# 第3章 栈、队列和数组

CHAPTER

## 3.1 考纲要求及分析

### 考纲要求

(1) 栈和队列的基本概念。
(2) 栈和队列的顺序存储结构。
(3) 栈和队列的链式存储结构。
(4) 栈和队列的应用。
(5) 特殊矩阵的压缩存储。

### 考纲分析

本章是必考内容，出题形式主要以选择题为主。本章要求：

(1) 理解栈和队列的定义及其操作特性，掌握栈和队列对插入和删除操作的定义。

(2) 掌握顺序栈、链栈、共享栈、顺序队列、循环队列、链队列的存储方法，以及栈空、栈满、队空、队满的判定条件。

(3) 掌握栈和队列的插入、删除、判空等基本操作的算法描述和时间性能。

(4) 理解栈和队列的应用，例如子程序调用、表达式求值、括号匹配等。

(5) 理解数组的定义和存储方法，注意数组元素具有相同的数据类型，因此，每个元素占用的存储单元数相同。

(6) 掌握特殊矩阵的压缩存储方法，以及特殊矩阵压缩存储后矩阵元素的寻址问题。

对于栈，常考的一类题是考查栈的后进先出特性，例如给定一个入栈序列，判断某个出栈序列的合法性或不合法性，共享栈也是一个常考点。对于队列，循环队列是一个常考点，注意队空、队满的判定条件，以及队列长度的计算。

本章有一个结合点是将栈、队列、链表和数组相结合，主要考查栈和队列的操作特性，以及链表和数组的存储特点；有一个重要应用是递归，主要考查栈在递归调用过程中的作用，以及应用栈实现从递归函数到非递归函数的转换。

由于栈和队列的算法比较简单，通常不会单独以算法设计题的形式出题，在树和图的算法设计中，栈和队列通常作为辅助数据结构，因此，需要熟练掌握栈和队列的基本操作语句。

对于特殊矩阵压缩存储后矩阵元素的寻址，要注意掌握寻址方法，不要死记公式。只有掌握了存储方法和寻址方法，才能对任意给定的数组下标范围灵活处理。

## 3.2 栈

### 3.2.1 考核知识点

**1. 栈的定义（★☆☆☆☆ ◆◇◇◇◇）**

**栈**是限定仅在表的一端进行插入和删除操作的线性表。允许插入和删除的一端称为栈顶，另一端称为栈底，不含任何数据元素的栈称为空栈。

**【说明】** 插入操作也称为入栈或压栈，删除操作也称为出栈或弹栈。

**2. 栈的操作特性（★★★★★ ◆◆◇◇◇）**

栈的操作具有后进先出的特性。

**3. 顺序栈及其实现（★★★☆☆ ◆◇◇◇◇）**

（1）存储结构定义

栈的顺序存储结构称为**顺序栈**，通常把数组中下标为0的一端作为栈底，同时附设指针top指示栈顶元素在数组中的位置。设数组长度为StackSize，其存储结构定义如下：

```
#define StackSize 100
typedef struct
{
  ElemType data[StackSize];          //ElemType 表示不确定的数据类型
  int top;                           //top 为栈顶元素所在的数组下标
} SeqStack;
```

（2）基本操作的实现

顺序栈的基本操作本质上是顺序表基本操作的简化，且时间复杂度均为$O(1)$。

顺序栈入栈算法 Push

```
void Push(SeqStack &S,ElemType x)
{
  if(S.top==StackSize-1) printf("上溢");
  else S.data[++S.top]=x;
}
```

顺序栈出栈算法 Pop

```
ElemType Pop(SeqStack &S)
{
  if(S.top==-1) printf("下溢");
  else {
    x=S.data[S.top--];
    return x;
  }
}
```

**4. 两栈共享空间(★★★☆☆ ◆◆◇◇◇)**

(1) 存储结构定义

在一个程序中如果同时使用具有相同数据类型的两个栈,可以充分利用顺序栈单向延伸的特性,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,每个栈从各自的端点向中间延伸,如图 3-1 所示。

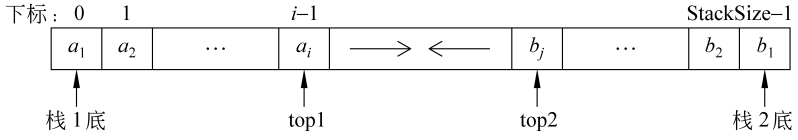



图 3-1　两栈共享空间示意图

两栈共享空间的存储结构定义如下：

```
#define StackSize 100                    //整个数组空间的大小
typedef struct
{
  ElemType data[StackSize];              //ElemType 表示不确定的数据类型
  int top1,top2;                         //top1 和 top2 分别为栈 1 和栈 2 的栈顶指针
} BothStack;
```

