第3章 栈和队列

3.1 栈

3.1.1 栈的基本概念

栈是一种特殊的线性表，其插入、删除操作只能在表的尾部进行。在栈中允许进行插入、删除操作的一端称为栈顶，另一端称为栈底。在栈$\{a_0, a_1, \cdots, a_{n-1}\}$中$a_0$称为栈底元素，$a_{n-1}$称为栈顶元素。通常，栈的插入操作称为入栈，栈的删除操作称为出栈。

由于栈的插入和删除操作只允许在栈顶进行，每次入栈的元素即成为栈顶元素，每次最先出栈的总是栈顶元素，因此栈是一种后进先出的线性表。就像一摞盘子，每次将一个盘子摞在最上面，每次从最上面取一只盘子，不能从中间插进或者抽出。

3.1.2 栈的抽象数据类型描述

栈中的数据元素和数据间的逻辑关系与线性表相同，是由n个具有相同数据类型的数据元素构成的有限序列，栈的抽象数据类型的Python语言描述如下：

```
1  from abc import ABCMeta,abstractmethod,abstractproperty
2
3  class IStack(metaclass = ABCMeta):
4      @abstractmethod
5      def clear(self):
6          '''将栈置空'''
7          pass
8      @abstractmethod
9      def isEmpty(self):
10         '''判断栈是否为空'''
11         pass
12     @abstractmethod
13     def length(self):
14         '''返回栈的数据元素个数'''
15         pass
16     @abstractmethod
17     def peek(self):
18         '''返回栈顶元素'''
19         pass
20     @abstractmethod
21     def push(self,x):
```

```
        '''数据元素 x 入栈'''
        pass
    @abstractmethod
    def pop(self):
        '''将栈顶元素出栈并返回'''
        pass
    @abstractmethod
    def display(self):
        '''输出栈中的所有元素'''
        pass
```

栈的抽象数据 Python 抽象类包含了栈的主要基本操作,如果要使用这个类还需要具体的类来实现。栈的 Python 抽象类的实现方法主要有以下两种。

(1) 基于顺序存储的实现,为顺序栈。

(2) 基于链式存储的实现,为链栈。

3.1.3 顺序栈

1. 顺序栈类的描述

顺序栈用数组实现,因为入栈和出栈操作都是在栈顶进行的,所以增加变量 top 来指示栈顶元素的位置,top 指向栈顶元素存储位置的下一个存储单元的位置,空栈时 top=0。

顺序栈类的 Python 语言描述如下:

```
class SqStack(IStack):
    def __init__(self,maxSize):
        self.maxSize = maxSize # 栈的最大存储单元个数
        self.stackElem = [None] * self.maxSize # 顺序栈存储空间
        self.top = 0 # 指向栈顶元素的下一个存储单元位置

    def clear(self):
        '''将栈置空'''
        self.top = 0

    def isEmpty(self):
        '''判断栈是否为空'''
        return self.top == 0

    def length(self):
        '''返回栈的数据元素个数'''
        return self.top

    def peek(self):
        '''返回栈顶元素'''
        if not self.isEmpty():
            return self.stackElem[self.top - 1]
        else:
            return None
    def push(self,x):
        '''数据元素 x 入栈'''
```

```
27            pass
28
29        def pop(self):
30            '''将栈顶元素出栈并返回'''
31            pass
32
33        def display(self):
34            '''输出栈中的所有元素'''
35            for i in range(self.top - 1, - 1, - 1):
36                print(self.stackElem[i],end = ' ')
```

2. 顺序栈基本操作的实现

1) 入栈操作

入栈操作 push(x)是将数据元素 x 作为栈顶元素插入顺序栈中,主要操作如下。

(1) 判断顺序栈是否为满,若满则抛出异常。

(2) 将 x 存入 top 所指的存储单元位置。

(3) top 加 1。

图 3.1 显示了进行入栈操作时栈的状态变化。

【算法 3.1】 入栈操作。

```
1   def push(self,x):
2       '''数据元素 x 入栈'''
3       if self.top == self.maxSize:
4           raise Exception("栈已满")
5       self.stackElem[self.top] = x
6       self.top += 1
```

