尚德机构
SUNLANDS

密训·资料

数据结构导论（全国）

1904

MI XUN ZI LIAO

编前语

试卷说明：本科目考试题型为单选、填空、简答、综合题。各题型分值分布情况如下表所示：

题型	分值
单选题	15 道×2 分=30 分
填空题	13 道×2 分=26 分
应用题	5 道×6 分=30 分
算法设计题	2 道×7 分=14 分

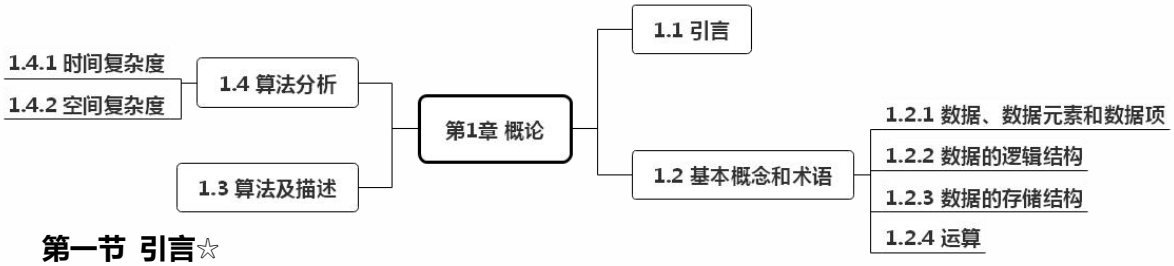
资料说明：本资料提供的都是考试重点，但是在重点中，按照知识点的重要程度，对知识点进行了标星。如下表：

星级	说明
1-2 星	易考单选、填空。2 星相对 1 星更重要。
3-4 星	易考应用题和算法设计题。4 星相对 3 星更重要。

星标数目越多，表示知识点越重要，越可能考到。但要注意的是，资料提供的是知识重点，并不是押题。文中加粗的是需要重点了解的地方。这份资料是为了指导学员考前最后的复习，帮助学员理清知识点间的逻辑，加速记忆。学员在拿到资料后，可根据星标数、资料里的逻辑整理和标识，了解哪些知识点是重点，然后重点复习。

目录

第一章 概论



第一节 引言☆

1976 年瑞士计算机科学家尼克劳斯•维尔特（Niklaus Wirth）曾提出一个著名公式： 算法+数据结构=程序。

第二节 基本概念和术语

一、数据、数据元素和数据项☆☆

数据	所有被计算机存储、处理的对象。
数据元素	数据的基本单位，是运算的基本单位。常常又简称为元素。
数据项	是数据的不可分割的最小标识单位。在数据库中数据项又称为字段或域。
从宏观上看，数据、数据元素和数据项实际上反映了数据组织的三个层次，数据可由若干个数据元素组成，而数据元素又可由若干个数据项组成。	

数据结构是相互之间存在一种或多种特定关系的数据元素的集合。它包括数据的逻辑结构、数据的存储结构和数据的基本运算。

二、数据的逻辑结构☆☆

1、数据的逻辑结构：指数据元素之间的逻辑关系。所谓逻辑关系是指数据元素之间的关联方式或“邻接关系”。

2、四类基本的逻辑结构：

集合	任意两个结点之间都没有邻接关系，组织形式松散。
线性结构	结点按逻辑关系依次排列形成一条“链”，结点之间一个一个依次相邻接。
树形结构	具有分支、层次特性，其形态像自然界中的树，上层的结点可以和下层多个结点相邻接，但下层结点只能和上层的一个结点相邻接。
图结构	最复杂，其中任何两个结点都可以相邻接。

三、数据的存储结构☆☆

1、数据的逻辑结构在计算机中的实现称为数据的存储结构（或物理结构）。

2、表示数据元素之间的关联方式：

顺序存储方式	指所有存储结点存放在一个连续的存储区里。利用结点在存储器中的相对位置来表示数据元素之间的逻辑关系。
链式存储方式	指每个存储结点除了含有一个数据元素外，还包含指针，每个指针指向一个与本结点有逻辑关系的结点，用指针表示数据元素之间的逻辑关系。
关系：一种逻辑结构可以采用一种或几种存储方式来表达数据元素之间的逻辑关系，相应的存储结构称为给定逻辑结构的存储实现或存储映像。	

四、运算☆

1、运算：指在某种逻辑结构上施加的操作，即对逻辑结构的加工。

2、线性表、栈和队列中的元素具有相同的逻辑结构（即线性结构），但有不同的运算集，它们是不同的数据结构。

第三节 算法及描述

运算的实现是指该运算的算法。算法是计算机科学的一个基本概念，也是程序设计的一个核心概念。

第四节 算法分析☆☆

评价算法好坏的因素：☆☆	
正确性	能正确地实现预定的功能，满足具体问题的需要。
易读性	易于阅读、理解和交流，便于调试、修改和扩充。
健壮性	即使输入非法数据，算法也能适当地做出反应或进行处理，不会产生预料不到的运行结果。
时空性	一个算法的时空性是指该算法的时间性能（或时间效率）和空间性能（或空间效率）， 前者是算法包含的计算量，后者是算法需要的存储量。

一、时间复杂度☆☆

1、大 O 表示法。T(n) =O(f(n))称为算法的渐进时间复杂度，简称时间复杂度。

2、时间复杂度常见的阶数：

(1) 常数阶 O(1)（即算法的时间复杂度与输入规模 n 无关）；

(2) 对数阶 O(log2n)；

(3) 线性阶 O(n)；

(4) 多项式阶 O(nC)，C 为大于 1 的正整数，常见的多项式阶有 O(n2)和 O(n3)；

(5) 指数阶 0 (Cn)，C 为大于 1 的正整数，常见的指数阶有 O(2n)。

n	$f(n)=\log_2 n$	$f(n)=n$	$f(n)=n^2$	$f(n)=n^3$	$f(n)=n^4$	$f(n)=n^{10}$	$f(n)=2^n$
10	0.0015 μs	0.005 μs	0.05 μs	0.5 μs	5 μs	5 s	0.5 μs
30	0.0025 μs	0.015 μs	0.45 μs	13.5 μs	405 μs	3.42 d	0.5 s
50	0.003 μs	0.025 μs	1.25 μs	62.5 μs	3.13 ms	1.55 y	6.5 d
10^3	0.005 μs	0.5 μs	0.5 ms	0.5 s	8.33 min	1.59×10^{13} y	16×10^{283} y
10^5	0.0085 μs	50 μs	5 s	5.79 d	1585 y	1.59×10^{33} y	

二、空间复杂度☆☆

空间复杂度是对一个算法在运行过程中临时占用存储空间大小的量度。存储空间量应包括：

（1）程序代码所占用的空间；

（2）输入数据所占用的空间；

（3）辅助变量所占用的空间，估算时，一般只需要分析辅助变量所占用的空间。

【习题演练】

1、数据的不可分割的最小标识单位是（ ）

A:数据项　　B:数据记录

C:数据元素　　D:数据变量

答案：A

解析：一般情况下，数据元素由数据项组成。在数据库中数据项又称为字段或域。它是数据的不可分割的最小标识单位。

2、具有分支、层次特性，上层的结点可以和下层多个结点相邻接，但下层结点只能和上层的一个结点相邻接，这种组织形式称为（ ）

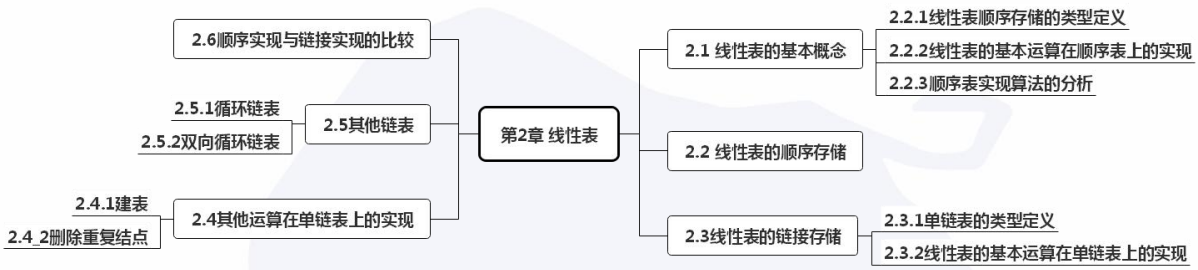
A:集合　　B:线性结构

C:树形结构　　D:图结构

答案：C

解析：树形结构具有分支层次特性，其形态像自然界的树，上层的结点可以和下层多个结点相邻接，但下层结点只能和上层的一个结点相邻接。

第二章 线性表



第一节 线性表的基本概念

概念	线性表是一种线性结构，它是由 n(n≥0)个数据元素组成的有穷序列，数据元素又称结点。结点个数 n 称为表长。
基本特征	线性表中结点具有一对一的关系，如果结点数不为零，则除起始结点没有直接前驱外，其他每个结点有且仅有一个直接前驱；除终端结点没有直接后继外，其他每个结点有且仅有一个直接后继。
基本运算及功能描述	(1) 初始化 Initiate (L)：建立一个空表 L=()，L 不含数据元素。 (2) 求表长 Length(L)：返回线性表 L 的长度。 (3) 读表元素 Get (L ,i) 返回线性表第 i 个数据元素 ,当 i 不满足 1≤i≤Length(L)时，返回一特殊值。 (4) 定位 Locate(L，x)：查找线性表中数据元素值等于 x 的结点序号，若有多个数据元素值与 x 相等，运算结果为这些结点中序号的最小值，若找不到该结点，则运算结果为 0。 (5) 插入 Insert(L，x，i)：在线性表 L 的第 i 个数据元素之前插入一个值为 x 的新数据元素，参数 i 的合法取值范围是 1≤i≤n+1。 (6) 删除 Delete (L，i)：删除线性表 L 的第 i 个数据元素 ai，i 的有效取值范围是 1≤i≤n。

