# 第一章 绪论

一、单项选择题

1．在数据结构与算法中，从逻辑上可以把数据结构分为\_\_\_\_\_\_\_\_。

A．紧凑结构和非紧凑结构 B．线性结构和非线性结构

C．内部结构和外部结构 D．动态结构和静态结构

答案：B。

从逻辑角度看，基本的数据结构包括4类，分别是集合、线性结构、树结构和图结构。其中，树结构和图结构属于非线性结构。集合也可以使用线性结构表示。

2．数据元素可以细分为\_\_\_\_\_\_\_\_。

A．数据项 B．字符 C．二进制位 D．数据记录

答案：A。

根据定义，数据元素可以细分为数据项。字符和二进制位都是表示数据的具体单位（选项B和C都是错误的）。数据元素有时称为记录（选项D错误）。

3．如果说线性结构中元素之间是一对一的关系，则树结构中元素之间的关系是\_\_\_\_\_\_\_\_。

A．一对一的 B．一对多的 C．多对多的 D．不确定的

答案：B。

线性结构中，除首元素和尾元素外，每个元素有唯一的直接前驱和唯一的直接后继，所以对于每个元素而言，它对应唯一的直接后继，形成一对一的关系。在树结构中，每个元素仅有唯一的直接前驱，但可以有多个直接后继，所以是一对多的关系。当然，树中的根（最前面的元素）和叶结点（后面的元素）除外。图结构中是多对多的关系，每个元素可以有多个直接前驱，也可以有多个直接后继。

4．在数据结构中，数据元素之间均为一个对一个的关系，称为\_\_\_\_\_\_\_\_。

A．线性结构 B．树结构 C．图结构 D．集合结构

答案：A。

基本概念。具体说明请参见第一章第3题的答案。

5．下列选项中，不属于数据结构常用存储方式的是\_\_\_\_\_\_\_\_。

A．顺序存储方式 B．链式存储方式 C．分布存储方式 D．散列存储方式

答案：C。

数据结构与算法课程中没有讨论分布存储方式，除了选项中给定的A、B和D以外，还讨论了索引存储方式。这是数据结构中常用的4种存储方式。

6．算法分析要评估的两个主要方面是\_\_\_\_\_\_\_\_。

A．正确性和简明性 B．时间复杂度和空间复杂度

C．可读性和可维护性 D．数据复杂性和程序复杂性

答案：B。

算法的正确性是必须的，简明性、可读性、可维护性等都不是算法要评估的内容（选项A和选项C均错误），它们都属于算法应具备的众多特性中的内容。数据复杂性是数据本身的状态，也不是算法要评估的，程序复杂性说得不明确（选项D错误）。算法要评估的是时间复杂度和空间复杂度。

7．下列选项中，定义抽象数据类型时不需要做的事情是\_\_\_\_\_\_\_\_。

A．给出类型的名字 B．定义类型上的操作

C．实现类型上的操作 D．用某种语言描述抽象数据类型

答案：C。

定义抽象数据类型时，需要给类型命名（选项A），需要指明相关的操作有哪些（选项B），需要使用程序语言或是自然语言描述抽象数据类型（选项D）。在确定了存储结构之后才能具体实现操作过程，所以在定义抽象数据类型阶段，不涉及这些操作的实现。

8．下列算法的时间复杂度中，算法效率最低的是\_\_\_\_\_\_\_\_。

A．*O*(*n*) B．*O*(*n*2) C．*O*(*2*n) D．*O*(log *n*)

答案：C

基本概念。一般而言，算法的效率用算法时间复杂度进行衡量，算法效率从高到低依次是：常数阶*O*(1)，对数阶*O*(log*n*)，线性阶*O*(*n*)，线性对数阶*O*(*n*log*n*)，多项式阶*O*(*n*2)，指数阶*O*(2n)。

9．设*n*是描述问题规模的非负整数，下面程序片段的时间复杂度是\_\_\_\_\_\_\_\_。

x=2;

while(x<n/2)

x=2\*x;

A．*O*(log2*n*) B．*O*(*n*) C．*O*(*n* log2*n*) D．*O*(*n*2)

答案：A。

*n*是输入规模。x从2开始，每次倍增，达到或超过n/2时倍增的次数与log2*n*的大小相当。

10．设*n*是描述问题规模的非负整数，下列程序片段的时间复杂度是\_\_\_\_\_\_\_\_。

x=1;

while(n>=(x+1)\*(x+1))

x=x+1;

A．*O*(*n1/*2) B．*O*(log2*n*) C．*O*(*n*) D．*O*(*n* log2*n*)

答案：A。

11．设计求斐波那契数列前n项的算法时，适宜使用的算法策略是\_\_\_\_\_\_\_\_。

A．分治法 B．递推法 C．递归法 D．穷举法

答案：B。

参见本章算法设计题2的解答。

二、填空题

1．在数据结构与算法课程中，将所有能输入计算机并被计算机程序处理的符号集合称为\_\_\_\_\_\_\_\_\_\_。

答案：数据。

基本概念。

2．构成数据的基本单位是\_\_\_\_\_\_\_\_\_\_。

答案：数据元素。

基本概念。在数据结构课程中，数据元素是构成数据的基本单位，同时，也是作为一个整体进行研究。

3．数据元素及其关系在计算机内的存储方式称为数据的\_\_\_\_\_\_\_\_\_\_。

答案：存储结构。

数据的结构分为逻辑结构和物理结构。逻辑结构是指数据元素之间的逻辑关系，物理结构是指数据结构在计算机中的表示及存储方式。

数据的逻辑结构与存储关系是无关的。

4．数据结构包括4类，分别是集合、线性结构、树结构和\_\_\_\_\_\_\_\_\_\_。

答案：图结构。

基本概念。

5．构成索引表的基本内容是\_\_\_\_\_\_\_\_\_\_。

答案：索引项。

索引表由索引项组成，索引项指示数据元素所在的物理位置。通过索引项，可以加快数据元素的查找速度。

6．一个算法必须在执行有穷步之后结束，这个特性是算法的\_\_\_\_\_\_\_\_\_\_。

答案：有穷性。

基本概念。算法必须满足5个重要特性：有0或多个输入值、有1或多个输出值、有穷性、确定性和可行性。

7．设A={1,3,-4,5,-6,4,6,-3,-2,6}，调用MaxSeq(A,n,C);后，C[1]的值是\_\_\_\_\_\_\_\_。

答案：9。

函数MaxSeq是求数组A的最大子数组，结果是{4,6,-3,-2,6}，C[1]中保存的这个子数组最后一个元素在A中的下标，得到9。

三、解答题

1．举例说明计算机能处理哪些数据？

【参考答案】计算机能处理的数据多种多样，大到可以是一幅地图、一本书、一部电影，小到可以是一个字符，甚至是计算机中的一位。可以是数值型的数据，也可以是非数值型的数据。既可以是简单结构的，也可以是复杂结构的。

2．线性结构中，如何理解元素之间是一对一的关系？

【参考答案】线性结构中，除表头元素和表尾元素外，每个元素有唯一的直接前驱和唯一的直接后继。所以对于每个元素而言，当有直接后继时，它与其直接后继形成一对一的关系。

3．定义代表交通工具的抽象数据类型vehicle，添加必要的数据和操作。

【参考答案】对于vehicle，可以定义表示其基本属性的数据，包括品牌brand、颜色color、价格price等。相应的操作可以是设置最大速度setSpeed(int speed)、设置自重setdeadweight(int weight)等。

ADT vehicle{ //交通工具的抽象数据类型定义

数据部分：

brand： //品牌，字符串

color： //颜色，字符串

price： //价格，实型

speed： //最大速度，实型

deadweight： //自重，实型，单位吨

操作部分：

setSpeed(speed)： //设置最大速度

输入：speed

输出：是否设置成功

前提条件：设置的speed值不能超出限制

setdeadweight(weight)： //设置自重

输入：weight

输出：是否设置成功

前提条件：设置的weight值不能超出限制

}

4．定义表示复数的抽象数据类型。

【参考答案】复数由实部和虚部表示，实部和虚部都是实型值。相应的操作有算术操作。

ADT complex{ //复数的抽象数据类型定义

数据部分：

real： //实部，实型

imaginary： //虚部，实型

操作部分：

add(complex1,complex2)： //加法

输入：两个复数

输出：它们的和

前提条件：两个复数正确

sub(complex1,complex2)： //减法

输入：两个复数

输出：它们的差

前提条件：两个复数正确

multiplication(complex1,complex2)： //乘法

输入：两个复数

输出：它们的积

前提条件：两个复数正确

division(complex1,complex2)： //除法

输入：两个复数

输出：它们的商

前提条件：两个复数正确

}

5．为什么不使用算法的绝对运行时间来衡量算法的时间效率？

【参考答案】同一个算法处理不同数量的数据时，所花费的绝对运行时间可能不同。同一个算法处理相同数量的数据时，在不同配置的电脑上的绝对运行时间也可能不同。所以，使用算法的绝对运行时间不能有效衡量算法的时间效率。

6．评估算法的空间效率时，需要考虑占用的哪些存储空间？

【参考答案】算法运行过程中，临时占用的空间大小是要考虑的存储空间。算法代码占用的空间、算法中初始数据占用的存储空间不考虑在内。

四、算法设计题

1．试设计一个算法，使用最少的比较次数找出三个不同整数*a*,*b*,*c*的中值。

【参考答案】

int middle(int a,int b,int c)

{

if(a>b){

if(b>c) return b; //a>b>c

else if(a>c) return c; //a>c>b

else return a; //c>b>a

}

else{

if(a>c) return a; //b>a>c

else if(b>c) return c; //b>c>a

else return b; //c>b>a

}

}

2．试设计一个算法，对于给定的正整数*n*，列出斐波那契数列的前*n*项，要求空间复杂度为*O*(1)。

【参考答案】斐波那契数列的前两项均为1，从第三项开始，每一项都是其前两项之和。使用整型变量f1、f2和f3表示数列中相邻三项的值，初始时，f1和f2的值均为1。然后计算新的项f3=f1+f2，计算后用f2的值更新f1，用f3的值更新f2。

void fibonacci(int n)

{

int f1=1,f2=1,f3,i;

if(n<=0){

printf("n值错误\n");

return;

}

else if(n==1){

printf("1\n");

return;

}

else{

printf("1 1 ");

for(i=3;i<=n;i++){

f3=f1+f2;

printf("%d ",f3);

f1=f2;

f2=f3;

}

printf("\n");

}

}

# 第二章 线性表

一、单项选择题

1．下列选项中，不属于链表特点的是\_\_\_\_\_\_\_\_。

A．插入、删除时不需要移动元素 B．可随机访问任一元素

C．不必事先估计存储空间 D．所需空间与元素个数成正比

答案：B。

链表中的元素可以不必保存在相邻的地址空间中，所以当链表中的元素个数有变化时，其他的元素不必像顺序表那样进行移动（选项A）。而且链表占用的空间随需分配，随用随分配，不必提前预估数量（选项C），表中有多少个元素就分配多少个表结点（选项D）。但要访问链表中的元素时，只能从表头开始，沿指针的指示逐个结点地查找下去，所以不能像在数组中那样通过下标定位到所需的地址，即不能实现随机访问。

2．下列关于线性表的叙述中，错误的是\_\_\_\_\_\_\_\_。

A．线性表采用链式存储方式，便于进行插入和删除操作

B．线性表采用顺序存储方式，便于进行插入和删除操作

C．线性表采用链式存储方式，不必占用一片连续的存储单元

D．线性表采用顺序存储方式，必须占用一片连续的存储单元

答案：B。

线性表既可以采用顺序存储结构，也可以采用链式存储结构。采用顺序存储结构时，各元素要连续存放（选项D正确），当有插入和删除操作时，通常会导致其他元素的移动（选项B错误）。当采用链式存储结构时，因为各结点的地址并不要求是相邻的（选项C正确），所以插入和删除操作不会引起其他元素的移动（选项A正确）。

3．若一个线性表分别采用下列存储结构进行存储中，则读取一个指定位置（序号）的元素所花费时间最少的是\_\_\_\_\_\_\_\_。

A．顺序表 B．单向链表 C．双向链表 D．循环链表

答案：A。

采用顺序存储结构保存线性表时，要读取一个指定位置（序号）的元素可以直接通过下标访问，下标与元素在内存中存储地址是一个简单的计算公式，因为可以*O*(1)访问线性表任一位置的元素，是花费时间最少的（选项A正确）。对于采用链式存储结构保存的线性表，无论是单向链表（选项B）、双向链表（选项C），还是循环链表（选项D），要读取指定位置（序号）的元素均需要从链表首结点开始进行计数，这样，操作的时间复杂度均为*O*(*n*)。

4．下列关于线性表的叙述中，错误的是\_\_\_\_\_\_\_\_。

A．除第一个元素外均有唯一的前驱

B．除最后一个元素外均有唯一的后继

C．元素的个数可以是0，此时表为空表

D．元素的个数若为无穷，则表为无穷表

答案：D。

基本概念。在数据结构中，线性表是一个有限元素的序列。

5．下列关于线性表L的叙述中，正确的是\_\_\_\_\_\_\_\_。

A．L是由n个相同元素组成的离散数据序列

B．L中任意一个元素有且仅有唯一直接前驱

C．可以在L中进行取元素、查找或排序等操作

D．不能在L的任意位置插入或删除一个元素

答案：C。

基本概念。线性表是由n个相同数据类型的数据元素组成的有限序列（A错误）；L中除第一个元素之外，其他元素有且仅有唯一的直接前驱，第一个元素没有直接前驱（B错误）；在线性表中可以进行插入、删除、取元素值、查找或排序等操作，所以C是正确的，D是错误的。

6．若线性表L最常用的操作是访问任一位置的元素和在最后进行插入和删除运算，为使操作的时间复杂度好，则下列选项中，为L选择的存储结构为\_\_\_\_\_\_\_\_。

A．顺序表 B．双向链表

C．带头结点的双向循环链表 D．单循环链表

答案：A。

采用顺序存储结构保存线性表时，能以*O*(1)访问线性表任一位置的元素。要在线性表中间位置进行插入和删除时，会引发部分元素在数组中的移动，但如果仅在最后位置进行插入和删除，则避免了元素的移动，操作也能达到*O*(1)。选项B、C和D都是链式存储结构，虽然它们进行插入和删除操作时能达到*O*(1)，但访问任一位置元素时，平均情况下，仅能达到*O*(*n*)。综合来看，选择顺序表更合适。

7．线性表L中最常用的操作是在最后一个元素之后插入一个元素和删除第一个元素，为使操作的时间复杂度好，则下列选项中，为L选择的存储结构为\_\_\_\_\_\_\_\_。

A．单链表 B．仅有头指针的单循环链表

C．双向链表 D．仅有尾指针的单循环链表

答案：D。

4个选项给出的都是链式存储结构。根据题意，操作的位置是表尾和表头，所以要选择能快速访问这两个位置的结构。

单链表中仅有表头指针，能快速访问表头，但不能快速访问表尾。单循环链表中，通过头指针能快速访问表头位置，访问表尾也是不方便的。双向链表也是类似的。选项D给出的链表，通过尾指针可以快速访问到表尾结点，实现在其之后的插入，也能通过表尾结点的next指针快速找到表头结点，实现删除操作。所以，仅有尾指针的单循环链表是最合适的结构。

8．为了提供全程对号的高速铁路售票系统，保证每一位旅客从上车到下车期间都有独立座位，短途乘客下车后该座位可以销售给其他乘客，铁路公司设计了基于内存的系统，系统中需要描述座位信息和每个座位的售票情况。保存座位信息和每个座位售票情况的数据结构分别是\_\_\_\_\_\_\_\_。

A．数组，数组 B．数组，链表

C．链表，数组 D．链表，链表

答案：B。

高铁上，每节车厢都有自己固定的座位数，不变化，所以可以使用数组来描述座位。但每个座位的售票情况是随时变化的，应当使用动态的结构来描述，故选用链表来描述。

9．若长度为*n*的线性表采用顺序存储结构，则在其第*i*(0≤*i*≤*n*)个位置插入一个新元素的算法的时间复杂度为\_\_\_\_\_\_\_\_。

A．*O*(1) B．*O*(*i*) C．*O*(*n*) D．*O*(*n*2)

答案：C。

顺序表中，插入操作会导致从插入位置至表尾的全部元素依次后移一个位置。平均情况下，要移动一半的元素，所以时间复杂度是*O*(*n*)。

10．对于含*n*个元素采用顺序存储的线性表，访问位置*i*（0≤i≤n-1）处的元素和在位置*i*（0≤i≤n）插入元素的时间复杂度分别为\_\_\_\_\_\_\_\_。

A．*O*(*n*)和*O*(*n*) B．*O*(*n*)和*O*(1) C．*O*(1)和*O*(*n*) D．*O*(1)和*O*(1)

答案：C。

顺序表中，访问结点时可以根据下标直接计算地址，所以时间复杂度是*O*(1)。而插入结点的时间复杂度是*O*(*n*)。

11．线性表(a0,a1,…,an-1)以链接方式存储时，访问位置*i*的元素的时间复杂度为\_\_\_\_\_\_\_\_。

A．*O*(1) B．*O*(*i*-1) C．*O*(*i*) D．*O*(*n*)

答案：D。

链式存储线性表时，如果要访问表中的结点，通常会从表头开始，逐个结点依次遍历过来，平均而言，要访问表中一半的元素。

12．在带头结点的非空单循环链表head中，指向尾结点的指针p满足的条件是\_\_\_\_\_\_\_\_。

A．p==NULL B．p==head

C．p->next==head D．p->next==NULL

答案：C。

若单循环链表head非空，则必有尾结点，尾结点的next指向头结点，也就是说，p指向结点的next值应该等于head的值（选项C正确）。

若指针p的值是NULL（选项A），则表示指针p没有指向任何结点。若p==head（选项B），则表示p与头指针指向相同，而头指针指向的是头结点（虚拟结点），在非空链表中，尾结点不是头结点。如果不是循环链表，则尾结点的next值为NULL（选项D）。

13．带头结点的单链表head为空的判定条件是\_\_\_\_\_\_\_\_。

A．head==NULL B．head!=NULL

C．head->next==head D．head->next==NULL

答案：D。

带头结点的空单链表是仅含有头结点（虚拟结点）的链表，头结点的next值为NULL，即它没有后继结点。因为含有头结点，故头指针head在任何时刻都不能为空（选项A错误）。对于空表与非空表，head!=NULL（选项B）都是成立的，所以用这个条件不能判定表是否为空。head->next==head表明头结点的next指针又指向头结点，这是空循环链表的状态。head->next==NULL表示的是头指针指向的结点没有后继结点，这正是空链表要满足的条件。

14．在双向循环链表中，在指针p所指结点之后插入指针s所指结点的操作是\_\_\_\_\_\_\_\_。

A．p->next=s;s->prev=p;p->next->prev=s;s->next=p->next;

B．p->next =s;p->next->prev=s; s->prev=p;s->next=p->next;

C．s->prev=p; s->next=p->next; p->next=s; p->next->prev=s;

D．s->prev=p; s->next=p->next;p->next->prev=s; p->next=s;

答案：D。

解答这样的题目时，画图是个不错的方法。比如，执行了选项A中前两个语句（p->next=s;s->prev=p;）后，得到的链表状态如图2-1所示。

p

…

A

B

…

X

s

图2-1 对应于选项A的单链表

其中，第3个语句“p->next->prev=s;”是将结点X的prev指针又指回自己，这显然是错误的。其他的3个选项，可以画出类似的链表状态。

15．在一个单链表中，已知q所指结点是p所指结点的前驱结点，若在q和p之间插入s结点，则执行的操作是\_\_\_\_\_\_\_\_。

A．s->next=p->next;p->next=s; B．p->next=s->next; s->next=p;

C．q->next=s;s->next=p; D．p->next=s;s->next=q;

答案：C。

以选项A为例，执行了其中的两条语句后，单链表的状态如图2-2所示。这表明，结点s插入在了结点p的后面，而不是插入在q和p之间。

p

q

s

………

B

C

…

X

图2-2 对应于选项A的单链表

16．在一个单链表中，若p所指结点不是终端结点，在p所指结点之后插入s所指结点，则执行的操作是\_\_\_\_\_\_\_\_。

A．s->next=p;p->next=s; B．s->next=p->next; p->next=s;

C．s->next=p->next; p=s; D．p->next=s;s->next=p;

答案：B。

17．下列关于带头结点的单向循环链表L的叙述中，正确的是\_\_\_\_\_\_\_\_。

A．L中最后一个结点的指针域要指向头结点

B．循环链表的长度为L中数据元素个数加1

C．因为L是环形的，所以无法计算链表长度

D．L中最后一个结点的指针域指向首个数据结点

答案：A。

带头结点的单向循环链表的基本形态如图2-3所示，不论是否有数据结点，最后一个结点的指针均是指向头结点的（选项A正确，选项D错误）。链表的长度为所包含的数据元素的个数，与链表的长度为L中数据元素个数加1是否有头结点无关（选项B错误），与链表中是否有环无关（选项C错误）。

b)带头结点的非空单向循环链表

L

A

B

C

D

a)带头结点的空单向循环链表

L

图2-3 带头结点的单向循环链表

18．含6个元素的线性表保存在如下所示的数组中，它表示的是一个静态\_\_\_\_\_\_\_\_。

|  |  |  |
| --- | --- | --- |
|  | 表元素关键字 | 表元素之间的联系 |
| 0 | 6 | 3 |
| 1 | 618 | 5 |
| 2 | 205 | 1 |
| 3 | 103 | 4 |
| 4 | 501 | 2 |
| 5 | 781 | 6 |
| 6 | 910 | 0 |
| 7 |  |  |

A．单链表 B．双向链表

C．单向循环链表 D．双向循环链表

答案：C。

静态链表是使用一维数组实现的链表。在本题中，使用“表元素之间的联系”来反映各个结点之间的关系（链接），其作用类似链表中的next。数组中保存了7个值，线性表中的元素值是6个，表明数组下标0处保存的特殊结点。根据给出的表元素之间的关系，可以画出如图2-4所示的链表。

head

205

1

618

5

910

0

781

6

2

501

103

4

6

3

图2-4 带头结点的单向循环链表

根据图示，链表为带有头结点的单向循环链表，头结点（虚拟结点）是下标为0的元素。

二、填空题

1．线性表是一个有限序列，组成线性表的是*n*（*n*≥0）个\_\_\_\_\_\_\_\_\_\_。

答案：同类型的数据元素。

线性表的定义要求组成线性表的数据元素是同类型的。

2．不带头结点的单链表L（head为头指针）为空的判定条件是\_\_\_\_\_\_\_\_\_\_。

答案：head==NULL。

不带头结点的空单链表中一个结点都没有，头指针的值为NULL。

3．带头结点的双向循环链表L（head为头指针）为空的条件是\_\_\_\_\_\_\_\_\_\_。

答案：head->next==head或是head->prev==head

带头结点的空双向循环链表中仅有一个头结点，且这个结点的next指针和prev指针都指向结点本身，即与head指向相同。

4．在带头结点的单链表中，当删除某一指定结点时，必须要找到该结点的\_\_\_\_\_\_\_\_\_\_。

答案：前驱。

被删除结点的后继结点，变为被删除结点原前驱结点的后继，所以要找到前驱结点，并修改它的next值。

5．在数组中保存的链表称为\_\_\_\_\_\_\_\_\_\_。

答案：静态链表。

基本概念。

6．设一个非空的静态链表保存在数组中，以-1表示表尾所在结点为表尾结点，则含-1的单元个数最多为\_\_\_\_\_\_\_\_\_\_。

答案：2。

7．若线性表中每个元素占*size*个单元，采用顺序存储方式，第一个数据元素的存储位置是1000，则线性表第*i*个元素的存储位置Loc(ai)=\_\_\_\_\_\_\_\_\_\_。

