深 圳 大 学 实 验 报 告

课程名称：智能无人系统与边缘计算

实验项目名称：基于 Python 实现循环队列

学院：电子与信息工程学院

专业：22 电子信息工程

指导教师：蒙山

报告人：陈应权 学号：2022280297 班级：文华班

实验时间：2024 年 9 月 20 日——2024 年 9 月 24 日

实验报告提交时间：2024 年 9 月 24 日

教务部制

实验目的与要求：

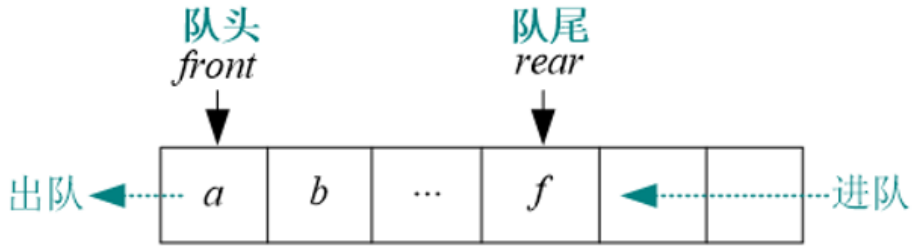
1. 图像读取和存储：编程实现从 minst_images 文件中读取不少于 1000 幅图像样本，每个样本数据存储在题述类的对象里。
2. 循环队列存储图像对象：设计一个循环队列存储题述类的对象，以 4Hz 频率向队列输入样本数据对应的类对象实例，队列需能够缓存不少于 5 秒时间的数据。
3. 图像显示：以 2Hz 的频率从队列中取出图像，并通过 OpenCV 的固定窗口顺序地显示每一幅图像。
4. 动态调节读取频率：当队列出现上溢（队列满）时，将读取频率提高到 8Hz；当队列下溢（队列空）时，读取频率降低回 2Hz，并相应地调整显示频率，如此周而复始。

实验内容：

一、队列与循环列表

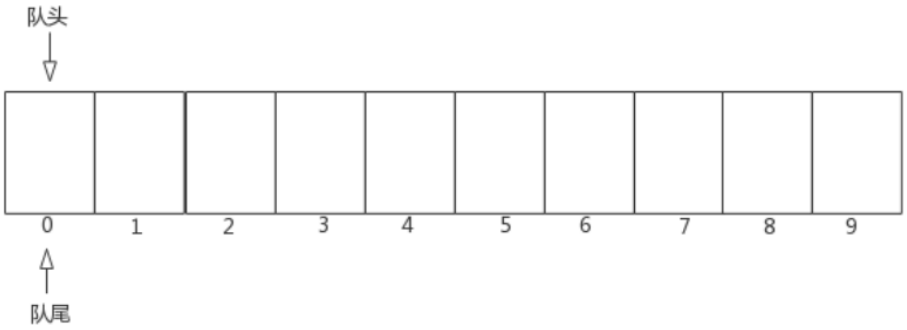
1.1 队列与循环队列

先进先出的线性序列，称为队列，队列也是一种线性表，只不过它是操作受限的线性表，只能在两端操作。一端进，一端出。进的一端称为队尾，出的一端称为队头，队列可以用顺序存储也可以用链式存储。队列的顺序存储形式，可以用一段连续的空间存储数据元素，用两个整型变量记录队头和队尾元素的下标。



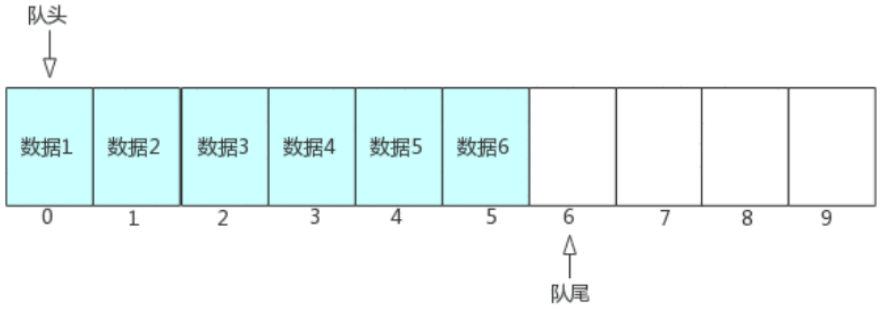
队列初始化

front,rear 分别代表指向队头和队尾的"指针"(数组下标),构造空队列只需要按照所需队列长度申请一块内存给基地址,并且将队头指针与队尾指针赋值为 0.



