

# 第一章 概论

## 1. 为什么要学习数据结构?

(什么是数据结构?)

- 为什么要学?
- 学习哪些内容?
- 怎么学?

## 2. 基本概念和术语

## 3. 数据类型和抽象数据类型

## 4. 算法描述与算法评价

## 5. 总结

# 1.1 为什么要学习数据结构？

摘自 <http://topic.csdn.net/u/20111211/11/aa2f9efe-916b-4f1c-9fbb-43a7448b38fb.html?87474>》

[Keeya0416](#): 虽然不用重复造轮子,但我们起码得知道轮子为什么做成圆的。

[Karnonlm](#): 会不会算法的区别,就是年薪10万和30万的区别。那种别人都已经做过的东西,拿来用用堆代码谁都会。真正有价值的程序都是从来没有人做过的,比如各种语言中第一个实现那些数据结构的人,这些人都是30万年薪都招不到的。各种应用中都有新的具体的未被人解决过的问题,这时就需要精通算法的人去解。这些牛人解出来后提供了API,给别人调用。

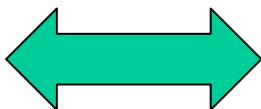
明白了么? 你觉得那些算法不重要,都有现成的,就是因为你现在只停留在堆别人写好的API的10万年薪的水平,而不是大牛的水平。

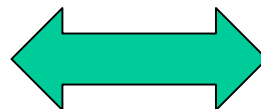
[AlexTuring](#): 总有人说数据结构用处不大.我感觉吧 这东西就是一种思想.时刻贯穿在你的代码中.一个写程序10年的人和一个初学者写同样要求的程序,老手写的代码就是不一样.他们的差距就在于对编程和算法的思想理解深度不一样吧

[awen12345678](#): 打个比方,要做一件普通的家具,我们选用通用的各种板材和相应的配件就可以完成,就像宜家的家具。要制作一件杰出的家具,如果你依然只能选用通用的板材和配件,你成功的概率就会很低。如果你自己会设计和制作相应的板材和配件,你的成功率就会高很多。软件制作中。MFC, STL库, 现存的各种链表等等就像是板材。你可以用他做出一些程序,但是也必然会有一些局限。学好数据结构,你就有机会自己做板材了。

