第 3 章 栈、队列和数组

3.1 本章导学

3.1.1 知识结构图

本章包括 3 部分，分别是栈、队列和数组，标☆的知识点为扩展与提高内容。本章学习要以栈和队列的操作特性为切入点，并注意将栈和队列、顺序栈和链栈、顺序队列和循环队列、循环队列与链队列进行对比。多维数组以矩阵的压缩存储方法为重点，辨析特殊矩阵和稀疏矩阵的压缩存储方法，找出矩阵的任意元素与其存储位置之间的关系。本章的知识结构如图 3-1 所示。

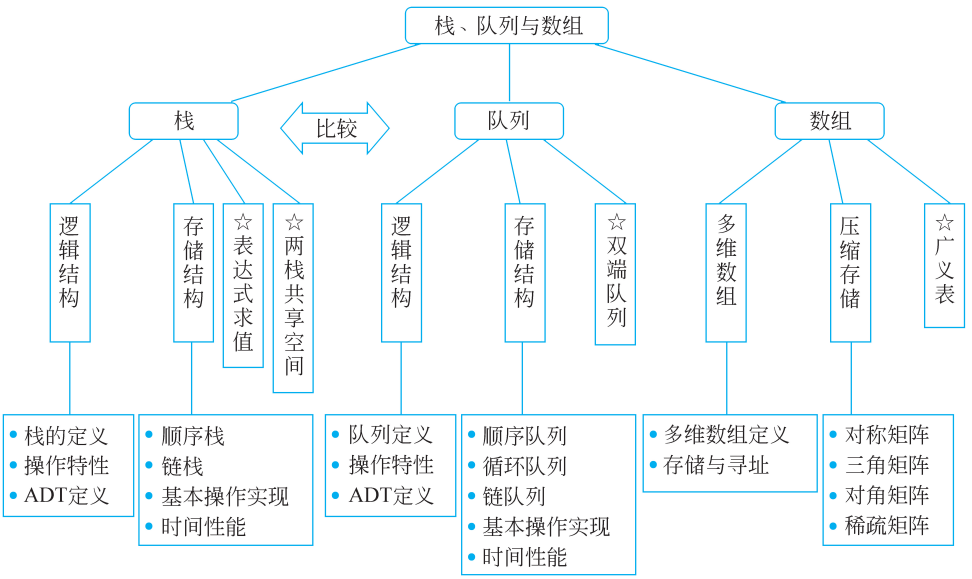


图 3-1 第 3 章的知识结构

3.1.2 重点整理

1. 栈是限定仅在表尾进行插入和删除操作的线性表。栈中元素除了具有线性关系外，还具有后进先出的操作特性。

2. 栈的顺序存储结构称为顺序栈，顺序栈本质上是顺序表的简化。通常把数组中下标为 0 的一端作为栈底，同时附设变量 top 指示栈顶元素在数组中的位置。顺序栈基本操作算法的时间复杂度均为 $O(1)$。

3. 栈的链接存储结构称为链栈，通常用单链表实现，并且用单链表的头部作为栈顶。链栈的插入和删除操作只需处理栈顶情况，时间复杂度均为 $O(1)$。

4. 队列是只允许在一端进行插入操作，在另一端进行删除操作的线性表。队列中的元素除了具有线性关系外，还具有先进先出的操作特性。

5. 顺序队列会出现假溢出问题，可以将顺序队列改造成首尾相接的循环队列。通常约定：变量 front 指向队头元素的前一个位置，变量 rear 指向队尾元素，凡是涉及队头或队尾指针的修改都需要将其与数组长度进行求模运算。

6. 在循环队列中，队空的判定条件是 front = rear；在浪费一个存储单元的情况下，队满的判定条件是(rear+1) %数组长度= front。

7. 队列的链接存储结构称为链队列，通常用单链表实现，并设置队头指针指向头结点，队尾指针指向终端结点。链队列的插入操作在队尾进行，删除操作在队头进行，时间复杂度均为 $O(1)$。

8. 在一个程序中，如果同时使用具有相同数据类型的两个顺序栈，可以使用一个数组存储两个栈，将一个栈的栈底位于该数组的始端，另一个栈的栈底位于该数组的末端，两个栈从各自的端点向中间延伸。

9. 如果允许在队列的两端进行插入和删除操作，则称为双端队列；如果允许在两端插入但只允许在一端删除，则称为二进一出队列；如果只允许在一端插入但允许在两端删除，则称为一进二出队列。

10. 中缀表达式的求值过程需要两个栈：栈 OPND 存放尚未参与计算的运算对象，栈 OPTR 存放优先级较低的运算符。

11. 数组是由类型相同的数据元素构成的有序集合，其特点是结构中的元素本身可以具有某种结构，但属于同一数据类型。数组是线性表的推广，例如，二维数组可以看作元素是线性表的线性表。

12. 在数组中通常只有两种操作：存取和修改，本质上只对应一种操作——寻址。由于数组一般不做插入和删除操作，因此，数组通常采用顺序存储结构，常用的映射方法有两种：按行优先和按列优先。

13. 矩阵压缩存储的基本思想是：为多个值相同的元素只分配一个存储空间；对零元素不分配存储空间。

14. 对称矩阵的压缩存储方法是将下三角中的元素按行优先存储到一维数组 SA 中，下三角中的元素 $a_{ij}(i \geqslant j)$在数组 SA 中的下标 k 与 i、j 的关系为 $k=i(i+1)/2+j$。三角矩阵的压缩存储方法与对称矩阵的压缩存储方法类似。

15. 对角矩阵的压缩存储方法是按行存储非零元素，按其压缩规律，找到相应的映像函数。例如，三对角矩阵压缩存储后的映像函数为 $k=2i+j$。

16. 稀疏矩阵的压缩存储需要将每个非零元素表示为三元组(行号，列号，非零元素)，将稀疏矩阵的非零元素对应的三元组所构成的集合按行优先的顺序排列成一个线性表，称为三元组表。三元组表有两种存储结构：顺序存储结构(称为三元组顺序表)和链接存储结构(称为十字链表)。