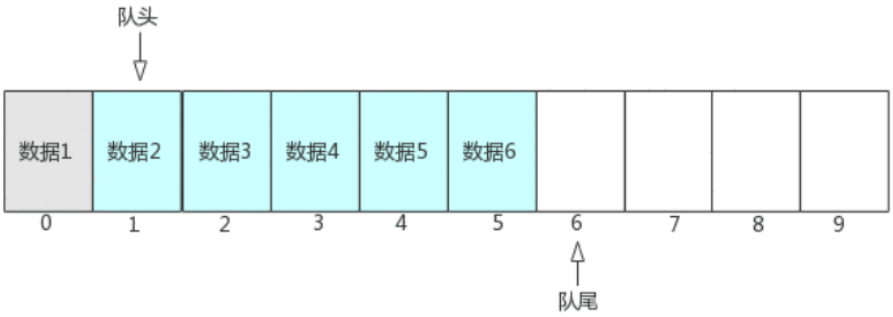
入队

将队尾指向的位置赋值,并且赋值后将队尾的位置后移

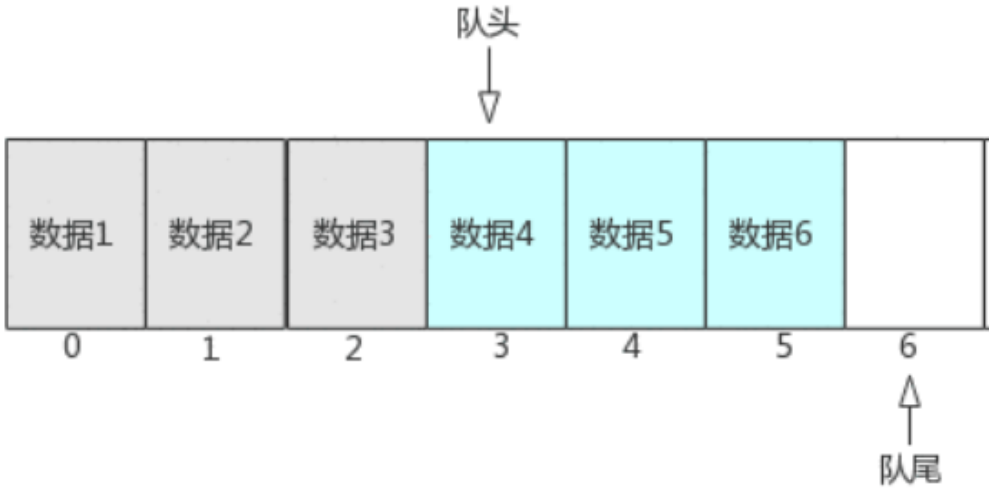


出队

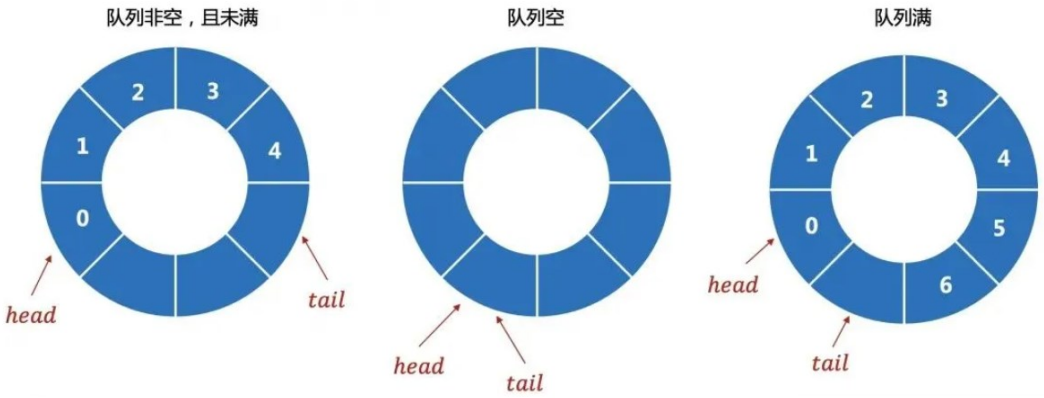
与入队类似，从队头指向的位置取值,取值后将队头的位置后移



普通的队列入队时会出现如下情况：



可以看出，此时 rear 已经指向末端，即队列已满，无法继续增加新数据，但申请的内存里，而队头前的数据早已读出，可以覆盖掉，为了利用存储空间，引入了**循环队列结构**



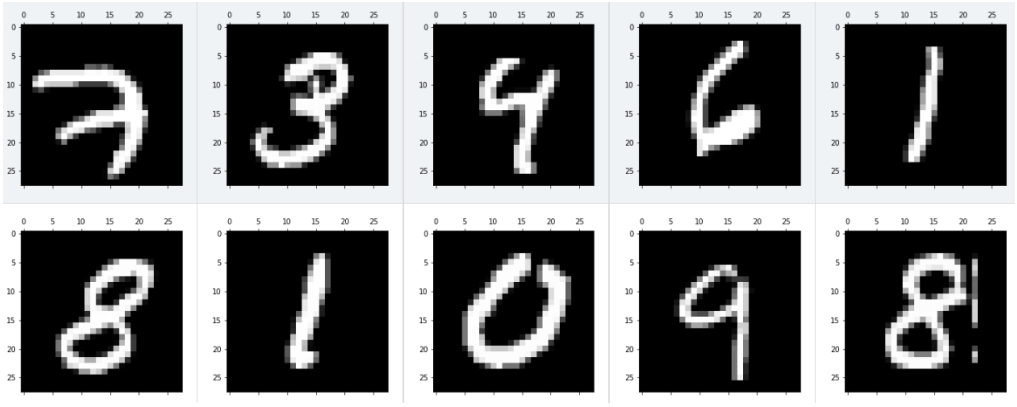
注意，循环队列的最后一位为预留位，是为了作为结束位，所以在创建循环队列申请内存时，创建长度应该为所用长度+1。

二、Minst 数据集

2.1Minst 数据集简介

MNIST 数据集来自美国国家标准与技术研究所，National Institute of Standards and Technology (NIST)。训练集（training set）由来自 250 个不同人手写的数字构成，其中 50%是高中学生，50%来自人口普查局（the Census Bureau）的工作人员。测试集（test set）也是同样比例的手写数字数据，但保证了测试集和训练集的作者集不相交。

MNIST 数据集一共有 7 万张图片，其中 6 万张是训练集，1 万张是测试集。每张图片是 28 × 28 的 0~9 的手写数字图片组成。每个图片是黑底白字的形式，黑底用 0 表示，白字用 0-1 之间的浮点数表示，越接近 1，颜色越白。



Minst 手写数据集是以字节的形式进行存储，Minst 的图像数据包前 16 个字节为 4 个整型 int 数据，涵义分别为，Magic 帧，图像数量，行数，列数，其后的数据全为图像像素数据。即在前 16 个字节后，连续的数据都是 28*28=784 为一幅图像的数据。

三、python 的 time 模块

time 库是 Python 中内置的处理时间的标准库，是最基础的时间处理库。
示例代码如下

```
import time
a=time.time()
time.sleep(1)
b=time.time()
print(b-a)
```