第二节 线性表的顺序存储

一、线性表顺序存储的类型定义

用顺序存储实现的线性表称为顺序表。一般使用数组来表示顺序表。

二、线性表的基本运算在顺序表上的实现☆☆☆☆

顺序表的“第 i 个数据元素”存放在数组下标为“i-1”的位置。

1、插入

顺序表的插入运算 InsertSeqlist (SeqList L,DataType x,int i)是指在顺序表的第 i(1≤i≤n+1)个元素之前 ,插入一个新元素 x。使长度为 n 的线性表(a1 ,a2 ,… ,ai-1， ai ,ai+1 ,…，an)变为长度为 n+1 的线性表(a1，a2，…，ai-1，x， ai，ai+1，….，an)。

插入算法的基本步骤是：首先将结点 ai~an 依次向后移动一个元素的位置，这样空出第 i 个数据元素的位置；然后将 x 置入该空位，最后表长加 1。所需移动元素的个数为 n-i+1。

具体的插入算法描述如下：

```
void InsertSeqlist(SeqList L,DataType x, int i)
 {   //将元素 x 插入到顺序表 L 的第 i 个数据元素之前
     if (L. length==Maxsize)   exit(“表已满”);
     if (i<1 || i>L. length+1)   exit(“位置错”)；//检查插入位置是否合法
     for(j=L.length;j>=i;j--)                    //初始 i=L.length
          L.data[j]=L.data[j-1];                 //依次后移
     L.data[i-1]=x;                              //元素 x 置入到下标为 i-1 的位置
     L.length++;                                 //表长度加 1
}
```

2、删除

删除运算 DeleteSeqlist(SeqList L,int i)是指将线性表的第 i(1≤i≤n)个数据元素删去，使长度为 n 的线性表（a1，a2，…，ai-1， ai，ai+1，….，an)变为长度为 n-1 的线性表(a1，a2，…，ai-1，ai+1，….，an)。

删除运算的基本步骤是：①结点 ai+1,…，an 依次向左移动一个元素位置（从而覆盖掉被删结点 ai）；②表长度减 1。此处无需考虑溢出，只判断参数 i 是否合法即可。所需移动元素的个数为 n-i。

算法描述如下：

```
void DeleteSeqList(SeqList L,int i) {
//删除线性表 L 中的第 i 个数据结点
     if(i<1 || i>L.length)                  //检查位置是否合法
          exit(“非法位置”);
     for(j=i;j<L.length;j ++)               //第 i 个元素的下标为 i-1
          L.data[j-1]=L.data[j];            //依次左移
     L.length--;                            //表长度减 1
}
```

3、定位

定位运算 LocateSeqlist(SeqList L,DataType x)的功能是查找出线性表 L 中值等于 x 的结点序号的最小值，当找不到值为 x 的结点时，返回结果 0。

三、顺序表实现算法的分析☆☆

（1）设表的长度 length=n，在插入算法中，元素的移动次数不仅与顺序表的长度 n 有关，还与插入的位置 i 有关。一般情况下元素比较和移动的次数为 n-i+1 次，插入算法的平均移动次数约为 n/2，其时间复杂度是 O(n)。

（2）删除算法：时间复杂度为 O(n)，元素平均移动次数约为（n-1）/2，时间复杂度为 O(n)。

（3）定位算法：以参数 x 与表中结点值的比较为标准操作，平均时间复杂度为 O(n)。求表长和读表元素算法的时间复杂度为 O(1)，就阶数而言，已达到最低。

第三节 线性表的链接存储

一、单链表的类型定义☆☆

1、结点结构

（1）data 部分称为数据域，用于存储线性表的一个数据元素。

（2）next 部分称为指针域或链域，用于存放一个指针，该指针指向本结点所含数据元素的直接后继结点。

2、单链表：所有结点通过指针链接形成链表。

（1）head 称为头指针变量，该变量的值是指向单链表的第一个结点的指针。可以用头指针变量来命名单链表。

（2）链表中第一个数据元素结点称为链表的首结点。

（3）链表中最后一个数据元素结点称为尾结点或终端结点。尾结点指针域的值 NULL 称为空指针，它不指向任何结点，表示链表结束。

3、单链表的类型定义如下：☆☆☆☆

```
typedef struct node
{     DataType data;              //数据域
struct node * next;          //指针域
}Node, *LinkList;
```

4、带头结点的单链表

在单链表的第一个结点之前增设一个类型相同的结点 称之为头结点 其他结点称为表结点。表结点的第一个和最后一个结点分别就是首结点和尾结点。

二、线性表的基本运算在单链表上的实现

1、求表长☆☆

在单链表存储结构中，线性表的表长等于单链表中数据元素的结点个数，即除了头结点以外的结点的个数。

设置一个工作指针 p，初始时，p 指向头结点，并设置一个计数器 cnt，初值设置为 0。然后，让工作指针 p 通过指针域逐个结点向尾结点移动，工作指针每向尾部移动一个结点，

让计数器加 1。直到工作指针 p->next 为 NULL 时，说明已经走到了表的尾部，这时已完成对所有结点的访问，计数器 cut 的值即是表的长度。

2、定位☆☆☆

线性表的定位运算，就是对给定表元素的值，找出这个元素的位置。在单链表的实现中，则是给定一个结点的值，找出这个结点是单链表的第几个结点。定位运算又称作按值查找。在定位运算中，也需要从头至尾访问链表，直至找到需要的结点，返回其序号。若未找到，返回 0。

3、插入☆☆☆☆

单链表的插入运算是将给定值为 x 的元素插入到链表 head 的第 i 个结点之前。

步骤：（1）先找到链表的第 i-1 个结点 q。（2）生成一个值为 x 的新结点 p，p 的指针域指向 q 的直接后继结点。（3）q 的指针域指向 P。

插入运算描述如下：

```
void InsertLinklist (LinkList head, DataType x, int i)
//在表 head 的第 i 个数据元素结点之前插入一个以 x 为值的新结点
{
    Node *p,*q;
    if (i==1)   q=head;
    else   q=GetLinklist (head, i-1);              //找第 i-1 个数据元素结点
    if (q==NULL)                                   //第 i-1 个结点不存在
        exit(“找不到插入的位置”);
    else
    {
        p=malloc(sizeof (Node) );p->data=x; //生成新结点
        p->next=q->next;                           //新结点链域指向*q 的后继结点
        q->next=p;                                 //修改*q 的链域
    }
}
```

链接操作 p->next=q->next 和 q->next=p 两条语句的执行顺序不能颠倒，否则结点*q 的链域值（即指向原表第 i 个结点的指针）将丢失。

4、删除☆☆☆

删除运算是给定一个值 i，将链表中第 i 个结点从链表中移出，并修改相关结点的指针域，以维持剩余结点的链接关系。如下图，将 ai 结点移出后，需要修改该结点的直接前驱结点的指针域，使其指向移出结点 ai 的直接后继结点。

第四节 其他链表

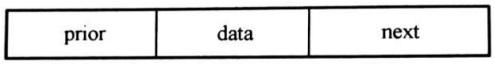
一、循环链表☆☆

在单链表中，如果让最后一个结点的指针域指向第一个结点可以构成循环链表。

二、双向循环链表☆☆

1、结点结构

在单链表的每个结点中再设置一个指向其直接前驱结点的指针域 prior，这样每个结点有两个指针，其结点结构如图：



2、双向循环链表的概念

prior 与 next 类型相同，它指向直接前驱结点。头结点的 prior 指向最后一个结点，最后一个结点的 next 指向头结点，由这种结点构成的链表称为双向循环链表。

3、删除

设 p 指向待删结点，删除*p 可通过下述语句完成：

(1) p->prior->next=p->next; //p 前驱结点的后链指向 p 的后继结点

(2) p->next->prior=p->prior; //p 后继结点的前链指向 p 的前驱结点

(3) free(p); //释放*p 的空间

（1）、（2）这两个语句的执行顺序可以颠倒。

4、插入

在 p 所指结点的后面插入一个新结点*t，需要修改四个指针:

(1) t->prior=p;

(2) t->next=p->next;

(3) p->next->prior=t;

(4) p->next=t;