3.2　重点难点释疑

3.2.1　浅析栈的操作特性

栈是限定仅在表尾进行插入和删除操作的线性表，栈中元素除了具有线性关系外，还具有后进先出的操作特性。需要强调的是，栈只是对线性表的插入和删除操作的位置进行了限制，并没有限定插入和删除操作进行的时间，也就是说，出栈可随时进行，只要某个元素位于栈顶就可以出栈。例如，有 3 个元素 a、b、c 依次进栈，且每个元素只允许进一次栈，则可能的出栈序列有 5 种：abc、acb、bac、bca、cba，设 I 代表入栈，O 代表出栈，操作过程如图 3-2 所示。

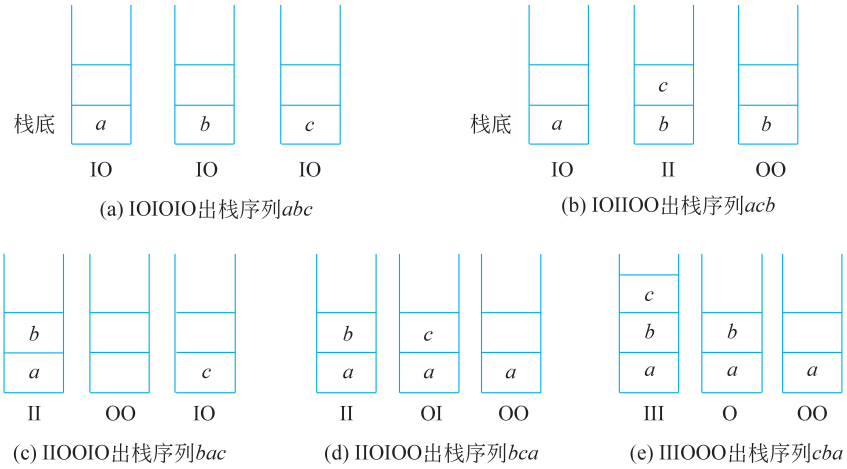


图 3-2　栈的操作过程

3.2.2　递归算法转换为非递归算法

将递归算法转换为非递归算法有两种方法：一种是直接转换法，不需要回溯，可以使用一些变量保存中间结果，将递归结构用循环结构替代；另一种是间接转换法，需要回溯，可以使用工作栈模拟系统栈保存中间结果。下面分别讨论这两种方法。

1. 直接转换法

直接转换法通常用来消除尾递归和单向递归。尾递归是指在递归算法中，递归调用语句只有一个，并且处在算法的最后。当递归调用返回时，将返回到上一层递归调用的下一条语句，而这个返回位置正好是算法的结束处，所以，可以使用一些变量保存中间结果，然后用循环结构替代递归结构。单向递归是指递归算法中虽然有多处递归调用语句，但各递归调用语句的参数之间没有关系，并且这些递归调用语句都处在递归算法的最后。显然，尾递归是单向递归的特例。

例 3-1　将求阶乘的递归算法转换为非递归算法。

解：设变量 s 保存递归的中间结果，递归与非递归算法如下。

```
long Fac1(int n)
{
  if (n == 1) return 1;
  else return n * Fac1(n-1);
}
```



```
long Fac2(int n)
{
  int s = 1, i;
  for (i = 2; i <= n; i++)
    s = s * i;
  return s;
}
```

例 3-2 将求斐波那契数列的递归算法转换为非递归算法。

解：设变量 s1 和 s2 分别保存 $f(n-1)$和 $f(n-2)$的值，递归与非递归算法如下。

```
int Fac3(int n)
{
  if (n == 1 || n == 2) return 1;
  else return Fac3(n-1) + Fac3(n-2);
}
```



```
int Fac4(int n)
{
  int i, s, s1 = 1, s2 = 1;
  for (i = 3; i <= n; i++)
  {
    s = s1 + s2;
    s2 = s1;        //保存 f(n-2)
    s1 = s;         //保存 f(n-1)
  }
  return s;
}
```

2. 间接转换法

间接转换法使用工作栈保存中间结果，通过模拟递归函数在执行过程中系统栈的变化得到非递归算法。间接转换法在数据结构中有较多实例，如二叉树遍历算法的非递归实现、图的深度优先遍历算法的非递归实现等。下面给出转换的算法框架：

```
将初始状态 s 进栈;
while (栈不为空)
{
  s = 栈顶元素;
  if (s 是要找的结果) 返回;
  else
  {
    t = s 的相关状态;
    if (t 存在) 将 t 进栈;
    else 退栈;
  }
}
```

3.2.3 循环队列中队空和队满的判定方法

对于循环队列有一个虽然小却十分重要的问题：如何确定队空和队满的判定条件。设存储循环队列的数组长度为 QueueSize，可以有以下 3 种判定方法：

方法一：浪费一个数组单元，则队满的判定条件是(rear＋1) mod QueueSize＝＝

front，从而保证了 front==rear 是队空的判定条件。

方法二：设置一个标志 flag，当 front==rear 且 flag==0 时为队空，当 front==rear 且 flag==1 时为队满。相应地要修改入队和出队算法，当有元素入队时，队列非空，将 flag 置 1；当有元素出队时，队列不满，将 flag 置 0。

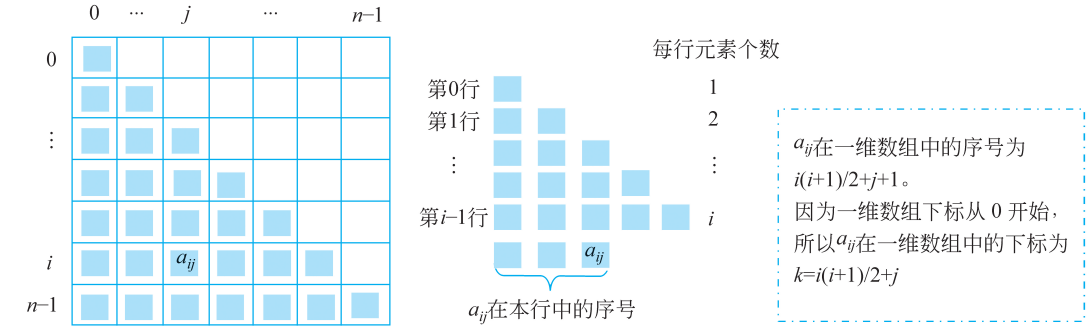
方法三：设置一个计数器 count 累计队列的长度，则当 count==0 时队列为空，当 count==QueueSize 时队列为满。相应地要修改入队和出队算法，入队时将 count 加 1，出队时将 count 减 1。

