```
            { if(k%3 == 0 && i+j+k == 100 && 5*i+3*j+k/3 == 100)
```

```
                printf("%d,%d,%d\n", i,j,k);
```

```
            }
```

循环次数=101*101*101
即约一百万次

算法描述和算法分析



★ 算法设计重要性

例 百钱买百鸡问题

用100元钱买100只鸡，母鸡5元/只，公鸡3元/只，小鸡3只1元，计算三种鸡可以各买几只？

解法：设母鸡、公鸡、小鸡各买 x , y , z 只。则有：

$$x + y + z = 100$$

$$5x + 3y + z/3 = 100$$

每层循环次数太多☹

方法2——用二重循环： $z=100-x-y$

```
for(i=0; i<=100; i++)
```

```
    for(j=0; j<=100; j++)
```

```
        { k=100 -i -j;
```

```
          if(k%3 == 0 && 5*i+3*j+k/3 == 100)
```

```
            printf("%d,%d,%d\n", i,j,k);
```

```
        }
```

循环次数=101*101
即约一万次

算法描述和算法分析



★ 算法设计重要性

例 百钱买百鸡问题

用100元钱买100只鸡，母鸡5元/只，公鸡3元/只，小鸡3只1元，计算三种鸡可以各买几只？

解法：设母鸡、公鸡、小鸡各买 x , y , z 只。则有：

$$x + y + z = 100$$

$$5x + 3y + z/3 = 100$$

能否用一层循环？

方法3——用二重循环：共100元，所以 $x \leq 20$, $y \leq 33$

```
for(i=0; i<=20; i++)
```

```
    for(j=0; j<=33; j++)
```

```
        {   k=100 -i -j;
```

```
            if(k%3 == 0 && 5*i+3*j+k/3 == 100)
```

```
                printf("%d,%d,%d\n", i,j,k);
```

```
        }
```

循环次数=21*34=714

算法描述和算法分析



★ 算法设计重要性

例 百钱买百鸡问题

用100元钱买100只鸡，母鸡5元/只，公鸡3元/只，小鸡3只1元，计算三种鸡可以各买几只？

解法：设母鸡、公鸡、小鸡各买 x , y , z 只。则有：

$$x + y + z = 100$$

$$5x + 3y + z/3 = 100 \longrightarrow 15x + 9y + z = 300$$

Good!☺

方法4——用一重循环：合并方程得到： $14*x + 8*y = 200$

简化为： $7*x + 4*y = 100$

所以有： $y = (100 - 7*x)/4$ ， $x \leq 14$ ，且 x 必为4的倍数

```
for(i=0; i<=14; i+=4)
```

```
{ j = (100 - 7*i)/4; k=100 -i -j;
```

```
    if(k%3 == 0 && 5*i+3*j+k/3 == 100)
```

```
        printf("%d,%d,%d\n", i,j,k);
```

```
}
```

循环次数=4

算法描述和算法分析

★算法时间效率度量方法

1. **事后统计**——先编写程序，之后利用计算机的计时功能，用一组或多组数据进行测试

缺点：①必须先编写算法相应程序

②所得时间依赖于计算机硬件、软件等因素，掩盖了算法本身的优劣

2. **事前分析估计**

程序在计算机上运行所消耗的时间取决于：

- ① 算法策略
- ② 问题规模
- ③ 程序语言
- ④ 编译质量
- ⑤ 计算机速度

显然，同一算法用不同的语言、不同的编译程序、在不同的计算机上运行，效率不同——所以使用“绝对时间”单位衡量算法效率不合适

算法描述和算法分析

★时间复杂度

- ✓ 算法中，语句执行次数越多则算法耗费时间越长。
- ✓ 语句执行次数称为**语句频度**或**时间频度**，记为 **$T(n)$** 。

$$T(n) = \sum \text{语句执行次数} \times \text{该语句执行时间} \\ \approx \sum \text{语句执行次数}$$

- ✓ 该方法可独立于机器的软件、硬件系统来分析算法在效率方面的优劣

```
x=0;
y=0;
for(k=0; k<2*n; k++)
    x++;
for(i=0; i<n; i++)
    for(j=0; j<n; j++)
        y ++;
```

