

哈工大计算机考研全套视频和资料，真题、考点、典型题、命题规律独家视频讲解！
详见：网学天地（www.e-studysky.com）；咨询QQ：2696670126

数据结构与算法

- 课程编号：T050307
- 英文译名：DATA STRUCTURE and ALGORITHMS
- 总学时：54/12
- 授课对象：计算机科学与技术 本科生
- 先修课程：高等数学、集合与图论、C程序设计
- 课程要求：必修课
- 课程分类：技术基础
- 授课教师：张 岩 李秀坤
- 答疑地点：综合楼513、D楼实验室
- 办公电话：86413213、86413341
- 课程网站：
ftp://cst.hit.edu.cn/.users/zhangyan/ftp_pub/student/DSA2004/

数据结构与算法

- 教学目的：
 - (1) 学会分析和研究计算机处理的数据对象的特性，掌握常用数据结构内在的逻辑关系、在机内的存储表示，掌握常用数据结构上的运算操作的动态性质和执行算法。
 - (2) 能够为实际应用选择适当的数据结构、存储结构和相应算法。
 - (3) 初步掌握算法性能的分析方法。
- 考核要求：
 - (1) 考试：70%；(2) 作业：10%；
 - (3) 实验：20%；(4) 缺勤：-10%

第1章 绪论

1.1 数据结构的研究对象

1.2 数据结构的发展概况

1.3 抽象数据类型(ADT)

1.4 逐步求精的程序设计方法

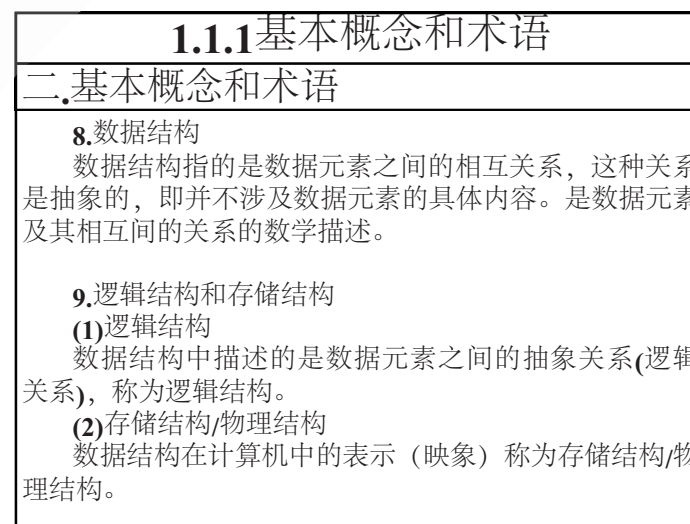
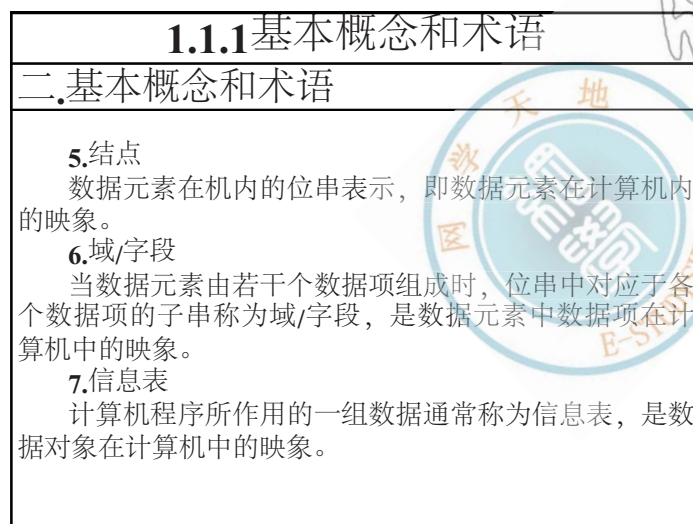
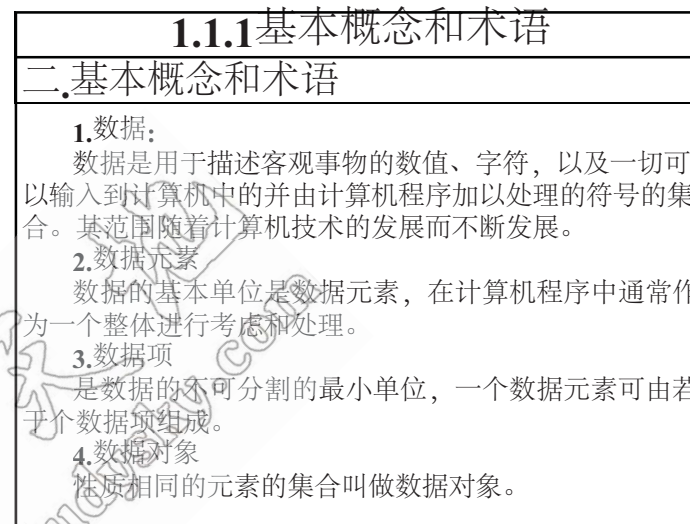
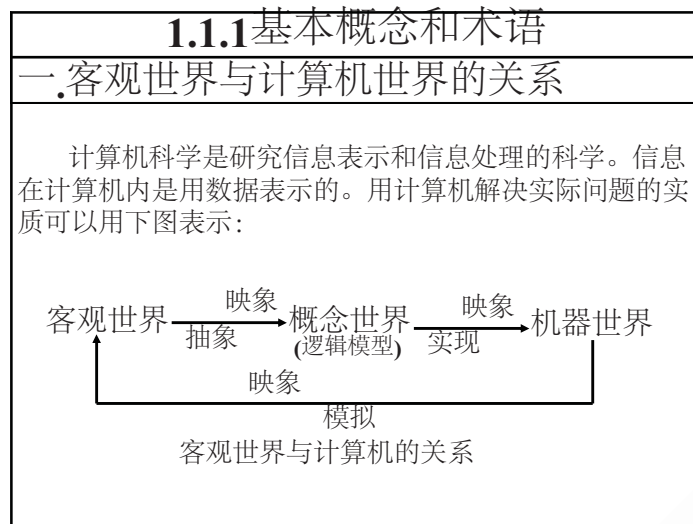
1.1 数据结构的研究对象

