梦享考研系列

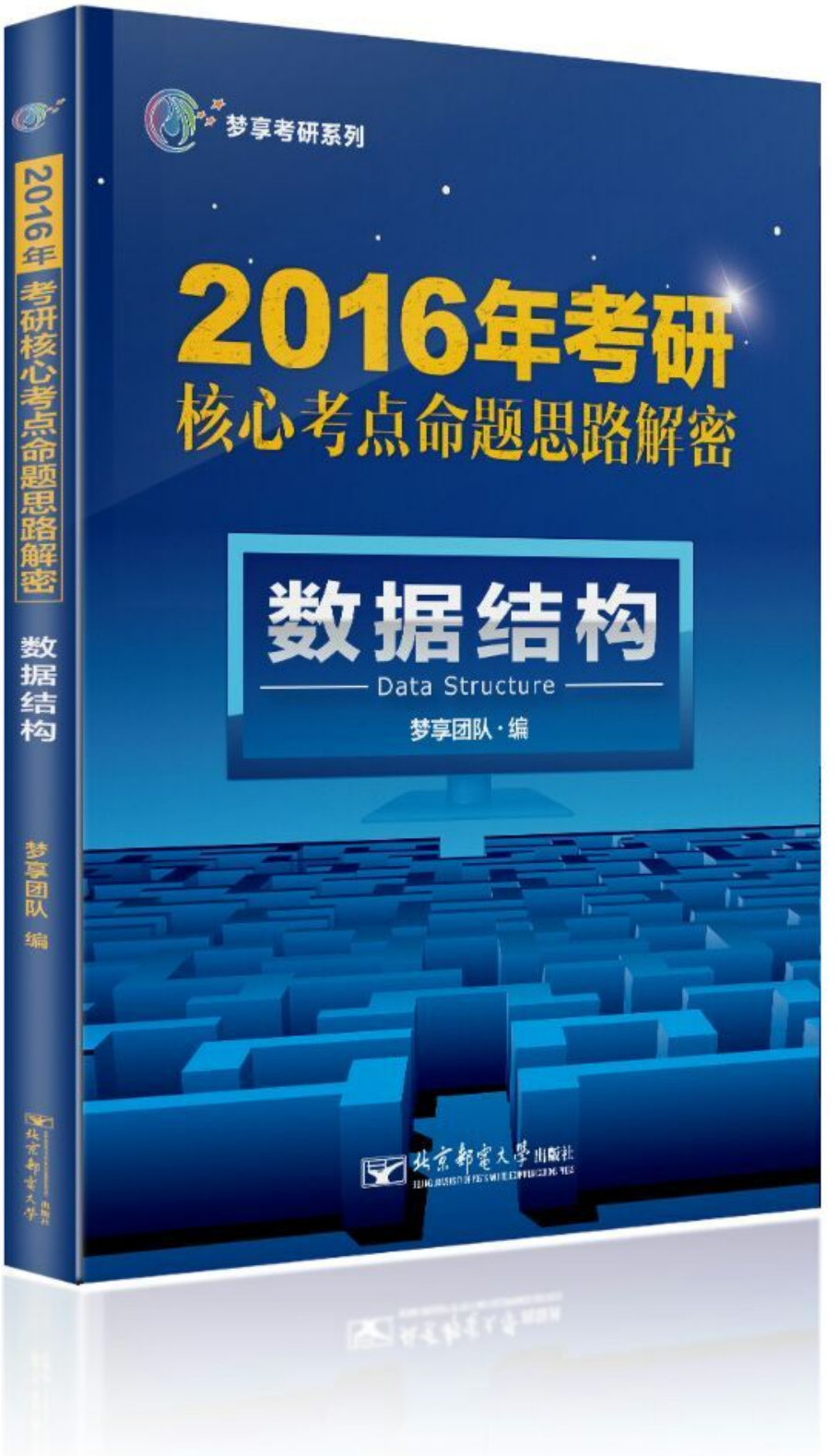
2016年考研核心考点命题思路解密

数据结构

梦享团队　组编

最好的习题集，从梦享做起...

一本可用于统考和非统考的辅导书



内容简介

《2016 年考研核心考点命题思路解密——数据结构》严格按照最新计算机考研 408 统考大纲的数据结构部分编写，涵盖大纲指定的所有考试内容。本书对统考大纲所涉及的知识点进行深入剖析和总结，并精心策划和部署每一个章节，对每一个章节的考点做了独家策划。

本书每一个考点中的命题，绝大部分来源于历年名校计算机考研真题和统考真题，少部分来源名校期末考试试题中的精华部分，是全国 408 统考大纲和高校考研真题的较好结合。为了提高考题的质量和解析的准确度，参考资料采用以考研权威教材、习题、考研真题为主，多方借鉴众多高校从事多年教育的教师课堂资料。梦享团队对每一个命题的思路和解题方法进行深入详细地讲解，并附上大量的图来帮助考生理解记忆，力求考生能够通过掌握一个题目而达到举一反三，有利于考生利用更少的时间掌握更多的知识。

本书可作为考生参加计算机专业研究生入学考试的备考复习用书（包括统考和非统考的考生），也可作为计算机专业的学生学习数据结构的练习用书。

前　　言

梦享团队队员以中科院、清华大学和北京交通大学 3 所高校的学生为主，其他名校学生为辅，都是上研不久的研究生，以及一些考研论坛上参与答疑多年的版主等。在考研复习和辅导上，梦享团队队员有着相对丰富的阅历。在考研的路上，梦享团队队员也经历过和大家一样的坎坷辛苦。我们深切地体会到，每一个考研的同学十分不容易。

计算机考研的同学注重基础知识的掌握，更加看重实战能力。但目前的考研教材绝大多数倾向于知识点的讲解，不注重培养考生的实战能力，导致很多考生知识很丰富，但是很难这些知识很好地运用于解题。编写偏向于实战的参考书不同于知识讲解，需要编者花费大量的时间来规划和布置章节、考点和解析考题。目前市面上现在能找到的计算机考研命题解析类参考资料，要么题目特别少但讲解特别详细啰嗦，要么题目太多的而对命题的讲解十分粗略甚至只有一个最终答案。因而，梦享团队决定写一套注重实战、解析详细、严格遵照大纲的参考书。

经过两年多的努力，“梦享考研系列”参考书终于一本一本地和大家见面了，到目前为止，我们总共有 5 本图书已经出版。其中，《计算机网络》完成于 2013 年 10 月，《数据结构》完成于 2014 年 3 月，《计算机操作系统》完成于 2014 年 9 月，《计算机组成原理》完成于 2015 年 3 月。最后一本书，《408 统考核心题型》，成于 2015 年 5 月。

为了提高图书的权威性，本套图书严格按照 408 统考大纲编写，涵盖了统考大纲所有指定的内容，并融合了历年名校考研真题的精华，是全国 408 统考大纲和高校考研真题的较好结合。为了提高考题的质量和解析的准确度，参考资料采用以考研权威教材、习题、考研真题为主，多方借鉴众多高校从事多年教育的教师课堂资料。

本书具有以下特色：

1. 组织严谨，结构清晰

梦享考研系列图书通过对统考大纲和历年高校考研真题的深入剖析和总结，精心规划和部署了各个章节，对每一个章节的考点作了独家策划，使得本套图书组织严谨，结构清晰，便于大家对各章考点各个击破。

2. 突出重点，注重实战

对于每一个计算机专业的考研同学而言，时间是相当有限的。除了四门统考的繁重专业课之外，我们还有数学、英语和政治，复习工作量相当大。所以，突出重点，让同学们

把极其有限的时间都花在刀刃上，是我们的首要工作。而提高同学们的实战能力，是我们系列图书的最终目的。

因此，在考题的挑选上，我们通过对统考和自主命题高校常出现的考题类型和知识点进行深入的总结，抛开在统考或者自主命题的考研真题上极少出现的极难、极易、极偏知识点，精心挑选了和考研难度相近的题型和考题供大家练习，提高同学们的实战能力。

此外，我们还根据考点的重要程度来完成考点内容分布，在较重要的考点部署较多的内容，在较重要的内容部署较多的命题，在较为不重要的知识点抓住重点布置核心题型。目的也在于突出考试难度、突出考试重点，方便大家进行实战训练，提高学习效率，让考生在更短的时间内掌握更多的知识点。

3. 解析详细，深入剖析

“梦享考研”系列图书一共 5 本，每一本都很厚，可能会吓怕很多同学。是不是题目太多了？不是的，其实考题并不多，我们并不提倡题海战术，也不提倡对于同一个知识点反复命题和赘述，我们提倡“少而精”。针对每一个考点可能出现的命题类型，我们精心挑选了极具代表性的命题供大家实战训练，并对这些习题进行详细、深入的剖析，揭露问题的本质和解题的精髓，有助于大家掌握解题方法和技巧，提高大家的实战能力，在较短的时间掌握更多的知识。

《2016 年计算机考研核心考点命题思路解密》系列图书是我们团队 2 年多的汗水结晶，融入了我团队的集体智慧。另外，真诚感谢我们团队新成员单开元、殷巧云、高楠、张丽方、胡明明、刘春、白洋等 10 几位同学提供的建议和帮助！

在接下来的更长时间里，我们团队将日益强大，用我们最诚挚的心和最大的努力，给大家展示出更好的图书。我们每年都会合理调整这套图书，使得这套图书更加受到大家青睐。

梦享团队我们会牢牢记住这样一句话——“助你们实现研究生梦想，是我们的梦想！”伴随着 2016 年考研的同学一起度过艰辛的追梦季节，伴随着大家一起度过每一个艰辛的日日夜夜！也祝福 2016 年考研的你们，获得圆满的成功！

安楠

2016 年 3 月于北京

目　录

第一章　绪论

考点 1 数据的逻辑结构、存储结构

温馨提示：本考点主要考查：1、集合结构、线性结构、树结构和图结构的特点；2、抽象数据类型的定义和表示方法。请同学们注意区分什么是数据的逻辑结构，什么是数据的存储结构。

一．选择题部分

1. （原书　第 1 题）在数据结构的讨论中把数据结构从逻辑上分为(　　)。

A. 内部结构与外部结构　　B. 静态结构与动态结构

C. 线性结构与非线性结构　　D. 紧凑结构与非紧凑结构

【考查内容】数据结构的逻辑划分。

【解析】数据的**逻辑结构**是对数据之间关系的描述，有时就把逻辑结构简称为数据结构。从逻辑上可以将数据结构分为线性结构和非线性结构，我们常见的顺序表，就是线性结构，而树形结构和图形结构是非线性结构。

线性结构中元素之间存在一对一关系，**非线性结构**中元素之间存在一对多关系或者多对多关系。

【参考答案】C

2. （原书　第 3 题）在存储数据时，通常不仅要存储各数据元素的值，而且还要存储(　　)。

A. 数据的处理方法　　B. 数据元素的类型

C. 数据元素之间的关系　　D. 数据的存储方法

【考查内容】数据存储。

【解析】**在存储数据时，不仅要存储各数据元素的值，还要存储数据元素之间的关系。**顺序存储方法把逻辑上相邻的结点存储在物理位置相邻的存储单元里，结点间的逻辑关系

由存储单元的邻接关系来体现。链式存储方法不要求逻辑上相邻的结点在物理位置上亦相邻，结点间的逻辑关系由附加的指针表示。

【参考答案】C

3. （原书　第 5 题）在决定选取何种存储结构时，一般不考虑(　　)。

A. 各结点的值如何

B. 结点个数的多少

C. 对数据有哪些运算

D. 所用的编程语言实现这种结构是否方便

【考查内容】影响存储结构选取的因素。

【解析】在决定选取那种存储结构时，一般不考虑结点的值如何，但是需要考虑**结点的个数、对数据有哪些运算、以及所用的编程语言支不支持这种存储结构**等。比如，结点个数动态增长时，采用顺序表就不太适合。再比如，常对线性表进行插入和删除操作，则采用顺序表不适合。再比如，Java 语言不支持指针，不能选用链式存储结构。

【参考答案】 A

4. （原书　第 11 题）数据结构 DS(Data Struct)可以被形式地定义为 DS=（D，R），其中 D 是(　　)的有限集合，R 是 D 上的关系有限集合。

A. 算法　　B. 数据元素　　C. 数据操作　　D. 数据对象

【考查内容】数据结构的形式化定义。

【解析】数据结构是一个二元组，可以定义数据结构为 Data_Structure=(D,R)，其中 D 是数据元素的有限集，R 是 D 上的关系的有限集。

【参考答案】 B

考点 2 算法以及算法的时间复杂度和空间复杂度

温馨提示：问题主要考查算法的时间复杂度和空间复杂度的概念，计算方法，数量级表示。对于统考的考生而言，分析算法的时间复杂度，是一种常见的选择题题型。另外，通过 for 循环或者 while 循环来让同学们计算算法的时间复杂度的题型，也请同学们多留意。

一．选择题部分

1. **（原书　第 3 题）**算法分析的目的是(　　)。

A. 找出数据结构的合理性

B. 研究算法中的输入和输出的关系

C. 分析算法的效率以求改进

D. 分析算法的易懂性和文档性

【考查内容】算法分析的目的。

【解析】对算法的讨论不能只研究它是否能在有穷步内终止，还应对算法的运行效率作出分析，判断算法的好坏，以便在已有的资源条件下作出最佳的决策。算法分析的目的，是评价算法的效率，通过评价选用更加好更加有效的算法来求解问题。

【参考答案】 C

2. **（原书　第 4 题）**设语句 x++的时间是单位时间，则以下语句的时间复杂度为(　　)。

```
for(i=1; i<=n; i++)
    for(j=i; j<=n; j++)
        x++;
```

A. O(1)　　B. $O(n^2)$　　C. O(n)　　D. $O(n^3)$

【考查内容】给定循环语句的时间复杂度分析方法。

【解析】我们常说的分析算法的时间复杂度，就是分析算法的规模 n 的函数 f(n)。本题中存在着两层 for 循环，当 i=1 时，内层循环执行 n 次，当 i=2 时内层循环执行 n-1 次…，分析可知，总共执行了近$n^2/2$ 次，故而时间复杂度为 O(n^2)。

要特别注意，在分析时间复杂度时，我们通常采用**抓取大端**的办法，进行粗略估计，不会进行详细计算。

【参考答案】 B

3. **（原书　第 5 题）**下面程序段的时间复杂度是(　　)。

```
for(i=0;i<m;i++)
    for(j=0;j<n;j++)
        a[i][j]=i*j;
```

A. $O(m^2)$　　B. $O(n^2)$　　C. O(m*n)　　D. O(m+n)

【解析】本题考查给定循环语句的时间复杂度分析方法。题中的程序有两层 for 循环，外层循环执行 m 次，而每执行一次外层循环，内层循环需要执行 n 次，故而总共执行 mn 次，算法的**时间复杂度**为O(mn)。

【参考答案】 C

二．综合应用题部分

1. **（原书　第 1 题）**下面程序段的时间复杂度是_____。

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

【考查内容】给定循环语句的时间复杂度分析方法。

【解析】由算法可以看出，存在两层循环，外层循环执行了 n 次，而每执行一次外层循环，内层循环需要执行 n 次，总共执行了n^2次，故而时间复杂度为$O(n^2)$。

本章到此就结束了，请问您有什么疑问吗？任何问题，欢迎您与我们作者进行交流！

shareOurDreams
梦享团队微信号

weCSdream
梦享团队官方微信公众号

梦享论坛团队
梦享团队新浪微博

第二章　线性表

考点 1 线性表的定义和基本操作

温馨提示：本考点主要考查线性表的定义及判别和抽象数据类型的描述，线性表中每一种操作的功能，对应的函数名、返回值类型和参数表中每个参数的作用。请同学们掌握线性表的基本概念和相关操作。

一．选择题部分

1. **（原书　第 1 题）**下面关于线性表的叙述中，错误的是（　　）。

A. 线性表采用顺序存储，必须占用一片连续的存储单元

B. 线性表采用顺序存储，便于进行插入和删除操作。

C. 线性表采用链式存储，不必占用一片连续的存储单元

D. 线性表采用链式存储，便于进行插入和删除操作

【考查内容】线性表的顺序存储方式和链式存储方式的特点和优缺点。

【解析】关于线性表，我们常接触到的存储结构有顺序存储和链式存储。**顺序存储结构**的地址是连续的（即必须占用一片连续的存储空间），所以可以通过计算地址实现随机存取。**链式存储结构**的存储地址不一定连续，只能通过每一个结点的指针顺序存取。

采用顺序存储结构时，插入和删除元素都需要移动大量元素，不便于插入和删除操作。采用链式存储结构便于插入和删除操作，但是查找只能顺序进行，时间复杂度为 O(n)。

【参考答案】 B

2. **（原书　第 7 题）**以下关于线性表的说法不正确的是（　　）。

A. 线性表中的数据元素可以是数字、字符、记录等不同类型

B. 线性表中包含的数据元素个数不是任意的

C. 线性表中的每个结点都有且只有一个直接前趋和直接后继

D. 存在这样的线性表：表中各结点都没有直接前趋和直接后继

【考查内容】线性表的基本概念。

【解析】线性表中的数据元素具有抽象（即不确定）的数据类型，可以是数字、字符、记录等不同类型，在设计具体的应用程序时，数据元素的抽象类型将被具体的数据类型所取代。

线性表中包含的数据元素个数不是任意的，必须是有限的。

在一个非空表 L=（a_1，a_2,……a_n）中，任意一对相邻的数据元素 a_{i-1} 和 a_i 之间($1<i\leqslant n$)存在序偶关系(a_{i-1},a_i)，且 a_{i-1} 称为 a_i 的前驱，a_i 称为 a_{i-1} 的后继。在这个序列中，a_1 无前驱，a_n 无后继，其他每个元素有且仅有一个前驱和一个后继。

若为空表，或者表中只有一个元素，则该元素没有直接前驱结点，也没有直接后继结点。

【参考答案】 C

3. **(原书　第 10 题)**在以下的叙述中，正确的是（　　）。

A. 线性表的顺序存储结构优于链表存储结构

B. 线性表的顺序存储结构适用于频繁插入/删除数据元素的情况

C. 线性表的链表存储结构适用于频繁插入/删除数据元素的情况

D. 线性表的链表存储结构优于顺序存储结构

【考查内容】线性表的顺序存储和链式存储的比较。

【解析】线性表的顺序存储结构和链式存储结构各有其优缺点，不能说哪一种结构一定优于另一种结构。顺序存储便于查找，但不利于频繁地插入和删除。而线性表的链式存储结构便于插入和删除，适用于频繁插入/删除数据元素的情况。

【参考答案】 C

考点 2 线性表的顺序存储

温馨提示：本考点主要考查：1、线性表的顺序存储结构的类型定义；2、线性表在顺序存储结构上的算法实现，及相应的时间复杂度。线性表是统考和非统考高校编程题的命题点，请同学们多注意。

一．选择题部分

1. **（原书　第 2 题）**为了最快地对线性结构的数据进行某数据元素的读取操作，则其数据存储结构宜采用（　　）方式。

A. 顺序存储　　B. 链式存储　　C. 索引存储　　D. 散列存储

【解析】为了对线性结构的数据进行某数据的读写操作，采用顺序存储结构最为合适，因为随机存取的效率是最高的。我们一般认为，顺序存储结构指的是数组，如一维数组、二维数组等。以一维数组为例，只需要知道数组起始地址和元素在数组中的下标，就可以直接计算元素所在位置。

链式存储、索引存储和散列存储都达不到这么高的效率。

【参考答案】 A

2. **（原书　第 3 题）**在 n 个结点的线性表的数组实现中，算法的时间复杂度是 **O(1)**的操作是（　　）。

A. 访问第 i（1<=i<=n）个结点和求第 i 个结点的直接前驱（1<i<=n）

B. 在第 i（1<=i<=n）个结点后插入一个新结点

C. 删除第 i（1<=i<=n）个结点

D. 以上都不对

【解析】对顺序表的读取操作，时间复杂度为 O(1)，故而 A 答案正确。

在第 i 个结点之后插入一个新结点，必须从后向前移动第 n、n-1、…、i+1 位置的元素，才能腾出第 i+1 个位置来存放该新插入结点。所以，B 答案错误。

假设 i 是随机的，则在第 i 个位置之后插入一个新结点平均需要移动近一半表长的元素，时间复杂度为 O(n)。同理，删除第 i 个元素的时间复杂度也是 O(n)。所以，C 答案错误。

【参考答案】 A

3. **（原书　第 7 题）**在一个长度为 n 的顺序表中删除第 i 个元素，需要向前移动（　　）个元素。

A. n-i　　B. n-i+1　　C. n-i-1　　D. i+1

【解析】本题考查顺序表删除元素时表的元素移动情况。在一个长度为 n 的顺序表中删除第 i 个元素，第 i+1 个位置的元素移到第原第 i 个位置，第 i+2 个位置的元素移动到第 i+1 个位置，…，第 n 个位置的元素移动到第 n-1 个位置，共需移动 n-i 个元素。

【参考答案】 A

4. **(原书　第 8 题)**顺序表中，在任何一个位置插入元素的概率相等，则插入一个元素所需移动的元素平均数是（　　）。

A. (n-1)/2　　B. n　　C. n+1　　D. (n+1)/2

【解析】本题考查在顺序表中等概率删除一个元素平均需要移动的元素个数。长度为 n 的顺序表共有 n+1 个插入位置，如图 2.1 所示。

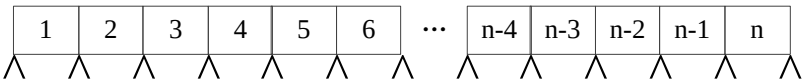


图 2.1

从表尾向表头方向开始计算，在第 n+1 个位置插入元素需要移动 0 个元素，在第 n 个位置插入元素需要移动 1 个元素，…，在第 1 个位置（位置从 1 开始算起）插入元素需要移动 n 个元素。故而，插入一个元素平均需要移动元素为

$$\text{Average} = \frac{\left(\sum_{i=0}^{n} i\right)}{n} = \frac{n(n+1)}{2} \times \left(\frac{1}{n}\right) = \frac{n+1}{2}$$

【参考答案】 D

5. **(原书　第 9 题)**在表长为 n 的顺序表中，当在任何位置删除一个元素的概率相同时，删除一个元素所需移动的平均个数为（　　）。

A. (n-1)/2　　B. n/2　　C. (n+1)/2　　D. n

【解析】若删除第 1 个位置的元素，第 2~n 位置的元素依次前移一个位置，共需要移动 n-1 次；若删除第二个位置的元素，第 3~n 位置的元素依次前移一个位置，共需要移动 n-2 次；…；若删除第 n-1 个位置的元素，第 n 位置的元素前移一个位置，共需要移动 1 次；若删除第 n 个位置的元素，则无需移动元素。故而，删除一个元素平均需要移动的元素个数为

$$\text{Average} = \frac{\left(\sum_{i=0}^{n-1} i\right)}{n} = \frac{n(n-1)}{2} \times \left(\frac{1}{n}\right) = \frac{n-1}{2}$$

【参考答案】 A

二．综合应用题部分

1. **(原书　第 2 题)**编写算法，将两个非递减有序顺序表 A 和 B 合并成一个新的非递减有序顺序表 C，已知顺序表 A 和 B 的元素个数分别为 m 和 n。

其中顺序表采用动态分配空间，定义如下：

```
typedef struct{
        ElemType *elem;    //存储空间基址
        int length;        //当前长度
        int listsize;      //当前分配的存储容量
 }SqList;
```

【解析】将两个非递减有序顺序表 A 和 B 合并成一个新的非递减有序的顺序表 C，可以用三个数组指针 i、j、k 分别指向 A、B、C 三个数组。数组中 A、B、C 的合并过程以及元素的变化如图 2.2 所示。

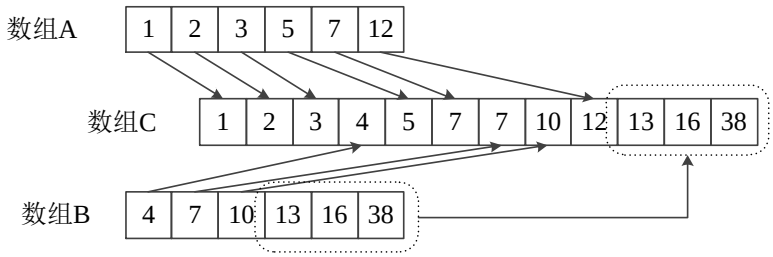


图 2.2

图 2.2 中，首先数组 A 的第 0 个元素与数组 B 的第 0 个元素相比较，发现 1 小于 4，将 1 加入顺序表 C，i++，k++；插入元素 2、3 皆是如此；当 i=4 时，数组 A 的元素 5 与数组 B 的元素 4 作比较，发现 5 大于 4，于是将 4 加入顺序表 C，j++，k++；如此比较下去，直到将 A 表中的元素全部插入到 C 表中。当 A 的元素已经全部插入到 C 表中，而另一个表 B 还有元素没有插入，则直接将表 B 剩余的元素插入到表 C 中。

根据上面的算法思想，算法如下：

```
void merge(SqList A, SqList B, SqList &C)
    {
        int i = 0, j = 0, k = 0;                    //A、B、C 三个数组指针 i、j、k
        while (i<A.length && j < B.length)          //退出循环的条件
            {
                if(A.elem[i] <= B.elem[j])          //如果(A.elem[i] <= B.elem[j]
                    {
                        C.elem[k] = A.elem[i]; //将 A.elem[i]接在数组 C 的 k 位置
                        i ++; k++;                  //数组 A 和数组 C 的数组指针加 1
                    }
```

```
            else                                    //如果(A.elem[i] > B.elem[j]
                {
                    C.elem[k] = B.elem[j]; //将 B.elem[j]接在数组 C 的 k 位置
                    j ++; k++;              //数组 B 和数组 C 的数组指针加 1
                }
        }
    while( i <A.length)                     //数组 A 还有元素但是 B 中所有元素
        {                                   //均在 C 中
            C.elem[k] = A.elem[i];          //依次将数组 A 中剩余元素接在 C 表表尾
            i ++; k++;
        }
    while( j <B.length)                     //数组 B 还有元素但是 A 中所有元素
        {                                   //均在 C 中
            C.elem[k] = B.elem[j];          //依次将数组 B 中剩余元素接在 C 表表尾
            j ++; k++;
        }
    C.length = k;                           //将 A、B 数组全部接完，表 C 长为原 A、C
}                                           //表长之和
```

2. （原书　第7题）线性表(a1,a2,a3,…,an)中元素递增有序且按顺序存储于计算机内。要求设计一算法完成：

(1). 用最少时间在表中查找数值为x 的元素。

(2). 若找到将其与后继元素位置相交换。

(3). 若找不到将其插入表中并使表中元素仍递增有序。

【解析】顺序存储的线性表递增有序，可以顺序查找，也可折半查找。题目要求“用最少的时间在表中查找数值为 x 的元素”，应使用折半查找方法。算法设计如下：

```
void  SearchExchangeInsert（ElemType a[ ]，ElemType x）
  /* a 是具有 n 个元素的递增有序线性表，顺序存储。本算法在表中查找数值为 x 的元素，如查到则与其后继交换位置；如查不到，则插入表中，且使表仍递增有序。*/
    {
      low=0; high=n-1;                  // low 和 high 指向线性表下界和上界的下标
```

```
    while（low<=high）
        {
            mid=（low+high）/2；    // 找中间位置
            if（a[mid]==x） break; // 找到 x，退出 while 循环。
            else if （a[mid] < x ）
                low=mid+1；          // 到中点 mid 的右半去查。
            else high=mid-1；        // 到中点 mid 的左部去查。
        }
     if（a[mid]==x && mid!=n）      // 若最后一个元素与 x 相等，则不存在
                                    //与其后继交换的操作。
        {
            t=a[mid];
            a[mid]=a[mid+1];
            a[mid+1]=t;
        }                           // 数值 x 与其后继元素位置交换
     if（low>high）                 // 查找失败，插入数据元素 x
        {
            for（i=n-1；i>high；i--）
                a[i+1]=a[i];        // 后移元素
            a[i+1]=x;               // 插入 x
        } //结束插入
}//结束算法
```

考点 3 线性表的链式存储

温馨提示：本考点主要考查：1、链接存储的概念；2、单链表、双链表和循环链表的相关概念和基本操作；3、线性表在链式存储上的算法实现，以及相应的时间复杂度。本考点在考研中，容易出现选择题和算法题，分值较大，请同学们务必掌握。

一．选择题部分

1. **(原书　第 2 题)**线性表采用链式存储时，结点的存储地址（　　）。

A. 必须是不连续的　　B. 连续与否均可

C. 必须是连续的　　D. 和头结点的存储地址相连续

【解析】本题考查链式存储结构的结点地址特点。线性表的链接存储结构称为单链表。单链表用一组任意的存储单元存放线性表的数据元素，这组存储单元可以连续，也可以不连续，甚至可以零散分布在内存中的任意位置。

【参考答案】 B

2. **(原书　第 5 题)**在一个长度为 n（n>1）的单链表上，设有头和尾两个指针，分别指向该链表的第一个结点和最后一个结点，则执行（　　）操作与链表的长度有关。

A. 删除单链表中的第一个元素

B. 删除单链表中的最后一个元素

C. 在单链表第一个元素前插入一个新元素

D. 在单链表最后一个元素后插入一个新元素

【解析】因为单链表具有头指针（设为 front），所以删除单链表的第一个元素的操作为

p=front; front=front->next; free(p);

但是删除最后一个结点（即尾指针所指向的结点）不能直接删除，需从表头开始查找尾结点的前驱结点，所以与表长有关。此外，在知道头尾指针的情况下，在单链表的第一个元素之前插入一个新元素，或在单链表的最后一个元素之后插入一个新元素，都是与表长无关的操作。

【参考答案】 B

3. **(原书　第 7 题)**在循环双链表的 p 所指的结点之前插入 s 所指结点的操作是(　　)。

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

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

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

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

【解析】本题考查在循环双链表中插入结点的正确操作。在循环双链表的 p 所指的结点之前插入指针 s 所指结点，如图 2.3 所示。

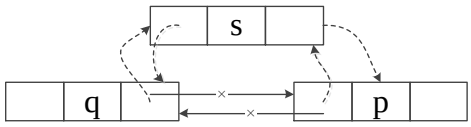


图 2.3

在插入结点时，很多同学很头痛，一不小心就错了。我们总结出了一个方法：那就是

(1). 先解决待插入结点 s 的两个指针域（s 的两个指针域的调整，可以交换顺序）；

(2). 再解决 s 结点的前驱结点 p 和后继结点 q 的指针域（p 和 q 的指针域调整，也可以交换顺序，但是当题目只告知 p 和 q 当中的一个时，另作考虑。如本题只告诉 p，则只能先让 p 的前驱结点的后继指针指向 s，再调整 p 的前驱指针指向 s）。

根据这两个指针调整步骤，我们先调整结点 s 的指针：

$$s->next = p; s->prior = p->prior;$$

再调整 p 的指针：

$$p->prior->next = s; p->prior = s;$$

【参考答案】 D

4. **(原书　第 18 题)** 在一个以 h 为头指针的双向循环链表中，指针 p 所指的元素是尾元素的条件是（　　）。

A. p==h　　B. h->rlink==p

C. p->llink==h　　D. p->rlink==h

【解析】设头指针 h 指向双循环链表的第一个结点，如图 2.4 所示。

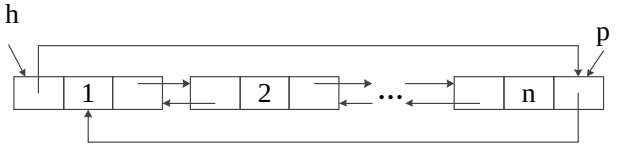


图 2.4

如上图所示，若 p 所指向的元素是尾元素，则 p->rlink=head，故而选择 D 答案。

【参考答案】 D

5. **(原书　第 20 题)** 已知指针 p 和 q 分别指向某单链表中第一个结点和最后一个结点。假设指针 s 指向另一个单链表中某个结点，则在 s 所指结点之后插入上述链表应执行的语句为（　　）。

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

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

【解析】 在指针 s 指向的某一个结点之后插入一段以 p 和 q 分别指向第一个结点和最后一个结点的链表，只需要令 q 的指针域指向 s 的后继结点（q->next=s->next），再调整 s 的指针域，使其指向新的后继结点 p 即可(s->next=p)，如图 2.5 所示。

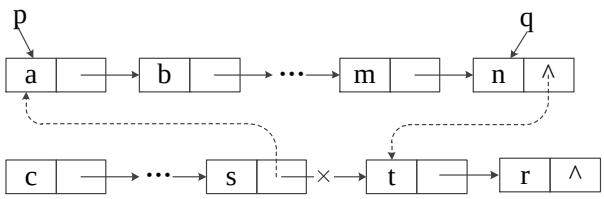


图 2.5

【参考答案】 A

6. （原书　第 25 题）在双向链表存储结构中，删除 p 所指的结点时须修改指针（　　）。

A. p->prior->next=p->next;　p->next->prior=p->prior;

B. p->prior=p->prior->prior;　p->prior->next=p;

C. p->next->prior=p;　p->next=p->next->next

D. p->next=p->prior->prior;　p->prior=p->next->next;

【解析】本题考查双向链表中删除结点的操作方法。为了便于大家理解，我们随便画一个满足题意的简单的双向链表，如图 2.6 所示。

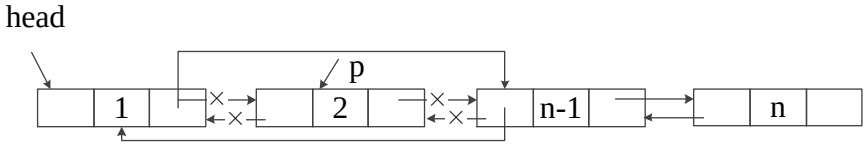


图 2.6

要删除 p 结点，必须先接好 p 的前驱和后继的指针，**保证链表不断**。其实，先接 p 的前驱结点的后继指针或者先接 p 的后继结点的前驱指针，都是可以的。先接 p 的前驱结点的后继指针再接 p 结点的后继结点的前驱指针则为

p−> prior−> next = p−> next; p−> next → prior = p−> prior;

先接 p 的后继结点的前驱指针再接 p 结点的前驱结点的后继指针则把上面的两条语句反过来。

【参考答案】 A

二．综合应用题部分

1. **（原书　第 8 题）**已知线性表中的元素以值递增有序排列，并以单链表作存储结构。试设计一个高效的算法，删除表中所有值大于 mink 且小于 maxk 的元素（若表中存在这样的元素），同时释放被删结点空间（注意：mink 和 maxk 是给定的两个参变量。它们的值可以和表中的元素相同，也可以不同）。

【解析】如图 2.7 所示，我们画了一种简单的情况，删除 mink=3，maxk=99 之间的元素，只需找到第一个不大于 mink 的元素结点，令 p 指向该结点，再继续搜索链表，找到第一个不小于 maxk 的元素结点，令 q 指针指向它。那么，删除所有大于 mink 小于 maxk 的元素结点只需令 p->next=q 即可。

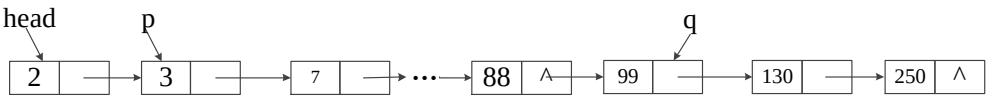


图 2.7

算法思想如下：

(1). 查找最后一个不大于 mink 的元素结点，由指针 p 指向；

(2). 如果还有比 mink 更大的元素，查找第一个不小于 maxk 的元素，由指针 q 指向；

(3). p->next=q，即删除表中所有值大于 mink 且小于 maxk 的元素。

算法实现如下：

```
void    Delete_Between ( node *head, int mink, int maxk )
  {
       node *p=head, *q;
       while(p->next->data<=mink) p=p->next;      /* p 是最后一个不大于 mink 的元素 */
       if(p->next)                                /*  如果还有比 mink 更大的元素 */
           {
                 q=p->next;
                 while(q->data<maxk) q=q->next;   /* q 是第一个不小于 maxk 的元素 */
                  p->next=q;
            }
  }/*Delete_Between */
```

2. **（原书　第 10 题）**设有一个循环双链表，其中有一结点的指针为 p，编写一个函数 p 与其右边的一个结点进行交换。结点结构为

left	data	right

【解析】本题的算法其实是一个很简单的算法，首先令一个指针 q 指向 p 的后继结点，若 q 为空，则交换失败，否则 p 与 q 所指向的结点的值交换。我们通过图 2.8，来看指针的调整过程。