答案：1000+(*i*-1)×size。

在顺序存储方式下，数组下标与线性表元素的位置的对应关系可以通过公式LOC(ai)=LOC(a0)+*i*×*d*（教材公式2-2）进行计算。根据题目，第一个数据元素是线性表中的a0，即第1个数据元素保存在数组下标为0的位置，公式中的*d*为*size*，则第*i*个元素保存在数组下标为*i*-1的位置，元素*i*的存储位置LOC(ai)=1000+(*i*-1)×*size*。

三、解答题

1．在单链表中设置头结点的作用是什么？

【参考答案】在单链表中进行插入和删除操作时，都要找到操作位置结点的前驱结点。但根据线性表当前指针的原始定义，前驱结点是不能通过当前指针直接找到的。所以，修改当前指针指向的结点，让其前移一个位置，即指向操作位置的前驱结点。这样修改后，通过当前指针很容易访问到插入和删除操作所涉及的各个结点。但当操作位置是第一个结点时，或是单链表是空表时，当前指针的指向又成为特殊情况，需要单独处理。为了能统一处理各种情况，给单链表添加一个虚拟结点，即头结点，从而减少实现插入和删除操作时对特殊情况的判定及处理情况。

2．什么是单链表中的头结点、第一个数据结点和头指针？

【参考答案】单链表中保存线性表中各数据的结点是数据结点，这些结点中的第一个是第一个数据结点。它的前面是一个虚拟结点，称为头结点。一个特殊的指向单链表中最前面结点的指针是头指针。如果单链表中一个结点都没有，则头指针为空。

3．假设线性表L=(51,40,19,15,24,36)，使用线性表ADT中定义的方法，列出删除19时对应的语句序列。

【参考答案】

if(isEmpty(&L)==FALSE){

int pos,x;

pos=find(&L,19);

if(pos>=0)removeList(&L,pos,&x);

}

4．已知线性表中一个元素占8个字节，一个指针占2个字节，数组的大小为20个元素。在顺序及链式两种存储方式中，线性表应该采用哪种方式保存？为什么？

【参考答案】设线性表中元素个数为*n*。单纯考虑空间复杂度。

采用顺序存储时，占用的空间大小=20×8=160个字节。

采用链式存储时，不妨设采用单链表存储，占用的空间大小=*n*×(8+2)=10*n*个字节。

列方程，10*n*=160，求得*n*=16。即若线性表中元素个数少于16个时，采用单链表保存更好，元素个数多于16个时，采用数组保存更好。等于16个时，两种方式保存的效果是一样的。

5．在单链表和双向链表中，能否从当前结点出发访问到表中的任何一个结点?

【参考答案】在单链表和双向链表中，如果是从头结点或第一个数据结点出发，能访问到表中的任一结点，但从其他结点出发，都只能访问到该结点及其全部的后继结点，它的前驱结点是访问不到的。

6．线性表(a0,a1,…,an-1)采用顺序存储方式时，ai和ai+1（0≤i<n-1）的物理位置相邻吗？采用链式方式时呢？

【参考答案】采用顺序存储方式时，ai和ai+1（0≤i<n-1）的物理位置是相邻的。采用链式存储时，ai和ai+1（0≤i<n-1）的物理位置不一定是相邻的。可能相邻也可能不相邻。

7．试分析双向链表中插入、删除、查找等基本操作的时间复杂度。

【参考答案】在双向链表中进行插入和删除操作时，如果给定了当前指针，则插入操作和删除操作的时间复杂度均为*O*(1)，因为操作过程中不需要进行元素的移动，也不需要将当前指针从表头后移到当前位置。查找操作的时间复杂度是根据查找目标在链表中的位置而定的。若在双向链表中一定能找到查找目标，则最优情况下，比较1次就能找到，时间复杂度为*O*(1)。最坏的情况下，需要进行*n*次比较，时间复杂度为*O*(*n*)。平均来看，需要查找约一半的链表，所以时间复杂度是*O*(*n*)。

8．静态链表中，如何同时保存单链表及空闲空间链表？

【参考答案】：静态链表采用的一片连续区域（数组）作为链表的存储空间。为有效使用存储空间，提高空间的使用效率，对于数组中所有未曾使用过的单元必定是保持一片连续单元，这样可以使用整型变量unused记录这片连续单元的首位置，凡是下标大于等于unused且不越界的单元，均为空闲单元。

对于数组中曾经使用过且目前空闲的所有单元，建立一个静态的空闲单元链表将其链接起来，用整型变量available记录这个链表首结点的下标。

9．静态链表中，unused的作用是什么？

【参考答案】：静态链表采用的一片连续区域（数组）作为链表的存储空间。为有效使用存储空间，提高空间的使用效率，对于数组中所有未曾使用过的单元必定是保持一片连续单元，这样可以使用整型变量unused记录这片连续单元的首位置，凡是下标大于等于unused且不越界的单元，均为空闲单元。

四、算法阅读题

1．在如主教材图2-13b所示的带头结点的非空循环链表中保存了若干整数。算法average求出这些整数的平均值。请在空白处填上适当内容将算法补充完整。

float average(LinkList head)

{

LinkNode \*p;

int counter=0,sum=0;

if(head==NULL){

printf("链表错误\n");

return FALSE;

}

if( （1） ) printf("这是一个空链表\n");

else{

p=head->next;

while( （2） ){

counter++;

sum+=p->data;

p=p->next;

}

}

return （3） ;

}

【参考答案】

（1）head->next==head

（2）p!=head

（3）sum\*1.0/counter

对于（1），根据后面输出的内容可知，这里需要补上判定空表的条件。带头结点的空循环链表只含有一个头结点，且该结点的next域指向结点本身。

对于（2），根据题意，循环体中要对表中的全部结点进行处理。进入循环之前，指针p指向了第一个数据结点。while循环的结束条件是什么呢？当指针p转回到表头时，表明已经处理完全部的数据结点，此时循环可以结束。也就是在p没有指向头结点时，循环不结束。当p==head时表明p指向头结点，当p!=head时表明p没有指向头结点。

对于（3），返回的是表中元素的平均值。while循环体中已经计算了全部元素的和及元素个数，两个值做除法就可以了。注意不要进行整除，两个整数相除，结果可能除不尽，是个实型值。

2．单链表的结点及链表定义如下：

typedef struct node{

int data; //数据域

struct node \*next; //指针域

}LinkNode;

typedef LinkNode \*LinkList; //单链表

算法largest找出这些整数中的最大值。请在空白处填上适当内容将算法补充完整。

int largest(LinkList head)

{

LinkNode \*p;

int maxitem;

if(head==NULL){

printf("链表错误\n");

return FALSE;

}

if( （1） ) printf("这是一个空链表\n");

else{

p=head->next;

maxitem=p->data;

p=p->next;

while(p!=NULL){

if( （2） ) （3） ;

p=p->next;

}

}

return maxitem;

}

【参考答案】

（1）head->next==NULL

（2）p->data>maxitem

（3）maxitem=p->data

对于（1），空单链表中仅含有一个头结点，头指针指向结点的next域为NULL。

根据题意，要记录表中元素的最大值，这由if语句来完成。显然，if的条件是筛选最大值，后面的语句是将最大值保存下来。return语句中返回的是maxitem的值，意味着这个变量是用来保存最大值的，所以（3）的内容很容易写下来，就是将当前结点中的值赋给maxitem。那么，当前结点需要满足什么条件呢？是要大于原来保存的maxitem值，这是（2）要填写的内容。

3．阅读程序，并回答下列问题。

int counterofodd(LinkList head)

{

LinkNode \*p;

int counter=0;

if(head==NULL){

printf("链表错误\n");

return FALSE;

}

if(head->next==NULL) printf("这是一个空链表\n");

else{

p=head->next;

while(p!=NULL){

if(p->data%2!=0) counter++;

p=p->next;

}

}

return counter;

}

（1）若单链表head中保存的是(1,3,5,7,10)，则执行counterofodd(head)后，返回的结果是什么？

（2）counterofodd()的功能是什么？

【参考答案】

（1）返回的结果是4。

（2）counterofodd()的功能是统计单链表head中保存的奇数的个数。

五、算法设计题

1．教材中表2-1给出一个线性表操作示例，请采用顺序存储方式保存线性表，编程使用基本操作验证表中的结论。

【参考答案】

验证程序的main函数实现如下。

//采用顺序表存储，验证基本操作

#include <stdio.h>

#include "SeqList.h"

int main(int argc, char \*\*argv)

{ LinearList listtest;

int i, temp;

printf("采用顺序表保存线性表\n");

if ( initList( &listtest )==FALSE ) {

printf(" 初始化失败\n");

return 0;

} else printf(" 完成初始化\n");

for ( i=0; i<6; i++ )

if ( !insertList(&listtest,i,2\*i) ) printf("插入错误\n");

display( &listtest );

if ( !removeList(&listtest, 3, &temp) ) printf("删除错误\n");

else {

printf("删除的3号元素是：%d\n", temp);

printf("删除后，");

display( &listtest );

}

if ( !setValue(&listtest, 3, -10) ) printf("修改值出错\n");

else {

printf("修改3号元素的值，");

display( &listtest );

}

if ( !getValue(&listtest, 3, &temp) ) printf("获取值出错\n");

else printf("获取3号元素的值：%d\n", temp);

printf("查找 10 的结果: %d\n", find(&listtest, 10));

printf("查找 9 的结果: %d\n", find(&listtest, 9));

return 0;

}

2．教材中表2-1给出一个线性表操作示例，请采用链式存储方式保存线性表，编程使用基本操作验证表中的结论。

【参考答案】

验证程序的main函数实现如下。

//采用带头结点的单链表验证基本操作

#include <stdio.h>

#ifndef LinearList

#define LinearList

#include "LinkedList.h"

#endif

//typedef LinkList LinearList;

int main(int argc, char \*\*argv)

{

LinkList head, pre;

int i, temp;

Position pos;

printf("采用单链表保存线性表\n");

if ( initList( &head )==FALSE ) {

printf(" 初始化失败\n");

return 0;

} else printf(" 完成初始化\n");

for ( i=0; i<6; i++ )

if ( !insertList(&head, i, 2\*i) )

printf("插入错误\n");

display( &head );

if ( !removeList(&head, 3, &temp) ) printf("删除错误\n");

else {

printf("删除的3号元素是：%d\n", temp);

printf("删除后，");

display( &head );

}

if ( !setValue(&head, 3, -10) ) printf("修改值出错\n");

else {

printf("修改3号元素后：");

display( &head );

}

if ( !getValue(&head, 3, &temp) ) printf("获取值出错\n");

else printf("获取3号元素的值：%d\n", temp);

printf("查找 10 的结果: %d\n", find(&head, 10));

printf("查找 9 的结果: %d\n", find(&head, 9));

return 0;

}

3．教材中表2-1给出一个线性表操作示例，请采用静态链式存储方式保存线性表，编程使用基本操作验证表中的结论。

【参考答案】

验证程序的main函数实现如下。

#include <stdio.h>

#include "ArrayList.h"

int main(int argc, char \*\*argv)

{ ArrayList listtest;

int temp, i;

initList( &listtest );

displayList( listtest );

for ( i=0; i<6; i++)

if ( !insertArrayList(&listtest, i, 2\*i) )

printf("插入错误\n");

displayList(listtest);

displayArray(listtest);

if ( !removeArrayList(&listtest, 3, &temp))

printf("删除错误\n");

else{

printf("删除的元素是：%d\n", temp);

printf("删除后，");

displayList( listtest );

}

if ( !setValue(&listtest, 3, -10) ) printf("修改值出错\n");

else{

printf("修改3号元素的值，");

displayList( listtest );

}

if ( !getValue(&listtest, 3, &temp) ) printf("获取值出错\n");

else printf("获取3号元素的值：%d\n", temp);

printf("查找 10 的结果: %d\n", find(&listtest, 10));

printf("查找 9 的结果: %d\n", find(&listtest, 9));

displayList(listtest);

displayArray(listtest);

return 0;

}

4．教材中表2-1给出一个线性表操作示例，请采用带头结点的双向链表存储方式保存线性表，编程使用基本操作验证表中的结论。

【参考答案】

int main(int argc, char \*\*argv)

{ ArrayList listtest;

int temp, i, a[]={ 618, 205, 103, 501, 781, 910};

Position pos;

initList( &listtest );

printf("初始化静态数组后的状态：\n");

displayArray( listtest );

for ( i=0; i<6; i++)

if ( !insertSortArrayList(&listtest, a[i]) )

printf("插入错误\n");

displayList(listtest);

displayArray(listtest);

if ( (pos=find(&listtest, 501))!=ERROR )

{

if ( !removeArrayList(&listtest, pos, &temp))

printf("删除错误\n");

else{

printf("删除的元素是：%d\n", temp);

printf("删除501后，");

displayList( listtest );

}

}

if ( (pos=find(&listtest, 103))!=ERROR )

{

if ( !removeArrayList(&listtest, pos, &temp))

printf("删除错误\n");

else{

printf("删除的元素是：%d\n", temp);

printf("删除103后，");

displayList( listtest );

}

}

printf("静态数组状态：\n");

displayArray(listtest);

if ( !insertSortArrayList(&listtest, 301) )

printf("插入错误\n");

printf("插入 301 之后：");

displayList(listtest);

printf("静态数组状态：\n");

displayArray(listtest);

return 0;

}

5．设有一个正整数序列组成的有序单链表（按递增次序有序，且允许有相等的整数存在），试编写能实现下列功能的算法：

（1）确定在序列中比正整数x大的数有几个（相同的数只计算一次，如有序序列{10,20,30,30,40,41,50,50,51}中比30大的数有4个）；

（2）将单链表中比正整数x小的数按递减次序重排，仍放置在单链表中；如保存有序序列{10,20,30,30,40,41,50,50,51}的单链表中，将比40小的数重排后，得到的单链表中保存的序列是{30,30,20,10,40,41,50,50,51}。

（3）将比x大的偶数从单链表中删除。

【参考答案】

（1）程序实现如下所示。

int largerthanx(LinkList head,int x)

{

int counter=0,lastone;

LinkNode \*p;

if(head==NULL){

printf("链表错误\n");

return FALSE;

}

if(head->next==NULL) printf("这是一个空链表\n");

else{

p=head->next;

lastone=p->data-1;

while(p!=NULL){

if(p->data>x ){

if(p->data>lastone){

counter++;

lastone=p->data;

}

}

p=p->next;

}

}

return counter;

}

因为单链表有序，所以，如果有相同的整数，则它们一定是相邻的。使用变量lastone记录前一个整数，如果当前整数与lastone相等，则不计数。lastone的初值设为比第一个整数小的任何数即可。

（2）程序实现如下所示。

int reversesmallthanx(LinkList head,int x)

{

LinkNode \*left,\*middle,\*right,\*temp;

if(head==NULL){

printf("链表错误\n");

return FALSE;

}

if(head->next==NULL){

printf("这是一个空链表\n");

return TRUE;

}

else{

temp=left=head->next;

if(left!=NULL) middle=left->next;

while(middle!=NULL&&middle->data<x){

right=middle->next;

middle->next=left;

left=middle;

middle=right;

}

if(middle==NULL){

temp->next=NULL;

head->next=left;

}

else{

temp->next=middle;

head->next=left;

}

}

return TRUE;

}

将单链表中比正整数x小的数按递减次序重排，实际上是将单链表前面的若干结点逆置。与教材第二章第五节中给出的reverse有些类似，但情况更复杂一些。当找不到比x小的元素时，单链表维持不变。当所有元素均小于x时，整个链表逆置。当部分元素小于x时，只逆置前面的这些元素，逆置后的子表还要与原来的后半部分子表拼起来。

（3）程序实现如下所示。

int removeevenlargethanx(LinkList head,int x)

{

LinkNode \*current;

int k,pos=0;

if(head==NULL){

printf("链表错误\n");

return FALSE;

}

if(head->next==NULL){

printf("这是一个空链表\n");

return TRUE;

}

else{

current=head;

while(current->next!=NULL){

if(current->next->data>x && current->next->data%2==0){

removeList(&head,pos,&k);

}

else{

current=current->next;

pos++;

}

}

}

return TRUE;

}

删除满足条件的元素时，可以调用单链表中实现的remove操作。要注意的是，删除一个结点后，当前指针current不改变，此时，current不后移，继续判断它指向结点的后继结点。但如果没有删除结点，则current需要后移一个位置。

6．在单链表L中保存了若干整数。实现算法将L分为两个单链表L1和L2，其中L1中保存奇数，L2中保存偶数。

【参考答案】

程序实现如下所示。

int partition(LinkList \*L,LinkList \*L1,LinkList \*L2)

{

LinkNode \*L1curr,\*L2curr;

LinkNode \*temp;

if((\*L)==NULL){

printf("链表错误\n");

return 0;

}

initList(L1);

initList(L2);

if((\*L)->next==NULL){

printf("这是一个空链表\n");

return TRUE;

}

else{

temp=(\*L)->next;

L1curr=\*L1;

L2curr=\*L2;

while(temp!=NULL){

(\*L)->next=temp->next;

temp->next=NULL;

if(temp->data%2==0){

L2curr->next=temp;

L2curr=L2curr->next;

(\*L2)->data++;

}

else{

L1curr->next=temp;

L1curr=L1curr->next;

(\*L1)->data++;

}

temp=(\*L)->next;

}

}

return TRUE;

}

将单链表L拆分为两个单链表L1和L2，实际上就是从L中删除一个结点，然后根据结点中值的奇偶性，或是将结点添加到L1中，或是将结点添加到L2中。

因为删除和插入都是依次进行的，所以，设置一个当前工作指针记录三个链表的当前位置。

7．已知带头结点的单链表的每个结点中保存一个整数，数据结点有*n*个。请设计算法以判断该链表中保存的值是否构成斐波那契数列中的前*n*项。若是算法返回1，否则返回0。

【参考答案】

程序实现如下所示。

int isFibonacci(LinkList head)

{

LinkNode \*p1,\*p2,\*p3;

int f1=1,f2=1,f3,i;

if(head==NULL){

printf("链表错误\n");

return FALSE;

}

if(head->next==NULL){

printf("这是一个空链表\n");

return FALSE;

}

if(head->data==1 && head->next->data==1) return TRUE;

if(head->data==2&&head->next->data==1 && head->next->next->data==1)

return TRUE;

p1=head->next;

if(p1->data!=1) return FALSE;

p2=p1->next;

if(p2->data!=1) return FALSE;

p3=p2->next;

while(p3!=NULL && p3->data==p1->data+p2->data){

p1=p2;

p2=p3;

p3=p3->next;

}

if(p3==NULL) return 1;

else return FALSE;

}

斐波那契数列中含有多个整数值，如果整数的个数少于3个，则为特殊情况，程序中需要对此进行特殊的处理。即单链表中数据结点的个数少于3个时，需要单独进行判定。前两个结点中的整数都是1。

当结点个数大于等于3个时，从第三个结点开始，一直到尾结点，判定每个结点中保存的整数是否是相邻的前两个结点中值的和。

8．实现算法，判断带头结点的单链表中所保存的整数，是否构成递增有序序列。若是递增有序序列，则算法返回1；若不是有序序列，则算法返回0。

【参考答案】程序实现如下所示。

int isincrease(LinkList head)

{

LinkNode \*p;

int currmax;

if(head==NULL){

printf("链表错误\n");

return FALSE;

}

if(head->next==NULL){

printf("这是一个空链表\n");

return TRUE;

}

p=head->next; //第一个数据结点

currmax=p->data;

p=p->next;

while(p!=NULL && p->data>=currmax){

currmax=p->data;

p=p->next;

}

if(p==NULL) return TRUE;

else return FALSE;

}

9．设线性表元素为整数，其中可能有相同的元素。试设计一个算法删除表中重复的元素（即相同元素只保留一个），使删除后表中各元素均不相同。给出在顺序存储和链式存储两种方式下的实现程序。

【参考答案】线性表如果有序，则很容易删除表中的重复元素。但当线性表无序时，需要进行更多的比较才能实现。

采用顺序存储方式的程序实现如下所示。

int unique(SeqList \*L)

{

int i,j,k,len;

int NA=-65535; //NA用来做标记

len=L->n;

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

for(j=0;j<i;j++){

if(L->element[i]==L->element[j]) L->element[i]=NA;

}

k=1;

for(i=1;i<len;i++){

if(L->element[i]!=NA){

L->element[k++]=L->element[i];

}

}

L->n=k;

return TRUE;

}

从数组中删除一个重复元素时，后面的元素都要依次向前移动。为避免过多的元素移动，当发现重复元素时，仅对重复元素进行标记，而不是立即进行删除。当查找到所有重复元素后，再将剩余元素依次保存到数组最前面的若干位置中。

使用NA标记要删除的元素。

采用链式存储方式的程序实现如下所示。

void unique(LinkList head)

{

LinkNode \*p,\*current;

int k,pos=0,currpos=0;

if(head==NULL){

printf("链表错误\n");

return;

}

if(head->next==NULL){

printf("这是一个空链表\n");

return;

}

if(head->data==1) return;

current=head;

pos=1;

currpos=0;

while(current!=NULL && current->next!=NULL){

p=current;

pos=currpos;

while(p->next!=NULL){

if(p->next->data==current->data){

removeList(&head,pos,&k);

}

else{

p=p->next;

pos++;

}

}

current=current->next;

currpos++;

}

return;

}

current是当前指针，指示当前结点。使用另一个指针p从当前位置开始向后依次扫描结点。当p指向结点的后继结点中的值与current所指结点中的值相等时，删除重复结点，即删除p所指结点的后继结点。

可以调用removeList函数完成删除操作。因为删除函数的第二个参数是位置，所以使用currpos记录current指向的结点的位置值，使用pos记录p指向结点的位置值。当p指向表尾时，表示一轮扫描完成，current指向下一个结点，继续同样的查重及删除操作。