# 未来考虑

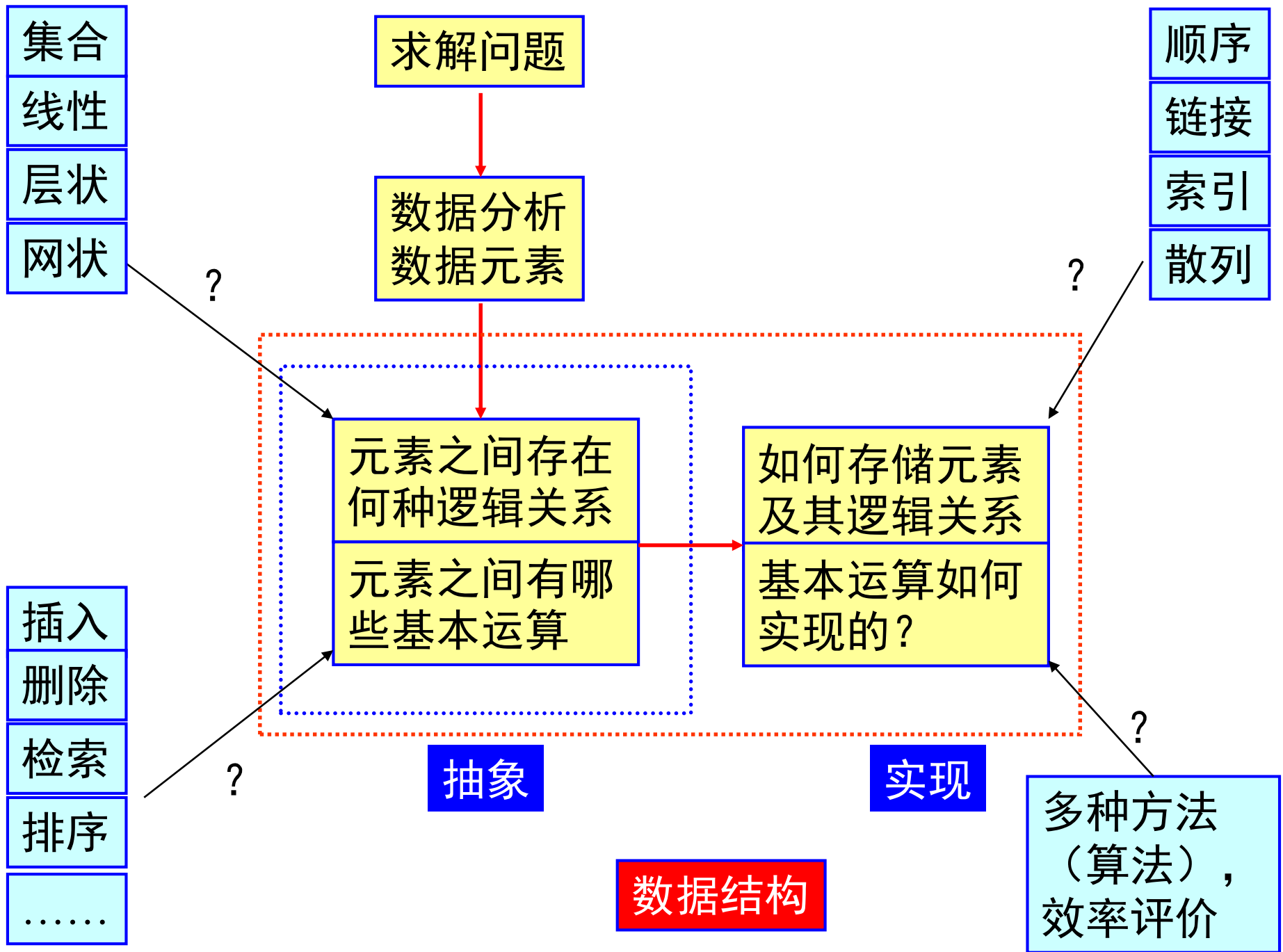
- 科研：总要碰到计算、分析，模型建立。
- 有了创新想法，自己不会写程序。

I  P

P  I

如果会，自己的想法、模型很容易实现。  
有问题，随时修改。时间、效率、准确！

- 计算机处理的对象：**数据**
- 数据之间存在什么样的关系？（**逻辑**上考虑）  
如何有效组织数据？（内在关系）
- 如何有效**存储**数据？（数据+关系）
- 数据之间有哪些**运算**？（运算与实现）  
这些运算如何实现的？



# 1. 计算机解决问题的过程

- 问题抽象，建立求解数学模型 [分析阶段]
- 设计相应的数据结构和算法 [设计阶段]
- 编写程序 [编码阶段]
- 上机调试、测试，得到最后结果 [测试与维护阶段]

# 2. 数值计算与非数值计算

- 对于数值计算问题，处理的对象为简单的数值，数学模型为数学方程；
- 对于非数值计算问题（如资料查询、交通管理等），处理的对象之间具有一定的逻辑关系，其数学模型不能简单地用数学方程描述，此时必须根据对象之间的逻辑关系建立描述问题的数据结构。

### 3. 两个非数值计算问题的描述

- 信息管理（图书、档案、职工等）
- 交通管理（城市交通管理、航线、铁路、公路等）

关键：

数据的表示（数据结构问题）和数据的处理（算法问题）

#### 线性问题

每人一个记录，多个数据项；  
什么样的逻辑关系？  
如何存储？  
什么样的操作？  
（统计、检索、排序等问题）

编号	姓名	性别	年龄	月收入
1	李泉	男	51	9800
2	王怡	女	47	9450
3	张三	男	35	8700
4	马小丁	男	27	8400
...	...	...	...	...

