

数据结构课程设计

北京邮电大学



姓 名 tobeApe6eHok

学 院 人工智能学院

专 业 智能科学与技术

班 级 1111111111

学 号 1111111111

班内序号 1

指导教师 周延泉

2022 年 4 月

目录

一、实验题目3

二、题目分析与算法设计3

三、数据结构描述4

四、程序清单5

五、程序复杂度分析7

六、程序运行结果7

数据结构课程设计实验三

一、实验题目

若希望循环队列中的元素都能得到利用，则需设置一个标志域 tag，并以 tag 的值为 0 或 1 来区别队头指针 front 和队尾指针 rear 相同时的队列状态是“空”还是满，试编写与此结构相应的入队与出队算法。

二、题目分析与算法设计

1、设计思路：

(1) 循环队列之所以能够消除顺序队列中的“假溢出”现象，是因为在循环队列中不断的对元素下标进行取余，所以在队列的 Front 和 Rear 指针加一的时候，要进行取模运算。

(2) 其次要定义 Rear 指向的元素的意义，有两种意义：

A、Rear 指向的元素是队尾元素

B、Rear 指向的位置是队尾元素的下一个位置，还未被利用

如果使用第一种意义，因为 Front 和 Rear 指针在除了没有元素的情况外，不会有相同的时候，所以没必要设置 Tag 标志，在队列非空情况下只需要在元素入队时判断 Rear 的下一个位置是否是 Front 即可，如果是，则已满，如果不是，则可入队；而如果采用第二种意义，则需要考虑 Tag 标志。根据题意，必须要采用第二种意义。

(3) 本次实验中我选用的是 python 语言，由于 python 语言是动态语言，数据的类型不受约束，所以支持泛型编程。

(4) 为了更好的表现算法的效果，我设计了一个循环队列可视化的函数，可以将整个循环队列所在的数组状态打印出来。

2、算法描述：

(1) 入队算法：

在入队时，要首先判断队列是否已满，即判断 Tag 表示是否为 True。如果已满，直接返回入队失败即可；如果未满，由于 Rear 指向的是第一个未被利用的位置，所以只需要将 Rear 下标的元素赋值即可，其后，要将 Rear 指向下一个位置，并且要检查入队后队列是否已满，

标志就是 $\text{Front} == \text{Rear}$ 。

(2) 出队算法:

在出队时,需要判断队列中是否有元素,队列为空时 $\text{Front} == \text{Rear}$,但是队列满时也满足,所以还要加上 $\text{Tag} == \text{False}$ 的条件。如果队列为空,则返回队列为空即可;如果队列中有元素,则直接将 Front 往下一个位置移动即可,并且此处要将 Tag 设置为 False ,因为任何情况下,如果队列出队,那么队列必定会变成未满情况。

三、数据结构描述

1、数据结构的选择:

面向对象编程会使得程序更加简洁,所以我将循环队列设置成了一个 `Deque` 类:

类中的变量和函数	作用
<code>Self.Element</code>	循环队列所在的数组
<code>Self.Front</code>	队列的队首元素指针
<code>Self.Rear</code>	队尾元素的后一个位置的指针
<code>Self.Tag</code>	标志 $\text{Front} == \text{Rear}$ 时,队列是否已满的标志域
<code>__init__(self, DequeSize)</code>	初始化 <code>Element</code> 为 <code>DequeSize</code> 大小的数组
<code>DequeRepr(self)</code>	将循环队列形式化为 <code>str</code> 类型,可以将循环队列形象的打印出来
<code>Push(self, Value)</code>	将 <code>Value</code> 作为元素入队, <code>Value</code> 可以是任何类型
<code>Pop(self)</code>	将队首元素出队

2、程序中数据结构的定义:

```

class Deque:
    def __init__(self, DequeSize:int)->None:
        self.Element=[None]*DequeSize
        self.Front, self.Rear=0,0
        self.Tag=False
        return

    def DequeRepr(self)->str:
        pass

    def Push(self, Value:any)->bool:
        pass

    def Pop(self)->bool:
        pass