3.2.4　特殊矩阵压缩存储的寻址计算

在特殊矩阵的压缩存储中，注意矩阵中行下标和列下标的范围以及存储矩阵的一维数组的起始下标，不要死记公式，要真正理解压缩存储的方法，从中找出矩阵的任意元素与其存储位置之间的关系。

例 3-3　假设对称矩阵 $\boldsymbol{A}$ 的行下标和列下标的范围均为 0～$n-1$，将矩阵中的元素按行优先存储到数组 SA[$n(n+1)/2$]中，请给出寻址公式。

解：将对称矩阵 $\boldsymbol{A}$ 的下三角元素 a_{ij}（$n-1\geqslant i\geqslant j\geqslant 0$）存储到数组 SA[$k$]中，如图 3-3 所示，$k$ 与 i、j 之间的关系为 $k=i(i+1)/2+j$。



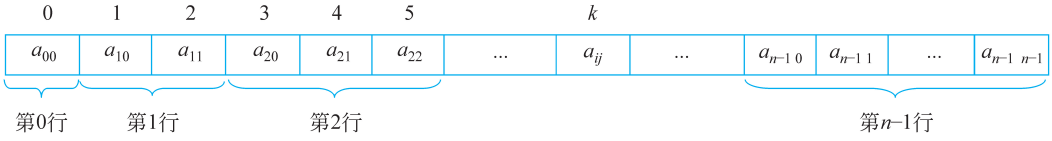
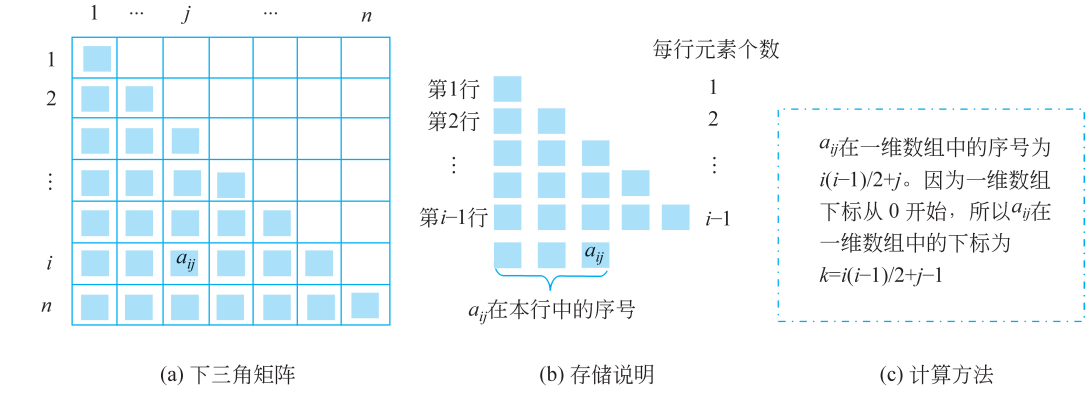


图 3-3　对称矩阵的压缩存储（矩阵行列下标均从 0 开始）

例 3-4　假设对称矩阵 $\boldsymbol{A}$ 的行下标和列下标的范围均为 1～n，将矩阵中的元素按行优先存储到数组 SA[$n(n+1)/2$]中，请给出寻址公式。

解：将对称矩阵 $\boldsymbol{A}$ 的下三角元素 a_{ij}（$n\geqslant i\geqslant j\geqslant 1$）存储到数组 SA[$k$]中，如图 3-4 所示，$k$ 与 i、j 之间的关系为 $k=i(i-1)/2+j-1$。



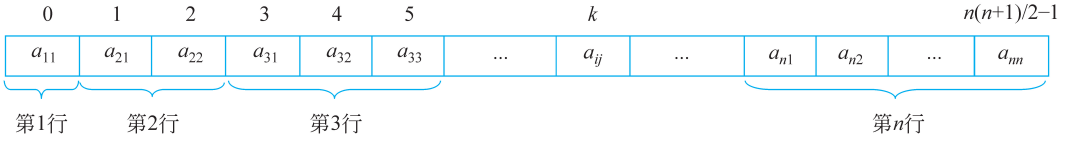


图 3-4 对称矩阵的压缩存储(矩阵行列下标均从 1 开始)

3.3 习 题 解 析

一、单项选择题

1. 一个栈的入栈序列是 1,2,3,4,5,则不可能的出栈序列是(　　)。

A. 54321　　B. 45321　　C. 43512　　D. 12345

【解答】 C

【分析】 此题有一个技巧：在出栈序列中任意元素后面不能出现比该元素小并且是升序(指的是元素的序号)的两个元素。

2. 若一个栈的入栈序列是 1,2,3,…,n,出栈序列的第一个元素是 n,则第 i 个出栈元素是(　　)。

A. 不确定　　B. $n-i$　　C. $n-i-1$　　D. $n-i+1$

【解答】 D

【分析】 出栈序列的第一个元素是 n,则出栈序列一定是入栈序列的逆序。

3. 若一个栈的入栈序列是 1,2,3,…,n,出栈序列是 $p_1,p_2,\cdots,p_n$,若 $p_1=3$,则 p_2 的值(　　)。

A. 一定是 2　　B. 一定是 1　　C. 不可能是 1　　D. 以上都不对

【解答】 C

【分析】 由于 $p_1=3$,说明 1,2,3 入栈后 3 出栈,此时可以将当前栈顶元素 2 出栈,也可以继续执行入栈操作,因此 p_2 的值可能是 2,但一定不能是 1,因为 1 不是栈顶元素。