1.1.1 基本概念和术语

1.1.2 四种基本的数据结构

1.1.3 数据结构的研究对象

[返回](#)



哈工大计算机考研全套视频和资料，真题、考点、典型题、命题规律独家视频讲解！
详见：网学天地（www.e-studysky.com）；咨询QQ：2696670126

1.1.1基本概念和术语
二.基本概念和术语
<p>(3)数据元素之间的关系（逻辑结构）在计算机中有两种表示方法：顺序映象(表示)和非顺序映象(表示)，从而导致两种不同的存储结构：顺序结构和链式结构。</p> <p>顺序映象（表示）的特点是借助数据元素在存储器中的相对位置来表示数据元素之间的逻辑关系。</p> <p>非顺序映象（表示）的特点是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。</p>
返回

1.1.2四种基本的逻辑结构
1.集合结构
<p>结构中的数据元素之间除了属于同一个集合的关系之外，别无其他关系。关系比较松散，可用其他结构来表示。</p>
2.线性结构
<p>结构中的数据元素之间存在一个对一的关系，即线性关系，每个元素至多有一个直接前导和后继。</p>

1.1.2四种基本的逻辑结构
3.树型结构
<p>结构中的数据元素之间存在一个对多个的关系，即层次关系，即每一层上的元素可能与下层的多个元素相关，而至多与上层的一个元素相关。</p>
4.网状/图型结构
<p>结构中的数据元素之间存在多个对多个的关系，即任意关系，任何元素之间都可能有关系。</p> <p>一般将逻辑结构分为线性结构（2）和非线性结构（1，3，4）两种。</p> <p>结构中的数据元素之间存在多个对多个的关系，即任意关系，任何元素之间都可能有关系。</p>
返回