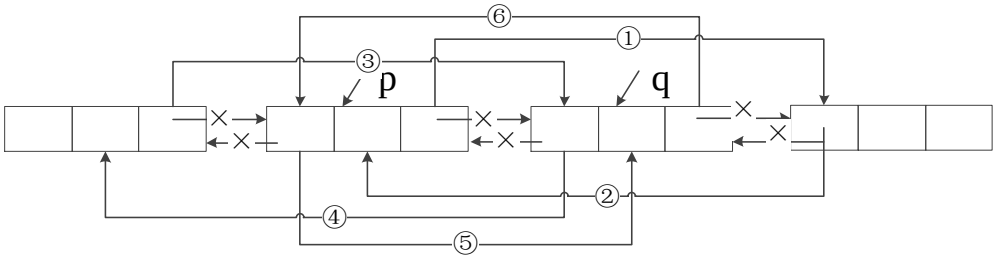


图 2.8

如图 2.13 所示，要交换 p 和 q 所指向的结点位置，可以通过以下的交换顺序实现：

① 令 p 结点的后继指针域指向 q 结点的后继结点；

② 令 q 结点的后继结点的前驱指针域指向 p 结点，到此完成指针 p 结点与 q 结点的后继结点的衔接；

③令 p 结点的前驱结点的后继指针指向 q 结点；

④令 q 结点的前驱指针指向 p 结点的前驱结点，到此完成将 q 结点接在原 p 结点的前驱结点后面的操作；

交换了 p 和 q 结点位置之后，q 结点成了 p 结点的前驱。接下来，重新接 p 和 q 之间的指针关系：

⑤令 p 结点的前驱指针指向 q 结点；

⑥令 q 结点的后继指针指向 p 结点；

到此，完成结点位置的交换。算法如下：

```
void swap(dnode *p)
      {
        dnode * q;
```

```
    q=p->right;
    if (q==NULL )                    //p 没有后继结点，交换失败
        return ;
    else                             //p 存在后继，则 p 与其后继交换
     {
        p->right=q->right;           //(1)
        p->right->left=p;            //(2)
        p->left->right=q;            //(3)
        q->left=p->left;             //(4)
        p->left=q;                   //(5)
        q->right=p;                  //(6)
      }
}
```

3. **（原书　第 15 题）**在一个非递减有序的线性表中，有数值相同的元素存在。若存储方式为单链表，设计算法去掉数值相同的元素，使表中不再有重复的元素。例如：（7，10，10，21，30，42，42，42，51，70）将变作（7，10，21，30，42，51，70），分析算法的时间复杂度。

【解析】算法思想：非递减有序表中的某一个元素的重复元素，分布在有序表中该元素位置的附近，可以从头到尾扫描单链表，若当前结点的元素值与后继结点的元素值不相等，则指针后移；否则删除该后继结点。

实现该算法需要两个工作指针：设 p 为工作指针，pre 为 p 所指向的结点的前驱结点的指针。设链表中至少有一个结点，算法描述如下：

```
LinkedList DeleteSameElement（LinkedList L）
  {                                  //L 为递增有序的单链表
     pre=L->next;                    //pre 指针指向第一个元素结点
     p=pre->next;                    //p 指向 pre 所指结点的后继结点
     while ( p !=null )              //如果 p 指针不为空
        if（p->data==pre->data）  // p 所指结点的值等于 pre 所指结点的值
            {
               u=p;                  //用指针 u 暂存 p 指针所指向的结点
```

```
                p=p->next;      //p 指针后移
                free (u);       //释放掉 u 指针所指结点的内存空间
            }
        else  {                 //p 和 pre 两指针所指向的结点值不相等
                pre=p;          //pre 指针后移
                p=p->next;      //p 指针后移
            }
    }
```

下面我们给出了一系列图来模拟删除过程，请同学们仔细体会！

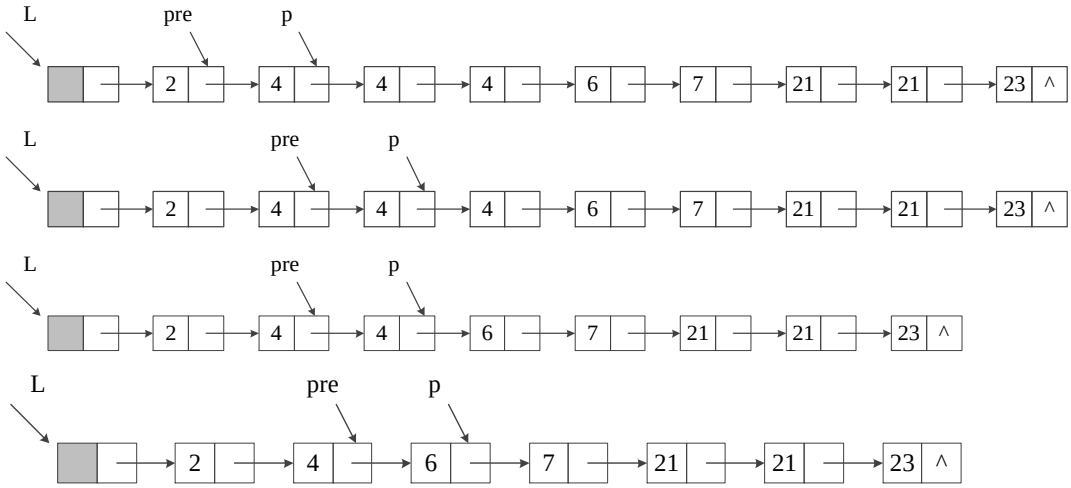


图 2.9

在图 2.9 中，我们展示了删除元素值为 4 的重复元素的过程，pre 一直都在第一个元素值为 4 的结点，p 一直指向 pre 的后继结点，若二者值相等，则删除之。

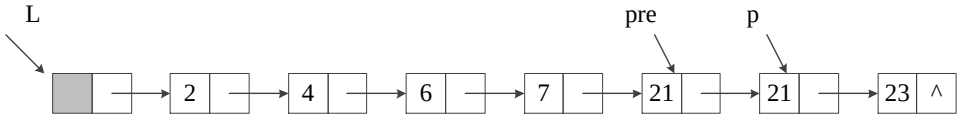


图 2.10

删除了重复的 4 之后， p 和 pre 指针顺着链表一直走到图 2.10 所示的位置，发现 pre 和 p 两个指针所指向的结点的值再一次相等，用 u 暂存 p 所指向的结点的指针，p 后移，释放掉 u 指针所指向结点的内存空间。最后得到的链表如图 2.11 所示。

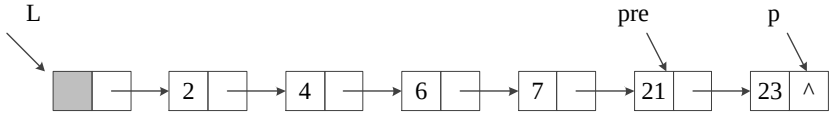


图 2.11

本章到此就结束了，请问您有什么疑问吗？任何问题，欢迎您与我们作者进行交流！

shareOurDreams
梦享团队微信号

weCSdream
梦享团队官方微信公众号

梦享论坛团队
梦享团队新浪微博

因为有你，所以有梦享！

梦享团队祝愿每一个考研人梦想成真！

第三章　栈、队列和数组

考点 1 栈和队列的概念和基本操作

温馨提示：本部分考查栈和队列的概念，以及基本操作。请同学们注意栈和队列的性质，并掌握二者的基本操作。

一．选择题部分

1. （原书　第 1 题）设有一个栈，元素依次进栈的顺序为 A、B、C、D、E。下列（　　）是不可能的出栈序列。

A. A,B,C,D,E　　B. B,C,D,E,A

C. E,A,B,C,D　　D. E,D,C,B,A

【考查内容】查栈的基本操作。

【解析】对于 A 答案，A 元素进栈出栈、B 元素进栈出栈、C 元素进栈出栈、D 元素进栈出栈、E 元素进栈出栈，即可得到出栈序列 A、B、C、D、E。

对于 B 答案，将 A 压入栈底，然后 B 元素进栈出栈、C 元素进栈出栈、D 元素进栈出栈、E 元素进栈出栈，即可得到出栈序列 B、C、D、E、A。

对于 D 答案，将 A、B、C、D、E 五个元素依次压栈，再出栈，即可得到序列 E、D、C、B、A。

对于 C 答案，要保证 E 先出栈，那么 A、B、C、D 四个元素只能压栈而不能出栈。也就是说，若出栈序列以 E 开头，则只有一个序列 E、D、C、B、A，故而该答案错误。

【参考答案】 C

2. （原书　第 5 题）若一个栈的输入序列为 1，2，3，…，n，输出序列的第一个元素是 i，则第 i 个输出的元素是（　　）。

A. i-j-1　　B. i-j　　C. n-i+1　　D. 不确定的

【考查内容】栈的性质。

【解析】当一个栈的输入序列为 1，2，3，…，n，输出序列的第一个元素是 i，那么，其输出序列不像第一个输出元素为 n 那样只有一种结果。因为输出序列的不唯一，所以难以判断第 i 个输出的元素是什么。

【参考答案】 D

3. **(原书　第 9 题)** 设栈 S 和队列 Q 的初始状态为空，元素 a,b,c,d,e,f 依次通过栈 S，一个元素出栈后即进队列 Q，若 6 个元素出队的序列是 a,c,f,e,d,b，则栈 S 的容量至少应该是（　　）。

A. 6　　B. 5　　C. 4　　D. 3

【考查内容】栈的基本操作。

【解析】本题其实我们可以看成只在栈 S 中操作的现象，因为队列 Q 并不影响出栈的顺序。6 个元素的出队列顺序为 a、c、f、e、d、b，即出栈顺序为 a、c、f、e、d、b。

注意题目的条件：元素 a、b、c、d、e、f 依次通过栈 S。我们可以通过以下的栈操作得到这个序列：

(1). PUSH(a)、POP(a)可以使得元素 a 第一个出栈；

(2). PUSH(b)、PUSH(c)、POP(c)，使得 c 成为第二个出栈的元素；

(3). 依次 PUSH(d)、PUSH(e)、PUSH(f)，再 POP(f) 、POP(e)、POP(d)、 POP(b)。

最后，得到了 a、c、f、e、d、b 出栈序列。故而，栈的大小至少为 4。

【参考答案】 C

4. **(原书　第 11 题)** 由两个栈共享一个向量空间的好处是：（　　）。

A. 减少存取时间，降低下溢发生的机率

B. 节省存储空间，降低上溢发生的机率

C. 减少存取时间，降低上溢发生的机率

D. 节省存储空间，降低下溢发生的机率

【考查内容】共享栈的优点。

【解析】**两栈共享空间，指的是使用一个数组来存储两个栈，让一个栈的栈底为该数组的始端，另一个栈的栈底为该数组的末端，两个栈从各自的栈底向中间延伸。**假设数组 A 的大小为 s，我们可以构建**共享栈**如图 3.1 所示。

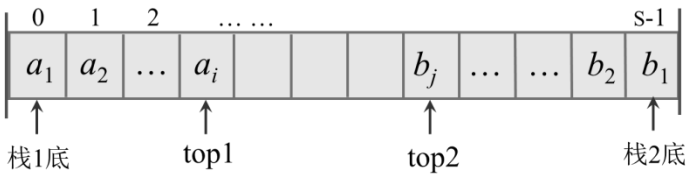


图 3.1 共享栈

其中，栈 1 的栈底固定在下标为 0 的一端，栈 2 的栈底固定在下标为 StackSize-1 即 s-1 的一端，而 top1 和 top2 分别为栈 1 和栈 2 的栈顶指针。不难发现，栈的大小为整个数组的空间。

栈 1 为空的条件是 top1=0，栈 2 为空的条件是 top2=StackSize-1，本题即 s-1，而栈满地条件则为 top2=top1+1。

从图 3.4 可以看出，共享栈更加有效地利用存储空间。而且两个栈可以相互调节，只有当 top2=top1+1 时，元素占满了数组的全部空间，栈才发生上溢。因而，共享栈降低了上溢发生的概率。但是栈的存取都是 O(1)的时间复杂度，并没有减少存取时间。

【参考答案】 B

5. **(原书　第 14 题)**若以 1234 作为双端队列的输入序列，则既不能由输入受限双端队列得到，也不能由输出受限双端队列得到的输出序列是（　　）。

A. 1234　　B. 4132　　C. 4231　　D. 4213

【考查内容】双端队列。

【解析】**双端队列**是限定插入和删除操作在表的两端进行的线性表，如图 3.2 所示。

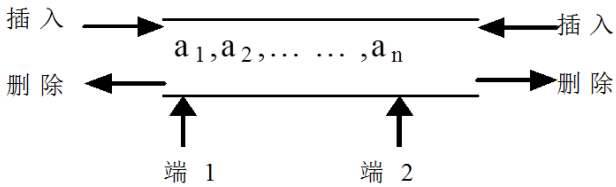


图 3.2

输出受限的双端队列，是指队列的一端允许插入和删除，另一端只允许插入（即有一端的输出受到限制）；**输入受限的双端队列**，是指队列的一端允许插入和删除，另一端只允许删除（即有一端的输入受到限制）。

对于 A 答案的 1234，只需从端 1 输入，从端 2 取出即可。这种情况下，限制端 1 为输出受限，限制端 2 为输入受限仍可以得到这个序列。

对于 B 答案，可以从端 1 依次插入 1、2、3、4，再从端 2 取 4，从端 1 取 1，从端 2 取 3，从端 1（或端 2）取 2，读出队列元素 4132。 故而，输出序列 4、1、3、2 能由输入受限（端 2 输入受限）的双端队列得到，但不能由输出受限的双端队列得到。

对于 D 答案，只需从端 1 插入元素 1、2，再从端 2 插入元素 3，再从端 1 插入元素 4，最后从端 1 读出序列 4213。

C 答案的序列不管是输入限制还是输出限制，都无法得到。

【参考答案】 C

考点 2 栈的顺序存储结构

温馨提示：本考点考查栈的顺序存储结构，请同学们注意顺序栈的特点，并注意顺序栈判空、判满、入栈和出栈的方法。

一．选择题部分

1. **（原书　第 1 题）**判定一个顺序栈 st（最多元素为 MaxSize）为空的条件是（　　）。

A. st->top != -1　　B. st->top == -1

C. st->top != MaxSize-1　　D. st->top == MaxSize-1

【考查内容】顺序栈的判空条件。

【解析】**顺序栈**，即栈的顺序存储结构。顺序栈利用一组地址连续的存储单元依次存放自栈底到栈顶的数据元素，同时附设指针 top 指示栈顶元素在顺序表中的位置，如图 3.3 所示。

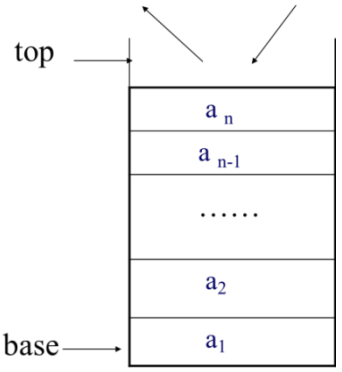


图 3.3 顺序栈

我们称 top 为栈顶指针，其初始值指向栈底，即 top=base，可作为栈空的标记。每当插入新的栈顶元素时指针 top 增 1，而删除栈顶元素时指针 top 减 1，非空栈中的栈顶指针始终在栈顶元素的下一位置上（**本书默认采用这种方案，在以后的相关题目中，我们不再赘述**）。

有的书则不表示栈底，只用一个栈顶指针来表示栈的情况，并把 top 指针指向栈顶元素。数组的下标从 0 开始表示时，st->top==0 表示栈中有一个元素，而栈空则表示为 st->top==-1。

【参考答案】 B

2. **（原书　第 3 题）**若栈采用顺序存储方式存储，现两栈共享空间 V[1...m]，top[1]、top[2]分别代表第 1 和第 2 个栈的栈顶，栈 1 的底在 V[1]，栈 2 的底在 V[m]，则栈满的条件是（　　）。

A. |top[2]-top[1]|=0　　B. top[1]+1=top[2]

C. top[1]+top[2]=m　　D. top[1]=top[2]

【考查内容】共享栈的判满条件。

【解析】在上一个考点中，我们谈到了共享栈的结构如图 3.4 所示。

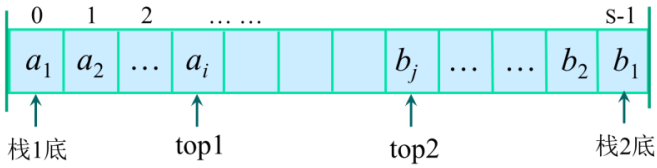


图 3.4

其中，栈 1 的栈底固定在下标为 0 的一端，栈 2 的栈底固定在下标为 StackSize-1 即 s-1 的一端，而 top1 和 top2 分别为栈 1 和栈 2 的栈顶指针，栈满地条件则为 top2=top1+1。

从上图可以看出，共享栈的建立，更加有效地利用存储空间。而且两个栈可以相互调节，只有当 top2=top1+1 时，元素占满了数组的全部空间，栈才发生上溢。

本题中，栈 1 的栈顶为 top[1]，栈 2 的栈顶为 top[2]，故而栈满的条件为 top[2]=top[1]+1，选择 B 答案。

【参考答案】 B

3. （原书　第 8 题）假定利用数组 a[n]顺序存储一个栈，用 top 表示栈顶指针，用 top==n+l 表示栈空，该数组所能存储的栈的最大长度为 n，则表示栈满的条件是（　　）。

A. top == -1　　B. top == 0　　C. top > l　　D. top == 1

【考查内容】栈的判满条件。

【解析】根据题意，栈顶在数组下标小的一端，栈底在数组下标大的一端。因为 top=n+1 表示栈空，那么 top=n 表示栈中有一个元素，top 指针指向栈顶位置。因为栈的长度为 n，故而可以存放 n 个元素，栈满的时候，top==1。同学们要小心，要是选择 top=0，则表中可存放 n+1 个元素，显然是不对的。

【参考答案】 D

二．综合应用题部分

1. （原书　第 5 题）一个双向栈 S 是在同一向量空间内实现的两个栈，它们的栈底分别设在向量空间的两端。 试为此双向栈设计初始化 InitStack (S) 、入栈 Push(S , i , x) 和出栈 Pop(S , i)等算法， 其中 i 为 0 或 1，用以表示栈号。

【解析】关于共享栈，我们已经说了很多了。共享栈的原理如图 3.5 所示。

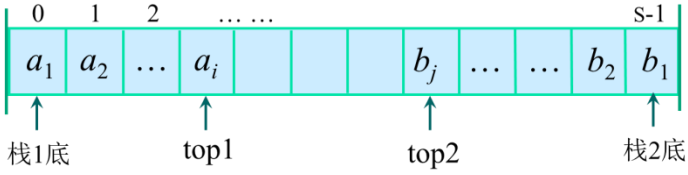


图 3.5

其中，栈 1 的栈底固定在下标为 0 的一端，栈 2 的栈底固定在下标为 StackSize-1 即 s-1 的一端，而 top1 和 top2 分别为栈 1 和栈 2 的栈顶指针，栈满地条件则为 top2=top1+1。那么，本题我们来看看共享栈的算法。

```
#define StackSize 100 // 假定分配了 100 个元素的向量空间
#define char DataType
```

```
typedef struct{
        DataType Data[StackSize];
        int top0; //需设两个指针
        int top1;
}DblStack

//初始化双向栈，设栈的空间下表为 0~StackSize-1
void InitStack( DblStack    *S )
    {
      S.top0 = -1;            //栈 1 的初始栈顶为-1
      S.top1 = StackSize; //栈 2 的初始栈底为 StackSize
    }

//判栈空(栈号  i)，初始化的双向栈是典型的栈空情况
int EmptyStack( DblStack *S, int i )
    {
          return (i == 0 && S.top0 == -1|| i == 1 && S.top1== StackSize) ;
    }

//判栈满,满时肯定两头相遇，此时正如我们解题过程中所示的 top1=top0+1
int FullStack( DblStack *S)
    {
      return (S.top1 == S.top0+1);
    }

//进栈(栈号 i)
void Push(DblStack *S, int i, DataType x)
    {
      if (FullStack( S ))                    //入栈前先判满
        Error("Stack overflow");             //上溢，退出运行
      if ( i == 0) S.Data[ ++ S.top0]= x;   //栈 0 入栈
      if ( i == 1) S.Data[ -- S.top1]= x;    //  栈 1 入栈
```

```
    }

//出栈(栈号 i)
  DataType Pop(DblStack *S, int i)
    {
      if (EmptyStack ( S,i) )                    //出栈判空
        Error("Stack underflow");                //下溢退出
      if( i==0 )
        return ( S.Data[ S.top0--] );          //返回栈 0 的栈顶元素，指针值减 1
      if( i==1 )
        return ( S.Data[ S.top1++] );          //栈 1 以数组下标大端为底，所以指针值加 1
    }
```

考点 3 队列的顺序存储结构

温馨提示：本考点考查队列的顺序存储方式，请同学们注意队列在该结构下的判空、判满、入队列和出队列的基本操作，并能把这些基础知识用于解题。

一．选择题部分

1. **（原书　第 2 题）**若用一个大小为 6 的数组来实现循环队列，且当前 rear 和 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

【考查内容】顺序队列的基本操作。

【解析】根据题意，循环队列 Q 是使用大小为 6 的数组 Q 来实现的。队空时，Q.fornt=Q.rear。若队列不为空，则 front 指向当前的队头元素，而 rear 指向当前队尾的下一个位置。当队头指针指向 3，而队尾指针指向 0 的时候，队头和队尾指针的位置如图 3.6 所示。队列占用的位置为 3~5 的位置，尚未占用的是 0~2 的位置。

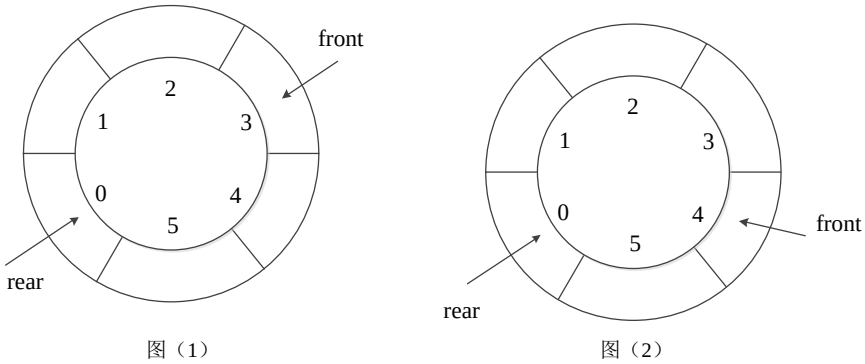


图 3.6

从队列中删除一个元素，Q.front=(Q.front+1)%MaxSize=（3+1）%6=4，front 指针指向数组下标为 4 的位置。此时，队列占用的位置为 4~5，没有占用的位置为 0~3，如图（2）所示。

当有两个元素入队列时，依次执行两次 Q.rear=（Q.rear+1）% Maxsize，其中 Maxsize=6，找到的位置依次为 1 和 2。这两个元素入队列之后，队列的变化分别如下图 3.11 和图 3.12 所示。

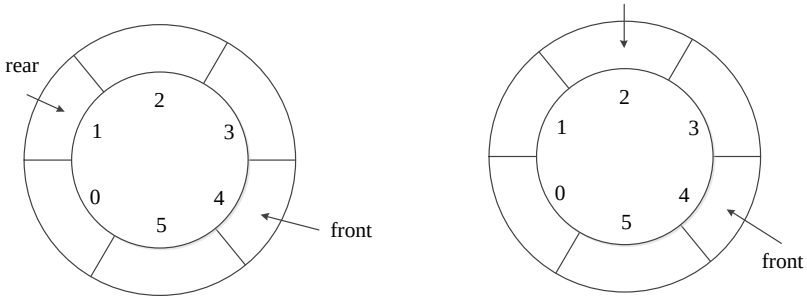


图 3.11　　**图 3.12**

从图 3.12 可以看出，此时 front 指针的值为 4，而 rear 指针的值为 2。所以，选择 B 答案。

【参考答案】 B

2. **（原书　第 4 题）**设数组 Data[0..m]作为循环队列 SQ 的存储空间，front 为队头指针，rear 为队尾指针。其中，front 指针指向队头元素，rear 指针指向队尾元素的下一个位置。则执行出队操作的语句为（　　）。

A. front=(front+1)%(m+1)　　B. front=(front+1)% m

C. rear=(rear+1)% m　　D. front=front+1

【考查内容】顺序队列的基本操作。

【解析】数组为 Data[0~m]，故而数组的大小为 m+1，元素出队列或者入队列时应该进行mod (m + 1)运算而不是mod m运算。出队列时，front = (front + 1)mod $(m + 1)$。

【参考答案】 A

3. （原书　第 7 题）若顺序存储的循环队列的 QueueMaxSize=n，则该队列最多可存储（　　）个元素。

A. n　　B. n-1　　C. n+1　　D. 不确定

【考查内容】顺序队列的性质。

【解析】循环队列 Q 中，若不留一个位置，怎么判断Q. front = Q. rear是队空还是队满呢？显然无法区别。因为队空的时候，显然满足Q. front = Q. rear；而队满的时候，若不留下一个空位置，rear 指向队尾的下一个位置，该位置刚好是 front 所指向的位置，也满足Q. front = Q. rear。

我们若给数组留一个空位置来区分队满和队空。那么，可用Q. front = Q. rear表示当前队列为空。当有元素需要入队时，先把元素放入 rear 指针值所指向的数组位置（数组下标），使得该元素成为队列的新队尾元素，接下来对 rear 执行Q. rear = (Q. rear + 1) % $Maxsize$操作，指向新的队尾元素的下一个位置，(Q. rear + 1)% $Maxsize == Q.front$时表示队满。

【参考答案】 B

二．综合应用题部分

1. （原书　第 1 题）假设循环队列定义为：

```
#define maxqsize 101    /*设队列的最大长度为 100*/
typedef struct sqque{
    elementype *base; /*  初始化的动态分配存储空间  */
    int length; /*  队列长度  */
    int rear ; /*  队尾指针*/
  }SqQueue;
```

(1). 写出判断队列满和空的条件；

(2). 写出插入和删除一个元素的算法的关键语句。

【解析】本题考查循环队列的基本操作。

(1). 队列为满为条件为 q.length==maxqsize，队列为空为条件为 q.length==0 。为什么队满的条件不是$q.length == maxqsize - 1$? 请同学们要注意了，我们在讲述循环队列时，说循环队列需要留一个位置，来区别栈满和栈空。但是本题已经有了 length 来表明当前的队列长度，即使队尾已经追上队头，也能用 length=maxsize 来表明栈满。故而，不需要预留空间来区分栈空和栈满。

(2). 插入元素之前，要先判断队列是不是队满。若队不满，则把新元素入队列，队列的长度加 1。我们假设队头指针指向队头位置，队尾指针指向队尾的下一个位置，那么插入元素的操作如下：

```
bool EnQueue(SqQueue &q, ElemType x)
    {
        if(q.length==maxqsize) return ERROR;
        q.base[q.rear]=x;
        q.rear=(q.rear+1)%maxqsize;
        q.length++;
        return OK;
    }
```

在删除元素之前，先看看队列是否已空，判断的条件只需要看看$q.length == 0$是否成立，若成立则队列为空，否则不为空。在队列不为空的情况下，删除元素的操作如下：

```
bool EnQueue(SqQueue &q, ElemType x)
    {
        if(q.length==0) return ERROR;
        head=(q.rear-q.length+1)%maxqsize;
        x=q.base[head];
        q.length--;
        return OK;
     }
```

考点 4 栈的链式存储结构

温馨提示：栈的链式存储是栈的一种存储结构，同学们注意区分顺序栈和链栈的存储特点，并会链栈的判空、判满、出栈和入栈等基本操作。本部分命制选择题的可能性很大，我们不在本考点分布大题。

一．选择题部分

1. **（原书　第 1 题）**向一个栈顶指针为 h 的带头结点的链栈中插入指针 s 所指的结点时，应执行（　　）操作。

A. h->next=s ;　　B. s->next=h ;

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

【解析】**链栈**一般是没有附加头结点的运算受限的单链表，栈顶指针就是链表的头指针，如图 3.7 所示。

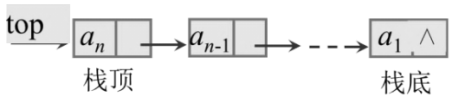


图 3.7

其实链栈的栈顶设在链表的头部而不设在尾部，是为了减少出栈的开销的。栈的性质决定了只能在栈顶一端操作，栈顶指针设在链表头部，方便栈操作。链栈有一个优点，那就是链栈中的结点是动态分配的，不会出现上溢的情况。

因为本题的链栈带有头结点，并令 h 指向头结点，该头结点的下一个结点，才是栈顶。故而插入 s 所指向的结点，首先要把 s->next=h->next，再令 h->next=s 即可完成插入新元素作为栈顶并调整 h 头结点的下一个结点为新栈顶操作。

再假设，若本题的链栈，是不带有头结点的链栈，栈顶指针仍为 h，则向栈中插入指针 s 所指向的结点时，只需将 s->next=h，再令 h=s 即可完成插入结点 s 并令 h 指向新的栈顶操作。

【参考答案】 D

2. **（原书　第 5 题）**最不适合用作链栈的链表是（　　）。

A. 只有表头指针没有表尾指针的循环双链表

B. 只有表尾指针没有表头指针的循环双链表

C. 只有表尾指针没有表头指针的循环单链表

D.　只有表头指针没有表尾指针的循环单链表

【解析】只有表头指针但是没有表尾指针的循环双链表，设每一个结点的左（Llink）右(Rlink)指针分别指向其前驱和后继，令 front 指向栈顶，如图 3.8 所示。

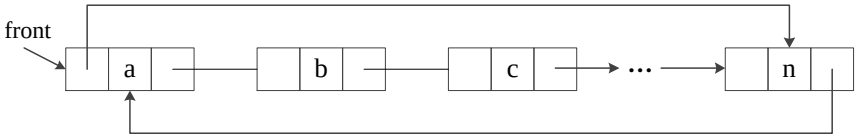


图 3.8

若使用只有表头指针但是没有表尾指针的循环双链表作为链栈，比如利用表头一端作为栈顶，那么新结点 s 的入栈操作可分以下步骤进行：

(1). 首先，调整结点 s 的指针，使得 s 的左指针指向结点 n，即 front->Llink，因为此时的 s 已经是新的栈顶元素。接下来再令 s 的右指针指向次栈顶元素，调整如下：

$$s->Rlink = head;\ \ s->Llink = front->Llink;$$

(2). 接下来，调整结点 n，使其右指针指向结点 s，即

$$h->Llink->next = s$$

(3). 接下来，需要调整 front 结点的指针，使得 front 的左指针域指向新的栈顶，即

$$front->Llink = s;$$

到此，操作已经完成。如图 3.9 所示。

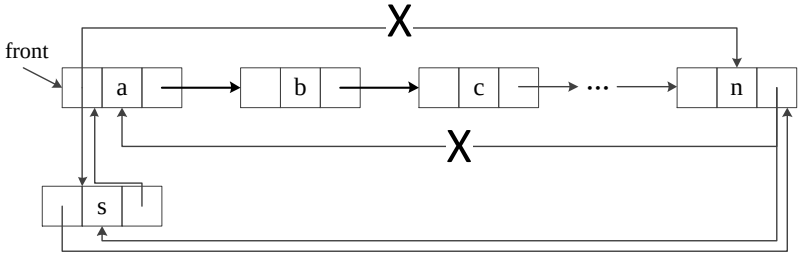


图 3.9

对于 B 答案，只有表尾指针没有表头指针的循环双链表，如图 3.10 所示。

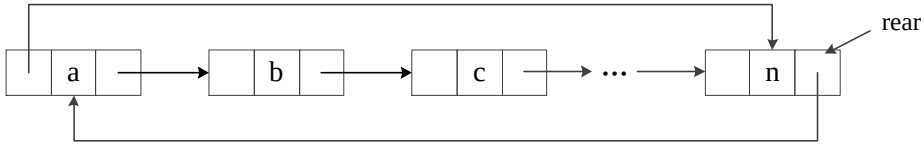


图 3.10

我们令 rear 指向表尾结点，仍然令链表的表头作为栈顶，则一个新元素入栈时只需调整指针如下：

$$s->Llink = rear;$$

$$rear->Rlink->Llink = s;$$

$$rear->Rlink = s;$$

调整如图 3.11 所示。

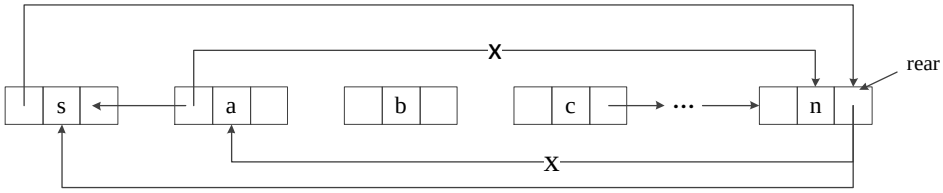


图 3.11

对于 C 答案，当循环单链表只有表尾指针没有表头指针时，如图 3.12 所示。

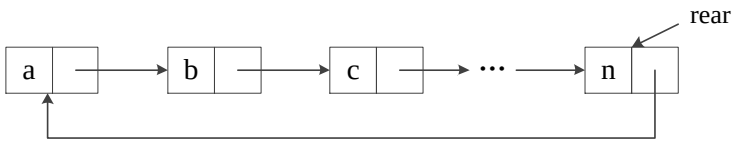


图 3.12

同样令链表表头为栈顶，插入 s 作为新的栈顶，则需令

$$s->next = rear->next;\ rear->next = s;$$

对于 D 答案，当只有头指针时，同样令链表的表头为栈顶，需要从 front 一直找到第 n 个结点，令第 n 个结点的指针域指向结点 s，再调整 s 的指针域指向 front，最后再令 front 指向新的栈顶 s，如图 3.13 所示。

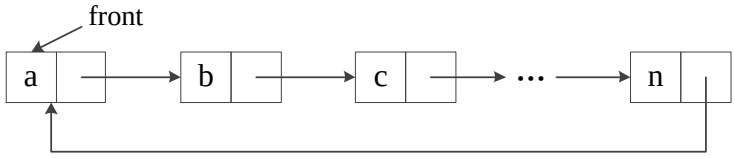


图 3.13

虽然 D 答案也可行，但每次压栈，都需要遍历一下链表，时间复杂度为 O(n)。除了 D 答案，其他答案的方法能使得每次压栈的时间复杂度为 O(1)，故而 D 最不适合用作链栈。

【参考答案】 D

考点 5 队列的链式存储结构

温馨提示：和栈的链式存储结构一样，队列也可以用链式存储结构来存储表示，也请同学们多注意该存储结构下，队列的基本操作。

一．选择题部分

1. **（原书　第 1 题）**用链接方式存储的队列，在进行插入运算时（　　）。

A. 仅修改头指针　　B. 头、尾指针都要修改

C. 仅修改尾指针　　D. 头、尾指针可能都要修改

【解析】队列的链式存储结构简称为链队列，它是限制仅在表头删除和表尾插入的单链表。显然仅有单链表的头指针不便于在表尾做插入操作，为此再增加一个尾指针，指向链表的最后一个结点，如图 3.14 所示。

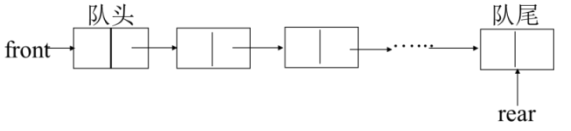


图 3.14

就一般情况而言，进行插入运算时，只需要考虑队尾元素。但是当链队列为空队列时，插入结点需要同时修改头尾指针。故而，选择 D 答案。

【参考答案】 D

2. **（原书　第 3 题）**用循环链表表示队列，设队列的长度为 n，若只设尾指针，则出队和入队的时间复杂度分别为（　　）。

A. O(1)，O(1)　　B. B. O(1)，O(n)

C. O(n)，O(1)　　D. O(n)，O(n)

【解析】用循环链表表示队列，如图 3.15 所示。

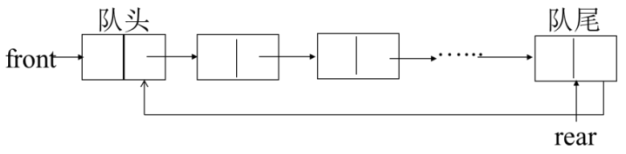


图 3.15

从上图可知，要出队列，只需要用 rear->next=front->next 即可，时间复杂度为 O(1)。要入队列，假如要入队列的结点为 p，只需用 r->next=p（将结点 p 接在队尾后面），r=p(令 p 为新的队尾)两个语句即可。显然，入队列的时间复杂度也是 O(1)。

【参考答案】 A

3. **(原书　第 5 题)**设指针变量 front 表示链式队列的队头指针，指针变量 rear 表示链式队列的队尾指针，指针变量 s 指向将要入队列的结点 X，则入队列的操作序列为(　　)。

A. front->next=s；front=s；　　B. s->next=rear；rear=s；

C. rear->next=s；rear=s；　　D. s->next=front；front=s；

【解析】指针变量 front 表示链式队列的队头指针，指针变量 rear 表示链式队列的队尾指针，指针变量 s 指向将要入队列的结点 X。入队列操作只需将该结点 X 接在 rear 所指向的结点后面，然后调整 rear 指针指向新的结点 s 即可，即

rear->next=s; rear=s；

故而，选择 C 答案。

【参考答案】 C

考点 6 栈和队列的应用

温馨提示，本考点在近些年的真题中，出现了：1、栈在括号匹配中的应用；2、栈在表达式求值中的应用；3、栈在递归中的应用；4、队列在层次遍历中的应用；5、队列在计算机系统中的应用（408 统考）。请同学们多关注这些方面的内容。