执行次数

1

1

$2n+1$

$2n$

$n+1$

$n^*(n+1)$

n^2

时间耗费

$$T(n) = 4 + 6n + 2n^2$$

$$\lim_{n \rightarrow \infty} T(n) / n^2 = 2$$

n 充分大时， $T(n)$ 与
 n^2 在数量级上相同，
记 **$T(n) = O(n^2)$**

算法描述和算法分析

★时间复杂度

✓ **时间复杂度**：算法耗用时间相对问题规模 n 的**增长率**，即基本操作重复执行次数的阶数。

✓ **大O表示法**： $T(n)=O(f(n))$

加法规则 $O(f(n))+O(g(n))=\max\{O(f(n)),O(g(n))\}$
 $O(f(cn))=O(f(n))$, c 为正整数

乘法规则 $O(f(n))*O(g(n))=O(f(n)*g(n))$

```
x=0;
```

```
y=0;
```

```
for(k=0; k<2*n; k++)
```

```
    x++;
```

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

```
    for(j=0; j<n; j++)
```

```
        y ++;
```

执行次数

1

1

2n+1

2n

n+1

$n*(n+1)$

n^2

时间耗费

$T(n) = 4 + 6n + 2n^2$

$\lim_{n \rightarrow \infty} T(n) / n^2 = 2$

n 充分大时， $T(n)$ 与
 n^2 在数量级上相同，
记 $T(n)=O(n^2)$

算法描述和算法分析

★时间复杂度

✓ **时间复杂度**：算法耗用时间相对问题规模 n 的 **增长率**，即基本操作重复执行次数的阶数。

✓ **大O表示法**： $T(n)=O(f(n))$

加法规则 $O(f(n))+O(g(n))=\max\{O(f(n)),O(g(n))\}$
 $O(f(cn))=O(f(n))$, c 为正整数

乘法规则 $O(f(n))*O(g(n))=O(f(n)*g(n))$

```
x=0;
```

```
y=0;
```

```
for(k=0; k<2*n; k++)
```

```
    x++;
```

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

```
    for(j=0; j<n; j++)
```

```
        y ++;
```

$T_1(n) = O(1)$

$T_2(n) = O(n)$

$T_3(n) = O(n^2)$

$$\begin{aligned} T(n) &= T_1(n) + T_2(n) + T_3(n) \\ &= O(\max(1, n, n^2)) \\ &= O(n^2) \end{aligned}$$

频度最大语句重复
执行次数的阶数

算法描述和算法分析

★时间复杂度

✓ **时间复杂度**：算法耗用时间相对问题规模 n 的**增长率**，即基本操作重复执行次数的阶数。

✓ **大O表示法**： $T(n)=O(f(n))$

加法规则 $O(f(n))+O(g(n))=\max\{O(f(n)),O(g(n))\}$
 $O(f(cn))=O(f(n))$, c 为正整数

乘法规则 $O(f(n))*O(g(n))=O(f(n)*g(n))$

例 $n \times n$ 矩阵相乘

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

$$T(n) = O(n^3)$$

算法描述和算法分析

★时间复杂度

✓ **时间复杂度**：算法耗用时间相对问题规模 n 的**增长率**，即基本操作重复执行次数的阶数。

✓ **大O表示法**： $T(n)=O(f(n))$

✓ **常用时间复杂度**：

$O(1)$ ——常量型

$O(n)$ 、 $O(n^2)$ 、 $O(n^3)$ ——多项式型

$O(\log_2 n)$ 、 $O(n\log_2 n)$ ——对数型

$O(2^n)$ 、 $O(e^n)$ ——指数型



$O(1)$ $O(\log_2 n)$ $O(n)$ $O(n\log_2 n)$ $O(n^2)$ $O(n^3)$ $O(n^k)$ $O(2^n)$

★ **空间复杂度**： $S(n)=O(f(n))$

辅助存储空间增长率

Back

本章小结

- 掌握数据结构、算法的基本概念及术语
- 了解算法的时间复杂度和空间复杂度分析

End