第五节 顺序实现与链接实现的比较☆

（1）对于按位置查找运算，顺序表是随机存取，时间复杂度为 O(1)。单链表需要对表元 素进行扫描，它时间为复杂度为 O(n)。

（2）对于定位运算，基本操作是比较，顺序表和单链表上的实现算法的时间复杂度都是相同的，均为 O(n)。

（3）对于插入、删除运算。在顺序表中，平均时间复杂度为 O(n)。在单链表中，其平均时间复杂度仍然为 O(n)。

【习题演练】

1、设顺序表的表长为 10，则执行插入算法的元素平均移动次数约为（ ）

A:4 B:5

C:6 D:7

答案：B

解析：设顺序表的长度为 n，插入算法的平均移动次数约为 n/2。

2、在单链表中，存储每个结点需要有两个域，一个是数据域，另一个是指针域，指针域指向该结点的（ ）

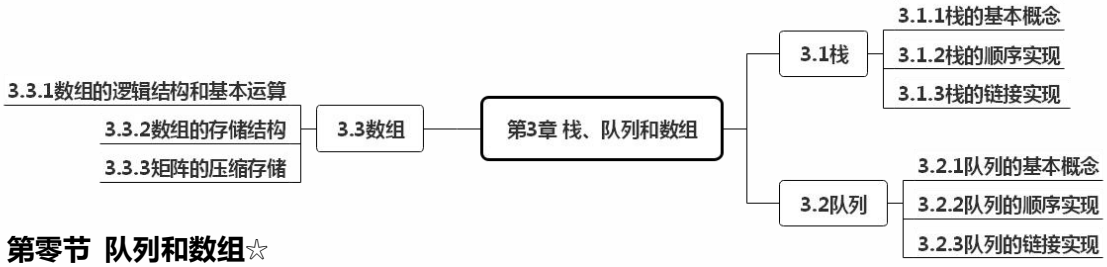
A:直接前趋 B:直接后继

C:开始结点　　D:终端结点

答案：B

解析：一个数据元素和一个指针组成单链表的一个结点。data 部分称为数据域，用于存储线性表的一个数据元素，next 部分称为指针域或链域,用于存放一个指针，该指针指向本结点所含数据元素的直接后继结点

第三章 队列和数组



第零节 队列和数组☆

栈和队列可看作是特殊的线性表。它们的特殊性表现在它们的基本运算是线性表运算的子集，它们是运算受限的线性表。

第一节 栈

一、栈的基本概念☆☆

1、栈：这种线性表上的插入和删除运算限定在表的某一端进行。允许进行插入和删除的一端称为栈顶，另一端称为栈底。不含任何数据元素的栈称为空栈。处于栈顶位置的数据元素称为栈顶元素。

2、栈的修改原则是后进先出，栈又称为后进先出线性表，简称后进先出表。栈的插入和删除运算分别称为进栈和出栈。

3、栈的基本运算：☆☆☆☆

(1) 初始化 InitStack(S)：构造一个空栈 S；

(2) 判栈空 EmptyStack(S)：若栈 S 为空栈，则结果为 1，否则结果为 0；

(3) 进栈 Push(S, x)：将元素 x 插入栈 S 中，使 x 成为栈 S 的栈顶元素；

(4) 出栈 Pop(S)：删除栈顶元素；

(5) 取栈顶 GetTop(S)：返回栈顶元素。

二、栈的顺序实现☆☆

①当空栈，栈顶下标值 top=0，如果此时做出栈运算，则产生“下溢”。

②当栈中的数据元素已经填满了，如果再进行进栈操作，会发生“上溢”。

基本运算	描述

初始化	int InitStack(SeqStk *stk)
	{
	　　stk->top=0;
	　　return 1;
	}

判栈空	int EmptyStack(SeqStk *stk)

	//若栈为空，则返回值 1，否则返回值 0
	{
	if (stk->top==0)
	return 1;
	else return 0;
	}

进栈	int Push(SeqStk *stk, DataType x)
	//若栈未满，元素 x 进栈 stk 中，否则提示出错信息
	{
	if (stk->top==maxsize-1) //判断找是否满
	{ error（“栈已满”）;
	return0;
	}
	else
	{ stk->top++; //栈未满，top 值加 1
	stk->data[stk->top]=x; //元素 x 进栈
	return 1;
	}
	}

出栈	int Pop {SeqStk *stk)
	{ if (EmptyStack(stk)) //判断是否下溢（栈空）
	{ error(“下溢”);
	return 0;
	}
	else //未下溢，栈顶元素出栈
	{ stk->top--; //top 值减 1
	return 1;
	}
	}

取栈顶元素	DataType GetTop(SeqStk *stk)
	//取栈顶数据元素,栈顶数据元素通过参数返回
	{ if (EmptyStack(stk)) return NULLData; //栈空，返回 NULLData
	else
	return stk->data[stk->top]; //返回栈顶数据元素
	}

三、栈的链接实现☆☆

基本运算	描述

进栈	void Push (LkStk *LS, DataType x)
	{ LkStk *temp;
	temp=(LkStk *)malloc(sizeof(LkStk)); //temp 指向申请的新结点
	temp->data=x; //新结点的 data 域赋值为 x
	temp->next=LS->next; //temp 的 next 域指向原来的栈顶结点
	LS->next=temp; //指向新的找顶结点
	}

出栈	int Pop(LkStk *LS)
	//栈顶数据元素通过参数返回，它的直接后继成为新的栈顶
	{ LkStk *temp;
	if (!EmptyStack(LS)) //判断栈是否为空
	{ temp=LS->next; //temp 指向栈顶结点
	LS->next=temp->next; //原栈顶的下一个结点成为新的栈顶
	free(temp); //释放原栈顶结点空间
	return 1;
	}
	else return 0 ;
	}

第二节 队列

一、队列的基本概念☆☆☆

概念	是有限个同类型数据元素的线性序列，是一种先进先出的线性表。
基本运算	(1) 队列初始化 InitQueue(Q)：设置一个空队列 Q； (2) 判队列空 EmptyQueue(Q)：若队列 Q 为空，则返回值为 1，否则返回值为 0； (3) 入队列 EnQUeUe(Q，x)：将数据元素 x 从队尾一端插入队列，使其成为队列的新尾元素； (4) 出队列 OutQueue(Q)：删除队列首元素； (5) 取队列首元素 GetHead(Q)：返回队列首元素的值。

二、队列的顺序实现☆☆

顺序队列	顺序存储实现的队列称为顺序队列，顺序队列结构类型中有三个域：data、front 和 rear。 入队列赋值语句：SQ.rear=SQ.rear+1; SQ.data[SQ.rear]=x 完成。 出队列赋值语句：SQ.front==SQ.front+1 完成。会出现“假溢出”现象。
循环队列	为了避免元素的移动，将存储队列元素的一维数组首尾相接，形成一个环状。 入队列操作语句：SQ.rear=(SQ.rear+1) % maxsize; SQ.data[SQ.rear]=x; 出队列赋值语句：SQ.fronts(SQ.front+1)% maxsize;

	循环队列满：（（CQ.rear+1）% maxsize==CQ.front）成立。 队列空条件：（CQ.rear==CQ.front）成立。

三、 队列的链接实现☆

队列的链接实现实际上是使用一个带有头结点的单链表来表示队列，称为链队列。头指针指向链表的头结点，单链表的头结点的 next 域指向队列首结点，尾指针指向队列尾结点，即单链表的最后一个结点。

由于链接实现需要动态申请空间，故链队列在一定范围内不会出现队列满的情况，当（LQ.front==LQ.rear）成立时，队列中无数据元素，此时队列为空。

第三节 数组

一、数组的逻辑结构和基本运算

一维数组又称向量，它由一组具有相同类型的数据元素组成，并存储在一组连续的存储单元中。若一维数组中的数据元素又是一维数组结构，则称为二维数组。

二、数组的存储结构☆☆

1、一维数组元素的内存单元地址是连续的，二维数组可有两种存储方法：一种是以列序为主序的存储；另一种是以行序为主序的存储（常用）。

2、数组元素的存储位置是下标的线性函数：

对于二维数组 a[m][n],如果每个元素占 k 个存储单元，以行为主序为例，讨论数组元素 a[i][j]位置与下标的关系。

由于下标从 0 开始，元素 a[i] [j]之前已经有 i 行元素，每行有 n 个元素，在第 i 行，有 j+1 个元素，总共有 n*i+j+1 个元素，第一个元素与 a[i] [j]相差 n*i+j+1—1 个位置， 故 a[i][j]的位置为：loc[i, j]=loc[0,0]+(n*i+j)*k。

三、矩阵的压缩存储☆☆☆☆

如果值相同的元素或者零元素在矩阵中的分布有一定规律，称此类矩阵为特殊矩阵。

矩阵的非零元素个数很少的矩阵称为稀疏矩阵。

1、特殊矩阵

（1）对称矩阵。若一个 n 阶方阵 A 中的元素满足下述条件：$a_{ij}=a_{ji}$　0≤i，j≤n-1，则 A 称为对称矩阵。假设以一维数组 M[n(n+1)/2]作为 n 阶对称矩阵 A 的存储结构。

设矩阵元素 a_{ij} 在数组 M 中的位置为 k，(i，j) 和 k 存在如下对应关系：

$$k=\begin{cases}\dfrac{(i+1)i}{2}+j & \text{当} i \geqslant j\\ \dfrac{(j+1)j}{2}+i & \text{当} i<j\end{cases}$$

（2）三角矩阵

以主对角线为界的上(下)半部分是一个固定的值 c 或零，这样的矩阵叫做下(上)三角矩阵。

上三角矩阵中，第 i 行除常数外有 n−i 个元素，第 0 行有 n 个元素，而 a_{ij} 之前已经有 i 行。前 i 行的元素个数总共有

$$\sum_{i=0}^{n-1}(n-i)=\frac{i}{2}(2n-i+1)$$

在第 i 行上，a_{ij} 是该行的第 j－i＋1 个元素，M[k]和 a_{ij} 的对应关系是：

$$k=\begin{cases}\dfrac{i(2n-i+1)}{2}+j-i & 当i\leqslant j\\ \dfrac{n(n+1)}{2} & 当i>j\end{cases}$$

下三角矩阵的存储和对称矩阵类似。M[k]和 a_{ij} 的对应关系是：

$$k=\begin{cases}\dfrac{i(i+1)}{2}+j & 当i\geqslant j\\ \dfrac{n(n+1)}{2} & 当i<j\end{cases}$$