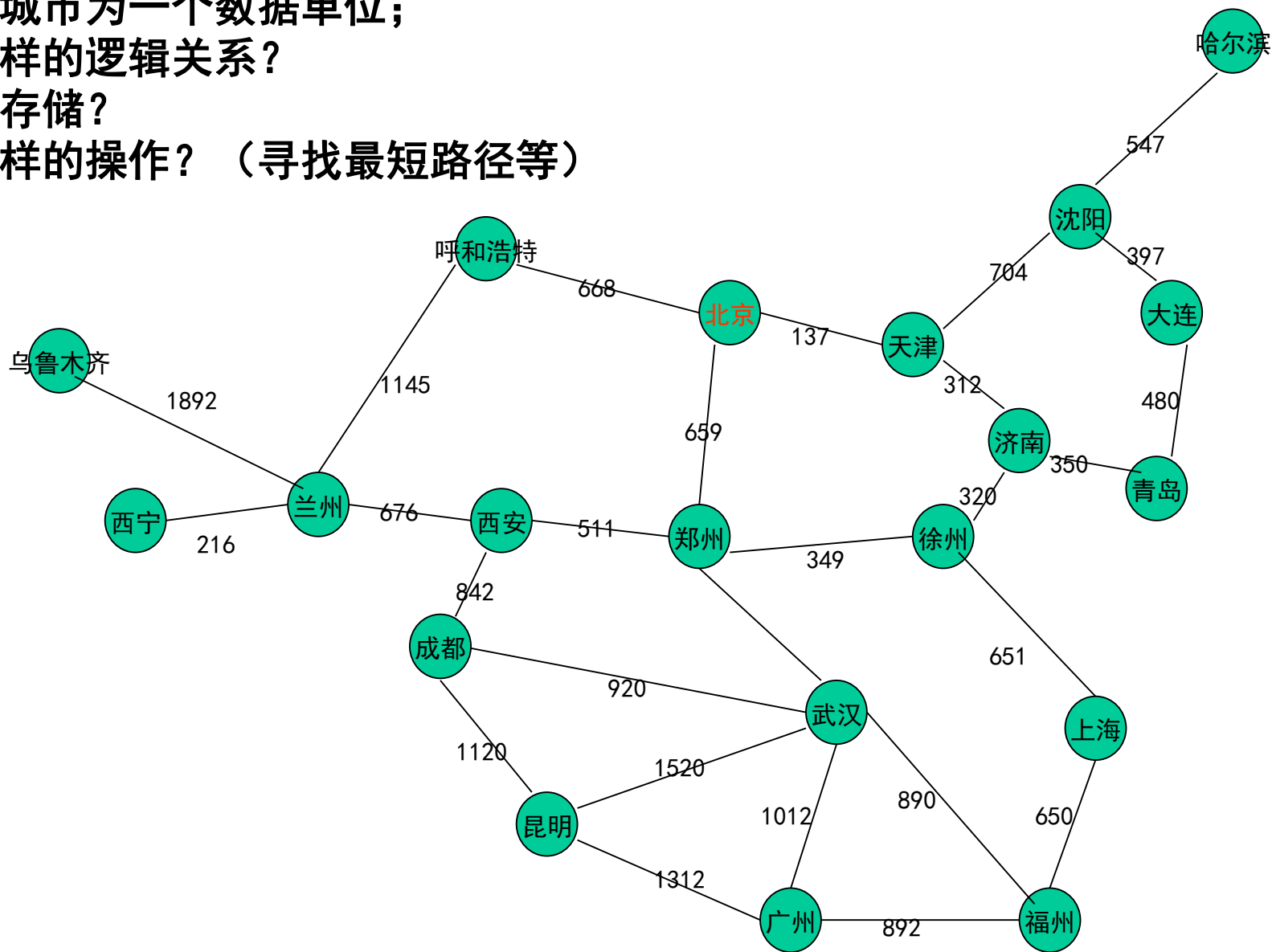
# 非线性问题

每个城市为一个数据单位；

什么样的逻辑关系？

如何存储？

什么样的操作？（寻找最短路径等）





## 交叉路口交通信号灯的管理：

**(1) 分析：**通过分析所有可能的行驶路线，分成若干组。对分组的要求是：任一个组内各个方向行驶的车辆可以同时安全行驶而不发生碰撞。

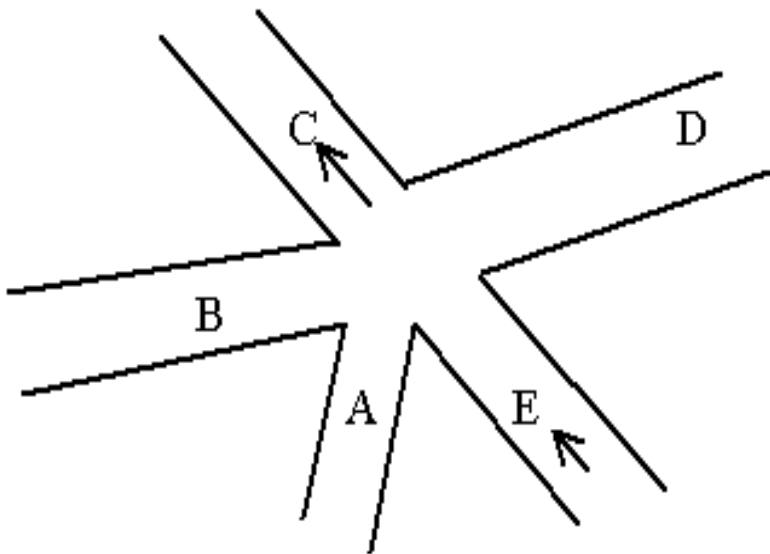


图 1.1 一个交叉路口的模型

根据这个路口的实际情况可以确定13个可能通行方向：

$A \rightarrow B$ ,  $A \rightarrow C$ ,  $A \rightarrow D$ ,  
 $B \rightarrow A$ ,  $B \rightarrow C$ ,  $B \rightarrow D$ ,  
 $D \rightarrow A$ ,  $D \rightarrow B$ ,  $D \rightarrow C$ ,  
 $E \rightarrow A$ ,  $E \rightarrow B$ ,  $E \rightarrow C$ ,  
 $E \rightarrow D$ 。