(2) 基本操作的实现

设 $i$ 表示整型数值,当 $i=1$ 时,表示对栈 1 进行操作,当 $i=2$ 时表示对栈 2 进行操作。当 top1＝－1 时,栈 1 为空;当 top2＝StackSize 时,栈 2 为空;当 top1＝top2－1(或 top2＝top1＋1)时为栈满。另外,当新元素压入栈 2 时,栈顶指针 top2 不是加 1 而是减 1;当从栈 2 删除元素时,top2 不是减 1 而是加 1。

两栈共享空间入栈算法 Push

```
void Push(BothStack &S,int i,ElemType x)
{
   if(S.top1==S.top2-1) printf("上溢");
   else {
```

```
        if(i==1) S.data[++S.top1]=x;                    //在栈1插入
        if(i==2) S.data[--S.top2]=x;                    //在栈2插入
    }
}
```

两栈共享空间出栈算法 Pop

```
ElemType Pop(BothStack &S,int i)
{
    if(i==1) {                                          //将栈1的栈顶元素出栈
        if(S.top1==-1) printf("下溢");
        else return S.data[S.top1--];
    }
    if(i==2) {                                          //将栈2的栈顶元素出栈
        if(S.top2==StackSize) printf("下溢");
        else return S.data[S.top2++];
    }
}
```

**【说明】** 只有当两个栈的空间需求有相反的关系时，两栈共享空间才会奏效，也就是说，最好一个栈增长时另一个栈缩短。

**5. 链栈及其实现（★★★☆☆ ◆◆◇◇◇）**

（1）存储结构定义

栈的链接存储结构称为**链栈**。通常链栈用单链表表示，其结点结构与单链表相同。

**【说明】** 链栈没有必要像单链表那样为了运算方便而附加一个头结点。

（2）基本操作的实现

链栈的插入和删除操作只需处理栈顶（即第一个位置）的情况，时间复杂度均为$O(1)$。

链栈插入算法 Push

```
void Push(Node *top,ElemType x)
{
  s=(Node *)malloc(sizeof(Node));
  s->data=x;                                //申请一个数据域为x的结点s
  s->next=top; top=s;                       //将结点s插在栈顶
}
```

链栈删除算法 Pop

```
ElemType Pop(Node *top)
{
  if(top==NULL) printf("下溢");
```

```
  else {
    x=top->data;                    //暂存栈顶元素
    p=top; top=top->next;           //将栈顶结点摘链
    free(p);
    return x;
  }
}
```

### 3.2.2　典型题解析

**1. 选择题**

试题点拨　主要考查栈的基本操作(初始化、进栈、出栈、判空等)在不同存储结构(顺序栈和链栈)下的执行过程,对于顺序栈,注意栈底的位置和栈顶指针的变化,对于链栈,注意如何用链表实现栈以及插入和删除操作的位置。栈最重要的考点是元素以同样顺序进栈后判断出栈的不同情况。两栈共享空间也是一个常见的考点,注意存储方法、栈底的位置和栈顶指针的变化。

(1) 经过以下栈运算后,x 的值是(　　)。

```
InitStack(s); Push(s,a); Push(s,b); Pop(s,x); GetTop(s,x);
```

A. a　　B. b　　C. 1　　D. 0

**【解答】** A

**【分析】** 本题要求熟悉栈的基本操作,理解所给运算的含义。InitStack(s)表示对栈 s 进行初始化;Push(s,a)表示将元素 a 压入栈 s 中;Pop(s,x)表示将栈 s 的栈顶元素弹出并送入变量 x 中;GetTop(s,x)表示取栈顶元素并送入变量 x 中但不删除该元素。

(2) 经过以下栈运算后,StackEmpty(s)的值是(　　)。

```
InitStack(s); Push(s,a); Push(s,b); Pop(s,x); Pop(s,y);
```

A. a　　B. b　　C. 1　　D. 0

**【解答】** C

**【分析】** 注意 StackEmpty(s)的返回值,如果栈 s 为空,则返回 1,否则返回 0。

(3) 一个栈的入栈序列是{1,2,3,4,5},则栈的输出序列不可能是(　　)。

A. {5,4,3,2,1}　B. {4,5,3,2,1}　C. {4,3,5,1,2}　D. {1,2,3,4,5}

**【解答】** C

**【分析】** 解答此题有一个技巧:在输出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。

(4) 若一个栈的输入序列是 1,2,3,…,$n$,输出序列的第一个元素是 $n$,则第 $i$ 个输出元素是(　　)。

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

**【解答】** D

**【分析】** 此时,输出序列一定是输入序列的逆序。

（5）若一个栈的输入序列是1,2,3,…,$n$,其输出序列是 $p_1,p_2,\cdots,p_n$,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。