10．设有两个按元素值递增有序排列的单链表A和B。试实现算法，将它们合并成一个单链表C，要求C的元素值仍按递增有序排列（允许C中有值相同的元素）。

【参考答案】程序实现如下所示。

void mergelist(LinkList \*A,LinkList \*B,LinkList \*C)

{

LinkNode \*pA=\*A,\*pB=\*B,\*pC=\*C,\*temp;

while(pA->next!=NULL && pB->next!=NULL){

if(pA->next->data<pB->next->data){

temp=pA->next;

pA->next=temp->next;

temp->next=NULL;

pC->next=temp;

(\*A)->data--;

}

else{

temp=pB->next;

pB->next=temp->next;

temp->next=NULL;

pC->next=temp;

(\*B)->data--;

}

pC=pC->next;

(\*C)->data++;

}

if(pA->next==NULL){

pC->next=pB->next;

pB->next=NULL;

(\*C)->data+=(\*B)->data;

}

else{

pC->next=pA->next;

pA->next=NULL;

(\*C)->data+=(\*A)->data;

}

return;

}

因为两个单链表A和B是递增有序的，合并后的链表也要求是递增有序的，所以可以分别从前向后扫描A和B。使用三个工作指针分别指向A、B和C的当前位置。比较A和B当前结点中元素的值，将更小值的结点移至链表C中。这个思想是二路归并排序中两个有序归并段合并的思想。可以参考第7章第五节的相关内容。

11．已知带头结点的单链表的结点定义如下：

typedef int ELEMType;

typedef struct node{ //单链表结点

ELEMType data; //数据域

struct node \*next; //指针域

}LinkNode;

typedef LinkNode \*LinkList; //单链表

请编写函数int appendList( LinkList head,ELEMType x )，将x插入到head指向的单链表的表尾。

【参考答案】

int appendList(LinkList \*head, ELEMType x)

{ LinkNode \*p,\*temp;

p=(LinkList)malloc(sizeof(LinkNode));

p->data=x;

p->next=NULL;

temp=\*head;

while(temp->next!=NULL){

temp=temp->next;

}

temp->next=p;

return TRUE;

}

12．已知链表结点及带头结点的单链表定义如下：

typedef int ELEMType;

typedef struct node{ //单链表结点

ELEMType data; //数据域

struct node \*next; //指针域

}LinkNode;

typedef LinkNode \*LinkList; //单链表

请编写函数int deleteNode( LinkList head, ElemType x )，将在head指向的单链表中删除data等于x的结点，如果删除成功，函数返回值为1，删除失败，返回值为0。

【参考答案】

int deleteNode(LinkList \*head,ELEMType x)

{ LinkNode \*p=\*head,\*temp;

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

{

p=p->next;

}

if(p!=NULL&&p->next!=NULL&&p->next->data==x)

{ temp=p->next;

p->next=p->next->next;

free(temp);

return TRUE;

}

else return FALSE;

}

13．判定给定的单链表中是否存在环？如果存在，函数返回1，否则返回0。

【参考答案】

int isRing(LinkList \*head)

{

LinkNode \*p1=\*head,\*p2=\*head;

if(\*head==NULL) {

printf("链表错误\n");

return ERROR;

}

if((\*head)->next==NULL){

printf("这是一个空链表\n");

return FALSE;

}

if(p2->next->next!=NULL){

p2=p2->next->next;

p1=p1->next;

}

else return FALSE;

while(p1!=p2&&p2!=NULL){

p2=p2->next;

if(p2!=NULL){

p1=p1->next;

p2=p2->next;

}

else return FALSE;

}

if(p1==p2) return TRUE;

else return FALSE;

}

# 第三章 栈和队列

一、单项选择题

1．栈操作数据的原则是\_\_\_\_\_\_\_\_。

A．先进先出 B．后进先出 C．后进后出 D．随机处理

答案：B。

栈的特性是后进先出。选项A是队列的特性。有些情况下，可以将选项C理解为先进先出，这也是队列的特性。选项D既不是栈的特性，也不是队列的特性。

2．入栈序列是a1,a3,a5,a2,a4,a6，出栈序列是a5,a4,a2,a6,a3,a1，则栈的容量最小是\_\_\_\_\_\_\_\_。

A．2 B．3 C．4 D．5

答案：C。

用图3-1表示栈状态的变化情况。

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a1入栈 |  | a3入栈 |  | a5入栈 |  | a5出栈 |  | a2入栈 |  | a4入栈 |
|  |  |  |  |  |  |  |  |  |  | a4 |
|  |  |  |  | a5 |  |  |  | a2 |  | a2 |
|  |  | a3 |  | a3 |  | a3 |  | a3 |  | a3 |
| a1 |  | a1 |  | a1 |  | a1 |  | a1 |  | a1 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a4出栈 |  | a2出栈 |  | a6入栈 |  | a6出栈 |  | a3出栈 |  | a1出栈 |
|  |  |  |  |  |  |  |  |  |  |  |
| a2 |  |  |  | a6 |  |  |  |  |  |  |
| a3 |  | a3 |  | a3 |  | a3 |  |  |  |  |
| a1 |  | a1 |  | a1 |  | a1 |  | a1 |  |  |

图3-1 栈的状态

可以看到，在a4入栈后，栈中有4个元素，这是栈含元素最多的时刻。所以栈的容量最小是4。

3．6个元素6,5,4,3,2,1依次进栈，不能得到的出栈序列是\_\_\_\_\_\_\_\_

A．5,4,3,6,1,2 B．4,5,3,1,2,6 C．3,4,6,5,2,1 D．2,3,4,1,5,6

答案：C。

可以使用操作过程来描述。

得到选项A的操作过程是：push(6),push(5),pop(5),push(4),pop(4),push(3),pop(3), pop(6),push(2),push(1),pop(1),pop(2)。

得到选项B的操作过程是：push(6),push(5),push(4),pop(4),pop(5),push(3),pop(3), push(2),push(1),pop(1),pop(2),pop(6)。

得到选项D的操作过程是：push(6),push(5),push(4),push(3),push(2),pop(2),pop(3), pop(4),push(1),pop(1),pop(5),pop(6)。

对于选项C，push(6),push(5),push(4),push(3),pop(3),pop(4)，此时栈顶元素是5，得不到元素6。

4．输入序列为A,B,C，借助于栈将其变为C,B,A，经过的栈操作为\_\_\_\_\_\_\_\_。

A．push,pop,push,pop,push,pop B．push,push,push,pop,pop,pop

C．push,push,pop,pop,push,pop D．push,pop,push,push,pop,pop

答案：B。

对应于选项A的操作序列，得到的出栈序列是A,B,C。对应于选项C的操作序列，得到的出栈序列是B,A,C。对应于选项D的操作序列，得到的出栈序列是A,C,B。

实际上，出栈序列与入栈序列互逆，所以必然是将所有元素依次入栈，然后依次出栈。

5．若一个栈存储在数组V[n]中，设栈空时栈顶指针top=n-1，则下列选项中，能正确表示x入栈的是\_\_\_\_\_\_\_\_。

A．top=top+1;V[top]=x; B．V[top]=x;top=top+1;

C．V[top]=x;top=top-1; D．top=top-1;V[top]=x;

答案：C。

根据题意，栈顶指针指向的是栈顶元素旁边的空位置，所以入栈时，元素x直接写入top所示的位置，然后top移至下一个空位置。又因为栈空时top指向的是数组下标最大处，则意味着入栈时top应向下标减小的方向改变。

6．用不带头结点的单链表存储队列时，队头指针指向队头结点，队尾指针指向队尾结点，则在进行删除操作时\_\_\_\_\_\_\_\_。

A．仅需要修改队头指针，不需要修改队尾指针

B．仅需要修改队尾指针，不需要修改队头指针

C．队头指针一定要修改，队尾指针也一定要修改

D．队头指针一定要修改，队尾指针可能要修改

答案：D。

用不带头结点的单链表保存队列时，通常需要两个指针分别指向队头结点和队尾结点。出队时，要从队头删除一个结点，此时必须修改队头指针，让它指向被删结点的后继结点。如果队列中还有其他结点，则队尾指针不会变化。但如果删除结点后队列变为空队列，则还需要修改队尾指针。

7．下列链队列的操作中，不可能修改队尾指针的是\_\_\_\_\_\_\_\_。

A．入队操作 B．清空队列操作

C．出队操作 D．判队列是否为空

答案：D。

入队操作肯定会修改队尾指针。出队操作通常仅修改队头指针，但如果出队后队列为空，则也会修改队尾指针。清空队列的操作既修改队头指针，也修改队尾指针。而判队列是否为空，仅需要查看指针的值，不会进行修改。

8．循环队列Q（最多元素为MaxSize），front指向队头元素所在位置，rear指向队尾元素后的空位置，则循环队列为满的条件是\_\_\_\_\_\_\_\_。

A．Q.rear==Q.front B．Q.rear+1==Q.front

C．Q.rear==Q.front+1 D．(Q.rear+1)%MaxSize==Q.front

答案：D。

基本概念。参考教材中图3-14及图3-16。

9．用不带头结点的单链表存储队列，其头指针指向队头结点，尾指针指向队尾结点，则在进行出队操作时应进行的操作是\_\_\_\_\_\_\_\_。

A．仅修改头指针 B．头、尾指针均不需要修改

C．仅修改尾指针 D．头、尾指针都可能要修改

答案：D。

出队操作肯定要修改队头指针，如果出队后队列为空，则队尾指针也要修改。

10．假设以数组A[0..m-1]存放循环队列的元素，队头指针front指向队头元素，队尾指针rear指向队尾元素后的空位置，则当前队列中的元素个数为\_\_\_\_\_\_\_\_。

A．(rear-front+m)%m B．rear-front+1

C．(front-rear+m+1)%m D．(rear-front)%m

答案：A。

循环队列有两种基本形态，如图3-2a和图3-2b示。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | a0 | … | am-1 |  |  |
|  |  | ↑  front |  |  | ↑  rear |  |
| a)循环队列示意图一 | | | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| … | v |  | … | … | u | … |
|  |  | ↑  rear |  |  | ↑  front |  |
| b)循环队列示意图二 | | | | | | |

对于图3-2a，队列中的元素个数=rear-front。对于图3-2b，队列中的元素个数=rear+m-front。两个表达式可以统一表示为(rear-front+m)%m。

11．栈和队列的共同点是\_\_\_\_\_\_\_\_。

A．都是后进先出 B．都是先进先出

C．只允许在端点处插入和删除元素 D．没有共同点

答案：C。

栈具有后进先出的特点，而队列具有先进先出的特点。所以选项A和B都不正确。栈的插入和删除操作都在表的一端进行，队列的插入和删除分别在表的两端进行。所以，选项C中概括的特点是栈和队列共同具有的。

12．循环队列存储在数组A[0..m]中，入队时修改队尾指针的操作为\_\_\_\_\_\_\_\_。

A．rear=rear+1 B．rear=(rear+1)%(m-1)

C．rear=(rear+1)%m D．rear=(rear+1)%(m+1)

答案：D。

根据题意，数组A的容量是m+1，用来保存循环队列。入队操作时队尾指针需要后移一个位置，即rear=rear+1。当rear已经指向A[m]时，此时入队操作需要将rear修改为0，即rear=(rear+1)%(m+1)。两个表达式可以统一表示为rear=(rear+1)%(m+1)。

13．若用一个大小为6的数组来实现循环队列，且当前队尾rear和队头front的值分别为0和3，当从队列中删除一个元素，再加入两个元素后，rear和front的值分别为\_\_\_\_\_\_\_\_。

A．1和5 B．2和4 C．4和2 D．5和1

答案：B。

在循环队列中插入一个元素时，rear后移一个位置。删除一个元素时，front后移一个位置。根据题意，rear和front的值如图3-3所示。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| ↑ rear |  |  | ↑ front |  |  |
| 图3-3 当前循环队列 | | | | | |

删除一个元素后，rear和front的值如图3-4所示。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| ↑ rear |  |  |  | ↑ front |  |
| 图3-4 删除一个元素后 | | | | | |

再加入两个元素后，rear和front的值如图3-5所示。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  | ↑ rear |  | ↑ front |  |
| 图3-5 再加入二个元素后 | | | | | |

14．某队列允许在其两端进行入队操作，但仅允许在一端进行出队操作。若元素a,b, c,d,e依次入此队列后再进行出队操作，则不可能得到的出队序列是\_\_\_\_\_\_\_\_。

A．b,a,c,d,e B．d,b,a,c,e C．d,b,c,a,e D．e,c,b,a,d

答案为C。

仅在线性表的端点位置进行插入、删除操作的线性表是受限的线性表。仅允许在一端进行插入、在另一端进行删除的线性表称为队列；仅允许在两端进行插入、删除的线性表称为双端队列；该题给出的是“允许在两端进行入队操作，但仅允许在一端进行出队操作”的线性表，这是一种输出受限的双端队列。

为简单起见，以1、2分别代表队列的两端，以I(1,x)表示元素x从队列1端入队，以I(2,x)表示元素x从队列2端入队。题目中所给的队列是输出受限的双端队列，不失一般性，假定仅允许在1端出队列，以O(1,x)表示元素x出队。

选项A、B和D中的出队序列，均可对应至少一种操作序列。

对应于选项A的一种操作序列（不唯一）如下：

I(1,a),I(1,b),I(2,c),I(2,d),I(2,e),O(1,b),O(1,a),O(1,c),O(1,d),O(1,e)

对应于该操作序列中的入队操作部分，队列状态变化的示意如下所示（每一行表示队列的一种状态）。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | a |  |  |  |  |  |
|  |  | b | a |  |  |  |  |  |
| 1 |  | b | a | c |  |  |  | 2 |
| 端 |  | b | a | c | d |  |  | 端 |
|  |  | b | a | c | d | e |  |  |

对应于选项B的一种操作序列（不唯一）如下：

I(1,a),I(1,b),I(2,c),I(1,d),I(2,e),O(1,d),O(1,b),O(1,a),O(1,c),O(1,e)

对应于该操作序列的入队操作部分，队列状态变化的示意如下所示。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | a |  |  |  |  |
|  |  |  | b | a |  |  |  |  |
| 1 |  |  | b | a | c |  |  | 2 |
| 端 |  | d | b | a | c |  |  | 端 |
|  |  | d | b | a | c | e |  |  |

对应于选项D的一种操作序列（不唯一）如下：

I(1,a),I(1,b),I(1,c),I(2,d),I(1,e),O(1,e),O(1,c),O(1,b),O(1,a),O(1,d)

对应于该操作序列的入队操作部分，队列状态变化的示意如下所示。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | a |  |  |  |
|  |  |  |  | b | a |  |  |  |
| 1 |  |  | c | b | a |  |  | 2 |
| 端 |  |  | c | b | a | d |  | 端 |
|  |  | e | c | b | a | d |  |  |

下面来说明，无论哪种操作序列，都得不到选项C中的出队序列。

选项C对应的出队序列中，第1个出队元素是d。这表示在d出队时刻之前，元素a、b、c、d均已入队且都仍在队中。现在我们考虑d之前的3个元素的入队情况。

题目中所给的队列允许在两端进行入队操作，所以元素a、b、c可以在队列的任一端入队。入队完成后，队列的状态有以下4种可能情况：

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| b、c在a的同侧入队 |  |  |  | c | b | a |  |  |  |
| b、c在a的同侧入队 | 1 |  |  | a | b | c |  |  | 2 |
| b、c在a的异侧入队 | 端 |  |  | b | a | c |  |  | 端 |
| b、c在a的异侧入队 |  |  |  | c | a | b |  |  |  |

元素d仍可在队列的两端入队，入队后，可得到以下共8种队列状态：

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| d在1端入队 |  |  | d | c | b | a |  |  |  |
|  | 1 |  | d | a | b | c |  |  | 2 |
|  | 端 |  | d | b | a | c |  |  | 端 |
|  |  |  | d | c | a | b |  |  |  |
| d在2端入队 |  |  |  | c | b | a | d |  |  |
|  | 1 |  |  | a | b | c | d |  | 2 |
|  | 端 |  |  | b | a | c | d |  | 端 |
|  |  |  |  | c | a | b | d |  |  |

对应以上这8种队列状态，均不能得到以“d,b,c”开头的出队序列，即不可能得到“d,b,c,a,e”的出队序列。

综上，选项C中的出队序列是得不到的。

二、填空题

1．如果对一个线性表的访问进行限制，要求对元素的插入、删除、读取等操作只能在线性表的一端进行，则该线性表称为\_\_\_\_\_\_\_\_\_\_。

答案：栈。

2．队列的最基本特性是\_\_\_\_\_\_\_\_\_\_。

答案：先进先出。

3．如果对一个线性表访问进行限制，要对元素的插入、删除、读取等操作只能在线性表的两端进行，则该线性表称为\_\_\_\_\_\_\_\_\_\_。

答案：队列。

4．设局域网中含有多台计算机与一台网络打印机，通常打印机中会设置一个打印数据缓冲区以满足多个打印任务的需求，该缓冲区的逻辑结构应该是\_\_\_\_\_\_\_\_\_\_。

答案：队列。

网络打印机通常按先来先服务的原则提供打印服务，符合队列的特性。

5．设栈S和队列Q的初始状态均为空，元素a,b,c,d,e,f依次进入S。若每个元素出栈后立即进入Q，且6个元素出队的顺序是b,d,c,f,e,a，则S的容量至少是\_\_\_\_\_\_\_\_\_\_。

答案：3。

对于队列来说，出队序列与入队序列是一样的。由题意，出队顺序是b,d,c,f,e,a，意味着入队序列也是b,d,c,f,e,a。而这个序列即是栈的出栈序列。实际上，我们是要计算当入栈序列是a,b,c,d,e,f，出栈序列是b,d,c,f,e,a时，占用的栈空间最大是多少。

用图3-6表示栈状态的变化过程。

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a入栈 |  | b入栈 |  | b出栈 |  | c入栈 |  | d入栈 |  | d出栈 |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | d |  |  |
|  |  | b |  |  |  | c |  | c |  | c |
| a |  | a |  | a |  | a |  | a |  | a |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| c出栈 |  | e入栈 |  | f入栈 |  | f出栈 |  | e出栈 |  | a出栈 |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | f |  |  |  |  |  |  |
|  |  | e |  | e |  | e |  |  |  |  |
| a |  | a |  | a |  | a |  | a |  |  |

图3-6 栈的状态

6．已知大小为10的循环队列Q，其头指针和尾指针分别为2和6，当Q的头指针和尾指针变为6和2时，入队列的元素个数至少有\_\_\_\_\_\_\_\_\_\_。

答案：6。

7．已知大小为10的循环队列Q，其头指针和尾指针分别为2和6，当Q的头指针和尾指针变为6和2时，出队列的元素个数至少有\_\_\_\_\_\_\_\_\_\_。

答案：4。

三、解答题

1．对于一个栈，设元素入栈的次序为：A,B,C,D,E，并给定下列各序列：

（1）A,B,C,D,E （2）B,C,D,E,A

（3）E,A,B,C,D （4）E,D,C,B,A

哪些是可以得到的出栈序列？若要得到相应序列，需要执行哪些栈操作？使用本章给出的基本操作写出操作序列。若其中某些输出序列不可能得到，试说明理由。

【参考答案】（1）、（2）和（4）都是可以得到的出栈序列。

对于（1），执行的栈操作序列是push(A),pop(A),push(B),pop(B),push(C),pop(C), push(D),pop(D),push(E),pop(E)。

对于（2），执行的栈操作序列是push(A),push(B),pop(B),push(C),pop(C),push(D), pop(D),push(E),pop(E),pop(A)。

对于（4），执行的栈操作序列是push(A),push(B),push(C),push(D),push(E),pop(E), pop(D),pop(C),pop(B),pop(A)。

（3）中的序列是得不到的。为得到出栈元素E，需要执行栈操作push(A),push(B), push(C),push(D),push(E),pop(E)，此时，栈顶是元素D，得不到元素A。

2．有5个元素，其入栈次序为：A,B,C,D,E，在各种可能的出栈序列中，以元素C,D最先出栈（即C第一个出栈，且D第二个出栈）的序列有哪些？

【参考答案】为了让C第一个出栈，D第二个出栈，需要执行栈操作push(A),push(B), push(C), pop(C),push(D),pop(D)，此时，栈中有两个元素A和B，且还有元素E尚未入栈。元素A和B的出栈相对次序只能是B,A，在这个过程中元素E可以入栈，然后出栈。这三个元素的出栈序列有：E,B,A、B,E,A和B,A,E。以元素C,D最先出栈的序列有C,D,E,B,A、C,D,B,E,A和C,D,B,A,E。

3．将两个栈存入数组V[0..m-1]中，应如何安排最好？这时栈空、栈满的条件分别是什么？

【参考答案】设两个栈为S1和S2，将S1和S2保存在一个一维数组中，可以选择对顶栈的方式，将数组的两端分别当作两个栈的栈底，数组的中间位置作为栈可使用的空间，如图3-7所示。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | … | *k*+1 |  |  |  | *h*-1 | … | *m*-1 |
|  |  |  |  |  |  |  |  |  |
| ←S1栈底 |  | ←S1栈顶 |  |  |  | ←S2栈顶 |  | ←S2栈底 |
| 图3-7 对顶栈示意图 | | | | | | | | |

初始时，栈S1的栈顶指针top1在V[0]，栈S2的栈顶指针top2在V[m-1]，栈空的条件是top1==0 && top2==m-1。当数组中没有空闲单元时，栈满，如图3-8所示。栈满的条件是top1==top2+1。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | … |  |  | … | *m*-1 |
|  |  |  |  |  |  |
|  |  | top2 | top1 |  |  |
| 图3-8 对顶栈满状态示意图 | | | | | |

4．简述顺序存储队列时假溢出的避免方法及队列满和空的条件。

【参考答案】将队列保存在一维数组中时，为使插入和删除操作的效率更高，规定插入和删除操作后不移动队列中的数据元素。因此，数据保存在数组从某一下标开始的一段连续单元中，并使用两个变量front和rear分别记录这段连续单元的首尾位置。当占用的这段连续空间移至数组的最后面，也就是记录尾位置的rear到达数组最大下标处时，会出现通常意义下的数组“满”状态。但数组最前面的位置可能是空闲的，所以这个“满”是假溢出。可以使用数组最前面空出的位置保存后续入队列的元素，即使用循环队列来避免顺序队列的假溢出。为了能分辨出循环队列的空与满，规定循环队列中保存的最多元素个数比数组最大容量少1。