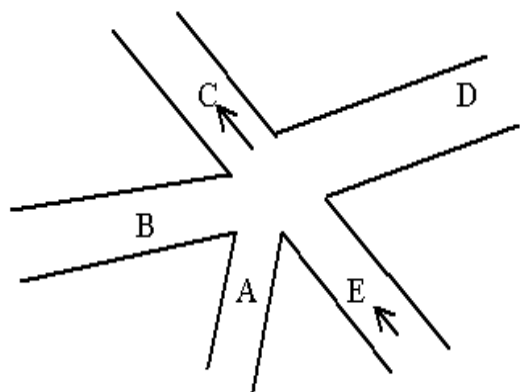


图 1.1 一个交叉路口的模型

**(2) 抽象：**为了叙述方便，把 $A \rightarrow B$ 简写成 $AB$ ，并且用一个小椭圆把它框起来，在不能同时行驶的路线间画一条连线（表示它们互相冲突），便可以得到图1.2所示的图式。

**对象：**通行方向  
**关系：**不能同时行驶  
**多对多的网状**  
**存储：**对象+关系  
**运算：**哪些基本运算  
 如何通过基本  
 运算实现问题求解？

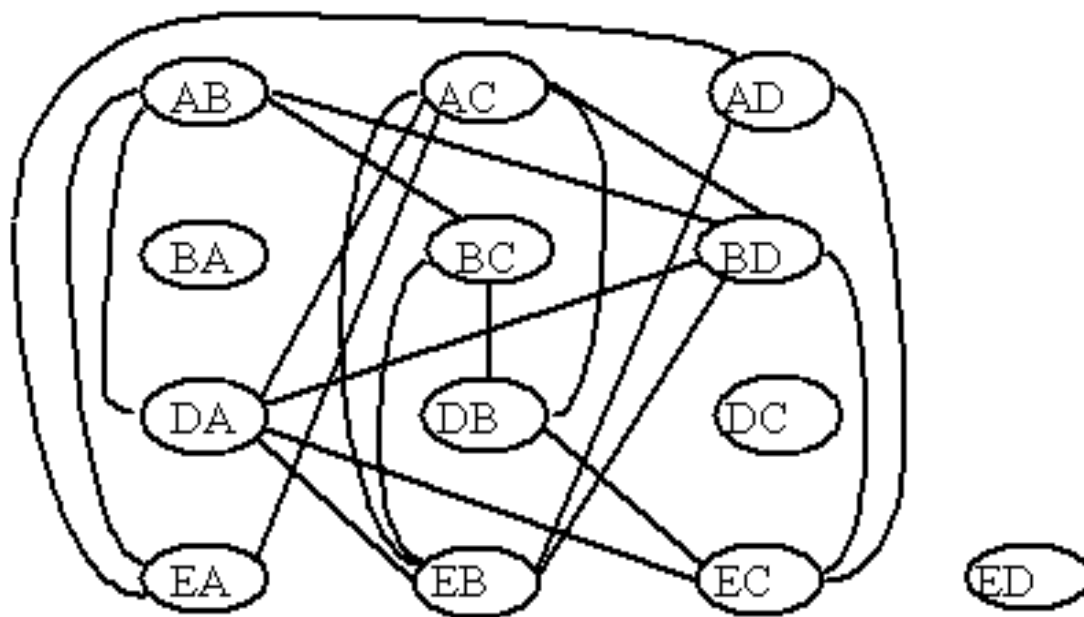


图 1.2 交叉路口的图式模型

- 这样做就把要解决的问题借助图的模型变成了一个抽象问题：将图1.2中的结点分组，使有线相连(互相冲突)的结点不在同一个组里。
- 如果把上图中的一个结点理解为一个国家，结点之间的连线看作两国有共同边界，上述问题就变成著名的“**着色问题**”：即求出要几种颜色可将图中所有国家着色，使得任意两个相邻的国家颜色都不相同。

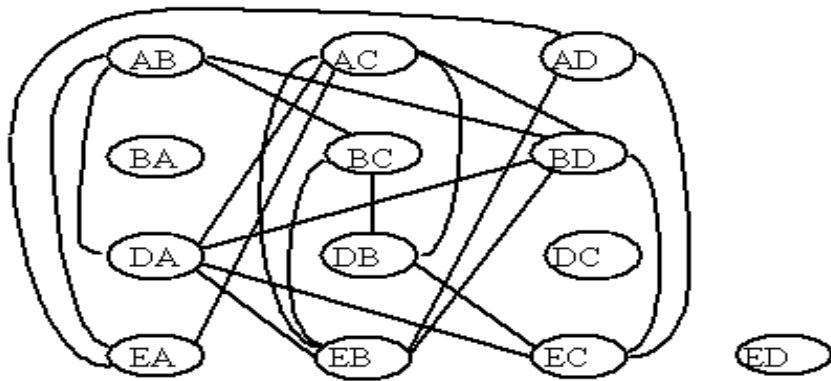


图 1.2 交叉路口的图式模型

### (3) 算法:

- 穷举法，贪心法，.....
- 假设需要着色的图是G，集合V1包括图中所有未被着色的结点，着色开始时V1是G所有结点集合(用G.V表示)。NEW表示每次分组中，已确定可以用新颜色着色的结点集合。
- 从V1中找出可用新颜色着色的结点集的工作可以用下面的程序框架描述：

**置NEW为空集合;**

**for 每个  $v \in V_1$  do**

```
{
    if v与NEW中所有结点间都没有边:
    {
        从V1中去掉v;
        将v加入NEW;
    }
}
```

#### (4) 程序:

```
int colorUp(Graph G)
{
    int color = 0;
    V1 = G.V;
    while(!isEmpty(V1))
    {
        emptySet(NEW);
        while (v ∈ V1 && notAdjacentWithSet(NEW, v, G))
        {
            addToSet(NEW, v);
            removeFromSet(V1, v);
        }
        ++color;
    }
    return (color);
}
```

#### (5) 结果:

把这种方法应用到关于交叉路口的例子中，可能得到如下一种分组：

颜色1:      **AB AC AD BA DC ED**  
颜色2:      **BC BD EA**  
颜色3:      **DA DB**  
颜色4:      **EB EC**

可行解  
最优解  
次优解

V1

1	0	1	0	...	1	1	0	0	0
---	---	---	---	-----	---	---	---	---	---

关系矩阵  
N\*N

0	0	1	0	...					
0	0	1	0						
1	1								
0	0								
...									
0									
0									
0									

New

1	3	...							
---	---	-----	--	--	--	--	--	--	--

## 4. 学习数据结构要解决的问题

- 要描述非数值计算问题，单靠数学方程是无法解决的，它涉及表、图、树等类的数据结构。因此，数据结构课是一门研究非数值计算问题的程序设计中计算机的操作对象以及它们之间的关系和运算操作等的一门学科。

