**数据结构复习**

学习内容：

掌握常见的**逻辑结构**（三大结构：线性结构、树形结构、图形结构，各大结构还有更细的分类，见教材目录，一般还包括集合结构，因集合结构不考虑元素之间关系，一般不讲）

掌握每种逻辑结构的**存储结构**

掌握常见**数据运算**的实现方法

**绪论**

**数据结构的三个方面：**

数据**逻辑结构：**数据元素之间的逻辑关系

数据**存储结**构：数据元素及关系在计算机内的表示，是逻辑结构在存储器中的映像

**数据运算：**数据的操作（包含对操作的定义及**算法**实现）。

逻辑结构分类：**集合、线性结构、树形结构、图形结构，**

**逻辑结构各自特点，生活中的实例**

**设计存储结构时**，既要**存储数据元素、又要存储数据元素之间的关**系

（**应理解各种逻辑结构对应的存储结构是如何存储数据元素的关系及其各种存储结构的优缺点**）

常见的存储结构：**顺序存储结构、链式存储结构**

**顺序存储结构**：一组连续的存储单元存放数据元素，逻辑上相邻的元素在存储器的存储位置也相邻。数据元素之间的逻辑关系由元素的存储位置来表示。

**链式存储结构**：每一个数据元素，对应一个结点，每个结点含数据域和指针域，数据域用来存放数据元素，结点指针用来指示数据元素之间的逻辑关系（邻接关系）。数据元素之间的逻辑关系用指针来表示。逻辑上相邻的数据元素，其结点的物理位置（内存中的位置）不一定相邻。

1、**数据逻辑结构与存储结构的关系**。

**数据逻辑结构是数据元素之间的逻辑关系。由实际问题抽象出来，与计算机无关。数据存储结构是数据元素及关系在计算机内的表示，是逻辑结构在存储器中的映像。一种逻辑结构可以有多种存储结构**

1. 数据结构与数据类型的区别和联系。

数据结构：具有一种或多种特定关系的数据元素的集合，一般包含3分方面的内容：**数据的逻辑结构、存储结构和数据的运算**。数据类型是一组性质相同的值得集合和定义在这个值集上的一组操作的总称。数据类型是已实现的数据结构。

1. **数据结构中运算描述和运算实现的异同**

**运算描述是逻辑结构施加的操作，是运算功能的描述**；**运算实现是完成该功能的具体步骤（算法）**

算法：**基于存储结构的运算实现的步骤或过程，是若干条指令的有穷序列。**

算法的5个重要特性：有穷性、确定性、可行性、有输入、有输出。

**算法性能分析包括时间复杂度分析和空间复杂度分析**。（**熟练掌握算法时间、空间分析**）

算法分析的目的是分析算法的效率以求改进算法。

算法分析一般不用分析算法的绝对执行时间，因为计算机的运行速度不同、编写程序的语言不同、执行程序的环境不同。

一般采用事前估算法，仅考虑算法本身的效率高低，是问题规模的函数。

**本课程采用大O来表示时间复杂度，将时间复杂度写成问题规模的函数 T(n)**。

如

T(n )= 2n2+2n+1 记为T(n)=O(n2)

T(n)=100 T(n)=O(1)

**简化的算法时间复杂度分析**：仅考虑算法的基本操作，即只考虑算法中最深层循环内的原操作。

如：

for (i=1; i<=n; i++)

for (j=1; j<=n; j++)

x++;

T(n)=O(n2)

**递归算法的时间复杂度分析**，先由算法推导出时间的递推式，然后计算T(n)，并用O表示

void fun(int a[]，int n，int k) //数组a共有n个元素

{ int i;

if (k==n-1)

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

printf(“%d\n”，a[i]); //执行n次

else

{ for (i=k;i<n;i++)

a[i]=a[i]+i\*i; //执行n-k次

fun(a，n，k+1);

}

}

T(n，k) = n 当k=n-1时

T(n，k) = (n-k)+T(n，k+1) 其他情况

T(n) = T(n，0) = n+T(n，1) = n+(n-1)+T(n，2)

　 = … = n+(n-1)+…+2+T(n，n-1)

　= n+(n-1)+ …+2+n

　 = O(n2)

**算法的空间复杂度仅考虑算法本身使用的临时空间**

练习参考（练习题1）3、11、12

11（2）表示选择排序算法 其时间复杂度为O(n 2)