2) 出栈操作

出栈操作 pop()是将栈顶元素从栈中删除并返回,主要步骤如下。

(1) 判断顺序栈是否为空,若空则返回 None。

(2) top 减 1。

(3) 返回 top 所指的栈顶元素的值。

图 3.2 显示了进行出栈操作时栈的状态变化。

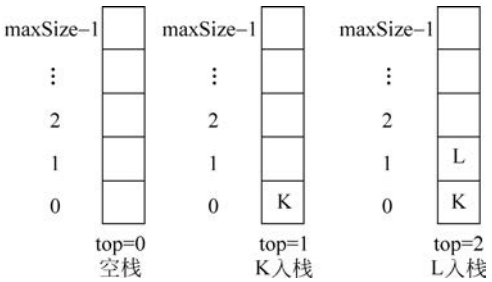

图 3.1 入栈操作——顺序栈

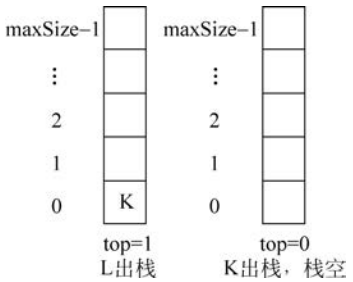

图 3.2 出栈操作——顺序栈

【算法 3.2】 出栈操作。

```
1   def pop(self):
```

```
    '''将栈顶元素出栈并返回'''
    if self.isEmpty():
        return None
    self.top -= 1
    return self.stackElem[self.top]
```

分析可得,入栈和出栈操作的实现为顺序表的尾插入和尾删除,时间复杂度为 $O(1)$。

【例 3.1】 利用顺序栈实现括号匹配的语法检查。

解:括号匹配是指程序中出现的括号,左、右括号的个数是相同的,并且需要先左后右依次出现。括号是可以嵌套的,一个右括号与其前面最近的一个左括号匹配,使用栈保存多个嵌套的左括号。

```
def isMatched(str):
    s = SqStack(100)
    for c in str:
        if c == '(':
            s.push('(')
        elif c == ')' and not s.isEmpty():
            s.pop()
        elif c == ')' and s.isEmpty():
            print("括号不匹配")
            return False
    if s.isEmpty():
        print("括号匹配")
        return True
    else:
        print("括号不匹配")
        return False
```

3.1.4 链栈

1. 链栈类的描述

采用链式存储结构的栈称为链栈。由于入栈和出栈只能在栈顶进行,不存在在栈的任意位置进行插入和删除的操作,因此不需要设置头节点,只需要将指针 top 指向栈顶元素节点,每个节点的指针域指向其后继节点即可。

链栈的存储结构如图 3.3 所示。

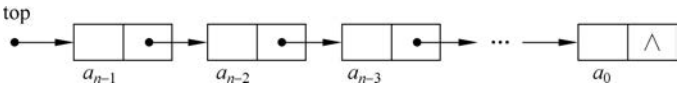

图 3.3 链栈的存储结构

实现 IStack 抽象类的链栈类的 Python 语言描述如下:

```
class Node(object):
    def __init__(self,data = None,next = None):
        self.data = data
        self.next = next

```

```
class LinkStack(IStack):
    def __init__(self):
        self.top = None

    def clear(self):
        '''将栈置空'''
        self.top = None

    def isEmpty(self):
        '''判断栈是否为空'''
        return self.top is None

    def length(self):
        '''返回栈的数据元素个数'''
        i = 0
        p = self.top
        while p is not None:
            p = p.next
            i += 1
        return i

    def peek(self):
        '''返回栈顶元素'''
        return self.top

    def push(self,x):
        '''数据元素 x 入栈'''
        s = Node(x,self.top)
        self.top = s

    def pop(self):
        '''将栈顶元素出栈并返回'''
        if self.isEmpty():
            return None
        p = self.top
        self.top = self.top.next
        return p.data

    def display(self):
        '''输出栈中的所有元素'''
        p = self.top
        while p is not None:
            print(p.data,end = ' ')
            p = p.next
```

2. 链栈基本操作的实现

1）入栈操作

入栈操作 push(x)是将数据域值为 x 的节点插入链栈的栈顶，主要步骤如下。

(1) 构造数据值域为 x 的新节点。

(2) 改变新节点和首节点的指针域，使新节点成为新的栈顶节点。

链栈进行入栈操作后的状态变化如图 3.4 所示。

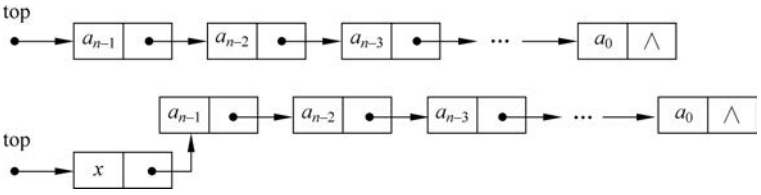

图 3.4　入栈操作——链栈

【算法 3.3】 入栈操作。

```
1  def push(self,x):
2      '''数据元素 x 入栈'''
3      s = Node(x,self.top)
4      self.top = s
```