队列满的判定条件是：(rear+1)%*n*==front，其中*n*是数组最大容量。

队列空的判定条件是：rear==front。

5．当rear==front时，为了能区分循环队列的空与满，教材中使用“让数组中始终剩余至少一个空位置”的解决方法。你还有其他的解决方法吗？

【参考答案】有两种替代的解决方法。

方法一，使用一个变量录循环队列的最后一个操作是入队还是出队。变量的类型可以是布尔类型、枚举类型或是整型。只要能区分两种情况即可。如果最后一个操作是入队操作，则当rear==front时，循环队列为满。如果最后一个操作是出队操作，则当rear==front时，循环队列为空。

方法二，使用一个整型变量记录队列长度。当到达数组最大容量时，循环队列为满。

6．简要叙述循环队列的数据结构，并写出其初始状态、队列空、队列满时的队头指针与队尾指针的值。

【参考答案】循环队列采用最大容量为maxSize的一维数组保存队列中的元素，并使用两个变量front和rear作为队头指针与队尾指针，分别记录队列的两端。其数据结构定义如下：

typedef int ELEMType;

typedef struct{

ELEMType element[maxSize];

int front,rear;

}SeqQueue;

初始状态时，队头指针front==0，队尾指针rear==0。

队列空时，rear==front。

队列满时，(rear+1)%maxSize==front。

7．假设循环队列的rear指向队尾元素本身，则队列空或满的条件分别是什么？

【参考答案】队尾指针rear指向队尾元素，队头指针front指向队头元素，则初始时，队头指针front==0，队尾指针rear==*n*-1，其中*n*是数组最大容量。

队列空时，(rear+1)%maxSize==front。

队列满时，(rear+2)%maxSize==front。

8．已知，两个栈S1和S2共享存储空间C(1,M)，其中S1的栈底在C[1]处，S2的栈底在C[M]处，top1是S1的栈顶指针，top2是S2的栈顶指针，均指向各自的栈顶元素。两个栈要最大限度地利用存储空间C(1,M)。请回答下列问题。

（1）判断两个栈空的条件分别是什么？

（2）判断栈满的条件是什么？

【参考答案】

（1）判断栈S1空的条件是：当top1=0时S1为空栈。

判断栈S2空的条件是：当top2=M+1时S2为空。

（2）栈满的条件是：当top2=top1+1时栈满。

9．已知循环队列的类型定义如下：

#define MAXQSIZE 100 //队列最大长度

typedef struct {

char \*base; //初始化的动态分配存储结构

int front; //头指针，若队列不空，指向队列头元素

int rear; //尾指针，若队列不空，指向队尾元素的下一个位置

}SqQueue;

请回答下列问题。

（1）判断队空的条件是什么？

（2）判断队满的条件是什么？

（3）计算队列的长度公式是什么？

【参考答案】

（1）判队空的条件：Q.front==Q.rear

（2）判队满的条件：Q.front==(Q.rear+1)%MAXQSIZE

（3）计算队列长度：(Q.rear-Q.front+MAXQSIZE)%MAXQSIZE

10．画出调用expression函数，将中缀表达式A\*(B+C)\*D转换为后缀表达式的过程中，运算符栈的变化。

【参考答案】

|  |  |  |  |
| --- | --- | --- | --- |
| 步骤 | 读入的当前符号 | 后缀形式 | 运算符栈（栈底到栈顶） |
| 1 | NULL | NULL | NULL |
| 2 | A | A | NULL |
| 3 | \* | A | \* |
| 4 | ( | A | \*( |
| 5 | B | AB | \*( |
| 6 | + | AB | \*(+ |
| 7 | C | ABC | \*(+ |
| 8 | ) | ABC+ | \* |
| 9 | \* | ABC+\* | \* |
| 10 | D | ABC+\*D | \* |
| 11 | NULL | ABC+\*D\* | NULL |

四、算法阅读题

1．给出对顶栈的定义如下。

typedef int ELEMType; //以int类型为例

typedef struct {

ELEMType element[maxSize]; //maxSize是数组最大容量，已定义的常量

int lefttop,righttop; //两个栈顶位置

}SeqTopStack;

其中，栈中的元素保存在数组element中，lefttop和righttop分别是左栈和右栈的栈顶指针。

以下程序实现了对顶栈的若干基本操作，请在空白处填上适当内容将算法补充完整。

（1）initStack函数初始化左栈和右栈。

int initStack(SeqTopStack \*mys) //初始化两个栈

{ mys->lefttop= （1） ;

mys->righttop= （2） ;

return 1;

}

【参考答案】

（1）0

（2）maxSize-1

初始化栈，就是设置栈顶指针，栈顶指针指向栈顶元素的下一个空位置。空栈时，栈顶指针为0。对于对顶栈，数组的两端分别作为栈底，左栈的栈顶指针指向位置0，右栈的栈顶指针指向最大位置maxSize-1。

（2）isEmpty函数判定指定的栈若为空则返回1，否则返回0。如果flag为-1时判定左栈，flag为1时判定右栈。

int isEmpty(SeqTopStack mys,int flag)

{ if(flag==-1 && （1） ) return 1;

if(flag==1 && （2） ) return 1;

else （3） ;

}

【参考答案】

（1）mys.lefttop==0

（2）mys.righttop==maxSize-1

（3）return 0

因为数组中同时保存两个栈，所以判定空栈时需要根据flag的指示分别判定。只要相应的栈顶指针位于初始位置（0及maxSize-1），即可表明相应的栈为空栈。

（3）isFull函数判定若栈满返回1，否则返回0。

int isFull(SeqTopStack mys)

{ if( （1） ) （2） ;

else return 0;

}

【参考答案】

（1）mys.lefttop==mys.righttop+1

（2）return 1

数组的空间被两个栈共同使用，所以栈满的判定不针对左栈或右栈。只要数组中没有空闲空间了，无论是左栈还是右栈，都是满的。

2．给出对顶栈的定义如下。

typedef int ELEMType; //以int类型为例

typedef struct{

ELEMType element[maxSize]; //maxSize是数组最大容量，已定义的常量

int lefttop,righttop; //两个栈顶位置

}SeqTopStack;

其中，栈中的元素保存在数组element中，lefttop和righttop分别是左栈和右栈的栈顶指针。

说明clear函数的功能。

int clear(SeqTopStack \*mys,int flag)

{ if(flag==-1) mys->lefttop=0;

else if(flag==1) mys->righttop=maxSize-1;

else{

mys->lefttop=0;

mys->righttop=maxSize-1;

}

return 1;

}

【参考答案】clear函数的功能是清空栈，flag为-1时清空左栈，flag为1时清空右栈，flag为0时清空两个栈。

五、程序设计题

1．现使用一个数组存储两个对顶栈，试实现入栈操作。

【参考答案】程序实现如下所示。

int push(SeqTopStack \*mys,ELEMType x,int flag)

//将元素x入栈，flag=-1入左栈，flag=1入右栈

{ if(!isFull(mys))

if(flag==-1){ //入左栈

mys->element[mys->lefttop++]=x;

}

else{ //入右栈

mys->element[mys->righttop--]=x;

}

else {

printf("栈满\n");

return 0;

}

return 1;

}

入栈时需要指明是入左栈还是入右栈。入左栈的操作与通常的入栈操作是类似的。入右栈的操作有些差别，入栈后栈顶指针向下标0的方向变化。

2．现使用一个数组存储两个对顶栈，试实现出栈操作。

【参考答案】程序实现如下所示。

int pop(SeqTopStack \*mys,ELEMType \*x,int flag)

//返回栈顶元素，flag=-1出左栈，flag=1出右栈

{ if(!isEmpty(mys,flag))

if(flag==-1){

\*x=mys->element[--mys->lefttop];

return 1;

}

else{

\*x=mys->element[++mys->righttop];

return 1;

}

else{

printf("栈空\n");

return 0;

}

}

出栈时需要指明是从左栈出栈还是从右栈出栈。从左栈出栈的操作与通常的出栈操作是类似的。从右栈出栈的操作有些差别，出栈后栈顶指针向下标maxSize-1的方向变化。

3．定义循环队列的rear指向队尾元素本身，实现入队和出队操作。

【参考答案】程序实现如下所示。

typedef int ELEMType;

typedef struct {

ELEMType element[maxSize];

int front, rear;

}SeqQueue;

int enqueue(SeqQueue \*myq, ELEMType x) //x入队

{ if(isFull(\*myq)==1) return 0;

myq->element[(++myq->rear)%maxSize]=x;

return 1;

}

int dequeue(SeqQueue \*myq, ELEMType\* x) //出队，并通过x返回

{ if(isEmpty(\*myq)==1) return 0;

\*x=myq->element[myq->front++];

return 1;

}

循环队列的rear指向队尾元素本身，不只影响入队和出队操作，还会影响队列其他基本操作的实现细节。请考生自行完成其他操作的实现。

4．给定的入栈序列和出栈序列，请编程判断对于入栈序列是否能够仅凭栈操作得到指定的出栈序列。

判断入栈序列是否能仅凭栈操作得到指定的出栈序列的函数checkStackStr实现和main函数实现如下。

#include <stdio.h>

#include <string.h>

#define ELEMTYPE

typedef char ELEMType;

#include "SeqStack.h"

int checkStackStr(char \*, char \*);

int checkStackStr(char \*instr, char \*outstr)

{

SeqStack mys; //使用顺序栈

ELEMType temp;

initStack(&mys);

if ( strlen(instr)!=strlen(outstr) ) //长度不等则返回FALSE

return FALSE;

while ( \*instr!='\0' && \*outstr!='\0' ) {

if ( isEmpty(&mys)==TRUE ) {

printf("push(%c)\n", \*instr);

push(&mys, \*instr++); //字符先入栈，然后指针instr增1

}

else {

gettop(&mys, &temp); //读取栈顶元素

if ( temp==\*outstr ) { //若当前字符与栈顶字符相同，则出栈

pop(&mys, &temp);

printf("pop(%c)\n", \*outstr++);

}

else { //若当前字符与栈顶字符不同，则入栈

printf("push(%c)\n", \*instr);

push(&mys, \*instr++);

}

}

}

if ( \*instr=='\0' && \*outstr!='\0' ) {

while ( isEmpty(&mys)==FALSE ) {

pop(&mys, &temp);

if ( temp==\*outstr ) {

printf( "pop(%c)\n", \*outstr++);

}

else return FALSE;

}

if ( isEmpty(&mys)==TRUE && \*outstr=='\0' )

return TRUE;

}

else if ( \*instr=='\0' && \*outstr=='\0' ) {

return TRUE;

}

return FALSE;

}

int main(int argc, char \*\* argv)

{ char instr[100], outstr[100];

printf("请输入入栈序列："); scanf("%s", instr);

printf(" 输入出栈序列："); scanf("%s", outstr);

if ( checkStackStr(instr, outstr)==TRUE )

printf("%s 是合法的出栈序列\n", outstr);

else

printf("%s 不是合法的出栈序列\n", outstr);

return 0;

}

# 第四章 数组、广义表和串

一、单项选择题

1．假定一个二维数组的定义语句为int a[3][4]={{3,4},{2,8,6}};，则元素a[1][2]的值为\_\_\_\_\_\_\_\_。

A．8 B．6 C．4 D．2

答案：B。

由定义可知，数组a的状态如下所示。

a[1][2]代表的是第2行第3列的元素6。

2．在C语言中，设有数组定义：char array[ ]="China";，则数组array所占用的空间为\_\_\_\_\_\_\_\_。

A．4个字节 B．5个字节 C．6个字节 D．7个字节

答案：C。

使用数组保存字符串常量时，系统自动添加结束符'\0'。给array赋值的字符串常量含5个字符，占5个字节，再加上结束符占一个字节，总共需要6个字节。

3．关于主对角线（从左上角到右下角）对称的矩阵为对称矩阵；如果一个矩阵中的各个元素取值为0或1，那么该矩阵可称为0-1矩阵。大小为*n*×*n*的0-1对称矩阵的个数是\_\_\_\_\_\_\_\_。

A．pow(2,*n*) B．pow(2,*n*×*n*/2)

C．pow(2,(*n*×*n*+*n*)/2) D．pow(2,(*n*×*n*-*n*)/2)

答案：C。

*n*×*n*的0-1矩阵的元素个数是*n*×*n*，对称矩阵中上三角部分与下三角部分对称相等，上三角部分或下三角部分再加上主对角线的元素个数是1+2+…+*n*=*n*×(*n*+1)/2=(*n*×*n*+*n*)/2。每个元素取值为0和1，有2种，故不同的对称矩阵个数为2(n×n+n)/2。pow函数是C语言中计算幂次的函数，pow(x,y)返回*x*的*y*次幂的值。

4．有一个二维数组A[10][5]，每个数据元素占1个字节，按行主序保存，且A[0][0]的存储地址是1000，则A[i][j]的地址是\_\_\_\_\_\_\_\_。

A．1000+10*i*+*j* B．1000+*i*+*j* C．1000+5*i*+*j* D．1000+10*i*+5*j*

答案：C。

二维数组A[10][5]对应的矩阵为10行5列。采用按行主序的方式保存时，A[i][j]的前面已经保存了0行、1行、…、*i*-1行（共*i*行）的元素，每行有5个，共5*i*个元素。当前行中还有*j*个元素排在前面。每个元素占1个字节，故A[i][j]相对于数组首地址的偏移量是5*i*+*j*，再加上首地址，就是A[i][j]的地址。

5．数组通常具有的两种基本操作是\_\_\_\_\_\_\_\_。

A．查找和修改 B．查找和索引 C．索引和修改 D．建立和删除

答案：A。

有些教材中，将数组下标称为索引，这是数组自身具有的特性，不是索引操作。在分块索引存储方式中，或是数据库中，都会建立索引。当数据有更新时，索引也需要更新。

6．将一个A[1..100, 1..100]的三对角矩阵以行主序存入一维数组B[1..298]中，元素a[66, 65]在数组B中的位置*k*等于\_\_\_\_\_\_\_\_。

A．195 B．196 C．197 D．198

答案：A。

对于三对角矩阵，采用压缩存储时只保存主角线及两条副对角线中的元素，按行主序保存时，除第一行和最后一行外，每行需保存3个元素。在保存a[66, 65]前，已经在B中保存了前65行的元素，元素个数=2+64\*3=194，a[66, 65]是第195个元素。

7．若采用三元组顺序表保存*m*×*n*有*t*个非零元素的稀疏矩阵A，则对矩阵A求转置的快速算法，最好能够达到的时间复杂度是\_\_\_\_\_\_\_\_。

A．*O*(*n*) B．*O*(*n*+*t*) C．*O*(*m*×*n*) D．*O*(*m*×*n*×*t*)

答案：B。

求转置算法中，需要对所有的三元组进行处理，处理*t*个三元组的时间复杂度是*O*(*t*)。为了能快速进行转置，算法中先统计转置后各行中的元素个数，时间复杂度是*O*(*n*)。

8．下列选项中，适合采用三元组表存储的是\_\_\_\_\_\_\_\_。

A．单向链表 B．双向链表 C．循环队列 D．稀疏矩阵

答案：D。

基本概念。

9．若稀疏矩阵采用三元组顺序表进行存储，除保存非零元素的信息之外，还需要保存的是

A．矩阵的行数、列数 B．矩阵的非零元素数量

C．矩阵的行数和非零元素数 D．矩阵的行数、列数和非零元素数

答案：D。

基本概念。

10．下面关于非空广义表的叙述中，错误的是\_\_\_\_\_\_\_\_。

A．广义表的表头一定不为空表 B．广义表是一个多层次的线性结构

C．广义表的表尾总是一个广义表 D．广义表中的数据元素有相对次序

答案：A。

当L=(())时，L非空，且表头是空表。

11．下列关于广义表L的叙述中，正确的是\_\_\_\_\_\_\_\_。

A．Tail(L)可以为空表 B．广义表不能是递归表

C．广义表L=()的长度为1 D．Head(L)的深度一定为0

答案：A。

当L中只有一个元素时，L的表尾即是空表。广义表可以是递归表，而空表的长度为0，深度为1。如果L的表头不是原子，则表头的深度不是0。

12．下列关于广义表的基本概念中，错误的是\_\_\_\_\_\_\_\_。

A．广义表中可包含原子和子表 B．空的广义表( )的长度为0

C．广义表的表头为一个子表 D．可以用广义表表示二叉树

答案：C。

广义表的表头可以是原子，不一定是子表。选项D用到了第五章的知识点。

13．已知广义表LS=(((a)),((b,(c)),(d,(e,f))),())，LS的长度是\_\_\_\_\_\_\_\_。

A．2 B．3 C．4 D．5

答案：B。

LS中，第一个元素是((a))，它也是表头，第二个元素是((b,(c)),(d,(e,f)))，第三个元素是()，所以长度是3。

14．已知广义表LS=(((a)),((b,(c)),(d,(e,f))),())，LS的深度是\_\_\_\_\_\_\_\_。

A．2 B．3 C．4 D．5

答案：C。

对LS中嵌套的括号进行计数，如图4-1所示。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ( | ( | ( | a | ) | ) | , | ( | ( | b | , | ( | C | ) | ) | , | ( | d | , | ( | e | , | f | ) | ) | ) | , | ( | ) | ) |
| 1 | 2 | 3 |  | 2 | 1 |  | 2 | 3 |  |  | 4 |  | 3 | 2 |  | 3 |  |  | 4 |  |  |  | 3 | 2 | 1 |  | 2 | 1 | 0 |

图4-1 对LS中嵌套的括号进行计数

可以看到，最大计数是4，由此得到LS的深度为4。

15．已知广义表LS=(((a)),((b,(c)),(d,(e,f))),())，Head(LS)的值是\_\_\_\_\_\_\_\_。

A．( ) B．a C．(a) D．((a))

答案：D。

16．已知广义表LS=(((a)),((b,(c)),(d,(e,f))),())，Tail(LS)的值是\_\_\_\_\_\_\_\_。

A．( ) B．(((b,(c)),(d,(e,f))),())

C．((b,(c)),(d,(e,f))) D．(( ))

答案：B。

二、填空题

1．采用静态方式压缩保存稀疏矩阵的方法是\_\_\_\_\_\_\_\_\_\_。

答案：三元组表法。

2．设有一维整型数组data，计算数组data元素数量的表达式是\_\_\_\_\_\_\_\_\_\_。

答案：sizeof(data)/sizeof(int)。

3．若*n*×*n*矩阵M是一个对称矩阵，Mij表示*i*行*j*列的元素，则对所有的*i*和*j*，Mij和Mji满足\_\_\_\_\_\_\_\_\_\_。

答案：Mij=Mji(*i*,*j*=0,1,…,*n*-1)。

4．现有广义表L=((a,b),(()),(a,(b)))，L中的元素个数是\_\_\_\_\_\_\_\_\_\_。

答案：3。

去掉最外层的括号，可以得到3个元素，分别是(a,b),(())和(a,(b))。

5．已知广义表L=(a,(b,(c,d),e))，Head(Tail(L))的结果是\_\_\_\_\_\_\_\_\_\_。

答案：(b,(c,d),e)。

先求L的表尾：Tail(L)=((b,(c,d),e))，再求这个中间结果的表头：Head(Tail(L))=(b,(c,d),e)。

6．已知广义表L=(a,(b,(c,d),e))，Head(Head(Tail(L)))的结果是\_\_\_\_\_\_\_\_\_\_。

答案：b。

Tail(L)=((b,(c,d),e))，Head(Tail(L))=(b,(c,d),e)，Head(Head(Tail(L)))=b。

7．在广义表中，取表头元素应采用的基本操作是\_\_\_\_\_\_\_\_\_\_。

答案：Head。

基本概念。

三、解答题

1．设有数组score[u1][u2][u3]，采用列主序方式保存，写出映射函数。

【参考答案】数组score[u1][u2][u3]是三维数组，按列主序保存时，映射函数为：

map(i1,i2,i3)=*i3*×*u*2×*u1*+*i*2×*u1*+*i1*

以D3Array[3][2][4]为例，按列主序下标排列的形式如图4-2所示。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0][0][0] | [1][0][0] | [2][0][0] | [0][1][0] | [1][1][0] | [2][1][0] |
| [0][0][1] | [1][0][1] | [2][0][1] | [0][1][1] | [1][1][1] | [2][1][1] |
| [0][0][2] | [1][0][2] | [2][0][2] | [0][1][2] | [1][1][2] | [2][1][2] |
| [0][0][3] | [1][0][3] | [2][0][3] | [0][1][3] | [1][1][3] | [2][1][3] |

图4-2 三维数组D3Array的下标排列形式

比如，保存元素D3Array[2][1][2]的单元，距离数组首地址的偏移量为：map(2,1,2)=2×2×3+1×3+2=17，与实际相符。保存D3Array[0][0][0]的单元编号为0的话，则D3Array[2][1][2]保存在编号为17的单元中。

2．有三对角矩阵A[*i*][*j*](0≤*i*,*j*≤*n*-1)如下所示。



采用行主序方式仅保存主对角线及两个副对角线中的元素。设元素A[*i*][*j*](0≤*i*,*j*≤*n*-1)保存在B[*k*](0≤*k*≤*n*(*n*+1)/2-1)中。给出地址映射函数。

【参考答案】

地址映射函数如下：

3．设有*n*×*n*对称矩阵A，aij=aji，如图所示。为了节约存储，只存对角线及对角线以上的元素。将上三角矩阵中的元素按列主序存放到一个一维数组B中。请问：

（1）存放对称矩阵A上三角部分的一维数组B至少需要有多少个元素？

（2）若在一维数组B中从下标为0号的位置开始存放，则矩阵A中的任一元素a*ij*在只存上三角部分的情形下，应存于一维数组的哪个下标位置*k*？给出*k*的计算公式。



【参考答案】

（1）存储上三角矩阵至少需要元素数量：*n*(*n*+1)/2

（2）*j*(*j*+1)/2+*i* (*i*<=*j*)

*i*(*i*+1)/2+*j* (*i*>*j*)

4．给出四维数组按行主序存储的地址映射函数。

【参考答案】对于四维数组FourDimenArray[u1][u2][u3][u4]，其行主序的映射函数应为：

map(i1,i2,i3,i4)=*i*1×*u*2×*u*3×*u*4+*i*2×*u*3×*u*4+*i*3×*u*4+*i*4

5．在模拟迷宫的应用时，规定了行走路径只考虑水平和垂直4个方向（上、下、左、右）。如果行走方向为8个，即除了水平和垂直方向外，还可以沿斜线走。具体来说，行走方向有正北、东北、正东、东南、正南、西南、正西、西北。则表示方向的move矩阵该如何定义？

【参考答案】move矩阵如下所示。

6．构造一个广义表，其表头和表尾相等。

【参考答案】R=(())，R的长度为1，深度为2。

Head(R)=()，Tail(R)=()。R的表头和表尾相等。

7．给定模式串="abcaabcacb"，计算特征向量。

【参考答案】

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| j | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| P | a | b | c | a | a | b | c | a | c | b |
| next(*j*) | -1 | 0 | 0 | 0 | 1 | 1 | 2 | 3 | 4 | 0 |