1.1.4数据结构的研究对象
数据结构的研究对象(研究内容)
<ol style="list-style-type: none">1.数据对象的结构形式,各种数据结构的性质(逻辑结构);2.数据对象和关系在计算机中的表示(物理结构/存储结构);3.数据结构上定义的基本操作(算法);4.算法的效率;5.数据结构的应用,如数据分类,检索.
返回

1.2 数据结构的发展概况

1. 程序设计方法学的发展极大地促进了数据结构的发展
 - (1) 计算机科学及其应用的发展时数据结构成为独立学科。
 - (2) 以数据为中心的程序设计方法推动了数据结构的发展。
- 面向过程的程序设计的特点是以程序为中心，侧重于建立程序，程序在简单的数据结构上进行复杂的运算，软件设计的主要工作就是设计求解问题的过程。注重于程序设计技巧，适合于数值计算。
- 结构化特别是面向对象的程序设计以数据(结构)为中心，系统采用复杂的数据结构来描述系统状态，程序围绕数据结构进行加工。这种编程语言更适合于非数值计算。
2. 数据结构在计算机科学中的地位
 - (1) 是计算机科学的一门专业基础和核心课程。
 - (2) 是学习、设计和实现操作系统、编译系统、数据库系统和其它应用系统的重要基础。

[返回](#)

1.3 抽象数据型(ADT)

1.3.1 抽象数据型的定义

1.3.2 数据型, 数据结构和ADT

1.3.3 抽象数据型的规格描述

1.3.4 抽象数据型的实现

1.3.5 多层次抽象技术

1.3.6 多层次抽象技术

[返回](#)

1.3.1 抽象数据型的定义

一. 抽象数据型的定义

1. 定义: 是一个数学模型和在该模型上定义的操作集合的总称。

注: ADT是程序设计语言中数据类型概念的进一步推广和进一步抽象。

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

2. ADT的优点

使用者只要知道这些操作的用途就可以编程序了；至于这些操作是怎样实现的，以及整型数在内存中是如何表示的，并不影响使用者所编程序的编码形式。

1.3.1 抽象数据型的定义

二. 抽象数据型的实现

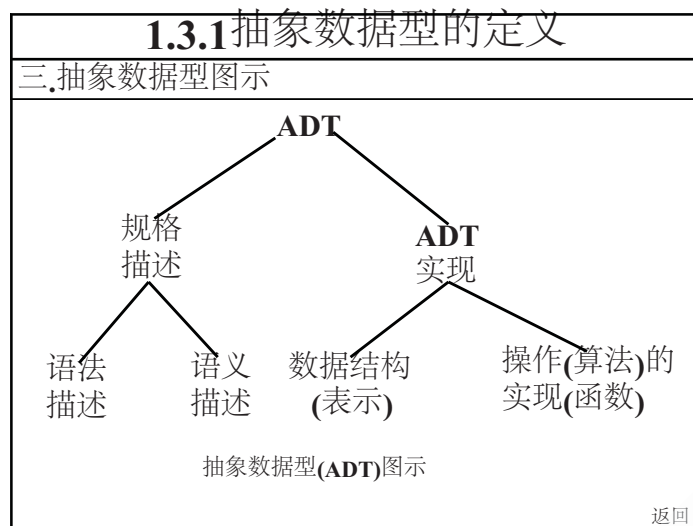
1. 抽象型实现的含义

就是将ADT转换成程序设计语言的说明语句，加上对应于该ADT中的每个操作的函数。换句话说，就是用适当的数据结构来表示ADT中的数学模型，并用一组函数来实现该模型上的各种操作。