2、稀疏矩阵

假设 m 行 n 列的矩阵有 t 个非零元素，当 t < < m*n 时，则称矩阵为稀疏矩阵。

稀疏矩阵压缩存储的三元组表示法：

用三个项来表示稀疏矩阵中的非零元素 a_{ij}，即（i， j，a_{ij}），其中 i 表示行序号，j 表示列序号，a_{ij} 是非零元素的值，通常称为三元组。将稀疏矩阵中的所有非零元素用这种三元组的形式表示，并按照一定的次序组织在一起，就形成了三元组表示法。

i	j	v

【习题演练】

1、在栈中进行插入和删除操作的一端称为（ ）

A:栈顶　　B:栈底

C:任意位置　　D:指定位置

答案：A

解析：允许进行插入和删除的一端称为栈顶，另一端称为栈底。

2、栈的运算特点是先进后出，元素 a、b、c、d 依次入栈，则不能得到的出栈序列是（ ）

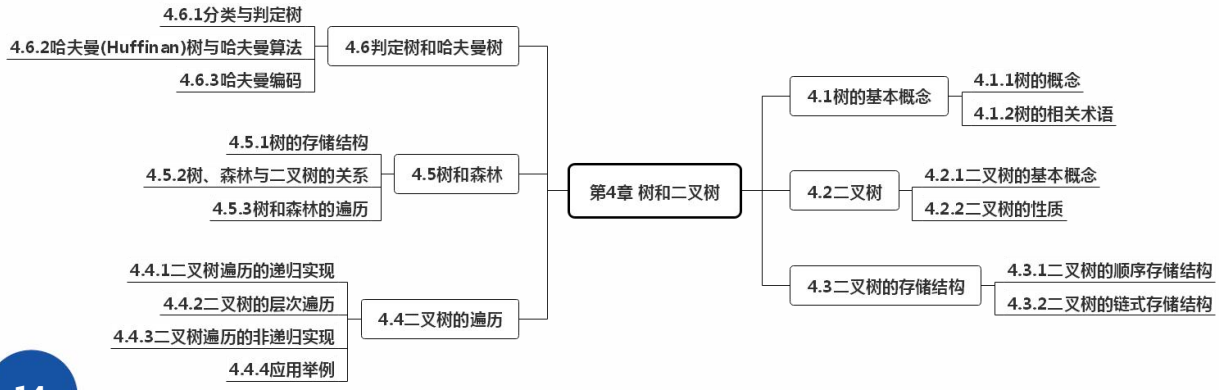
A:abcd　　B:dcba

C:cabd　　D:bcda

答案：C

解析：选项 C 中，若要 c 先出栈，则 a、b 都要先入栈，所以 c 出栈后 b 在 a 的上面，不可能 a 先出栈。

第四章 树和二叉树



第一节 树的基本概念

一、树的概念☆

1、树是 n(n 多 0)个结点的有限集合，一棵树满足以下两个条件：

（1）当 n=0 时，称为空树；

（2）当 n>0 时，有且仅有一个称为根的结点，除根结点外，真余结点分 m(m≥0)个互不相交的非空集合 T_1，T_2，…，T_m，这些集合中的每一个都是一棵树，称为根的子树。

2、森林（Forest）是 m(m>0)棵互不相交的树的集合。树的每个结点的子树是森林。删除一个非空树的根结点，它的子树便构成森林。

二、树的相关术语☆☆

(1) 结点的度：树上任一结点所拥有的子树的数目称为该结点的度。

(2) 叶子：度为 0 的结点称为叶子或终端结点。

(3) 树的度：一棵树中所有结点的度的最大值称为该树的度。

(4) 一个结点的子树的根称为该结点的孩子（或称子结点）。相应地该结点称为孩子的双亲（也称父结点）。

(5) 结点的层次：从根开始算起，根的层次为 1,其余结点的层次为其双亲的层次加 1。

(6) 树的高度：一棵树中所有结点层次数的最大值称为该树的高度或深度。

(7) 有序树：若树中各结点的子树从左到右是有次序的，不能互换，称为有序树。有序树中最左边子树的根称为第 1 个孩子，左边第 i 个子树的根称为第 i 个孩子。

(8) 无序树：若树中各结点的子树是无次序的，可以互换，则称为无序树。

第二节 二叉树

一、二叉树的基本概念

1、二叉树（Binary Tree）是 n(n≥0)个元素的有限集合，该集合或者为空，或者由一个根及两棵互不相交的左子树和右子树组成，其中左子树和右子树也均为二叉树。

注意：左右子树不可换位置，都可为空。

2、二叉树的基本运算包括：

(1) 初始化 Initiate(BT)：建立一棵空二叉树，BT=0。

(2) 求双亲 Parent(BT, X)：求出二叉树 BT 上结点 X 的双亲结点，若 X 是 BT 的根或 X 根本不是 BT 上的结点，运算结果为 NULL。

(3) 求左孩子 Lchild(BT ,X)和求右孩子 Rchild(BT ,X)：分别求出二叉树 BT 上结点 X 的左、右孩子；若 X 为 BT 的叶子或 X 不在 BT 上，运算结果为 NULL。

(4) 建二叉树 Create (BT)：建立一棵二叉树 BT。

(5) 先序遍历 PreOrder(BT)：按先序对二叉树 BT 进行遍历，每个结点被访问一次且仅被访问一次，若 BT 为空，则运算为空操作。

(6) 中序遍历 InOrder (BT)：按中序对二叉树 BT 进行遍历，每个结点被访问一次且仅被访问一次，若 BT 为空，则运算为空操作。

(7) 后序遍历 PostOrder(BT)：按后序对二叉树 BT 进行遍历，每个结点被访问一次且仅被访问一次，若 BT 为空，则运算为空操作。

(8) 层次遍历 LevelOrder(BT)：按层从上往下，同一层中结点按从左往右的顺序，对二叉树进行遍历，每个结点被访问一次且仅被访问一次，若 BT 为空，则运算为空操作。

二、二叉树的性质☆☆☆☆

满二叉树：深度为 k（k≥1）且有 2k-1 个结点的二叉树称为满二叉树。由性质 2 知，满二叉树上的结点数已达到了二叉树可以容纳的最大值。

完全二叉树：如果对满二叉树按从上到下，从左到右的顺序编号，并在最下一层删去部分结点（删后最后一层仍有结点），如果删除的这些结点的编号是连续的且删除的结点中含有最大编号的结点，那么这棵二叉树就是完全二叉树。

满二叉树一定是完全二叉树， 完全二叉树不一定是满二叉树。

性质 1	二叉树第 i（i≥1）层上至多有个结点。
性质 2	深度为 k（k≥1）的二叉树至多有 2k-1 个结点。
性质 3	对任何一棵二叉树，若度数为 0 的结点（叶结点）个数为 n0，度数为 2 的结点个数为 n2，则 n0=n2+1。
性质 4	含有 n 个结点的**完全二叉树**的深度为⌊log2n⌋+1。
性质 5	如果将一棵有 n 个结点的**完全二叉树**按层编号，按层编号是指：将一棵二叉树中的所有 n 个结点按从第一层到最大层，每层从左到右的顺序依次标记为 1, 2,…， n。则对任一编号为 i（1≤i≤n）的结点 A 有： (1) 若 i=1，则结点 A 是根；若 i>1,则 A 的双亲 Parent(A)的编号为⌊i/2⌋； (2) 若 2*i>n，则结点 A 既无左孩子，也无右孩子；否则 A 的左孩子 Lchild(X)的编号为 2*i; (3) 若 2*i+l>n ,则结点 A 无右孩子 ;否则 ,A 的右孩子 Rchild(X)的编号为 2*i + 1。

第三节 二叉树的存储结构

一、二叉树的顺序存储结构☆☆☆

二叉树的顺序存储结构可以用一维数组来实现。

如果需要顺序存储的非完全二叉树，首先必须用某种方法将其转化为完全二叉树，为此可增设若干个虚拟结点。

这样可以用与处理完全二叉树相同的方法实现二叉树的基本运算。但这种方法的缺点是造成了空间的浪费。

二、二叉树的链式存储结构☆☆☆☆

1、二叉树有不同的链式存储结构，其中最常用的是二叉链表与三叉链表。其结点形式如下：

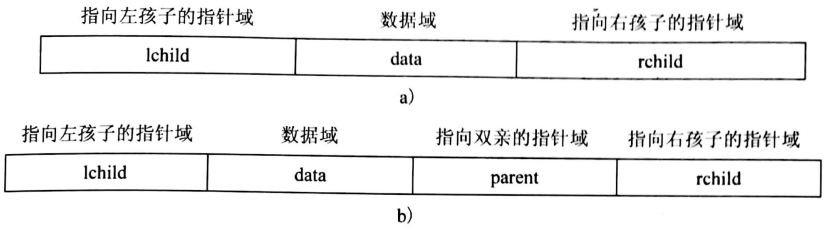


图 4-9 二叉链表和三叉链表结点结构

a) 二叉链表的结点 b) 三叉链表的结点

2、类型定义

（1）二叉链表

```
typedef struct btnode {
    DataType data;
    struct btnode *lchild, *rchild;      //指向左右孩子的指针
}*BinTree;
```

具有 n 个结点的二叉树中，有 2n 个指针域，其中只有 n-1 个用来指向结点的左、右孩子，其余的 n+1 个指针域为 NULL。

（2）三叉链表

```
typedef struct ttnode {
    datatype data;
    struct ttnode *lchild,*parent,*rchild;
}*TBinTree;
TBinTree root;
```