4. 设计判断表达式中左右括号是否配对的算法,采用(　　)数据结构最佳。

A. 顺序表　　B. 栈　　C. 队列　　D. 链表

【解答】 B

【分析】　每个右括号与它前面的最后一个没有匹配的左括号配对，因此配对规则具有后进先出性。

5. 当字符序列 t3_依次通过栈，输出长度为 3 且可用作 C 语言标识符的序列个数是(　　)。

A. 4　　B. 5　　C. 3　　D. 6

【解答】　C

【分析】　输出长度为 3 说明将字符序列全部出栈，可以作为 C 语言标识符的序列只能以字母 t 或下画线开头，而栈的输出序列中以字母 t 或下画线开头的有 3 个，分别是 t3_、t_3 和 _3t。

6. 在栈顶指针为 top 的带头结点的链栈中插入指针 s 所指结点，执行的操作是(　　)。

A. top->next=s;

B. s->next=top;

C. s->next=top; top->next=s;

D. s->next=top->next; top->next=s;

【解答】　D

【分析】　链栈带头结点，将结点 s 插在头结点的后面。

7. 在栈顶指针为 top 的不带头结点的链栈中删除栈顶结点，用 x 保存被删除结点的值，执行的操作是(　　)。

A. x=top; top=top->next;

B. x=top->data;

C. top=top->next; x=top->data;

D. x=top->data; top=top->next;

【解答】　D

【分析】　链栈不带头结点，栈顶结点即指针 top 指向的结点，先暂存 top->data，再将栈顶结点摘链。

8. 在解决计算机主机与打印机之间速度不匹配问题时通常设置一个打印缓冲区，该缓冲区应该是一个(　　)结构。

A. 栈　　B. 队列　　C. 数组　　D. 线性表

【解答】　B

【分析】　先进入打印缓冲区的文件先被打印，因此缓冲区具有先进先出特性。

9. 一个队列的入队顺序是 1,2,3,4，则队列的输出顺序是(　　)。

A. 4321　　B. 1234　　C. 1432　　D. 3241

【解答】　B

【分析】　队列的入队顺序和出队顺序总是一致的。

10. 设数组 $S[n]$作为两个栈 S1 和 S2 的存储空间，对任何一个栈只有当 $S[n]$全满时不能进行进栈操作。为这两个栈分配空间的最佳方案是(　　)。

A. S1 的栈底位置为 0，S2 的栈底位置为 $n-1$

B. S1 的栈底位置为 0，S2 的栈底位置为 $n/2$

C. S1 的栈底位置为 0，S2 的栈底位置为 n

D. S1 的栈底位置为 0，S2 的栈底位置为 1

【解答】 A

【分析】 两栈共享空间要保证两个栈是相向增长，因此两个栈的栈底应该分别位于数组的两端。

11. 假设用数组 $A[21]$存储循环队列，front 指向队头元素的前一个位置，rear 指向队尾元素，假设当前 front 和 rear 的值分别为 8 和 3，则该队列的长度为(　　)。

A. 5　　B. 11　　C. 16　　D. 24

【解答】 C

【分析】 队列长度是(rear－front＋21) mod 21 ＝ (3－8＋21) mod 21 ＝ 16。

12. 假设循环队列存储在数组 $A[0]$～$A[m]$中，则入队操作为(　　)。

A. rear＝rear＋1　　B. rear＝(rear＋1) mod (m－1)

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

【解答】 D

【分析】 入队操作将 rear 加 1 后与数组长度取模，注意数组长度是 m＋1。

13. 在链队列中，设指针 f 和 r 分别指向队首结点和队尾结点，则插入 s 所指结点的操作是(　　)。

A. f－＞next＝s; f＝s;　　B. r－＞next＝s; r＝s;

C. s－＞next＝r; r＝s;　　D. s－＞next＝f; f＝s;

【解答】 B

【分析】 结点 s 插在队尾的后面，无须修改队头指针。

14. 设栈 S 和队列 Q 的初始状态为空，元素 e1、e2、e3、e4、e5、e6 依次通过栈 S，一个元素出栈后即进入队列 Q。若 6 个元素出队的顺序是 e2、e4、e3、e6、e5、e1，则栈 S 的容量至少应该是(　　)。

A. 6　　B. 4　　C. 3　　D. 2

【解答】 C

【分析】 由于队列具有先进先出性，所以，此题中队列形同虚设，即出栈顺序也是 e2、e4、e3、e6、e5、e1，操作序列是 IIOIIOOIIOOO，栈中元素的最大个数是 3。

15. 在表达式 3＊2^(4＋2＊2－6＊3)－5 求值过程中，当扫描到 6 时，对象栈和算符栈的元素分别为(　　)，其中^表示乘幂。

A. 3,2,4,4; ＃ ＊ ^(＋－　　B. 3,2,8; ＃ ＊ ^－

C. 3,2,4,2,2; ＃ ＊ ^(－　　D. 3,2,8; ＃ ＊ ^(－

【解答】 D

【分析】 当扫描到 6 时，已经计算了 2＊2＝4 和 4＋4＝8，因此，对象栈为 3，2，8；算符栈中还有未计算的运算符 ＃ ＊ ^(－。

16. 二维数组 A 行下标的范围是 0～8，列下标的范围是 0～9。若 A 按行优先方式存储，元素 $A[8][5]$的起始地址与当 A 按列优先方式存储(　　)元素的起始地址一致。

A. $A[8][5]$　　B. $A[3][10]$　　C. $A[5][8]$　　D. $A[4][9]$

【解答】 D

【分析】 二维数组 A 有 9 行 10 列，设数组 $A[9][10]$的起始地址是 d，则元素 $A[8][5]$按行优先存储的起始地址为 $d+8\times10+5=d+85$。设元素 $A[i][j]$按列优先存储的起始地址与之相同，则 $d+9\times j+i=d+85$，解得 $i=4$，$j=9$。