12题表示递归版的归并排序算法，（合并算法在第10章归并排序），

其时间复杂度为O(n log n)

学习指导书 P19 第10题，是折半查找的递归算法，其时间复杂度为O(n log n)

非递归版本在第9章（折半查找）

**第2章**

线性表两类存储结构（顺序存储结构、链式存储结构）的差异、优缺点；

顺序表的查找、插入、删除实现算法，时间复杂度；

单链表的查找、插入、删除实现算法，时间复杂度；

有序表的二路归并算法的思路及其实现算法，以及时间复杂度分析

线性表的**顺序存储结构**：把线性表中的所有元素按照顺序存储方法进行存储。

按逻辑顺序依次存储到存储器中一片连续的存储空间中。

**顺序表查找第i个元素（按位查找）的时间复杂度为O(1)**,该性质称为**随机存取**特性；

**顺序表、单链表 类型说明，示意图**

对应**单链表，**当访问过一个结点后，只能接着访问它的后继结点，而无法访问它的前驱结点，。

**单链表查找第i个元素的时间复杂度为O(n)**，**没有随机存取特性**。

**顺序表插入和删除需要大量的移动（分别为后移，前移），**

**而单链表插入和删除只需要修改相应的指针域**

知识要点可以参见学习指导第2章要点归纳（1）-（15）

**顺序表（用顺序存储结构存储的线性表）的优点**：

存储密度大，无须为表示线性表中元素之间的逻辑关系而增加额外的存储空间。具**有**随机存取特性。

**顺序表缺点**：插入和删除操作需要移动大量元素。初始空间大小分配难以掌握。

**适合情况:** 插入和删除操作不频繁，数据元素个数事先大致可以知道，适合于经常按位查找

**链表（用链式存储结构存储的线性表）的优点**：由于采用结点的动态分配方式，具有良好的适应性。插入和删除操作只需修改相关指针域，不需要移动元素。

**缺点**：存储密度小，为表示线性表中元素之间的逻辑关系而需要增加额外的存储空间（指针域）。不具有随机存取特性。

**链表适合情况:** 插入和删除操作频繁，数据元素个数事先不太确定的情形。

1. 简述**线性表两种存储结构各自的主要特点**。

答：线性表的两种存储结构分别是顺序存储结构和链式存储结构。顺序存储结构的主

要特点如下：

① 数据元素中只有自身的数据域，没有关联指针域。因此，顺序存储结构的**存储密度**

**较大**。

② 顺序存储结构需要分配一整块比较大存储空间，所以**存储空间利用率较低**。

③ 逻辑上相邻的两个元素在物理上也是相邻的，通过元素的逻辑序号可以直接其元素

值，即具有**随机存取特性**。

④ **插入和删除操作会引起大量元素的移动**。

链式存储结构的主要特点如下：

① 数据结点中除自身的数据域，还有表示逻辑关系的指针域。因此，链式存储结构比

顺序存储结构的**存储密度小**。

② 链式存储结构的每个结点是单独分配的，每个结点的存储空间相对较小，所以**存储**

**空间利用率较高**。

③ 在逻辑上相邻的结点在物理上不一定相邻，因此**不具有随机存取特性**。

④ **插入和删除操作方便灵活，不必移动结点，只需修改结点中的指针域即可**。

2. 简述单链表设置头结点的主要作用。

答：对单链表设置头结点的主要作用如下：

① **对于带头结点的单链表**，在单链表的任何结点之前插入结点或删除结点，所要做的

都是修改前一个结点的指针域，因为任何结点都有前驱结点（**若单链表没有头结点，则首**

**结点没有前驱结点，在其前插入结点和删除该结点时操作复杂些**），所以算法设计方便。

② 对于带头结点的单链表，在表空时也存在一个头结点，因此**空表与非空表的处理是**

**一样的。**

**单链表设置头结点的优点：（和前面意思一致）**

第一个结点的操作和表中其他结点的操作相一致，无需进行特殊处理；

无论链表是否为空，都有一个头结点，因此空表和非空表的处理也就统一了。

**基于单链表的算法设计**

可以和顺序表的设计对照，找出一些规律；

比如要找到第i个结点，必须设置一个指针变量p=L，通过反复执行p=p->next，i次。

p=L ；k=0；

while(p!=NULL && k<i)