一. 选择题部分

1. **(原书　第 1 题)**将一个递归算法改为对应的非递归算法时，通常需要使用（　　）。

A. 栈　　B. 队列　　C. 循环队列　　D. 优先队列

【解析】**递归算法**是一种分而治之的、把复杂问题分解为简单问题的求解方法。对求解某些复杂问题，递归算法分析问题的方法是十分有效的。但是，递归算法的时间/空间效率通常比较差。

因此，求解某些问题时，我们希望用递归算法分析问题，用非递归算法具体求解问题，这就需要把递归算法转换为非递归算法。

把递归算法转化为非递归算法有如下三种基本方法：

(1). 通过分析，跳过分解过程，直接用循环结构的算法实现求值过程。

(2). 算法自己用栈模拟系统的运行时栈，通过分析只保存必须保存的信息，从而用非递归算法替代递归算法。

(3). 利用栈保存参数，由于栈的后进先出特性吻合递归算法的执行过程，因而可以用非递归算法替代递归算法。

【参考答案】 A

2. **(原书　第 2 题)**算术表达式 A+B*C-D/E 转为前缀表达式后为（　　）。

A. -A+B*C/DE　　B. -A+B*CD/E

C. -+*ABC/DE　　D. -+A*BC/DE

【解析】本题考查将算术表达式转换成前缀表达式的方法。表达式求值是一个比较重要的题型，表达式的转换，无非在前缀、中缀、后缀之间转换表达式，请同学们要掌握。**遇到表达式的转换，我们先画成树，然后按照树的遍历给出相应的顺序即可。**

对于本题，我们给出了相应的中缀表达式树，如图 3.16 所示。请同学们要注意，一般题目告诉大家算术表达式是什么，意思一般就是说，算术表达式的中缀表达式是什么样的。大家可以根据题意，构造相应的树。

算术表达式对应的树，其非叶子结点都是运算符，叶子结点都是操作。

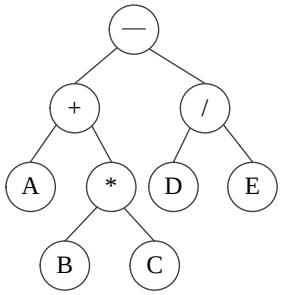


图 3.16

如图 3.26 所示，将算术表达式 A+B*C-D/E 转换成前缀表达式，即先序遍历上图的二叉树，可以得到序列：－＋A＊B C／D E，故而选择 D 答案。

【参考答案】 D

3. **(原书　第 5 题)**中缀表达式 A*(B+C)/(D-E+F)的后缀表达式是（　　）。

A. A*B+C/D-E+F　　B. AB*C+D/E-F+

C. ABC+*DE-F+/　　D. ABCDEF*+/-+

【解析】中缀表达式转后缀表达式，是我们常见的一种表达式转换。我们也直接画一棵中缀表达式树，后序遍历该树得到的序列即是所求。

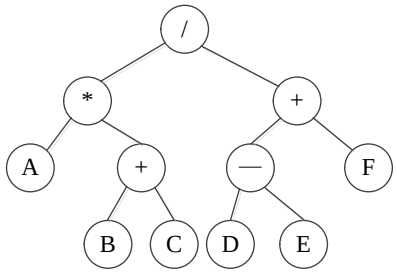


图 3.17

我们可以对如图 3.17 所示的树发起后序遍历，得到序列A B C＋＊D E －F＋ /，对应于 C 答案。

【参考答案】 D

考点 7 特殊矩阵的压缩存储

温馨提示：特殊矩阵的压缩存储，考查内容经典总结如下：1、什么是特殊矩阵？特殊矩阵包括哪几种？（对称矩阵、三角矩阵和三对角矩阵等）2、稀疏矩阵有什么特点？3、特殊矩阵怎么存储？4、怎么计算矩阵中某一个元素转换成一维数组之后的下标？请同学们务必掌握这几种题型。

一．选择题部分

1. **(原书　第 2 题)**稀疏矩阵一般的压缩存储方式有两种，即（　　）。

A. 二维数组和三维数组　　B. 三元组和散列

C.　三元组和十字链表　　D.　散列和十字链表

【解析】本题考查常见的两种稀疏矩阵压缩存储方式。一般来说，稀疏矩阵非 0 元素的分布是没有规律的。因此，在存储非 0 元素的同时，还需要存储适当的辅助信息。稀疏矩阵的常见压缩存储方式有三元组表和十字链表两种方式，这两种方式在本考点接下来的考题中我们会详细讲解，这里先不解释。

【参考答案】 C

2. **（原书　第 4 题）**设二维数组 A[1… m,1… n]按行存储在数组 B 中，则二维数组元素 A[i,j]在一维数组 B 中的下标为（　　）。

A.　n*(i-1)+j　　B.　n*(i-1)+j-1

C.　i*(j-1)　　D.　j*m+i-1

【解析】类似的道理在第 3 题中已经有了讲解，我们在本题不再赘述。矩阵行下标和列下标均从 1 开始，按照行优先存储，由于任一元素a_{ij}前面有 i-1 行,每行 n 个元素，假设可计算 A[i,j]的数组下标为

$$\mathrm{LOC}\left(a_{i,\ j}\right)=\mathrm{LOC}\left(a_{1,\ 1}\right)+((\mathrm{i}-1)\times \mathrm{n}+\mathrm{j})\times \mathrm{L}$$

本题中，相当于 L=1，$\mathrm{LOC}\left(a_{1,\ 1}\right)=0$，故而

$$\mathrm{LOC}\left(a_{i,\ j}\right)=\mathrm{LOC}\left(a_{1,\ 1}\right)+\left((\mathrm{i}-1)\times \mathrm{n}+\mathrm{j}\right)\times 1=\mathrm{n}\times(\mathrm{i}-1)+\mathrm{j}$$

有时候，为了检测看看我们是不是算对了，还可以试着取几个简单的值算算看，比如我们去第 1 行第 1 列的元素，该元素在数组 A 中的下标应该是 1，而

$$\mathrm{LOC}\left(a_{1,\ 1}\right)=\mathrm{n}\times(1-1)+1=1$$

刚好成立，说明公式应该是正确的。

【参考答案】 A

3. **（原书　第 13 题）**一 n×n 的三角矩阵 A=[a_{ij}]如下：

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ & a_{22} & \cdots & a_{2n} \\ & & \ddots & \vdots \\ & & & a_{nn} \end{bmatrix}$$

将三角矩阵中元素 a_{ij}(i≤j)按行序为主序的顺序存储在一维数组 B[1..n(n+1)/2]中，则 a_{ij} 在 B 中的位置是（　　）。

A. (i-1)(2n+i)/2+i-j+1　　B. (i-1)(2n-i+2)/2+j-i+1

C. (i-1)(2n-i)/2+j-i　　D. (i-1)(2n-i+2)/2+j-i

【解析】注意上三角矩阵的压缩存储，元素$a_{i,j}$前面共有 1~i-1 共 i-1 行，元素个数分别为 n、n-1、…、n-(i-1)个，在第 i 行共有 1~j-1 共 j-1 个元素，因此$a_{i,j}$在 B 中的位置为

$$\mathrm{LOC(A[i][j])} = \mathrm{LOC(B[0])} + [n + (n-1) + \cdots + (n-i+2) + i - j + 1] \times \mathrm{d}$$

$$= \frac{(\mathrm{i}-1)(2\mathrm{n}-\mathrm{i}+2)}{2} + j - i + 1$$

【参考答案】 B

第四章　树与二叉树

考点 1 树的概念

温馨提示：本考点考查树的基本概念，都是一些基本知识，请同学们注意树的基本概念，并将树和图区分开来。

一．选择题部分

1. （原书　第 5 题）设某二叉树中度数为 0 的结点数为 N_0，度数为 1 的结点数为 N_l，度数为 2 的结点数为 N_2，则下列等式成立的是(　　)。

A. $N_0=N_1+1$　　B. $N_0=N_l+N_2$

C. $N_0=N_2+1$　　D. $N_0=2N_1+l$

【解析】任何一棵二叉树，设其终端结点的个数为n_0，度为 1 的结点个数为n_1，度为 2 的结点个数为n_2，二叉树的总结点个数 n 为度分别为 0、1、2 的结点数之和，即

$$n = n_0 + n_1 + n_2$$

除根结点外，其余结点都有一条边进入，设边数为 e，有 n=e+1。由于这些边是由度为 1 或者 2 的结点发出的，所以又有 e=$n_1 + 2n_2$，于是得

$$n = e + 1 = n_1 + 2n_2 + 1$$

故而

$$n_0 + n_1 + n_2 = n_1 + 2n_2 + 1$$

即$n_0 = n_2 + 1.$

【参考答案】 C

2. （原书　第 7 题）下列有关树的概念错误的是(　　)。

A. 一棵树中只有一个无前驱的结点

B. 一棵树的度为树中各个结点的度数之和

C. 一棵树中，每个结点的度数之和等于结点总数减 1

D. 一棵树中每个结点的度数之和与边的条数相等

【解析】本题考查树的相关概念。一棵树中只有根结点没有前驱结点，除根结点之外，每个结点都有一个前驱结点，即父结点，故而 A 答案正确。

通常我们把一个结点拥有子树的个数称为该结点的度数，所以结点的度数之和等于除根结点外所有结点的个数，即每个结点的度数之和等于结点总数减 1，故而 C 答案正确。

结点的度等于该结点子树的个数，而结点与子树之间是以边连接的，所以一棵树中每个结点的度数之和与边的条数相等，D 答案正确。

一棵树的度等于结点中度数最大的结点的度数，而不是树中各结点的度数之和。故而，A 答案错误。

【参考答案】 B

二．综合应用题部分

1. （原书　第 1 题）有一棵树如图 4.1 所示，回答下面的问题：

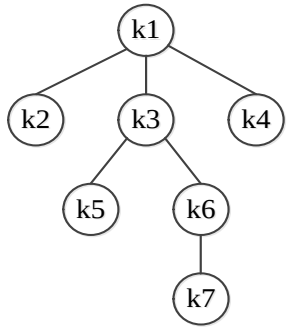


图 4.1

(1). 这棵树的根结点是_____.

(2). 这棵树的叶子结点是_____.

(3). 结点 k3 的度是_____.

(4). 这棵树的度是_____.

(5). 这棵树的深度是_____.

(6). 结点 k3 的子女是_____.

(7). 结点 k3 的父结点是_____.

【解析】由图可知，

(1). 这棵树的根结点是结点 k1。

(2). 这棵树的叶子结点是 k2、k5、k7、k4 。

(3). k3 结点有 2 个孩子结点 k5 与 k6，故而其度为 2。

(4). 树的度等于数中度最大的结点的度数，本题中度数最大的结点为根结点，度数为 3。故而，树的度为 3。

(5). 很显然本题的树的层数为 4 层，第一层（层数我们从 1 开始算）有结点 k1，第二层有结点 k2、k3 和 k4，第三层有结点 k5 和 k6，第四层有结点 k7。因而，树的深度为 4。

(6). k3 结点的子女结点为 k5、k6。要特别注意，k3 的子女结点即把 k3 作为父结点的结点，k3 的孩子的孩子不再是 k3 的孩子而是 k3 的子孙，某一个结点的父结点的父结点也不再是该结点的父结点而是该结点的祖先。

(7). k3 结点的父结点是结点 k1。

考点 2 二叉树

温馨提示：本考点考查二叉树，包括：1、二叉树的定义及其主要特征；2、叉树的顺序存储结构和链式存储结构；3、二叉树的遍历；4、线索二叉树的基本概念和构造。这些知识点都很重要，请同学们务必自己掌握。我们将这 4 个知识点区分开来，分别命题，请大家多练习。

考点 2.1 二叉树的定义及其主要特征

一．选择题部分

1. **（原书　第 1 题）**在下述论述中，正确的是(　　)。

(1). 有一个结点的二叉树的度为 0；

(2). 二叉树的度为 2；

(3). 二叉树的左右子树可任意交换；

(4). 深度为 K 的顺序二叉树的结点个数小于或等于深度相同的满二叉树。

A.　(1)、(2)、(3)　　B.　(2)、(3)、(4)

C.　(2)、(4)　　D.　(1)、(4)

【解析】（1）中，若二叉树只有一个结点，那么该结点不存在孩子结点，所以二叉树的度为 0 是正确的。

（2）中，二叉树的度不一定为 2。二叉树只是说明了二叉树最大的度数为 2，并没有说明二叉树不存在度数为 1 或者为 0 的情况，如（1）中的二叉树的度就为 0。

（3）中，二叉树的左右子树有先后顺序，不能随意更换，更换了就不是原来的二叉树了。这跟树不同，树的子树没有先后顺序，能任意交换。

（4）中，满二叉树始终满足相同深度下结点个数最大，故而该说法正确。

综上，（1）（4）正确，（2）（3）错误，故而选择 D 答案。

【参考答案】 D

2. （原书　第 10 题）设高度为 h 的二叉树上只有度为 0 和度为 2 的结点，则此类二叉树中所包含的结点数至少为(　　)。

A. 2h　　B. 2h-1　　C. 2h+1　　D. h+1

【解析】高度为 h，而且只有度数为 0 或者 2 的结点，这是什么样的树呢？如图 4.2 所示的二叉树就是一棵这样的二叉树。

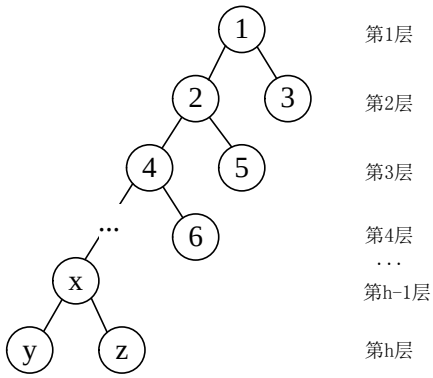


图 4.2

由图 4.2 可以观察出，除了第 1 层，其他每一层都是两个结点，故而总共有结点 2(h-1)+1=2h-1。

【参考答案】 B

3. （原书　第 13 题）以下说法错误的是(　　)。

A. 存在这样的二叉树，对它采用任何次序遍历其结点访问序列均相同

B. 二叉树是树的特殊情形

C. 由树转换成二叉树，其根结点的右子树总是空的

D. 在二叉树只有一棵子树的情况下也要明确指出该子树是左子树还是右子树

【解析】尽管二叉树与树有许多相似之处，但二叉树不是树的特殊情形，故而 B 答案错误。

对于 A 答案，如果二叉树中只有一个根结点，则不管采用什么样的遍历算法，所得访问序列都是一样的。

对于 C 答案，树转换成二叉树时，树的左孩子为树的一个孩子，第二个孩子成为其左孩子的右孩子，第三个孩子（如果存在的话）成为其左孩子的右孩子的右孩子…。故而，根结点的右子树总是空的。

二叉树是有序树，其左右孩子有次序之分，即使只有一棵子树也要明确指出该子树是左子树还是右子树。

【参考答案】 B

二．综合应用题部分

1. （原书　第 5 题）设计在链式存储结构上交换二叉树中所有结点左右子树的算法。

```
typedef struct node
 {
    int data;
    struct   node   *lchild, *rchild;
 } bitree;
```

【解析】二叉树是递归定义的，我们也采用递归的方法来交换结点的左右孩子。首先，判断二叉树是否为空树，若为空树，无需交换。否则，递归交换左右孩子的指针。在链式存储结构上交换二叉树所有结点的左右孩子的算法如下：

```
void swapbitree(bitree   *bt)
   {
        bitree *p;
        if(bt==0) return;
        swapbitree(bt->lchild);        //交换根结点左子树上的结点的左右孩子结点（递归）
        swapbitree(bt->rchild);        //交换根结点右子树上的结点的左右孩子结点（递归）
        p=bt->lchild;                  //最后，交换根结点的左右孩子结点
        bt->lchild=bt->rchild;
        bt->rchild=p;
    }
```

考点 2.2 二叉树的顺序存储结构和链式存储结构

一．选择题部分

1. （原书　第 2 题）用顺序存储的方法将完全二叉树中的所有结点逐层存放在数组中 R[1..n]，结点 R[i]若有左孩子，其左孩子的编号为结点(　　)。

A. R[2i+1]　　B. R[2i]　　C. R[i/2]　　D. R[2i-1]

【解析】将完全二叉树存储在数组中 R[1…n]中，结点 R[i]若有左孩子，其左孩子的编号为结点 R[2i]，若有右孩子，则右孩子的编号为结点 R[2i+1]。故而，本题选择 B 答案。

值得注意的是，一般二叉树的顺序存储，将每个结点与完全二叉树上的结点相对照，然后存储在数组的相应位置中，缺少的结点用“0”补齐。

【参考答案】 B

二．综合应用题部分

1. （原书　第 1 题）一棵二叉树的结点数据存储在如表 4.1 的数组中。

表 4.1

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
e	a	f		d		g			c	j			h	i					b

(1). 画出对应的二叉树。

(2). 画出此二叉树对应的森林。

【解析】

(1). 本题的二叉树如图 4.3 所示。

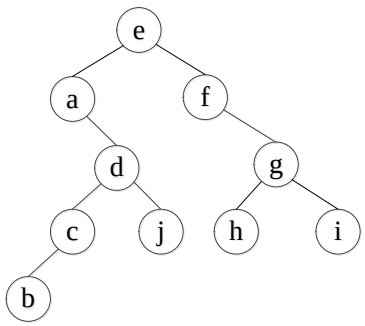


图 4.3

(2). 该二叉树对于的森林如图 4.4 所示。

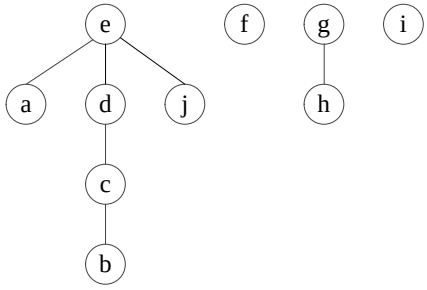


图 4.4

考点 2.3 二叉树的遍历

一. 选择题部分

1. **(原书　第 2 题)**设 x 和 y 是二叉树中的任意两个结点，若在先根序列中 x 在 y 之前，而在后根序列中 x 在 y 之后，则 x 和 y 的关系是(　　)。

A. x 是 y 的左兄弟　　B. x 是 y 的右兄弟

C. x 是 y 的祖先　　D. x 是 y 的子孙

【解析】对于类似的单选题，我们只需要列举一种情况就可以了，不必每种情况都看看能不能实现。

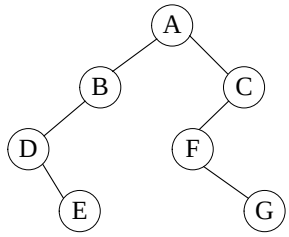


图 4.5

如图 4.5 中的 A、E 结点，在先根遍历时，A 在 E 的前面；在后跟遍历时，E 在 A 的前面，A 是 E 的祖先。故而，选择 C 答案。

【参考答案】 C

以下 2~4 题基于图 4.6。

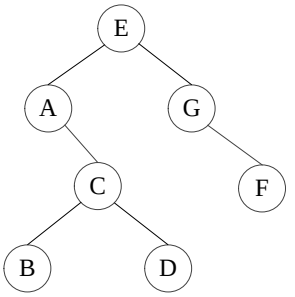


图 4.6

2. **(原书　第 10 题)**该二叉树结点的前序遍历的序列为(　　)。

A. E、G、F、A、C、D、B

B. E、A、G、C、F、B、D

C. E、A、C、B、D、G、F

D. E、G、A、C、D、F、B

【解析】**前序遍历**首先访问根结点，其次遍历左子树，最后遍历右子树。在遍历左、右子树时，仍然先访问根结点，然后遍历左子树，最后遍历右子树，即遍历左右子树时仍然采用前序遍历方法。

对题中的二叉树进行前序遍历，可以得到序列 E→A→C→B→D→G→F，故而选择 C 答案。

【参考答案】 C

3. **(原书　第 11 题)**该二叉树结点的中序遍历的序列为(　　)。

A. A、B、C、D、E、G、F

B. E、A、G、C、F、B、D

C. E、A、C、B、D、G、F

D. B、D、C、A、F、G、E

【解析】**中序遍历**首先遍历左子树，然后访问根结点，最后遍历右子树。在遍历左、右子树时，仍然先遍历左子树，再访问根结点，最后遍历右子树，即遍历左右子树时仍然采用中序遍历方法。

对题中的二叉树进行中序遍历，可以得到序列 A→B→C→D→E→G→F，故而选择 A 答案。

【参考答案】 A

4. **(原书　第 12 题)**该二叉树的按层遍历的序列为（　　）。

A. E、G、F、A、C、D、B

B. E、A、C、B、D、G、F

C. E、A、G、C、F、B、D

D. E、G、A、C、D、F、B

【解析】**层次遍历**，也是常见的二叉树遍历算法。层次遍历二叉树，即从上到下，从左到右遍历二叉树的结点。对题中的二叉树进行层次遍历，可以得到序列 E→A→G→E→F→B→D，故而选择 B 答案。

【参考答案】 C

二．综合应用题部分

1. **(原书　第 2 题)**将图 4.7 所示的一棵普通树转换成一棵二叉树，并写出它先序、中序、后序三种遍历的遍历序列。

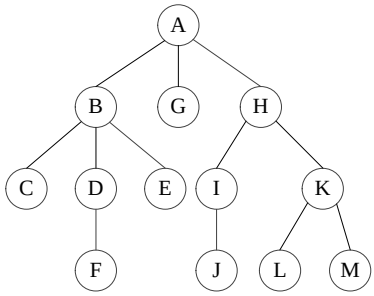


图 4.7

【解析】将题中的树转换成如图 4.8 所示的二叉树。

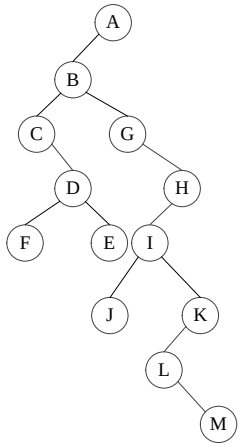


图 4.8

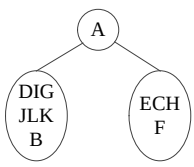
对该二叉树进行先序遍历，得到序列为 A→B→C→D→F→E→G→H→I→J→K→L→M。对该二叉树进行中序遍历，得到序列为 C→F→D→E→B→G→J→I→L→M→K→H→A。对该二叉树进行后序遍历，得到序列序列为 F→E→D→C→J→M→L→K→I→H→G→B→A。

2. **（原书　第 4 题）**已知一棵二叉树的中序序列和后序序列分别如下，请画出该二叉树。

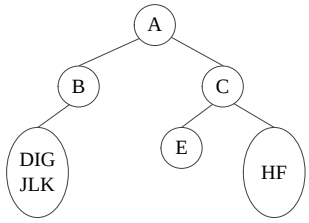
中序序列：D　I　G　J　L　K　B　A　E　C　H　F

后序序列：I　L　K　J　G　D　B　E　H　F　C　A

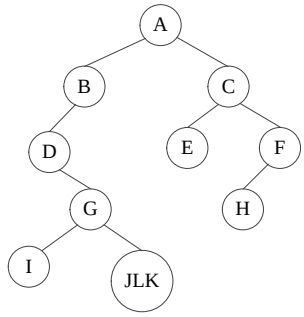
【解析】已知二叉树的中序遍历序列为 D→I→G→J→L→K→B→A→E→C→H→F，后序序列为 I→L→K→J→G→D→B→E→H→F→C→A。可知 A 是二叉树的根结点，该二叉树的构建过程如图 4.9 所示。



图（1）



图（2）



图（3）

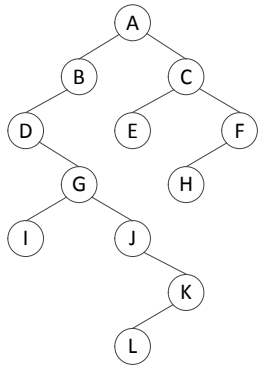


图 4.9

3. **（原书　第 9 题）**请在下划线上填入适当的语句，完成以下算法。

```
Status Preordertraverse(Bitree T,Status(*Visit)(Telemtype e)){
//先序非递归遍历二叉树。
Initstack ( S );     Push ( S,T );
```

```
While ( !stackempty(S) && p!=NULL )
  {
     if (p )
       {
          visit (p->data ) ;
          _____________________;
          _____________________;
        }
     else
       {
          push( S, p);
          _____________________;
       }
  }
return ok;
}
```

【解析】二叉树的前序遍历的递归算法基本思想是：

首先判断根结点是否为空，为空则遍历结束，不为空则将左子作为新的根结点重新进行下述判断，左子树遍历结束后，再将根结点的右孩子作为根结点判断，直至结束。到达每一个结点时，打印该结点数据，即得先序遍历结果。

若二叉树非空，则依次进行如下操作：

(1). 访问跟结点；

(2). 前序遍历左子树；

(3). 前序遍历右子树。

二叉树的前序遍历非递归算法的基本思想是：建立一个栈 S，当指针到达根结点时，打印根结点，判断根结点是否有左孩子和右孩子，有左孩子和右孩子的话就打印左孩子同时将右孩子入栈，将左孩子作为新的根结点进行判断，方法同上。

若当前结点没有左孩子，则直接将右孩子打印，同时将右孩子作为新的根结点判断。若当前结点没有右孩子，则打印左孩子，同时将左孩子作为新的根结点判断。

若当前结点既没有左孩子也没有右孩子，则当前结点为叶子结点，此时将从栈中出栈一个元素，作为当前的根结点，打印结点元素，同时将当前结点同样按上述方法判断，依次进行。直至当前结点的左右子都为空，且栈为空时，遍历结束。

该算法的过程如图 4.10 所示。

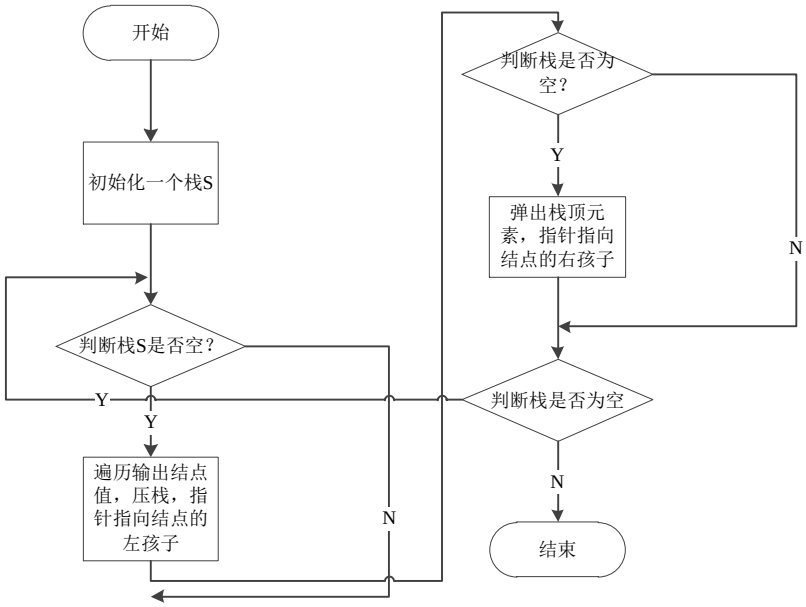


图 4.10

故而，对算法解析如下：

```
Status Preordertraverse(Bitree T,Status(*Visit)(Telemtype e)){
//先序非递归遍历二叉树。
Initstack ( S );
Push ( S,T );
While ( !stackempty( S )&&p!=NULL )
  {
    if ( p )
      {
            visit (p->data ) ;
            ____push(S,p)_____;
            ____p=p->lchild_____;
      }
```

```
        else
          {
                pop(S,p);
                p=p->rchild      ;
           }
      }
   return ok;
}
```

考点 2.4 线索二叉树的基本概念和构造

一．选择题部分

1. （原书　第 1 题）引入二叉线索树的目的是(　　)

   A. 加快查找结点的前驱或后继的速度

   B. 为了能在二叉树中方便的进行插入与删除

   C. 为了能方便的找到双亲

   D. 使二叉树的遍历结果唯一

【解析】本题考查引入线索二叉树的目的。二叉树的线索化，其实在上一个考点的一部分经典考题里面，我们已经有所涉及。那么，引入二叉线索树的目的是什么呢？目的是有效利用空指针域，提高遍历运算的效率，加快查找结点的前驱或者后继结点的速度，简化遍历算法。对于有 n 个结点的线索二叉树上含有 n+1 条线索。

【参考答案】 A

2. （原书　第 4 题）在中序线索二叉树中，若某结点有右孩子，则该结点的直接后继是(　　)

   A. 左子树的最右下结点　　B. 右子树的最右下结点

   C. 左子树的最左下结点　　D. 右子树的最左下结点

【解析】二叉树的线索化说透了就是按照某种遍历算法将结点的线索引向前驱或者后继（若不存在左孩子，则左指针引向该遍历序列的直接前驱结点；若不存在右孩子，则右指针引向该遍历序列的直接后继结点）。

若中序遍历时，某结点有右孩子，那么该结点中序遍历的直接后继应该是其右子树的最左下的结点。故而，本题选择 D 答案。

【参考答案】 D

二．综合应用题部分

1. **（原书　第 2 题）**对图 4.11 给出的二叉树，分别进行：

(1). 先序前驱线索化；

(2). 后序后继线索化；

(3). 中序线索化。

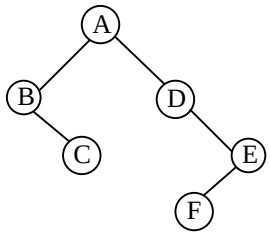


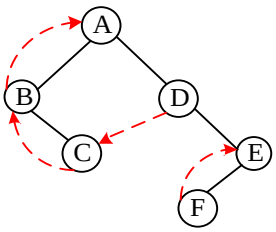
图 4.11

【解析】本题考查二叉树的线索化。二叉树的线索化方式很多，请同学们要充分理解线索化的概念，并能够很具题目的要求，对二叉树进行线索化。

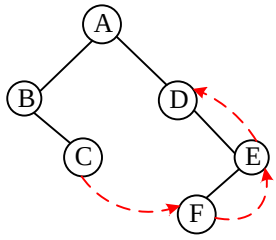
(1). 先序前驱线索化，无需对后继线索化。对题中的二叉树进行先序遍历，可以得到序列为 A→B→C→D→E→F，可以得到该二叉树的先序前驱线索化如图 4.12（1）所示。

(2). 后序后继线索化，无需对前驱线索化。对题中的二叉树进行后序遍历，可以得到序列为 C→B→F→E→D→A，可以得到该二叉树的后序后继线索化如图 4.12（2）所示。

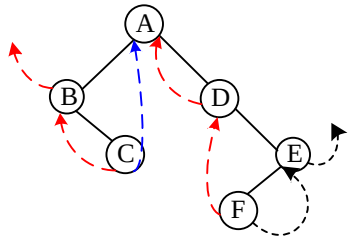
(3). 中序线索化。对题中的二叉树进行中序遍历，可以得到序列为 B→C→A→D→F→E，可以得到该二叉树的先序前驱线索化如图 4.12（3）所示。



图（1）先序前驱线索



图（2）后序后继线索



图（3） 中序线索化

图 4.12

考点 3 树与森林

温馨提示：本考点主要考查：1、树的存储结构；2、森林与二叉树的转换；3、树和森林的遍历。本部分内容的命题形式比较固定，解题方法也比较固定，同学们可以通过掌握一个题的解题方法，而掌握一类题型的解题方法。

一．选择题部分

1. **（原书　第 5 题）**设 F 是由 T1、T2 和 T3 三棵树组成的森林，与 F 对应的二叉树为 B，T1、T2 和 T3 的结点数分别为 N1、N2 和 N3，则二叉树 B 的根结点的左子树的结点数为(　　)。

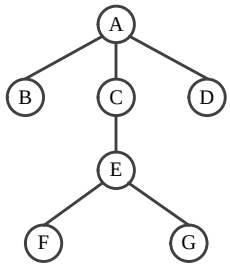
A. N1-1　　B. N2-1　　C. N2+N3　　D. N1+N3

【解析】本题道理与第 1 题相同。F 是由 T1、T2 和 T3 三棵树组成的森林，与 F 对应的二叉树为 B，T1、T2 和 T3 的结点数分别为 N1、N2 和 N3，则二叉树 B 的根结点的左子树的结点数 N1-1。

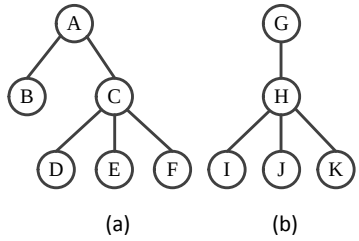
【参考答案】 A

二．综合应用题部分

1. **（原书　第 4 题）**将图 4.13 所示的树转换成二叉树，并写出该二叉树的先序遍历序列。



第4题图



第5题图

图 4.13

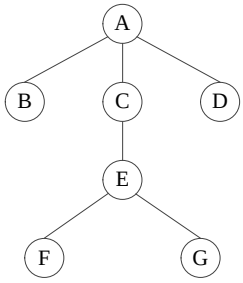
【解析】将一般树转化为二叉树的思路，主要根据树的孩子-兄弟存储方式而来，步骤是：

(1). 加线：即在各兄弟结点之间用虚线相连，可理解为每个结点的兄弟指针指向它的一个兄弟。

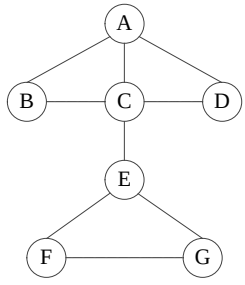
(2). 抹线：对每个结点仅保留它与其最左一个孩子的连线，抹去该结点与其他孩子之间的连线（可理解为每个结点仅有一个孩子指针，让它指向自己的长子）。

(3). 旋转：把虚线改为实线从水平方向向下旋转成右斜下方向，原树中实线成左斜下方向。这样，树就转换成二叉树了。

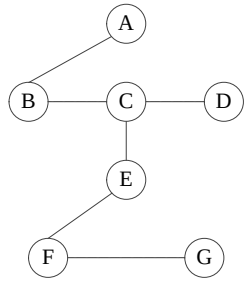
以下图（2）、（3）、（4）分别对应以上步骤。树转换得到了如图 4.14 所示的二叉树。



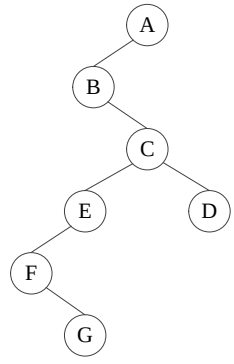
图（1）



图（2）



图（3）



图（4）

图 4.14

该二叉树的先序遍历顺序为 A、B、C、E、F、G、D。

2. **（原书　第 7 题）**已知某森林的二叉树如图 4.15 所示，试画出它所表示的森林。

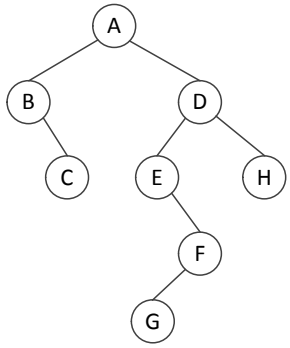


图 4.15

【解析】将二叉树还原成一般树，其还原过程也分为三步：

(1). **加线：**若某结点 i 是双亲结点的左孩子，则将该结点 i 的右孩子以及当且仅当连续地沿着右孩子的右链不断搜索到所有右孩子，都分别与结点 i 的双亲结点用虚线连接。

(2). **抹线：**把原二叉树中所有双亲结点与其右孩子的连线抹去。这里的右孩子实质上是原一般树中结点的兄弟，抹去的连线是兄弟间的关系。

(3). **进行整理：**把虚线改为实线，把结点按层次排列。

上面三个过程分别对应图 4.16 的（2）、（3）和（4）。

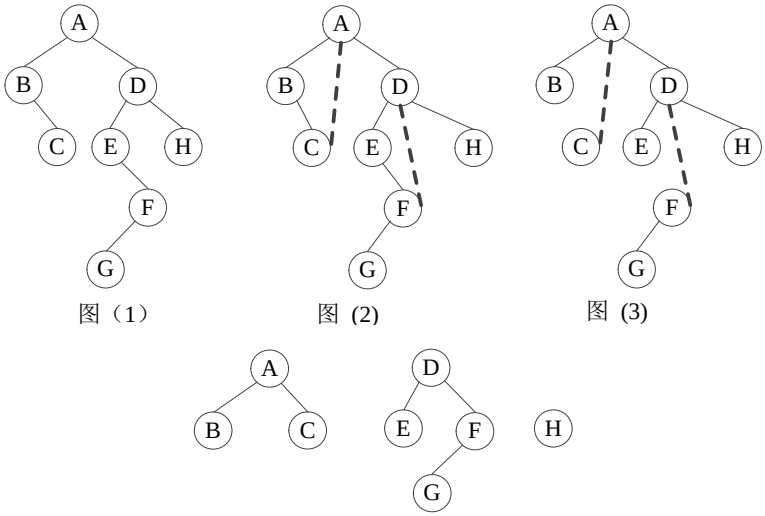


图 4.16

最后得到的二叉树如图 4.16（4）所示。

考点 4 树与二叉树的应用

温馨提示：本考点主要考查：1、二叉排序树；2、平衡二叉树；3、哈夫曼(Huffman)树和哈夫曼编码。这三个知识点都特别重要，我们分别对这三个考点的核心题型做了整理，请各位童鞋务必掌握。