2) 出栈操作

出栈操作 pop()是将栈顶元素从链栈中删除并返回其数据域的值,主要步骤如下。

(1) 判断链栈是否为空,若空则返回 None。

(2) 修改 top 指针域的值,返回被删节点的数据域值。

链栈进行出栈操作后的状态变化如图 3.5 所示。

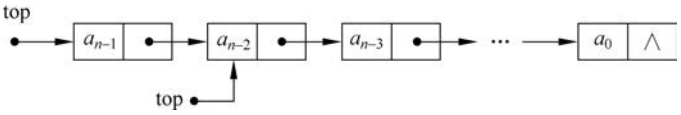

图 3.5　出栈操作——链栈

【算法 3.4】 出栈操作。

```
1  def pop(self):
2      '''将栈顶元素出栈并返回'''
3      if self.isEmpty():
4          return None
5      p = self.top
6      self.top = self.top.next
7      return p.data
```

分析可得,使用单链表实现栈,入栈和出栈操作的实现为单链表的头插入和头删除,时间复杂度为 $O(1)$。

【例 3.2】 设有编号为 1、2、3、4 的 4 辆列车顺序进入一个栈式结构的车站,具体写出这 4 辆列车开出车站的所有可能的顺序。

解:共 14 种。

全进之后再出的情况只有 1 种:4,3,2,1。

进 3 个之后再出的情况有 3 种:3,4,2,1; 3,2,4,1; 3,2,1,4。

进两个之后再出的情况有 5 种:2,4,3,1; 2,3,4,1; 2,1, 3,4; 2,1,4,3; 2,3,1,4。

进一个之后再出的情况有 5 种:1,4,3,2; 1,3,2,4; 1,3,4,2; 1,2,3,4; 1,2,4,3。

【例 3.3】 在执行操作序列 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？

解：操作序列的执行过程如图 3.6 所示。

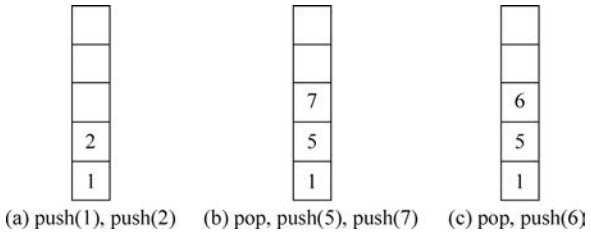

图 3.6 操作序列的执行过程

所以栈顶元素为 6，栈底元素为 1。


观看视频

【例 3.4】 编程实现汉诺塔问题的求解。假设有 3 个分别命名为 x、y、z 的塔座，在塔座 x 上从上到下插有 n 个直径和序号均为 1,2,…,n 的圆盘。要求将塔座 x 上的 n 个圆盘借助塔座 y 移动到塔座 z 上，仍按照相同的序号排列，并且每次只能移动一个圆盘，在任何时候都不能将一个较大的圆盘压在较小的圆盘之上。

解：分析问题可知，当 $n=1$ 时只要将序号为 1 的圆盘从 x 直接移动到 z 即可；当 $n>1$ 时则需要将序号小于 n 的 $n-1$ 个圆盘移动到 y 上，再将序号为 n 的圆盘移动到 z 上，然后将 y 上的 $n-1$ 个圆盘移动到 z 上。如何将 $n-1$ 个圆盘移动到 z 上是一个和原问题相似的问题，只是规模变小，可以用同样的方法求解。代码如下：

```
1  def move(s,t):
2      print("将",s,"塔座上最顶端圆盘移动到",t,"塔座上")
3
4  def hanoi(n,x,y,z):
5      if n == 1:
6          move(x,z)
7      else:
8          hanoi(n-1,x,z,y) # 将 x 上 n-1 个圆盘从 x 移动到 y
9          move(x,z) # 将 1 个圆盘从塔座 x 移动到塔座 z
10         hanoi(n-1,y,x,z) # 将 y 上 n-1 个圆盘从 y 移动到 z
11
12 hanoi(3,'x','y','z')
```

3.2 队　　列

3.2.1 队列的基本概念

队列是一种特殊的线性表，其插入操作只能在表的尾部进行，删除操作只能在表头进行。在队列中允许进行插入操作的一端称为队尾，允许进行删除操作的另一端称为队首。在队列$\{a_0,a_1,\cdots,a_{n-1}\}$中 a_0 称为队首元素，a_{n-1} 称为队尾元素。通常，队列的插入操作称为入队，队列的删除操作称为出队。没有数据元素的队列称为空队列。

由于插入和删除操作分别在队尾和队首进行,最先入队的元素总是最先出队,因此队列具有先进先出的特点。

3.2.2 队列的抽象数据类型描述

队列中的数据元素和数据间的逻辑关系与线性表相同,是由 n 个具有相同数据类型的数据元素构成的有限序列,队列的抽象数据类型的 Python 语言描述如下:

```
from abc import ABCMeta,abstractmethod,abstractproperty

class IQueue(metaclass = ABCMeta):
    @abstractmethod
    def clear(self):
        '''将队列置空'''
        pass
    @abstractmethod
    def isEmpty(self):
        '''判断队列是否为空'''
        pass
    @abstractmethod
    def length(self):
        '''返回队列的数据元素个数'''
        pass
    @abstractmethod
    def peek(self):
        '''返回队首元素'''
        pass
    @abstractmethod
    def offer(self,x):
        '''将数据元素 x 插入队列成为队尾元素'''
        pass
    @abstractmethod
    def poll(self):
        '''将队首元素删除并返回其值'''
        pass
    @abstractmethod
    def display(self):
        '''输出队列中的所有元素'''
        pass
```