```

四、程序清单

我的实验环境是：WIN10 + VS code + python 3.10.0，以下是我调试正确的代码：

```

class Deque:

    def __init__(self, DequeSize:int)->None:
        """初始化循环队列, 队列中最多含有 DequeSize 个元素 (DequeSize>0)"""
        self.Element=[None]*DequeSize          # 初始化循环队列的元素列表
        self.Front, self.Rear=0,0              # 初始化队首、队尾的位置
        self.Tag=False                          # 将队列满的标志设置为 False, 代表不满
        return

    def DequeRepr(self)->str:
        """
        `为了形式化展示队列的状态, 可以形象化队列`
        `将队列转换成 str 类型, 直接打印返回的 str 即可`
        """

    def Between(num:int)->bool:
        """判断一个下标 num 是否在 self.Front 和 self.Rear 之间"""
        # 如果 Front==Rear 只需要判断队列是否已满
        # 如果未滿, 则没有元素, 返回 False
        # 如果已滿, 则 num 在 Front 和 Rear 之间
        if self.Front==self.Rear:
            return self.Tag
        # num 在 Front 和 Rear 之间的情况只有如下三种
        return num<self.Rear<self.Front or self.Rear<self.Front<=num or
self.Front<=num<self.Rear
        # 将 Front 和 Rear 之间的元素使用类型内置的 __repr__ 函数转成字符串
        # 将不在 Front 和 Rear 之间的元素的 str 设置为 'None'
        EleStr=[type(self.Element[E]).__repr__(self.Element[E]) if Between(E) else
'None' for E in range(len(self.Element))]

```

```

# 设置队列中每个元素所占的空间
Length=max([len(i) for i in EleStr])+6
# 将所有元素安排在一行上
LS='|'+'|'.join([i.center(Length,' ') for i in EleStr])+'|'
# 将 Front 和 Rear 指针打印出来
LEFTS=' '*((1+Length)*self.Front+Length//2-1)+'Front'+'\n'+
'*((1+Length)*self.Front+Length//2-1)+' ↓ →→→→'
RIGHTS=' '*((1+Length)*self.Rear+Length//2-1)+'→↑'+'\n'+
'*((1+Length)*self.Rear+Length//2-1)+'Rear'
# 设置上下行
GAP='- '*len(LS)
FIRST='\n 整个数组的状态如下:\n\n'+LEFTS+'\n'+GAP+'\n'+LS+'\n'+GAP+'\n'+RIGHTS
return FIRST

def Push(self,Value:any)->bool:
    """元素入队"""
    # 如果已满,则无法入队
    if self.Tag:
        return False
    # 将 Rear 设置为 Value
    self.Element[self.Rear]=Value
    # 将 Rear 指向下一个位置
    self.Rear=(self.Rear+1)%len(self.Element)
    # 如果 Front==Rear 且此时队列必定非空,那么队列必定已满
    self.Tag=True if self.Front==self.Rear else False
    return True

def Pop(self)->bool:
    """元素出队"""
    # 判断队列是否为空
    if not self.Tag and self.Front==self.Rear:
        return False
    # 如果非空,则直接将队首设置成下一个元素即可
    self.Front=(self.Front+1)%len(self.Element)
    # 出队后,队列必定不满
    self.Tag=False
    return True

# 测试函数
def main()->None:
    """测试函数"""
    IScontinue=1
    TestDeque=Deque(int(input('请输入要创建的循环队列的大小:')))
    print(TestDeque.DequeRepr())
    MENU='\n 可进行如下操作:\n1、入队\n2、出队\n 请输入要进行的操作序号:'
    while IScontinue==1:
        cho=int(input(MENU))
        if cho==1:
            result=TestDeque.Push(int(input('请输入要入队的元素:')))
            if not result:
                print("\033[%dm\033[1;%dm%s\033[0m"%(41,37,'队列已满,入队失败.'))
            else:
                print("\033[%dm\033[1;%dm%s\033[0m"%(42,37,'入队成功.'))
        elif cho==2:
            result=TestDeque.Pop()
            if not result:
                print("\033[%dm\033[1;%dm%s\033[0m"%(41,37,'队列中没有元素,不可以出队.'))
            else:
                print("\033[%dm\033[1;%dm%s\033[0m"%(42,37,'出队成功.'))
        else: continue

```

```
print(TestDeque.DequeRepr())
IScontinue=int(input('请输入是否继续: 1 or 0\n'))
if __name__=='__main__':
    main()
```

五、程序复杂度分析

1、时间复杂度:

(1) Push(): 只需要将元素放到队尾, Rear 加一, 判断队列是否已满即可, 时间复杂度是 $O(1)$ 。

(2) Pop(): 只需要将 Front 加一, 改变 Tag 为 False 即可, 时间复杂度是 $O(1)$ 。

2、空间复杂度:

整个过程中, 只需要开辟模拟循环队列的数组和 4 个变量的空间即可, 不需要额外的空间。设队列的容量是 n , 所以空间复杂度是 $O(n)$ 。

六、程序运行结果

我的实验环境是 WIN10 + VS code + python 3.10.0, 我的测试样例和正确的测试结果如下:

- 1、创建队列大小是 5
- 2、分别将 1000、1001、1002、1003、1004 入队
- 3、再将 1005 入队 (提示 “队列已满, 入队失败.”)
- 4、队首出队
- 5、队首出队
- 6、1005 入队
- 7、将所有的元素全部出队
- 8、再次出队 (提示 “队列中没有元素, 不可以出队.”)
- 9、1006 入队, 结束。

```
E:\VSCodework>python -u "e:\VSCodework\pythonwork\GeneAlgorithm.py"
请输入要创建的循环队列的大小:5
```

整个数组的状态如下:

```
Front
↓ →→→→→
-----
|  None  |  None  |  None  |  None  |  None  |
-----
→→↑
Rear
```

可进行如下操作:

1、入队

2、出队

请输入要进行的操作序号:[]

请输入要进行的操作序号:1

请输入要入队的元素:1000

入队成功.

整个数组的状态如下:

```
Front
↓ →→→→→
-----
| 1000  |  None  |  None  |  None  |  None  |
-----
          →→↑
          Rear
```

请输入是否继续: 1 or 0

1

可进行如下操作:

1、入队

2、出队

请输入要进行的操作序号:1

请输入要入队的元素:1001

入队成功.

整个数组的状态如下:

```
Front
↓ →→→→→
-----
| 1000  | 1001  |  None  |  None  |  None  |
-----
                →→↑
                Rear
```

请输入是否继续: 1 or 0

1

可进行如下操作:

1、入队

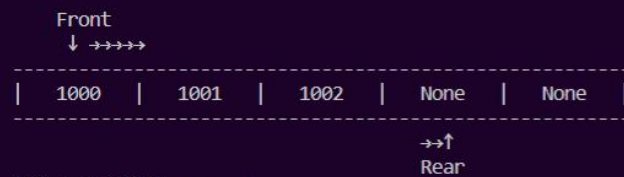
2、出队

请输入要进行的操作序号:1

请输入要入队的元素:1002

入队成功

整个数组的状态如下:



请输入是否继续: 1 or 0

1

可进行如下操作:

1、入队

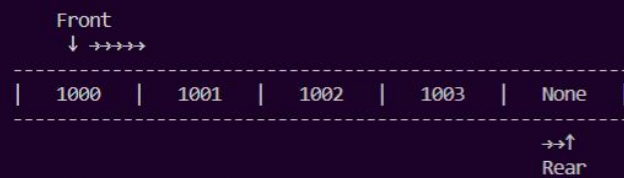
2、出队

请输入要进行的操作序号:1

请输入要入队的元素:1003

入队成功

整个数组的状态如下:



请输入是否继续: 1 or 0

1

可进行如下操作:

1、入队

2、出队

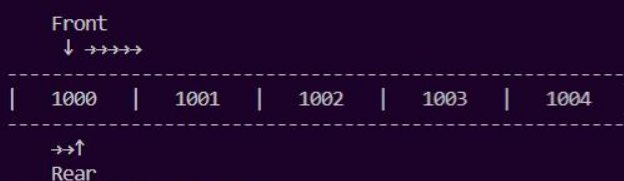
请输入要进行的操作序号:1

请输入要进行的操作序号:1

请输入要入队的元素:1004

入队成功

整个数组的状态如下:



请输入是否继续: 1 or 0

1

可进行如下操作:

1、入队

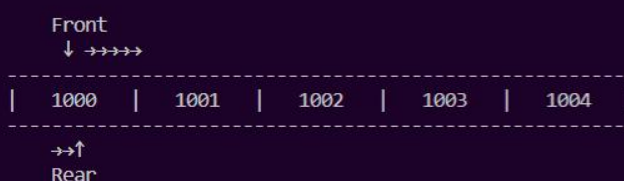
2、出队

请输入要进行的操作序号:1

请输入要入队的元素:1005

队列已满,入队失败.

整个数组的状态如下:



请输入是否继续: 1 or 0

0

请输入是否继续: 1 or 0
1

可进行如下操作:

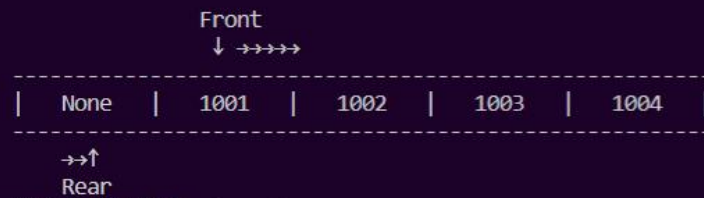
1、入队

2、出队

请输入要进行的操作序号:2

出队成功.