考点 4.1 二叉排序树

一．选择题部分

1. (原书　第 2 题)分别以下列序列构造二叉排序树，与用其它三个序列所构造的结果不同的是(　　)。

   A. (15，13，14，6，17，16，18)

   B. (15，17，16，18，13，6，14)

   C. (15，6，13，14，17，16，18)

   D. (15，13，6，14，17，18，16)

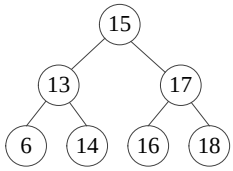
【解析】本题考查二叉排序树的构造方法。我们可以把二叉排序树的构造过程看成在二叉排序树上查找某关键字不成功而插入该关键字的过程。二叉排序树的插入过程也是递归的。其插入过程如下：

(1). 若二叉排序树为空，则待插入结点作为根结点插入到空树中；

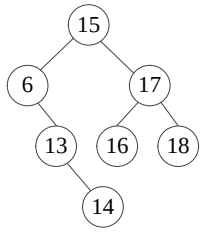
(2). 当非空时，将待插结点关键字和二叉排序树的根关键字 t->key 进行比较，

- 若 s->key = t->key,则无须插入；
- 若 s->key< t->key,则插入到根的左子树中；
- 若 s->key> t->key,则插入到根的右子树中。

对于本题，我们分别对 A、B、C、D 四个答案的关键字序列构造二叉排序树，所得二叉排序树如图 4.17 所示。



A、B、D答案对应的二叉排序树



C答案对应的二叉排序树

图 4.17

由于 C 答案所提供的关键字序列所构造的二叉排序树不同于 A、B、D 答案所提供的关键字所能构造的二叉排序树，故而本题选择 C 答案。

【参考答案】 C

2. **（原书　第 8 题）**若构造一棵具有 n 个结点的二叉排序树，最坏的情况下其深度不超过（　　）。

A. $\frac{n}{2}$　　B. n　　C. $\frac{n+1}{2}$　　D. n+1

【解析】最坏的情况下，二叉排序树为单支树，比如构造一棵{1,2,3,4,5,…,n}的二叉树，则得到的二叉排序树如图 4.18 所示。

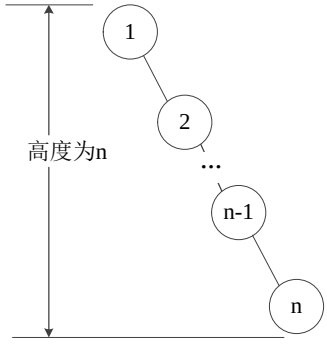


图 4.18

由图 4.18 可以看出，最坏的情况下，每插入一个结点，都在该二叉排序树的尾部插入，该二叉排序树插入结点的复杂度类似于在链表的表尾增加一个结点，该二叉排序树的深度为 n。

【参考答案】 B

二．综合应用题部分

1. **（原书　第 2 题）**已知长度为 12 的表{Jan，Feb，Mar，Apr，May，June，July，Aug，Sep，Oct，Nov，Dec}

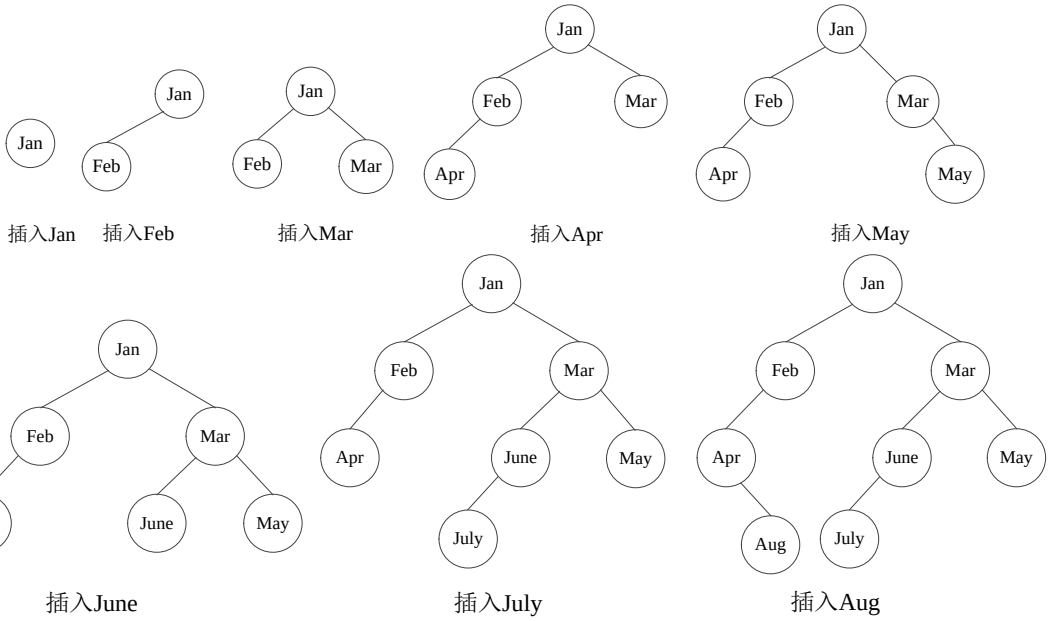
(1). 试按表中元素的次序依次插入一棵初始为空的二叉排序树，并求在等概率情况下查找成功的平均查找长度。

(2). 用表中元素构造一棵最佳二叉排序树，求在等概率的情况下查找成功的平均查找长度。

(3). 按表中元素顺序构造一棵 AVL 树，并求其在等概率情况下查找成功的平均查找长度。

【解析】

(1). 依次将 Jan，Feb，Mar，Apr，May，June，July，Aug，Sep，Oct，Nov，Dec 等 12 个元素按照字母顺序插入到一棵初始为空的二叉排序树中，构造二叉排序树的过程如图 4.19 所示。



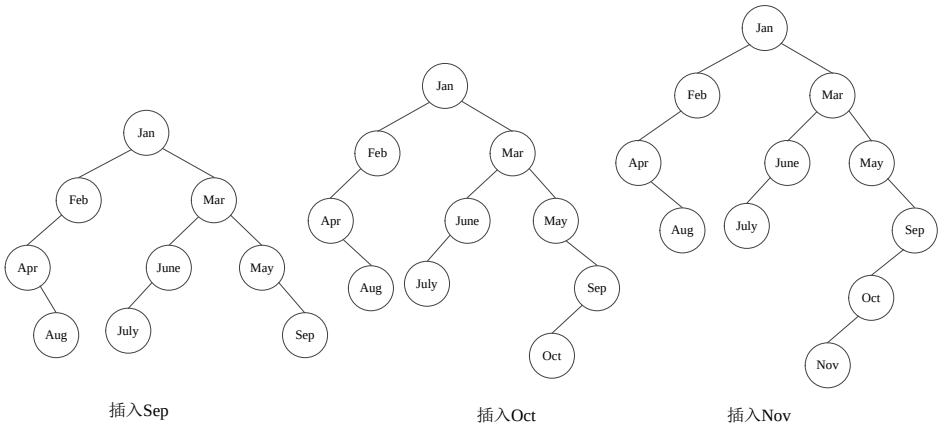


图 4.19

最后得到的二叉排序树如图 4.20 所示。

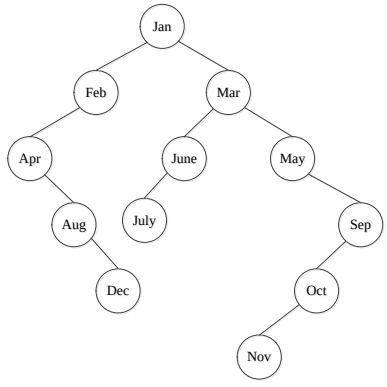


图 4.20

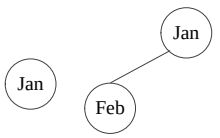
在等概率情况下，查找成功的平均查找长度为

ASLsuss =(1*1+2*2+3*3+4*3+5*2+6*1)/12=42/12=3.5

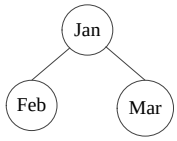
(2). 最佳二叉排序树的形状和 12 个结点的折半查找判定树相同，查找成功的平均查找长度为

ASLsuss =(1*1+2*2+4*3+5*4)/12=37/12

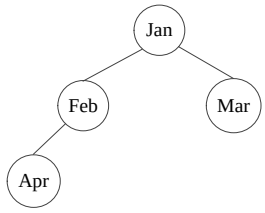
(3). 构造平衡二叉树的过程如图 4.21 所示。



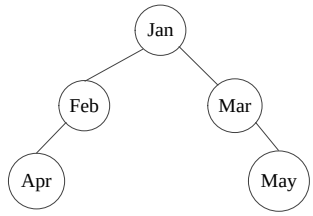
插入Jan　插入Feb



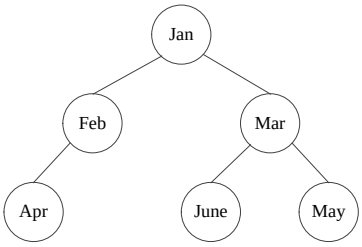
插入Mar



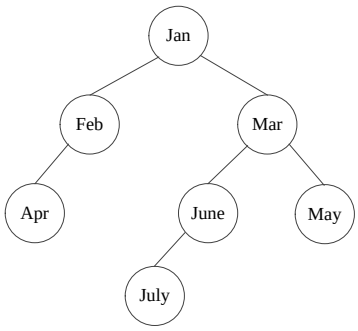
插入Apr



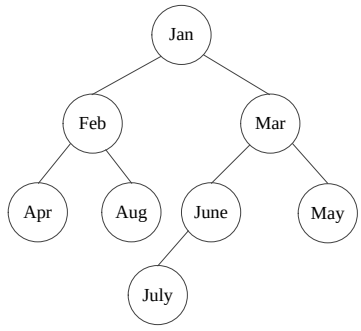
插入May



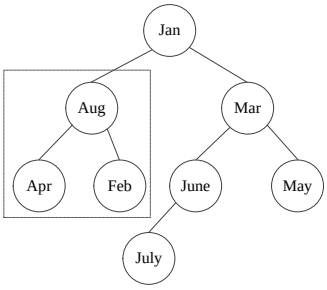
插入June



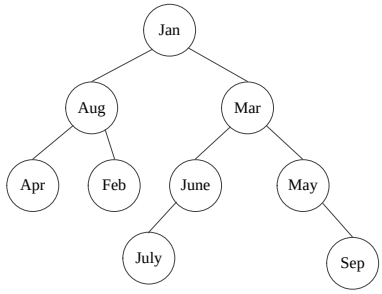
插入July



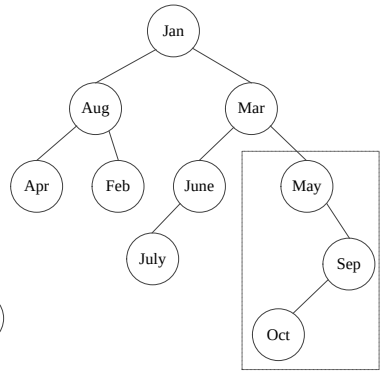
插入Aug后失去平衡



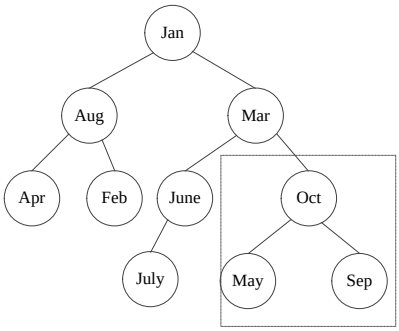
插入Aug后，旋转平衡



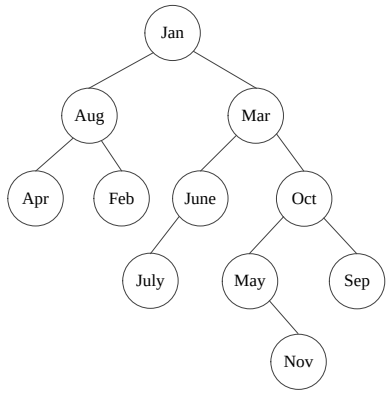
插入Sep



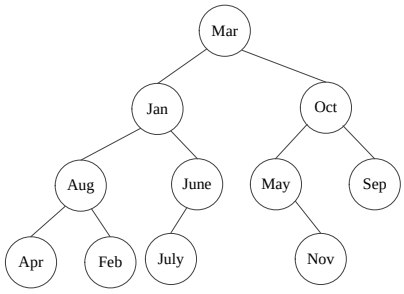
插入Oct后失去平衡



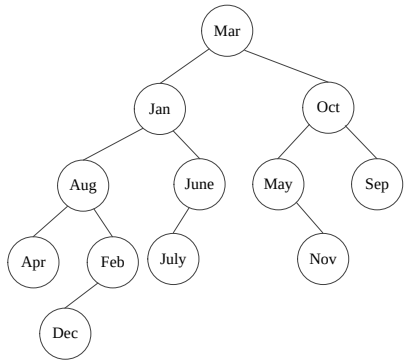
插入Oct后旋转平衡



插入Nov后Jan的左右子树失去平衡



旋转平衡



插入Dec,得到最终的平衡二叉树

图 4.21

对上图插入 Dec 之后得到的平衡二叉树，查找成功的平均查找长度为

ASLsuss =(1*1+2*2+3*4+4*4+5*1)/12=38/12

2. **（原书　第 6 题）**设计递归和非递归算法实现在二叉排序树上查找关键值 k。

【解析】本题考查二叉排序树的查找过程算法。二叉排序树的查找过程为：

(1). 若查找树为空，查找失败。

(2). 查找树非空，将给定值 key 与查找树的根结点关键码比较。

(3). 若相等，查找成功，结束查找过程，否则：

◆　当给值 key 小于根结点关键码，查找将在以左孩子为根的子树上继续进行，转(1)。

◆　当给值 key 大于根结点关键码，查找将在以右孩子为根的子树上继续进行，转(1)。

利用递归算法来完成二叉排序树的查找过程，算法如下：

```
Bin_Sort_Tree_Linklist*bt_search(Bin_Sort_Tree bt,keytype k)
```

```
{
    //在根指针为 bt 的二叉排序树上查找一个关键字值为 k 的结点，
    //若查找成功返回指向该结点的指针，否则返回空指针
    if(bt=NULL)‖(bt->key==k)
        return bt;
    else if(k<bt->key)return bt_search(bt->lchild,k);          //在左子树中搜索
    else return bt_search(bt->rchild,k);                       //在右子树中搜索
 }
```

这个过程也可以用非递归算法实现，算法描述如下：

```
Bin_Sort_Tree_Linklist*bt_search1(Bin_Sort_Tree bt,keytype k)
 {
     p=bt;//指针 p 指向根结点，搜索从根结点开始
     while(p!=NULL && p->key!=k)
         {
             if(k<p->key)p=p->lchild;
             else p=p->rchild;
         }
     return(p);
 }
```

考点 4.2　平衡二叉树

一．选择题部分

1. **(原书　第 1 题)** 由元素序列（27，16，75，38，51）构造平衡二叉树，则首次出现的最小不平衡子树的根（即离插入结点最近且平衡因子的绝对值为 2 的结点）为(　　)。

A. 27　　B. 38　　C. 51　　D. 75

【解析】由元素序列（27，16，75，38，51）构造平衡二叉树，首次出现的最小不平衡子树的根（即离插入结点最近且平衡因子的绝对值为 2 的结点）为第一次需要旋转的部分，我们来看看构造平衡二叉树的过程（如图 4.22）：

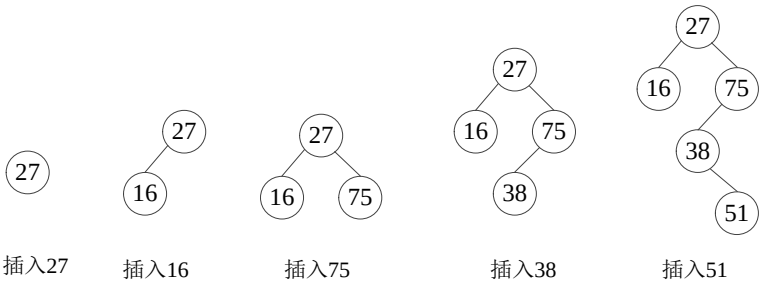


图 4.22

如图 4.22 所示，当插入结点 51 之后，有 27 和 75 两个结点失去平衡，但是首次出现的最小不平衡子树应该是离插入结点 51 最近的且平衡因子的绝对值为 2 的结点 75，而不是 27。

【参考答案】 D

二．综合应用题部分

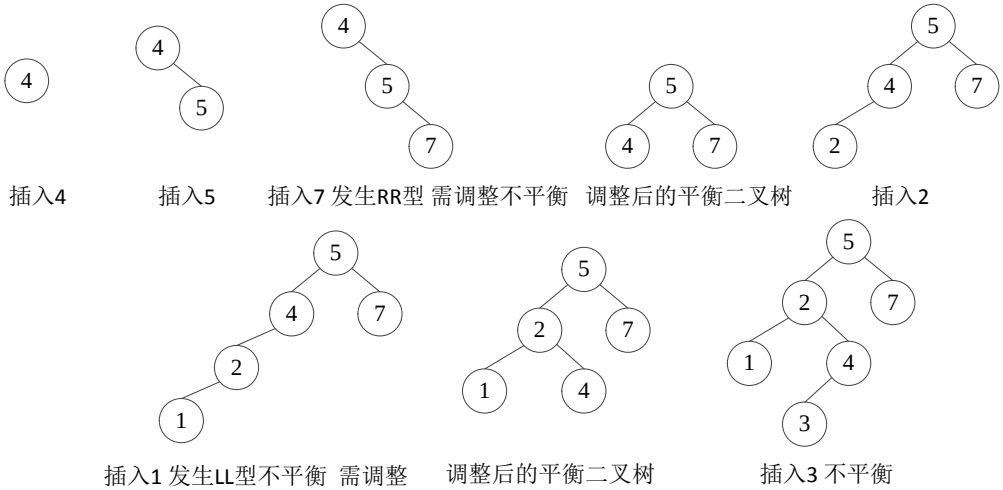
1. **（原书　第 2 题）**设关键字的输入序列为{4，5，7，2，1，3，6}

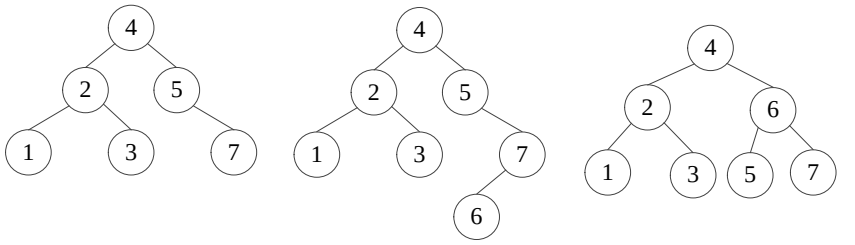
(1). 从空树开始构造平衡二叉树，画出每加入一个新结点时二叉树的形态，若发生不平衡，指明需做的平衡旋转类型及平衡旋转的结果。

(2). 上面的数据作为待排序的数据，写出用快速排序进行一趟划分后的数据序列。

【解析】

(1). 平衡二叉树的构造过程如图 4.23 所示。





调整后的平衡二叉树　　插入6　　调整后的最终平衡二叉树

图 4.23

【经典总结】

调整的过程若我们还不是很熟练，为了提高我们调整后的树的正确率，可采用中序遍历二叉树的方法看看是不是一个递增有序的序列。若是递增，则可能就正确了。若非递增，则肯定旋转错了。这是一条十分有用的方法，但是只能简单的找到平衡调整错误，有些平衡调整错误检测不出来。

(2). 一趟划分后的数据序列为 3、1、2、4、7、5、6。

考点 4.3 哈夫曼树和哈夫曼编码

一. 选择题部分

1. **(原书　第 2 题)** 由五个分别带权值为 9，2，3，5，14 的叶子结点构成的一棵哈夫曼树，该树的带权路径长度为(　　)。

A. 60　　B. 66　　C. 67　　D. 50

【解析】哈夫曼树的构造过程如下：

(1). 根据给定的 n 个权值{w1, w2, …, wn}，构造 n 棵二叉树的集合 F = {T1, T2, … , Tn}，其中每棵二叉树中均只含一个带权值为 wi 的根结点，其左、右子树为空树;

(2). 在 F 中选取根结点的权值最小的两棵二叉树，分别作为左、右子树构造一棵新的二叉树，并置这棵新的二叉树根结点的权值为其左、右子树根结点的权值之和;

(3). 从 F 中删去这两棵树，同时将刚生成的新树加入到 F 中;

(4). 重复 (2) 和 (3) 两步，直至 F 中只含一棵树为止。

由五个分别带权值为 9，2，3，5，14 的叶子结点构成的一棵哈夫曼树，该哈夫曼树如图 4.24 所示。

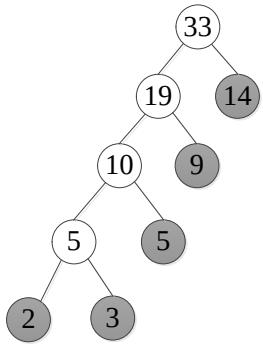


图 4.24

由图 4.24 可知，该哈夫曼树的带权路径长度为

$$WPS=(2+3)\times 4+5\times 3+9\times 2+14=67$$

【参考答案】 C

2. **（原书　第 3 题）**设某哈夫曼树中有 199 个结点，则该哈夫曼树中有(　　)个叶子结点。

A. 99　　B. 100　　C. 101　　D. 102

【解析】正如第一题所言，哈夫曼树其实是二叉树的一种。哈夫曼树的特点是没有度为 1 的结点，根据二叉树的性质可知 n0=n2+1。所以，具有 n 个叶子结点的二叉树具有 n-1 个度为 2 的结点。

当 n=199 时，$n_0+n_2=199$，可知 $n_0=100$，$n_2=99$。故而，本题选择 B 答案。

【参考答案】 B

二．综合应用题部分

1. **（原书　第 5 题）**在一份电文中共使用 8 种字符，即 a, b, c, d, e, f, g, h，它们出现的频率依次为 0.07, 0.19,0.02,0.06, 0.32, 0.03, 0.21, 0.10，试画出对应的哈夫曼树，求出每个字符的哈夫曼编码，并求出传送电文的总长度。

【解析】题中已给出 8 种字符的出现频率，如表 4.2 所示。

表 4.2

字符	a	b	c	d	e	f	g	h
出现频率	0.07	0.19	0.02	0.06	0.32	0.03	0.21	0.10

可构造哈夫曼树如图 4.25 所示。

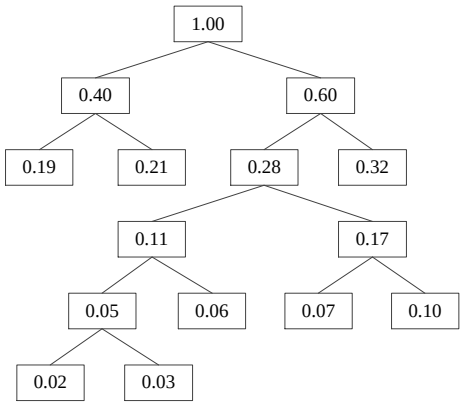


图 4.25

根据上图可知，每个字符的哈夫曼编码如表 4.3 所示。

表 4.3

字符	a	b	c	d	e	f	g	h
哈夫曼编码	1010	00	10000	1001	11	10001	01	1011

传送电文的总长度为：

$$WPS=0.07\times 4+0.19\times 2+0.02\times 5+0.06\times 4+0.32\times 2+0.03\times 5+0.21\times 2+0.10\times 4=2.61$$

我们可以简单地整理一下写法，写成：

$$\text{哈夫曼树总的编码长度}=\sum_1^n \text{第 i 层的叶子结点数目}\times(i-1)$$

根据公式，我们可以把传送电文的总长度写成：

$$WPS=0\times(1-1)+0\times(2-1)+(0.19+0.21+0.32)\times(3-1)+0\times(4-1)+(0.06+0.07+0.10)\times(5-1)+(0.02+0.03)\times(6-1)=1.44+0.92+0.25=2.61$$

本章到此就结束了，请问您有什么疑问吗？任何问题，欢迎您与我们作者进行交流！

shareOurDreams

梦享团队微信号

weCSdream

梦享团队官方微信公众号

梦享论坛团队

梦享团队新浪微博

第五章　图

考点 1 图的基本概念

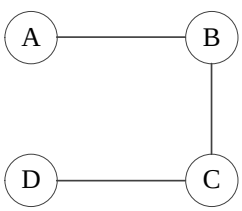
温馨提示：本部分考查图的基本概念，包括：1、顶点、边、度、完全图、子图、路径、连通图等基本概念；2、图的顶点集和边集的表示。本考点考查的都是图的基本内容，同学们稍微了解一下即可。

一．选择题部分

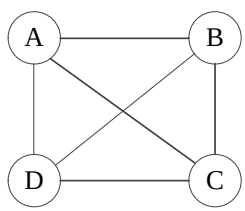
1. **（原书　第 1 题）**具有 n 个结点的连通图至少有（　　）条边。

A. n-1　　B. n　　C. n(n-1)/2　　D. 2n

【解析】数据结构某些定义比较复杂，我们简单化来理解。所谓的**连通图**，可以认为是任意两个结点之间都有通路的图。什么情况下，连通图的边数量最少呢？当然是把所有的结点都串联起来，而且还不构成圈（差一条边，如图 5.1(1)所示）的时候。对于 n 个结点的连通图，边数最少时为 n-1。



图（1）



图（2）

图 5.1

什么情况下，边数最多呢？既然连通图任意两个结点之间有边，那么边数最多时，任意两个结点之间都有一条边，我们称这样的图为无向完全图。如图 5.1(2)所示。对于有 n 个结点的连通图，边数最多时为 n(n-1)/2。

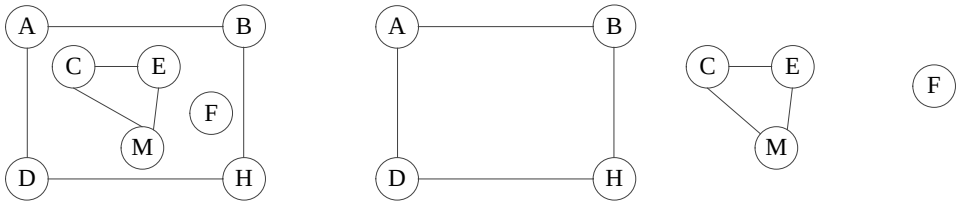
【参考答案】 A

2. **（原书　第 9 题）**连通分量指的是（　　）。

A. 无向图中的极小连通子图　　B. 无向图中的极大连通子图

C. 有向图中的极小连通子图　　D. 有向图中的极大连通子图

【解析】本题考查连通分量的基本概念。连通分量是针对无向图而言的，是无向图中的极大连通子图。如图 5.4 左图所示，无向图 G 有 3 个连通分量，分别如图 5.2 右图所示。



无向图G　　无向图G的三个连通分量

图 5.2

【参考答案】 B

考点 2 图的存储及基本操作

温馨提示：本考点考查图的邻接矩阵、邻接表、邻接多重表和十字链表四种存储结构及相应的空间复杂度。请同学们学会用这四种存储方式表示图。

一. 选择题部分

1. （原书　第 1 题）图的邻接矩阵表示法适用于表示（　　）。

A. 无向图　　B. 有向图

C. 稠密图　　D. 稀疏图

【解析】常见的图的存储方式有主要有两种：邻接矩阵和邻接表。**邻接矩阵属于顺序存储结构，邻接表属于链式存储结构**。对于无向图的邻接矩阵，具有以下性质：

(1). 矩阵是对称的，可压缩存储（上（下）三角）；

(2). 第 i 行或第 i 列中 1 的个数为顶点 i 的度；

(3). 矩阵中 1 的个数的一半为图中边的数目；

(4). 很容易判断顶点 i 和顶点 j 之间是否有边相连(看矩阵中 i 行 j 列值是否为 1)。

对于有向图的邻接矩阵，具有以下性质：

(1). 矩阵不一定是对称的，在箭头“**有去有回**”的情况下才对称，即两个顶点之间要么没有边，要么两个顶点都有到对方的边；

(2). 第 i 行中 1 的个数为顶点 i 的出度，第 i 列中 1 的个数为顶点 i 的入度；

(3). 矩阵中 1 的个数为图中弧的数目；

(4). 很容易判断顶点 i 和顶点 j 是否有弧相连。

邻接矩阵易于实现图的操作，n 个顶点需要n^2个单元存储边(弧)，空间效率为 O(n^2)，对稀疏图而言尤其浪费空间。故而，邻接矩阵一般用于存储稠密图。

【参考答案】 C

2. **(原书　第 7 题)** 已知由 7 个顶点组成的无向图的邻接矩阵为：

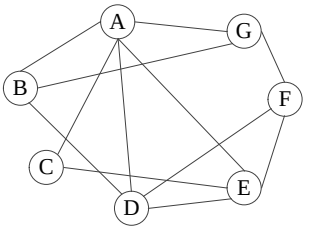
$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \\ G \end{array}\begin{array}{c} \begin{array}{ccccccc} A & B & C & D & E & F & G \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

则从顶点 A 出发进行深度优先遍历可以得到的序列是：（　　）

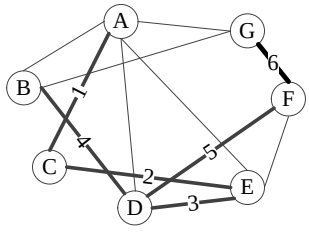
A. ACEDBFG　　B. ACDGFBE

C. AECDBGF　　D. ABDGFEC

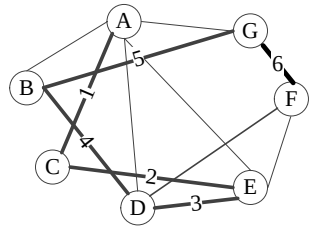
【解析】对于邻接矩阵，可能我们不能立即观察出一个深度或者广度优先遍历序列来。对于本题，我们画图 5.3 来解决邻接矩阵不便观察的问题。



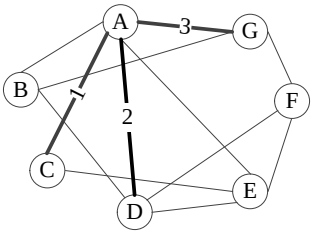
本题邻接矩阵对应的图



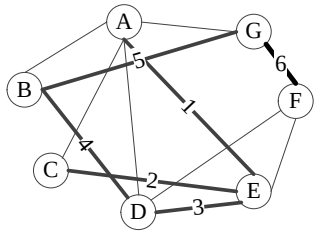
A答案对应的图



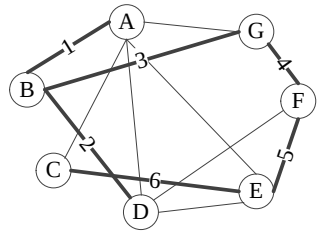
A答案正确的深度优先遍历序列



B答案对应的图



C答案对应的图



D答案对应的图

图 5.3

本题中，A、B、D 既不是深度优先遍历序列，也不是广度优先遍历序列。C 答案是深度优先遍历，是本题的答案。

【参考答案】 C

3. **（原书　第 11 题）**带权有向图 G 用邻接矩阵 A 存储，则顶点 i 的入度等于 A 中（　　）。

A. 第 i 行非无穷的元素之和

B. 第 i 列非无穷且非 0 的元素个数之和

C. 第 i 行非无穷且非 0 的元素个数

D. 第 i 行与第 i 列非无穷且非 0 的元素之和

【解析】对于邻接矩阵表示的有向图，第 i 行表示的是顶点V_i 的出度，而第 i 列表示的是顶点V_i 的入度。带权图不同于不带权图，带权图 A[i][j]=x 表示从V_i 到 V_j 的边的权值为 x，而在不带权图中，邻接矩阵中只有 0、1 两个值，值为 1 表示从V_i 到 V_j有边，为 0 则表示没有边。故而，第 i 列非无穷且非 0 元素的**元素个数**之和，即为顶点 A 的入度。

【参考答案】 B

二．综合应用题部分

1. **（原书　第 2 题）**已知某无向图的邻接表存储结构如图 5.4 所示。

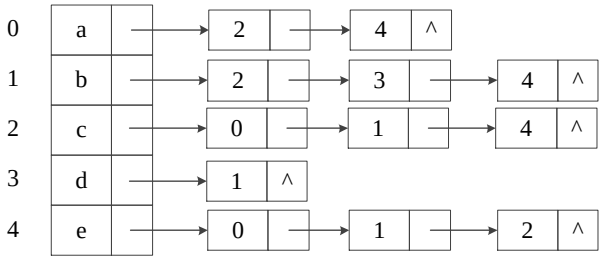


图 5.4

(1). 请画出该图。

(2). 根据存储结构给出其深度优先遍历序列及广度优先遍历序列。

(3). 画出其深度优先生成树及广度优先生成树。

【解析】依题意，有

(1). 根据该无向图的邻接表，可以得到该图如图 5.5 所示。

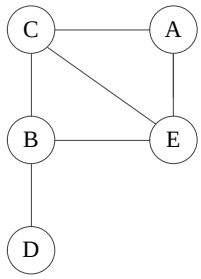
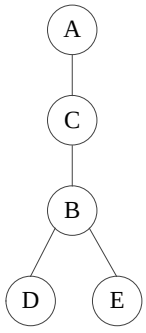


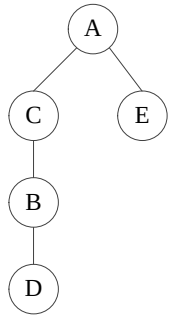
图 5.5

(2). 本题原题并未指出从哪一个顶点开始发出深度优先遍历和广度优先遍历。我们选择 A 作为遍历发出的起始结点，发出深度优先遍历得到序列：A、C、B、D、E，发出广度优先得到序列 A、C、E、B、D。

(3). 深度优先遍历生成树和广度优先遍历生成树如图 5.6 所示。



深度优先遍历生成树



广度优先遍历生成树

图 5.6

考点 3 图的遍历

温馨提示：本考点主要考查图的深度优先、广度优先搜索遍历和层次遍历的过程。请同学们掌握用邻接表和邻接矩阵存储表示下，图的深度优先遍历、广度优先遍历和层次遍历的方法和事件复杂度。

一．选择题部分

1. **（原书　第 5 题）**已知一有向图的邻接表存储结构如图 5.7 所示。

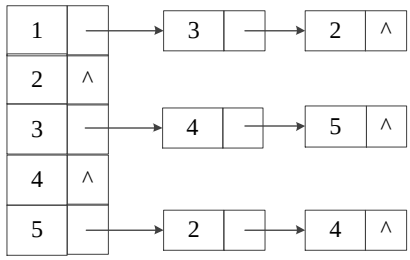


图 5.7

(1). 根据有向图的深度优先遍历算法，从顶点 v1 出发，所得到的顶点序列是(　　)。

A. v1,v2,v3,v5,v4　　B. v1,v2,v3,v4,v5

C. v1,v3,v4,v5,v2　　D. v1,v4,v3,v5,v2

(2). 根据有向图的宽度优先遍历算法，从顶点 v1 出发，所得到的顶点序列是(　　)。

A. v1,v2,v3,v4,v5　　B. v1,v3,v2,v4,v5

C. v1,v2,v3,v5,v4　　D. v1,v4,v3,v5,v2

【解析】为了说明深度优先遍历和广度优先遍历，我们给出了本题。请同学们多留意遍历方法。

(1). 从顶点 v1 出发，发起深度优先遍历，经过 v1→v3, 再 v3→v4，v4 没有后续顶点，转回 v3，v3 的后续顶点中还有 v5 没有被遍历过，于是找到 v3→v5，再由 v5→v2。因而，得到一个序列 v1→v3→v4→v5→v2。

【参考答案】 C

(2). 从顶点 v1 出发，发起广度优先遍历。因为从 v1 发出，有一条有向边 v1→v3。因此，有序列 v1→v3→v2，可有序列 v1→v3→v2→v4→v5，选择 B 答案。

【参考答案】 B

二. 综合应用题部分

1. **(原书　第 6 题)**图 5.8 是一带权有向图的邻接表法存储表示。其中，出边表中的每个结点均含有三个字段，依次为边的另一个顶点表中的序号、边上的权值和所指向下一个边结点的指针。