队列的抽象数据 Python 抽象类包含了队列的主要基本操作,如果要使用这个接口,还需要具体的类来实现。Python 抽象类的实现方法主要有以下两种。

(1) 基于顺序存储的实现,为顺序队列。

(2) 基于链式存储的实现,为链队列。

3.2.3 顺序队列

1. 顺序队列类的描述及实现

顺序队列的存储结构与顺序栈类似,可用数组实现,因为入队和出队操作分别在队尾和

队首进行，所以增加变量 front 指示队首元素的位置，rear 指示队尾元素的下一个存储单元的位置。顺序队列进行入队操作的状态变化如图 3.7 所示，进行出队操作后的状态变化如图 3.8 所示。

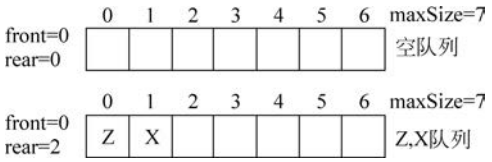

图 3.7　入队操作——顺序队列

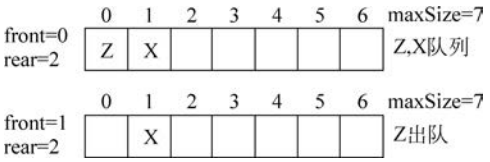

图 3.8　出队操作——顺序队列

顺序队列类的 Python 语言描述如下：

```
class SqQueue(IQueue):
    def __init__(self,maxSize):
        self.maxSize = maxSize # 队列的最大存储单元个数
        self.queueElem = [None] * self.maxSize # 队列的存储空间
        self.front = 0 # 指向队首元素
        self.rear = 0 # 指向队尾元素的下一个存储单元

    def clear(self):
        '''将队列置空'''
        self.front = 0
        self.rear = 0

    def isEmpty(self):
        '''判断队列是否为空'''
        return self.rear == self.front

    def length(self):
        '''返回队列的数据元素个数'''
        return self.rear - self.front

    def peek(self):
        '''返回队首元素'''
        if self.isEmpty():
            return None
        else:
            return self.queueElem[self.front]

    def offer(self,x):
        '''将数据元素 x 插入队列成为队尾元素'''
        if self.rear == self.maxSize:
            raise Exception("队列已满")
        self.queueElem[self.rear] = x
        self.rear += 1

    def poll(self):
        '''将队首元素删除并返回其值'''
        if self.isEmpty():
```

```
            return None
        p = self.queueElem[self.front]
        self.front += 1
        return p

    def display(self):
        '''输出队列中的所有元素'''
        for i in range(self.front,self.rear):
            print(self.queueElem[i],end = ' ')
```

2. 循环顺序队列类的描述及实现

分析发现,顺序队列的多次入队和出队操作会造成有存储空间却不能进行入队操作的"假溢出"现象,如图 3.9 所示。顺序队列之所以会出现"假溢出"现象是因为顺序队列的存储单元没有重复使用机制。为了解决顺序队列因数组下标越界而引起的"溢出"问题,可将顺序序列的首尾相连,形成循环顺序队列。循环顺序队列进行入队和出队操作后的状态变化如图 3.10 所示。

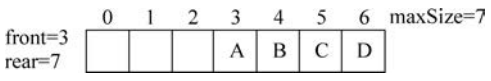

图 3.9 "假溢出"现象

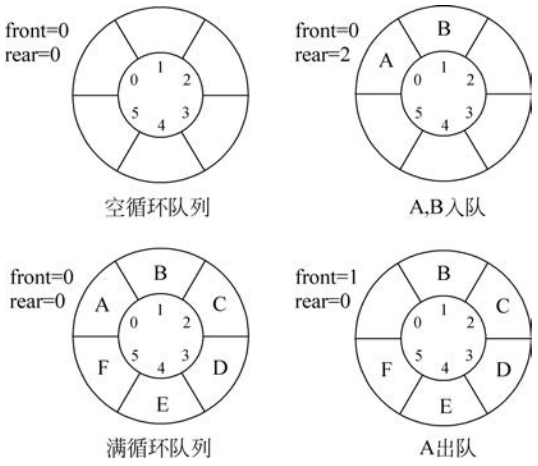

图 3.10 循环顺序队列入队和出队操作

有新的问题产生——队空和队满的判定条件都变为 front==rear。为了解决这一问题,可少利用一个存储单元,队列最多存放 maxSize−1 个数据元素,队空的判定条件为 front==rear,队满的判定条件为 front==(rear+1)%maxSize。