{

p=p->next;

k=k+1;

}

**注意:** 循环结束有可能没有找第i个结点（p==NULL），

当p!=NULL时找到第i个结点。

查找值为x的结点时只需将上面的代码稍微改动：

p=L->next ；

while(p!=NULL && p->data !=x)

p=p->next;

**注意:**循环结束有可能没有找到值为x结点（当p==NULL），当p!=NULL时找到。

如果要删除第i个结点，应先找到第i-1个结点，再执行删除操作；

插入第i个结点也是类似的。

**例**

设计一个算法，删除一个单链表L中值为x的结点(假设各个结点值不同)

先考虑查找值为x 的结点，如前所示，

但现在是删除值为x的结点，所以需要设置一个pre指针，指向p的前驱

bool Delete( LinkNode \*& L, ElemType x)

{ LinkNode \*p, \*pre;

pre=L; p=L->next ；

while(p!=NULL && p->data !=x)

{pre=p;

p=p->next;

}

if(p==NULL) return false; //没找到，返回false

else //p->data==x 删除p

{ pre->next=p->next; free(p); return true; }

}

删除最大值对应的结点，先要找到最大值对应的结点，然后删除

**例**：设计一个算法，删除一个单链表L中元素值最大的结点（假设最大值结点是唯一的）

先考虑只是查找单链表L中元素值最大的结点，可以这样操作

大致想法：

设置两个指针 p,pmax,

初值为pmax=p=L->next;

p后移（p=p->next）; 若pmax->data<p->data, 则pmax=p;

重复上一步，直到p指向结束

但现在是要删除单链表L中元素值最大的结点，所以还需设置另外一对指针，pre,maxpre,分别指向p,pmax的前驱

所以可以这样编程：

maxpre=L; pmax=L->next;

pre=L->next; p=L->next->next;

while(p!=NULL)

{

if (pmax->data<p->data)

{ maxpre=pre;

maxp=p;

}

pre=p;

p=p->next;

}

maxpre->next=pmax->next; //删除

free(pmax);

详细代码参见教材；

插入类似，见习题14

练习参考：例题2.7 2.8 习题14 答案见学习指导

**关键是要拟好思路，真正自己正确编写出两、三个程序，而不是仅仅看答案!**

**第3章**

**栈、队列、线性表的关系：**

栈、队列都是受限的线性表，逻辑结构都是线性结构；

栈在同一端插入（入栈）、删除（出栈）；特点是后进先出;

队列在一端插入（入队）、另一端删除(出队)；特点是先进先出;

线性表可以在任意合法位置插入（1=<i<=n+1）、删除(1=<i<=n)

**顺序栈：**

top总是指向栈顶元素，初始值为-1

当top=MaxSize-1时不能再进栈－栈满

进栈时top增1，出栈时top减1

栈空条件：top=-1

栈满条件：top=MaxSize-1

进栈e操作：top++; 将e放在top处

退栈操作：从top处取出元素e; top--

**链栈：**用带头结点的单链表实现

栈空条件：s->next=NULL

栈满条件：不考虑

进栈e操作：将存放e的结点插入到头结点之后

退栈操作：取出头结点之后结点的元素并删除之

**顺序队：**

约定rear总是指向队尾元素

元素进队，rear增1

约定front指向当前队中队头元素的前一位置

元素出队，front增1

当rear=MaxSize-1时不能再进队 （MaxSize-1代表数组的最大下标）

**顺序队列4要素：**

队空条件：front = rear

队满条件：rear = MaxSize－1

元素e进队：rear++; data[rear]=e;

元素e出队：front++; e=data[front];