8．给定主串"abcaabaab"及模式串"abaab"，采用KMP算法进行模式匹配，计算字符之间的总比较次数。

【参考答案】先计算特征向量。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| j | 0 | 1 | 2 | 3 | 4 |
| P | a | b | a | a | b |
| next(*j*) | -1 | 0 | 0 | 1 | 1 |

模式匹配过程如下，使用下划线表示失配位置。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 主串 | a | b | c | a | a | b | a | a | b |  | *j* | next(*j*) | 比较次数 |
| 模式串 | a | b | a | a | b |  |  |  |  | 第1趟 | 2 | 0 | 3 |
|  |  |  | a | b | a | a | b |  |  | 第2趟 | 0 | -1 | 1 |
|  |  |  |  | a | b | a | a | b |  | 第3趟 | 1 | 0 | 2 |
|  |  |  |  |  | a | b | a | a | b | 第4趟 |  |  | 5 |

匹配成功，字符之间的总比较次数为3+1+2+5=11。

四、算法设计题

实现算法，对给定的模式串，计算KMP算法中特征向量next值。

【参考答案】程序实现如下所示。

void getnext(char \*p,int next[]){

int j=0,k=-1,i;

next[0]=-1;

while(j<strlen(p)){

while(k>=0 && p[j]!=p[k])k=next[k];

j++;

k++;

next[j]=k;

}

}

# 第五章 树与二叉树

一、单项选择题

1．下列关于二叉树的叙述中，正确的是\_\_\_\_\_\_\_\_。

A．二叉树的度为2 B．一棵二叉树的度可以小于2

C．二叉树中至少有一个结点的度为2 D．二叉树中任何一个结点的度都为2

答案：B。

树中结点的度的最大值是树的度。对于二叉树，每个结点的度可以是0、1或2。如果二叉树仅含有根结点，则它的度为0。如果二叉树中所有分支结点均仅含有一个子结点，则二叉树的度为1。如果二叉树中有的分支结点含有2个子结点，则二叉树的度为2。所以二叉树的度可以是0，可以是1，也可以是2。

2．下列存储形式中，不是树的存储形式的是\_\_\_\_\_\_\_\_。

A．父结点表示法 B．孩子结点表示法

C．孩子-兄弟表示法 D．顺序存储表示法

答案：D。

二叉树有顺序存储表示，它针对每层的最多结点数，在数组中预留出相应的位置。但对于树而言，因为每层结点的个数没有上限，所以没有办法在数组中预留空间。

另外三种存储形式都是树的存储形式。

3．下列关于二叉树的叙述中，正确的是\_\_\_\_\_\_\_\_。

I．只有一个结点的二叉树的度为0

II．二叉树的度为2

III．二叉树的左右子树可任意交换

IV．深度为K的完全二叉树的结点个数小于或等于深度相同的满二叉树

A．仅I、IV B．仅II、IV C．仅I、II、III D．仅II、III、IV

答案：A。

单结点的二叉树只含有根结点，这个结点也是叶结点，度为0，所以树的度为0。I是正确的。但二叉树中结点的度可以是0、1或2，所以树的度不一定必须是2。II不正确。二叉树中左子树与右子树是不同的，不能任意交换。III不正确。满二叉树是完全二叉树的特例，深度一样的完全二叉树与满二叉树，只在最后一层可能存在差异，在这一层上，完全二叉树可能是不满的，而满二叉树一定是满的，前面的所有层中，完全二叉树的结点个数与满二叉树的结点个数都是相同的。IV是正确的。

4．一棵二叉树共有20个结点，其中度为l的结点个数是7个，则叶结点个数是\_\_\_\_\_\_\_\_。

A．4 B．7 C．9 D．13

答案：B。

二叉树中，设*n*0是叶结点的个数，*n*1是度为1的结点的个数，*n*2是度为2的结点的个数，由性质3可知，*n*0=*n*2+1，由题目条件知*n*1=7，则*n*0+*n*2=20-*n*1=13，解得*n*0=7。

5．深度为4的二叉树中，结点个数最多是\_\_\_\_\_\_\_\_。

A．10 B．16 C．31 D．32

答案：C。

相同高度的满二叉树中结点个数最多。深度为4的满二叉树的高度是5。

6．对含*n*个结点的完全二叉树从上到下且从左至右进行1至*n*编号，根结点编号为1，则对树中任意一个编号为*i*的结点，正确的是\_\_\_\_\_\_\_\_。

A．若2*i*>=*n*，则结点*i*无左子结点

B．若2*i*+1>=*n*，则结点*i*无右子结点

C．若*i*为偶数，则*i*是其父结点的左子结点

D．若*i*为奇数，则*i*是其父结点的右子结点

答案：C。

按照题目中给出的编号规则，结点*i*的左子结点是2*i*，右子结点是2*i*+1，故选项A和B中，不等式中取等号时，结论是错误的。反过来看，偶数编号的结点是其父结点的左子结点，奇数编号的结点是其父结点的右子结点，唯一的例外是根结点，根的编号是奇数，没有父结点，所以选项D也是错误的。

7．采用顺序存储方式保存一棵二叉树，根结点保存在数组的A[0]中，若编号为*i*结点有右子结点，则该子结点的下标为\_\_\_\_\_\_\_\_。

A．2*i*-1 B．2*i* C．2*i*+1 D．2*i*+2

答案：D。

基本概念。下标从0开始。

8．在一非空二叉树的中序遍历序列中，根结点的右边\_\_\_\_\_\_\_\_。

A．只有左子树上的所有结点 B．只有右子树上的所有结点

C．只有左子树上的部分结点 D．只有右子树上的部分结点

答案：B。

中序遍历序列中，根的前面是左子树中所有结点的中序遍历序列，后面是右子树中所有结点的中序遍历序列。

9．若一棵满二叉树有*m*个叶结点，*n*个结点，高度为*h*，则下列关系式中正确的是\_\_\_\_\_\_\_\_。

A．*h*+*m*==2*n* B．*n*==*h*+*m* C．*m*==*h*-1 D．*n*==2*h*-1

答案：D。

满二叉树中，高度为*h*时，结点个数*n*=2*h*-1。

10．一棵二叉树高度为*h*，所有结点的度或为0或为2，则这棵二叉树的结点数最少为\_\_\_\_\_\_\_\_。

A．*h*+1 B．2*h*-1 C．2*h* D．2*h*+1

答案：B。

基本概念。没有度为1的结点，满二叉树中结点个数最多。

11．设*n*、*m*为二叉树T中的两个结点，在中序遍历序列中，*n*出现在*m*之前的条件是\_\_\_\_\_\_\_\_。

A．*n*在*m*左侧 B．*n*在*m*右侧 C．*n*是*m*子孙 D．*n*是*m*祖先

答案：A。

将二叉树中的各结点向下投影成一个序列，二叉树中若结点*n*在*m*的左侧，则中序遍历时先遍历到*n*后遍历到*m*。

12．由3个结点可以构造出不同形态的二叉树的数量是\_\_\_\_\_\_\_\_。

A．3个 B．4个 C．5个 D．6个

答案：C。

基本概念。

13．下列四个序列中，能构成最大堆的是\_\_\_\_\_\_\_\_。

A．75,65,30,15,25,45,20,10 B．75,65,45,10,30,25,20,15

C．75,45,65,30,15,25,20,10 D．75,45,65,10,25,30,20,15

答案：C。

将4个选项中给定的数据序列对应到完全二叉树中。如图5-1所示。可以看出，选项C对应的是堆，且是最大堆。

b)选项B对应的完全二叉树

a)选项A对应的完全二叉树

c)选项C对应的完全二叉树

d)选项D对应的完全二叉树

图5-1 各选项对应的完全二叉树

选项A中，元素30既小于其父结点75，也小于其左子结点45。选项B中，元素10既小于其父结点65，也小于其左子结点15。选项D中，元素10既小于其父结点45，也小于其左子结点15。都不符合堆的定义。

14．在13题的最大堆中删除堆顶后，得到的最大堆是\_\_\_\_\_\_\_\_。

A．65,30,15,25,45,20,10 B．65,45,25,30,15,10,20

C．65,45,30,15,25,20,10 D．65,45,10,25,30,20,15

答案：B。

图5-2 删除堆顶后的新堆

删除堆顶75后，由最后一个元素10递补，然后进行整堆，10先后与65、25进行交换，得到的新堆如图5-2所示。

15．在13题的最大堆中插入新元素55后，得到的最大堆是\_\_\_\_\_\_\_\_。

A．75,55,65,45,15,25,20,10,30 B．75,65,55,45,10,30,25,20,15

C．75,55,45,65,30,15,25,20,10 D．75,45,65,55,10,25,30,20,15

答案：A。

新元素55先保存在最后的位置，作为30的右子结点。由于55大于30，故两者相交换。同样的原因，55与45相交换。得到最终的新堆，如图5-3所示。

图5-3 插入新元素后的新堆

16．与树的后根遍历序列相同的是对应二叉树的\_\_\_\_\_\_\_\_。

A．先序遍历序列 B．中序遍历序列

C．后序遍历序列 D．按层遍历序列

答案：B。

基本概念。

17．设森林F中有3棵树，第1棵、第2棵和第3棵树的结点个数分别为M1、M2和M3。与森林F对应的二叉树根结点的右子树上的结点个数是\_\_\_\_\_\_\_\_。

A．M1 B．M3 C．M1+M2 D．M2+M3

答案：D。

森林F中的第2棵和第3棵树将转换为对应二叉树的右子树。所以二叉树根结点右子树上的结点个数等于第2棵和第3棵树中的结点个数之和。

18．一个非空二叉树的先序遍历序列与后序遍历序列正好互为逆序序列，则该二叉树的形状应为\_\_\_\_\_\_\_\_。

A．从树根结点开始左右子树相互对称 B．所有非叶结点只有向左的一个分支

C．所有非叶结点只有向右的一个分支 D．所有非叶结点全部是度为1的结点

答案：D。

二叉树退化为单链树时，先序遍历序列与后序遍历序列正好互为逆序序列。

19．设给定权值总数为*n*个，用其构造的哈夫曼树的结点总数为\_\_\_\_\_\_\_\_。

A．不确定 B．2*n*-1 C．2*n* D．2*n*+1

答案：B。

给定的权值为*n*个，意味着哈夫曼树中的叶子结点个数为*n*个。哈夫曼树中没有度为1的结点，由满二叉树定理可知，度为2的结点个数比叶结点个数少1，为*n*-1，故哈夫曼树中结点总数为*n*+*n*-1=2*n*-1。

20．下列编码集中，不具有前缀特性的是\_\_\_\_\_\_\_\_。

A．{0,10,110,1111} B．{00,10,011,110,111}

C．{00,010,0110,1000} D．{10,11,001,101,0001}

答案：D。

对于选项D中给出的编码方案，10是101的前缀，所以不具有前缀特性。对于其他3种编码方案，可以分别画出对应的编码树，以验证它们都具有前缀特性。例如，选项A所示编码方案对应的编码树树形如图5-4所示，可以看出，编码对应的各结点都是叶结点，以方形结点表示。类似地，选项B所示编码方案对应的编码树树形如图5-5所示。

图5-4 对应于选项A的编码树

图5-5 对应于选项B的编码树

21．一棵哈夫曼树共有215个结点，则树中编码的字符个数是\_\_\_\_\_\_\_\_。

A．107 B．108 C．215 D．214

答案：B。

二、填空题

1．具有18个结点的完全二叉树，它的高度是\_\_\_\_\_\_\_\_\_\_。

答案：5。

高度为4的满二叉树含有15个结点，小于18，所以含18个结点的完全二叉树，必有结点位于4层（3个结点），所以高度为5。

也可以根据二叉树的性质4进行推导。

具有*n*个结点的完全二叉树的高度为，根据题意，*n*=18，

。

2．已知完全二叉树的5层（根在0层）有8个叶结点，则整个二叉树的结点数最多是\_\_\_\_\_\_\_\_\_\_。

答案：111。

由完全二叉树的定义可知，叶结点只能出现在最后两层中。

根据题意，“5层有8个叶结点”，而最后两层中都可能有叶结点，所以5层既可能是指二叉树的最后一层（树共有6层），也可能是指树的倒数第二层（树共有7层）。而题目中要求含“结点个数最多”，显然，对于完全二叉树来说，7层的树所含的结点个数要多于6层的树所含的结点个数，所以该题中对应到的完全二叉树应该有7层。

前6层的结点总数

除掉8个叶结点外，5层还有25-8=24个分支结点，其中每一个结点都有2个子结点（位于6层中），所以6层还有24×2=48个叶结点，故该完全二叉树总的结点数*M*＝63+24×2=111。

3．已知一棵二叉树的先序遍历结果为A,B,C,D,E,F，中序遍历结果为C,B,A,E,D,F，则后序遍历的结果为\_\_\_\_\_\_\_\_\_\_。

答案：C,B,E,F,D,A。

给定二叉树的先序遍历序列和中序遍历序列，可以唯一还原该二叉树。根据题意，二叉树的先序遍历结果为A,B,C,D,E,F，意味着根是A。而其中序遍历结果为C,B,A,E,D,F，表明A的左子树中有C,B两个结点，右子树中有E,D,F三个结点。

对于含两个结点的左子树，其先序遍历序列是B,C，中序遍历序列是C,B。B是根，C是左子结点。

对于含三个结点的右子树，先序遍历是D,E,F，中序遍历是E,D,F。故右子树的根是D，左子结点是E，右子结点是F。

得到的二叉树如图5-6所示。

图5-6 习题3的二叉树

4．二叉树的先序遍历结果是E,F,H,I,G,J,K，中序遍历结果是H,F,I,E,J,K,G，该二叉树根的右子树的根是\_\_\_\_\_\_\_\_\_\_。

答案：G。

由先序遍历序列E,F,H,I,G,J,K可知，E是根。从中序遍历序列H,F,I,E,J,K,G中可知，右子树中含有J,K,G三个结点，而这三个结点的先序遍历序列是G,J,K，即G是该子树的根。

二叉树如图5-7所示。

图5-7 习题4的二叉树

5．若一棵二叉树具有10个度为2的结点，5个度为1的结点，则度为0的结点个数是\_\_\_\_\_\_\_\_\_\_。

答案：11。

根据二叉树的性质3（满二叉树定理），有*n*0=*n*2+1。由题意知，*n*2=10，故*n*0=11。与度为1的结点个数5无关。

6．设树T的度为4，其中度为1、2、3和4的结点个数分别为4、2、1和1，则T中的叶子结点个数为\_\_\_\_\_\_\_\_\_\_。

答案：8。

可以将满二叉树定理推广到任意*k*叉树中。设*k*叉树中叶结点个数为*n*0，度为1的结点个数为*n*1，…，度为*k*的结点个数为*n*k，则

。

根据题意，*k*=4，度为1的结点个数为4，度为2的结点个数为2，度为3的结点个数为1，度为4的结点个数为1个。叶结点个数=(2-1)×2+(3-1)×1+(4-1)×1+1=8。

7．与算术表达式a+b\*(c+d/e)对应的后缀表达式为\_\_\_\_\_\_\_\_\_\_。

答案：abcde/+×+。

使用第三章第三节介绍的表达式转换算法，可以得到对应于a+b\*(c+d/e)的后缀表达式。或是画出对应于表达式的表达式树，如图5-8所示，然后再进行后序遍历，也可以得到相应的后缀表达式。

图5-8 表示a+b\*(c+d/e)的表达式树

8．设森林F对应的二叉树为B，它有*m*个结点，B的根为p，p的右子树结点个数为*n*，则森林F中第一棵树的结点个数是\_\_\_\_\_\_\_\_\_\_。

答案：*m*-*n*。

森林F与其对应的二叉树B中结点数是相同的，都是m个结点。B的右子树是由F中除第一棵树外的其他树转换而来的。F中全部结点个数减去B中右子树中的全部结点个数，即为F中第一棵树中的结点个数。

9．已知二叉树的后序遍历序列为D,G,E,B,F,C,A，中序遍历序列为D,B,G,E,A,C,F，则先序遍历序列是\_\_\_\_\_\_\_\_\_\_。

答案：A,B,D,E,G,C,F。

二叉树如图5-9所示。

图5-9 习题9的二叉树

10．对于任何一棵二叉树，若它有*n*0个叶结点，*n*2个度为2的结点，则*n*0=\_\_\_\_\_\_\_\_\_\_。

答案：*n*2+1。

11．如果深度为*k*的二叉树为满二叉树，则树中结点的个数是\_\_\_\_\_\_\_\_\_\_。

答案：2k+1-1。

深度为*k*的满二叉树的高度*h*=*k*+1，满二叉树中的结点个数为2h-1。

12．若用二叉链表存储二叉树，则含有*n*个结点的二叉树中的空指针个数为\_\_\_\_\_\_\_\_\_\_。

答案：*n*+1。

13．设有含*n*个结点的完全二叉树，如果对结点按照自上到下、从左到右的次序从1开始进行连续的编号，则第*i*（2≤*i*≤*n*）个结点的父结点编号为\_\_\_\_\_\_\_\_\_\_。

答案：*i*/2。

14．设有含*n*个结点的完全二叉树，如果对结点按照自上到下、从左到右的次序从1开始进行连续的编号，则第*i*个结点有左子结点的条件是\_\_\_\_\_\_\_\_\_\_。

答案：2*i*≤*n*。

15．假设用孩子-兄弟表示法存储一个森林，则对森林进行的先序遍历，等同于对其孩子-兄弟链表进行的\_\_\_\_\_\_\_\_\_\_。

答案：先序遍历。

16．假设用孩子-兄弟表示法存储一个森林，则森林进行的后序遍历，等同于对其孩子-兄弟链表进行的\_\_\_\_\_\_\_\_\_\_。

答案：中序遍历。

三、解答题

1．请使用一棵树表示你所在学校的院系机构。

【参考答案】一个学校的院系机构数目众多，通常划分为专业院系、职能部门、直属单位及研究机构等几大类。每一类中选择一个单位，如图5-10所示。

图5-10 一个学校的院系机构

2．当使用一棵树描述本章目录（第五章、节、小节）时，请回答下列问题：

（1）树中共有多少个结点？

（2）每个结点的度分别是多少？

（3）树的深度是多少？

【参考答案】使用缩进形式描述第五章的目录，如下所示。

第五章 树与二叉树（5）

第一节 树的基本概念（0）

第二节 二叉树（2）

一、二叉树的定义及其主要特性（0）

二、二叉树的存储（0）

第三节 二叉树的操作（3）

一、二叉树的生成（0）

二、二叉树的遍历（0）

三、二叉树的应用（0）

第四节 树和森林（3）

一、树的存储结构（0）

二、树、森林与二叉树的转换（0）

三、树和森林的遍历（0）

第五节 哈夫曼树及哈夫曼编码（3）

一、编码（0）

二、哈夫曼树（0）

三、哈夫曼编码（0）

（1）根在0层，1个结点。本章共有5节，即1层有5个结点。第一节没有子结点，第二节到第五节分别有2、3、3、3个子结点。树中共有1+5+2+3+3+3=17个结点。

（2）各结点的度列在结点后的括号内。

（3）树的深度是2。

3．有4个结点的二叉树共有多少种不同的树形？分别画出。

【参考答案】有4个结点的二叉树共有14种不同的树形。所图5-11所示。

图5-11 含4个结点的不同二叉树树形

4．若某棵树有*n*1个度为1的结点，*n*2个度为2的结点，…，*n*m个度为*m*的结点，问这棵树共有多少个叶结点？给出推导过程。

【参考答案】叶结点个数。

满二叉树定理可以推广到任意*m*叉树中。

设*m*叉树中叶结点数为*n*0，度为1的结点数为*n*1，…，度为*m*的结点数为*n*m。结点总数，树中含有的分支数B=*n*-1。

而对于度为*k*的结点，可以带有*k*个分支，故树中分支数，将几个方程联立，有，整理得。

5．使用同样结构的结点构造的双向链表和二叉链表，有什么不同？

【参考答案】双向链表和二叉链表都是链式存储结构，它们的结点结构都是含有一个数据类型域及两个指针类型域，结构是相同的，但指针指向的含义是不同的。

构造的双向链表是线性结构，链表结点中的两个指针，一个指针指向其前驱，另一个指针指向其后继。链表中相邻的两个结点，可以分别通过向前的指针和向后的指针互相指示。

二叉链表是层次结构，结点中所含的两个指针都是指向子结点的，任意两个结点都不会出现相互指示的情况。

6．给出图5-12所示二叉树的先序、中序、后序遍历序列。

图5-12 习题6的二叉树

【参考答案】先序遍历序列：k,c,a,e,d,h,n,s,p。中序遍历序列：a,c,d,e,h,k,n,p,s。后序遍历序列：a,d,h,e,c,p,s,n,k。

7．现有某二叉树，其先序遍历序列是：A,B,C,D,E,F,G,H,I，中序遍历序列是：B,C,A, E,D,G,H,F,I，画出该二叉树。

【参考答案】得到的二叉树如图5-13所示。

图5-13 习题7的二叉树

8．将图5-12（习题6）所示的二叉树还原为森林。

【参考答案】将图5-12所示的二叉树还原为森林，如图5-14所示。

图5-14 图5-12所示的二叉树对应的森林

9．树的集合表示法能用来表示二叉树吗？修改集合表示法，使之能表示二叉树，以图5-43所示的二叉树为例，给出其集合表示。

【参考答案】

树的集合表示法不能直接表示二叉树。使用集合表示法表示树的一般形式如下：

(根结点,(子树1),(子树2),…,(子树n))

当根只有一棵子树时，树可以表示为(根结点,(子树1))。但对于二叉树，一棵子树既可能是左子树，也可能是右子树，使用集合表示法区分不了这种差别。

可以改进集合表示法，使用一个特殊符号作为占位符，表示空的子树。使用改进的集合表示法表示二叉树的一般形式如下：

(根结点,(左子树),(右子树))

即使子树为空，也需要表示出来，即要占位。图5-12（习题6）所示二叉树的集合表示如下所示：

(k,(c,(a,(),())(e,(d,(),())(h,(),()))),(n,(),(s,(p,(),()),())))

可以再简化这个表示方法。当子树为空时，可以使用一个特殊符号表示，例如使用#替代()。采用简化表示方式后，图5-12（习题6）所示二叉树的集合表示如下所示：

(k,(c,(a,#,#)(e,(d,#,#)(h,#,#))),(n,#,(s,(p,#,#),#)))