循环顺序队列类和顺序队列类的 Python 语言描述相似,仅是指示变量 front 和 rear 的修改以及队满的判定条件发生了变化。

循环顺序队列的 Python 语言描述如下:

```
class CircleQueue(IQueue):
    def __init__(self,maxSize):
        self.maxSize = maxSize # 队列的最大存储单元个数
        self.queueElem = [None] * self.maxSize # 队列的存储空间
```

```
        self.front = 0 # 指向队首元素
        self.rear = 0 # 指向队尾元素的下一个存储单元

    def clear(self):
        '''将队列置空'''
        self.front = 0
        self.rear = 0

    def isEmpty(self):
        '''判断队列是否为空'''
        return self.rear == self.front

    def length(self):
        '''返回队列的数据元素个数'''
        return (self.rear - self.front + self.maxSize) % self.maxSize

    def peek(self):
        '''返回队首元素'''
        if self.isEmpty():
            return None
        else:
            return self.queueElem[self.front]

    def offer(self,x):
        '''将数据元素 x 插入队列成为队尾元素'''
        if (self.rear + 1) % self.maxSize == self.front:
            raise Exception("队列已满")
        self.queueElem[self.rear] = x
        self.rear = (self.rear + 1) % self.maxSize

    def poll(self):
        '''将队首元素删除并返回其值'''
        if self.isEmpty():
            return None
        p = self.queueElem[self.front]
        self.front = (self.front + 1) % self.maxSize
        return p

    def display(self):
        '''输出队列中的所有元素'''
        i = self.front
        while i!= self.rear:
            print(self.queueElem[i],end = ' ')
            i = (i + 1) % self.maxSize
```

【例 3.5】 假定用于循环顺序存储一个队列的数组的长度为 N，队首和队尾指针分别为 front 和 rear，写出求此队列长度（即所含元素个数）的公式。

解：当 rear 大于或等于 front 时队列长度为 rear－front，也可以表示为（rear－front＋N）％N；当 rear 小于 front 时队列被分成两个部分，前部分在数组尾部，其元素个数为

$N-1-$front,后部分在数组首部,其元素个数为 rear$+1$,两者相加为 rear$-$front$+N$。综上所述,在任何情况下队列长度的计算公式都为(rear$-$front$+N$)$\%N$。

【例 3.6】 在执行操作序列 EnQueue(1)、EnQueue(3)、DeQueue、EnQueue(5)、EnQueue(7)、DeQueue、EnQueue(9)之后队首元素和队尾元素分别是什么? EnQueue(k)表示整数 k 入队,DeQueue 表示队首元素出队。

解:上述操作的执行过程如图 3.11 所示。

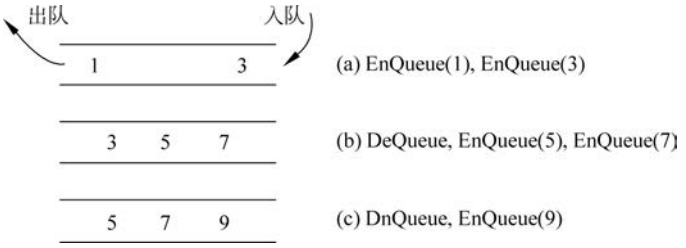

图 3.11 操作的执行过程

所以队首元素为 5,队尾元素为 9。

3.2.4 链队列

链队列用单链表实现,由于入队和出队分别在队列的队尾和队首进行,不存在在队列的任意位置进行插入和删除的情况,所以不需要设置头节点,只需要将指针 front 和 rear 分别指向队首节点和队尾节点,每个节点的指针域指向其后继节点即可。