**什么是假溢出，如何解决：**

**假溢出** 顺序队因为采用rear==MaxSize-1作为队满条件，当队满条件为真时，队中可能还有若干空位置。这种溢出并不是真正的溢出，称为假溢出。

**解决方案** 把数组的前端和后端连接起来，形成一个环形的顺序表，即把存储队列元素的表从逻辑上看成一个环，称为**环形队列或循环队列**。

**环形队列四要素：**

队空条件：front = rear

队满条件：(rear+1)%MaxSize = front

进队e操作：rear=(rear+1)%MaxSize; 将e放在rear处

出队操作：front=(front+1)%MaxSize; 取出front处元

**栈的应用：**括号匹配问题，进制转换问题，表达式求解问题，二叉树的非递归（前序、中序、后序）遍历算法 求解迷宫问题 图的非递归深度优先遍历等

**队列的应用：**二叉树的层次遍历，求解迷宫问题 图的广度优先遍历等

**在实际应用中** 一般栈和队列都用来存放临时数据，如果先保存的元素先处理，应该采用队列，如果后保存的元素先处理，应该采用栈

**练习参考：**P115 1、4、7 答案见学习指导

1. **树和二叉树**

**树的定义：**除根结点外，每个结点有且仅有一个前驱结点 每个结点可以有零个或多个后继结点

**树的表示**：掌握树形表示法和括号表示法

**树的基本术语**

**树的性质**：

1. 树中的结点数等于所有结点的度数之和加1 （这里的度实际上相指离散数学中的出度）
2. 度为m的树中第i层上至多有mi-1个结点（i≥1）
3. 高度为h的m次树至多有 个结点
4. 具有n个结点的m次树的最小高度为logm(n(m-1)+1)

**树的遍历运算**是指按某种方式访问树中的每一个结点且每一个结点只被访问一次

一般树有 先根 后根 层次遍历三种

**树的存储结构**主要有双亲存储结构、孩子存储结构、孩子兄弟存储结构

**注意各自的优缺点**

**二叉树**

**二叉树的递归定义**：二叉树是有限的结点集合，这个集合或者是空，

或者由一个根结点和两棵互不相交的称为左子树和右子树的二叉树组成

**满二叉树和完全二叉树**

**二叉树的性质**

**完全二叉树的性质**

若i≤n/2，则编号为i的结点为分支结点，否则为叶结点。

 除树根结点外，若一个结点的编号为i，则它的双亲结点的编号为i/2。

若编号为i的结点有左孩子结点，则左孩子结点的编号为2i；若编号为i的结点有右孩子结点，则右孩子结点的编号为2i+1。

**二叉树的存储**

**二叉树的顺序存储 类型说明，示意图，优缺点，适用情况**

对于完全二叉树来说，其顺序存储是十分合适的。

对于一般的二叉树，特别是对于那些单分支结点较多的二叉树来说是很不合适的，因为可能只有少数存储单元被利用，特别是对退化的二叉树（即每个分支结点都是单分支的），空间浪费更是惊人。

在**顺序存储结构中，找一个结点的双亲和孩子都很容易：**序号i结点的父结点的序号为[i/2]

左右孩子结点序号依次为2i 和2i+1

**二叉树的链式存储**

**类型说明 示意图 优缺点 适用情形**

除了指针外，二叉链比较节省存储空间。占用的存储空间与树形没有关系，只与树中结点个数有关。

在二叉链中，找一个结点的孩子很容易，但找其双亲不方便

一**颗树采用孩子兄弟链存储结构表示 与由树转为得到的二叉树的二叉链表示相同**

**二叉树的3种递归遍历（前序、中序、后序），层次遍历 （熟练掌握）**

**基于二叉树遍历的算法设计**

**先要会三种遍历本身的含义，其次要理解对应的算法实现**

**比如中序遍历：访问左子树，访问根，访问右子树**

**对应的函数大致为**

f( b)

{ f(b->lchild);

访问根b；

f(b->rchild);

}

**这里还应设置递归出口，当根为空时跳出**

**所以应修改为**

f( b)

{

if(b!=NULL)

f(b->lchild);

访问根b；

f(b->rchild);

}

**当然对参数应该给出类型说明，访问也要根据具体情况而言，这里假设为输出结点的值**

void f( BTNode \*b)

{

if(b!=NULL)

{f(b->lchild);

printf（“%c”,b->data）;（//访问根b；这里假设数据元素data的类型为字符型

f(b->rchild);

}

}

**例**

假设二叉树采用二叉链存储结构存储，设计一个算法，计算一棵给定二叉树的所有结点个数

基本思想:利用其中一种遍历方法，设置结点计数器k, 每访问一个结点，结点计数器加1；

递归出口，当树为空时k=0;

int f1( BTNode \*b)