17. 如果一维数组 $a[50]$和二维数组 $b[10][5]$具有相同的基类型和首地址，假设二维数组 b 以列优先方式存储，则 $a[18]$的地址和(　　)的地址相同。

A. $b[1][7]$　　B. $b[1][8]$　　C. $b[8][1]$　　D. $b[7][1]$

【解答】 C

【分析】 $a[18]$是数组 a 的第 19 个元素，设 $b[i][j]$是数组 b 以列优先方式存储的第 19 个元素，则 $10\times j+(i+1)=19$，解得 $i=8$，$j=1$。

18. 将 100×100 的三对角矩阵 $\boldsymbol{A}$ 按行优先压缩存储在数组 $B[298]$中，则元素 $A[66][65]$在数组 B 中的下标为(　　)。

A. 198　　B. 195　　C. 197　　D. 196

【解答】 C

【分析】 矩阵 $\boldsymbol{A}$ 的第 0 行有 2 个非零元素，第 1～65 行每行有 3 个非零元素，元素 $A[66][65]$是第66 行的第一个非零元素，则 $A[66][65]$在数组 B 中的下标为 $2+65\times3=197$。

19. 若将 n 阶上三角矩阵 $\boldsymbol{A}$ 按列优先方式压缩存储在数组 $B[n(n+1)/2]$中，则存放到 $B[k]$中的非零元素 a_{ij}($1\leqslant i$，$j\leqslant n$)的下标 i、j 与 k 的对应关系是(　　)。

A. $\dfrac{i(i+1)}{2}+j$　　B. $\dfrac{i(i-1)}{2}+j-1$

C. $\dfrac{j(j+1)}{2}+i$　　D. $\dfrac{j(j-1)}{2}+i-1$

【解答】 D

【分析】 按列优先方式存储，元素 a_{ij} 的左面有 $j-1$ 列，共计 $1+2+\cdots+(j-1)=j(j-1)/2$ 个元素，元素 a_{ij} 在第 j 列是第 i 个元素，注意到数组 B 的下标从 0 开始，则 $k=\dfrac{j(j-1)}{2}+i-1$。

20. 假设 100×90 的稀疏矩阵 $\boldsymbol{A}$ 的元素类型为整型，其中非零元素个数为 10，设整型数据占 2 字节，则采用三元组顺序表存储时，需要的字节数是(　　)。

A. 60　　B. 66　　C. 18000　　D. 33

【解答】 B

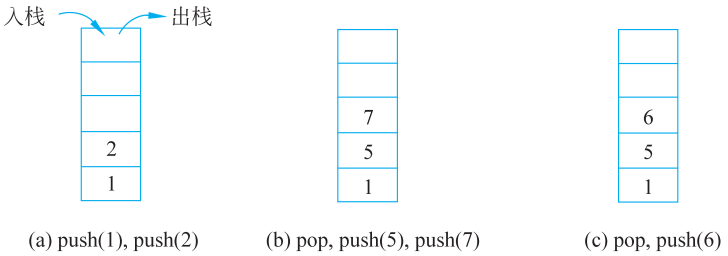
【分析】 稀疏矩阵用三元组(行号，列号，元素值)表示非零元素，每个非零元素需要 $3\times2=6$ 字节。三元组顺序表还要存储稀疏矩阵的行数、列数和非零元素个数，因此，需要的字节数是 $10\times(3\times2)+2+2+2=66$。

二、解答下列问题

1. 在操作序列 push(1)、push(2)、pop、push(5)、push(7)、pop、push(6)之后，栈顶元素和栈底元素分别是什么？画出操作序列的执行过程。(push(k)表示整数 k 入栈，pop

表示栈顶元素出栈。)

【解答】 栈顶元素为 6,栈底元素为 1,执行过程如图 3-5 所示。

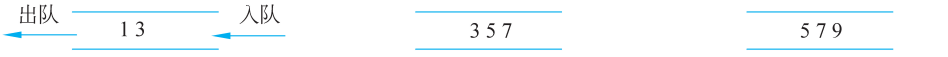


(a) push(1), push(2) (b) pop, push(5), push(7) (c) pop, push(6)

图 3-5 栈的执行过程

2. 在操作序列 EnQueue(1)、EnQueue(3)、DeQueue、EnQueue(5)、EnQueue(7)、DeQueue、EnQueue(9)之后,队头元素和队尾元素分别是什么? 画出操作序列的执行过程。(EnQueue(k)表示整数 k 入队,DeQueue 表示队头元素出队。)

【解答】 队头元素为 5,队尾元素为 9,执行过程如图 3-6 所示。



(a) EnQueue(1), EnQueue(3) (b) DeQueue, EnQueue(5), EnQueue(7) (c) DeQueue, EnQueue(9)

图 3-6 队列的执行过程

3. 假设 I 和 O 分别表示入栈和出栈操作。若入栈序列为 1、2、3、4,能否得到如下出栈序列? 若能,请给出相应的 I 和 O 操作串;若不能,请说明原因。

(1) 3、1、4、2。

(2) 1、3、2、4。

【解答】 序列(1)不能,3 是第一个出栈元素,则栈中元素自栈底依次是 1、2,此时要么 2 出栈,要么 4 进栈后再出栈。由于 1 不是栈顶元素,因此第 2 个出栈元素不可能是 1。序列(2)可以,操作串是 IOIIOOIO。

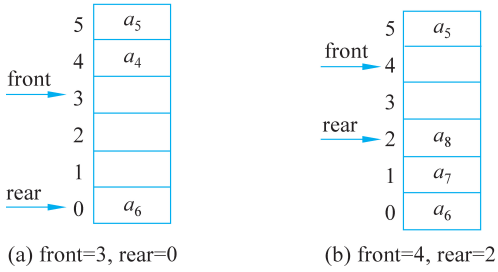
4. 假设 I 和 O 分别表示入栈和出栈操作。栈的初态和终态均为空,入栈和出栈的操作序列可表示为仅由 I 和 O 组成的序列,称可以操作的序列为合法序列,否则称为非法序列。如何判定所给的操作序列是否合法呢? 请结合具体实例给出判定规则。