代码输出

```
1.011791706085205
```

time.time()会返回从世界标准时间的 1970 年 1 月 1 日 00：00：00 开始到当前这一时刻为止的总秒数。

time.sleep()为延时函数，单位是秒，在实验中发现延时精度误差较大，并于 cpu 性能正相关，使用时应多注意。

实验过程、结果及结论：

1. 实验过程：

(1) 读取 MNIST 图像：

从 MNIST 数据集文件中读取图像数据，并将其存储为 MNISTImage 类的对象。每个图像样本包含 784 个像素点，以 28×28 的二维数组存储。

(2) 数据写入队列的线程：

通过 input_image ()函数实现每秒 4 帧的图像样本写入操作。该函数使用线程来模拟数据的按频率写入。每插入一个图像对象后，线程会休眠 0.25 秒（4Hz），以保持一定的插入节奏。

(3) 从队列读取数据并显示图像：

读取图像数据并调整读取频率的关键。通过检查队列的状态，动态调整读取频率（初始为 2Hz，满时变为 8Hz，空时回到 2Hz），并相应地改变显示频率。当读取频率变化时，程序会输出提示信息，告知频率的调整情况。

(4) 图像显示功能：

在主线程中，实时从显示队列中取出图像并展示。每次展示图像都会在一个固定的窗口中依次展示队列内存在的图像。

(5) 线程并发处理：

分别创建了两个线程，一个负责向队列写入数据，另一个负责从队列读取数据并显示图像。并发线程处理实现了数据的实时写入、读取和显示功能。

(6) 频率动态调整：
当队列满时，读取频率从 2Hz 增加至 8Hz，以加速图像的读取和显示；
当队列空时，读取频率回到 2Hz，确保程序不会因读取过快而无图像可显示。
(7) 队列情况展示：
通过按照 队伍内现有图像个数/队伍最大容量长度 的显示方法来在每一轮读取图片后进行实时展示，方便用户感知队伍的情况。
(8) 多线程处理:
为了方便处理，将其划分为两个现程，输入线程和展示线程，其中由于 opencv 的展示必须是主线程，于是将展示线程设为主线程。此外，在 ImageDisplay 类中启动了一个名为 input_thread 的线程，用于从数据集中读取图像并将其放入循环队列中。这个线程和主线程并行运行，从而实现图像的输入与显示同时进行。

下表为实验中用到的重要参数：

表 1：实验用到的重要参数

描述	数值
MNIST 图像的宽度	28
MNIST 图像的高度	28
图像的总像素数量（宽度×高度）	784
从文件中读取的图像数量	1000
循环队列的容量	40
图像写入队列的初始频率	4 Hz
图像从队列读取的初始频率	2 Hz
队列满时，读取频率	8 Hz
队列空时，读取频率	2 Hz

2. 实验结果：

(1) 文本输出结果：以下截取了 3 个情况的转化的部分截取。
1) 初始显示
Magic Number: 2051, Number of Images: 10000, Image Size: 28x28
Current queue size: 0/40
Current queue size: 1/40
Current queue size: 2/40
Current queue size: 3/40
Current queue size: 4/40
Current queue size: 5/40
Current queue size: 6/40
其中第一句为验证信息，为了验证读取函数没有问题。

2) 队伍满溢后，输出频率由 2Hz 转化为 8Hz。

Current queue size: 38/40
Queue is full, increasing read frequency to 8Hz.
Current queue size: 39/40
Current queue size: 38/40
Current queue size: 38/40
Current queue size: 37/40
Current queue size: 37/40
Current queue size: 37/40
Current queue size: 36/40
Current queue size: 36/40
Current queue size: 35/40
Current queue size: 35/40
Current queue size: 34/40
Current queue size: 34/40
Current queue size: 33/40

3) 队伍内清零后，输出频率由 8Hz 变回 2Hz。

Current queue size: 0/40
Queue is empty, decreasing read frequency to 2Hz.
Current queue size: 0/40
Current queue size: 1/40
Current queue size: 2/40
Current queue size: 3/40
Current queue size: 4/40
Current queue size: 5/40