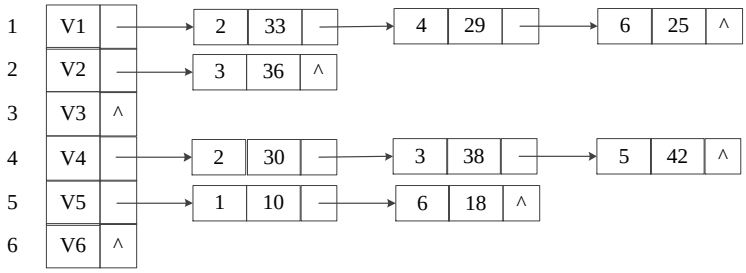


图 5.8

试求：

(1). 该带权有向图的图形。

(2). 从顶点 V1 为起点的广度优先遍历得到的顶点序列。

(3). 以顶点 V1 为起点的深度优先遍历的生成树。

(4). 由顶点 V1 到顶点 V3 的最短路径。

【解析】本题考查了图的邻接表存储下的深度优先遍历和广度优先遍历，并请考生给出最短路径的求法。

(1). 该有向图如图 5.9 所示。

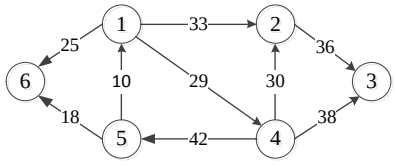


图 5.9

(2). 从顶点 V1 开始，发出广度优先遍历，可经过如图 5.10 所示的一序列顶点得到一个序列 V1→V2→V4→V6→V3→V5。

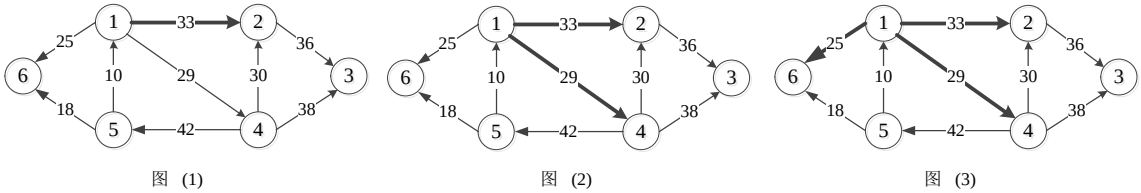


图 5.10

(3). 从顶点 V1 开始，发出深度优先遍历，可经过如图 5.11 的一序列顶点得到一棵生成树，如粗线部分所示。

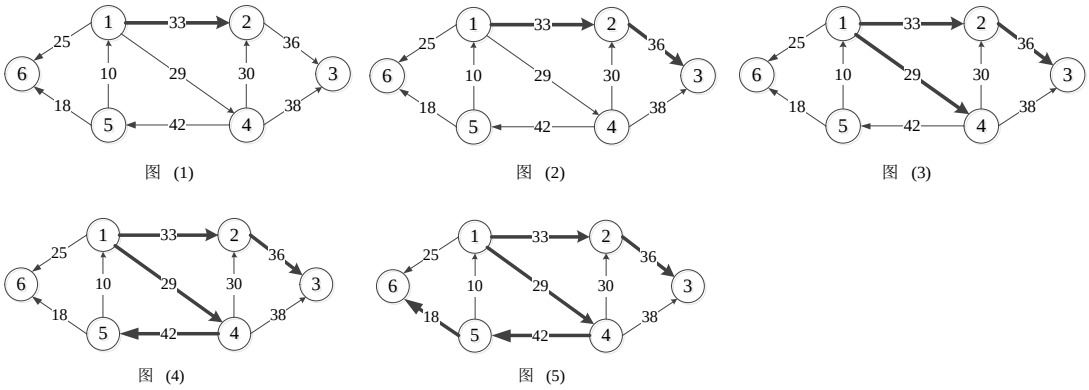


图 (1)　图 (2)　图 (3)

图 (4)　图 (5)

图 5.11

(4). V1 到 V3 只有 3 条路径，即 1→2→ 3、1→ 4 → 3与 1→ 4 → 2 → 3，权值分别是 69、67、95。所以，V1 到 V3 的最短路径是 V1→ V4 → V3，权值的大小为 67。

2. **（原书　第 7 题）**假设图采用邻接表的存储定义。下面是从图 G 的顶点 v 出发，对从 v 可达的顶点进行广度优先遍历的算法，请在空格处填上适当的语句，使算法完整。

```
void BFS(ALgraph G, int v)
  {
      int queue[MAXVER],front,rear;
      listnode *p;
      front=rear=0;
      printf(G.vexs[v].data);
      visited[v]=1;
      __________;
      while(__________)
        {
          v=queue[++front];
          p=__________;
          while(p!=NULL)
            {
              if(visited[p->adjvex]= =0)
                {
```

```
                    v=__________;
                    printf(G.vexs[v].data);
                    visited[v]=1;
                    queue[++rear]=v;
                }
            __________;
        }
    }
}
```

【解析】对邻接表进行广度优先遍历，首先第一个顶点入队列，如果队列不为空，则队头元素出队列，并让该队头元素对应链表的所有未访问的顶点入队列，如此循环，直到队列为空遍历结束，所得到的元素出队列顺序便是所求的广度优先遍历序列。

算法解析如下：

```
void BFS(ALgraph G, int v)
  { //从顶点 v 开始，对图 G 发起广度优先遍历
      int queue[MAXVER], front, rear;     //初始化队列，大小为 MAXVER
      listnode *p;
      front=rear=0;                  //初始化队列为空
      printf(G.vexs[v].data);
      visited[v]=1;                  //置已访问标记
      queue[++rear]=v;               //顶点 v 入队列
      while(__front!=rear__)         //队列不为空
        {
          v=queue[++front];     //队头元素出队列
          p=__G.vexs[v].first__;    //p 指向 v 结点对应链表的第一个邻接结点
          while(p!=NULL)             //v 链表所有未访问的结点入队列
              {
                  if(visited[p->adjvex]= =0) //经过一个结点时，看看是不是未访问
                      {                      //若访问过，则跳过，未访问则访问之
                          v=__p->adjvex__;
                          printf(G.vexs[v].data);     //输出该结点的值
```

```
                visited[v]=1;        //置访问标记
                queue[++rear]=v;    //入队列
            }
            p=p->next    ;
        }
    }
}
```

考点 4 图的应用

温馨提示：本考点主要考查图的应用，包括最小生成树、最短路径、拓扑排序和关键路径。本考点特别重要，非统考高校通常在该部分命制选择题、判断题、做图题和算法题。对于统考考生而言，也有可能命中选择题和综合题，请大家多注意，把基础知识掌握好。

考点 4.1 最小生成树

一. 选择题部分

1. **(原书　第 2 题)** 任何一个连通图的最小生成树（　　）。

   A. 只有一棵　　B. 有一棵或多棵

   C. 一定有多棵　　D. 可能不存在

【解析】任何一个连通图的最小生成树可能不止一棵，而且至少有一棵。

【参考答案】 B

2. **(原书　第 3 题)** 已知一个图 G 如图 5.12 所示，在该图的最小生成树中各边上数值之和为（　　）。

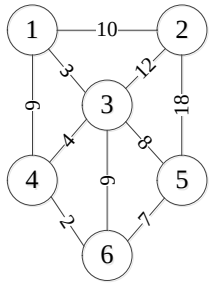


图 5.12

A.　21　　B.　26　　C.　28　　D.　33

【解析】我们可以利用克鲁斯卡尔算法构造最小生成树，过程如图 5.13 所示。

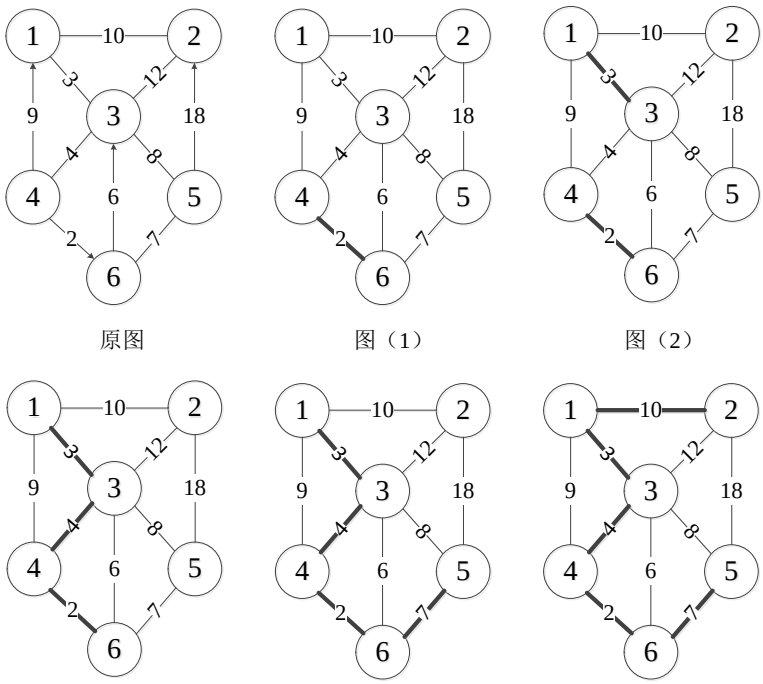


图 5.13

由图 5.13（5）可知，原图的最小生成树各边上的权值之和为 2+3+4+7+10=26，故而选择 B 答案。

【参考答案】 B

二．综合应用题部分

1.　**（原书　第 3 题）**对于如图 5.14 所示的带权无向图，用图示说明：

（1）利用 prim 算法从顶点 a 开始构造最小生成树的过程；

（2）利用 kruskal 算法构造最小生成树的过程。

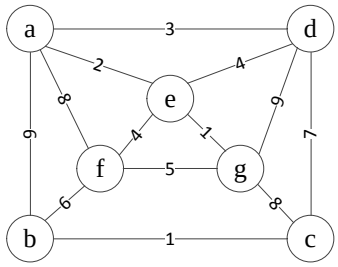


图 5.14

【解析】本题考查无向带权图在 Prim 和 Kruskal 算法下的最小生成树。

(1). 利用 Prim 算法从顶点 a 开始构造最小生成树的过程如图 5.15 所示。

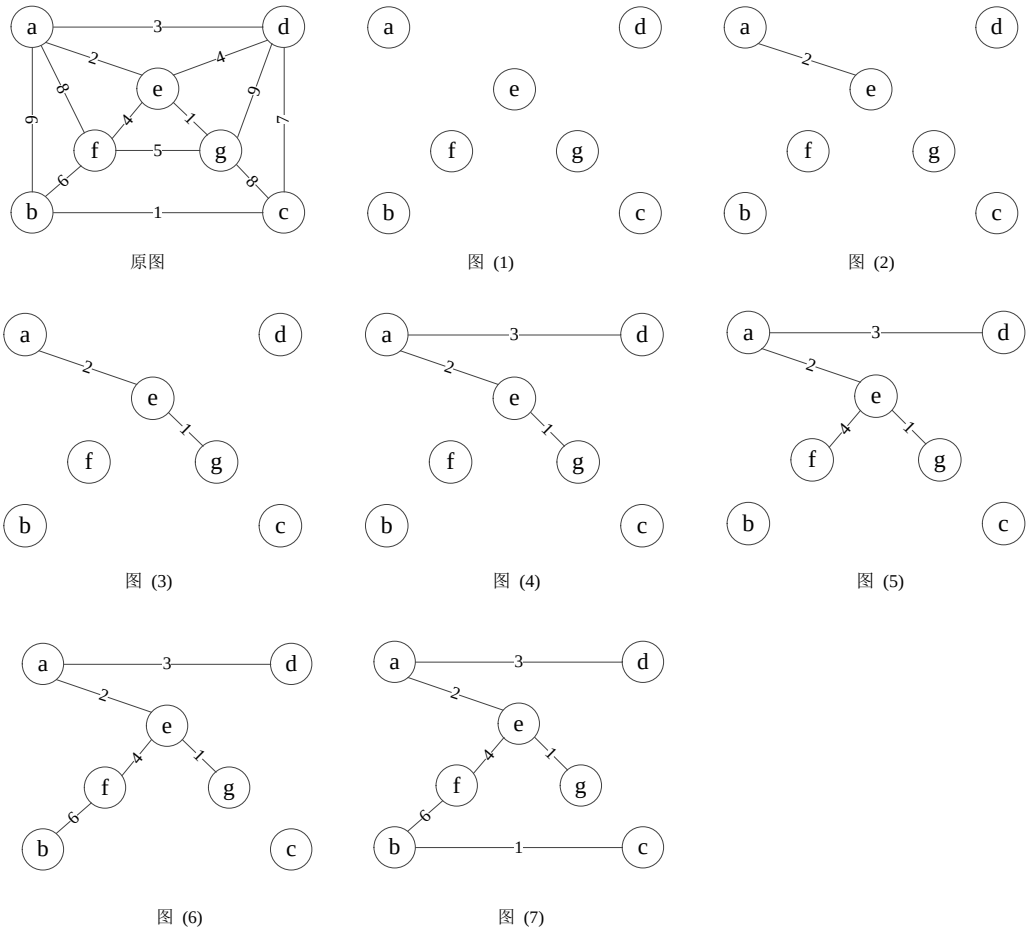


图 5.15

(2). 利用 Kruskal 算法构造最小生成树的过程如图 5.16 所示。

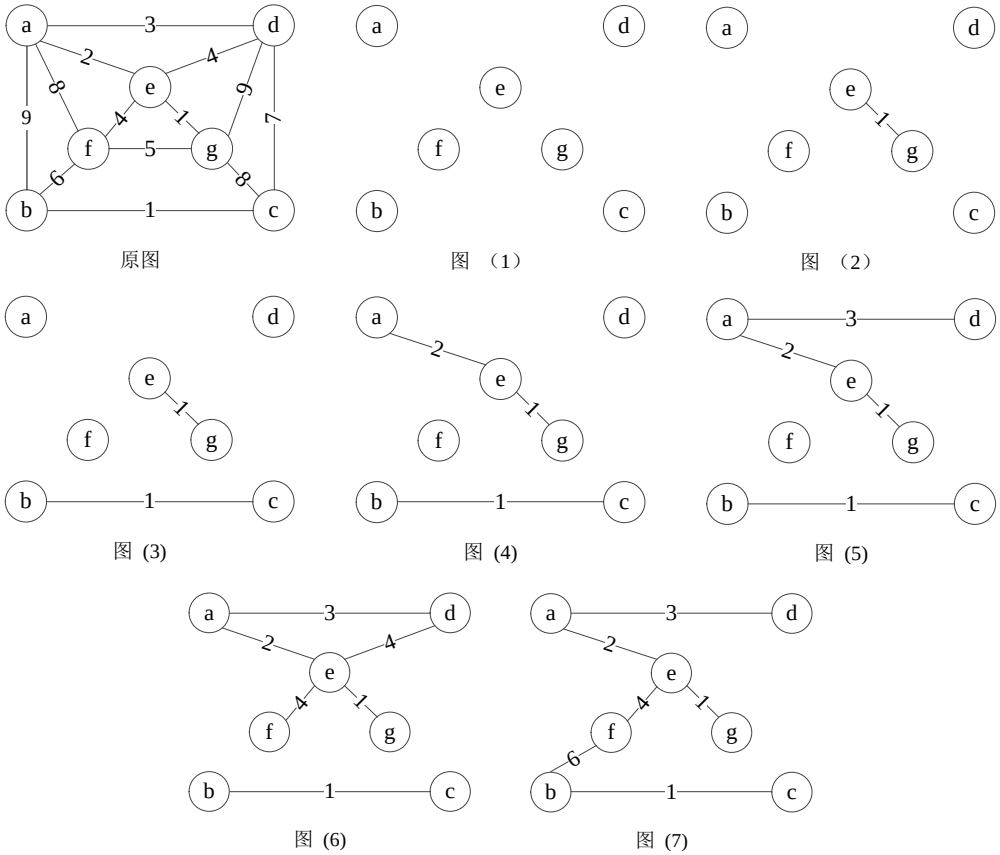


图 5.16

2. **（原书　第 7 题）**已知世界六大城市为:北京(Pe)、纽约(N)、巴黎(Pa)、 伦敦(L) 、 东京(T) 、 墨西哥(M),下表 5.1 给定了这六大城市之间的交通里程:

表 5.1　世界六大城市交通里程表(单位:百公里)

	pe	n	pa	L	T	M
Pe		109	82	81	21	124
N	109		58	55	108	32
Pa	82	58		3	97	92
L	81	55	3		95	89
T	21	108	97	95		113
M	124	32	92	89	113	

(1). 画出这六大城市的交通网络图;

(2). 画出该图的邻接表;

(3). 画出该图按权值递增的顺序来构造的最小(代价)生成树.

【解析】本题考查连通图、图的邻接表表示以及图 的最小生成树。

(1). 根据表 5.1 可知，六大城市的交通网络图刚好是一个无向完全图，如图 5.17 所示。

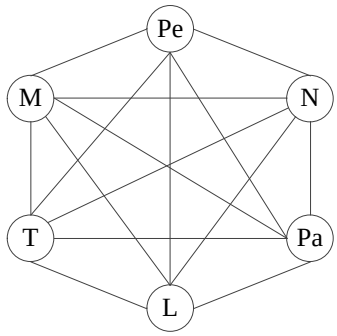


图 5.17

(2). 根据题意，我们设数组有两个域，即顶点域和指向第一个弧对应顶点的指针。我们还需再设顶点表结点有三个域：顶点信息、权值和指向下一条弧的指针。我们可以设计邻接表如图 5.18 所示。

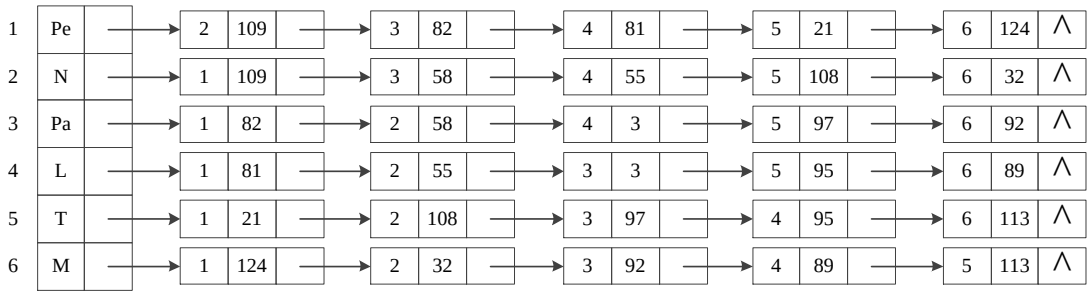


图 5.18

(3). 按照权值递增的顺序来构造最小生成树，这是克鲁斯卡尔算法的精髓。我们利用克鲁斯卡尔算法构造最小生成树，过程如下：

① Pa 与 L 之间的边的权值是 3，是最小的，选中这条边；

② T 和 PE 之间有一条边，权值为 21，是权值次小的边，选中该边；

③ N 与 M 有一条权值为 32 的边，是未选中的所有边中权值最小的边，选中该边；

④　L 与 N 之间存在一条权值为 55 的边，是当前未选中的所有边中权值最小的边，选中该边；

⑤　N 与 Pa 之间有一条权值为 58 的边，是未选中的所有边中权值最小的边，但若选中该边，则构成回路，该边不能选；

⑥　L 和 pe 之间存在一条权值为 81 的边，是目前系统中未选中的所有边中权值最小的边，选中该边。

最终得到的最小生成树如图 5.19 所示。

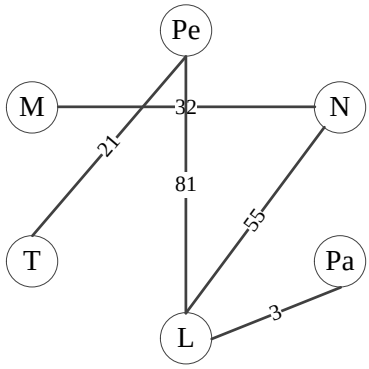


图 5.19

考点 4.2 最短路径

一．选择题部分

1. **(原书　第 1 题)**对于有 n 个顶点的有向图，由弗洛伊德（Floyd）算法求每一对顶之间的最短路径的时间复杂度是（　　）。

A.　O(1)　　B.　O(n)　　C.　O(n)　　D.　$O(n^3)$

【解析】**Floyd** 算法是一种动态规划算法，应用于稠密图效果最佳，边权可正可负。此算法简单有效，可以算出任意两个结点之间的最短距离。由于三重循环结构紧凑，时间复杂度为 $O(n^3)$。

【参考答案】 D

二．简答题部分

1. **（原书　第 2 题）**已知有向图的带权矩阵为：

$$\begin{bmatrix} \infty & 12 & \infty & 15 & 8 & \infty \\ \infty & \infty & 13 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty \\ 6 & \infty & 25 & \infty & \infty & 5 \\ \infty & \infty & \infty & 5 & \infty & 20 \\ \infty & \infty & 2 & \infty & 7 & \infty \end{bmatrix}$$

(1). 画出该有向图。

(2). 按 Dijkstra 算法,给出从顶点 1(顶点标号从 1 计)到其余顶点的最短路径长度以及经过的中间点。

(3). 画出该图邻接表存储结构示意图。

(4). 画出对应无向图的最小生成树，给出生成树边权之和。(如果去掉方向后,一对顶点之间有两条以上的边,只保留权值最小的边)

【解析】

(1). 本题中的矩阵对应的有向图如图 5.20 所示。

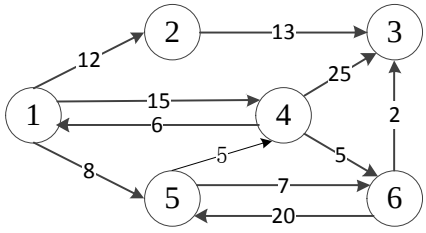


图 5.20

(2). 用 Dijkstra 算法求从顶点 1 开始的到其余顶点的最短路径（给出路径长度和中间点） 1->5 :8　1->2:12　1->(5)->4:13　1->(5-,4)6 :18　1->(5,4,6)->3:20.

(3). 画出邻接表，如图 5.21 所示。

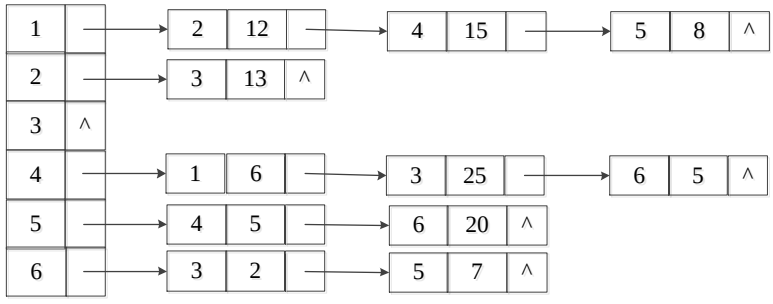


图 5.21

(4). 按照题目要求，如果去掉方向后,一对顶点之间有两条以上的边,只保留权值最小的边。那么，我们得到的无向图如图 5.22 所示。

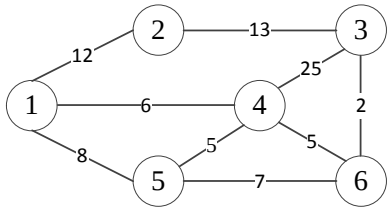


图 5.22

用 Kruskal 算法来构建最小生成树，如图 5.23 所示。

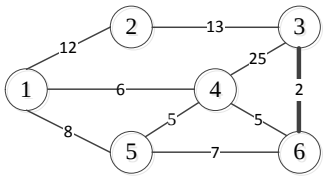


图 (1)

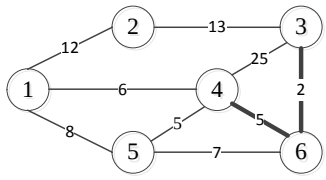


图 (2)

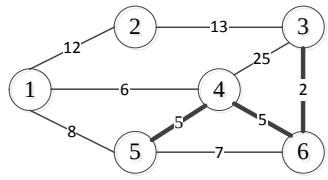


图 (3)

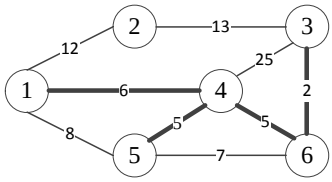


图 (4)

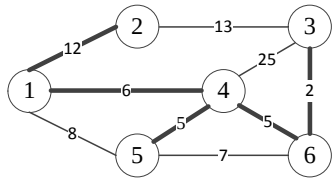


图 (5)

图 5.23

考点 4.3 拓扑排序

一. 选择题部分

1. **(原书　第 1 题)**下面哪一方法可以判断出一个有向图是否有环（回路）？（　　）。

A. 广度优先遍历　　B. 拓扑排序

C. 求最短路径　　D. 求关键路径

【解析】对有向图可用拓扑排序的方法判断是否有环，求最短路径与求关键路径都与环没任何关系。

在 AOV 网络中，如果顶点 Vi 的活动必须在顶点 Vj 的活动以前进行，则称 Vi 为 Vj 的前趋顶点，而称 Vj 为 Vi 的后继顶点，这种前趋后继关系有传递性。此外，任何活动 i 不能以它自己作为自己的前驱或后继，这叫做反自反性。

从前驱和后继的传递性和反自反性来看，AOV 网中不能出现回路(有向环)，回路表示顶点之间的先后关系进入了死循环。

判断 AOV 网是否有环的方法是对该 AOV 网进行拓扑排序，将 AOV 网中顶点排列成一个线性有序序列，若该线性序列中包含 AOV 网全部顶点，则 AOV 网无环。否则，AOV 网中存在环，该 AOV 网所代表的工程是不可行的。

【参考答案】 B

2. **（原书　第 5 题）**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, V5, V6, V4, V2, V7

C. V1, V3, V4, V5, V2, V6, V7　　D. V1, V2, V5, V3, V4, V6, V7

【解析】本题考查有向图的拓扑排序方法。本题中的图 G 如图 5.24（原图）所示。对图 G 进行拓扑排序，其中的一个排序过程如图 5.24 所示。

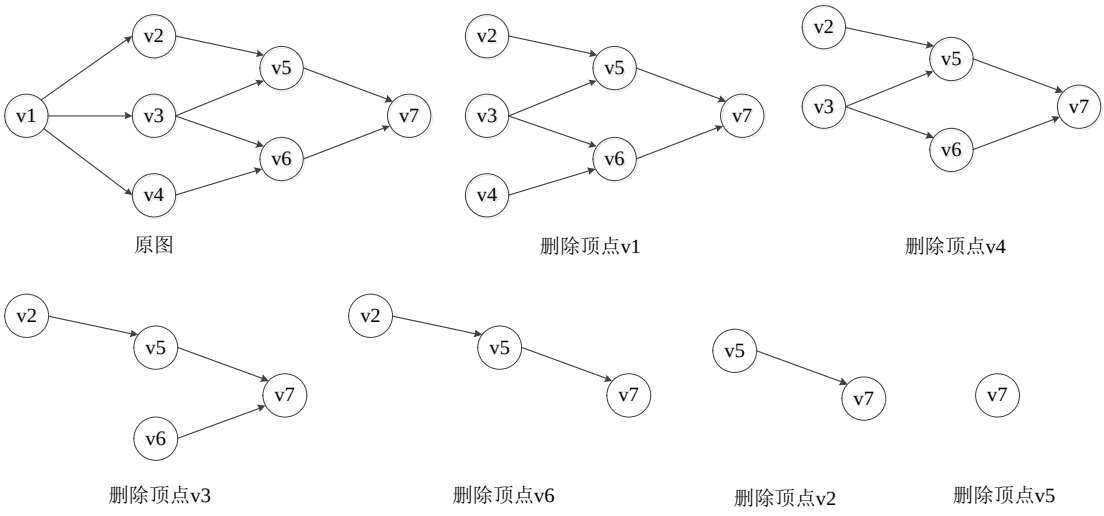


图 5.24

由图 5.24 可得到一个拓扑序列为 V1、V3、V4、V6、V2、V5、V7。当然，本题的图 G 的拓扑排序可以得到多个正确的序列，比如 V1、V2、V3、V4、V5、V6、V7 或 V1、V3、V4、V2、V6、V5、V7 等等。

【参考答案】 A

3. **(原书　第 7 题)**在有向图 G 的拓扑序列中，若顶点 Vi 在顶点 Vj 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 Vi 到 Vj 的路径

C. G 中没有弧<Vi,Vj>　　D. G 中有一条从 Vj 到 Vi 的路径

【解析】在有向图的拓扑排序中，若顶点 Vi 在顶点 Vj 之前，未必有 Vi 到 Vj 的路径，如在第 5 题中的序列（V1, V3, V4, V6, V2, V5, V7）中，V3 在 V4 之前，但是并不存在 V3 到 V4 的路径。故而，B 答案错误。而 V1 在 V7 之前，也并没有弧<V1, V7>，故而 A、C 都不一定成立。

若存在一条 Vj 到 Vi 的路径，那么 Vi 的完成，必须以 Vj 的完成为前提，但是拓扑排序中 Vi 在 Vj 之前。显然，如果假设成立的话，图 G 中存在环，不能进行拓扑排序。故而，D 答案不成立。

【参考答案】 D

考点 4.4 关键路径

一. 选择题部分

1. **(原书　第 2 题)**下面不正确的说法是（　　）。

(1). 在 AOE 网中，减小一个关键活动上的权值后，整个工期也就相应减小；

(2). AOE 网工程工期为关键活动上的权之和；

(3). 在关键路径上的活动都是关键活动，而关键活动也必在关键路径上。

A. （1）　　B. （2）　　C. （3）　　D. （1）、（2）

【解析】 在 AOE 网中，关键路径可能不止一条，关键路径上的活动都是关键活动，关键活动也必然在关键路径上，工程工期为关键活动上的权值之和。故而，并不是加快任何一个关键活动都可以缩短整个工程完成的时间，只有加快那些包括在所有的关键路径上的关键活动才能达到这个目的，A 答案错误。

AOE 网若有多条关键路径，则这多条关键路径的权值之和必然相等，而且最大。

【参考答案】 A

2. **（原书　第 3 题）**关于 AOE 网，下面的叙述中不正确的是（　　）。

A. 关键活动不按期完成就会影响整个工程的完成时间

B. 任何一个关键活动提前完成，将使整个工程提前完成

C. 所有关键活动都提前完成，则整个工程将提前完成

D. 某些关键活动若提前完成，将使整个工程提前完成

【解析】在 AOE 网中，并不是加快任何一个关键活动都可以缩短整个工程完成的时间，只有加快那些包括在所有的关键路径上的关键活动才能达到这个目的。也就是说，只有在不改变 AOE 网的关键路径的前提下，加快包含在关键路径上的关键活动才可以缩短整个工程的完成时间。

【参考答案】 B

二．综合应用题部分

1. **（原书　第 4 题）**试对下图 5.25 所示的 AOE 网络，解答下列问题。

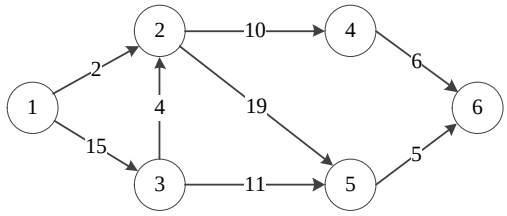


图 5.25

(1). 求每个事件的最早开始时间 Ve[i]和最迟开始时间 Vl[i]。

	1 ①	2 ③	3 ②	4 ④	5 ⑤	6 ⑥
Ve						
Vl						

(2). 求每个活动的最早开始时间 e( )和最迟开始时间 l( )。

	<1, 2>	<1, 3>	<3, 2>	<2, 4>	<2, 5>	<3, 5>	<4, 6>	<5, 6>
e								
l								
l-e								

(3). 确定哪些活动是关键活动。画出由所有关键活动构成的图，指出哪些活动加速可使整个工程提前完成。

(4). 这个工程最早可能在什么时间结束。

【解析】按拓扑有序的顺序计算各个顶点的最早可能开始时间 Ve 和最迟允许开始时间 Vl。然后再计算各个活动的最早可能开始时间 e 和最迟允许开始时间 l，根据 l - e 是否为 0 来确定关键活动，从而确定关键路径。

(1). 每个事件的最早开始时间 Ve[i]和最迟开始时间 Vl[i]如表 5.2 所示。

表 5.2

	①	2 ③	3 ②	4 ④	5 ⑤	6 ⑥
Ve	0	15	19	29	38	43
Vl	0	15	19	37	38	43

(2). 每个活动的最早开始时间 e( )和最迟开始时间 l( )如表 5.3 所示。

表 5.3

	<1, 2>	<1, 3>	<3, 2>	<2, 4>	<2, 5>	<3, 5>	<4, 6>	<5, 6>
e	0	0	15	19	19	15	29	38
l	17	0	15	27	19	27	37	38
l-e	17	0	0	8	0	12	8	0

(3). 关键活动为：<1, 3>，<3, 2>，<2, 5>，<5, 6>，加速这些活动可使整个工程提前完成。由所有关键活动构成的图如下图 5.26 所示。（关键路径为：<1, 3><3, 2><2, 5><5, 6>）

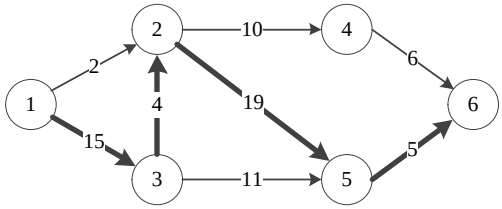


图 5.26

(4). 此工程最早完成时间为 43 天。

本章到此就结束了，请问您有什么疑问吗？任何问题，欢迎您与我们作者进行交流！

shareOurDreams
梦享团队微信号

weCSdream
梦享团队官方微信公众号

梦享论坛团队
梦享团队新浪微博

第六章　查找

考点 1 查找的基本概念

温馨提示：本部分考查查找的基本概念。

一．选择题部分

1. **(原书　第 2 题)**静态查找表和动态查找表的区别在于（　　）。

A. 前者是顺序存储，而后者是链式存储

B. 前者只能进行查找操作，而后者可进行查找、插入和删除操作

C. 前者只能顺序查找，而后者只能折半查找

D. 前者可被排序，而后者不能被排序

【解析】本题考查动态查找和静态查找的区别。动态查找表在查找过程中插入元素或者从查找表中删除元素，静态查找表只是查找特定元素或者检索特定元素的属性。通俗的说，动态查找表可以对查找表结构进行修改，而静态查找表只是查询。

静态查找的查找操作有：（1）查看某特定的关键字是否在表中（**判断性查找**）；（2）检索某特定关键字数据元素的各种属性（**检索性查找**）。这两种操作都只是获取已经存在的一个表中的数据信息，不对表的数据元素和结构进行任何改变，这就是所谓的静态查找。

动态查找的主要操作有：（1）首先也有一个“判断性查找”的过程，如果某特定的关键字在表中不存在，则按照一定的规则将其插入表中；（2）如果已经存在，则可以对其执行删除操作。动态查找的过程虽然只是多了“插入”和“删除”的操作，但是在对具体的表执行这两种操作时，往往并不是那么简单。

【参考答案】 B

2. **(原书　第 8 题)**要进行线性查找，则线性表__(1)__；要进行二分查找，则线性表__(2)__；要进行散列查找，则线性表__(3)__。某顺序存储的表格，其中有 90000 个元素，已按关键项的值的上升顺序排列。现假定对各个元素进行查找的概率是相同的，并且各个

元素的关键项的值皆不相同。当用顺序查找法查找时，平均比较次数约为 (4) ，最大比较次数为 (5) 。

(1)~(3):

A. 必须以顺序方式存储

B. 必须以链表方式存储

C. 必须以散列方式存储

D. 既可以以顺序方式，也可以以链表方式存储

E. 必须以顺序方式存储且数据元素已按值递增或递减的次序排好

F. 必须以链表方式存储且数据元素已按值递增或递减的次序排好

(4)~(5)：

A. 25000　　B. 30000　　C. 45000　　D. 90000

【解析】

(1). 顺序存储和链式存储方式都支持线性查找。

(2). 二分查找时，数据必须以顺序方式查找，而且必须有序。若是链式存储，则只能支持顺序查找。若是数据无序，则不能二分查找。

(3). 要进行散列查找，则元素必须以散列方式进行存储。

(4). 某顺序存储的表格，其中有 90000 个元素，已按关键项的值的上升顺序排列。现假定对各个元素进行查找的概率是相同的，并且各个元素的关键项的值皆不相同。当用顺序查找法查找时，平均比较次数约为表长的一半，即 45000。最坏的情况下，元素在表尾的位置，需要比较约 90000 次。

【参考答案】 (1) D (2) E (3) C (4) C (5) D

考点 2 顺序查找法

温馨提示：顺序查找法通常考查查找一个元素的平均查找长度。对于自主命题的高校而言，可能会有简单的编程题。请同学们掌握这些基础知识。。

一. 选择题部分

1. **(原书 第 3 题)**对于静态表的顺序查找法，若在表头设置岗哨，则正确的查找方式为（　　）。

A. 从第 0 个元素往后查找该数据元素

B. 从第 1 个元素往后查找该数据元素

C. 从第 n 个元素往开始前查找该数据元素

D. 与查找顺序无关

【解析】对静态表的顺序查找，通常在表头或者表尾设置**岗哨**，道理其实都是一样的。本题中，若在表头设置岗哨，则应该从表尾开始向表头方向查找，即从第 n 个元素开始向前查找数据元素，若查找成功，则返回该元素的位置。若返回的结果是 0，则表示读到了岗哨，查找失败。

【参考答案】 C

2. **(原书　第 5 题)**顺序查找不论在顺序线性表中还是在链式线性表中的时间复杂度为（　　）。

A. O(n)　　B. $O(n^2)$　　C. $O(n^{1/2})$　　D. $O(1og_2n)$

【解析】本题考查顺序查找的时间复杂度。无论是链式存储结构还是顺序存储结构，顺序查找都是从第一个元素开始，一直到表尾顺序查找元素，其查找的平均长度约为一半表长。当有 n 个元素时，需查找的平均长度约为 n/2，因而时间复杂度为 O(n)。