### 总结：

- 常见的计算机语言并不支持图、树、集合等数据结构，通常只提供基本的数据类型（整数、实数等）和一些数据构造手段（如数组、结构、指针等）。因此，复杂数据结构的设计以及相应的操作必须有用户自己实现。
- 用计算机求解问题，首先分析问题的需求，生成抽象模型，然后设计适当的数据结构和有关的算法，最后采用计算机编程语言精确地描述所需要的数据和算法，实现程序。

## 5. 《数据结构》的定位

- 《图论》 -- 《离散数学》 -- 《数据结构》
- 计算机专业中介于数学、硬件和软件三者之间的核心课程；
- 程序=算法+数据结构（瑞士著名计算机科学家N.Wirth）
- 选择一个好的数据结构，设计一个好的算法
- 简单的计算机应用体现不出好“数据结构”的优点，但随着问题的复杂，数据结构的选择成为一个重要的问题。
- 一个简单问题的两种设计方法（数组与动态分配问题）。
- 数据结构课程不仅仅是掌握各种常用的数据结构，更重要的是能够根据要解决的问题合理地选择数据结构，设计出时间、空间高效的算法。积累编程经验、提高程序设计的水平。

## 1.2 基本概念和术语

### 1. 数据、数据元素、数据对象、数据类型

- **数据**：指能够被计算机识别、存储和加工处理的信息载体。
- **数据元素**：就是数据的基本单位，在某些情况下，数据元素也称为元素、结点、顶点、记录。数据元素有时可以由若干**数据项**组成。
- **数据对象**：性质相同的数据元素集合，是数据的一个子集。
- **数据类型**：是一个值的集合以及在这些值上定义的一组操作的总称。

原子数据类型（不可分解）

结构数据类型（可分解，原子、结构的组合）



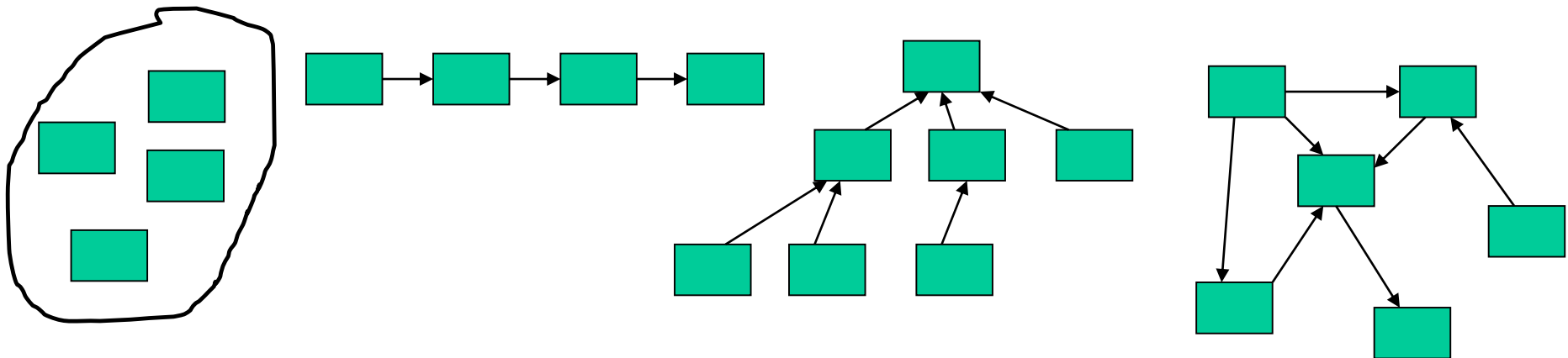
## 2. 数据结构

- **数据结构**：指的是数据之间的相互关系，即数据的组织形式。一般包括三个方面的内容:数据的**逻辑结构**、**存储结构**和**数据的运算**。
- **逻辑结构**：数据元素之间的逻辑关系； 二元组表示：(D, R)
- **存储结构**：数据元素及其关系内存中的映像与表示；
- **数据运算**：定义在逻辑结构上的一系列操作以及这些操作在存储结构上的实现；数据的运算是定义在逻辑结构上的，而具体的实现是基于存储结构。

### 3. 四种逻辑关系

- **集合**：数据元素仅仅“同属于一个集合”
- **线性结构**：数据元素之间仅存在一对一的关系（唯一后继、唯一前驱）；
- **树形结构**：数据元素之间存在一对多的关系（层状结构，唯一前驱，多个后继）；
- **图状或网状结构**：数据元素之间存在多对多的关系，元素之间的关系是任意的（多个前驱，多个后继）；

[集合、线性表、字符串、栈与队列、树与二叉树、字典、图]