第四节 二叉树的遍历☆☆☆☆

一、二叉树遍历的递归实现

分类	操作	算法

先序遍历	(1) 访问根结点；
	(2) 先序遍历左子树；
	(3) 先序遍历右子树。
		void preorder(BinTree bt){
		if (bt!=NULL){
		visit(bt);
		preorder (bt ->lchild);
		preorder (bt ->rchild);
		}
		}

中序遍历	(1) 中序遍历左子树；
	(2) 访问根结点；
	(3) 中序遍历右子树。
		void inorder(BinTree bt){
		if (bt!=NULL){
		inorder (bt ->lchild);
		visit(bt);
		inorder (bt ->rchild);
		}
		}

后序遍历	(1) 后序遍历左子树；
	(2) 后序遍历右子树；
	(3) 访问根结点。
		void postorder(BinTree bt){
		if(bt!=NULL){
		postorder (bt->lchild);
		postorder (bt->rchild);
		visit(bt);}
		}

二、应用举例☆☆☆☆

利用二叉树结点的先序序列和中序序列，可以确定这棵二叉树。由二叉树结点的中序序列和后序序列也可以确定一棵二叉树。

第五节 树和森林

一、树的存储结构☆☆

孩子链表表示法	主体是一个数组元素个数和树中结点个数相同的一维数组。树上的一个结点 X 以及该结点的所有孩子结点组成一个带头结点的单链表，单链表的头结点含有两个域：数据域和指针域。其中，数据域用于存储结点 X 中的数据元素，指针域用于存储指向 X 第一个孩子结点的指针。 除头结点外，其余所有表结点也含两个域：孩子域(child)和指针域(next)，孩子域存储其中一个孩子的序号，指针域指向其下一个孩子的结点。
孩子兄弟链表表示法	存储时每个结点除了数据域外，还有指向该结点的第一个孩子和下一个兄弟结点的指针。 注意：孩子兄弟链表的结构形式与二叉链表完全相同，但结点中指针的含义不同。二叉链表中结点的左、右指针分别指向左、右孩子；而孩子兄弟链表中结点的两个指针分别指向孩子和兄弟。
双亲表示法	由一个一维数组构成。数组的每个分量包含两个域：数据域和双亲域。数据域用于存储树上一个结点中数据元素，双亲域用于存储本结点的双亲结点在数组中的序号(下标值)。

二、树、森林与二叉树的关系☆☆☆☆

树转换成二叉树	(1) 将所有兄弟结点连接起来； (2) 保留第一个兄弟结点与父结点的连接，断开其他兄弟结点与父结点的连接，然后以根结点为轴心按顺时针的方向旋转 45°角。
森林转换成二叉树	(1) 将每棵树转换成相应的二叉树； (2) 将（1）中得到的各棵二叉树的根结点看作是兄弟连接起来
二叉树转换成森林	(1) 在待转换的二叉树中，断开根结点与右孩子的连线，得到两棵二叉树，其中一棵是以二叉树 B 的根结点为根的二叉树，另一棵是以根结点的右孩子 E 为根结点的二叉树。图 4-22a 中，断开 A 与 E 的连线后得到两棵如图 4-22b 所示两棵的二叉树 B1 和 B2。 (2) 在二叉树 B1 中，连接 A 与 C，A 与 D。然后将 B 和 C 的连线断开，C 和 D 的连线断开。 (3) 重复步骤（1）（2）对 B2 进行转换。

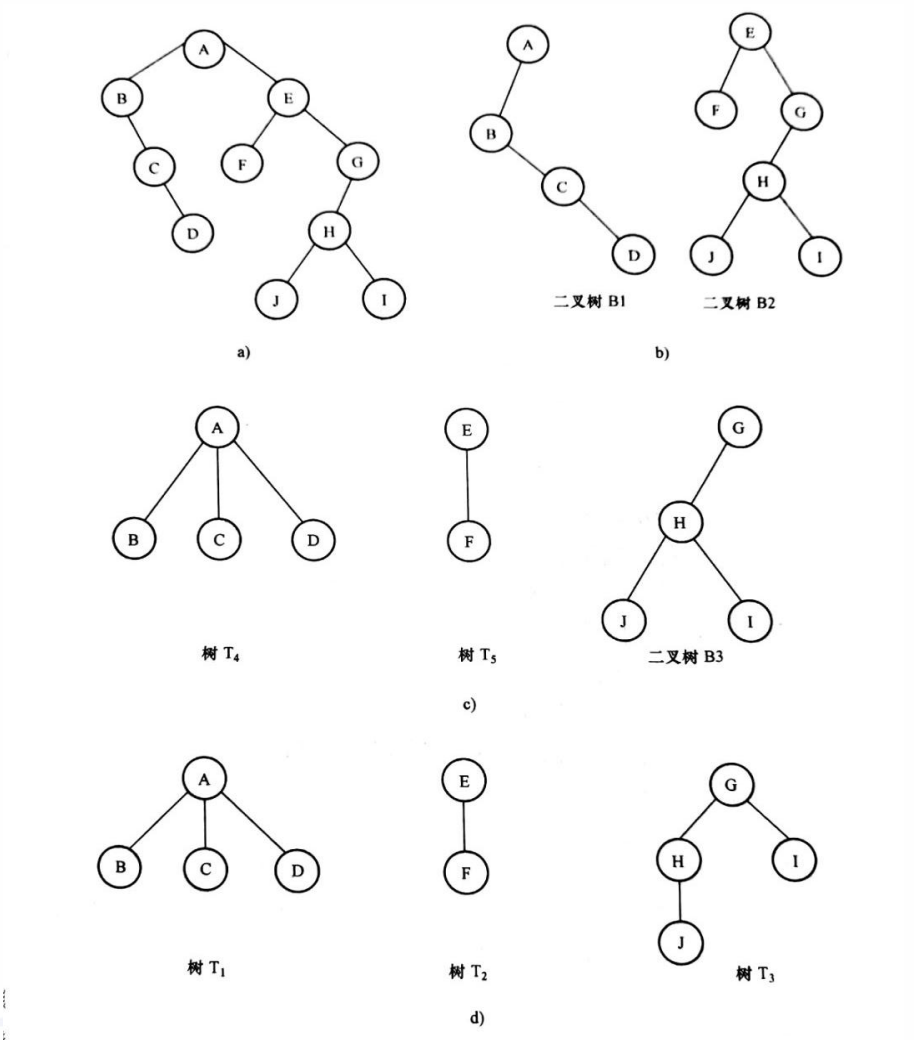


图 4-22　从二叉树到森林的转换过程示例图

a) 二叉树 B　b) 第一次转换结果　c) 第二次转换结果　d) 第三次转换结果

三、树和森林的遍历☆

1、树的遍历：（1）先序遍历（2）后序遍历（3）层次遍历。

2、森林的遍历：（1）先序遍历森林。（2）中序遍历森林。

第六节 判定树和哈夫曼树

一、分类与判定树☆

分类是一种常用运算，其作用是将输入数据按预定的标准划分成不同的种类。

用于描述分类过程的二叉树称为判定树。

二、哈夫曼(*Huffman*)树与哈夫曼算法☆☆☆☆

1、给定一组值 p_1…，p_k，如何构造一棵有 k 个叶子且分别以这些值为权的判定树，使得其平均比较次数最小。满足上述条件的判定树称为哈夫曼树。

假定从根结点到 n_i 结点之间的结点数是 l_i，平均比较次数 WPL（T）= $\sum_{i=1}^{k} p_i \times l_i$

2、哈夫曼算法非形式的描述如下：

(1) 由给定的值{p_1，…，p_k}构造森林 F= {T_1，…，T_k}，其中每个 Ti 为一棵只有根结点且其权为 p_i 的二叉树。

(2) 从 F 中选取根结点的权最小的两棵二叉树 T_i 和 T_j，构造一棵分别以 T_i 和 T_j 为左、右子树的新的二叉树 T_h，置 T_h 根结点的权为 T_i 和 T_j 根结点的权值之和。

(3) 从 F 中删去 T_i、T_j，并将 T_h 加入 F。若 F 中仍多于一棵二叉树，则返回②，直到 F 中只含一棵二叉树为止，这棵二叉树就是哈夫曼树。

上述哈夫曼算法可知，假设初始森林中共有 n 棵二叉树，由此可知，最终求得的哈夫曼树中共有 2n-1 个结点，其中 n 个叶结点是初始森林中的 n 个结点，并且哈夫曼树中没有度数为 1 的分支结点。

三、哈夫曼编码☆☆☆

假设一个待传输的文本中共有 n 种不同的字符，每种字符出现的次数为 w_i，该种字符的编码长度为 m_i。设总的编码个数为 S，则

平均编码长度= $\frac{1}{S}\sum_{i=1}^{n} w_i m_i = \sum_{i=1}^{n} \frac{w_i}{S} m_i = \sum_{i=1}^{n} p_i m_i$

其中，p_i 是第 i 种字符的出现频率。当平均码长达到最短时，文本总的码长也达到最短。

由上一节知识知道，将{p_1，…，p_n}作为一组叶结点的权值，用哈夫曼算法，可以构造出一棵具有最小带权路径长度的二叉树，这个最小带权路径长度恰好是 $\sum_{i=1}^{n} p_i m_i$。将该二叉树中每个结点的左分支标志“0”，每个结点的右分支标志为“1”，这样，从根到每个叶结点形成序列，将该序列作为叶结点对应字符的编码，由此得到的二进制编码称为哈夫曼编码。