【参考答案】 A

考点 3 折半查找法

温馨提示：折半查找算法是本章的重点内容，也是数据结构的重点考点，主要考查：1、折半查找的条件；2、折半查找条件下的关键字比较次数、平均时间复杂度；3、折半查找树的建立。请同学们一定要把本考点的知识掌握，并运用自如。

一．选择题部分

1. **(原书　第 1 题)**对线性表进行折半搜索时，要求线性表必须（　　）。

A. 以链接方式存储且结点按关键字有序排列

B. 以数组方式存储

C. 以数组方式存储且结点按关键字有序排列

D.　以链接方式存储

【解析】本题考查折半查找只能在顺序存储结构上进行的特性。**折半查找**是一种高效的查找方法，它可以明显减少比较次数，提高查找效率。但是，折半查找的先决条件是查找表中的数据元素必须有序。除此之外，折半查找还要求数据以顺序存储方式存储在线性表中。（简单记忆为：**“顺序存储，数据有序”**。）

【参考答案】 C

2. **（原书　第 3 题）**有一个有序表为｛1，3，9，12，32，41，45，62，75，77，82，95，100｝，当折半查找值为 82 的结点时，（　　）次比较后查找成功。

A.　11　　B.　5　　C.　4　　D.　8

【解析】**本考点大部分题目都考查在有序表上利用折半查找算法查找元素的方法以及平均查找长度，我们不再赘述**。简单地画一个顺序表 A 来帮助我们分析，如表 6.1 所示。

表 6.1

数组下标	1	2	3	4	5	6	7	8	9	10	11	12	13
关键字	1	3	9	12	32	41	45	62	75	77	82	95	100

对上表进行折半查找元素 82，首先，$\lfloor(low+high)/2\rfloor=\lfloor(1+13)/2\rfloor=7$，显然 A[7]=45<82，故而，在 A[8]~A[13]中再进行折半查找。

此时，low=7+1=8，high=13，$\lfloor(low+high)/2\rfloor=\lfloor(8+13)/2\rfloor=10$，显然 A[10]=77<82，继续在 A[11]~A[13]中查找。

此时，low=10+1=11，high=13，因为$\lfloor(low+high)/2\rfloor=\lfloor(11+13)/2\rfloor=12$，A[12]=95>82，故而在 A[11]查找。因为 high=mid-1=12-1=11=low，A[low]=A[high]=82，查找成功。

【参考答案】 C

3. **（原书　第 15 题）**在有 11 个关键字的有序表中进行折半查找，查找失败时的最少比较次数和最多比较次数分别是（　　）。

A.　1 和 4　　B.　3 和 4

C.　1 和 3　　D.　4 和 5

【解析】 折半查找的过程为：给定值首先和处于待查区间"中间位置"的关键字进行比较，若相等，则查找成功，否则将查找区间缩小到"前半个区间" 或 "后半个区间" 之后继续进行查找。对 11 个关键字的有序表，构建其二分查找树如图 6.1 所示。

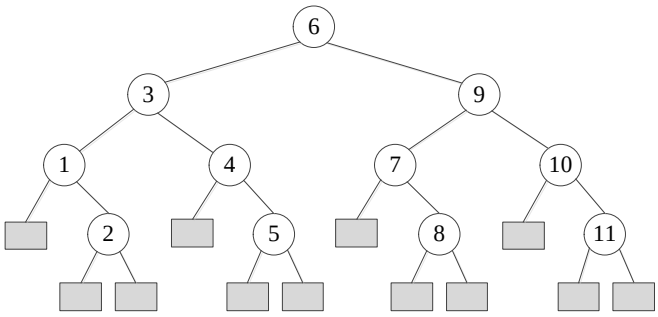


图 6.1

从上图可以看出，查找失败的最少比较次数为 3，最多比较次数为 4 次，故而选择 B 答案。要特别注意，有的书把失败的比较也算作一次比较，这里我们不算一次比较。

【参考答案】 B

二．综合应用题部分

1. **（原书　第 3 题）**设有序顺序表中的元素依次为 017, 094, 154, 170, 275, 503, 509, 512, 553, 612, 677, 765, 897, 908。试画出对其进行折半搜索时的二叉搜索树(即二分查找树)，并计算搜索成功的平均搜索长度和搜索不成功的平均搜索长度。

【解析】 对有序顺序表中的元素（017, 094, 154, 170, 275, 503, 509, 512, 553, 612, 677, 765, 897, 908）构建二分查找树，如图 6.2 所示。

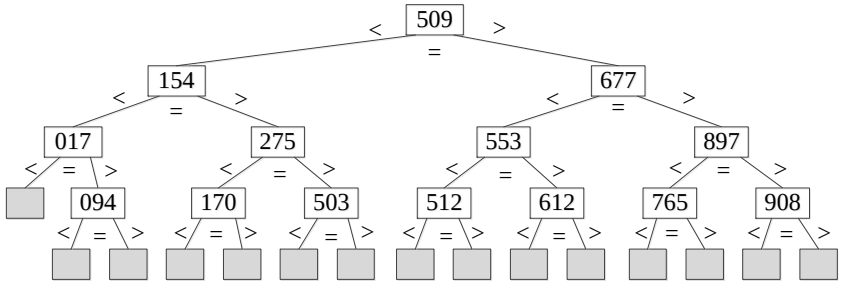


图 6.2

由上图可知，对该二分查找树的查找成功平均查找长度为（不算空的比较次数）

$$ASL_{succ}=\frac{1+2\times 2+3\times 4+4\times 7}{14}=\frac{45}{14}$$

对该二分查找树进行查找，其查找失败的平均查找长度为(不算空的比较次数)

$$ASL_{\text{unsucc}}=\frac{1\times 3+4\times 14}{15}=\frac{59}{15}$$

2. **（原书　第 5 题）**假定对有序表：（3，4，5，7，24，30，42，54，63，72，87，95）进行折半查找，试回答下列问题：

(1). 画出描述折半查找过程的判定树；

(2). 若查找元素 54，需依次与哪些元素比较？

(3). 若查找元素 90，需依次与哪些元素比较？

(4). 假定每个元素的查找概率相等，求查找成功时的平均查找长度。

【解析】

(1). 根据题目所给序列，设有序表在数组中开始存放的下标为 1，可以画出判定树如下图 6.3 所示（注：mid=⌊(1+12)/2⌋=6）；

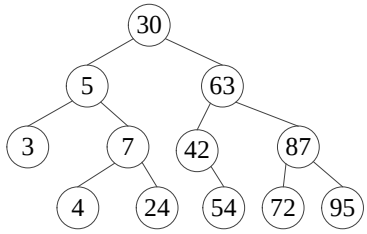


图 6.3

(2). 从有序表的折半查找判定树可知，查找元素 54，需依次与 30, 63, 42, 54 等元素比较；

(3). 查找元素 90，需依次与 30, 63,87, 95, 72 等元素比较。当然，有序表中并没有 90 这个元素，比较到 72 的右孩子，发现为空，则返回查找失败的信息；

(4). 首先，我们需要统计每个元素的查找次数。判定树的前 3 层共查找 1＋2×2＋4×3=17 次。但是，最后一层未满，不能用 8×4，只有 5 个结点，所以查找成功共用 5×4=20 次。所以，查找成功的平均查找长度为

$$ASL_{\text{succ}} = (1 + 2 \times 2 + 4 \times 3 + 5 \times 4) \times \frac{1}{12} = \frac{37}{12}$$

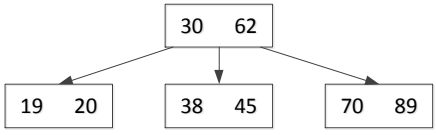
考点 4　B 树及其基本操作、B+树的基本概念

温馨提示：B-1 树部分，主要考查：1、考查方式是 B-树的基本概念；2、B-树的建立；3、结点插入和删除时，B-树的调整；4、B+树。本考点对考生提出的不是编程方面的要求，

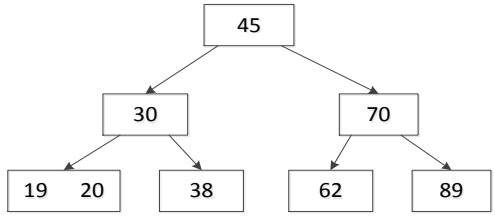
而是对 B-树的建立、插入和删除结点时对 B-树进行调整的手工操作。而且，手工操作很多考生掌握不到精髓，我们特别给了经典的总结，希望有助于大家掌握本考点的内容。

一．选择题部分

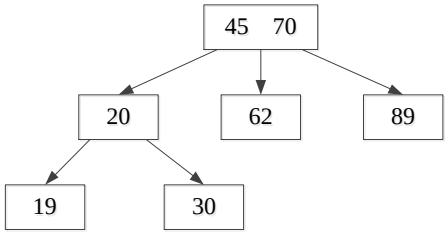
1. **（原书　第 2 题）**设输入序列为 20, 45, 30, 89, 70, 38, 62，19 依次插入到一棵 2-3 树中(初始状态为空)，该 B-树为（　　）。再删除 38，该 B-树为（　　）。



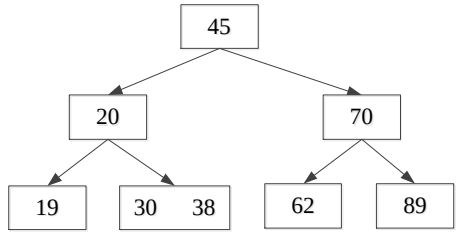
A.



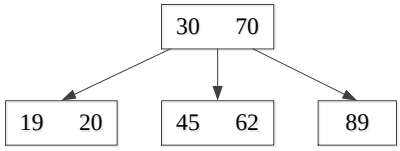
B.



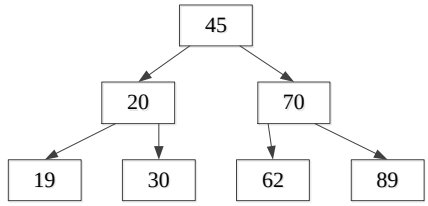
C.



D.

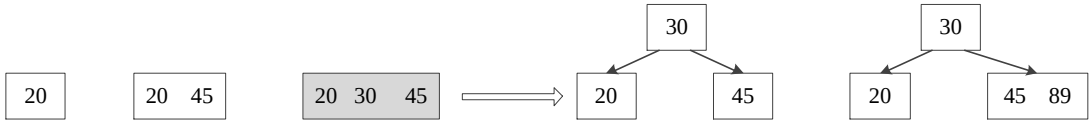


E.



F.

【解析】构建 B-树的过程如图 6.4 所示。



插入20　　插入45　　插入30，结点溢出　　结点分裂　　插入89

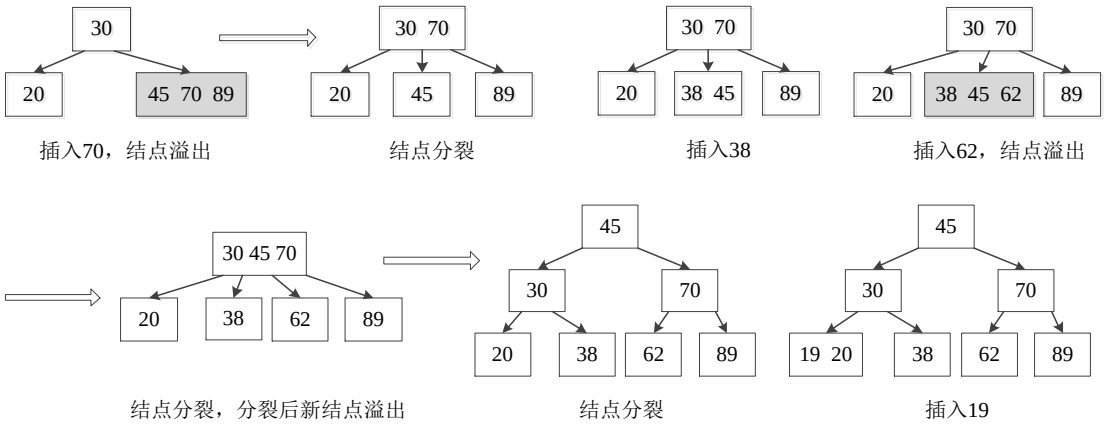


图 6.4

B-树的删除操作不同于插入操作，其步骤如下：

① 首先查找待删除关键字所在结点，并且要求删除之后，结点中的关键字个数不小于$\lceil m/2 \rceil - 1$;

② 否则，要从其左（或右）兄弟结点“借”关键字；

③ 若其左右兄弟都没有关键字可以借（结点中只有最少量的关键字），则必须进行结点的合并。

下面我们再来看看删除结点 38 之后，B-树的变化情况。如图 6.5 所示。待删除关键字 38 所在结点向其左兄弟接一个关键字，于是把父结点的 30“拉下来”，把兄弟结点里面最靠近自己的那个关键字往父结点“拉上去”，最后得到图 6.5（2）所示的 B-树。

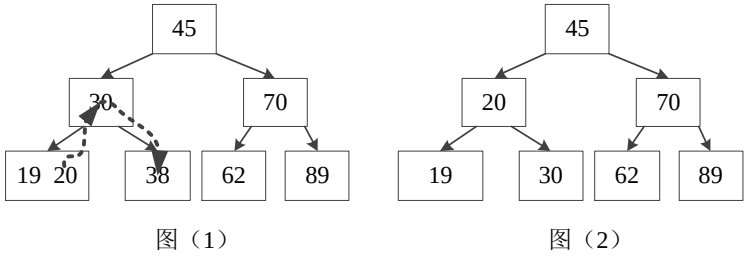


图 6.5

【参考答案】 B　F

2. **（原书　第 6 题）**下面关于 m 阶 B 树说法正确的是（　　）。

① 每个结点至少有两棵非空子树；

② 树中每个结点至多有 m−1 个关键字；

③ 所有叶子在同一层上；

④ 当插入一个数据项引起 B 树结点分裂后，树长高一层。

A. ①②③　　B. ②③　　C. ②③④　　D. ③

【解析】对于 m 阶 B 树，除了根结点至少要有两棵子树之外，其他非叶子结点至少有$\lceil m/2 \rceil$棵子树，故而①错误。树中，每个结点至多有 m-1 个关键字，而且所有叶子都在同一层上，故而②③显然正确。但是，插入一个关键字使得 B 树结点分裂，并不一定会引起树长高一层，如第 2 题中插入结点 70，B-树的前后高度都是 2，故而④错误。

【参考答案】 B

二．综合应用题部分

1. **（原书　第 2 题）**请画出如图 6.6 所示的 5 阶 B 树中插入一个关键码 360 后得到的 B 树。

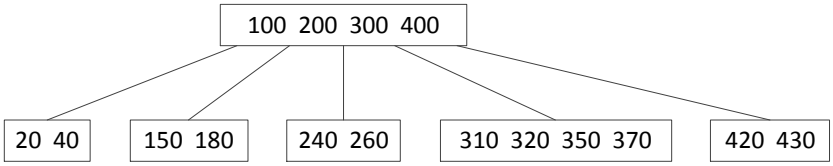


图 6.6

【解析】原来的 B-树如图 6.7 所示。

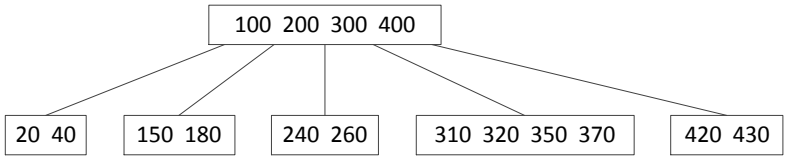


图 6.7

插入 360 之后，出现了两次结点溢出。我们先来看看第一次结点溢出。

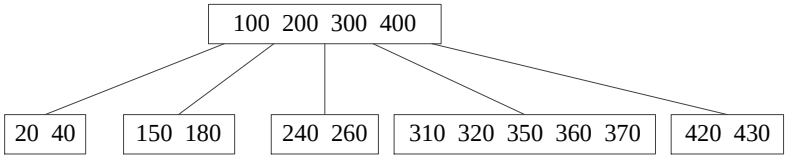


图 6.8

结点 310　320　350　360　370 溢出，从中间分裂，350 上提。于是，等到新的树。如图 6.9 所示。

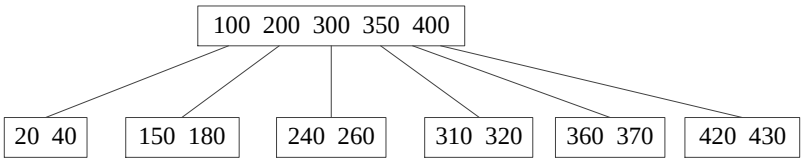


图 6.9

350 的上移，使得新的根结点溢出，需再分裂。如下图所示，结点 100　200　300　350　400 内的 300 上移，形成新的根结点。于是得到最终的 B-树如图 6.10 所示。

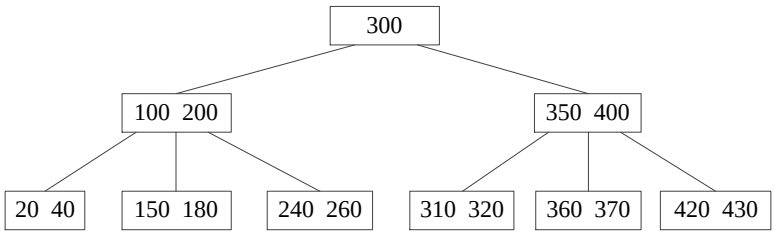


图 6.10

2.　**（原书　第 6 题）**已知 2 棵 B-树如下图 6.11(a)、6.11(b)所示（省略外结点）：

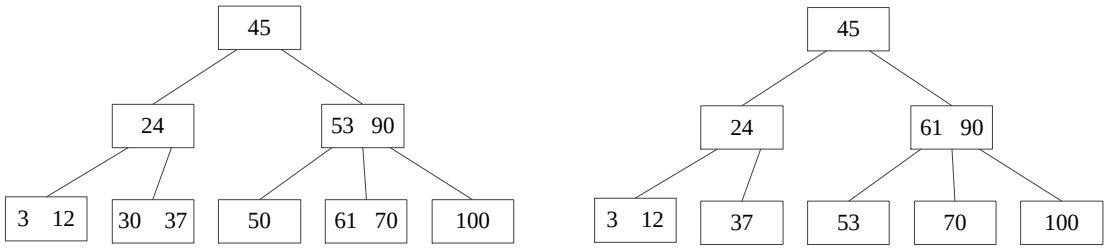


图 6.11

(1). 对树(a)，请分别画出先后插入 26、85 两个新结点后的树形。

(2). 对树(b)，请画出先后删除 53、37 两个结点后的树形。

【解析】本题考查 B 树的基本操作。

(1). 对树（a）,先后插入 26，85 两个新结点后的树形如图 6.12 所示。

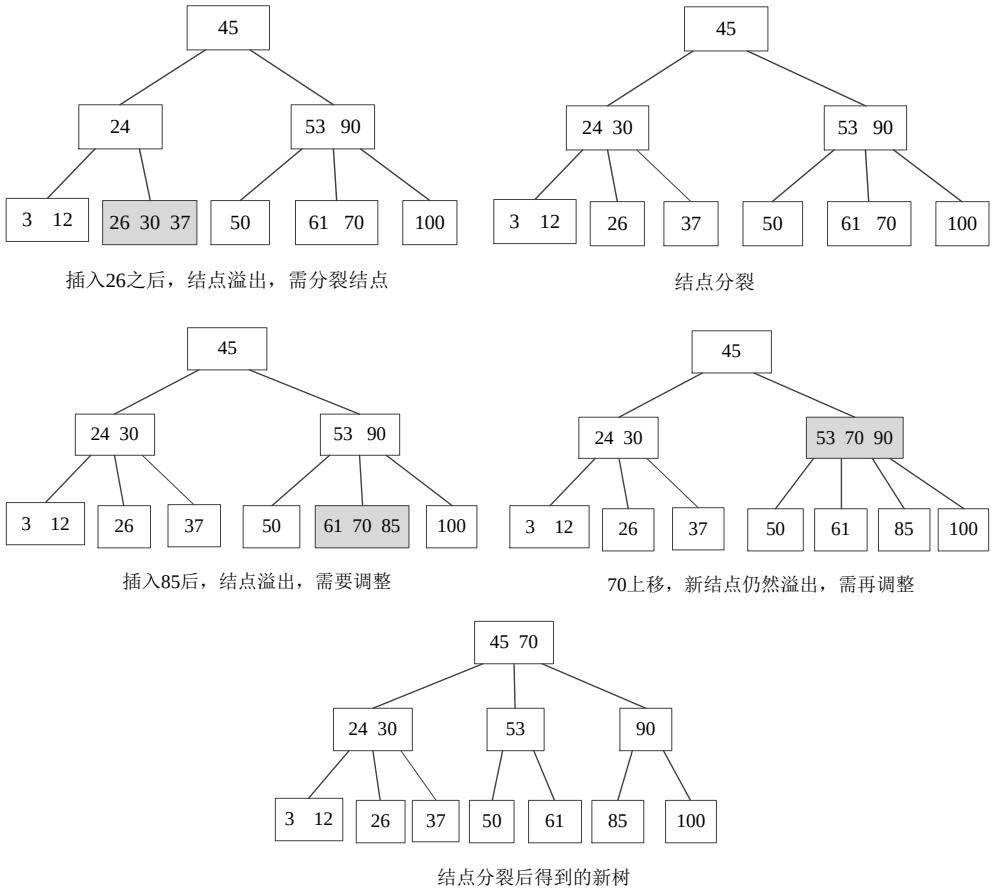


图 6.12

(2). 有些同学只会插，不会删。第二小题中，我们来看看删除结点中关键字的调整办法。

删除 53，其父结点有两个值，而兄弟结点只有一个值不够借，于是把父结点最靠近 53 的拉下来，得到新的 B-树，如图 6.13 所示。

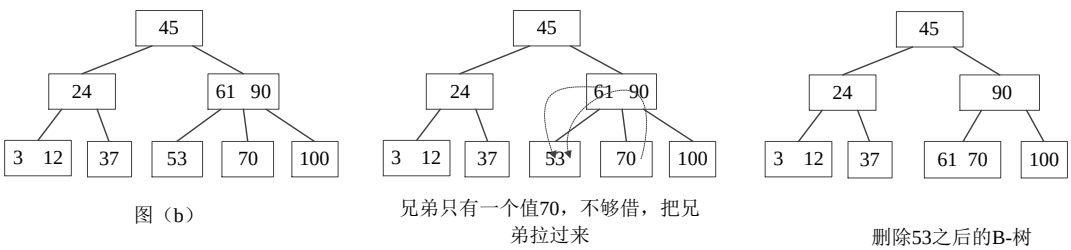
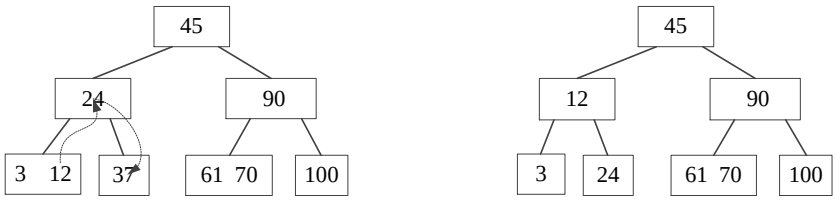


图 6.13

删除 37，其兄弟够借，于是把 24“拉下来”，把兄弟中最靠近 37 的值“拉”到父结点，如图 6.14 所示。



兄弟够借，把与兄弟之间的父节点值拉下来，把兄弟结点最近的值拉上去

删除37之后的B-树

图 6.14

考点 5　散列表

温馨提示：本考点主要命题有以下几种形式：1、什么是哈希表的特点？影响哈希表查找效率的因素有哪些？有哪些解决冲突的方法？2、怎么构建哈希表、怎么样利用冲突解决方案（如链地址法、二次探测再散列等）来解决冲突。这部分需要考生手工画图；3、计算哈希表的在查找成功和查找失败的情况下平均查找长度（或查找某一个元素的比较次数）。哈希表在 408 统考或者高校自主命题中，都可能出大题，请同学们务必掌握。

一．选择题部分

1. **（原书　第 1 题）**以下说法错误的是（　　）。

A. 散列法存储的思想是由关键字值决定数据的存储地址

B. 散列表的结点中只包含数据元素自身的信息，不包含指针。

C. 负载因子是散列表的一个重要参数，它反映了散列表的饱满程度。

D. 散列表的查找效率主要取决于散列表构造时选取的散列函数和处理冲突的方法。

【解析】散列表解决冲突的办法有多种，比如**线性探测法**、**平方探测法**、**再散列法**等开放定址法（即：可存放新表项的空闲地址既向其同义词表项开放，也向其非同义词表项开放）。当然，还有**拉链法**，拉链法的散列表中的结点除了包含元素自身信息，还会有其同义词的指针，而散列地址为 i 的同义词链表的头指针存放在散列表的第 i 个单元中。

【参考答案】 B

2. **（原书　第 6 题）**设哈希表长 m=14，哈希函数 H(key)=key%11。表中已有 4 个结点：

addr (15)=4; addr (38)=5; addr (61)=6; addr (84)=7

如用二次探测再散列处理冲突，关键字为 49 的结点的地址是（　　）。

A. 8　　B. 3　　C. 5　　D. 9

【解析】二次探测再散列法，简单地理解，就是先探测 H(key)=key mod 11，若成功，则得到 key 的地址，否则依次探测 H（key）=(key±i*i) mod 14，直到探测到 key 的地址为止。【注意，处理冲突为(mod 表长)】

因为 H（49）=49 mod 11=5，与 38 的地址发生冲突。再计算

$$H_1 = (5 + 1^2)\ mod\ 14 = 6$$

可知依然发生了冲突。再接着计算

$$H_1 = (5 - 1^2)\ mod\ 14 = 4$$

可知依然发生了冲突。再接着计算

$$H_2 = (5 + 2^2)\ mod\ 14 = 9$$

因为没有关键字存放在下标为 9 的位置，所以 49 可以放在下标为 9 的位置。

【参考答案】 D

3. **（原书　第 12 题）**下面关于哈希查找的说法，不正确的是（　　）。

A. 采用链地址法处理冲突时，查找一个元素的时间是相同的

B. 采用链地址法处理冲突时，若插入规定总是在链首，则插入任一个元素的时间是相同的

C. 用链地址法处理冲突，不会引起二次聚集现象

D. 用链地址法处理冲突，适合表长不确定的情况

【解析】**拉链法**解决冲突的做法是：将所有关键字为同义词的结点链接在同一个单链表中。若选定的散列表长度为 m，则可将散列表定义为一个由 m 个头指针组成的指针数组 T[0..m-1]。凡是散列地址为 i 的结点，均插入到以 T[i]为头指针的单链表中。T 中各分量的初值均应为空指针。

与开放定址法相比，拉链法有如下几个优点：

(1). 拉链法处理冲突简单，且无堆积现象，即非同义词决不会发生冲突，因此平均查找长度较短；

(2). 由于拉链法中各链表上的结点空间是动态申请的，故它更适合于造表前无法确定表长的情况；

(3). 开放定址法为减少冲突，要求装填因子 α 较小，故当结点规模较大时会浪费很多空间。而拉链法中可取 α ≥1，且结点较大时，拉链法中增加的指针域可忽略不计，因此节省空间。

显然，采用链地址法处理冲突时，因为还得加上在链表上查找元素的开销，所以查找一个元素的时间是不一样的。

【参考答案】 A

二．综合应用题部分

1. **（原书　第 3 题）**已知 Hash 函数为 H（K）=K mod 13 ，散列地址为 0 --14，用二次探测再散列处理冲突，给出关键字（23，34，56，24，75，12，49，52，36，92，06，55）在散列表中的分布，并求在等概率情况下查找成功的平均查找长度。

【解析】二次探测再散列不同于线性探测再散列，每一次加的都是 i 的平方，但是二者的原理大同小异。二次探测再散列的增量序列为：$d_i=1^2$，-1^2，2^2，-2^2，3^2，…，$\pm k^2$，($k\leqslant m/2$), 沿着增量序列选择不同的增量按照开放定址公式：

$$H_i=(H(key)+d_i)\ MOD\ m \quad i=1，2，\cdots，k\ (k\leqslant m-1)$$

寻找新的地址(构造后继散列地址)。

下面我们来计算题中所给元素在 H(K)=K mod 13，并在二次探测再散列 $H_i=(H(key)+d_i)mod\ 15$ 来处理冲突的情况下哈希表地址：

H(23)=23 mod 13=10；H(34)=34 mod 13=8；H(56)=56 mod 13=4；H(24)=24 mod 13=11；

H(75)=75 mod 13=10，与关键字 23 发生了冲突。使用二次探测再散列的办法，计算下一个 $H_1=(10+1^2)$ mod 15=11，此时又与关键字 24 冲突。再探测下一个位置 $H_2=(10-1^2)$ mod 15=9；

H(12)=12 mod 13=12；

H(49)=49 mod 13=10；与关键字 23 发生了冲突。使用二次探测再散列的办法，计算下一个 $H_1=(10+1^2)$ mod 15=11，此时又与关键字 24 冲突。再探测下一个位置 $H_1=(10-1^2)$ mod 15=9，又与关键字 75 冲突。继续探测下一个位置 $H_2=(49+2^2)$ mod 15=14；

H(52)=52 mod 13=0；

H(36)=36 mod 13=10；与关键字 23 发生了冲突。使用二次探测再散列的办法，计算下一个 $H_1=(10+1^2)$ mod 15=11，此时又与关键字 24 冲突。再探测下一个位置 $H_1=(10-1^2)$ mod

15=9，又与关键字 75 冲突。继续探测下一个位置 $H_2=(10+2^2) \bmod 15=14$，仍然冲突，继续计算下一个位置 $H_2=(10-2^2) \bmod 15=6$；

H(92)=92 mod 13=1；

H(06)=6 mod 13=6，与 36 发生冲突，探测 $H_1=(6+1^2) \bmod 15=7$；

H(55)=55 mod 13=3。

综上，可构造哈希表如表 6.2 所示。

表 6.2

地址	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
关键字	52	92		55	56		36	06	34	75	23	24	12		49
成功	1	1		1	1		5	2	1	3	1	1	1		4

上表的最后一行表示对应的关键字比较成功的次数，那么，在等概率条件下查找成功的平均查找长度为

$$ASL_{succ}=\frac{8\times 1+1\times 2+1\times 3+1\times 4+1\times 5}{12}=\frac{11}{6}$$

2. **（原书　第 6 题）**设散列表的地址范围是[0..9]，散列函数为 H（key）=（key ^{2} +2）MOD 9,并采用链表处理冲突，请画出元素 7、4、5、3、6、2、8、9 依次插入散列表的存储结构。

【解析】其实，我们解析了这么多关于散列表的选择题和综合题了，细心的同学会发现，具体的哈希函数和处理冲突的函数，我们都是分开写的，目的就是为了在这一个题中跟大家说明，这二者没有必然的关系。

链地址法（拉链法）将所有关键字为同义词的记录存放在一个单链表中，并用一维数组存放头指针。

下面，我们计算一下，看看每一个关键字都该挂到哪一条拉链上：

$H(4)=(4^2+2) \bmod 9=0$，$H(5)=(5^2+2) \bmod 9=0$，挂到头指针存放的数组下标为 0 的链表上；

$H(3)=(3^2+2) \bmod 9=2$，$H(6)=(6^2+2) \bmod 9=2$，$H(9)=(9^2+2) \bmod 9=2$，挂到头指针存放的数组下标为 2 的链表上；

$H(8)=(8^2+2) \bmod 9=3$，挂到头指针存放的数组下标为 3 的链表上；

H(2) = $(2^2 + 2) \bmod 9 = 6$，H(7) = $(7^2 + 2) \bmod 9 = 6$，挂到头指针存放的数组下标为 6 的链表上；

综上，利用拉链法得到关键字序列所构造的哈希表如图 6.15 所示。

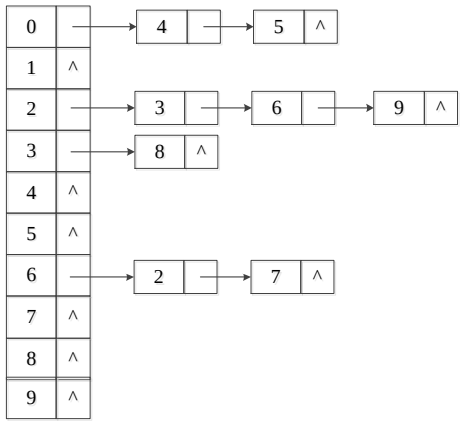


图 6.15

本章到此就结束了，请问您有什么疑问吗？任何问题，欢迎您与我们作者进行交流！

shareOurDreams
梦享团队微信号

weCSdream
梦享团队官方微信公众号

梦享论坛团队
梦享团队新浪微博

因为有你，所以有梦享！

梦享团队祝愿每一个考研人梦想成真！

第七章　排序

考点 1 排序的基本概念

温馨提示：本考点考查排序的基本概念。本考点历年统考和非统考高校命题都比较少，请同学们作一个简单的了解即可。

一．选择题部分

1. （原书　第 1 题）一个排序算法时间复杂度大小（　　）有关。

A. 不与所需移动记录的数目　　B. 与该算法的稳定性

C. 与所需比较关键字的次数　　D. 与所需辅助存储空间的大小

【解析】本题考查影响排序算法时间复杂度的因素。内部排序算法种类繁多，但就其排序所遵循的原则而言，大致可分为五大类：插入排序、交换排序、选择排序、归并排序和基数排序。算法性能主要从时间复杂度、空间复杂度和稳定性三个方面来比较。

一个算法的时间复杂度，与所需要比较的关键字次数、需要移动的记录数量都有关，但是与算法是否稳定没有关系，与所需要的辅助空间也没有关系。

【查缺补漏】排序算法的分类和基本原理

(1). 插入排序的原理：向有序序列中依次插入无序序列中待排序的记录，直到无序序列为空，对应的有序序列即为排序的结果，其主旨是“插入”。

(2). 交换排序的原理：先比较大小，如果逆序就进行交换，直到有序。其主旨是“若逆序就交换”。

(3). 选择排序的原理：从无序序列找关键字最小的记录，再放到已排好序的序列后面，直到所有关键字全部有序，其主旨是“选择”。

(4). 归并排序的原理：依次对两个有序子序列进行“合并”，直到合并为一个有序序列为止，其主旨是“合并”。

(5). 基数排序的原理：按待排序记录的关键字的组成成分进行排序的一种方法，即依次比较各个记录关键字相应“**位**”的值进行排序，直到比较完所有的“位”，即得到一个有序的序列。

各种排序方法的工作原理不同，对应的性能也有很大的差别，如表 7.1 所示。

【参考答案】 C

2. **（原书　第 2 题）**某内排序方法的稳定性是指（　　）。

A. 该排序算法不允许有相同的关键字记录

B. 该排序算法允许有相同的关键字记录

C. 平均时间为 O(nlogn)的排序方法

D. 以上都不对

【解析】本题考查内部排序算法稳定性的基本概念。排序算法的稳定性大家应该都知道，通俗地讲就是能保证排序前 2 个相等的数在序列的前后位置顺序和排序后它们两个的前后位置顺序相同。故而，A、B、C 答案的说法都是不正确的。

【参考答案】 D

考点 2 插入排序

温馨提示：本考点主要考查折半插入排序和直接插入排序。请同学们注意插入排序的复杂度、平均移动元素次数，并围绕这两个点展开复习。其实，希尔排序也是一种插入排序，但是我们参考 408 统考大纲，将该内容独立出来考查。

一. 选择题部分

1. **（原书　第 2 题）**n 个关键码排序，如果选用直接插入排序方法，则元素的移动次数在最坏情况下可以达到（　　）。

A. $\frac{1}{2}n^2$　　B. n(n-1)/2

C. n/2　　D. (n-1)/2

【解析】我们举一个例子，比如说，要对元素 5、4、3、2、1 这五个元素用直接插入的方法非递减排序。那么，4 插在 5 前面，需要移动一次元素。3 插在 4、5 前面，需要移动两次元素。2 要在 3、4、5 前面，需要将 3、4、5 这三个元素后移，才能给 2 腾出位置来。1 再插入到已有序的 2、3、4、5 序列中，需要把这 4 个元素都后移，才能腾出位置来放 1。故而，总共移动了 4+3+2+1=10 次。

当有 n 个元素时，最坏的情况下元素的移动次数为

$$N = 1 + 2 + \cdots + (n-2) + (n-1) = n(n-1)/2$$

故而，选择 B 答案。

【参考答案】 B

2. **（原书　第 6 题）**用直接插入排序方法对下面四个序列进行排序（由小到大），元素比较次数最少的是（　　）。

A. 94,32,40,90,80,46,21,69　　B. 32,40,21,46,69,94,90,80

C. 21,32,46,40,80,69,90,94　　D. 90,69,80,46,21,32,94,40

【解析】本题无需同学们一个一个选项去排序，太繁琐。插入排序在元素基本递增有序的情况下元素比较次数最少，为 n-1 次。在元素是呈递减序列的情况下元素比较和移动次数最多，为$n(n-1)/2$ 次。故而，本题中，C 答案已经基本递增有序，比较和移动次数应该是最少的。