【解答】 在入栈和出栈操作序列的任一位置,入栈次数(即 I 的个数)都必须大于或等于出栈次数(即 O 的个数),否则在形式上视作非法序列。由于栈的初态和终态都为空,因此整个序列的入栈次数必须等于出栈次数,否则在形式上视为非法序列。例如,IOIIOIOO 和 IIIOOIOO 是合法序列,IIOIOIIO 和 IIIOIOIO 是非法序列。

5. 若用一个长度为 6 的数组实现循环队列,且当前 rear 和 front 的值分别为 0 和 3,从队列中删除一个元素,再增加两个元素后,rear 和 front 的值分别是多少? 请画出操作示意图。

【解答】 当前 rear 和 front 的值分别为 0 和 3,则队列中有 3 个元素,如图 3-7(a)所示。删除一个元素后 front 的值为 4,增加两个元素后 rear 的值为 2,如图 3-7(b)所示。

6. 对于循环队列,可以用队列中的元素个数代替队尾位置,请定义循环队列的这种



(a) front=3, rear=0　　(b) front=4, rear=2

图 3-7　循环队列的操作示意图

存储结构。

【解答】　设变量 front 存储队头元素的前一个位置，变量 count 存储队列的元素个数，可以通过队头位置和队列长度计算出队尾元素的位置，存储结构定义如下：

```
#define QueueSize 100                    //定义数组的最大长度
typedef int DataType;                    //定义队列元素的数据类型,假设为 int 型
typedef struct
{
  DataType data[QueueSize];              //存放队列元素的数组
  int front,count;                       //队头位置和队列长度
} CirQueue;
```

7. 利用两个栈 S1 和 S2 模拟一个队列，利用栈的运算实现队列的插入和删除操作，请简述算法思想。

【解答】　利用两个栈 S1 和 S2 模拟一个队列。在队列中插入元素时，用栈 S1 存放已经入队的元素，即通过向栈 S1 执行入栈操作实现。在队列中删除元素时，将栈 S1 的元素全部送入栈 S2 中，再从栈 S2 中删除栈顶元素，最后将栈 S2 的元素全部送入栈 S1 中。判断队空的条件是栈 S1 和 S2 同时为空。

8. 给出表达式 A－B＊C/D 的求值过程，说明在求值过程中栈的作用。

【解答】　设栈 OPND 存储运算对象，栈 OPTR 存储优先级较低的运算符，在计算表达式的过程中，栈 OPND 和 OPTR 的变化过程如表 3-1 所示。算术表达式中的下画线标示当前扫描的字符。

表 3-1　栈 OPND 和 OPTR 的变化过程

步骤	栈 OPND	栈 OPTR	算术表达式
初始		#	A－B＊C/D #
1	A	#	A－B＊C/D #
2	A	#－	－B＊C/D #
3	AB	#－	B＊C/D #
4	AB	#－＊	＊C/D #
5	ABC	#－＊	C/D #

续表

步骤	栈 OPND	栈 OPTR	算术表达式
6	$AT_1(T_1=B*C)$	#－/	/D #
7	AT_1D	#－/	D #
8	$AT_2(T_2=T_1/D)$	#－	#
9	$T_3(T_3=A-T_2)$	#	#

9. 如果一维数组的元素个数非常多，但存在大量重复数据，并且所有值相同的元素位于连续的位置，请设计压缩存储方法。

【解答】 设二元组(data, count)存储元素值及该值的元素个数。例如，数组 $r[]=\{1,1,1,5,5,3,3,3,3,3,3,3,3,9\}$，对应的压缩存储为 $B[]=\{\{1,3\},\{5,2\},\{3,8\},\{9,1\}\}$。显然，值相同的元素越多，采用这种压缩存储方法越节省存储空间。

10. 对于二维数组 $a[n][2n-1]$，将 3 个顶点分别为 $a[0][n-1]$、$a[n-1][0]$和 $a[n-1][2n-2]$的三角形内的所有元素按行序存放在一维数组 $B[n\times n]$中，且元素 $a[0][n-1]$存放在 $B[0]$中。例如当 $n=3$ 时，数组 $a[3][5]$中的三角形如图 3-8 所示，存储结果如图 3-9 所示。如果将三角形中的元素 a_{ij} 存放在 $B[k]$中，请给出下标 i、j 与 k 的对应关系。

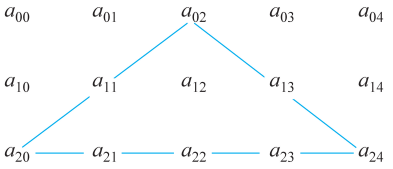


图 3-8 二维数组中的三角形

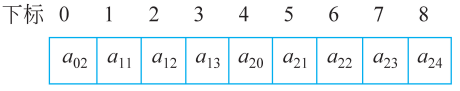


图 3-9 三角形的存储示意图

【解答】 在二维数组的三角形中，元素 a_{ij} 的上面有第 0～$i-1$ 行，共 i 行，第 0 行的元素个数为 1，第 2 行的元素个数为 3……第 $i-1$ 行的元素个数为 $2i-1$，共计 $1+3+\cdots+(2i-1)=i^2$ 个元素。

设三角形第 i 行第一个元素的列号为 m，当 $i=0$ 时 $m=n-1$，当 $i=1$ 时 $m=n-2$……则第 i 行第 1 个元素的列号 $m=n-i-1$，元素 a_{ij} 是第 i 行的第 $j-m+1=j-n+i+2$ 个元素。因为数组 B 的下标从 0 开始，所以有 $k=i^2+i+j-n+1$。

11. 设有五对角矩阵 $\boldsymbol{B}=(b_{ij})_{20\times 20}$，按特殊矩阵压缩存储的方式将 5 条对角线上的元素存于数组 $A[m]$中，计算元素 $b_{15,16}$在数组 A 中的存储位置。