【习题演练】

1、含有 100 个结点的二叉树采用二叉链表存储时，空指针域 NULL 的个数是（ ）

A:99 个　　B:100 个

C:101 个　　D:200 个

答案：C

解析：具有 n 个结点的二叉树中，有 2n 个指针域，其中只有 n-1 个用来指向结点的左、右孩子，其余的 n+1 个指针域为 NULL。

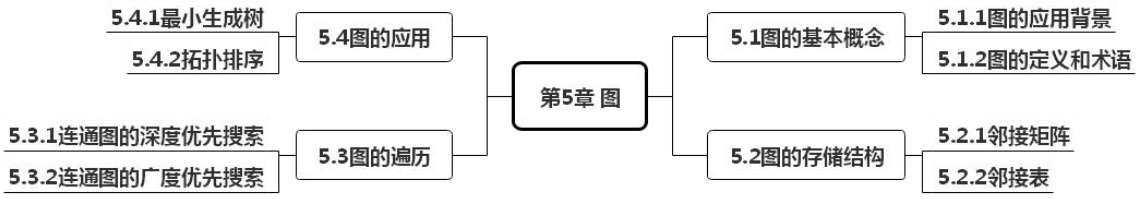
2、设一个完全二叉树共含有 196 个结点，则该完全二叉树中含有叶结点的个数是_____。

答案：98

解析：二叉树性质 3：对任何一棵二叉树，若度数为 0 的结点（叶结点）个数为 n0，度数为 2 的结点个数为 n2，则 n0=n2+1；已知：结点总数 n=n0+n1+n2；因为是完全二叉树，总结点数是 196，是偶数，故度为 1 的结点个数 n1=1。

综上可得 n2=97，故 n0=97+1=98。

第五章 图



第零节 图☆

在树形结构中，结点间具有层次关系，每一层结点只能和上一层中的至多一个结点相关，但可能和下一层的多个结点相关。而在图结构中，任意两个结点之间都可能相关，即结点之间的邻接关系可以是任意的。图结构可以描述多种复杂的数据对象，应用较为广泛。

第一节 图的基本概念

一、图的应用背景☆

图结构中的圆圈称为顶点，连线称为边，连线附带的数值称为边的权。

三、图的定义和术语☆☆

图： G=(V，E)，其中，V 是顶点的有穷非空集合；E 是边的集合，边是 V 中顶点的偶对。	有向图	若顶点的偶对是有序的，有序偶对用尖括号<>括起来；
	无向图	若顶点的偶对是无序的，无序偶对用圆括号（）括起来。
弧：有向图的边	弧头	有序偶对<v，w>表示有向图 G 中从 v 到 w 的一条弧，w 称为弧头或终点。
	弧尾	有序偶对<v，w>表示有向图 G 中从 v 到 w 的一条弧，v 称为弧尾或始点
完全图	无向完全图	任何两点之间都有边的无向图称为无向完全图。一个具有 n 个顶点的无向完全图的边数为 n(n-1)/2。
	有向完全图	任何两点之间都有弧的有向图称为有向完全图。一个具有 n 个顶点的有向完全图的弧数为 n(n-1)。
权	权	图的边附带数值，这个数值叫权
	带权图	权在实际应用中可表示从一个顶点到另一个顶点的距离、代价或耗费等。每条边都带权的图称为带权图。
度：无向图中顶点 v 的度是与该顶点相关联的边的数目，记为 D(v)。 D(v)=ID(v)+OD(v)。	入度	如果 G 是一个有向图，则把以顶点 v 为终点的弧的数目称为 v 的入度，记为 ID(v)；
	出度	把以顶点 v 为始点的弧的数目称为 v 的出度，记为 OD(v)。

子图	设 G=(V，E)是一个图，若 E'是 E 的子集，V'是 V 的子集，并且 E'中的边仅与 V'中的顶点相关联，则图 G'=(V'，E')称为图 G 的子图。	
路径	路径	在无向图 G= (V，E)中，从顶点 v 到顶点 v'的路径是一个顶点序列 :v ,v_{i1} ,v_{i2} ,… ,v_{im}, v',其中(v, v_{i1})，（v_{i1}，v_{i2})，…，（v_{im}，v')为图 G 中的边。
	路径长度	路径上边(或弧)的数目称为路径长度。
	简单路径	序列中顶点不重复出现的路径
	回路或环	第一个顶点和最后一个顶点相同的路径
	简单回路或简单环	除了第一个顶点和最后一个顶点外，其余顶点不重复的回路
连通	连通	在无向图中，如果从顶点 v 到顶点 v'有路径，则称 v 和 v'是连通的。
	连通图	如果图中的任意两个顶点 vi 和 vj 都是连通的，则称 G 为连通图。
	连通分量	是无向图中的极大连通子图。
	强连通图	对于有向图来说，如果图中任意一对顶点 vi 和 vj (其中 i≠j)都有顶点 vi 到顶点 vj 的路径，也有从 vj 到 vi 的路径，即两个顶点间双向连通，那么称该有向图是强连通图。
	强连通分量	有向图的极大强连通子图
生成树	生成树	一个连通图的生成树，是含有该连通图的全部顶点的一个极小连通子图。
	生成森林	在非连通图中，由每个连通分量都可得到一个极小连通子图，即一棵生成树。那么这些连通分量的生成树就组成了一个非连通图的生成森林。

第二节 图的存储结构

一、邻接矩阵☆☆☆☆

1、邻接矩阵就是用矩阵来描述图中顶点之间的关联关系，用二维数组来实现矩阵。

设 G=(V，E)是一个图，其中 V={v0, v1，…，vn-1}，那么 G 的邻接矩阵 A 定义为如下的 n 阶方阵：

$$A[i][j]=\begin{cases}1 & \text{若}(v_i,\ v_j)\text{或}<v_i,\ v_j>\text{是 E 中的边}\\ 0 & \text{若}(v_i,\ v_j)\text{或}<v_i,\ v_j>\text{不是 E 中的边}\end{cases}$$

注意：无向图的邻接矩阵是一个对称矩阵。

2、无向带权图邻接矩阵的建立算法描述如下：

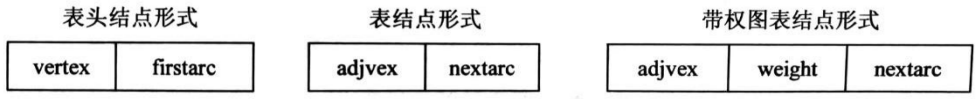
```
void CreatGraph(Graph *g){
    int i,j,n,e,w;
    char ch ;
```

```
    scanf("%d %d",&n,&e);                    //读入顶点个数和边数
    g->vexnum=n;
    g->arcnum=e;
    for (i=0;i<g->vexnum;i++){
        scanf ("%c" ,&ch);                   //读入顶点信息
        g->vexs[i]=ch;
    }
    for (i=0;i<g->vexmim;i++)
        for (j=0;j<g->vexnum;j++)
            g->arcs[i][j]=KAX_INT;           //初始化邻接矩阵
    for (k=0;k<g->arcnum;k++){
        scanf("%d   %d   %d",&i,&j,&w); //读入边(顶点对)和权值
        g->arcs[i][j]=w;
        g->arcs[j][i]=w;
    }
}
```

二、邻接表☆☆☆☆

1、邻接表是顺序存储与链式存储相结合的存储方法。

对于无向图，第 i 个单链表中的结点表示依赖于顶点 vi 的边，对于有向图是以顶点 vi 为尾的弧，这个单链表称为顶点 vi 的邻接表。



2、在邻接表上如何求顶点的度

无向图中顶点 v_i 的度恰为第 i 个单链表中的结点数。对有向图，第 i 个单链表中的结点个数只是顶点 v_i 的出度。为了求入度，必须遍历整个邻接表。在所有单链表中，其邻接点域的值为 i 的结点的个数是顶点 v_i 的入度。对于有向图，逆邻接表可以很容易求出 v_i 的入度。

3、建立有向图的邻接表的算法描述如下：

```
CreateAdjlist(Graph *g){
    int n,e,i,j,k;
    ArcNode *p;
    scanf ( "%d %d%" ,n,e);                  //读入顶点数和边数
    g->vexnum=n;g->arcnum=e;
    for (i=0;i<n;){
        g->adjlis[i].vertex=i;               //初始化顶点 vi 的信息
        g->adjlis[i].firstarc=NULL;          //初始化 i 的第一个邻接点为 NULL
    }
```

```
    for (k=0; k<e; k++ ){                    //输入 e 条弧
        scanf { "%d %d" ,&i,&j );
        p= (ArcNode*)malloc(sizeof (ArcNode);    //生成 j 的表结点
        p->adjvex=j;
        p->nextarc=g->adjlis[i].firstarc;        //将结点 j 链到 i 的单链表中
        g->adjlis[i].firstarc=p;
    }
}
```

第三节 图的遍历☆☆

方法	基本思想
深度优先搜索	连通图深度优先搜索的基本思想：假定以图中某个顶点 v_i 为出发点，首先访问出发点 v_i，然后任选一个 v_i 的未访问过的邻接点 v_j，以 v_j 为新的出发点继续进行深度优先搜索，依此类推，直至图中所有顶点都被访问过。深度优先搜索遍历类似于树的**先序遍历**。
广度优先搜索	连通图广度优先搜索的基本思想是：从图中某个顶点 v_i 出发，在访问了 v_i 之后依次访问 v_i 的所有邻接点，然后依次从这些邻接点出发按广度优先搜索方法遍历图的其他顶点，重复这一过程，直至所有顶点都被访问到。广度优先搜索遍历类似于树的按**层次遍历**的过程。