图 3.12 所示为链队列进行入队操作后的状态变化。

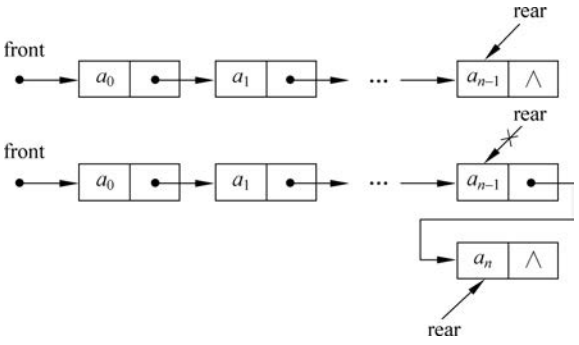

图 3.12 入队操作——链队列

图 3.13 所示为链队列进行出队操作后的状态变化。

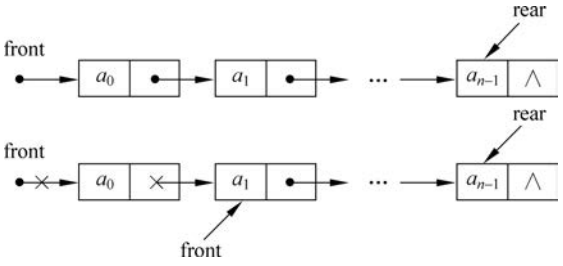

图 3.13 出队操作——链队列

利用 Node 类，链队列的 Python 语言描述如下：

```
class Node(object):
    def __init__(self,data = None,next = None):
        self.data = data
        self.next = next

class LinkQueue(IQueue):
    def __init__(self):
        self.front = None # 队首指针
        self.rear = None # 队尾指针

    def clear(self):
        '''将队列置空'''
        self.front = None
        self.rear = None

    def isEmpty(self):
        '''判断队列是否为空'''
        return self.front is None

    def length(self):
        '''返回队列的数据元素个数'''
        p = self.front
        i = 0
        while p is not None:
            p = p.next
            i += 1
        return i

    def peek(self):
        '''返回队首元素'''
        if self.isEmpty():
            return None
        else:
            return self.front.data

    def offer(self,x):
        '''将数据元素 x 插入队列成为队尾元素'''
        s = Node(x,None)
        if not self.isEmpty():
            self.rear.next = s
        else:
            self.front = s
        self.rear = s

    def poll(self):
        '''将队首元素删除并返回其值'''
        if self.isEmpty():
            return None
```

```
            p = self.front
            self.front = self.front.next
            if p == self.rear: # 删除节点为队尾节点时需要修改 rear
                self.rear = None
            return p.data

        def display(self):
            '''输出队列中的所有元素'''
            p = self.front
            while p is not None:
                print(p.data,end = ' ')
                p = p.next
```

3.2.5 优先级队列

有些应用中的排队等待问题若仅按照“先来先服务”原则不能满足要求,还需要将任务的重要程度作为排队的依据。例如操作系统中的进程调度管理,每个进程都有一个优先级值表示进程的紧急程度,优先级高的进程先执行,同级进程按照先进先出原则排队等待,因此操作系统需要使用优先级队列来管理和调度进程。例如,打印机的输出任务队列,对于先后到达的打印几百页和几页的任务将需要打印的页数较少的任务先完成,这样使得任务的总的等待时间最小。

优先级队列是在普通队列的基础之上将队列中的数据元素按照关键字的值进行有序排列。优先级队列在队首进行删除操作,但为了保证队列的优先级顺序,插入操作不一定在队尾进行,而是按照优先级插入队列的合适位置。

和其他数据结构类似,优先级队列也可以采用顺序和链式两种存储结构。但为了快速地访问优先级高的元素以及快速地插入数据元素,通常使用链式存储结构。

1. 优先级队列节点类的描述

```
class PriorityNode:
    def __init__(self,data = None,priority = None,next = None):
        self.data = data # 节点的数据域
        self.priority = priority # 节点的优先级
        self.next = next
```

2. 优先级队列类的描述及实现

```
class PriorityQueue(IQueue):
    def __init__(self):
        self.front = None # 队首指针
        self.rear = None # 队尾指针

    def clear(self):
        '''将队列置空'''
        self.front = None
        self.rear = None

    def isEmpty(self):
```

```
        '''判断队列是否为空'''
        return self.front is None

    def length(self):
        '''返回队列的数据元素个数'''
        p = self.front
        i = 0
        while p is not None:
            p = p.next
            i += 1
        return i

    def peek(self):
        '''返回队首元素'''
        if self.isEmpty():
            return None
        else:
            return self.front.data

    def offer(self,x,priority):
        '''将数据元素 x 插入队列 priority 位置'''
        s = PriorityNode(x,priority,None)
        if not self.isEmpty():
            p = self.front
            q = self.front
            while p is not None and p.priority <= s.priority:
                q = p
                p = p.next
            # 元素位置的 3 种情况
            if p is None: # 队尾
                self.rear.next = s
                self.rear = s
            elif p == self.front: # 队首
                s.next = self.front
                self.front = s
            else: # 队中
                q.next = s
                s.next = p
        else:
            self.front = self.rear = s

    def poll(self):
        '''将队首元素删除并返回其值'''
        if self.isEmpty():
            return None
        p = self.front
        self.front = self.front.next
        if p == self.rear: # 删除节点为队尾节点时需要修改 rear
            self.rear = None
        return p.data
```

```
62
63      def display(self):
64          '''输出队列中的所有元素'''
65          p = self.front
66          while p is not None:
67              print(p.data,end = ' ')
68              p = p.next
```

注意：在此优先队级列中，数据元素的优先级别依据优先数的大小进行判定，即优先数越小优先级别越大。

3. 优先级队列类的应用

【例 3.7】 利用优先级队列模仿操作系统的进程管理问题，要求优先级高的进程先获得 CPU，优先级相同的进程先到的先获得 CPU。假设操作系统中的进程由进程号和进程优先级两部分组成，规定优先级值越小，优先级越高。