2. 注意事项

(1) 同一数学模型上定义不同的操作集，则它们代表不同的ADT；

(2) 把ADT的描述与用某中程序设计语言实现ADT加以区别，这是大型程序设计中当前的发展趋势。如下图所示。



1.3.2 数据型、数据结构和ADT
1.三个概念的各自含义及相互关系
(1)各自含义 数据型是该类型变量的存储格式和所有可能取值的集合； 数据结构则是抽象数据型中数学模型的表示； 抽象数据型是一个数学模型及在该模型上定义的操作集的总称。
(2)相互关系 数据型是根据数据结构分类的，同类型的数据元素的数据结构相同； 数据结构是ADT中数学模型的表示； ADT是数据类型的进一步推广和进一步抽象。
2.信息聚集的三种方式
数组、结构（体）和文件
返回

1.3.3 抽象数据型的规格描述
一.ADT的规格描述
即ADT的形式化描述，包括语法规格和语义规格描述
二.规格描述的作用
(1)是程序操作/算法实现的依据； (2)是作为与程序并存的文档,用于程序的调试和维护。
三.规格描述原则
规格描述必须具有：
(1)完整性: 要能反映所定义的抽象数据型的全部特性 (2)统一性: 应是一个前后协调的整体，不应自相矛盾 (3)通用性: 所定义的抽象数据型应适用于尽量广泛的对象 (4)独立性: 应尽可能不依赖于程序语言

1.3.3 抽象数据型的规格描述
四. ADT的规格描述方法
1.语法规格描述
就是给出所指定的操作的名称以及输入输出类型；
2.语义规格描述
给出符合语法规定的表达式的语义,即表达形式所起的作用是什么或者效果是什么或者是让程序作什么(但不涉及;怎么做;)
3.ADT规格描述方法
首先描述它的定义域； 然后描述各个操作及其作用(给予法和语义规格描述)

哈工大计算机考研全套视频和资料，真题、考点、典型题、命题规律独家视频讲解！
详见：网学天地（www.e-studysky.com）；咨询QQ：2696670126

1.3.3抽象数据型的规格描述

四.ADT的规格描述方法

4.规格描述举例（ADT栈）

首先，ADT栈是同类单元Elementtype的集合；
然后，用语法和语义规格来描述各个操作及其作用，
并且体现ADT栈的后进先出LIFO的特征。

语法规格描述部分：

```
type Stack[Elementtype];  
NEWSTACK() → Stack,  
PUSH(Elementtype,Stack) → Stack,  
POP(Stack) → Stack ∪ {UNDEFINED},  
TOP(Stack) → Elementtype ∪ {UNDEFINED},  
EMPTY(Stack) → Boolean;
```

语法部分给出所指定的操作的名称以及输入输出类型

1.3.3抽象数据型规格描述

四.ADT的规格描述方法

4.规格描述举例（ADT栈）

语义规格描述部分：

```
declare stk:Stack,elm:Elementtype;  
POP(NEWSTACK)=NEWSTACK, //UNDEFINED  
POP(PUSH(elm,stk))=stk,  
TOP(NEWSTACK)=UNDEFINED,  
TOP(PUSH(elm,stk))=elm,  
EMPTY(NEWSTACK)=TRUE,  
EMPTY(PUSH(elm,stk))=FALSE;
```

语义部分主要有一组等式组成，严格规定了各种操作的功能及相互关系

1.3.3抽象数据型规格描述

四.ADT的规格描述方法

5.注意事项

- (1)对于同一ADT，在同样语法定义之下可以有不同语义定义；
- (2)不同的语义定义，将影响ADT的实现，因此要做到语义定义和实现的协调一致。

[返回](#)

1.3.4抽象数据型实现

一.ADT的实现原则

- (1)应符合规格描述的定义；
- (2)应有尽可能好的通用性；
- (3)应尽可能具有良好的独立性，在结构上应成为独立的模块；将内部细节屏蔽起来。

二.ADT的实现举例（ADT栈的带头结点的链式实现）

1)数据结构描述

```
enum boolean{FALSE,TRUE};  
struct node{  
    elementtype val;  
    node *next;  
}; //结点的类型  
typedef node *stack; //栈的类型
```