第四节 图的应用

一、最小生成树☆☆☆☆

一个图的最小生成树是图所有生成树中权总和最小的生成树。以下为最小生成树算法：

Prim 算法	假设 G=(V, E)是一个带权图，生成的最小生成树为 MinT=(V，T)，其中 V 为顶点的集合，T 为边的集合。求 T 的步骤如下： (1) 初始化：U={u_0}，T={ }。其中 U 为一个新设置的顶点的集合，初始 U 中只含有顶点 u_0，这里假设在构造最小生成树时，从顶点 u_0 出发； (2) 对所有 u∈U，v∈V-U(其中 u, v 表示顶点) 的边(u, v)中，找一条权最小的边(u'，v')，将这条边加入到集合 T 中，将顶点 v'加入到集合 U 中； (3) 如果 U=V，则算法结束；否则重复第（2）、（3）步。
克鲁斯卡尔(Kruskal) 算法	(1) 设 G=(V，E)，令最小生成树初始状态为只有 n 个顶点而无边的非连通图 T= (V, {})，每个顶点自成一个连通分量； (2) 在 E 中选取代价最小的边，若该边依附的顶点落在 T 中不同的连通分量上，则将此边加入到 T 中，否则，舍去此边，选取下一条代价最小的边； (3) 依此类推，重复（2)，直至 T 中所有顶点都在同一连通分量上为止。
Dijkstm 算法求单源最短路径	给定一个带权有向图 G=(V，E)，其中每条边的权是非负实数。另外，给定 V 中的一个顶点， 称为源。要计算从源到其他各顶点的最短路径长度。这里的长度是指路径上各边权值之和。

二、拓扑排序

1、 AOV 网☆☆

如果以图中的顶点来表示活动，有向边表示活动之间的优先关系，这种用顶点表示活动的有向图称为 AOV 网。AOV 网中的弧表示了活动之间存在着的制约关系。

2、拓扑排序☆☆☆☆

①有向图拓扑排序算法的基本步骤：

(1) 图中选择一个入度为 0 的顶点，输出该顶点；

(2) 从图中删除该顶点及其相关联的弧，调整被删弧的弧头结点的入度（入度减 1）；

(3) 重复执行（1）、（2）直到所有入度为 0 的顶点均被输出，拓扑排序完成，或者图中再也没有入度为 0 的顶点。

任何一个无环有向图，其全部顶点可以排成一个拓扑序列。

②拓扑排序算法描述如下：

```
Tp_Sort(Graph g) {
    建立图 g 中入度为 0 的顶点的栈 s;
    m=0;                              //m 记录输出的顶点个数
    while (!EmptyStack(S)){           //当栈非空
        Pop (S, v);                   //弹出栈顶元素，赋给 v
        输出 v;
        m ++;
        w=Firstvex(g,v);              //图 g 中顶点 v 的第一个邻接点
        while (w 存在 ) {
            W 的入度减 1;
            if (w 的入度==0)
                push (s, w);
            Nextvex(g,v,w);           //图 g 中顶点 v 的下一个邻接点
        }
    }
    if (m < n )  printf ("图中有环\n");
}
```

拓扑排序算法的时间复杂度为 O(n+e)，n 是图的顶点个数，e 是图的弧的数目。

【习题演练】

1、有向图中某顶点 v 的入度为 2，出度为 3，则该顶点的度为（ ）

A:3　　B:4

C:5　　D:6

答案：C

解析：有向图中顶点 v 的度为入度与出度的和。

2、无向图的邻接矩阵为（ ）

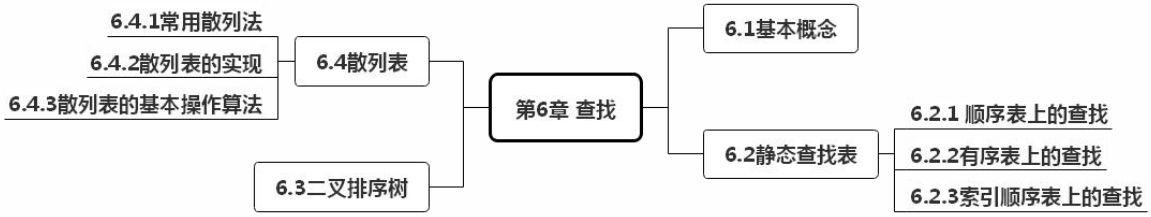
A:对角矩阵　　B:对称矩阵

C:稀疏矩阵　　D:一般矩阵

答案：B

解析：无向图的邻接矩阵是一个对称矩阵。

第六章 查找



第一节 基本概念☆☆

查找表(Search Table)是由同一类型的数据元素构成的集合，它是一种以查找为“核心”，同时包括其他运算的非常灵活的数据结构。

分类	操作
静态查找表	建表、查找、读表中元素
动态查找表	初始化、查找、读表中元素、插入、删除

第二节 静态查找表

一、顺序表上的查找☆☆☆☆

1、将“查找成功时的平均查找长度”（记为 ASL)作为评价查找算法时间性能的度量。

$ASL=\sum_{i=1}^{n}P_iC_i$，其中，P_i为查找第 i 个元素(即给定值 key 与顺序表中第 i 个元素的键值相等)的概率，且$\sum_{i=1}^{n}P_i=1$，C_i表示在找到第 i 个元素时，与给定值已进行比较的键值个数。

2、查找算法描述如下：

```
int SearchSqTable (SqTable T, KeyType key){
/*在顺序表 T 中，从后往前查找键值等于 key 的数据元素,若找到，则返回该元素在 T 中的位置，否则返回 0,标记查找不成功*/
    T.elem[0].key=key;              //设置岗哨
    i=T.n;                          //设置比较位置初值
    while{T.elem[i].key!=key)
        i--;                        //未找到时，修改比较位置继续查找
    return i;
}
```

二、有序表上的查找☆☆

1、有序表：如果顺序表中数据元素是按照键值大小的顺序排列的，则称为有序表。适合用二分查找。

2、用给定值 key 与处在中间位置的数据元素 T.elem[mid]的键值 T.elem[mid] .key 进行比较，可根据三种比较结果区分三种情况：

(1) key=T.elem[mid].key，查找成功，T.elemEmid]即为待查元素；

(2) key<T.elem[mid].key,说明若待查元素若在表中，则一定排在 T.elem[mid] 之前；

(3) key>T.elem[mid].key,说明若待查元素若在表中，则一定排在 T.elem[mid]之后。

3、平均查找长度

二分查找的查找长度不超过 L$\log_2$n」+1。

二分查找的平均查找长度为 $\mathrm{ASL}_{\mathrm{b}}=\frac{n+1}{n}\log_2(n+1)-1$

当 n 较大时可得：$\mathrm{ASL}_{\mathrm{b}}\approx\log2(n+1)-1$

三、索引顺序表上的查找

1、索引顺序表是结合了顺序查找和二分查找的优点构造的一种带索引的存储结构。一个索引顺序表由两部分组成：一个索引表和一个顺序表。

2、若静态查找表用索引顺序表表示，则查找操作可用分块查找来实现，也称为索引顺序查找。分块查找分两步进行：先确定待查数据元素所在的块；然后在块内顺序查找。

3、分块查找的平均查找长度：等于两阶段各自的查找长度之和。若每块含 S 个元素，且第一阶段采用顺序查找，则在等概率假定下，分块查找的平均查找长度为 $\mathrm{ASL}_{\mathrm{bs}}=\frac{1}{2}\left(\frac{n}{s}+s\right)+1$其中，n 为顺序表中的数据元素数目。当 s 取$\sqrt{\mathrm{n}}$时，$\mathrm{ASL}_{\mathrm{bs}}$达到最小值$\sqrt{\mathrm{n}}$+1。

第三节 二叉排序树

1、二叉排序树上的查找

```
BinTree SearchBST(BinTree bst ,KeyType key)
/*在根指针 bst 所指的二叉排序树上递归地查找键值等于 key 的结点。若成功，则返回指向该结 点的指针，否则返回空指针*/
{    if (bst==NULL) return NULL;              //不成功时返回 NULL 作为标记
     else  if (key==bst->key) return bst;     //成功时返回结点地址
          else  if(key<bst->key)
                    return SearchBST (bst->lchild, key);    //继续在左子树中查找
               else
                    return SearchBST (bst->rchild, key);    //继续在右子树中查找
}
```

2、二叉排序树的插入☆☆☆

在二叉排序树上进行插入的原则是：必须要保证插入一个新结点后，仍为一棵二叉排序树。这个结点是查找不成功时查找路径上访问的最后一个结点的左孩子或右孩子。

3、二叉排序树的查找分析☆☆☆☆

二叉排序树上的平均查找长度是介于 O(n)和 O($\log_2$n)之间的 ,其查找效率与树的形态有关。

二叉排序树的平均查找长度 ASL≤1+$\log_2$n。

第四节 散列表☆

数据元素的键值和存储位置之间建立的对应关系 H 称为散列函数，用键值通过散列函数获取存储位置的这种存储方式构造的存储结构称为散列表。设有散列函数 H 和键值 k_1、k_2($k_1 \neq k_2$)， 若 $H(k_1)=H(k_2)$，则这种现象称为“冲突”，且称键值 k_1 和 k_2 互为同义词。