{ if (b==NULL) return 0;

if(b!=NULL)

return f1(b->lchild)+f1(b->rchild)+1

}

**例**

假设二叉树采用二叉链存储结构存储，设计一个算法，输出一棵给定二叉树的所有叶子结点。

基本思想：将遍历算法中的输出任何结点改为条件输出（只输出叶子结点）

**void f( BTNode \*b)**

**{**

if(b!=NULL)

{f(b->lchild); //递归调用左子树

**if(b->lchild==NULL &&b->rchild ==NULL) //如果是叶子结点，输出**

printf（“%c”,b->data）;

f(b->rchild);//递归调用右子树

}

}

如何统计叶子结点个数？

int n=0;

**void f( BTNode \*b)**

**{**

if(b!=NULL)

{f(b->lchild);

**if(b->lchild==NULL &&b->rchild ==NULL)**

**n++;**

f(b->rchild);

}

}

**注意：**这里n是全局变量，写在函数体外

**例：**

假设二叉树中每个结点值为单个字符，采用二叉链存储结构存储。设计一个算法

求二叉树 *b* 中最小值的结点值。

基本思路，设置指针p,开始指向b,在遍历的过程中 如果有比p->data小的，置p为该节点

（或者说，min置初值为b->data, 在遍历的到某结点时 如果有比min小,置min为该结点的值 ）

int min=32767;//设min初值为无穷大

void f(BTNode \*b)

{ if(b!=NULL) //当b为空递归出口

{ if (b->data<min) min=b;

f(b->lchild);

f(b->rchild);

}

}

注意这里设置min为全局变量

当然也可以设置为引用型参数：

**void F(BTNode \*b,char &min)**

{

if (b->data<min)

min=b->data;

F(b->lchild,min);

//在左子树中找最小结点值

F(b->rchild,min);

//在右子树中找最小结点值

}

注意在参数传递时 要给**min赋一个较小初值，也可以直接赋值为根结点的值**

**假设已存在树的跟结点为b**

**min=b->data;**

**F（b, min）;**

**输出min的值即可**

**例**

假设二叉树中每个结点值为单个字符，采用二叉链存储结构存储。设计一个算法，

采用先序遍历方法求二叉树 *b* 中值为 *x* 的结点的子孙，假设值为 *x* 的结点是唯一的。

基本思路，需要先查找到x对应的结点p，再以p为根结点进行遍历,但注意不能访问p本身。

void F1( BTNode \*b) //前序遍历算法

{

if(b!=NULL)

{

printf（“%c”,b->data）;

F1(b->lchild);

F1(b->rchild);

}

}

void F2(**BTNode \*b, BTNode \*&p, Elemtype x** )//查找x对应的结点

{ if(b!=NULL)

{ if（b->data==x） { p=b;return ;}

F2(b->lchild);

F2(b->rchild);

}

void F3(**BTNode \*b,Elemtype x** )

{

F2(b,p, x);//调用F1，查找x对应的结点p

F1(p->lchild);//访问p的左子树

F1(p->lchild);//访问p的右子树

}

}

代码中可能还有地方需要完善，比如，定义，参数传递等

可以将F2 、F3合并为F：

void F(**BTNode \*b, Elemtype x** )

{ if(b!=NULL)

{ if（b->data==x） //若根节点的data==x，则遍历b的左右子树

{ F1(b->lchild);

F1(b->rchild); return；

}

}

F(b->lchild);//递归调用左子树

F(b->rchild);//递归调用右子树

}

学习指导书的代码应该是准确的

**哈夫曼树；WPL; 哈夫曼编码；存储结构；类型说明；**

参考练习：例7.13 7.14 习题13、15、17 习题答案见学习指导

第8章

**邻接表、邻接矩阵的定义、类型说明、示意图、适用情况**

邻接矩阵 特别适合于稠密图的存储 邻接矩阵的存储空间为O(n2)

邻接表 特别适合于稀疏图存储 邻接表的存储空间为O(n+e)