（6）当字符序列 t3_依次通过栈,输出长度为3且可用做C语言标识符的序列有(　　)。

A. 4个　　B. 5个　　C. 3个　　D. 6个

【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或下划线“_”开头,而栈的输出序列中以字母t或下划线“_”开头的有三个,分别是t3_、t_3和_3t。

（7）在一个具有 $n$ 个单元的顺序栈中,假定以地址低端(即下标为0的单元)作为栈底,以top作为栈顶指针,当出栈时,top的变化为(　　)。

A. 不变　　B. top=0;　　C. top=top-1;　　D. top=top+1;

【解答】 C

【分析】 栈底固定在数组的低端,出栈时栈顶指针top需要减1;如果栈底固定在数组的高端,则出栈时栈顶指针需要加1。

（8）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(　　)。

A. h->next=s;　　B. s->next=h;

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

【解答】 D

【分析】 结点s应插在头结点的后面。

（9）从栈顶指针为top的链栈中删除一个结点,用x保存被删除结点的值,则执行(　　)。

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】 删除链栈中的第一个结点,即指针top指向的结点。备选答案A保存的是栈顶指针;备选答案B只保存了被删结点的值,没有执行删除操作;备选答案C保存的是被删结点的后继结点的值。

（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[$n$]全满时才不能进行进栈操作。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】** 两栈共享空间中两个栈是相向增长的，栈底应该分别指向两个栈中的第一个元素的位置，并注意题目表明数组下标是从 0 开始的。

## 3.3 队列

### 3.3.1 考核知识点

**1. 队列的定义（★☆☆☆☆ ◆◇◇◇◇）**

队列是只允许在一端进行插入操作，而另一端进行删除操作的线性表。允许插入的一端称为队尾，允许删除的一端称为队头。

插入操作也称入队或进队，删除操作也称出队。

**2. 队列的操作特性（★★★★☆ ◆◆◇◇◇）**

队列的操作具有先进先出的特性。

**3. 队列的顺序存储及其实现（★★★★★ ◆◆◇◇◇）**

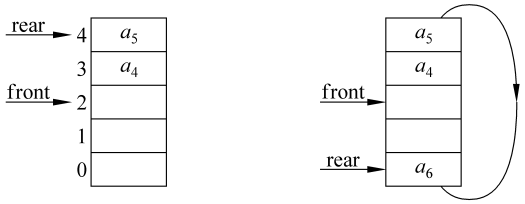
（1）顺序队列

顺序队列是用数组存放队列元素，通常设置队头、队尾两个指针，并且约定：队头指针 front 指向队头元素的前一个位置，队尾指针 rear 指向队尾元素。

**【说明】** 还有其他约定方式，例如队头指针 front 指向队头元素，队尾指针 rear 指向队尾元素；或队头指针 front 指向队头元素，队尾指针 rear 指向队尾元素的后一个位置。

（2）顺序队列的“假溢出”现象

随着插入和删除操作的进行，整个队列向数组中下标较大的位置移过去，从而产生了队列的“单向移动性”。当元素被插入到数组中下标最大的位置上之后，队列的空间就用尽了，尽管此时数组的低端还有空闲空间。这种现象叫做“假溢出”，如图 3-2(a)所示。




(a) front=2 rear=5 假溢出　　(b) front=2 rear=0 解决假溢出

图 3-2　顺序队列的假溢出及其解决方法

（3）循环队列

解决假溢出的方法是将存储队列的数组看成是头尾相接的循环结构，即允许队列直接从数组中下标最大的位置延续到下标最小的位置，如图 3-2(b)所示。队列的这种头尾

相接的顺序存储结构称为**循环队列**（也称环形队列）。设数组长度为 QueueSize，循环队列的存储结构定义如下：

```
#define QueueSize 100
typedef struct
{
  ElemType data[QueueSize];                    //ElemType 表示不确定的数据类型
  int front,rear;
} CirQueue;
```

循环队列中，在浪费一个数组元素空间的情况下，队空的条件是 front＝rear，队满的条件是(rear＋1)％QueueSize＝front，队列长度为(rear－front＋QueueSize)％QueueSize。

(4) 循环队列基本操作的实现

循环队列基本操作的实现非常简单，且时间复杂度均为 $O(1)$。

循环队列入队算法 EnQueue

```
void EnQueue(CirQueue &Q,ElemType x)
{
   if((Q.rear+1) %QueueSize==Q.front) printf("上溢");
   else {
      Q.rear=(Q.rear+1) %QueueSize;          //队尾指针在循环意义下加 1
      Q.data[Q.rear]=x;                      //在队尾处插入元素
   }
}
```