一、常用散列法☆☆

数字分析法	又称数字选择法，其方法是收集所有可能出现的键值，排列在一起，对键值的每一位进行分析，选择分布较均匀的若干位组成散列地址。所取的位数取决于散列表的表长，见表长为 100，则取 2 位即可。
除留余数法	是一种简单有效且最常用的构造方法，其方法是选择一个不大于散列表长 n 的正整数 p ,以键值除以 p 所得的余数作为散列地址 ,即 H (key) = key mod p (p≤n)注意：通常选 p 为小于散列表长度 n 的素数。
平方取中法	以键值平方的中间几位作为散列地址。通常在选定散列函数时不一定能知道键值的分布情况。所得散列地址比较均匀。
基数转换法	将键值看成另一种进制的数再转换成原来进制的数，然后选其中几位作为散列地址。

二、散列表的实现☆☆☆

假设散列表的地址集为 0∽(n-1)，冲突是指由键值得到的散列地址上已存有元素，通常用来解决冲突的方法有：

线性探测法	对任何键值 key，设 H (key) =d，设散列表的容量为 m，则线性探测法生成的后继散列地址序列为 d+1，d+2，…，m-1，0，1，…，d-1
二次探测法	生成的后继散列地址不是连续的，而是跳跃式的，以便为 后续数据元素留下空间从而减少堆积。按照二次探测法，键值 key 的散列地址序列为 $d_0=H(key)$；$d_i=(d_0+i) \bmod m$。其中，m 为散列表的表长，$i=1^2$，-1^2，2^2，-2^2，…，$\pm k^2(k \leq m/2)$。
链地址法	链地址是对每一个同义词都建一个单链表来解决冲突，其组织方式如下： 设选定的散列函数为 H, H 的值域(即散列地址的范围)为 0∽(n-1)。设置一个“指针向量” Pointer HP [n]，其中的每个指针 HP[i]指向一个单链表，该单链表用于存储所有散列地址为 i 的数据元素。每一个这样的单链表称为一个同义词子表。

三、散列表的基本操作算法

1、链地址法散列表

（1）查找：首先计算给定值 key 的散列地址 i,由它到指针向量中找到指向 key 的同义词子表的表头指针。然后，在该同义词子表中顺序查找键值为 key 的结点。算法描述如下：

```
Pointer SearchLinkHash(KeyType key, LinkHash HP){
/*在散列表 HP 中查找键值等于 key 的结点，成功时返回指向该结点的指针，不成功时返回空指针*/
    i=H (key);          //计算 key 的散列地址
    P=HP [i];           //同义词子表表头指针传给 p
```

```
    if (p==NULL)   return NULL;
    while ( (p!=NULL) && (p->key!^key))
        p=p->next;        //未达同义词子表表尾且未找到时，继续扫描
    return p;
}
```

2、线性探测法散列表

（1）查找

```
int SearchOpenHash(KeyType key,OpenHash HL){
/*在散列表 HL 中查找键值为 key 的结点。成功时返回该位置；不成功时返回标志 0,假定以线性探测法解决冲突*/
    d=H(key);                          //计算散列地址
    i=d;
    while( (HL[i].key!-NULLkey)        //HL[i]填有元素，NULLkey 表示空的键值
            && (HL[i] .keyl=key))      //未找到
        i=(i+1)%m;                     //未成功且未查遍整个 HL 时继续扫描
    if (HL[i].key==key) return i;      //查找成功
    else   return 0;                   //查找失败
}
```

【习题演练】

1、静态查找表是以具有相同特征的数据元素集合为逻辑结构，包括建表、_____、读表中元素三种基本运算。

答案：查找

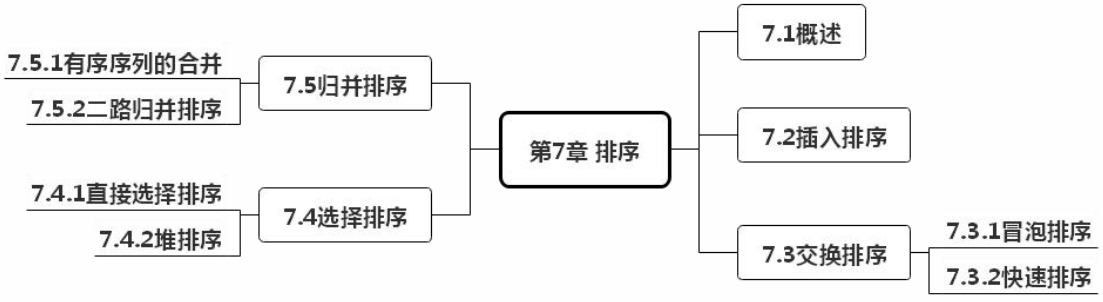
解析：静态查找表是以具有相同特性的数据元素集合为逻辑结构，包括下列三种基本运算：建表、查找、 读表中元素。

2、对长度为 n 的有序顺序表进行二分查找， 则查找表中的任意一个元素时，无论查找成功与失败， 最多与表中_____个元素进行比较。

答案：$\lfloor \log_2 n \rfloor +1$

解析：二分查找算法每进行一次键值与给定值的比较，查找区间的长度至少减小为原来二分之一， “二分查找”由此得名。由此易推算出二分查找的查找长度不超过$\lfloor \log_2 n \rfloor +1$。

第七章 排序



第一节 概述☆

n 个记录的序列为{R_1，R_2,…，R_n}，其相应的键值序列为{k_1， k_2，…，k_n},假设 $k_i=k_j$，若在排序前的序列中民在均之前，即 i<j，经过排序后，R_i 仍在 R_j 之前，则称所用的排序方法是稳定的；反之，则称所用的排序方法是不稳定的。

排序分为：内部排序和外部排序。内部排序的方法：插入排序、交换排序、选择排序和归并排序等四种。

第二节 4 种排序方法

插入排序 ☆☆	直接插入排序	依次将每个记录插入到一个已排好序的有序表中去，从而得到一个新的、记录数增加 1 的有序表。直接插入排序类似图书馆中整理图书的过程。	是稳定的。 时间复杂度为 O(n2)。 空间复杂度为 O(1)。
交换排序 ☆☆☆☆	冒泡排序	①将第一个记录的键值和第二个记录的键值进行比较，若为逆序(即 R[1].key>R[2].key)，则将这两个记录交换，然后继续比较第二个和第三个记录的键值。依次类推，直到完成第 n-1 个记录和第 n 个记录的键值比较交换为止。上述过程称为第一趟起泡，其结果使键值最大的记录移到了第 n 个位置上。 ②第二趟起泡排序，即对前 n-1 个记录进行同样操作，其结果是次大键值的记录安置在第 n-1 个位置上。 ③重复以上过程，当在一趟起泡过程中没有进行记录交换的操作时，整个排序过程终止。	是稳定的。 时间复杂度为 O(n2)。
	快速排序	在 n 个记录中取某一个记录的键值为标准，通常取第一个记录键值为基准，通过一趟排序将待排的记录分为小于或等于这个键值和大于这个键值的两个独立的部分，这时一部分的记录键值均比另一部分记录的键值小，然后，对这两部分记录继续分别进行快速排序，以达到整个序列有序。	是不稳定的。 就平均时间性能而言，快速排序方法最佳，其时间复杂度为 O(nlog2n)。但在最坏情况下，近似于 O(n2)。对较大的 n 值效果较好。
选择排序☆☆	直接选择排序	在第 i 次选择操作中，通过 n-i 次键值间比较，从 n-i+1 个记录中选出键值最小的记录，并和第 i(1≤i≤n-l)个记录交换。	是不稳定的。 时间复杂度为 $O(n^2)$。 但不适宜于 n 较大的情况。
	堆排序	①最小堆可以看成是一棵以 k_1 为根的完全	是不稳定的。

		二叉树，任一结点的值都不大于它的两个孩子的值(若存在孩子的话)。k_1 是堆中最小的元素，并且这种二叉树的任一子树本身也是一个堆。 ②最大堆可以看成是一棵以 k_1 为根的完全二叉树，任一结点的值都不小于它的两个孩子的值(若存在孩子的话)。k_1 是堆中最大的元素，并且这种二叉树的任一子树本身也是一个堆。	对于 n 个记录进行排序所需的平均时间是 O(n log2n)。在最坏情况下，其时间复杂度也为 O(n log2n)。相对于快速排序来说，这是堆排序的最大优点。
归并排序	有序序列的合并	核心操作是两个有序子序列的合并。	此算法的执行时间为 O(n-h+1)。
	二路归并排序 ☆☆☆☆	将两个有序表合并成一个有序表的排序方法，其基本思想：假设序列中有 n 个记录，可看成是 n 个有序的子序列，每个序列的长度为 1。首先将每相邻的两个记录合并，得到⌈ n/2⌉ 个较大的有序子序列 ,每个子序列包含 2 个记录，再将上述子序列两两合并，得到⌈⌈ n/2⌉ /2⌉ 个有序子序列，如此反复 ,直至得到一个长度为 n 的有序序列为止，排序结束。	是稳定的。 时间复杂度为 $O(n\log_2 n)$。在 n 较大时，归并排序的时间性能优于堆排序，但它所需的辅助存储量较多。

【习题演练】

1、快速排序属于（ ）

A:插入排序　　B:交换排序

C:选择排序　　D:归并排序

答案：B

解析：快速排序是交换排序的一种，实质上是对冒泡排序的一种改进。

2、下列排序方法中不稳定的是（ ）

A:冒泡排序　　B:二路归并

C:堆排序　　D:直接插入排序

答案：C

解析：堆排序是不稳定的。其余排序方法都是稳定的。