【解答】 假设矩阵 $\boldsymbol{B}$ 的行列下标均从 1 开始，第 1 行有 3 个非零元素，第 2 行有 4 个非零元素，第 3～14 行有 5 个非零元素，$b_{15,16}$是第 15 行的第 4 个非零元素，则 $k=3+4+12\times 5+4=71$，即元素 $b_{15,16}$是数组 A 的第 71 个元素。因为数组 A 的下标从 0 开始，所以元素 $b_{15,16}$在数组 A 中的存储位置(即下标)是 70。

$$\begin{pmatrix} 0 & 0 & 2 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

图 3-10 稀疏矩阵

12. 对于如图 3-10 所示的稀疏矩阵，请画出该矩阵的三元组顺

序表和十字链表存储示意图。

【解答】　三元组顺序表如图 3-11 所示，十字链表如图 3-12 所示。

下标	行号	列号	非零元素
0	1	3	2
1	2	1	3
2	3	3	1
3	3	4	5
4（行数）			
5（列数）			
4（非零元个数）			

图 3-11　稀疏矩阵的三元组顺序表

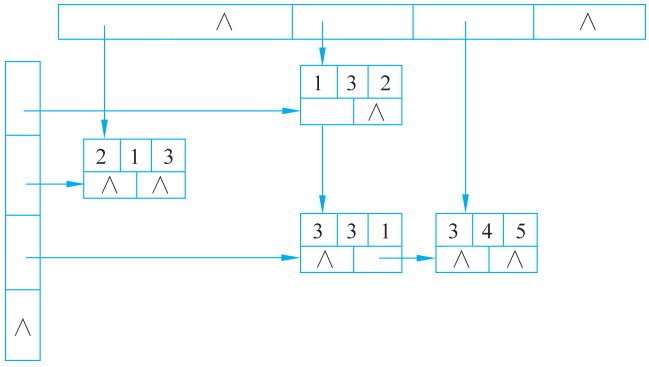


图 3-12　稀疏矩阵的十字链表

三、算法设计题

1. 设顺序栈有 $2n$ 个元素，从栈顶到栈底的元素依次为 $a_{2n}, a_{2n-1}, \cdots, a_1$，要求通过一个循环队列重新排列栈中元素，使得从栈顶到栈底的元素依次为 $a_{2n}, a_{2n-2}, \cdots, a_2, a_{2n-1}, a_{2n-3}, \cdots, a_1$，请给出算法的操作步骤，要求空间复杂度和时间复杂度均为 $O(n)$。

【解答】　根据栈和队列的操作特性，算法的操作步骤如下：

（1）将所有元素依次出栈并入队。

（2）将队列元素依次出队，将偶数号元素入队，将奇数号元素入栈。

（3）将奇数号元素依次出栈并入队。

（4）将偶数号元素依次出队并入栈。

（5）将偶数号元素依次出栈并入队。

（6）将所有元素依次出队并入栈。

2. 假设以不带头结点的循环单链表存储队列，并且只设一个指针指向队尾结点，请设计相应的入队和出队算法。

【解答】　入队操作在循环单链表的尾部进行，在终端结点之后插入一个新结点。出队操作在循环单链表的头部进行，删除开始结点。由于循环单链表不带头结点，需要处理空表的特殊情况。单链表的结点结构定义参见主教材。算法如下：

```
Node * Enqueue(Node * rear, int x)
{
  Node * s = NULL;
  s = (Node * )malloc(sizeof(Node));
  s->data = x;
  if (rear == NULL)                          //处理空表的特殊情况
  {
    rear = s; rear->next = s;
  }
  else                                       //处理除空表以外的一般情况
```

```
  {
    s->next = rear->next;
    rear->next = s; rear = s;
  }
  return rear;
}
```

```
int Dequeue(Node * rear)
{
  Node * s = NULL;
  if (rear == NULL) {printf("underflow"); return 0;}    //判断表空
  else
  {
    s = rear->next;
    if (s == rear) rear = NULL;                         //链表中只有一个结点
    else rear->next = s->next;
    free(s);
  }
}
```

3. 将十进制整数转换为二进制至九进制中的任一进制整数。

【解答】 算法基于的原理：$N=(N \text{ div } d)\times d + N \bmod d$（div 为整除运算，mod 为求余运算），先得到的余数为低位，后得到的余数为高位，因此，将每一步求得的余数放入栈中，再将栈元素依次输出，即可得到转换结果。假设采用顺序栈存储转换后的结果，算法如下：

```
void Decimaltor(int num, int r)
{
  int S[100], top = -1, k;                              //假设采用顺序栈
  while (num != 0)
  {
    k = num % r;                                        //得到余数
    S[++top] = k;
    num = num / r;                                      //得到商
  }
  while (top != -1)
    printf("%d", S[top--]);
}
```

4. 假设算术表达式包含 3 种括号：圆括号“(”和“)”、方括号“[”和“]”以及花括号“{”和“}”，并且这 3 种括号可以按任意次序嵌套使用。判断给定表达式所含括号是否配对出现。

【解答】 设字符数组 A 存储算术表达式。扫描表达式。对于左括号，执行入栈操作。对于右括号，如果栈顶元素是匹配的左括号，则执行出栈操作；否则匹配失败，结束扫描。算法如下：

```
int Prool(char A[ ])
{
  int S[100], top = -1, i;
  for (i = 0;A[i] != '\0'; i++)
  {
    if(A[i] == '(' || A[i] == '[' || A[i] == '{' )
      S[++top] = A[i];
    else
    {
      switch A[i]
      {
        case ')': if (top == -1 || S[top--] != '(') return 0;
        case ']': if (top == -1 || S[top--] != '[') return 0;
        case '}': if (top == -1 || S[top--] != {') return 0;
      }
    }
  }
  if (top == -1) return 1;
  else return 0;
}
```

5. 在循环队列中设置标志 flag，当 front = rear 且 flag = 0 时为队空，当 front = rear 且 flag = 1 时为队满。请设计相应的入队和出队算法。