## 4. 四种存储结构

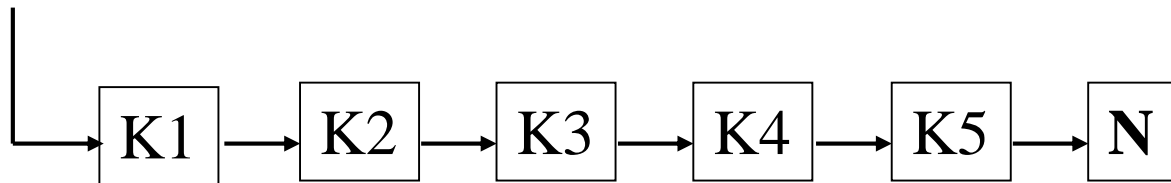
- **顺序存储结构**：它是把逻辑上相邻的结点存储在物理位置相邻的存储单元里，结点间的逻辑关系由存储单元的邻接关系来体现。
- **链接（链状）存储结构**：它不要求逻辑上相邻的结点在物理位置上亦相邻，结点间的逻辑关系是由附加的指针字段表示的。
- **索引存储结构**：除建立存储结点信息外，还建立附加的索引表来标识结点的地址。
- **散列存储结构**：就是根据结点的关键字（数值）直接计算出该结点的存储地址。

四种存储结构既可单独使用，又可组合使用

## 顺序存储

A[0]	K1		
A[1]	K2		
A[2]	K3		
A[3]	K4		
A[4]	K5		

## 链接存储



## 索引存储

K1	0	3
K4	3	2

最小key

起始位置

元素个数

## 散列存储

$$D = (K_i - 100) \% 10$$

0	K3		
1			
2	K5		
3	K1		
4	K2		
5	K4		

## 5. 数据的运算

- 定义在逻辑结构上的一系列操作以及这些操作在存储结构上的实现；数据的运算是定义在逻辑结构上的，而具体的实现是基于存储结构。
  - 常用的运算：检索、插入、删除、定位、修改、排序等；
- 

本课程中讨论的各种数据结构皆按照三个方面进行：

- 逻辑定义（逻辑结构）
- 存储结构及其各种运算的实现

# 1.3 数据类型和抽象数据类型

## 1. 数据类型

- 具有相同性质的计算机数据的集合及在这类数据上的一组运算。
- 隐含规定了程序执行期间变量的取值范围和允许的操作；
- 如整型：[-minint, maxint]； +, -, \*, /, %
- 原子类型：int, char, float,....不可分解
- 结构类型：struct A { .....}, 可以分解

## 2. 抽象数据类型 (Abstract Data Type, ADT)

- **定义：**一个数学模型以及定义在该模型上的一组操作。  
抽象数据类型的定义仅取决于它的一组逻辑特性，而与其在计算机内的表示和实现无关。即：不论其内部结构如何变化，只要其数学特征不变，都不影响其外部使用。
- **抽象数据类型的三元组表示：**  $(D, R, P)$   
D:数据对象； R:D上的关系集合； P:对D的基本操作集合

ADT 抽象数据类型 {  
    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

逻辑结构+基本操作运算定义

# 线性表的抽象数据类型

ADT LIST {

**数据对象:**  $D = \{ a \mid a_i \in \text{elemset}, i = 1, 2, \dots, n; n \geq 0 \}$

/\* 定义线性表, 具有n个数据元素的有限序列 \*/

**数据关系:**  $R = \{ \langle a_{i-1}, a_i \rangle, a_{i-1}, a_i \in D, i = 1, 2, \dots, n \}$

/\* 说明线性表数据元素之间的相互关系 \*/

**基本操作:**

SetNull(L)    初始条件: 线性表L存在

运算结果: 将线性表L置空;

Length(L)    初始条件: 线性表L存在

运算结果: 返回线性表L的数据元素个数 (表长) ;

Get (L, i)

Locate(L, x)

Insert(L, x, i)

Delete(L, i)

} ADT LIST



**ADT circle is**  
**data**

float r ;

**operations**

void constructor( ) 处理： 构造一个圆

float area ( )

{ return(3.14\*r\*r); }

float circumference( )

{ return(2\*3.14\*r); }

**end ADT circle;**

# 1.4 算法描述与算法评价

## 1. 算法的描述

- 算法是对特定问题求解**方法和步骤**的一种描述，它是指令的一组有限序列，其中每个指令表示一个或多个操作。
- **算法+数据结构=程序**：揭示了算法与数据结构的关系
- 选择了一个好的数据结构，还需要一个好的算法，才能更好地解决问题。

## 2. 算法的五个重要特性

- **输入**：0或多个外界输入，初始值
- **输出**：一个或多个输出
- **有穷性**：有限步完成（**程序可以不满足该条件**）
- **确定性**：每条指令有明确的含义，无二义性，相同的输入得到相同结果
- **可行性**：算法必须能执行，所有的操作都可以通过已经实现了的基本运算执行有限次来实现。

### 3. 描述算法的工具

- 自然语言，数学语言或约定的符号描述；
- 计算机高级语言描述：C、Pascal、C++、Java...