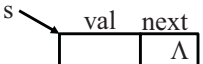
1.3.4 抽象数据型的实现

二.ADT的实现举例（ADT栈的带头结点的链式实现）

2) 基本操作的实现

```

stack NEWSTACK()
{
    stack s;
    s=new node;
    /*s=(node *)malloc(sizeof(node));*/
    s->next=NULL;
    return s;
}
    
```



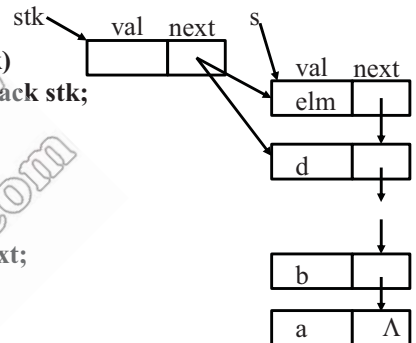
1.3.4 抽象数据型的实现

二.ADT的实现举例（ADT栈的带头结点的链式实现）

2) 基本操作的实现

```

void PUSH(elm,stk)
elementtype elm;stack stk;
{
    stack s;
    s=new node;
    s->val=elm;
    s->next=stk->next;
    stk->next=s;
}
    
```



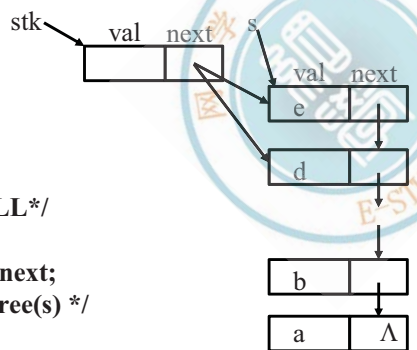
1.3.4 抽象数据型的实现

二.ADT的实现举例（ADT栈的带头结点的链式实现）

2) 基本操作的实现

```

void POP(stk)
stack stk;
{
    stack s;
    if (stk->next){
        /*stk->next!=NULL*/
        s=stk->next;
        stk->next=s->next;
        delete s; /* free(s) */
    }
}
    
```



1.3.4 抽象数据型的实现

二.ADT的实现举例（ADT栈的带头结点的链式实现）

2) 基本操作的实现

```

elementtype TOP(stk)
stack stk;
{
    if (stk->next)
        return (stk->next->val);
    else
        return NULLELE;
}
    
```

哈工大计算机考研全套视频和资料，真题、考点、典型题、命题规律独家视频讲解！
详见：网学天地（www.e-studysky.com）；咨询QQ：2696670126

1.3.4 抽象数据型的实现

二.ADT的实现举例（ADT栈的带头结点的链式实现）

2) 基本操作的实现

```
boolean EMPTY(stk)
stack stk;
{   if (stk->next)
    return FALSE;
    else
    return TRUE;
}
```

[返回](#)

1.3.5 多层次抽象技术

1. 逐层抽象方法

对于较复杂的数据类型，先将较简单、较基本的数据类型抽象出来，给出定义；再用已定义的数据类型去定义更复杂的数据类型，完成对后者的抽象。就是说，用已定义的类型来表述正要定义的数据类型的定义域，并用前者的操作来表述后者的操作这就是所谓逐层抽象的方法。

逐层抽象实质是用已知的简单数据类型定义更复杂的数据类型。

1.3.5 多层次抽象技术

2. 优点

(1) 由于在定义高层数据类型时不必考虑低层数据类型及其操作的内部细节，因而对复杂数据类型进行抽象可以简化许多琐事。

(2) 多层次抽象化通常可以采用自底向上的方式进行，先抽象出最基本的数据类型，然后利用它们定义上一层数据类型，如此逐层向上，直至到达最高层的数据类型为止，这样可以防止低层次倒过来引用高层数据类型，保证整个系统的有序层次结构。