10．给定一组权值：15,22,14,6,7,5,7，构造一棵哈夫曼树，并计算树的外部带权路径长度。

【参考答案】初始时，如图5-15所示。

5

7

14

15

22

6

7

图5-15 构造哈夫曼树的初始步骤

合并权值最小的两棵树，如图5-16所示。

图5-16 合并权值最小的两棵树

7

14

15

22

7

5

6

持续合并权值最小的两棵树，直到最终得到哈夫曼树，如图5-17所示。注意所构造的哈夫曼树不是唯一的，但树的WPL是唯一的。

22

5

6

7

7

14

15

图5-17 哈夫曼树

WPL=14×2+15×2+22×2+5×4+6×4+7×4+7×4=202。

11．假设字符a,b,c,d,e,f在某文本中出现的频率分别为7,9,12,22,23,27，求这些字符的哈夫曼编码。

【参考答案】构造的哈夫曼树如图5-18所示。哈夫曼编码如表5-1所示。

f

c

a

b

d

e

图5-18 哈夫曼树

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 表5-1 哈夫曼编码 | | | | | | |
| 字符 | a | b | c | d | e | f |
| 编码 | 1110 | 1111 | 110 | 00 | 01 | 10 |

注意，哈夫曼树和哈夫曼编码都不是唯一的。

12．画出算术表达式((a+b)+c\*(d+e)+f)\*(g+h)的表达式树。

【参考答案】表达式树如图5-19所示。

f

c

e

d

b

a

h

g

图5-19 习题11的表达式树

13．已知一棵二叉树T，请回答下列问题。

（1）如果知道T的先序遍历序列和层序遍历序列，是否可以唯一还原T？为什么？

（2）如果知道T的中序遍历序列和层序遍历序列，是否可以唯一还原T？为什么？

【参考答案】

（1）如果知道T的先序遍历序列和层序遍历序列，不能唯一还原二叉树T。

图5-20a和图5-20b是两棵不同的二叉树，但它们的先序遍历序列相同，层序遍历序列也相同。

a)二叉树1

b)二叉树1

图5-20两棵二叉树

（2）如果知道T的中序遍历序列和层序遍历序列，可以唯一还原一棵二叉树。

层序遍历的第一个结点是根，在中序遍历序列中，根的前面是左子树中的全部结点，后面是右子树中的全部结点。根据这些结点的信息，将层序遍历序列中除根以外的全部结点分为两组，一组是由左子树中的全部结点组成的，这个即是左子树的层序遍历序列，另一组是由右子树中的全部结点组成的，这个即是右子树的层序遍历序列。原问题划分为规模更小的两个问题，使用递归方法即可求解。

14．将图5-27所示的两棵树合并为一棵树，令W为R的第三个子结点，画出保存该树的存储结构。

【参考答案】

保存两棵树的存储结构是：

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 下标 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| data | R | A | B | C | D | E | F | W | X | Y | Z |
| parent | -7 | 0 | 0 | 1 | 1 | 2 | 4 | -4 | 7 | 7 | 7 |

合并后的存储结构是：

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 下标 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| data | R | A | B | C | D | E | F | W | X | Y | Z |
| parent | -11 | 0 | 0 | 1 | 1 | 2 | 4 | 0 | 7 | 7 | 7 |

四、算法阅读题

二叉链表中结点类的定义及二叉树的定义如下所示。

typedef int ELEMType;

typedef struct BNode //二叉树结点

{ ELEMType data; //数据域

struct BNode \*left,\*right; //指向左子结点、右子结点的指针

}BinTNode;

typedef BinTNode \*BTree; //二叉树

以下程序实现了二叉树的若干基本操作，请在空白处填上适当内容将算法补充完整。

1．返回二叉树的高度。

int high(BTree root)

{

int i,j;

if( （1） ) return 0;

i= （2） ;

j= （3） ;

if(i>=j) return i+1;

else return j+1;

}

【参考答案】

（1）root==NULL

（2）high(root->left)

（3）high(root->right)

2．返回二叉树中度为1的结点个数。

int oneDNodeNumber(BTree root)

{

if(root==NULL) （1） ;

if(( （2） )||(root->left!=NULL && root->right==NULL))

return （3） ;

else return oneDNodeNumber(root->left)+oneDNodeNumber(root->right);

}

【参考答案】

（1）return 0

（2）root->left==NULL && root->right!=NULL

（3）oneDNodeNumber(root->left)+oneDNodeNumber(root->right)+1

3．二叉链表中结点类的定义及二叉树的定义如下所示。

typedef int ELEMType;

typedef struct BNode //二叉树结点

{ ELEMType data; //数据域

struct BNode \*left,\*right; //指向左子结点、右子结点的指针

}BinTNode;

typedef BinTNode \*BTree; //二叉树

说明change函数的功能。

void change(BTree root){

BinTNode \*temp;

if(root==NULL) return;

change(root->left);

change(root->right);

temp=root->left;

root->left=root->right;

root->right=temp;

return;

}

【参考答案】change函数的功能是将二叉树中所有结点的左、右子树相互交换。

4．函数CountLeave的功能是计算二叉树中叶结点的数量。

typedef struct BiTNode

{

ElemType data;

struct BiTNode \*lchild,\*rchild;

}\*BiTree; //二叉树结点定义

int CountLeave(BiTree T) //返回值为二叉树T中叶结点的个数

{

if(!T) return （1） ;

if ( T->lchild==NULL && T->rchild==NULL )

return （2） ;

else

return （3） ;

}

为使函数能够完成指定的功能，请在程序中的横线处填入适当的内容。

【参考答案】

（1）0

（2）1

（3）CountLeave( T->lchild ) + CountLeave( T->rchild )

五、算法设计题

1．设二叉树以二叉链表的形式存储，试设计算法，返回二叉树中度为2结点的个数。

【参考答案】程序实现如下所示。

typedef int ELEMType;

typedef struct BNode //二叉树结点

{ ELEMType data; //数据域

struct BNode \*left,\*right; //指向左子结点、右子结点的指针

}BinTNode;

typedef BinTNode \*BTree; //二叉树

int twoDNodeNumber(BTree root)

{

if(root==NULL) return 0;

if ((root->left!=NULL && root->right!=NULL))

return twoDNodeNumber(root->left)+twoDNodeNumber(root->right)+1;

else return twoDNodeNumber(root->left)+twoDNodeNumber(root->right);

}

2．设二叉树以二叉链表的形式存储，每个结点的数据域中存放一个整数。试设计算法，求二叉树中所有结点中所保存的整数之和。

【参考答案】程序实现如下所示。

int sumofTree(BTree root)

{

if(root==NULL) return 0;

return sumofTree(root->left)+sumofTree(root->right)+root->data;

}

3．将二叉树中每层的结点个数定义为该层的宽度，各层最宽的宽度定义为二叉树的宽度。设二叉树以二叉链表的形式存储，试设计算法，返回二叉树的宽度。

【参考答案】程序实现如下所示。

int WidthofTree(BTree root)

{

BinTNode temp,interval;

SeqBTreeQueue myq; //使用第三章实现的队列

int treewidth=-1,count=0;

printf("\nwidth\n");

initQueue(&myq); //初始化队列myq

interval.data=NA;

if(root==NULL) return 0;

enqueue(&myq,root); //根入队列

enqueue(&myq,&interval);

//treewidth=1;

printf("width1\n");

while(!isEmptyQ(&myq)){//队列不空，意味着还有结点待遍历

dequeue(&myq,&temp);

if(temp.data!=NA){

printf ("%d \t",temp.data);

count++;

if(temp.left!=NULL) enqueue(&myq,temp.left);

if(temp.right!=NULL) enqueue(&myq,temp.right);

}

else if(!isEmptyQ(&myq)){

if(count>treewidth)

treewidth=count;

count=0;

enqueue(&myq,&interval);

}

}

return treewidth;

}

# 第六章 图结构

一、单项选择题

1．设无向图的顶点个数为*n*，则该图所含的边数最多是\_\_\_\_\_\_\_\_。

A．*n*-1 B．*n*(*n*-1)/2 C．*n*(*n*+1)/2 D．*n*2

答案：B。

完全图所含的边数最多。对于无向图，每个顶点都和其他顶点之间有边相连，不含有顶点到自己的边。每对顶点之间仅能有一条边。

2．含*n*个顶点的连通无向图中，边数至少是\_\_\_\_\_\_\_\_。

A．*n*-1 B．*n* C．*n*+1 D．*n*log*n*；

答案：A。

含*n*个顶点的无向图中，如果边数少于*n*-1，则必定是不连通的。边数等于*n*-1时，可以构成一个连通图。边数多于*n*-1时，不仅能构成连通图，还一定会出现回路。所以*n*-1条边是最少的。

但并不是至少含有*n*-1条边的图一定是连通图。比如，一个三角形加一个孤立点构成的图中，顶点个数为*n*=4，边数为3=*n*-1，但图是不连通的。含有至少*n*-1条边的图，要看边与顶点的关联情况，才能知道是不是构成连通图。当边数足够多时，比如，含*n*个顶点的图中至少有(*n*-1)×(*n*-2)/2+1条边时，图一定是连通图。(*n*-1)×(*n*-2)/2条边使得*n*-1个顶点的子图成为完全图，现在得到两个连通分量，还剩余一条边，在含n-1个顶点的连通分量中不能再增加边了，所以这条边只能是连接n-1个顶点中的任一个与唯一的孤立顶点，使得孤立顶点与这个完全子图连通，图成为连通图。

3．下列叙述中错误的是\_\_\_\_\_\_\_\_。

A．图的深度优先搜索遍历是一个递归过程

B．图的深度优先搜索遍历不适用于有向图

C．深度优先遍历和广度优先遍历是图遍历的两种基本算法

D．图的遍历是从给定的顶点出发每一个顶点仅被访问一次

答案：B。

图的深度优先搜索和广度优先搜索是图遍历的两种基本策略（选项C正确）。深度优先搜索通常采用递归方式实现（选项A正确），既适用于对无向图的遍历，也适用于对有向图的遍历（选项B错误）。选项D是图遍历的定义。

4．无向图G=(V,E)，其中，V={a,b,c,d,e,f}，E={(a,b),(a,e),(a,c),(b,e),(c,f),(f,d),(e,d)}，对该图进行深度优先遍历，得到的顶点序列是\_\_\_\_\_\_\_\_。

A．a,b,e,c,d,f B．a,c,f,e,b,d C．a,e,b,c,f,d D．a,e,d,f,c,b

答案：D。

要完成这个题，可以先画出题目中所给的图，如图6-1所示。

图6-1 习题4的图

根据深度优先搜索策略，遍历了a,b,e三个顶点后，接下来应该寻找顶点e的未遍历邻接点（d）进行遍历，而不会遍历到顶点c（选项A是错误的）。

在遍历了a,c,f三个顶点后，接下来应该寻找顶点f的未遍历邻接点（d）进行遍历，而不会遍历到顶点e（选项B是错误的）。

在遍历了a,e,b三个顶点后，顶点b的所有邻接点均已遍历，应该回退到顶点e，选择e的下一个未遍历的邻接点（d）进行遍历，而不会遍历到顶点c（选项C是错误的）。

对于选项D，从顶点a开始遍历，依次访问了顶点a,e,d,f,c后，c的邻接点都被遍历过了，开始回退，到达顶点e后，它还有未遍历的邻接点b，访问b，遍历完成。

5．已知有向图G=(V,E)，其中V={v1,v2,v3,v4,v5,v6,v7}，E={(v1,v2),(v1,v3),(v1,v4), (v2,v5),(v3,v5),(v3,v6),(v4,v6),(v5,v7),(v6,v7)}，G的拓扑序列是\_\_\_\_\_\_\_\_。

A．v1,v3,v4,v6,v2,v5,v7 B．v1,v3,v2,v6,v4,v5,v7

C．v1,v3,v4,v5,v2,v6,v7 D．v1,v2,v5,v3,v4,v6,v7

答案：A。

要完成这个题，可以先画出题目中所给的图，如图6-2所示。

图6-2 习题5的图

对于选项B，v6出现在v4之前，但图中存在边(v4,v6)，这是矛盾的，故选项B错误。对于选项C，v5出现在v2之前，但图中存在边(v2,v5)，这是矛盾的，故选项C错误。对于选项D，v5出现在v3之前，但图中存在边(v3,v5)，这是矛盾的，故选项D错误。

6．在有向图G的拓扑排序中，若顶点v*i*在顶点v*j*之前，则下列情形不可能出现的是\_\_\_\_\_\_\_\_。

A．G中有弧(vi,vj) B．G中有一条从v*i*到v*j*的路径

C．G中没有弧(vi,vj) D．G中有一条从v*j*到v*i*的路径

答案：D。

在拓扑排序中，若有一条从v*j*到v*i*的路径，则表明v*j*是v*i*的先决条件，必须先完成v*j*后才能开始v*i*，那么在拓扑排序中，顶点v*i*不可能出现在顶点v*j*之前。

二、解答题

1．举例说明，带权图中的权值可以表示什么含义。

【参考答案】权值可以表示图中很多的信息。比如，在一个反映城市间公路网的图中，可以用顶点表示城市，边表示道路，用边上的权值表示两城市间道路的里程数、走行时间、所需费用等等，权还可以用来表示该条公路的车流量、公路等级、道路的限速等其他信息。对于一个电子线路图，边上的权值可以表示两个端点之间的电阻、电流或电压值。对于反映工程进度的图而言，边上的权值可以表示从前一个工程到后一个工程所需要的时间等等。

2．给出图6-4a、图6-4b的邻接矩阵和邻接表。

图6-4 一个无向图和一个有向图

a)无向图

b)有向图

【参考答案】

邻接矩阵的定义如下：

图6-4a的邻接矩阵A如下所示。

邻接表表结点的结构如下所示。

顶点下标

指针

图6-4a的邻接表如图6-5a所示。

图6-4b的邻接矩阵A如下所示。

图6-4b的邻接表如图6-5b所示。

1

2

0

0

1

3

2

4

0

3

1

2

3

4

5

0

1

2

3

4

4

3

a)图6-4a的邻接表

1

0

1

2

4

1

2

∧

3

4

5

∧

0

1

2

3

4

4

b)图6-4b的邻接表

图6-5 图6-4的邻接表

3．求图6-4a各顶点的度及图6-4b各顶点的入度和出席。

【参考答案】图6-4a各顶点的度如表6-1所示。

表6-1 图6-4a各顶点的度

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 顶点 | 1 | 2 | 3 | 4 | 5 |
| 度 | 3 | 2 | 2 | 3 | 2 |

图6-4b各顶点的入度和出度如表6-2所示。

表6-2 图6-4b各顶点的入度和出度

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 顶点 | 1 | 2 | 3 | 4 | 5 |
| 入度 | 1 | 2 | 1 | 0 | 2 |
| 出度 | 2 | 0 | 1 | 3 | 0 |

4．给出带权图6-6的邻接矩阵和邻接表。

6

2

7

4

14

10

5

5

6

7

3

2

6

图6-6 带权图

【参考答案】

带权图邻接矩阵的定义如下所示。

图6-6的邻接矩阵A如下所示。

带权图邻接表的表结点结构如下所示。

顶点下标

权值

指针

图6-6的邻接表如图6-7所示。

图6-7 图6-6的邻接表

v0

0

1

6

3

6

2

4

v1

1

0

6

5

7

4

7

v2

2

0

4

6

6

3

5

v3

3

v4

4

v5

5

v7

7

v6

6

0

6

6

5

2

5

4

14

1

7

7

3

3

14

5

10

1

7

7

2

4

10

2

6

7

2

3

5

4

3

6

2

5

2

5．给出图6-4a的几个不同的子图。

【参考答案】选择有不同特点的子图，列出4个，如图6-8所示。

图6-8 图6-4a的几个不同的子图

第一个是包含原图中全部顶点，但没有任何边的子图。第二个是包含了部分顶点和部分边的子图。第三个是包含了全部顶点及部分边的子图。第四个是原图，图也是自身的子图。

6．对图6-9，从顶点1开始进行深度优先搜索，试给出5种不同的遍历序列。

图6-9 无向图

【参考答案】从顶点1开始的深度优先搜索遍历序列是12354、12453、14235、14532、13245、13542。给出其中的5种即可。

7．对图6-9，从顶点1开始进行广度优先搜索，试给出5种不同的遍历序列。

【参考答案】从顶点1开始的广度优先搜索遍历序列是12345、12435、13245、13425、14235、14325。给出其中的5种即可。

8．对图6-9，给出它的几棵不同的深度优先生成树。

【参考答案】得到深度优先遍历序列12354、12453、14235、14532、13245、13542所对应的深度优先生成树分别如图6-10、图6-11、图6-12、图6-13、图6-14、图6-15所示。

图6-11 对应于12453的生成树

图6-10 对应于12354的生成树

图6-15 对应于13542的生成树

图6-14 对应于13245的生成树

图6-13 对应于14532的生成树

图6-12 对应于14235的生成树

9．对图6-9，给出它的几棵不同的广度优先生成树。

【参考答案】得到广度优先遍历序列12345、13245、13425所对应的广度优先生成树如图6-16所示。得到广度优先遍历序列12435、14235、14325所对应的广度优先生成树如图6-17所示。

图6-17 广度优先生成树

图6-16 广度优先生成树

10．对图6-18，使用普里姆算法求最小生成树。

【参考答案】可以任意选择起始顶点，不失一般性，选择顶点1为起始顶点。求解过程如表6-3所示。

3

5

2

7

9

8

8

图6-18 带权无向图

表6-3 使用普里姆算法求最小生成树

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| V  closedge | 2 | 3 | 4 | 5 | U | V-U | 选中的边 |
| vex  lowcost | 1  7 | 1  5 | 1  8 |  | {1} | {2,3,4,5} | (1,3) |
| vex  lowcost | 3  3 |  | 1  8 | 3  9 | {1,3} | {2,4,5} | (2,3) |
| vex  lowcost |  |  | 1  8 | 3  9 | {1,2,3} | {4,5} | (1,4) |
| vex  lowcost |  |  |  | 4  2 | {1,2,3,4} | {5} | (4,5) |
| vex  lowcost |  |  |  |  | {1,2,3,4,5} | Φ |  |

得到的最小生成树如图6-19所示。树的权值为18。

3

5

2

8

图6-20 最小生成树

3

5

2

8

图6-19 最小生成树

在第二步选中了边(2,3)而将顶点2加入最小生成树T后，到顶点4的有最小权值的边有两条，分别是(1,4)和(2,4)，边上的权都是8。所以，第三步选择权值为8的边时，既可以选中(1,4)也可以选中(2,4)。如果选中的是(2,4)，则得到的最小生成树如图6-20所示。

11．对图6-18，使用克鲁斯卡尔算法求最小生成树。

【参考答案】先将边按权值进行升序排序，如表6-4所示。

表6-4 按权值排序的各条边

|  |  |  |
| --- | --- | --- |
| 是否选中 | 边 | 权值 |
| √ | (4,5) | 2 |
| √ | (2,3) | 3 |
| √ | (1,3) | 5 |
| × | (1,2) | 7 |
| √ | (1,4) | 8 |
| × | (2,4) | 8 |
| × | (3,5) | 9 |

得到的最小生成树如图6-21所示。

3

5

2

8

图6-22 最小生成树

3

5

2

8

图6-21 最小生成树

对边进行排序时，如果边(2,4)排在边(1,4)的前面，则选中的边是(2,4)而不是(1,4)，得到的最小生成树如图6-22所示。

12．试写出图6-23的6种不同的拓扑排序。

①

②

④

③

⑤

⑥

⑦

⑧

⑨

⑩

图6-23 一个有向无环图

【参考答案】拓扑排序如下所示。

1,2,3,4,5,6,7,8,9,10

1,2,4,3,5,6,7,8,9,10

1,4,3,2,5,6,7,8,9,10

1,2,3,4,6,5,7,8,9,10

1,2,4,3,6,5,7,8,9,10

1,4,3,2,5,6,7,8,9,10

实际上，还可以写出更多的拓扑排序，比如：

1,2,3,4,5,6,8,7,9,10

1,2,4,3,5,6,9,8,7,10

1,4,3,2,5,6,8,9,7,10

1,4,3,2,6,5,9,8,7,10

13．求图6-24中顶点1到其余各顶点的最短路径长度。

图6-24 一个带权图

30

25

10

15

3

8

6

3

10

7

10

20

16

【参考答案】图6-24的邻接矩阵如下。

使用Dijkstra算法求解从顶点1到其余各顶点最短路径的过程如图6-25所示。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 终点 | 从顶点1到各顶点的dist值和最短路径 | | | | | | |
| 2 | 30  (1,2) | 25  (1,5,2) | 25  (1,5,2) | 25  (1,5,2) |  |  |  |
| 3 | ∞ | ∞ | 20  (1,5,6,3) |  |  |  |  |
| 4 | ∞ | ∞ | ∞ | 45  (1,5,6,3,4) | 45  (1,5,6,3,4) | 31  (1,5,6,7,4) |  |
| 5 | 10  (1,5) |  |  |  |  |  |  |
| 6 | ∞ | 17  (1,5,6) |  |  |  |  |  |
| 7 | ∞ | ∞ | 25  (1,5,6,7) | 25  (1,5,6,7) | 25  (1,5,6,7) |  |  |
| 8 | ∞ | ∞ | ∞ | ∞ | ∞ | 35  (1,5,6,7) | 35  (1,5,6,7) |
| vj | 5 | 6 | 3 | 2 | 7 | 4 | 8 |
| S | 1,5 | 1,5,6 | 1,5,6,3 | 1,5,6,3,2 | 1,5,6,3,2,7 | 1,5,6,3,2,7,4 | 1,5,6,3,2,7,4,8 |

图6-25 求最短路径过程

14．求图6-25所示的AOE网各事件的最早开始时间和最迟开始时间。

30

25

10

15

3

8

6

3

10

7

10

20

16

8

14

12

18

3

3

图6-25 AOE网

【参考答案】图6-25所示AOE网各事件的最早开始时间和最迟开始时间如表6-5所示。

表6-5 各事件的最早开始时间和最迟开始时间

|  |  |  |
| --- | --- | --- |
| 事件 | 最早开始时间 | 最迟开始时间 |
| 1 | 0 | 0 |
| 2 | 12 | 12 |
| 3 | 8 | 8 |
| 4 | 21 | 21 |
| 5 | 18 | 18 |
| 6 | 27 | 27 |

三、算法设计题

1．对于给定的无向图G，采用邻接矩阵存储，试设计算法int getD(MGraph g,VType u)，返回顶点u的度。

【参考答案】程序实现如下所示。

int getD(MGraph g,VType u){ //返回顶点u的度

int degree=0,i,j;

i=VerToNum(g,u);

for(j=0;j<g.numVertices;j++) degree+=g.AdjMatrix[i][j];

return degree;