【参考答案】 C

二．综合应用部分

1. **（原书　第 1 题）**以下为直接插入排序的算法。请分析算法，并在横线上填充适当的语句。

```
void straightsort(seqlist r,int n)
    {
        for(i=_____;i<=n;i++)
            {
                r[0]=r[i];
                j=i-1;
                while(r[0].key<r[j].key)
                    {
                        r[j+1]= _____;
```

```
                    j--;
                }
            r[j+1]= _____;
        }
    }
```

【解析】**插入排序的基本思想**是：插入第 i 个对象时，前面的 R[1],R[2],…,R[i-1]已经排好序，此时，用 R[i]的关键字与 R[i-1], R[i-2],…的关键字依次进行比较，找到插入位置即将 R[i]插入，插入位置及其后的元素**从后往前**依次后移。

对算法解析如下：

```
void straightsort(seqlist r,int n)
{                                          //对顺序表 r 进行直接插入排序
    for(i=__2__;i<=n;i++)                  //从第 2 个位置开始，把 2~n 的对象插入
        {
            r[0]=r[i];                     //复制 r[i]到 r[0]为哨兵
            j=i-1;
            while(r[0].key<r[j].key)       //待插入位置记录从后向前依次后移
                {
                    r[j+1]= __r[j]__;
                    j--;
                }
             r[j+1]= _____r[0]_____;   //插入到正确位置
        }
}
```

算法中引入附加记录 R[0]有两个作用：

其一是进入查找循环之前，它保存了 R[i]的副本，使得不至于因记录的后移而丢失 R[i]中的内容；

其二是在 while 循环“监视”下标变量 j 是否越界，一旦越界（即 j<1），R[0]自动控制 while 循环的结束，从而避免了在 while 循环内的每一次都要检测 j 是否越界（即省略了循环条件 j>=1）。因此，我们把 R[0]称为“监视哨”。

考点 3 冒泡排序

温馨提示：冒泡排序的命题方式有两种：1、围绕记录交换次数展开；2、让同学们简单地写一个冒泡排序程序（这种命题方式在 408 中可能性很小，自主命题高校出题的可能性比较大）。请同学们围绕常见的命题方式来复习。

一．选择题部分

1. （原书　第 1 题）设一组初始记录关键字序列为(Q，H，C，Y，P，A，M，S，R，D，F，X)，则按字母升序的第一趟冒泡排序结束后的结果是（　　）。

A. F，H，C，D，P，A，M，Q，R，S，Y，X

B. P，A，C，S，Q，D，F，X，R，H，M，Y

C. A，D，C，R，F，Q，M，S，Y，P，H，X

D. H，C，Q，P，A，M，S，R，D，F，X，Y

【解析】我们把这个题目当做一个大题来细讲冒泡排序算法，希望同学们掌握冒泡排序算法的精髓。我们对初始记录关键字序列为(Q，H，C，Y，P，A，M，S，R，D，F，X)按字母升序的第一趟冒泡排序，其过程如下：

初始序列：

Q	H	C	Y	P	A	M	S	R	D	F	X

第 1 个元素 H 与第 2 个元素 Q 进行比较，发现 Q 应该在 H 的前面，于是 Q 与 H 交换位置，得到新的记录序列，如下表所示。

H	Q	C	Y	P	A	M	S	R	D	F	X

第 2 个元素 Q 与第 3 个元素 C 进行比较，发现 Q 应该在 C 的后面，于是 Q 与 C 交换位置，得到新的记录序列，如下表所示。

H	C	Q	Y	P	A	M	S	R	D	F	X

第 3 个元素 Q 与第 4 个元素 Y 进行比较，发现 Q 应该在 Y 的前面，Q 与 Y 不需要交换位置。

H	C	Q	Y	P	A	M	S	R	D	F	X

第 4 个元素 Y 与第 5 个元素 P 进行比较，发现 Y 应该在 P 的后面，于是 Y 与 P 交换位置，得到新的记录序列，如下表所示。

H	C	Q	P	Y	A	M	S	R	D	F	X

第 5 个元素 Y 与第 6 个元素 A 进行比较，发现 Y 应该在 A 的后面，于是 Y 与 A 交换位置，得到新的记录序列，如下表所示。

H	C	Q	P	A	Y	M	S	R	D	F	X

第 6 个元素 Y 与第 7 个元素 M 进行比较，发现 Y 应该在 M 的后面，于是 Y 与 M 交换位置，得到新的记录序列，如下表所示。

H	C	Q	P	A	M	Y	S	R	D	F	X

第 7 个元素 Y 与第 8 个元素 S 进行比较，发现 Y 应该在 S 的后面，于是 Y 与 S 交换位置，得到新的记录序列，如下表所示。

H	C	Q	P	A	M	S	Y	R	D	F	X

第 8 个元素 Y 与第 9 个元素 R 进行比较，发现 Y 应该在 R 的后面，于是 Y 与 R 交换位置，得到新的记录序列，如下表所示。

H	C	Q	P	A	M	S	R	Y	D	F	X

第 9 个元素 Y 与第 10 个元素 D 进行比较，发现 Y 应该在 D 的后面，于是 Y 与 D 交换位置，得到新的记录序列，如下表所示。

H	C	Q	P	A	M	S	R	D	Y	F	X

第 10 个元素 Y 与第 11 个元素 F 进行比较，发现 Y 应该在 F 的后面，于是 Y 与 F 交换位置，得到新的记录序列，如下表所示。

H	C	Q	P	A	M	S	R	D	F	Y	X

第 11 个元素 Y 与第 12 个元素 X 进行比较，发现 Y 应该在 X 的后面，于是 Y 与 X 交换位置，得到新的记录序列，如下表所示。

H	C	Q	P	A	M	S	R	D	F	X	Y

上表也是对初始记录关键字序列为(Q，H，C，Y，P，A，M，S，R，D，F，X) 按字母升序的**第一趟**冒泡排序的结果。故而，选择 D 答案。

【参考答案】 D

二．综合应用题部分

1. **（原书　第 1 题）**下面程序段的功能是实现冒泡排序算法，请在下划线处填上正确的语句。

```
void bubble(int    r[n])
    {
      for(i=0;i<=n-2; i++)
          {
```

```
        exchange=0
        for(j=n-2;_____;j--)
            if (r[j+1]>r[j])
              {
                  temp=r[j+1];
                  _____;
                  r[j]=temp;
                  exchange=1;
              }
            if (exchange==0) return;
        }
    }
```

【解析】 **冒泡排序的基本思想**是：假设待排序的 n 个对象的序列为 R[0]、R[1]、...、R[n-1]，起始时排序范围是从 R[0]到 R[n-1]。在当前的排序范围之内，自右至左对相邻的两个结点依次进行比较，让值较大的结点往下移(下沉)，让值较小的结点往上移(上冒)。每趟起泡都能保证值最小的结点上移至最左边，下一遍的排序范围为从下一结点到 R[n-1]。

根据算法基本思想，我们对算法解析如下：

```
void bubble(int   r[n])
  {
      for(i=0;i<=n-2; i++)
       {
          exchange=0;                    //标志位，
          for(j=n-2;___j>=i___;j--)
            if (r[j+1]>r[j])
              {
                  temp=r[j+1];
                  __r[j+1]=r[j]___;
                  r[j]=temp;
                  exchange=1;
              }
            if (exchange==0) return;   //某一轮排序没有发生交换，说明已经有序
```

```
    }
}
```

考点 4　简单选择排序

温馨提示：本考点主要考查简单选择排序下的比较次数和移动次数、排序的时间复杂度和稳定性，也可能考查简单选择排序算法的编写，请同学们多注意这些常见的考查点。

一．选择题部分

1. **（原书　第 1 题）**从未排序序列中挑选元素，并将其依次放入已排序序列（初始时为空）的一端的方法，称为（　　）。

A. 归并排序　　B. 冒泡排序　　C. 插入排序　　D. 选择排序

【解析】本题考查选择排序算法的基本思想。排序方法中，从未排序序列中挑选元素，并将其依次放入已排序序列（初始时为空）的一端的方法，称为选择排序。

选择排序的基本思想是：将待排序的结点分为已排序(初始为空)和为未排序两组，依次将未排序的结点中值最小的结点插入已排序的组中。

而从未排序序列中依次取出元素与已排序序列（初始时为空）中的元素进行比较，将其放入已排序序列的正确位置上的方法，是插入排序，请同学们牢记这二者的区别。

常见的选择排序算法有直接选择排序（又叫做简单选择排序）和堆排序。

【参考答案】 D

2. **（原书　第 2 题）**采用简单选择排序，比较次数与移动次数分别为（　　）。

A. O(n), O(logn)　　B. O(logn), O(n*n)

C. O(n*n), O(n)　　D. 0(nlogn), O(n)

【解析】简单选择排序的比较次数 KCN 与对象的初始排列无关。设整个待排序对象序列有 n 个对象，则第 i 趟选择具有最小排序码对象所需的比较次数总是 n-i-1 次。总的排序码比较次数为：

$$KCN=(n-1) + (n-2) +\ldots\ldots+ 2 + 1 =n(n-1)/2$$

当这组对象的初始状态是按其关键字从小到大有序的时候， 对象的移动次数达到最少，此时 RMN = 0。最坏情况是每一趟都要进行交换，总的对象移动次数为 RMN = 3(n-1)。

故而比较次数时 O(n^2)，移动次数是 O(n)。

【经典补充】

利用数组实现时，有如下实现方法：

表 7.1

序号	1	2	3	4	5	6	7	8
数据	49	38	65	97	76	13	27	49

步骤如下：

(1). 先找到最小的数 13，和第一个数 49 交换；

(2). 在 2~8 中找到最小的数 27，和第二个数 38 交换；

(3). 在 3~8 中找到最小的数 38，与第三个数 65 交换；

…

如此进行下去，直到数组中的元素递增有序。

【参考答案】 C

二．综合应用题部分

1. 以下简单选择排序算法，请将算法补充完整:

```
void SelectSort(RecordType R[ ] , int length)
  {
     n=length;
     for ( i=1 ;   i<= n-1   ; i++)
        {
             _____;
             for ( j=i+1; j<=n ; j++ )
                if   (_____)
                     k=j;
             if (k!=i)
              {
                     x=R[i] ;
                     R[i]=R[k] ;
                     R[k]=x;
```

```
            }
        }
    }
```

【解析】简单选择排序是一种简单的排序方法，它的**基本步骤**是：

(1). 在一组对象 R[i]～R[n]中选择具有最小排序码的对象；

(2). 若它不是这组对象中的第一个对象，则将它与这组对象中的第一个对象对调；

(3). 在这组对象中去除这个具有最小排序码的对象。在剩下的对象 r[i+1]～r[n]中重复执行第(1)、(2)步，直到剩余对象只有一个为止。

对算法的解析如下：

```
void SelectSort(RecordType R[ ] , int length)
    {//对记录序列 R[1…n]作简单选择排序
        n=length;
        for ( i=1 ;   i<= n-1   ; i++)                  //共进行 n-1 趟排序
          {
                k=i          ;                          //记录下最小元素位置
                for ( j=i+1; j<=n ; j++ )                //在 R[i,…,n]中选择关键字最小的记录
                  if   (   R[j].key < R[k].key   )
                        k=j;
                if (k!=i)                                //与第 i 个记录交换
                  {
                        x=R[i] ;
                        R[i]=R[k] ;
                        R[k]=x;
                  }
           }
    }
```

考点 5　希尔排序

温馨提示：本考点考查希尔排序算法。希尔排序算法和基数排序算法的解题思路都挺单一，请同学们掌握解题思路，举一反三。

一．选择题部分

1. **（原书　第 2 题）**对记录的关键字为{50 , 26 , 38 , 80 , 70 , 90 , 8 , 30 , 40 , 20 }进行排序，各趟排序结束时的结果为：

50 , 26 , 38 , 80 , 70 , 90 , 8 , 30 , 40 , 20

50 , 8 , 30 , 40 , 20 , 90 , 26 , 38 , 80 , 70

26 , 8 , 30 , 40 , 20 , 80 , 50 , 38 , 90 , 70

8 , 20 , 26 , 30 , 38 , 40 , 50 , 70 , 80 , 90

其使用的排序方法是（　　）。

A.　快速排序　　B.　冒泡排序　　C.　希尔排序　　D.　插入排序

【解析】快速排序的第一个元素往往作为枢轴元素。第一趟排序完毕，枢轴元素大于其左边所有元素，小于其右边所有元素。很显然，题中的排序不符合快速排序的特点，排除 A 答案。

对于冒泡排序，第一趟排序结束，必有一个最大或者最小的元素落在最终位置。很显然，本题的排序方法也不是冒泡排序。

插入排序从未排序序列中依次取出元素与已排序序列中的元素进行比较，将其放入已排序序列的正确位置上，直到所有对象全部插入为止。很显然，本题也不是插入排序。

本题只剩下希尔排序了，事实上，本题是利用增量为 d=5、3、1 来对关键字{50 , 26 , 38 , 80 , 70 , 90 , 8 , 30 , 40 , 20 } 进行希尔排序。其排序过程如图 7.1 所示。

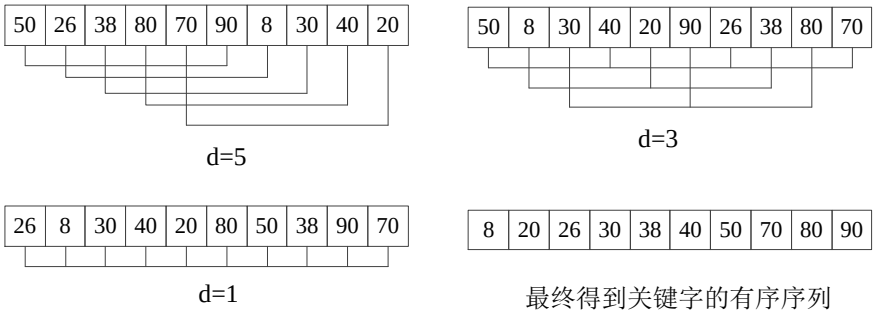


图 7.1

由上图可以看出，对关键字｛50 , 26 , 38 , 80 , 70 , 90 , 8 , 30 , 40 , 20 ｝进行希尔排序，各趟排序的结果跟题中所给的各趟排序结果一致，故而是希尔排序。

【参考答案】 C

2. **(原书　第 3 题)**对序列{15，9，7，8，20，-1，4，} 用希尔排序方法排序，经一趟后序列变为{15，-l，4，8，20，9，7}，则该次采用的增量是（　　）。

A. l　　B. 4　　C. 3　　D. 2

【解析】对序列{15，9，7，8，20，-1，4，} 用希尔排序方法排序，经一趟排序后序列变为{15，-l，4，8，20，9，7}。可以得知，-1 和 9 交换了位置，增量是 4 或者 2 或者 1。假设增量为 1，则-1 应该排在最前面的位置。再假设增量为 2，则 15、4、20、7 一组，其他记录是另一组，排序的结果应该是 4、-1、7、8、15、9、20，显然与题中的结果不符。故而增量 d=4。

【参考答案】 B

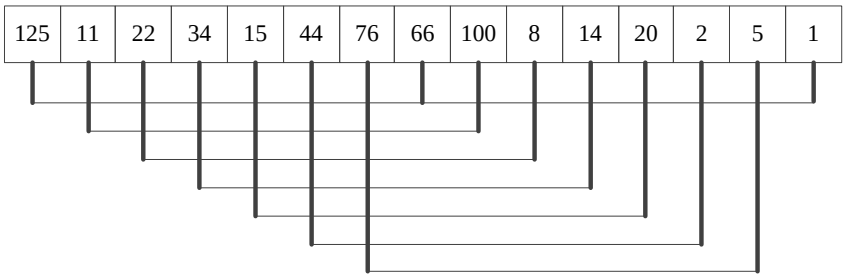
二．综合应用题部分

1. **(原书　第 2 题)**对下面数据表，写出采用 SHELL 排序算法排序的每一趟的结果，并标出数据移动情况。

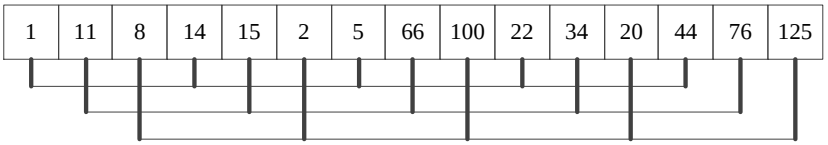
（125，11，22，　34，15，44，76，66，100，8，14，20，2，5，1）

【解析】将无序数组分割为若干个子序列，子序列不是逐段分割的，而是相隔特定的增量的子序列，对各个子序列进行插入排序；然后再选择一个更小的增量，再将数组分割为多个子序列进行排序……最后选择增量为 1，即使用直接插入排序，使数组最终有序。

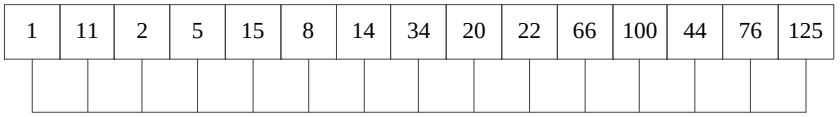
很多书首选增量为 n/2，以此递推，每次增量为原先的 1/2，直到增量为 1。其中，n 为初始记录的个数（n 比较大时，采用这种方法有点繁琐，但是我们做题时 n 基本不会很大）。本题也采用这种办法，分别取增量 d=7、3、1，进行希尔排序，其过程如图 7.2 所示。



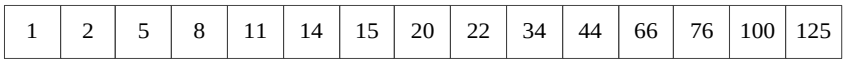
增量 d=7



增量 d=3



增量 d=1



希尔排序得到的有序序列

图 7.2

考点 6　快速排序

温馨提示：快速排序主要考查两点：1、快速排序算法的特点；2、快速排序算法实现；3、快速排序的过程或者一趟排序的结果。本考点历年考查很多，是复习的重点，请同学们务必掌握。

一．选择题部分

1. **（原书　第 1 题）**设一组初始记录关键字序列(5，2，6，3，8)，以第一个记录关键字 5 为基准进行一趟快速排序的结果为（　　）。

A. 2，3，5，8，6　　B. 3，2，5，8，6

C. 3，2，5，6，8　　D. 2，3，6，5，8

【解析】快速排序是对冒泡排序的一种改进。它的**基本思想**是：通过一躺排序将要排序的数据分割成独立的两部分，其中一部分的所有数据都比另外一部分的所有数据都要小，然后再按此方法对这两部分数据分别进行快速排序，整个排序过程可以递归进行，直到整个数据变成有序序列。

假设要排序的数组是 A[1]……A[N]，首先任意选取一个数据（通常选用第一个数据）作为关键数据，然后将所有比它的数都放到它前面，所有比它大的数都放到它后面，这个过程称为一躺快速排序。一躺快速排序的算法是：

(1). 设置两个变量 I、J，排序开始的时候 I：=1，J：=N；

(2). 以第一个数组元素作为关键数据，赋值给 X，即 X：=A[1]；

(3). 从 J 开始向前搜索，即由后开始向前搜索（J：=J-1），找到第一个小于 X 的值，两者交换；

(4). 从 I 开始向后搜索，即由前开始向后搜索（I：=I+1），找到第一个大于 X 的值，两者交换；

(5). 重复第 3、4 步，直到 I=J。

根据以上算法思想，对初始记录关键字序列(5，2，6，3，8)，设该序列存放在 R[5]数组中，以第一个记录关键字 5 为基准进行一趟快速排序，其过程如图 7.3 所示。

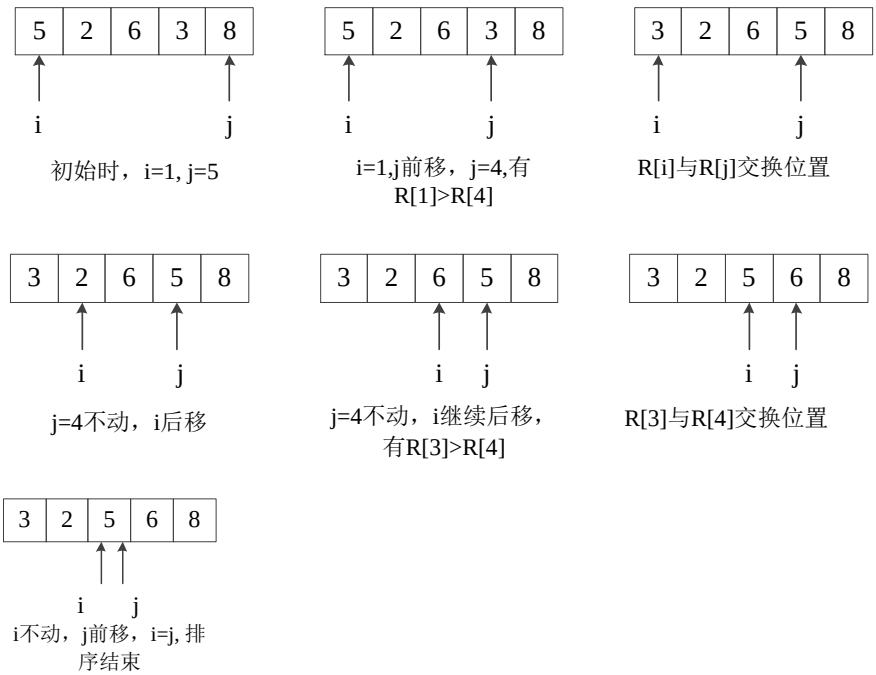


图 7.3

由图 7.3 可知，排序的结果为 3、2、5、6、8，故而选择 C 答案。

【参考答案】 C

2.　**（原书　第 9 题）**从未排序序列中选择一个元素，该元素将未排序序列分成前后两个部分，前一部分中所有元素都小于等于所选元素。后一部分中所有元素都大于等于所选元素，而所选元素处在排序的最终位置。这种排序方法称为（　　）排序法。

A.　插入排序　　B.　希尔排序　　C.　快速排序　　D.　堆排序

【解析】本题考查快速排序的算法思想。

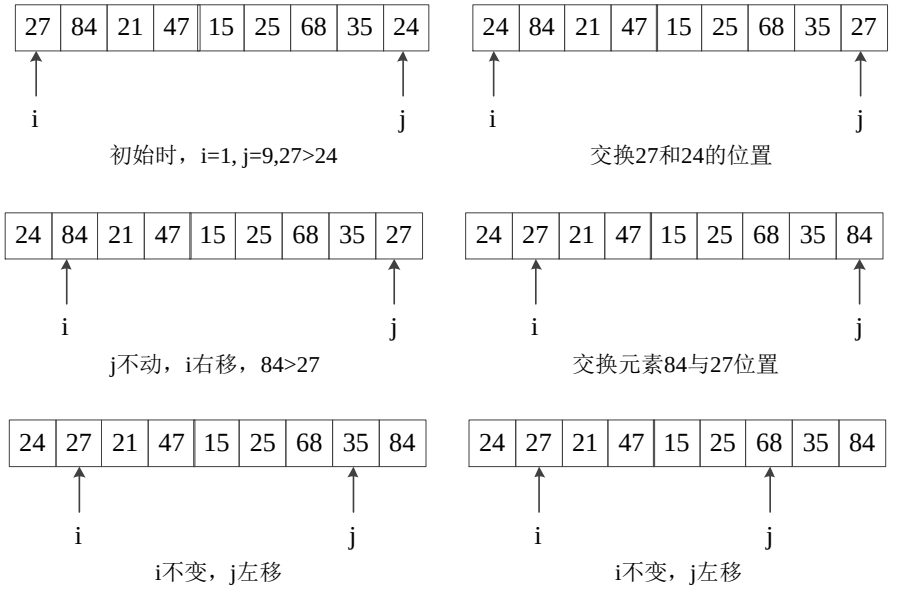
快速排序的基本思想是：通过一躺排序将要排序的数据分割成独立的两部分，其中一部分的所有数据都比另外一部分的所有数据都要小，然后再按次方法对这两部分数据分别进行快速排序，整个排序过程可以递归进行，以此达到整个数据变成有序序列。本题描述的正是快速排序算法的思想。

【参考答案】 C

二．综合应用题部分

1.　**（原书　第 1 题）**有一组键值 27,84,21,47,15,25,68,35,24,采用快速排序方法由小到大进行排序，请写出每趟的结果。

【解析】在选择题部分，我们都只给出了一趟排序的结果。那么，对整个序列进行快速排序，是什么样的呢？我们来看看这个题目的详细排序过程，如图 7.4 所示。



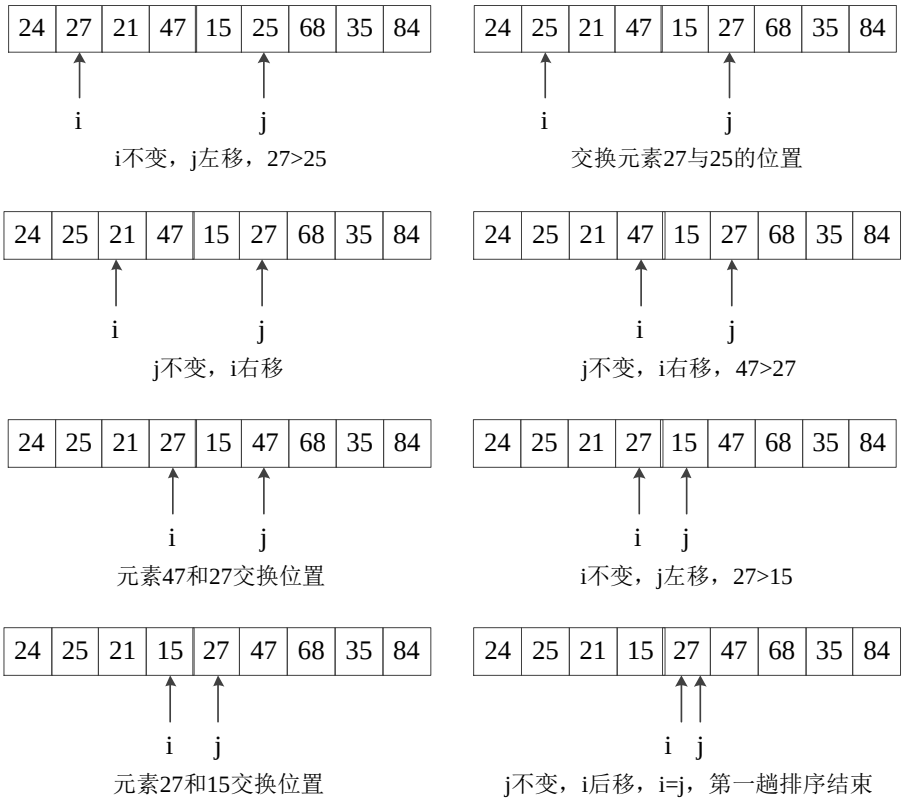
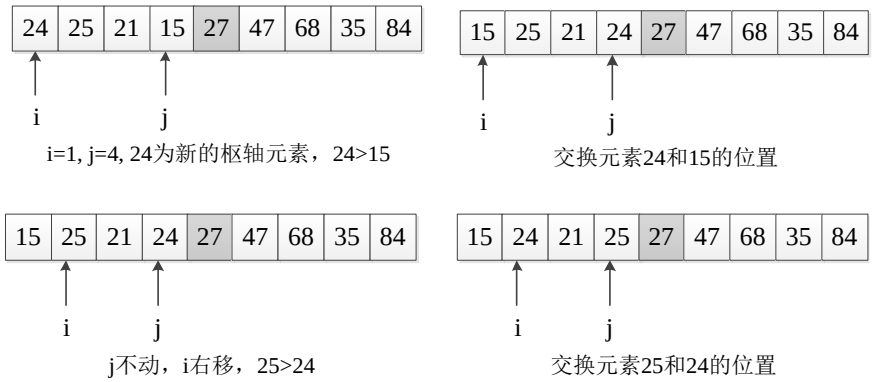


图 7.4

到此，第一次划分结束。为了方便大家掌握过程，也为了方便我们画图，我们先画枢轴元素 27 左边序列，再画其右边序列的快速排序。本章我们花了大量时间来画图，这些图来之不易啊。我们接下来进一步排序，过程如图 7.5 所示。



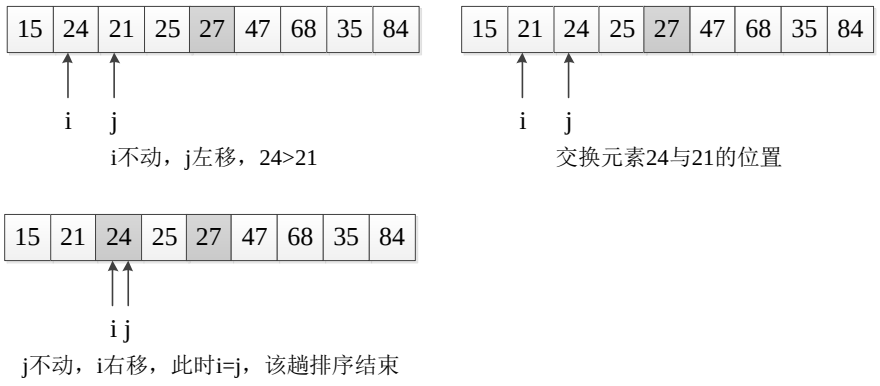


图 7.5

对此，元素 27 的左边关键字序列都已经有序，再对 27 的右边关键字序列进行排序，其过程如图 7.6 所示。

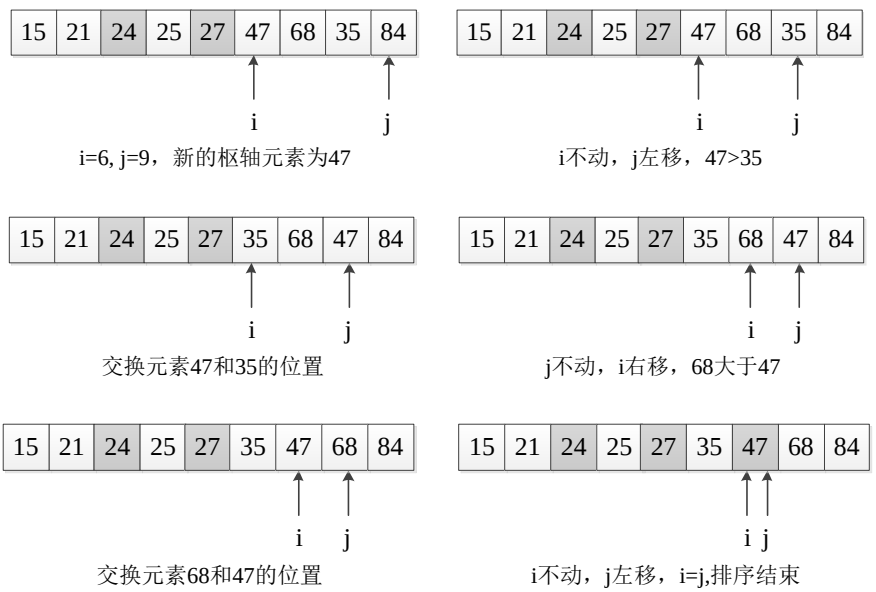


图 7.6

【特别注意】请同学阅读快速排序算法，有些书枢轴元素是最后（即 i=j 时）才插入的，有些书则每次交换均交换枢轴元素的位置。我们在以上过程中每一步都用到了枢轴元素，只是为了方便大家观察。

考点 7 堆排序

温馨提示：堆排序主要命题有两种方式：1、堆排序的特点；2、堆排序的过程（要求同学们手工操作），包括堆的建立和输出、调整过程。堆排序过程希望同学们会手工操作。

一．选择题部分

1. （原书　第 2 题）堆是一个键值序列{K1, K2, ... , Ki, ... , Kn}，对 i=1,2,...,$\lfloor n/2 \rfloor$，小顶堆满足（　　）。

   A. $K_i <= K_{2i}$ 且 $K_i <= K_{2i+1}$ (2i+1<=n)

   B. $K_i < K_{2i} < K_{2i+1}$

   C. $K_i <= K_{2i}$ 或 $K_i <= K_{2i+1}$ (2i+1<=n)

   D. $K_i <= K_{2i} <= K_{2i+1}$

【解析】根据最大堆和最小堆的定义可知，根结点（亦称为堆顶）的关键字是堆里所有节点关键字中最小者的堆成为最小堆，根结点（亦称为堆顶）的关键字是堆里所有节点关键字中最大者的堆成为最大堆。

根据堆的定义，对于每一个结点 i，若其左右孩子存在，分别为 Key[2i]和 Key[2i+1]，则满足 Key[i]>=Key[2i]&&key>=key[2i+1] 称为大顶堆，满足 Key[i]<= key[2i+1] && Key[i] <=key[2i+2] 称为小顶堆。

换句话说，大顶堆满足只要一个结点存在左右孩子结点，则必须满足该结点大于左孩子结点且大于右孩子结点。小顶堆满足只要一个结点存在左右孩子结点，则必须满足该结点小于左孩子结点且小于右孩子结点。故而，本题选择 A 答案。

【参考答案】 A

2. （原书　第 4 题）假定对元素序列（7, 3, 5, 9, 1, 12）进行堆排序，并且采用小根堆，则由初始数据构成的初始堆为（　　）。

   A. 1, 3, 5, 7, 9, 12　　B. 1, 3, 5, 9, 7, 12

   C. 1, 5, 3, 7, 9, 12　　D. 1, 5, 3, 9, 12, 7

【解析】对于堆排序，最重要的两个操作就是构造初始堆和调整堆，其实构造初始堆事实上也是调整堆的过程，只不过构造初始堆是对所有的非叶节点都进行调整。

借助本题，我们来讲解一下建立初始堆的过程，如下：

第一步：利用元素序列（7, 3, 5, 9, 1, 12）构造完全二叉树，如图 7.7 所示。

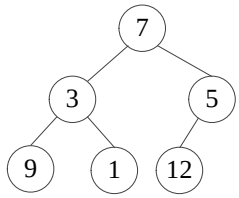


图 7.