整个数组的状态如下:



请输入是否继续: 1 or 0
1

可进行如下操作:

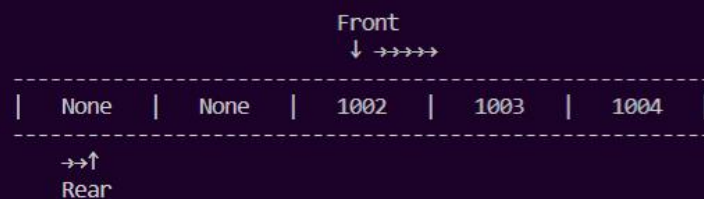
1、入队

2、出队

请输入要进行的操作序号:2

出队成功.

整个数组的状态如下:



请输入是否继续: 1 or 0
0

请输入是否继续: 1 or 0
1

可进行如下操作:

1、入队

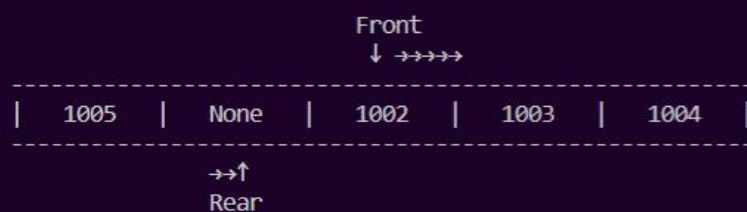
2、出队

请输入要进行的操作序号:1

请输入要入队的元素:1005

入队成功.

整个数组的状态如下:



请输入是否继续: 1 or 0
1

```

请输入是否继续: 1 or 0
1

```

可进行如下操作：

1、入队

2、出队

请输入要进行的操作序号:2

出队成功.

整个数组的状态如下：

Front
↓ →→→→→

1005	None	None	1003	1004
------	------	------	------	------

→ → ↑

Rear

请输入是否继续: 1 or 0

1

可进行如下操作：

1、入队

2、出队

请输入要进行的操作序号:2

出队成功.

整个数组的状态如下：

Front
↓ →→→→→

1005	None	None	None	1004
------	------	------	------	------

→→↑

Rear

请输入是否继续: 1 or 0

1

可进行如下操作：

1、入队

2、出队

请输入要进行的操作序号:2

出队成功.

整个数组的状态如下：

Front
↓ →→→→→

1005	None	None	None	None
------	------	------	------	------

→→↑

Rear

请输入是否继续: 1 or 0

1

可进行如下操作：

1、入队

2、出队

请输入要进行的操作序号:2

出队成功.

整个数组的状态如下：

Front
↓ →→→→→

None	None	None	None	None
------	------	------	------	------

→→↑

Rear

请输入是否继续: 1 or 0

1

rear
请输入是否继续: 1 or 0
1

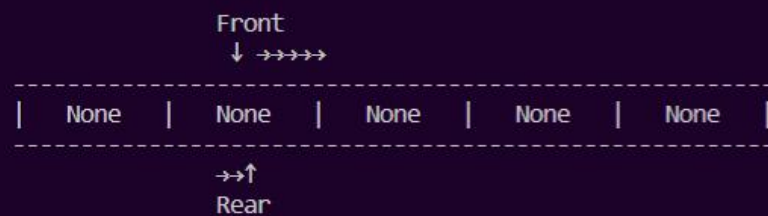
可进行如下操作:

- 1、入队
- 2、出队

请输入要进行的操作序号:2

队列中没有元素,不可以出队.

整个数组的状态如下:



请输入是否继续: 1 or 0
1

可进行如下操作:

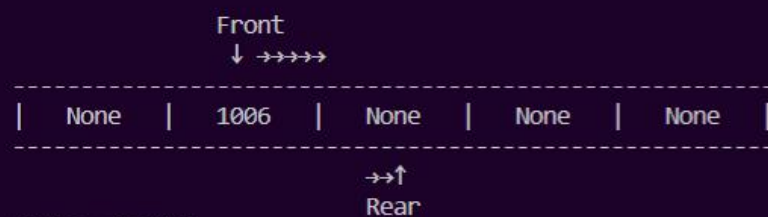
- 1、入队
- 2、出队

请输入要进行的操作序号:1

请输入要入队的元素:1006

入队成功.

整个数组的状态如下:



请输入是否继续: 1 or 0
0