3. 缺点

多层次抽象化的最终目的是建立最高层的数据类型，因此低层应该服从高层的要求。自底向上方式是底层的抽象带有一定的盲目性，在抽象过程中，可能从高层返回低层作修正，也就是不得不穿插一些自顶向下的过程。

[返回](#)

1.3.6 抽象数据型的优点

抽象数据型的优点

采用抽象数据类型的方法进行软件（特别是大型软件）系统的设计，具有许多明显的优点：

首先，它降低了软件设计的复杂性。

其次，抽象数据类型可提高程序的可读性和可维护性。

第三，由于抽象数据类型的使用降低了程序的复杂度，使程序的各部分相对分离，因而程序的正确性容易得到保证。

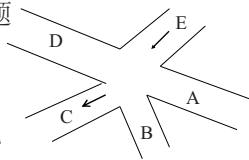
第四，有利于软件重用。

[返回](#)

1.4 逐步求精的程序设计方法
1.4.1 如何求解一个问题
<p>一. 算法的定义</p> <p>1. 计算 能由一个给定的计算模型机械地执行的规则(或步骤)序列称为该计算模型的一个计算。 注：一个计算机程序是一个计算(计算模型是计算机)； 计算可能永远不停止——不是算法。</p> <p>2. 算法 是一个满足下列条件的一个计算： (1) 有穷性/终止性：总是在执行有穷步后停止； (2) 确定性：每一步必须有严格的定义和确定的动作； (3) 能行性：每个动作都能被精确地机械执行； (4) 输入：有0个和多个满足约束条件的输入； (5) 输出：有一个或多个满足约束条件的结果。</p>

1.4 逐步求精的程序设计方法
1.4.1 如何求解一个问题
<p>一. 算法的定义</p> <p>3. 算法的逐步求精 就是对用自然语言等描述的算法逐步细致化、精确化和形式化，追求的目标是把算法变成可以执行的程序。</p>

1.4 逐步求精的程序设计方法
1.4.1 如何求解一个问题
<p>二. 求解问题的一般过程</p> <p>1. 模型化：对实际问题进行分析，选择适当的数学模型来描述问题，即模型化； 2. 确定算法：根据模型，找出解决问题的方法(算法)； 3. 逐步求精：就是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。</p>

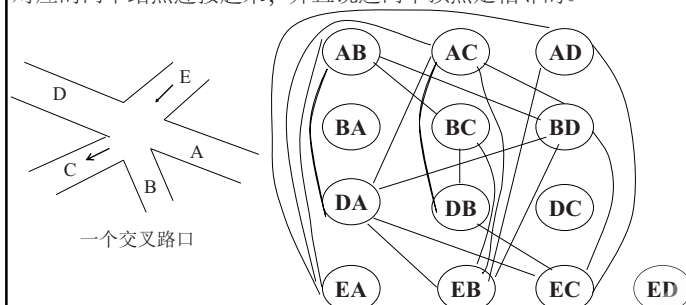
1.4 逐步求精的程序设计方法
1.4.2 算法逐步求精
<p>例1.4.2 交叉路口的交通安全管理问题</p> <p>问题描述： 一个具有有多条通路的交叉路口，当允许某些通路上的车辆在交叉路口拐弯时，必须对其他一些通路上的车辆加以限制，不允许同时在交叉路口拐弯，以免发生碰撞，所有这些可能的拐弯，组成一个集合。</p> <p>图1.5 一个交叉路口</p>  <p>基本要求： 把这个集合分成尽可能少的组，使得每组中所有的拐弯，都能同时进行而不发生碰撞。这样，每组对应一个指挥灯，因而实现了用尽可能少的指挥灯完成交叉路口的管理。</p> <p>有5条组成的交叉路口，其中有2条路是单行道，把从一条路到另一条路的通路称为拐弯；有的拐弯，可以同时通过，有些拐弯，不能同时通过，共有13个拐弯；要求： (1) 设置一组交通灯，实现安全管理 (2) 使交通灯的数目最少。</p>