### 4. 算法的设计要求

解决问题：选择恰当的数据结构，设计一个算法，再进行编程实现。如何设计一个好的算法？

- **正确性**：经得起一切输入数据的考验；
- **可读性**：让人阅读得懂，便于交流。注意注释行的作用；
- **健壮性**：输入数据错误时，进行必要的处理。不能得到莫名其妙的结果；
- **高效率**：执行时间尽可能地短，对存储要求尽可能地少，既省时又节省空间。

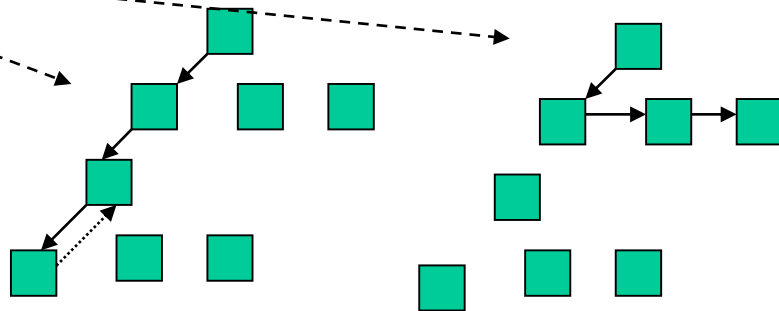
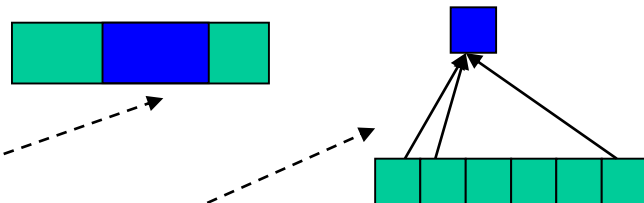
综合考虑，时间、空间往往相互抵制，快速往往需要高存储要求，低存储要求往往导致执行时间效率下降。时间换空间，空间换时间...

## 5. 算法的分类

试探，穷举

- **贪心法** [分步完成，局部最优得到整体最优，前面的着色问题求解]
- **分治法** [问题规模缩小，分而治之。如“字典”中的折半检索算法等]
- **动态规划法** [问题分解（缩小规模），得到各个分解结果，再自底往上求最后结果。如“图”中的Floyd求最短路径算法]
- **回溯法** [彻底搜索，深度优先试探求得。如迷宫问题求解。]
- **分支界限法** [彻底搜索，广度优先试探求得。如农夫过河问题求解]

教材第九章详细讨论。



## 6. 算法的评价指标

同一个问题，可以设计出多个算法，这些算法的优劣如何衡量？

- **时间效率**：执行算法耗费的时间；
- **空间效率**：运行算法需要耗费的存储空间，主要考虑辅助空间；
- 理解、阅读、编写、调试的难易等。

## 7. 时间效率度量方法

- 算法的执行时间=简单操作执行次数  $\times$  执行简单操作所需时间 = 算法中每条语句的执行时间之和

- 每条语句的执行时间 = 语句执行次数 x 语句执行时间
- 语句执行时间与硬件有关，与算法本身无关。因此，算法的执行时间主要取决于算法中各个语句的执行次数（执行频率）。

例如：

```

for (i = 1; i <= n; i++)                /* n+1次          */
{
    for (j = 1; j < n; j++)              /* nx(n+1)次      */
    {
        c[i][j] = 0;                    /* nxn次          */
        for (k = 0; k < n; k++)          /* nxnx(n+1)次    */
        {
            c[i][j] = c[i][j]+a[i][k]*b[k][j]; } /* nxnxn次        */
    }
}

```

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

是问题规模 $n$ 的函数，当  $n \rightarrow \infty$  , $T(n)$ 与  $n^3$  同阶或同数量级，  
表示为：

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

① 算法的时间效率度量方法：基本操作的重复执行次数。

基本操作的重复执行次数与问题的规模 $n$ 有关，是问题规模 $n$ 的某个函数 $f(n)$ 。因此，算法的时间复杂度记为： $T(n) = O(f(n))$ 。它表示随着问题规模 $n$ 的增大，算法的执行时间的增长率和 $f(n)$ 的增长率相同。

如果存在正常数 $c$ 和 $n_0$ ，当问题规模 $n \geq n_0$ 后， $T(n) \leq c * f(n)$   
当 $n$ 充分大时， $T(n)$ 不大于 $f(n)$ 的一个常数倍。

② 如何选择基本操作？ 循环最内层的原操作

③ 常用的时间复杂度，按数量级递增排列依次为：

常数阶 $O(1)$ 、对数阶 $O(\log_2 n)$ 、线性阶 $O(n)$ 、线性对数阶 $O(n \log_2 n)$ 、平方阶 $O(n^2)$ 、立方阶 $O(n^3)$ 、 $k$ 次方阶 $O(n^k)$ 、指数阶 $O(k^n)$ 。