}

无向图中，邻接矩阵的一行中非零元的个数即为该对应顶点的度。也可以统计邻接矩阵的一列中非零元的个数。

2．对于给定的无向图G，采用邻接矩阵存储，试设计算法isPath(MGraph g,VType \*path)，判定给定的顶点序列path是否是图中的路径。

【参考答案】程序实现如下所示。

int isPath(MGraph g,VType \*path){

int i,k,len,start,end;

len=sizeof(path);

if(len==0) return FALSE;

for(i=0;i<len-1;i++){

start=VerToNum(g,path[i]);

end=VerToNum(g,path[i+1]);

if(g.AdjMatrix[start][end]==0) return FALSE;

}

return TRUE;

}

# 第7章 内部排序

一、单项选择题

1．下列关于稳定排序方法的叙述中，正确的是\_\_\_\_\_\_\_\_。

I．该排序算法可以处理相同的关键字

II．该排序算法不可以处理相同的关键字

III．该排序算法处理相同的关键字时能确定它们的排序结果

IV．该排序算法处理相同的关键字时不能确定它们的排序结果

A．仅I、III B．仅I、IV C．仅II、III D．仅II、IV

答案：A。

具有相等关键字的两个记录，经过稳定的排序后，它们的相对次序不改变。初始序列中是什么样的相对次序，排序后的相对次序也是什么样的。这是稳定排序具有的特性。

根据排序方法稳定性的定义，可知I和III是正确的。

2．下列排序方法中，排序趟数与序列的初始状态有关的是\_\_\_\_\_\_\_\_。

A．插入排序 B．选择排序 C．起泡排序 D．快速排序

答案：D。

对*n*个数据项，插入排序、选择排序和起泡排序都需要进行*n*-1趟排序。对于快速排序，序列的初始状态不同，可能会影响划分过程的结果，即经过划分后，两个子段中所含数据的个数会不同。比如，如果每次都大致将数据一分为二，划分的两个子段中所含的数据个数相当，则排序的趟数约为log*n*。而如果每次划分都出现极端情况，一个子段为空，则排序趟数为*n*-1。

3．若用起泡排序对关键字序列18,16,14,12,10,8进行升序排序，所需进行的关键字比较总次数是\_\_\_\_\_\_\_\_。

A．10 B．15 C．21 D．34

答案：B。

给定的关键字序列呈降序排列，要求进行升序排序。按照起泡排序的算法，元素18要与后面的全部5个元素进行比较，然后位于最终的排序位置。在这一趟中需要进行5次比较。类似的，元素16要与后面的除18之外的4个元素进行比较，最后排在18的前面。这一趟排序需要进行4次比较。元素14排序到位的话，需要进行3次比较。元素12排序到位的话，需要进行2次比较。元素10排序到位的话，需要进行1次比较。其他元素都到位了，意味着元素8也到位了。所以总的比较次数=5+4+3+2+1=15。

实际上，题目中所给的关键字序列是降序的，对于升序序列来说，这是最坏的初始排列。所以关键字比较次数最多，需要*n*(*n*-1)/2次比较。*n*=6时，需要15次比较。

4．下列排序算法中，不能保证每趟排序至少能将一个元素放到其最终的位置上的是\_\_\_\_\_\_\_\_。

A．快速排序 B．希尔排序 C．堆排序 D．起泡排序

答案：B。

采用快速排序，每趟排序都至少有一个枢轴放置到最终位置上。堆排序中，每趟都将堆顶元素放置到最终位置上。起泡排序中，每趟会选出本趟的极值（最大或最小）放置在最终位置上。但希尔排序不能保证。

5．一组记录的关键字值为46,79,56,38,40,84，利用partition划分算法，以第一个记录为枢轴，得到的一次划分结果为\_\_\_\_\_\_\_\_。

A．38,40,46,56,79,84 B．40,38,46,79,84,56

C．40,38,46,56,79,84 D．40,38,46,84,56,79

答案：B。

初始：46 79 56 38 40 84

选择第一个元素做枢轴并将枢轴放到最后的位置：

pivot:46 84 79 56 38 40 46

此时，left对应于84，right对应于40。接下来，从left开始向右寻找大于枢轴的元素，当前元素即大于枢轴，停在84处。然后从right开始向左寻找小于枢轴的元素，也是在当前位置，停在40处。如下所示。

84 79 56 38 40 46

↑ ↑

left right

交换两个元素，如下所示。

40 79 56 38 84 46

↑ ↑

left right

接下来，寻找第二对满足条件的数据，如下所示。

40 79 56 38 84 46

↑ ↑

left right

交换两个元素，如下所示。

40 38 56 79 84 46

↑ ↑

left right

继续寻找下一对数据，如下所示。

40 38 56 79 84 46

↑ ↑

right left

此时，right>left，需要将枢轴归位，交换56与46，得到划分结果40 38 46 79 84 56。

6．在下面的排序方法中，辅助空间为*O*(*n*)的是\_\_\_\_\_\_\_\_。

A．希尔排序 B．堆排序 C．选择排序 D．归并排序

答案：D。

希尔排序、堆排序及选择排序都可以实现“原地排序”，即只需要若干个辅助变量，在保存数据的原数组空间上完成排序操作，空间复杂度是*O*(1)。归并排序过程中，需要申请一个与原数组等大的辅助数组，空间复杂度是*O*(*n*)的。

7．下列排序算法中，当待排序数据已有序时，花费时间反而最多的是\_\_\_\_\_\_\_\_。

A．起泡排序 B．希尔排序 C．快速排序 D．堆排序

答案：C。

当待排序数据已有序时，采用最原始的枢轴选择方法，可能导致所选的枢轴是数据中最小或最大的元素，这样的枢轴将原始数据划分为一个为空、另一个含其余全部数据的两组，因此排序的趟数最多，花费的时间最多。

另外三个排序方法，当待排序数据已有序时，数据交换次数达到最少，花费的时间均能达到各排序方法的最优情况。

8．数组中有10000个元素，如果仅要求求出其中最大的4个元素，则最节省时间的算法是\_\_\_\_\_\_\_\_。

A．直接插入排序 B．希尔排序 C．快速排序 D．选择排序

答案：D。

有的排序方法需要将所有数据全部排序完毕，才能知道最大的元素是哪个。有些排序方法，边排序边得到部分最终结果。

题意要求选出10000个元素中最大的4个，显然，使用后一类的排序方法更能节省时间，因为除最大的4个数据外，其余的数据不需要完全排序。

选项中给出的4个排序方法中，直接插入排序和希尔排序都需要将全部数据排序后才能知道最大的4个元素是什么。对于快速排序，使用枢轴将全部数据划分为整体有序的两部分。如果枢轴是所有数据中第*k*大的元素，且*k<*9996，则需要在较大的部分中继续进行划分。划分的趟数是不确定的。

对于选择排序，每趟排序都能选出本趟中的最大值，4趟排序后即可选出最大的4个元素。它属于不完全排序方法（即边排序边得到部分最终结果）。

4趟选择排序中，需要进行的比较操作是*n*-1+*n*-2+*n*-3+*n*-4=4*n*-10，交换次数不会超过比较次数。

9．从未排序序列中依次取出一个元素与已排序序列中的元素依次进行比较，然后将其放在已排序序列的合适位置，该排序方法称为\_\_\_\_\_\_\_\_。

A．直接插入排序 B．选择排序 C．希尔排序 D．二路归并排序

答案：A。

题目中描述的正是直接插入排序的含义。

10．使用数据序列10,15,20,25,30建立初始最大堆时，数据对交换的次数是\_\_\_\_\_\_\_\_。

A．1 B．2 C．3 D．4

答案：C。

建立初始最大堆的过程如图7-1所示。

a) 初始完全二叉树

b) 30与15交换

c) 30与10交换

d) 10与25交换

图7-1 建初始最大堆的过程

二、填空题

1．对含*n*个数据的序列进行排序，直接插入排序在最好情况下的时间复杂度为\_\_\_\_\_\_\_\_\_\_。

答案：*O*(*n*)。

初始数据已有序时，直接插入排序能达到最优时间复杂度，从第二个元素到最后一个元素，每个元素仅需和其直接前驱进行比较，不需要交换，即能完成排序。排序中关键字比较次数为*n*-1，交换次数为0。

2．现有含9个关键字的最小堆，排在关键字升序第3位的元素（第3小）所处的位置个数可能是\_\_\_\_\_\_\_\_\_\_。

答案：6。

含9个关键字的最小堆如图7-2所示。堆中，a0是最小值，a1和a2都可能是第3小的值。如果a1<a2，则a3和a4也可能是第3小的值。如果a1>a2，则a5和a6也可能是第3小的值。所以，第3小的值可能位于从a1到a6的任一位置中，共6个。

图7-2 含9个关键字的最小堆

3．一组记录的关键字值为45,79,59,63,65,26,80,17，利用partition划分算法，以第一个记录为枢轴，划分过程中，与元素79相交换的关键字是\_\_\_\_\_\_\_\_\_\_。

答案：26。

根据partition算法，划分过程如下所示。

初始：45 79 59 63 65 26 80 17

选择第一个元素做枢轴并将枢轴放到最后的位置：

pivot:45 17 79 59 63 65 26 80 45

此时，left对应于17，right对应于80。接下来，从left开始向右寻找大于枢轴的元素，当前元素即大于枢轴，停在79处。然后从right开始向左寻找小于枢轴的元素，停在26处。如下所示。

17 79 59 63 65 26 80 45

↑ ↑

left right

交换两个元素（79和26）。

4．已知一组关键字为：45,78,57,30,40,89，利用堆排序对其进行升序排序，建立的初始堆是\_\_\_\_\_\_\_\_\_\_。

答案：89,78,57,30,40,45。

5．设关键字序列为16,15,32,11,6,30,22,46,7，采用基数排序进行升序排序。若关键字序列保存在含9个元素的数组A中，则一趟基数排序后，元素16的下标位置为\_\_\_\_\_\_\_\_\_\_。

答案：5。

一趟分配与收集后，得到的结果是30,11,32,22,15,16,6,46,7，元素16在下标5处。

三、解答题

1．设一维整数数组内保存整数序列，回答下列问题：

（1）试写出整数序列2,3,8,6,1中的所有逆序对。

（2）什么情况下，由1,2,…,*n*组成的序列中逆序对最多？

【参考答案】（1）逆序对：(2,1),(3,1),(8,1),(6,1),(8,6)。

（2）数据序列为*n*,*n*-1,…,3,2,1时，逆序对最多。每个元素都和其后面的所有元素呈逆序，所以总的逆序对个数=*n*-1+*n*-2+…+2+1=*n*×(*n*-1)/2。

2．有关键字序列：38,19,65,13,97,49,41,95,1,73，采用起泡排序方法进行升序排序，请写出每趟排序的结果。

【参考答案】每趟起泡排序的结果如下所示。

初始： 38 19 65 13 97 49 41 95 1 73

第1趟： 19 38 13 65 49 41 95 1 73 97

第2趟： 19 13 38 49 41 65 1 73 95 97

第3趟： 13 19 38 41 49 1 65 73 95 97

第4趟： 13 19 38 41 1 49 65 73 95 97

第5趟： 13 19 38 1 41 49 65 73 95 97

第6趟： 13 19 1 38 41 49 65 73 95 97

第7趟： 13 1 19 38 41 49 65 73 95 97

第8趟： 1 13 19 38 41 49 65 73 95 97

第9趟： 1 13 19 38 41 49 65 73 95 97

3．对数据序列：66,61,200,30,80,150,4,8,100,12,20,31,1,5,44，写出采用希尔排序算法排序的每一趟结果。设增量序列d={5,3,1}。

【参考答案】每趟希尔排序的结果如下所示。

初始： 66 61 200 30 80 150 4 8 100 12 20 31 1 5 44

第1趟：d=5 20 4 1 5 12 66 31 8 30 44 150 61 200 100 80

第2趟：d=3 5 4 1 20 8 30 31 12 61 44 100 66 200 150 80

第3趟：d=1 1 4 5 8 12 20 30 31 44 61 66 80 100 150 200

4．有关键字序列：38,19,65,13,97,49,41,95,1,73，采用直接插入排序方法进行升序排序，请写出每趟排序的结果。

【参考答案】每趟直接插入排序的结果如下所示。

初始： 38 19 65 13 97 49 41 95 1 73

第1趟：19 38 65 13 97 49 41 95 1 73

第2趟：19 38 65 13 97 49 41 95 1 73

第3趟：13 19 38 65 97 49 41 95 1 73

第4趟：13 19 38 65 97 49 41 95 1 73

第5趟：13 19 38 49 65 97 41 95 1 73

第6趟：13 19 38 41 49 65 97 95 1 73

第7趟：13 19 38 41 49 65 95 97 1 73

第8趟：1 13 19 38 41 49 65 95 97 73

第9趟：1 13 19 38 41 49 65 73 95 97

5．有关键字序列：38,19,65,13,97,49,41,95,1,73，采用简单选择排序方法进行升序排序，请写出每趟排序的结果。

【参考答案】每趟简单选择排序的结果如下所示。

初始： 38 19 65 13 97 49 41 95 1 73

第1趟：1 19 65 13 97 49 41 95 38 73

第2趟：1 13 65 19 97 49 41 95 38 73

第3趟：1 13 19 65 97 49 41 95 38 73

第4趟：1 13 19 38 97 49 41 95 65 73

第5趟：1 13 19 38 41 49 97 95 65 73

第6趟：1 13 19 38 41 49 97 95 65 73

第7趟：1 13 19 38 41 49 65 95 97 73

第8趟：1 13 19 38 41 49 65 95 97 73

第9趟：1 13 19 38 41 49 65 95 97 73

6．对数据序列：31,5,44,55,61,200,30,60,20,1,80,150,4,29，采用改进的起泡排序方法进行降序排序，请写出每趟排序的结果。

【参考答案】每趟改进的起泡排序的结果如下所示。

初始： 31 5 44 55 61 200 30 60 20 1 80 150 4 29

第1趟：5 31 44 55 61 30 60 20 1 80 150 4 29 200

第2趟：5 31 44 55 30 60 20 1 61 80 4 29 150 200

第3趟：5 31 44 30 55 20 1 60 61 4 29 80 150 200

第4趟：5 31 30 44 20 1 55 60 4 29 61 80 150 200

第5趟：5 30 31 20 1 44 55 4 29 60 61 80 150 200

第6趟：5 30 20 1 31 44 4 29 55 60 61 80 150 200

第7趟：5 20 1 30 31 4 29 44 55 60 61 80 150 200

第8趟：5 1 20 30 4 29 31 44 55 60 61 80 150 200

第9趟：1 5 20 4 29 30 31 44 55 60 61 80 150 200

第10趟：1 5 4 20 29 30 31 44 55 60 61 80 150 200

第11趟：1 4 5 20 29 30 31 44 55 60 61 80 150 200

第12趟：1 4 5 20 29 30 31 44 55 60 61 80 150 200

7．对数据序列：31,5,44,55,61,200,30,60,20,1,80,150,4,29，写出采用快速排序算法排序的过程。

【参考答案】采用partition划分算法，快速排序算法排序的过程如下所示。

初始： 31 5 44 55 61 200 30 60 20 1 80 150 4 29

第1次：29 5 4 1 20 30 31 60 61 55 80 150 44 200

第2次：20 5 4 1 29 30 31 60 61 55 80 150 44 200

第3次：1 5 4 20 29 30 31 60 61 55 80 150 44 200

第4次：1 5 4 20 29 30 31 60 61 55 80 150 44 200

第5次：1 4 5 20 29 30 31 60 61 55 80 150 44 200

第6次：1 4 5 20 29 30 31 44 55 60 80 150 200 61

第7次：1 4 5 20 29 30 31 44 55 60 80 150 200 61

第8次：1 4 5 20 29 30 31 44 55 60 61 80 200 150

第9次：1 4 5 20 29 30 31 44 55 60 61 80 150 200

8．将数据序列：70,12,30,1,5,31,44,56,61建成一个最大堆。

【参考答案】建最大堆的过程如图7-3所示。

b)调整以1为根的子树

a)初始完全二叉树

c)调整以30为根的子树

d)调整以12为根的子树

e)调整以70为根的子树，即最大堆

图7-3 建最大堆的过程

9．对数据序列：70,12,30,1,5,31,44,56,61，采用堆排序方法进行升序排序，请写出每趟排序的结果。

【参考答案】习题8中已经将数据序列建成最大堆，在此基础上，进行堆排序，如图7-4所示。

b)输出堆顶70并整堆后

a)初始最大堆

d)输出堆顶56并整堆后

c)输出堆顶61并整堆后

f)输出堆顶31并整堆后

e)输出堆顶44并整堆后

h)输出堆顶12并整堆后

g)输出堆顶30并整堆后

i)输出堆顶5后得到排序结果

图7-4 堆排序过程

10．对数据序列：31,5,44,55,61, 30,60,20,1,4,29，采用基数排序方法进行升序排序，请写出每趟排序的结果。

【参考答案】第一趟分配的结果如图7-5所示。

60

30

20

61

31

1

4

44

55

5

29

0

1

2

3

4

5

6

7

8

9

图7-5 基数排序第1趟（按个位数进行排序）

一趟收集的结果是：30,60,20,31,61,1,44,4,5,55,29。再进行第二趟分配，结果如图7-6所示。

0

1

2

3

4

5

6

7

8

9

4

1

5

29

20

31

30

44

55

61

60

图7-6 基数排序第2趟（按十位数进行排序）

得到排序结果：1,4,5,20,29,30,31,44,55,60,61。

四、算法阅读题

1．设一系列正整数存放在一个数组中，算法evenOddSort将所有偶数存放在数组的前半部分，将所有的奇数存放在数组的后半部分。请在空白处填上适当内容将算法补充完整。

void evenOddSort(int \*data, int n) //奇偶排序

{

int left,right,mostleft=0,mostright=n-1;

left=mostleft; //left从左向右找

right=mostright; //right从右向左找

for(;;){

while( （1） && left<mostright){

left++;

}

while(data[right]%2!=0 && （2） ){

right--;

}

if(left<right)

swap( （3） );

else

break;

}

display(data,n);

return;

}

【参考答案】

（1）data[left]%2==0

（2）right>=mostleft

（3）&data[left],&data[right]

五、算法设计题

1．将3个关键字进行排序，关键字之间的比较次数最多为多少？试实现这样一个排序方法。

【参考答案】对3个关键字进行排序，关键字之间的比较次数最多为3次。

可以使用二叉树描述元素之间的比较过程及结果，这样的二叉树称为比较树。不妨设3个元素为a,b,c，其排序过程的比较树如图7-7所示。

图7-7 3个元素排序的比较树

a<b<c

a<c<b

c<a<b

b<a<c

b<c<a

c<b<a

比较树的叶结点表示的是排序结果，分支结点表示的是要进行的比较操作，左分支表示条件成立，右分支表示条件不成立。从根结点到叶结点的路径上，经过的分支结点的个数，即是能得到该叶结点所表示的排序结果而需要进行的比较操作次数。最大为3，即3个元素最多需要3次比较操作就能得到排序结果。

程序实现如下所示。

void threeSort(int a, int b, int c){

if(a<b)

if(b<c) printf("排序结果：%d %d %d\n",a,b,c);

else if(a<c) printf("排序结果：%d %d %d\n",a,c,b);

else printf("排序结果：%d %d %d\n",c,a,b);

else if(b<c)

if(a<c) printf("排序结果：%d %d %d\n",b,a,c);

else printf("排序结果：%d %d %d\n",b,c,a);

else printf("排序结果：%d %d %d\n",c,b,a);

}

2．设计一个算法，计算含*n*个元素的数据序列中的逆序数。

【参考答案】如果某元素u大于其后面的任一个元素v，则u与v构成逆序对。设数据序列保存在数组a中，元素个数为len。使用count统计逆序个数，初始时值为0。对a中的每个元素a[i]，查看排在其后面的每个元素a[j]，如果a[i]>a[j]，则count加1。

程序实现如下所示。

int rever(int \*a,int len){

int count=0;

int i,j;

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

for(j=i+1;j<len;j++){

if(a[i]>a[j])count++;

}

return count;

}

3．设有*n*个整数的数组，其元素值仅取0、1、2三种，试设计一个时间复杂度为*O*(*n*)的算法，将这个数组中元素按升序排序。

【参考答案】划分程序partion可以将数据序列分为整体有序的两部分。借助于partion思想，先将0和1分到一个组内，将2分到另一个组内。再对含0和1的组进行划分，0在一组，1在另一组，从而完成初始数据的排序。

第一趟partion是对全部数据进行筛选，第二趟partion仅对含0和1的部分进行筛选。所以算法进行比较的次数最多为2*n*，时间复杂度为*O*(*n*)。

算法实现如下所示。

void flagSort(int \*data,int n)//三类值排序

{

int left,right,mostleft=0,mostright=n-1;

left=mostleft; //left从左向右找

right=mostright; //right从右向左找

for(;;){//完成第一趟划分

while(data[left]<=1 && left<mostright){

left++;

}

while(data[right]==2 && right>=mostleft){

right--;

}

if(left<right) //交换两个元素

swap(&data[left],&data[right]);

else

break;

}

right=left-1; //right从右向左找

left=mostleft; //left从左向右找

for(;;){//完成第二趟划分

while(data[left]<1 && left<mostright){

left++;

}

while(data[right]==1 && right>=mostleft){

right--;

}

if(left<right) //交换两个元素

swap(&data[left],&data[right]);

else

break;

}

display(data,n);//显示排序结果

return;

}

# 第8章 查找

一、单项选择题

1．对*n*个元素的查找表做顺序查找时，若查找每个元素的概率相同，则查找成功的平均查找长度为\_\_\_\_\_\_\_\_。

A．(*n*+1)/2 B．*n*/2 C．*n* D．((1+*n*)×*n*)/2

答案：A。

在查找表的位置*i*（0≤*i*≤*n*-1）处查找到目标时，进行了*i*+1次比较。所以总的查找次数为1+2+…+*n*=*n*×(*n*+1)/2，取平均值为(*n*+1)/2。

2．下列关于折半查找的叙述中，正确的是\_\_\_\_\_\_\_\_。

A．表必须有序，表可以顺序方式存储，也可以链表方式存储

B．表必须有序且表中数据必须是整型、实型或字符型

C．表必须有序，而且只能从小到大排列

D．表必须有序，且表只能以顺序方式存储

答案：D。

折半查找要求查找表中的数据必须有序，既可以是升序的，也可以是降序的。故选项C错误。能进行折半查找的存储结构只能是数组，因为需要快速定位到查找区间的中间位置，而在链表中不能快速定位到中间位置。所以选项A是错误的。进行查找时，查找表中的数据类型必须是能进行比较的，只要在类型上定义了比较操作，就可以进行查找。能进行比较操作的类型不仅仅是整型、实型或字符型。选项B是错误的。