第二步：构造初始堆，从最后一个非叶子节点开始调整，调整过程如图 7.8 所示。

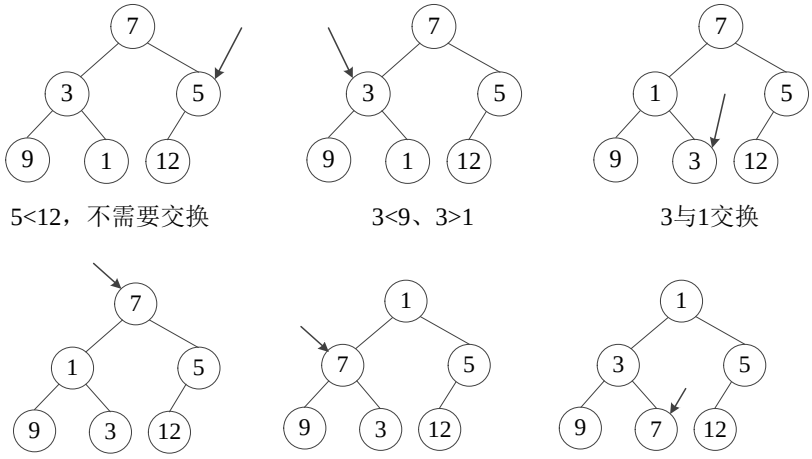


图 7.8

由图 7.8 得到初始堆为 1、3、5、9、7、12，故而选择 B 答案。

【参考答案】 B

3. **（原书　第 5 题）**一组记录的排序码为（46，69，56，38，40，84），则利用堆排序的方法建立大顶堆的初始堆为（　　）。

A. 69，46，56，38，40，80　　B. 84，69, 56，38，　40，46

C. 84，69，56，46，40，38　　D. 84，56，69，40，46，38

【解析】本题和第 4 题的道理一样，只是构建的是大顶堆的初始堆而已。我们也给出详细的初始堆构建过程，如下图所示。

首先，用题中所给的序列 46，69，56，38，40，84 构造完全二叉树，如图 7.9 所示。

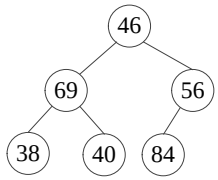


图 7.9

然后，调整上图中的堆，使其成为大顶堆，对应的调整过程如图 7.10 所示。

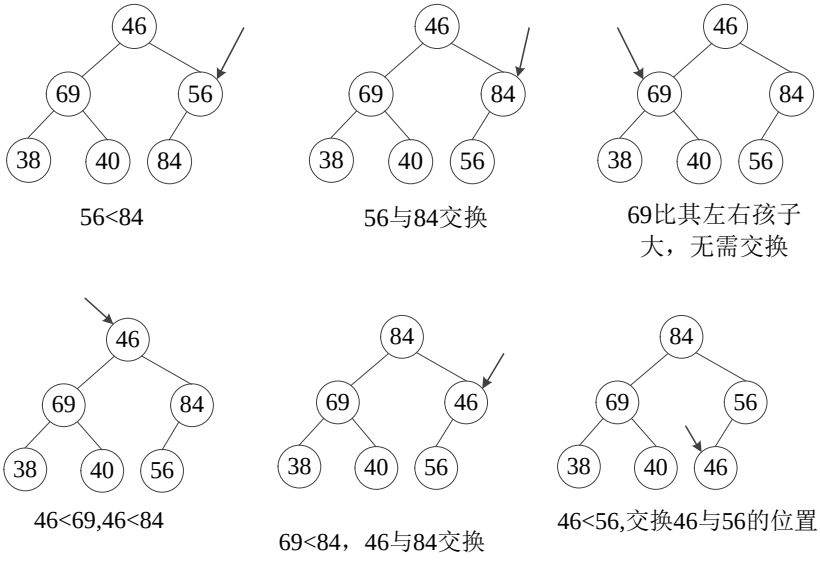


图 7.10

由图 7.10 可知，46 与 56 交换位置之后，完成了大顶堆的初始堆的构建。通过上述过程可以得到大顶堆的初始堆 84、69、56、38、40、46。故而，选择 B 答案。

关于堆排序的过程，我们在选择题部分只讲初始堆的构建，在综合题部分，我们将给出详细具体的堆排序过程。

【参考答案】 B

二．综合应用题部分

1. **(原书　第 1 题)**将下列关键字序列进行堆排序（大根堆），画出堆排序过程。

5　56　20　23　40　38　29　61　35　76　28　98

【解析】其基本思想为(大顶堆)：

(1). 将初始待排序关键字序列(R[1],R[2]....R[n])构建成大顶堆，此堆为初始的无序区；

(2). 将堆顶元素 R[1]与最后一个元素 R[n]交换，此时得到新的无序区(R[1],R[2],..... R[n-1])和新的有序区(R[n]),且满足 R[1,2...n-1]<=R[n];

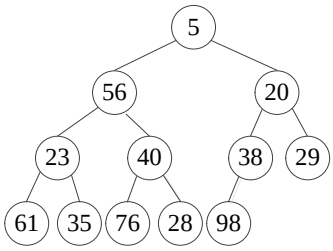
(3). 由于交换后新的堆顶 R[1]可能违反堆的性质，因此需要对当前无序区(R[1],R[2],..... R[n-1]) 调整为新堆，然后再次将 R[1]与无序区最后一个元素交换，得到新的无序区(R[1],R[2].... R[n-2])和新的有序区(R[n-1],R[n])。

(4). 不断重(2)和(3)，直到有序区的元素个数为 n-1，则整个排序过程完成。

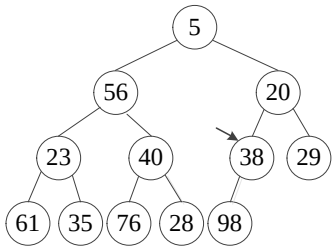
对于本题，我们对关键字序列

5　56　20　23　40　38　29　61　35　76　28　98

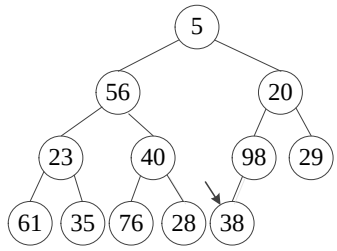
构建初始的大顶堆，如图 7.11 所示。



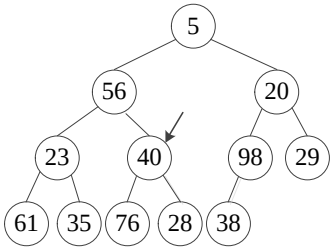
初始堆，待调整



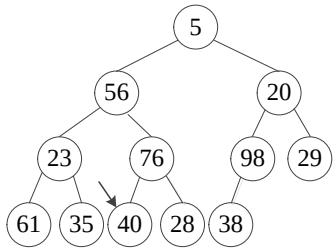
从第一个非叶子结点38开始调整



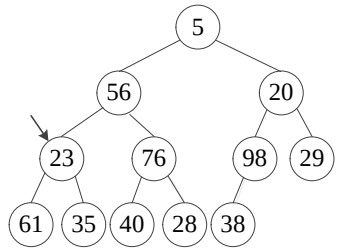
38<98，交换位置



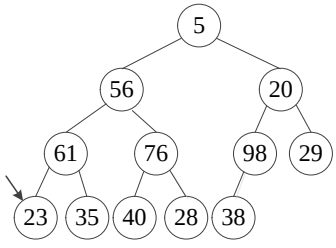
28<40<76



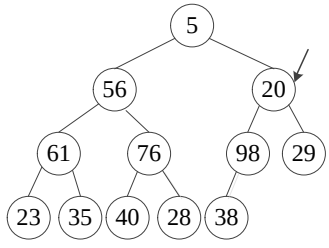
40与其左孩子76交换位置



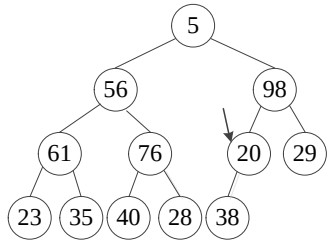
23<61且23<35



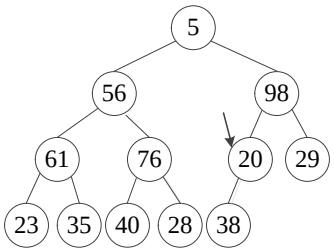
25与其左右孩子较大者61交换位置



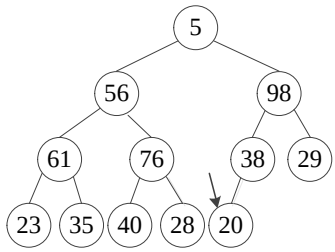
20小于其左右孩子98和29



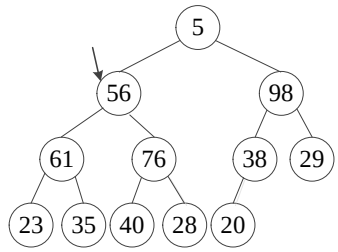
20与其左右孩子较大者98交换位置



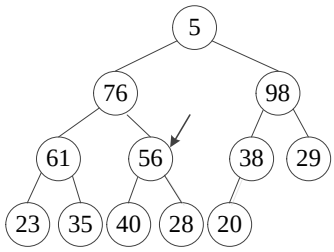
20仍小于其左孩子38



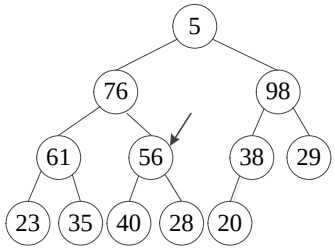
20与其左孩子38交换位置



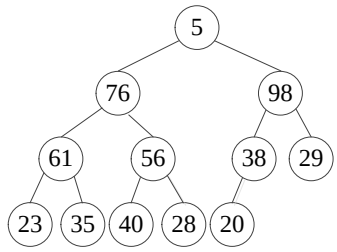
56小于其左右孩子61和76



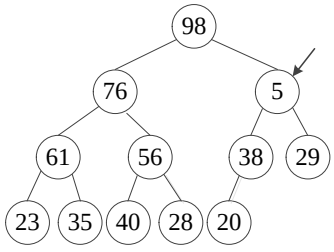
56与左右孩子较大者76交换位置



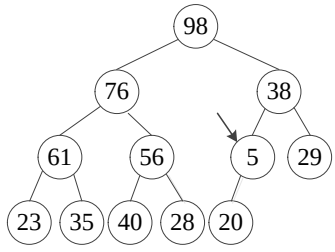
56大于其左右孩子，无需调整



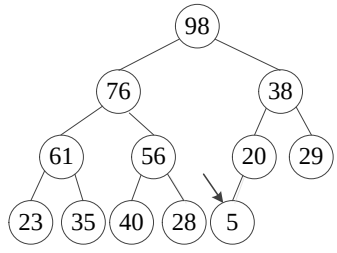
5小于其左右孩子76和98



5与其左右孩子较大者98交换位置



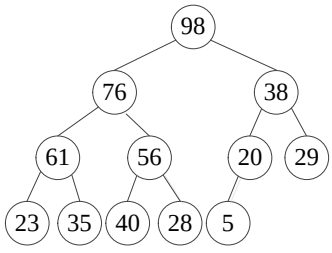
5<38且5<29,5与较大者38交换



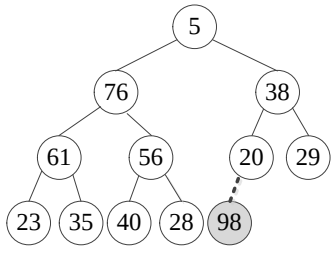
5小于20，与20交换形成大顶堆

图 7.11

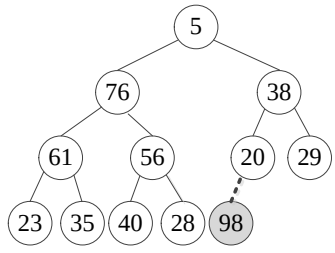
接下来，进行堆排序，从大到小输出关键字，如图 7.12 所示。



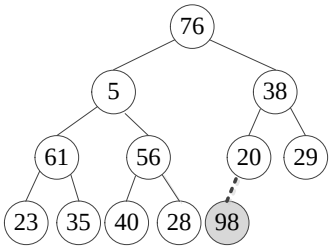
初始大顶堆



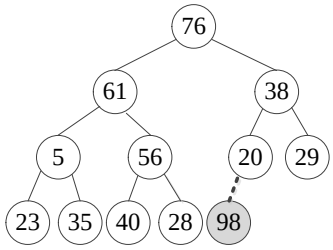
输出98, 98与5交换，堆待调整



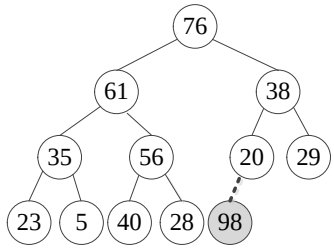
5<76且5<38



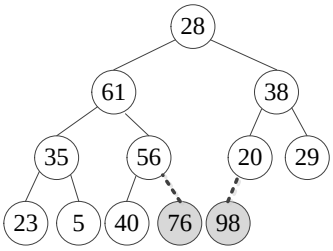
5与76交换，换后5<61且5<56



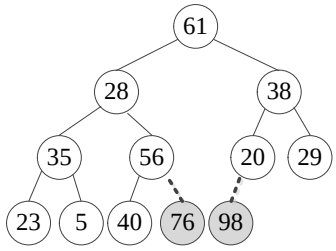
5与左右孩子较大者61交换



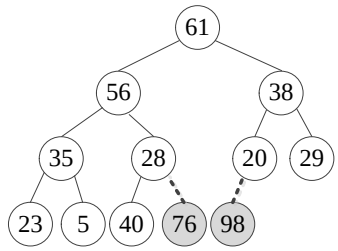
5继续与其左右孩子较大者35交换



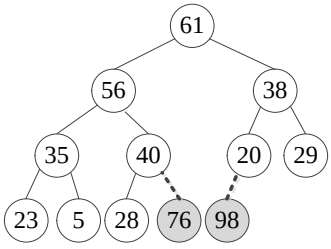
输出76,76与28交换，换后28小于左右孩子



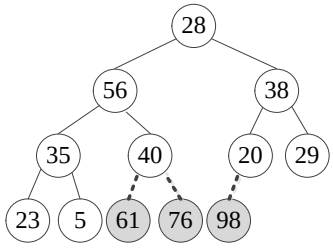
28与左右孩子较大者61交换



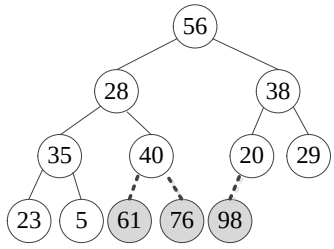
28>5且28<56，与56交换



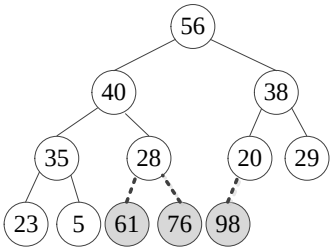
28小于其左孩子，继续与40交换



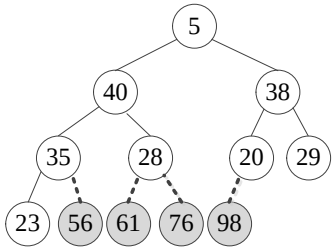
输出61,61与28交换，换后28小于左右孩子



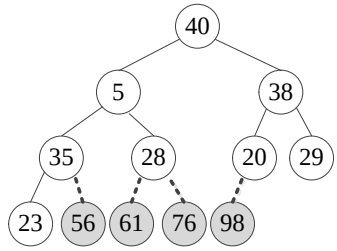
28与其左右孩子较大者56交换，换后小于左右孩子35和40



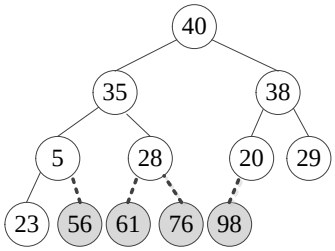
28继续与其左右孩子较大者40交换



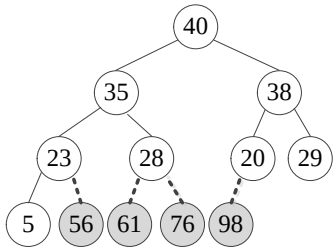
输出56,56与5交换，换后5小于其左后孩子40和38



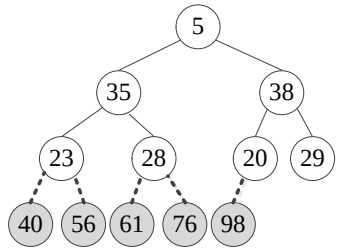
5与其左右孩子较大者40互换，换后5小于其左右孩子35和28



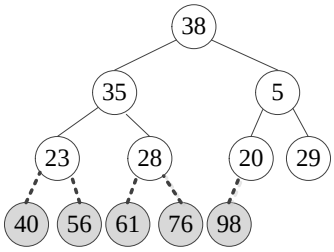
5与其左右孩子较大者35互换，换后5小于其左孩子23



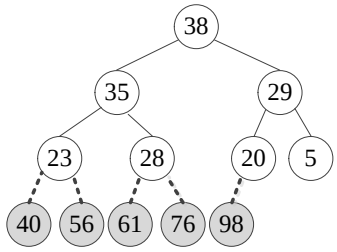
5与其左孩子23互换



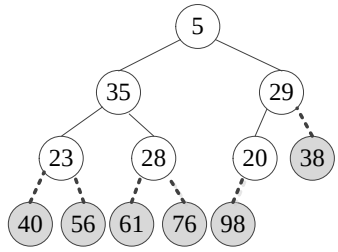
输出40,40与5互换，换后5小于其左右孩子35和38



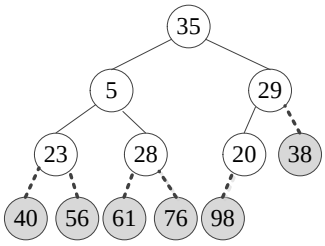
5与其左右孩子较大者38互换，换后5仍小于其新的左右孩子20和29



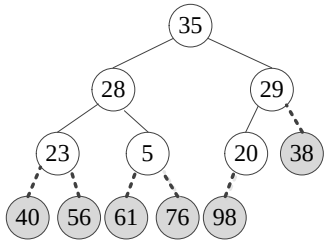
5与其左右孩子较大者29互换



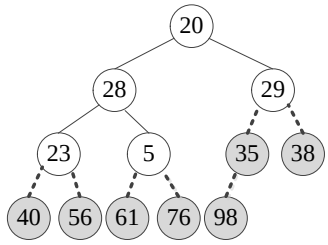
输出38,38与5互换，换后5小于其左右孩子35和29



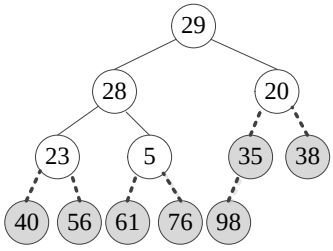
5与其左右孩子较大者35互换，换后5仍小于其左右孩子23和28



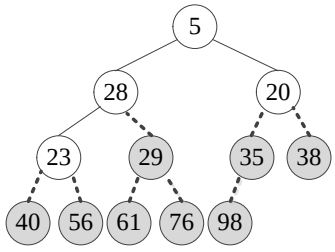
5继续与其左右孩子较大者28互换，形成新的大顶堆



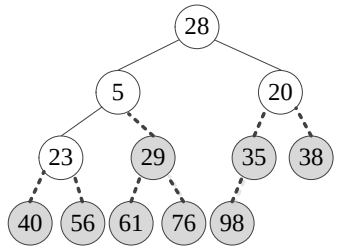
输出35,35与20互换，换后20小于其左右孩子28和29



20与其左右孩子较大者29互换，形成新的大顶堆



输出29,29与5互换，互换后，5小于其左右孩子



5与其左右孩子较大者28互换，换后5仍然小于新的左孩子23

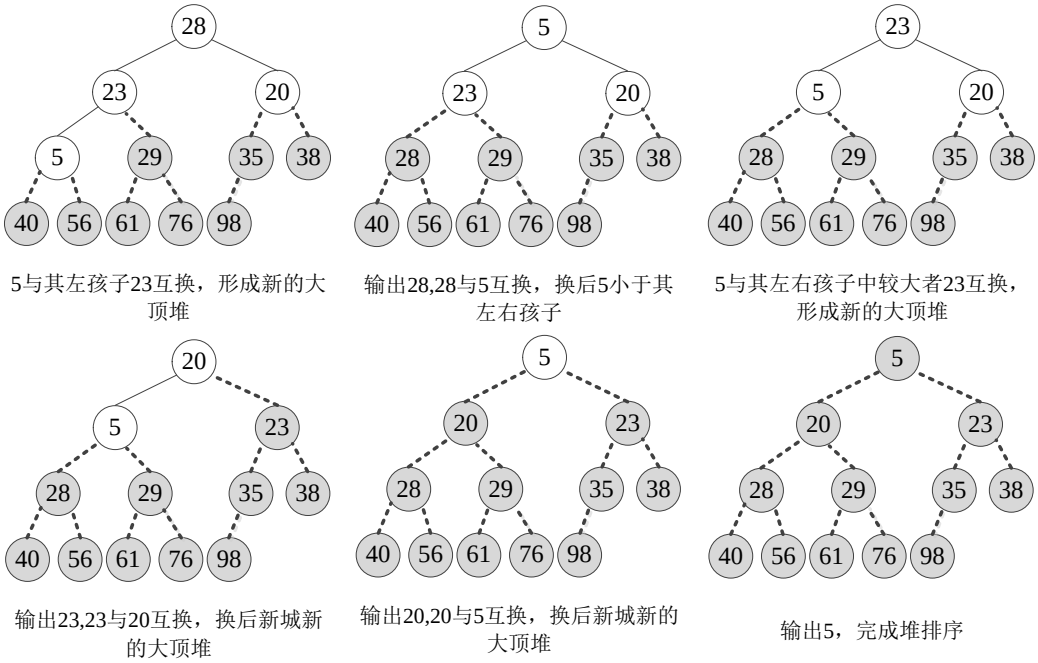


图 7.13

终于画完了，画完我就哭了，画了一天，画错了好几次，重新画的工作量，那叫一个大啊！不过，这个堆排序的图，的确十分详细，希望大家多看几遍，掌握其宗旨，不枉楠哥画图的一片苦心！

考点 8 二路归并排序

温馨提示：本考点主要考查归并排序的方法和某一趟归并排序的结果，希望同学们会手工对关键字进行归并排序。

一．选择题部分

1. **（原书　第 1 题）**设一组初始记录关键字序列为(25，50，15，35，80，85，20，40，36，70)，其中含有 5 个长度为 2 的有序子表，则用归并排序的方法对该记录关键字序列进行 2 趟归并后的结果为（　　）。

   A. 15，25，35，50，20，40，80，85，36，70

   B. 15，25，35，50，80，20，85，40，70，36

C.　15，25，35，50，80，85，20，36，40，70

D.　15，25，35，50，80，20，36，40，70，85

【解析】归并排序的基本思想为：将两个或两个以上的有序子序列“归并”为一个有序序列。 在内部排序中，通常采用的是 2-路归并排序。即：将两个位置相邻的有序子序列归并为一个有序序列。

归并排序的过程为：假设初始序列含有 n 个记录，则可看成 n 个有序的子序列，每个子序列长度为 1。然后两两归并，得到$\lceil n/2 \rceil$个长度为 2 或 1 的有序子序列；再两两归并，……如此重复，直至得到一个长度为 n 的有序序列为止。

对本题所给的序列进行二路归并排序，过程如图 7.14 所示。

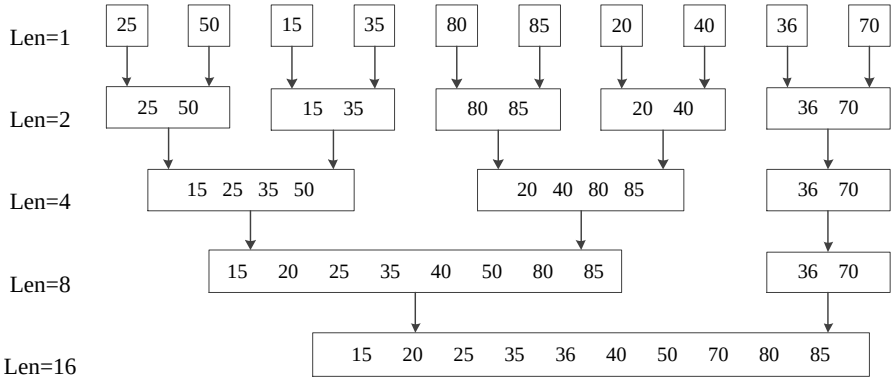


图 7.14

本题告知，初始记录关键字序列为(25，50，15，35，80，85，20，40，36，70)，其中含有 5 个长度为 2 的有序子表。故而，进行 2 趟归并的结果对应于 Len=4 所在的行的排序结果，即 15、25、35、50、20、40、80、85、36、70。

【参考答案】 A

2. **(原书　第 2 题)** 二路归并排序的时间复杂度为（　　）。

A.　O(n)　　B.　O(n^2)　　C.　O(n$\log_2$n)　　D.　O(1$\log_2$n)

【解析】一趟归并操作是将 r[1]~r[n]中相邻的长度为 h 的有序序列进行两两归并，这需要 $O(n)$时间。整个归并排序需要进行 $\log_2 n$ 趟，因此，总的时间代价是 $O(n\log_2 n)$。

【参考答案】 C

考点 9 基数排序

温馨提示：本考点主要考查桶排序的方法，可能会要求同学们手工操作，并给出某一趟分配和收集的结果。

一．选择题部分

1. **（原书　第 1 题）**设一组初始记录关键字序列为(345，253，674，924，627)，则用基数排序需要进行（　　）趟的分配和回收才能使得初始关键字序列变成有序序列。

A. 3　　B. 4　　C. 5　　D. 8

【解析】本题属于比较简单的基数排序常识。对于 3 位数，需要进行三次分配和收集才能使得初始关键字序列变成有序序列。读者可以参考本考点的综合应用题部分的习题。

【参考答案】 A

二．简答题部分

1. **（原书　第 1 题）**对 278、109、063、930、589、184、505、269、008、083 进行基数排序，并给出详细的排序过程。

【解析】基数排序通过“分配”和“收集”过程来实现排序。首先初始化 10 个“桶”，桶其实就是队列。按个位大小把待排序列依次放入到这十个桶中。第一趟的排序过程如图 7.15 所示。

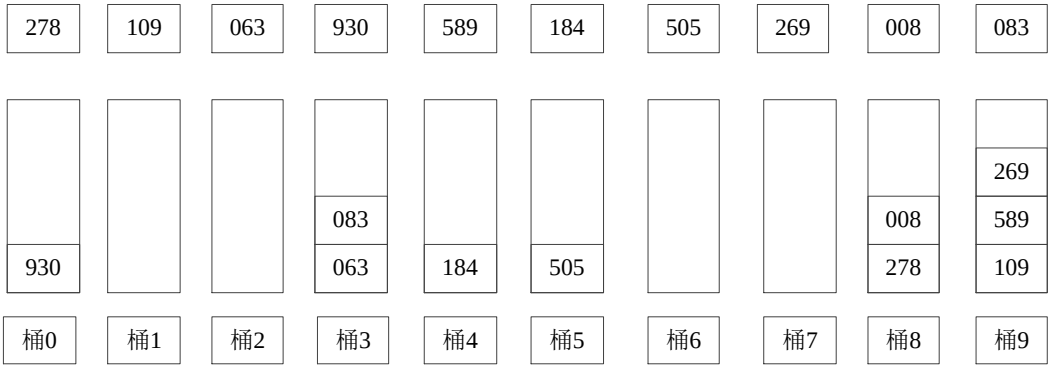


图 7.15

按照 0~9 的顺序对桶中的元素进行收集，每次收集都不要忘了，“桶”其实是队列，故而满足先进先出的规则。要是不按照先进先出，排序就不能顺利完成了。第一趟基数排序的收集结果如图 7.16 所示。

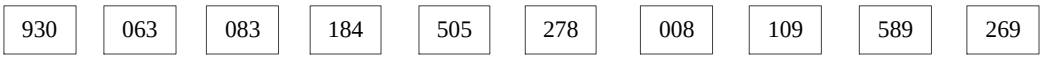


图 7.16

对第一趟的收集结果，我们发现个位已经有序了，但是十位和百位仍然是杂乱无章的。接下来，对十位进行排序。依次按照十位的大小**将第一趟收集的结果**放入下列的桶中，使得桶的编号和十位上的数相等。第二轮分配如图 7.17 所示。

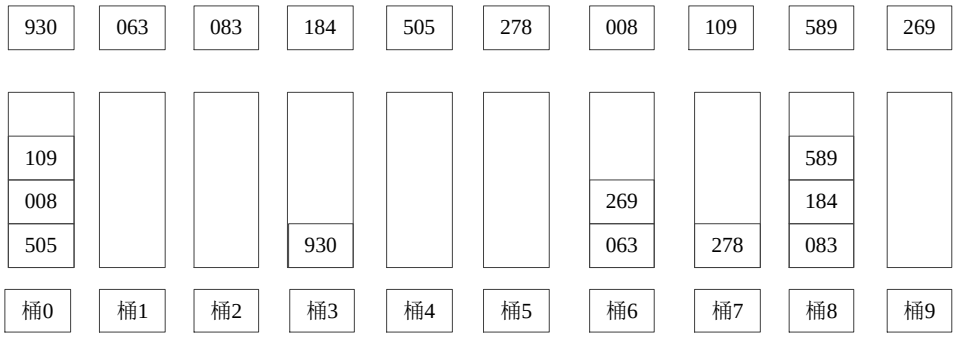


图 7.17

到此，第二轮分配已经完成，可以对各个桶中的元素进行收集了。仍然按照桶 0~桶 9 的顺序收集，而且满足桶中的元素先进先出，得到图 7.18 的收集结果。

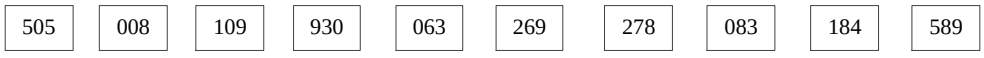


图 7.39

经过以上两次分配，个位和十位上的数字已经有序了，但是百位上的数还没有序，需要再进行一次分配和收集。下图 7.40 是对百位上的数的分配过程。

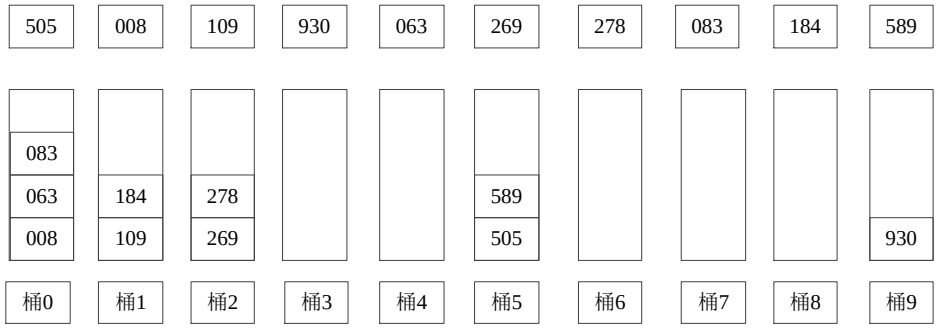


图 7.18

经过这三轮的分配，已经完成了对个位、十位和百位上的数字的排序，再进行一次收集，得到如图 7.19 所示的有序序列。

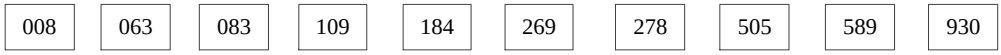


图 7.19

考点 10　外部排序算法及其应用

温馨提示：本考点主要考查外部排序算法。该部分知识点在历年考研中出现比较少，我们从《2016 年考研核心考点命题思路解密—数据结构》中抽取简单的例题，供同学们复习使用。

一．选择题部分

1. **(原书　第 1 题)**不定长文件是指（　　）。

A. 文件的长度不固定

B. 记录的长度不固定

C. 字段的长度不固定

D. 关键字项的长度不固定

【解析】若文件中记录含有的信息长度相同，则称这类记录为定长记录。由这种定长记录组成的文件称为定长文件。若文件中记录含有的信息长度不等，则称为不定长文件。

【参考答案】 B

2. **(原书　第 2 题)**对于大文件的排序要研究在外设上的排序技术，即（　　）。

A. 快速排序法　　B. 内排序法

C. 外排序法　　D. 交叉排序法

【解析】内排序是排序过程中参与排序的数据全部在内存中所做的排序，排序过程中无需进行内外存数据传送，决定排序方法时间性能的主要因素是数据排序码的比较次数和数据对象的移动次数。

外排序是在排序的过程中参与排序的数据太多，在内存中容纳不下，因此在排序过程中需要不断进行内外存的信息传送的排序。

【参考答案】 B

二．综合应用题部分

1. **(原书　第 1 题)**假设有 12 个初始归并段，其长度分别为 85，68，62，9，18，60，20，3，6，8，44，30；现要作 4 路外部归并排序，试画出表示归并过程的最佳归并树，并计算树的 wpl。

【解析】归并段的个数为 n=12，归并路数 k=4。(n-1)%(k-1)=11%3=2，不为 0。因此必须附加(k-1)-(n-1)%(k-1)=3-2=1 个长度为 0 的虚段，才能构成 4 路归并树。该 4 路归并树的构建过程如图 7.20~7.23 所示。

图 7.20

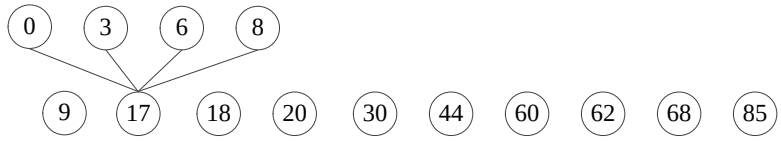


图 7.21

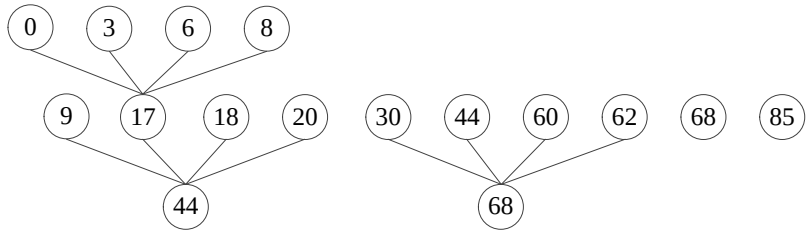


图 7.22

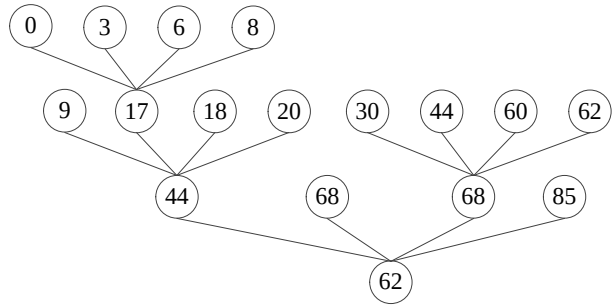


图 7.23

由图 7.23 可知，该最佳归并树的

WPL=(3+6+8)*3+(9+18+20+30+44+60+62)*2+(68+85)*1=690。

【查缺补漏】

如何判定虚设段的个数？

令 M 为初始归并段的个数，K 为归并的路数，若（M-1) MOD (K-1)= 0，则不加虚设段。否则，需加 P 个虚设段，其中 P = K-(M-1) MOD (K-1)－1.

2. **（原书　第 2 题）**设已有 12 个不等长的初始归并段，各归并段的长度（包含记录数）分别

是 RUN1 (25)，RUN2 (13)，RUN3 (04)，RUN4 (55)，RUN5 (30)，RUN6 (47)，RUN7 (19)，RUN8 (80)，RUN9 (76)，RUN10 (92)，RUN11 (55)，RUN12 (89) 若采用的是 4 路平衡归并排序，试画出其最佳归并树，并计算每趟归并时的读记录数。（括号内即为该归并段包含的记录数）

【解析】因为 (12 - 1) % (4 - 1) = 2，所以需要补充 4 - 2 - 1 = 1 个空归并段，命名为 RUN0 (0)。依次构造 4 叉哈夫曼树，如图 7.24 所示。

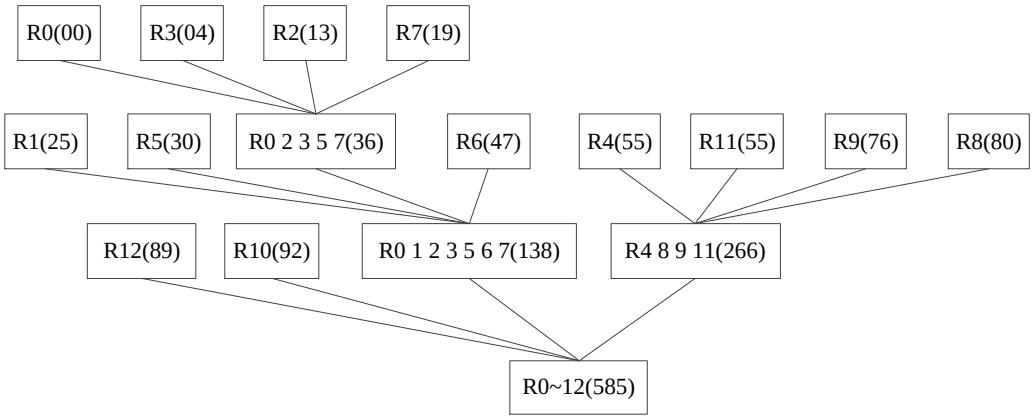


图 7.24

如图 7.24 所示的树即最佳归并树。第一趟读记录数为 36，第二趟读记录数为 138 + 266 = 404，第三趟读记录数为 89 + 92 +138 + 266 = 585。

“梦享考研系列”辅导书——答疑解惑

尊敬的读者朋友！

您好！首先，欢迎您使用我梦享团队编写的“梦享考研系列”辅导书，我们衷心地感谢您的支持！

让每一个考研人圆梦，是我们团队的梦想；默默陪伴着您走过每一个寻梦的日日夜夜，做您不离不弃的朋友，是我们的愿望。

宝剑锋从磨砺出，梅花香自苦寒来。每一个甜蜜的果实，都在挥洒汗水之后才能摘取到。对于每一个考生来说，一本绝佳的考研辅导书，就像一把利剑，拥有这把利剑，我们能够百战百胜，所向披靡。我们深知我们团队的能力有限，但我们一直希望，我们这套书，就是大家手中的那一把利剑！为此，我们团队特意向各位考生征求好的教材写法，好的题目解析方法等，希望把大家的精华汇聚到一本书里，为更多的考生提供更好的辅导书。

除了“梦享考研系列”辅导书之外，梦享团队致力于给大家更好的考研服务。为了给广大读者提供一个答疑和交流的平台，梦享团队开通了微信号、微信公众号和新浪微博账号等三种平台。欢迎大家来跟我们交流，一起成长。

shareOurDreams
梦享团队微信号

weCSdream
梦享团队官方微信公众号

梦享论坛团队
梦享团队新浪微博

因为有你，所以有梦享！

梦享团队祝愿每一个考研人梦想成真！

参考文献

[1] 汤子瀛. 计算机操作系统[M]. 西安：西安电子科技大学出版社，2001.

[2] Tanenbaum A.S. 现代操作系统[M]. 北京：机械工业出版社，2009.

[3] 翔高教育. 计算机学科专业基础综合习题精编[M]. 上海：复旦大学出版社，2010.

[4] 崔巍，等. 计算机学科专业基础综合辅导讲义[M]. 北京：原子能出版社，2011.

[5] 严蔚敏. 数据结构（C 语言版）[M]. 北京：清华大学出版社，2007.

[6] 严蔚敏，等. 数据结构题集（C 语言版）[M]. 北京：清华大学出版社，1999.

[7] 谢希仁. 数据结构（第 5 版）[M]. 北京：电子工业出版社，2008.

[8] 谢希仁. 数据结构释疑与习题解答[M]. 北京：电子工业出版社，2011.

[9] 唐朔飞. 计算机组成原理（第 2 版）[M]. 北京：高等教育出版社，2008.

[10] 唐朔飞.计算机组成原理学习指导与习题解答（第 2 版）[M]. 北京：高等教育出版社，2012.