1.4 逐步求精的程序设计方法

1.4.2 算法逐步求精

例1.4.2的求解过程
模型化：

(1) 用图作为交叉路口的数学模型；(2) 每个拐弯，对应图中的一个顶点；(3) 若两个拐弯，不能同时进行，则用一条边把这两个拐弯，所对应的两个结点连接起来，并且说这两个顶点是相邻的。



1.4 逐步求精的程序设计方法

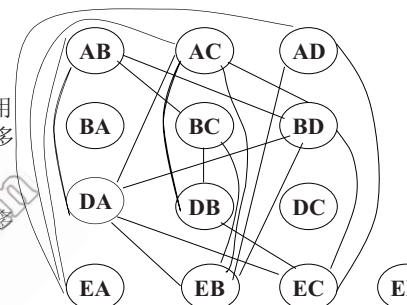
1.4.2 算法逐步求精

确定算法：

(1) 穷举法；(2) 试探法

(3) 贪心法

贪心算法的思想是首先用第一种颜色对图中尽可能多的顶点着色(尽可能多表现出贪心)；然后用第二种颜色对余下的顶点中尽可能多的顶点着色，如此等等，直至所有的顶点都着色。



当用一种新颜色对余下的顶点着色时我们采取下列步骤：

- (1) 选取一个未着色的顶点，并且用新颜色对它着色。
- (2) 扫描所有未着色的顶点集，对其中的每个顶点，确定它是否与已着新颜色的任何顶点相邻。若不相邻，则用新颜色对它着色。

1.4 逐步求精的程序设计方法

1.4.2 算法逐步求精

用C语言描述的“贪心”算法如下：

```
void greedy(G,newclr)
    GRAPH G;
    SET newclr;
    /*类型GRAPH和SET有待具体说明*/
    /*本程序把G中可以着同一色的顶点放入newclr*/
    {
        (1) newclr= $\Phi$ 
        (2) while (G中有未着色的顶点v)
        (3)     if(v不与newclr中的任何顶点相邻){
        (4)         对v着色;
        (5)         将v放入newclr; }
    };
```

其中，G是被着色的图，newclr的初值为空，算法执行的结果形成可以着相同颜色的顶点的集合newclr。只要重复调用greedy算法，直到图中的所有顶点都被着色为止，即可求出问题的解。

1.4 逐步求精的程序设计方法

1.4.2 算法逐步求精

逐步求精：第一步求精：

```
void greedy(G,newclr)
    GRAPH G;SET newclr; /*类型GRAPH和SET有待具体说明*/
    { int found;
        (1) newclr= $\Phi$ ;
        (2) while(G中有未着色的顶点v){
            (3.1) found=0; /*found的初值为false*/
            (3.2) for (newclr中的每一个顶点w)
            (3.3)     if(v与w相邻)
            (3.4)         found=1;
            (3.5) if(found==0){ /*v与newclr中的任何顶点都不相邻*/
                (4)     对v着色;
                (5)     将v放入newclr;
            }
        }
    };
```

1.4 逐步求精的程序设计方法

1.4.2 算法逐步求精

逐步求精：第二步求精：

```
void greedy(GRAPH G, SET newclr)
{ int found;
  newclr=%;
  v=G中第一个未着色的顶点;
  while (v!=0){ /*G中还有未着色的顶点v*/
    found=0;
    w=newclr中的第一个顶点;
    while(w!=0){ /*newclr中的顶点还没取尽*/
      if(v与w相邻)
        found=1;
      w=newclr中的下一个顶点;
    };
    if(found==0){
      对v着色;
      将v放入newclr;
    };
    v=G中下一个未着色的顶点;
  }
};
```