3．与二叉查找树的查找效率有关的是二叉树的\_\_\_\_\_\_\_\_。

A．高度 B．结点个数 C．树形 D．结点的位置

答案：C。

二叉查找树的查找过程从树根开始，沿着一条路径向叶结点的方向前进，将查找目标依次与该路径上的各结点进行比较。查找效率是指对所有位置进行查找的平均查找长度，不是单指对某结点的查找。所以选项D错误。

以高度均为3的满二叉树T1和退化为单链的二叉树T2为例。如图8-1所示。

图8-1 高度相同的二叉查找树示例

a)满二叉树T1

b)退化为单链的T2

T1的结点总数是7，高度*h*==3。平均查找长度=(1×1+2×2+3×4)/7≈2.43。T2的结点总数是3，高度也是3。平均查找长度=(1×1+2×1+3×1)/3=2。可见高度相同的两棵不同二叉树的平均查找长度可能是不同的。

再看都含7个结点的两棵树。如图8-2所示。

图8-2 结点个数相同的二叉查找树示例

a)含7个结点的树T3

b)含7个结点的树T4

……

T4的平均查找长度=(1+2+…+7)/7=4，T3的平均查找长度仍约为2.43，可见结点个数相同的两棵树，树形不同，查找效率也可能不同。

4．下列二叉查找树的情形中，其查找效率最差的是\_\_\_\_\_\_\_\_。

A．结点太多 B．完全二叉树

C．没有度为2的结点 D．结点太复杂

答案：C。

参看习题3。二叉查找树中没有度为2的结点，意味着二叉查找树中所有分支结点均只有一个子结点，此时树的高度最大，查找效率最低。

5．分别以下列序列构造二叉查找树，得到的树形与其他三个不同的是\_\_\_\_\_\_\_\_。

A．100,80,90,60,120,110,130

B．100,120,110,130,80,60,90

C．100,60,80,90,120,110,130

D．100,80,60,90,120,130,110

答案：C。

选项A、B和D构造的二叉查找树如图8-3a所示。选项C构造的二叉查找树如图8-3b所示。

b)选项C对应的树形

a)选项A、B、D对应的树形

图8-3 习题5对应的树形

6．结点数为4的AVL树可能的结构个数有\_\_\_\_\_\_\_\_。

A．4 B．5 C．6 D．7

答案：A。

树中含有4个结点，根在0层（1个结点），1层必有2个结点。否则，剩余的2个结点只能放在2层，且为1层结点的两个子结点，导致根的平衡因子为2或-1，树失平衡。

画出相应的树形，如图8-4所示。

d)树形之四

c)树形之三

b)树形之二

a)树形之一

图8-4 习题6对应的树形

7．若将关键字1,2,3,4,5,6,7依次插入到初始为空的AVL树*T*中，则*T*中平衡因子为0的分支结点的个数是

A．0 B．1 C．2 D．3

答案：D。

实际上，得到的AVL树是一棵满二叉树，所有结点的平衡因子均为0，其中有三个是分支结点，四个是叶结点。

8．下列关于*m*阶B树的说法中，错误的是\_\_\_\_\_\_\_\_。

A．根结点至多有*m*棵子树

B．所有叶子都在同一层次上

C．非叶结点至少有*m*/2（*m*为偶数）或*m*/2+1（*m*为奇数）棵子树

D．根结点中的数据是有序的

答案：C。

一棵*m*阶B树或者为空，或者为满足下列性质的*m*叉树：

（1）树中每个结点至多有*m*棵子树；

（2）根结点至少有两棵子树；

（3）除根结点之外，每个结点至少有⎡*m*/2⎤棵子树；

（4）所有叶结点都出现在同一层上；

（5）所有结点都包含如下形式的数据：

（n,A0,K1,A1,K2,A2,…,Kn,An）

其中*n*为关键字的个数，Ki(*i*=1,2,…,*n*)为关键字，且满足K1<K2<…<Kn。

由（1）知选项A正确，由（4）知选项B正确，由（5）知选项D正确。而选项C中没有将根结点的特例排除。当有关键字插入到B树中时，因结点中关键字个数的限制，可能导致结点分裂，中间关键字提升至父结点中。如果根结点分裂为两个结点，则生成的新根结点中仅有一个关键字两棵子树。这是特例。

9．下列关于*m*阶B树的说法中，正确的是\_\_\_\_\_\_\_\_。

I．每个结点至少有两棵非空子树

II．树中每个结点至多有m-1个关键字

III．所有叶子在同一层上

IV．当插入一个数据项引起B树结点分裂后，树长高一层

A．仅III B．仅II、III C．仅I、II、III D．仅II、III、IV

答案：B。

B树的每个分支结点中至少含有一个关键字，故至少含有两棵非空子树。但叶结点的所有子树都是空子树，其非空子树的个数为0。I．是错误的。B树的定义中规定，每个结点最多有*m*棵子树，而结点中子树的个数比关键字的个数多1，所以每个结点中关键字的个数最多有*m*-1个。II．是正确的。B树的所有叶子都在同一层上。III．是正确的。当在B树中插入一个数据项并引起结点分裂时，使得中间关键字提升至父结点中。这又可能导致父结点中关键字个数超限结点分裂。如果这个过程一直进行到根结点，即根结点分裂才会使树高长一层。如果没能让根结点分裂，则树高维持不变。IV．错误。

10．下列关于哈希（Hash，散列）查找的说法中，正确的是\_\_\_\_\_\_\_\_。

A．哈希函数构造的越复杂越好，因为这样随机性好，冲突小

B．除留余数法是所有哈希函数中最好的

C．不存在特别好与坏的哈希函数，要视情况而定

D．若需在哈希表中删去一个元素，不管用何种方法解决冲突都只要简单的将该元素删去即可

答案：C。

哈希方法中，哈希函数的使用频度非常高，所以通常不能构造非常复杂的哈希函数。另外，也不是越复杂，随机性越好。选项A是错误的。评判哈希函数好坏的重要指标是算法简单，随机性好，冲突小。不同的关键字集合会采用不同的哈希函数构造方法。没有一种方法适用于所有的情况且是最好的。选项B非常片面，错误。从另一方面来看，哈希函数要依具体的应用情况而定，每类哈希函数都有适用的具体情况。选项C的说法非常客观。至于从哈希表中删除关键字的操作，在开放地址法中，因为涉及到发生冲突后关键字的查找要循探测再散列的探测序列重新访问一遍，所以如果直接删除某关键字会使得这个探测序列断掉，之后的关键字不会被查找到。D的选项错误。

11．设哈希表长为14，哈希函数是H(key)=key mod 11，表中已有数据的关键字为15,38,61,84共4个，现要将关键字为49的结点加到表中，用二次探测再散列法解决冲突，则放入的位置是\_\_\_\_\_\_\_\_。

A．8 B．3 C．5 D．9

答案：D。

题目中所给的5个关键字对应的哈希值如表8-1所示。

表8-1 关键字对应的哈希值（地址）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 15 | 38 | 61 | 84 | 49 |
| H | 4 | 5 | 6 | 7 | 5 |

将关键字15,38,61,84依次放入位置4,5,6,7，均是一次性放置到位，没有发生冲突。如表8-2所示。

表8-2 放置了4个关键字的哈希表

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|  |  |  |  | 15 | 38 | 61 | 84 |  |  |  |  |  |  |

接下来放置49，其哈希值为5，49与38在位置5发生冲突。采用二次探测再散列方法解决冲突。探测的地址依次是5+1=6，5-1=4，5+4=9，找到开放地址，所以49放入的位置是9。

12．哈希函数有一个共同的性质，即函数值应当以\_\_\_\_\_\_\_\_取其值域的每个值。

A．最大概率 B．最小概率 C．随机概率 D．相等概率

答案：D。

哈希的含义即是均匀分布，即函数值映射到值域中每个值的概率都是一样的。既不能是最大概率，也不能是最小概率，更不能是随机概率。

13．哈希表的地址区间为0-17，哈希函数为H(K)=K mod 17。采用线性探测法处理冲突，并将关键字序列26,25,72,38,8,18,59依次存储到哈希表中。

（1）元素59存放在哈希表中的位置是\_\_\_\_\_\_\_\_。

A．8 B．9 C．10 D．11

答案：D。

题目中所给的7个关键字对应的哈希值如表8-3所示

表8-3 关键字对应的哈希值（地址）

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 26 | 25 | 72 | 38 | 8 | 18 | 59 |
| H | 9 | 8 | 4 | 4 | 8 | 1 | 8 |

将关键字26,25,72依次放入各自的位置，没有发生冲突。接下来放置38，它与72在位置4发生冲突。采用线性探测再散列解决冲突，38保存在位置5。关键字8与25在位置8处发生冲突，线性探测后又与26在位置9处发生冲突，最后放置在位置10处。18放置在位置1处。59在位置8、9、10分别与25、26、8发生冲突，最后放置在位置11处。得到的哈希表如表8-4所示。

表8-4 放置了7个关键字的哈希表

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
|  | 18 |  |  | 72 | 38 |  |  | 25 | 26 | 8 | 59 |  |  |  |  |  |  |

（2）存放元素59需要搜索的次数是\_\_\_\_\_\_\_\_。

A．2 B．3 C．4 D．5

答案：C。

59在位置8、9、10分别与25、26、8发生冲突，在位置11处找到空闲位置，搜索的次数是4次。

二、解答题

1．在顺序存储的条件下，当各记录满足什么条件时，可以分别采用什么策略进行查找？

【参考答案】当记录无序时，可采用顺序查找方法。当记录有序时，可采用折半查找方法。当记录整体有序时，可采用索引顺序查找方法。

2．在4阶B树中，每个结点所含子树个数的上下限分别是多少？

【参考答案】在4阶B树中，每个结点所含子树最少是2棵，最多是4棵。

3．在5阶B树中，每个结点所含关键字个数最多是多少？最小是多少？

【参考答案】在5阶B树中，每个结点中所含关键字个数最多是4个，最少是2个。

4．由6个结点构造的二叉查找树，其最大高度和最小高度分别是多少？

【参考答案】含6个结点的二叉查找树，最大高度是6，最小高度是3。例如，图8-5所示的是一棵由6个结点组成的二叉查找树，其中没有度为2的结点，所以高度能达到最大。图8-6所示的是一棵高度最小的二叉查找树。类似于满二叉树或完全二叉树，将结点尽量排满每层，以达到树高度最小的目的。

图8-5 含6个结点的BST树

图8-6 含6个结点的BST树

5．二叉查找树具有什么特点？

【参考答案】二叉查找树是一棵“有序”树。树中各结点保存的关键字需要满足相应的有序性。具体来说，对每个结点来说，其左子树中所有结点中的关键字小于该结点中的关键字，右子树中所有结点中的关键字大于该结点中的关键字。所以，当对二叉查找树进行中序遍历时，能得到一个升序序列。

6．二叉查找树中最大的关键字和最小的关键字分别位于什么位置？

【参考答案】从根结点沿右孩子指针向下查找，直到右孩子指针为空的结点中保存的是树中最大的关键字。类似地，从根结点沿左孩子指针向下查找，直到左孩子指针为空的结点中保存的是树中最小的关键字。

7．设有一个包含15个关键字的有序表，其中关键字的次序为：1,2,3,4,5,6,7,8,9,10,11, 12,13,14,15，当用折半查找法查找关键字2,10,7时，其比较次数分别是多少？

【参考答案】可以画出对应于所给查找表的折半查找判定树，如图8-7所示。

图8-7 折半查找判定树

进行查找时，从根开始，沿一条路径向下，将查找目标与路径上的各结点依次进行比较。可以看到，查找关键字2时，依次比较的关键字分别是8,4,2，所以比较次数为3次。查找关键字10时，依次比较的关键字分别是8,12,10，所以比较次数为3次。查找关键字7时，依次比较的关键字分别是8,4,6,7，所以比较次数为4次。

具体的查找过程如下。以lo表示最左位置，hi表示最右位置，mi表示中间位置。

查找关键字2的过程如图8-8所示。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |  |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |  |
| ↑ lo |  |  |  |  |  |  | ↑ mi |  |  |  |  |  |  | ↑ hi | 第一次 |
| ↑ lo |  |  | ↑ mi |  |  | ↑ hi |  |  |  |  |  |  |  |  | 第二次 |
| ↑ lo | ↑ mi | ↑ hi |  |  |  |  |  |  |  |  |  |  |  |  | 第三次，成功 |
| 图8-8 查找关键字2的过程 | | | | | | | | | | | | | | | |

查找关键字10的过程如图8-9所示。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |  |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |  |
| ↑ lo |  |  |  |  |  |  | ↑ mi |  |  |  |  |  |  | ↑ hi | 第一次 |
|  |  |  |  |  |  |  |  | ↑ lo |  |  | ↑ mi |  |  | ↑ hi | 第二次 |
|  |  |  |  |  |  |  |  | ↑ lo | ↑ mi | ↑ hi |  |  |  |  | 第三次，成功 |
| 图8-9 查找关键字10的过程 | | | | | | | | | | | | | | | |

查找关键字7的过程如图8-10所示。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |  |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |  |
| ↑ lo |  |  |  |  |  |  | ↑ mi |  |  |  |  |  |  | ↑ hi | 第一次 |
| ↑ lo |  |  | ↑ mi |  |  | ↑ hi |  |  |  |  |  |  |  |  | 第二次 |
|  |  |  |  | ↑ lo | ↑ mi | ↑ hi |  |  |  |  |  |  |  |  | 第三次 |
|  |  |  |  |  |  | ↑ |  |  |  |  |  |  |  |  | 第四次，成功 |
|  |  |  |  |  | lo mi hi | | |  |  |  |  |  |  |  |  |
| 图8-10 查找关键字7的过程 | | | | | | | | | | | | | | | |

8．设由空树开始，依次插入关键字D,E,G,B,C,J,A,I，构成二叉查找树。画出这棵树的生成过程。

【参考答案】二叉查找树的生成过程如图8-11所示。

b)插入E后的二叉查找树

a)插入D后的二叉查找树

(c)插入G后的二叉查找树

(d)插入B后的二叉查找树

(e)插入C后的二叉查找树

(f)插入J后的二叉查找树

(f)插入A后的二叉查找树

(g)插入I后的二叉查找树

图8-11 二叉查找树的生成过程

9．设有依以下次序出现的关键字22,41,53,46,30,13,1,67，构造一棵二叉查找树，画出最后得到的树形。

【参考答案】得到的二叉查找树如图8-12所示。

图8-12 生成的二叉查找树

10．设有依以下次序出现的关键字22,41,53,46,50,13，构造一棵AVL树，画出最后得到的树形。

【参考答案】得到的AVL树如图8-13所示。

图8-13 生成的AVL树

11．设有依以下次序出现的关键字35,16,18,20,5,50,22,60,3,17,45,7，构造一棵3阶B树。要求从空树开始，每插入一个关键字，画出一个树形。

【参考答案】3阶B树的生成过程如图8-14所示。

插入20后

插入18后

插入35后

插入16后

插入22后

插入60后

插入5后

插入50后

插入3后

插入17后

插入45后

插入7后

图8-14 3阶B树的生成过程

12．构造哈希函数的基本原则是什么？列出三种常用的构造方法。

【参考答案】构造哈希函数的基本原则有两条：

（1）算法简单，运算量小；

（2）均匀分布，减少冲突。

算法简单及冲突少，都是为了提高哈希方法的效率。

常用的构造方法有很多，包括直接定址法、平方取中法、除留余数法等。

13．设有一组关键字，出现次序为：105,97,28,52,37,22,16,90,45,79,59,76，要求用哈希方法将它们存入长度为15的表中。

（1）采用除留余数法构造哈希函数；

（2）用二次探测再散列法解决冲突。

给出最后得到的哈希表。

【参考答案】（1）哈希表长是15，除留余数法中，*p*应为小于等于表长15的最大素数。本题取*p*=13，哈希函数为H(*k*)=*k* mod *p*。先计算各关键字对应的哈希值，如表8-5所示。

表8-5 关键字对应的哈希值（地址）

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 105 | 97 | 28 | 52 | 37 | 22 | 16 | 90 | 45 | 79 | 59 | 76 |
| H | 1 | 6 | 2 | 0 | 11 | 9 | 3 | 12 | 6 | 1 | 7 | 11 |

（2）将各关键字放入哈希表中，采用二次探测再散列法解决冲突。放置前8个关键字时不会引起冲突，直接放置到位。如表8-6所示。

表8-6 放置了前8个关键字的哈希表

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| 52 | 105 | 28 | 16 |  |  | 97 |  |  | 22 |  | 37 | 90 |  |  |

关键字45与关键字97在位置6发生冲突，放置在6+1=7的位置。关键字79与关键字105在位置1发生冲突，探查的位置序列依次是：1+1=2（仍冲突），1-1=0（仍冲突），1+4=5（不冲突），所以将关键字79放置在位置5。关键字59要放置的位置7，放置了45，故探查7+1=8位置，这是开放的。最后一个关键字76，探查的位置依次是11，11+1=12，11-1=10，所以将76放置在位置10。最终得到的哈希表如表8-7所示。

表8-7 最终的哈希表

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| 52 | 105 | 28 | 16 |  | 79 | 97 | 45 | 59 | 22 | 76 | 37 | 90 |  |  |

14．已知哈希表的地址空间为A[0..11]，哈希函数H(k)=k mod 11，采用线性探测法处理冲突。请将下列数据25,16,38,47,79,82,51,39,89,151,231依次插入到哈希表中，并计算出在等概率情况下查找成功时的平均查找长度。

【参考答案】根据哈希函数H(k)=k mod 11，计算各关键字的哈希值，如表8-8所示。

表8-8 关键字对应的哈希值（地址）

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 25 | 16 | 38 | 47 | 79 | 82 | 51 | 39 | 89 | 151 | 231 |
| H | 3 | 5 | 5 | 3 | 2 | 5 | 7 | 6 | 1 | 8 | 0 |

采用线性探测法解决冲突，得到的哈希表及各关键字的探查次数如表8-9所示。

表8-9 最终的哈希表及各关键字的探查次数

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 231 | 89 | 79 | 25 | 47 | 16 | 38 | 82 | 51 | 39 | 151 |  |
| 1 | 1 | 1 | 1 | 2 | 1 | 2 | 3 | 2 | 4 | 3 |  |

查找成功时的平均查找长度ASL成功=(1×5+2×3+3×2+4)/11=21/11≈1.91。

15．设哈希表的地址范围为0~17，哈希函数为：H(K)=K mod 17，K为关键字，用线性探测再散列法处理冲突，输入关键字序列：10,24,32,17,31,30,46,47,40,63,49，构造哈希表，试回答下列问题：

（1）给出最终的哈希表；

（2）若查找关键字63，需要依次与哪些关键字比较?

（3）若查找关键字65，需要依次与哪些关键字比较?

（4）假定每个关键字的查找概率相等，求查找成功时的平均查找长度。

【参考答案】（1）先计算各关键字的哈希值，如表8-10所示。

表8-10 关键字对应的哈希值（地址）

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 10 | 24 | 32 | 17 | 31 | 30 | 46 | 47 | 40 | 63 | 49 |
| H | 10 | 7 | 15 | 0 | 14 | 13 | 12 | 13 | 6 | 12 | 15 |

得到的哈希表如表8-11所示。

表8-11 最终的哈希表

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| 17 | 49 |  |  |  |  | 40 | 24 |  |  | 10 |  | 46 | 30 | 31 | 32 | 47 | 63 |

（2）查找关键字63，先计算其哈希值为12，探查位置依次是12,13,14,15,16,17，所以依次与46,30,31,32,47,63进行比较，最后在位置17处查找成功。

（3）查找关键字65，先计算其哈希值为14，探查位置依次是14,15,16,17,0,1,2，所以依次与31,32,47,63,17,49进行比较，最后在位置2处遇到开放地址，查找失败。

三、算法设计题

1．设二叉查找树中保存的是整数。编写算法，返回一棵二叉查找树上最大关键字与最小关键字的值。

【参考答案】二叉查找树中最大关键字位于树的“右下角”，即从根开始，沿右孩子指针向下查找，其右孩子指针为空的结点中保存的是二叉查找树中的最大值。类似地，二叉查找树中最小关键字位于树的“左下角”，即从根开始，沿左孩子指针向下查找，其左孩子指针为空的结点中保存的是二叉查找树中的最小值。

程序实现如下所示。

int MaxOfBstSearch(BstTree t) //返回二叉查找树中的最大值

{

//若树为空，则返回NA

BstNode \*temp=t;

if(t==NULL) return NA;

while(temp->right!=NULL){

temp=temp->right;

}

return temp->data;

}

int MinOfBstSearch(BstTree t) //返回二叉查找树中的最小值

{

//若树为空，则返回NA

BstNode \*temp=t;

if(t==NULL) return NA;

while(temp->left!=NULL){

temp=temp->left;

}

return temp->data;

}

2．设二叉查找树中保存的是整数。编写算法，返回一棵二叉查找树上全部关键字的平均值。

【参考答案】可以按任何一种遍历策略，访问二叉查找树中的结点，然后将结点中保存的关键字求和，进而求平均值。

程序实现如下所示。

void Average1(BstTree root,int \*sum,int \*count);

float Average(BstTree root)

{ int sum=0,nodecount=0;

if (root==NULL){

return 0;

}

Average1(root,&sum,&nodecount);

return sum\*1.0/nodecount;

}

void Average1(BstTree root,int\* sum,int\* nodecount)

{

if(root==NULL){

return;

}

\*sum+=root->data;

(\*nodecount)++;

Average1(root->left,sum,nodecount);

Average1(root->right,sum,nodecount);

return;

}

3．设二叉查找树中保存的是整数。编写算法，输出二叉查找树中满足k1≤x≤k2的所有关键字x。

【参考答案】可以使用任一种遍历策略，对二叉查找树中的每个结点，若结点中保存的关键字值介于k1和k2之间，则输出。如果使用中序遍历策略，则输出的结果呈升序排序。

//二叉树区间查找

void searchInorder(BstTree root,int k1, int k2)

{

if (root==NULL){

return;

}

searchInorder(root->left,k1,k2);

if(root->data>=k1 && root->data<=k2) printf ("%d\t",root->data);

searchInorder(root->right,k1,k2);

}

4．设二叉查找树中保存的是整数。编写算法，判别给定二叉树是否为二叉查找树。

【参考答案】如果一棵二叉树T是二叉查找树，则中序遍历T会得到升序序列。使用变量current记录遍历过程中目前已经得到的最大值，如果当前遍历的结点中的值小于current，则不是二叉查找树；否则继续遍历。初始时，给current赋最小值：int current=NA;。主程序中调用时，带两个参数：isBST(root,&current)。

程序实现如下所示。

int isBST(BstTree root,int \*current)

{

if (root==NULL) return 1;

if(!isBST(root->left,current)) return 0;

if(root->data>=(\*current)) (\*current)=root->data;

else return 0;

if(!isBST(root->right,current)) return 0;

return 1;

}