对于稠密图和稀疏图，采用邻接矩阵和邻接表哪个更好些？

答：邻接矩阵适合于稠密图，因为邻接矩阵占用的存储空间与边数无关。邻接表适合

于稀疏图，因为邻接表占用的存储空间与边数有关。

图的深度优先遍历、广度优先遍历

邻接表 画法，基于邻接表的两种遍历

两个最小生成树算法

普里姆（Prim）算法

克鲁斯卡尔（Kruskal）算法

Dijkstra算法

参考练习 练习8 第3、5、6、8 答案见习题指导

第9章

**折半查找 条件：顺序存储、有序**

**算法执行过程 复杂度 O(log n)**

**二叉排序树 定义 插入过程**

**平衡二叉树 定义 插入过程**

**平衡二叉树的查找最坏复杂度为 O(log n)，而二叉排序树查找最坏复杂度为 O( n)**

哈希表 定义 构造 哈希冲突 参见例题9.10 9.11

第10章

归并排序 快速排序

执行过程 代码 复杂度

**1、数据结构基本概念及算法分析**

**数据结构**、**逻辑结构、存储结构**的**定义及关系**

逻辑结构的类型：**集合 线性结构 树形结构 图形结构**

（）算法的基本概念

(2)算法性能分析与度量：算法的性能标准；算**法的空间复杂度与时间复杂度概念与分析方法**；时间复杂度的渐进表示法；

**2、线性表**

(1) 顺序表：顺序表的定义和特点；顺序表的类定义；顺序表的查找、插入和删除；**顺序表的优缺点**

(2) **单链表**：单链表的结构；**单链表的类定义**；**单链表中的插入与删除**；带表头结点的单链表；；**链表的优缺点**　**基于单链表的算法设计**

**3、栈和队列**

　　（1） 栈：栈的抽象数据类型；栈的顺序存储表示；栈的链接存储表示

（2） 队列：队列的抽象数据类型；队列的顺序存储表示；队列的链接存储表示；

（3） **栈和队列的相同点和不同点；**

**（4）环形队列如何解决假溢出**

**4、树与森林**

（1）树和森林的概念：树的定义；树的术语；

（2）二叉树：二叉树的定义；二叉树的性质；

　 （3）**二叉树的表示：顺序表表示；链式存储表示**

（4）**二叉树遍历：中序遍历；前序遍历；后序遍历；层次遍历**

**（5）基于二叉树遍历的算法设计**

　（7) **哈夫曼树**：**带权路径长度；哈夫曼树；哈夫曼编码；存储方法**

**5、 图**

　（1）图的基本概念：

　（2）**图的存储表示：邻接矩阵；邻接表。**

（3）**图的遍历与连通性：深度优先搜索；广度优先搜索**；

（4）图的基本算法： **Prim普里姆算法；Kruskal克鲁斯卡尔算法**； **Dijkstra最短路径**；拓扑排序AOV；关键路径AOE。

**6、查找**

　  (1) 查找、查找表及平均查找长度的基本概念

(2）顺序查找；基于有序顺序表的**折半查找算法及分析**

　　(3) 二叉排序树：**定义**，**二叉排序树的查找、插入**；

　　(4)平衡二叉树 (AVL树)：**AVL树的定义,查找，插入，平衡二叉树的调整**；

  (5) 散列：散列表的基本概念，构造方法(除留余数法)，解决冲突的方法(线性探测，二次线性探测)，查找性能的分析。

**7、排序**

(1）排序的基本术语与概念

(2) 插入排序：直接插入排序；折半插入排序；希尔排序

(3) 交换排序: 冒泡排序；**快速排序**

(4) 选择排序：简单选择排序；堆排序

(5) 归并排序：**二路归并排序**

**考试形式**

1. **试卷成绩及考试时间**

**本试卷满分为100分，考试时间为120分**

1. **答题方式**

**答题方式为闭卷、笔试**

1. **推荐参考书**
2. **李春葆，《数据结构教程（第5版）》，清华大学出版社**
3. **李春葆，《数据结构教程（第5版）学习指导》，清华大学出版社**
4. **严蔚敏，吴伟民，数据结构（C语言版）**
5. **试卷内容结构**

**考试具有一定的综合性，难以准确统计哪章占分多少**

**侧重于考察三种数据结构（线性、树形、图形）存储结构的分析设计、算法的分析设计 既要理解算法的基本思路、又要能读懂代码，分析算法的时间空间复杂度**

**数据结构基本概念10%（数据结构中存储结构、逻辑结构、算法分析与设计贯穿始终）**

**顺序与链式线性表：20%，栈与队列：5%**

**树与二叉树：25%，图：25%，查找与排序：15%**

1. **试卷题型结构**

**简答题20分，解答题20分，综合题30分，算法设计题30分**