解：

```
pq = PriorityQueue()
pq.offer(1,20)
pq.offer(2,30)
pq.offer(3,10)
pq.offer(4,50)
print("进程服务的顺序为: ")
while not pq.isEmpty():
    print(pq.poll())
```

3.3 栈和队列的比较

栈和队列的比较如表 3.1 所示。

表 3.1 栈和队列的比较

相同点	(1) 均为线性结构，数据元素间具有“一对一”的逻辑关系； (2) 都有顺序存储和链式存储两种实现方式； (3) 操作受限，插入操作均在表尾进行(优先级队列除外)； (4) 插入与删除操作都具有常数时间
不同点	(1) 栈删除操作在表尾进行，具有后进先出特性；队列的删除操作在表头进行，具有先进先出特性； (2) 顺序栈可以实现多栈空间共享，而顺序队列则不同

3.4 实　　验


观看视频

3.4.1 用队列实现栈

使用两个队列实现一个后入先出(LIFO)的栈，并支持栈的 4 种基本操作(push、top、pop 和 empty)。

思路：在使用队列实现栈时，应满足队列前端的元素是最后入栈的元素。用两个队列实现栈的操作，其中一个队列用于存储栈内的元素，另一个队列用于辅助处理入栈操作。

```
class MyStack:
    def __init__(self):
        self.queue1 = SqQueue(50)
        self.queue2 = SqQueue(50)
    def push(self, x: int) -> None:
        self.queue2.offer(x)
        while not self.queue1.isEmpty():
            self.queue2.offer(self.queue1.poll())
        self.queue1, self.queue2 = self.queue2, self.queue1
    def pop(self) -> int:
        return self.queue1.poll()
    def top(self) -> int:
        return self.queue1.queueElem[self.queue1.front]
    def empty(self) -> bool:
        return self.queue1.isEmpty()
```

3.4.2 用栈实现队列

使用两个栈实现先入先出队列。队列应当支持一般队列支持的所有操作(push、pop、peek、empty)。

思路：用到两个栈，其中一个用于反转元素的入队顺序；另一个用于存储元素的最终顺序。

```
class MyQueue:
    def __init__(self):
        self.stack1 = SqStack(50)
        self.stack2 = SqStack(50)
    def move(self):
        while not self.stack1.isEmpty():
            self.stack2.push(self.stack1.pop())
    def push(self,x):
        self.stack1.push(x)
    def pop(self):
        if self.stack2.isEmpty():
            self.move()
        return self.stack2.pop()
    def peek(self):
        if self.stack2.isEmpty():
            self.move()
        return self.stack2.peek()
    def empty(self):
        return self.stack1.isEmpty() and self.stack2.isEmpty()
```

3.4.3 栈的最小值

请设计一个栈，除了常规栈支持的 push 与 pop 函数以外，还支持 min 函数，该函数返回栈元素中的最小值。执行 push、pop 和 min 操作的时间复杂度必须为 $O(1)$。

思路：可以在每个元素入栈时把当前栈的最小值存起来。当栈顶元素是该元素时，当前栈的最小值始终是对应存储的最小值。

```
class min_stack:
    def __init__(self):
        self.stack = SqStack(50)
        self.min_stack = SqStack(50)
        self.min_stack.push(math.inf)
    def push(self,x):
        self.stack.push(x)
        self.min_stack.push(min(x, self.min_stack.peek()))
    def pop(self):
        self.min_stack.pop()
        return self.stack.pop()
    def top(self):
        return self.stack.peek()
    def getMin(self):
        return self.min_stack.peek()
```

小　　结

(1) 栈是一种特殊的线性表，它只允许在栈顶进行插入和删除操作，具有后进先出的特性，各种运算的时间复杂度为 $O(1)$。栈采用顺序存储结构或者链式存储结构。

(2) 队列是一种特殊的线性表，它只允许在表头进行删除操作、在表尾进行插入操作，具有先进先出的特性，各种运算的时间复杂度为 $O(1)$。队列采用顺序存储结构或者链式存储结构。