循环队列出队算法 DeQueue

```
ElemType DeQueue(CirQueue &Q)
{
   if(Q.rear==Q.front) printf("下溢");
   else {
     Q.front=(Q.front+1) %QueueSize;         //队头指针在循环意义下加 1
     return Q.data[Q.front];                 //队头指针加 1 后指向被删除元素
  }
}
```

**4. 队列的链接存储及其实现**（★★★☆☆ ◆◆◇◇◇）

(1) 存储结构定义

队列的链接存储结构称为**链队列**，根据队列的先进先出原则，为了操作方便，设置队头指针指向链队列的头结点，队尾指针指向终端结点。为了使空队列和非空队列的操作一致，链队列也加上头结点。链队列的存储结构定义如下：

```
typedef struct Node
{
    ElemType data;                    //ElemType 表示不确定的数据类型
    struct Node * next;
} Node;
typedef struct
{
  Node * front, * rear;
}LinkQueue;
```

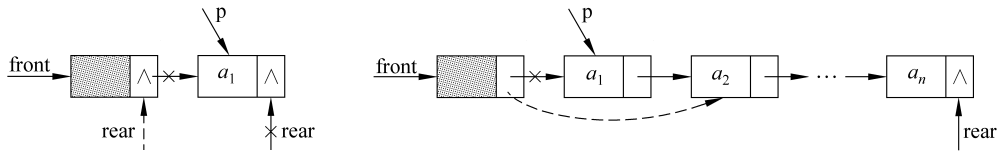
(2) 基本操作的实现

链队列的基本操作本质上是单链表基本操作的简化，插入操作只考虑在尾部进行，删除操作只考虑在头部进行。

链队列入队算法 EnQueue

```
void EnQueue(LinkQueue &Q,ElemType x)
{
   s=(Node * )malloc(sizeof(Node));
   s->data=x;                         //申请一个数据域为 x 的结点 s
   s->next=NULL;
   Q.rear->next=s;                    //将结点 s 插入到队尾
   Q.rear=s;
}
```

链队列的出队操作需要注意在队列长度为 1 时的特殊处理，其操作过程如图 3-3 所示。




(a) 特殊情况：队列长度等于1　　(b) 一般情况：队列长度大于 1

图 3-3　链队列出队操作示意图

链队列出队算法 DeQueue

```
ElemType DeQueue(LinkQueue &Q)
{
  if(Q.rear==Q.front) printf("下溢");
  else {
    p=Q.front->next; x=p->data;          //暂存队头元素
    Q.front->next=p->next;               //将队头元素所在结点摘链
    if(p->next==NULL) Q.rear=Q.front;    //判断出队前队列长度是否为 1
```

```
    free(p);
    return x;
    }
}
```

### 3.3.2 典型题解析

**1. 选择题**

试题点拨 主要考查队列的基本操作（初始化、入队、出队、判空等）在不同存储结构（循环队列和链队列）下的执行过程，循环队列是本章一个重要的考点，注意循环的含义，在循环队列中对指针的操作都需要与数组空间的长度取模。

（1）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。

（2）循环队列存储在数组 $A[0..m]$ 中，则入队时的操作为（　　）。

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】** 注意数组空间的长度为 $m+1$，因此 rear+1 的结果要与 $m+1$ 取模。

（3）若用一个长度为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，则从队列中删除一个元素，再增加两个元素后，rear 和 front 的值分别为（　　）。

A. 1和5　　B. 2和4　　C. 4和2　　D. 5和1

**【解答】** B

**【分析】** 从队列中删除一个元素时，执行 front=(front+1)mod 6，向队列中插入一个元素时，执行 rear=(rear+1) mod 6。

（4）已知循环队列的存储空间为数组 $A[21]$，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. 6　　C. 16　　D. 17

**【解答】** C

**【分析】** 队列长度为(rear−front+21)mod 21=16，此时队列元素在数组中的位置依次是 $A[9],A[10],\cdots,A[20],A[0],A[1],\cdots,A[3]$。

（5）用不带头结点的单链表存储队列时，其队头指针指向队头结点，队尾指针指向队尾结点，则执行删除操作时，（　　）。

A. 仅修改队首指针　　B. 仅修改队尾指针

C. 队首指针和队尾指针都需要修改　　D. 队首指针和队尾指针可能都需要修改

**【解答】** D

**【分析】** 链队列不带头结点，所以删除队头元素一定要修改队首指针，注意到队列长