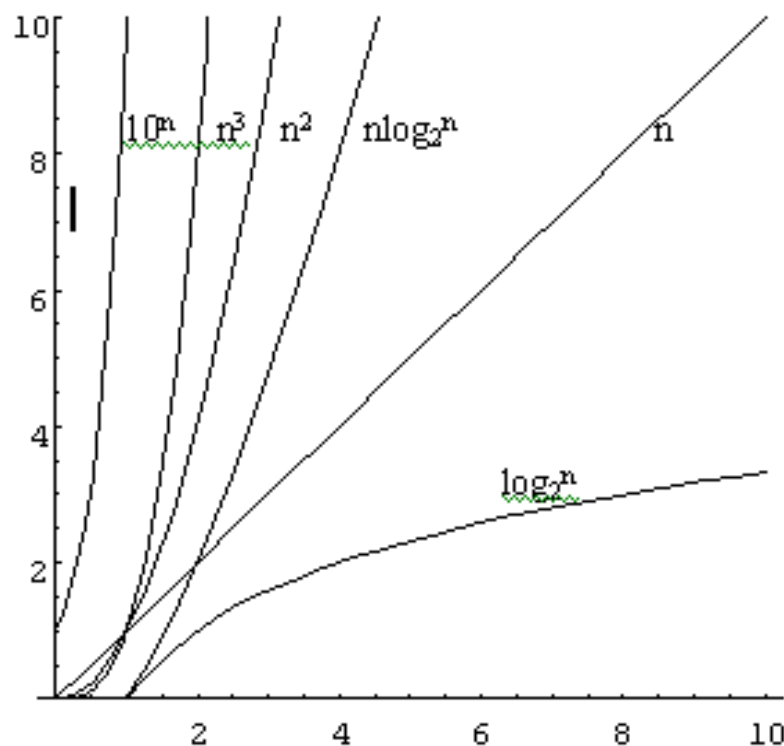


图 1.3  $f(n)$  函数曲线变化速度的比较

#### ④ O表示法的运算：

✓ 加法

$$O(f1(n)) + O(f2(n)) = O(f1(n) + f2(n)) = O(\max(f1(n), f2(n)))$$

✓ 乘法

$$O(f1(n)) * O(f2(n)) = O(f1(n) * f2(n))$$

- ⑤ 有些算法（如排序等）基本操作的执行次数除了与问题的规模n有关外，还与输入数据有关。此时，往往用基本操作的**平均执行次数**衡量算法的时间效率。另外，也常常用算法在输入数据集最坏情况下，基本操作的**最多执行次数**作为算法的时间效率度量。

## 8. 空间效率度量

— **空间复杂度**：  $S(n) = O(f(n))$

— **算法的存储空间** = 存储算法的空间 + 算法的输入、输出数据占用空间 + 程序执行过程占用的临时辅助空间

因此，讨论算法的空间效率主要关心执行算法需要的额外辅助空间的数量。



(1)

```
for (i=0;i<n;i++)  
  for (j=0;j<n;j++)  
    A[i][j]=0;
```

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

(3)

```
i=s=0;  
while (s<n)  
{ i++; s+=i; //s=s+i }
```

$T(n)= O(n^{1/2})$

(2)

```
s=0;  
for (i=0;i<n;i++)  
  for (j=0;j<n;j++)  
    for (k=0;k<n;k++)  
      s=s+B[i][j][k];  
sum=s;
```

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

(4)

```
i=1;  
while (i<=n) i=i*2;
```

$T(n)= O(\log_2 n)$

(5)

```
x=91; y=100;  
while (y>0)  
{  
    If (x>100) { x=x-10;  y--; }  
    else x++;  
}
```

**$T(n) = O(1)$**

◇ 这个程序看起来有点吓人，  
总共循环运行了1000次，但是我们看到这段程序的运行是和n无关的，  
就算它再循环一万年，我们也不管它，只是一个常数阶的函数。

# 一个数据结构示例

学生成绩表，记录了一个班的学生各门课的成绩。按学生的姓名为一行记成的表。这个表就是一个数据结构。每个记录(有姓名，学号，成绩等字段)就是一个元素，对于整个表来说，只有一个起始元素(它的前面无记录)和一个终止元素(它的后面无记录)，其它的元素则各有一个也只有一个直接前趋和直接后继(它的前面和后面均有且只有一个记录)。这几个关系就确定了这个表的**逻辑结构（线性结构，线性序列）**。

那么我们怎样把这个表中的数据存储在计算机里呢？用高级语言如何表示各结点之间的关系呢？是用一片连续的内存单元来存放这些记录(如用数组表示)还是随机存放各结点数据再用指针进行链接呢？这就是**存储结构**的问题，我们都是从高级语言的层次来讨论这个问题的。

最后，我们有了这个表（数据结构），肯定要用它，那么就是要对这张表中的记录进行查询、修改、删除等操作，对这个表可以进行哪些操作以及如何实现这些操作就是**数据的运算**问题了。

## 1.5 总结

- 求解问题的主要步骤
- 求解问题的两大类
- 什么是数据结构？
- 四类逻辑结构
- 四类存储结构
- 算法的设计取决于逻辑结构，实现依赖于存储结构
- 抽象数据类型（三元组表示）
- 算法的定义，算法与程序的主要区别
- 算法的描述方法
- 算法的五个重要特性和设计算法的四个基本要求
- 空间效率、时间效率的相互制约
- 时间效率的度量方法（基本操作的执行次数）
- 空间效率的度量方法（额外辅助空间的数量）

# 作业：

- P27：复习题+算法分析题，要求在阅读第一章的基础上回答。不要求写到作业本上，但一定掌握。
- 着色问题求解。
  - 上机前，完成源码编写，初步完成“上机报告”。
  - 上机完成后，提交“上机报告”。

上机实习报告的书写规范。

# 上机报告（1~2页，不需要源代码）

## 1) 总体描述

问题的理解（简单）

## 2) 设计与实现

数据元素的组织形式：数据结构

采用什么算法/方法？

什么语言（C/C++），哪种调试环境？

重要变量的简单描述。

## 3) 总结

收获：掌握了哪些知识？

问题与解决方法：碰到什么问题，如何解决的？

（包括：代码、编译和连接中碰到的错误、调试以及执行）