1.4 逐步求精的程序设计方法

1.4.2 算法逐步求精

逐步求精：第三步求精：

由上一步求精的结果可见，算法中大部分操作都归结为对图和集合的操作。设G和S分别是抽象数据类型GRAPH和SET的实例，我们在G上规定如下操作：

- (1)FIRSTG(G)返回G中的第一个未加标记的（未着色的）元素；若G中没有这样的元素存在，则返回NULL。
- (2)EDGE(v,w,G)若v和w在G中相邻，则返回true，否则返回false。
- (3)MARK(v,G)标记G中的元素v。
- (4)ADDG(v,G)将元素v放入G中。
- (5)NEXTG(G)返回G中下一个未标记得元素，若G中没有这样的元素存在，则返回NULL。

在S上规定如下操作：

- (1)MAKENULL(S)将集合S置空。
- (2)FIRSTS(S)返回S中的第一个元素；若S为空集，则返回NULL。
- (3)NEXTS(S)返回S中的下一个元素；若S中没有下一个元素，则返回NULL。
- (4)ADDS(v,S)将v放入S中。

1.4 逐步求精的程序设计方法

1.4.2 算法逐步求精

逐步求精：第三步求精：

```
void greedy(G,newclr)
{ GRAPH G;SET newclr;
  /*类型GRAPH和SET有待说明*/
  {
    int found;
    elementtype v,w;/*elementtype可以自定义*/
    MAKENULL(newclr);
    v=FIRSTG(G);
    while(v!=NULL){
      found=0;
      w=FIRSTS(newclr);
      while(w!=NULL){
        if(EDGE(v,w,G))
          found=1;
        w=NEXTS(newclr);
      };
      v=NEXTG(G);
    }
  }
};
```

1.4 逐步求精的程序设计方法

1.4.2 算法逐步求精

ADT的实现：

按上述函数，最后一步工作就是给出类型elementtype的定义和实现抽象数据类型GRAPH及SET。此后，上述函数就是可执的程序了。

哈工大计算机考研全套视频和资料，真题、考点、典型题、命题规律独家视频讲解！
详见：网学天地（www.e-studysky.com）；咨询QQ：2696670126

1.5 本书采用的类语言描述

一、关于本书采用的类语言描述：

- ①结构类型说明
- ②输入输出约定(**cin** >> **v**, **cout** << **v**)
- ③ **new** 和 **delete**
- ④引入引用类型
- ⑤其他

二、关于引用类型：

所谓的引用就是给变量(对象)起个别名. 换句话说, 就是这个别名和原来的变量(对象)共用一个地址.

无论对原变量(对象)或对其引用的修改, 其实都是对同一地址内容进行修改, 因而变量和对它的引用总具有相同的值.

C++用引用运算符&来定义一个引用类型. 如,

```
int num=50;
int &ref=num; //类型名& 引用名=同类型变量的值
num=num+30; //num、ref均为80
ref=ref+10; //num、ref均为90
```

1.5 本书采用的类语言描述

1. 引用与函数参数----引用调用举例

```
#include<iostream.h>
void Swap(int& a, int& b); //(int, int) (int*, int*)
void main()
{
    int x(5), y(10); // C++的初始化形式<=>x=5
    cout<<"x="<<x<<"    y="<<y<<endl;
    Swap(x, y); // (x, y) (&x, &y)
    cout<<"x="<<x<<"    y="<<y<<endl;
}

void Swap(int& a, int& b); //(int a, int b) (int* a, int* b)

int t;
t=a, a=b; b=t;
```

1.5 本书采用的类语言描述

2. 返回引用的函数

函数返回一个引用的主要目的

是可以使该函数出现在运算符的左边(其他情况一个函数不能成为赋值表达式的左值).

返回引用的函数举例

```
#include<iostream.h>
int a[]={2,4,6,8,10,12};
int &index(int i);
void main()
{
    index(3)=16; //a[3]=16
    cout<<index(3)<<endl; //16
    cout<<a[3]<<endl; //16
}
```

//函数的返回值为对数组元素的引用

```
int& index(int i)
{
    return a[i]; }
```