【解答】 当有元素入队时，队列非空，将 flag 置 1；当有元素出队时，队列不满，将 flag 置 0。循环队列的存储结构定义请参见主教材。算法如下：

```
int Push(CirQueue * Q, int x)
{
  if (Q->front == Q->rear && flag == 1) {printf("overflow"); return 0; }
  Q->rear = (Q->rear+1) % QueueSize;
  Q->data[Q->rear] = x;
  flag = 1;
  return 1;
}
```

```
int Pop(CirQueue Q, int &x)
{
  if (Q->front == Q->rear && flag == 0) {printf("underflow"); return 0; }
  Q->front = (Q->front+1) % QueueSize;
  x = Q->data[front];
  flag = 0;
  return 1;
}
```

6. 给定具有 n 个字符的序列，依次通过一个栈可以产生多种出栈序列，请判断一个序列是否是可能的出栈序列。

【解答】 设数组 in[n]存储输入的字符序列，数组 out[n]表示要判断的出栈序列，设

变量 i 和 j 分别指示正在处理的入栈和出栈字符，扫描数组 in[n]模拟栈的操作：将 in[i]入栈；当栈顶元素等于 out[j]时，执行出栈。如果 out[n]是可能的出栈序列，则数组 in[n]和out[n]恰好匹配，并且处理完成后栈为空。设数组 S[n]表示顺序栈，算法如下：

```
int IsSerial(char in[ ], char out[ ], int n)
{
  int S[n], top = -1, i, j = 0;
  for (i = 0; i < n; i++)
  {
    S[++top] = in[i];
    while (top != -1 && S[top] == out[j])
    {
      top--; j++;
    }
  }
  if (top == -1) return 1;
  else return 0;
}
```

7. 双端队列 Q 限定在线性表的两端进行插入和删除操作，若采用顺序存储结构存储双端队列，请设计算法实现在指定端 L(表示左端)和 R(表示右端)执行入队操作。

【解答】 采用循环队列的存储思想，设 left 指向队列左端第一个元素(可以看成是左端的队头)的前一个位置，right 指向右端第一个元素(可以看成是右端的队头)的位置。队列为空的条件是 left = right，队列为满的条件是(right+1) mod QueueSize = left。双端队列的存储结构定义和入队算法如下：

```
enum OpertionTag {L, R};                          //L 表示队列的左端,R 表示队列的右端
#define QueueSize 100
typedef int DataType;
typedef struct                                    // 定义双端队列存储结构
{
  DataType data[QueueSize];
  int left, right;
} DulQueue;
int QueueInsert (DulQueue *Q, DataType x, OpertionTag side)
{
  if ((Q->right+1) % QueueSize == Q->left) {printf("上溢"); return 0; }
  if (side == L)
  {
    Q->data[Q->left] = x;                         //left 已经指向插入的位置
    Q->left = (Q->left - 1) % QueueSize;          //左端指针在循环意义下减 1
  }
  else
  {
    Q->right = (Q->right + 1) % QueueSize;        //右端指针在循环意义下加 1
    Q->data[Q->right] = x;
```

```
    }
    return 1;
}
```

8. 若在矩阵 $\boldsymbol{A}$ 中存在一个元素 a_{ij}（$1\leqslant i\leqslant n$，$1\leqslant j\leqslant m$），该元素是第 i 行的最小值，同时又是第 j 列的最大值，则称此元素为矩阵的一个鞍点。设计算法求矩阵 $\boldsymbol{A}$ 的鞍点个数，并分析时间复杂度。

【解答】 在矩阵 $\boldsymbol{A}$ 中逐行查找该行的最小值，然后判断该元素是否是所在列的最大值。设变量 count 累计鞍点个数，变量 k 记载某行最小值的列下标，算法如下：

```
int Andian(int A[100][100], int m, int n)
{
  int i, j, k, min, count = 0;
  for (i = 0; i < n; i++)
  {
    min = A[i][0]; k = 0;                          //min 为第 i 行的最小值
    for (j = 1; j < m; j++)
      if (A[i][j] < min) { min = a[i][j]; k = j; }
    for (j = 0; j < n; j++)
      if (a[j][k] > min) break;
    if (j == n) count++;
  }
  return count;
}
```

外层 for 循环共执行 n 次，内层第一个 for 循环执行 m 次，第二个 for 循环最坏情况下执行 n 次，因此时间复杂度为 $O(n^2+mn)$。

9. 假设矩阵 $\boldsymbol{A}$ 满足 $A[i][j]\leqslant A[i][j+1](0\leqslant i\leqslant n,0\leqslant j\leqslant m-1)$ 和 $A[i][j]\leqslant A[i+1][j](0\leqslant i\leqslant n-1,0\leqslant j\leqslant m)$。给定元素值 x，设计算法判断 x 是否在矩阵 $\boldsymbol{A}$ 中，要求时间复杂度为 $O(m+n)$。

【解答】 本题要求时间复杂度为 $O(m+n)$，因此不能采用常规的二层循环进行查找。由于矩阵元素分别按行和按列排序，可以从右上角开始，将元素 $A[i][j]$ 与 x 进行比较，有以下 3 种情况：①$A[i][j]>x$，向左继续查找；②$A[i][j]<x$，向下继续查找；③$A[i][j]=x$，查找成功。如果下标超出矩阵范围，则查找失败。算法如下：

```
int Search(int A[100][100], int n, int m, int x)
{
  int i = 0, j =m-1, flag = 0;                     //flag 是查找成功的标志
  while (i < n && j >= 0)
    if (A[i][j] == x) { flag = 1; break; }
    else if (A[i][j] > x) j--;
    else i++;
  if (1 == flag) {
    printf("元素%d 在矩阵中的位置是:(%2d, %2d)\n", x, i, j);
```

```
        return 1;
      }
      else return 0;
    }
```

查找 x 的路线从矩阵 $\boldsymbol{A}$ 的右上角开始，向下（当 $x>A[i][j]$ 时）或向左（当 $x<A[i][j]$ 时）。向下最多执行 m 次，向左最多执行 n 次。最好情况是元素 x 在右上角 $A[0][m-1]$，比较 1 次；最坏情况是元素 x 在左下角 $A[n-1][0]$，比较 $m+n$ 次。因此，时间复杂度是 $O(m+n)$。