(2) 截屏结果：

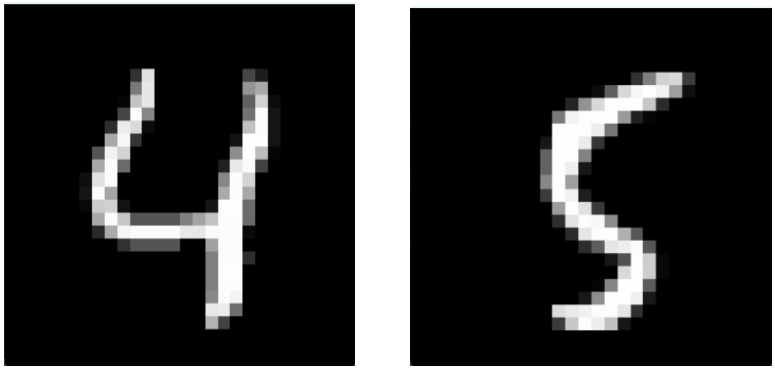


图 1：图像显示的部分结果

3. 实验结论：

本次实验成功设计了一个能够存储 MNIST 数据集图像信息的类，并实现了循环队列和动态调节读取频率的功能。主要收获和结论如下：

实验结论：

1) **循环队列实现效果：** 通过本次实验，我们成功实现了一个循环队列来管理图像数据的缓存。循环队列能够有效地处理图像读取和显示过程中的并发问题，保证了数据的连续流动。
2) **图像读取和存储：** 实验中，我们从 MNIST 数据集文件中读取了 1000 幅图像样本，并将它们存储在 MNISTImage 类的对象中。每个样本数据都成功地以 28×28 的二维数组形式存储，满足了实验要求。
3) **队列性能：** 循环队列能够以 4Hz 的频率向队列输入图像样本数据，并能缓存不少于 5 秒即 40 幅图像的数据量。这表明队列具有足够的容量来应对数据的流入。
4) **动态频率调整：** 实验中观察到，当队列满时，读取频率能够自动提高到 8Hz，而当队列空时，读取频率能够降低回 2Hz。这种动态调整机制有效地平衡了图像的读取和显示过程，避免了队列的上溢和下溢。
5) **图像显示：** 图像以 2Hz 的频率从队列中取出并显示，满足了实验要求。通过 OpenCV 创建的固定窗口，我们能够顺序地展示每一幅图像，图像显示效果良好。
6) **线程并发处理：** 实验中创建的两个线程——一个负责向队列写入数据，另一个负责从队列读取数据并显示图像——能够并行运行，实现了数据的实时写入、读取和显示。
7) **实验改进：** 尽管实验达到了预期目标，但在实验过程中也发现了一些可以改进的地方。例如，线程同步机制可以进一步优化，以减少队列满和空时的等待时间。此外，图像显示的延迟可以进一步降低，以提供更流畅的视觉效果。
8) **总结：** 综上所述，本次实验成功地实现了循环队列的设计与应用，展示了其在图像处理中的有效性。通过动态调整读取频率，我们能够高效地管理图像数据流，确保了图像显示的连续性和稳定性。这不仅验证了循环队列理论在实际应用中的有效性，也为我们提供了多线程编程和图像处理的实践经验。