(3) 循环队列是将顺序队列的首尾相连，解决“假溢出”现象的发生。

(4) 优先级队列是在普通队列的基础之上将队列中的数据元素按照关键字的值进行有序排列。在表头进行删除操作，插入操作按照优先级插入队列的合适位置。

习　题　3

一、选择题

1. 对于栈操作数据的原则是(　　)。

A. 先进先出　　B. 后进先出　　C. 后进后出　　D. 不分顺序

2. 在做入栈运算时应先判别栈是否(　①　)，在做出栈运算时应先判别栈是否(②)。当栈中元素为 n 个，做进栈运算时发生上溢，则说明该栈的最大容量为(　③　)。

①②：A. 空　　B. 满　　C. 上溢　　D. 下溢

③　　A. $n-1$　　B. n　　C. $n+1$　　D. $n/2$

3. 一个栈的输入序列为 1,2,3,…,n，若输出序列的第一个元素是 n，输出的第 i($1\leqslant i\leqslant n$)个元素是(　　)。

A. 不确定　　B. $n-i+1$　　C. i　　D. $n-i$

4. 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 j 个输出元素是(　　)。

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

5. 若已知一个栈的入栈序列是 1,2,3,…,n,其输出序列为 $p_1,p_2,p_3,\cdots,p_n$,若 p_n 是 n,则 p_i 是(　　)。

A. 1　　B. $n-i$　　C. $n-i+1$　　D. 不确定

6. 有 6 个元素 6,5,4,3,2,1 顺序入栈,下列不是合法的出栈序列是(　　)。

A. 5,4,3,6,1,2　　B. 4,5,3,1,2,6

C. 3,4,6,5,2,1　　D. 2,3,4,1,5,6

7. 设栈的输入序列是 1,2,3,4,则(　　)不可能是其出栈序列。

A. 1,2,4,3　　B. 2,1,3,4

C. 1,4,3,2　　D. 4,3,1,2

E. 3,2,1,4

8. 一个栈的输入序列为 1,2,3,4,5,则下列序列中不可能是栈的输出序列的是(　　)。

A. 2,3,4,1,5　　B. 5,4,1,3,2

C. 2,3,1,4,5　　D. 1,5,4,3,2

9. 设一个栈的输入序列是 1,2,3,4,5,则下列序列中是栈的合法输出序列的是(　　)。

A. 5,1,2,3,4　　B. 4,5,1,3,2

C. 4,3,1,2,5　　D. 3,2,1,5,4

10. 某堆栈的输入序列为 a,b,c,d,下面序列中,不可能是它的输出序列的是(　　)。

A. a,c,b,d　　B. b,c,d,a

C. c,d,b,a　　D. d,c,a,b

11. 设 a,b,c,d,e,f 以所给的次序入栈,若在入栈操作时,允许出栈操作,则下面得不到的序列为(　　)。

A. f,e,d,c,b,a　　B. b,c,a,f,e,d

C. d,c,e,f,b,a　　D. c,a,b,d,e,f

12. 设有 3 个元素 X、Y、Z 顺序入栈(入栈的过程中允许出栈),下列得不到的出栈排列是(　　)。

A. X,Y,Z　　B. Y,Z,X

C. Z,X,Y　　D. Z,Y,X

13. 输入序列为 A,B,C,变为 C,B,A 时经过的栈操作为(　　)。

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

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

14. 若一个栈以向量 $\boldsymbol{V}$[1..n]存储,初始栈顶指针 top 为 $n+1$,则下面 x 入栈的正确操作是(　　)。

A.
```
top=top+1
V[top]=x
```

B.
```
V[top]=x
top=top+1
```

C.
```
top=top-1
V[top]=x
```

D.
```
V[top]=x
top=top-1
```

15. 若栈采用顺序存储方式存储,现两栈共享空间 $V[1..m]$,top[i]代表第 i 个栈($i=1,2$)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]+1=top[1]　　B. top[1]+1=top[2]

C. top[1]+top[2]=m　　D. top[1]=top[2]

二、填空题

1. 栈和队列都是_____结构;对于栈只能在_____插入和删除元素;对于队列只能在_____插入和_____删除元素。

2. 栈是一种特殊的线性表,允许插入和删除运算的一端称为_____,不允许插入和删除运算的一端称为_____。

3. _____是被限定为只能在表的一端进行插入运算、在表的另一端进行删除运算的线性表。

4. 在一个循环队列中,队首指针指向队尾元素的_____位置。

5. 在具有 n 个单元的循环队列中,队满时共有_____个元素。

6. 向栈中压入元素的操作是先_____,后_____。

7. 从循环队列中删除一个元素,其操作是先_____,后_____。

8. 顺序栈用 data[1..n]存储数据,栈顶指针是 top,top 初始值为0,则值为 x 的元素入栈的操作是_____。

9. 引入循环队列的目的是克服_____。

三、算法设计题

1. 把十进制整数转换为二至九进制数并输出。

2. 堆栈在计算机语言的编译过程中用来进行语法检查,试编写一个算法检查一个字符串中的花括号、方括号和圆括号是否配对,若能够全部配对则返回逻辑“真”,否则返回逻辑“假”。

3. 斐波那契(Fibonacci)数列的定义为它的第1项和第2项分别为0和1,以后各项为其前两项之和。若斐波那契数列中的第 n 项用 Fib(n)表示,则计算公式如下:

$$\mathrm{Fib}(n)=n-1(n=1\text{ 或 }2)$$

$$\mathrm{Fib}(n)=\mathrm{Fib}(n-1)+\mathrm{Fib}(n-2)(n>2)$$

试编写出计算 Fib(n)的递归算法和非递归算法。