实验代码：

```python
import numpy as np
import struct
import time
import threading
import cv2
from collections import deque


# 图像类，用于存储单个图像的索引和数据
class MNISTImage:
```

```
        def __init__(self, index, data):
            self.index = index  # 图像索引
            self.data = data    # 图像数据


    # 数据集类，用于加载MNIST图像数据集
    class MNISTDataset:
        def __init__(self, file_path):
            self.images = []  # 存储加载的图像
            self.load_images(file_path)  # 调用加载函数

        def load_images(self, file_path):
            with open(file_path, 'rb') as f:  # 以二进制模式打开文件
                magic, num_images, num_rows, num_cols = struct.unpack('>IIII', f.read(16))
                # 读取文件头信息
                print(f"Magic Number: {magic}, Number of Images: {num_images}, Image Size: {num_rows}x{num_cols}")

                for index in range(1000):  # 读取最多1000幅图像
                    img_data = f.read(28 * 28)  # 读取28x28的图像数据
                    if not img_data:
                        break  # 如果没有数据，则退出
                    image_array = np.frombuffer(img_data, dtype=np.uint8)  # 将数据转换为数组
                    self.images.append(MNISTImage(index, image_array))  # 添加到图像列表


    # 循环队列类，用于管理图像的缓存
    class CircularQueue:
        def __init__(self, max_size):
            self.queue = deque(maxlen=max_size)  # 创建一个固定大小的双端队列
            self.lock = threading.Lock()  # 创建一个锁以确保线程安全

        def enqueue(self, item):
            with self.lock:  # 加锁以确保线程安全
                self.queue.append(item)  # 将项添加到队列

        def dequeue(self):
            with self.lock:  # 加锁以确保线程安全
                if self.queue:
```

```
                return self.queue.popleft()  # 移除并返回队列头部的项
            return None

    def is_full(self):
        return len(self.queue) == self.queue.maxlen  # 检查队列是否满

    def is_empty(self):
        return len(self.queue) == 0  # 检查队列是否空

    def size(self):
        return len(self.queue)  # 返回当前队列大小

    def maxlen(self):
        return self.queue.maxlen  # 返回队列的最大长度


# 输入输出管理类，用于处理图像的输入和显示
class ImageDisplay:
    def __init__(self, queue, mnist_dataset):
        self.queue = queue  # 保存队列
        self.read_frequency = 2  # 初始读取频率为2Hz
        self.mnist_dataset = mnist_dataset  # 保存数据集
        self.input_thread = threading.Thread(target=self.input_images)  # 创建输入线程
        self.input_thread.start()  # 启动输入线程

    def input_images(self):
        for image in self.mnist_dataset.images:
            while self.queue.is_full():
                time.sleep(0.1)  # 等待直到队列有空间
            self.queue.enqueue(image)  # 将图像放入队列
            time.sleep(0.25)  # 设置输入频率为4Hz

    def display_images(self):
        cv2.namedWindow('MNIST Images', cv2.WINDOW_NORMAL)  # 创建一个窗口
        while True:
            if not self.queue.is_empty():
                image = self.queue.dequeue()  # 从队列中取出图像
                img = image.data.reshape(28, 28)  # 将数据转换为图像格式
                cv2.imshow('MNIST Images', img)  # 显示图像
```

```
88
89                      # 显示队列状态
90                      print(f"Current queue size: {self.queue.size()
    }/{self.queue.maxlen()}")
91
92                      time.sleep(1 / self.read_frequency)  # 控制显示
    频率
93
94                      # 等待键盘输入，确保窗口更新
95                      if cv2.waitKey(1) & 0xFF == ord('q'):
96                          break  # 按 'q' 键退出
97
98                  # 根据队列状态调整读取频率
99                  if self.queue.is_empty() and self.read_frequency =
    = 8:
100                     self.read_frequency = 2  # 降低回 2Hz
101                     print("Queue is empty, decreasing read frequen
    cy to 2Hz.")
102                 elif self.queue.is_full() and self.read_frequency
    == 2:
103                     self.read_frequency = 8  # 提升到 8Hz
104                     print("Queue is full, increasing read frequenc
    y to 8Hz.")
105
106
107 # 主程序
108 if __name__ == "__main__":
109     # 加载数据集，替换为实际的 MNIST 文件路径
110     mnist_dataset = MNISTDataset("E:\Grade3\智能\实验三 基于
    Python 实现循环队列(1)\实验三 基于 Python 实现循环队列
    \minst_images")
111     queue = CircularQueue(max_size=40)  # 创建一个最大大小为40 的
    循环队列
112     display_manager = ImageDisplay(queue, mnist_dataset)  # 创
    建显示管理对象
113
114     # 在主线程中调用显示函数
115     display_manager.display_images()
116
117     # 释放 OpenCV 窗口
118     cv2.destroyAllWindows()
```

指导教师批阅意见： 成绩评定： 指导教师签字： 年　　月　　日
备注：

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。