CSI33 Data Structures

Sharon Persinger

Fall 2019

Day 10 October 2

Queues

- First in, first out data structure - FIFO
- Items are added at the end - enqueue
- Items are removed from the front or head – dequeue
- Familiar situation – bus line, checkout line, ticket line …

Queue ADT – implemented using Python list

```
#-------------------------------------------------------------
class Queue:
    #--------------------------------------------------------
    def __init__(self):
        '''create an empty FIFO queue'''
    #--------------------------------------------------------
    def size(self):
        '''return number of items in the queue
         post: returns number of items in the queue'‘’
#-----------------------------------------------------------------
    def enqueue(self, x):
        '''insert x at end of queue
          post: x is added to the queue'''
```

```
#-------------------------------------------------------
    def front(self):
        '''return first item in queue
        pre: queue is not empty; IndexError is
         raised if empty
        post: returns first item in the queue'''
    #---------------------------------------------------
    def dequeue(self):
        '''remove and return first item in queue
        pre: queue is not empty; IndexError is
         raised if empty
        post: removes and returns first item in the
         queue'''
#-------------------------------------------------------
```

Analysis of queue implementation using Python list

- enqueue using insert at position 0
  - Recopy the entire queue with every insertion - so $\Theta(n)$ where n is the number of elements in the queue
- dequeue using pop
  - Constant time
- What if we decided to enqueue at the end using append and dequeue by deleting at the beginning position 0?
  - Still have the recopying issue, now to move very element up each time there is a dequeue
- Easy to implement

Applications

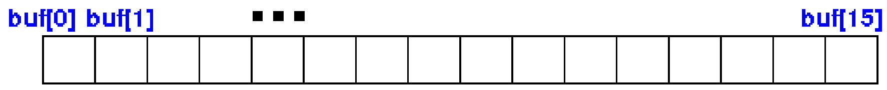
- Operating systems - Manage shared resources, such as a printer
- Determine whether a string is a palindrome
  - Queue to read the string forward
  - Stack to read the string backward
  - Module palindrome.py

Analysis of queue implementation using Linked List

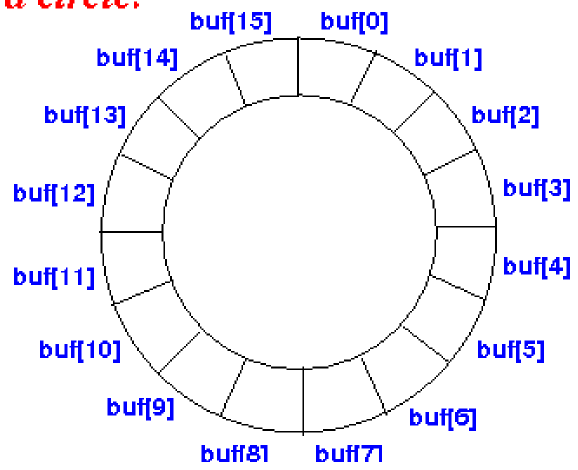
- enqueue using insert at tail or append
  - Keep a tail reference so this is constant time
- Dequeue using remove at head
  - Again constant time
- Why do it this way instead of dequeue at the tail?
- Need a Linked List with tail implementation

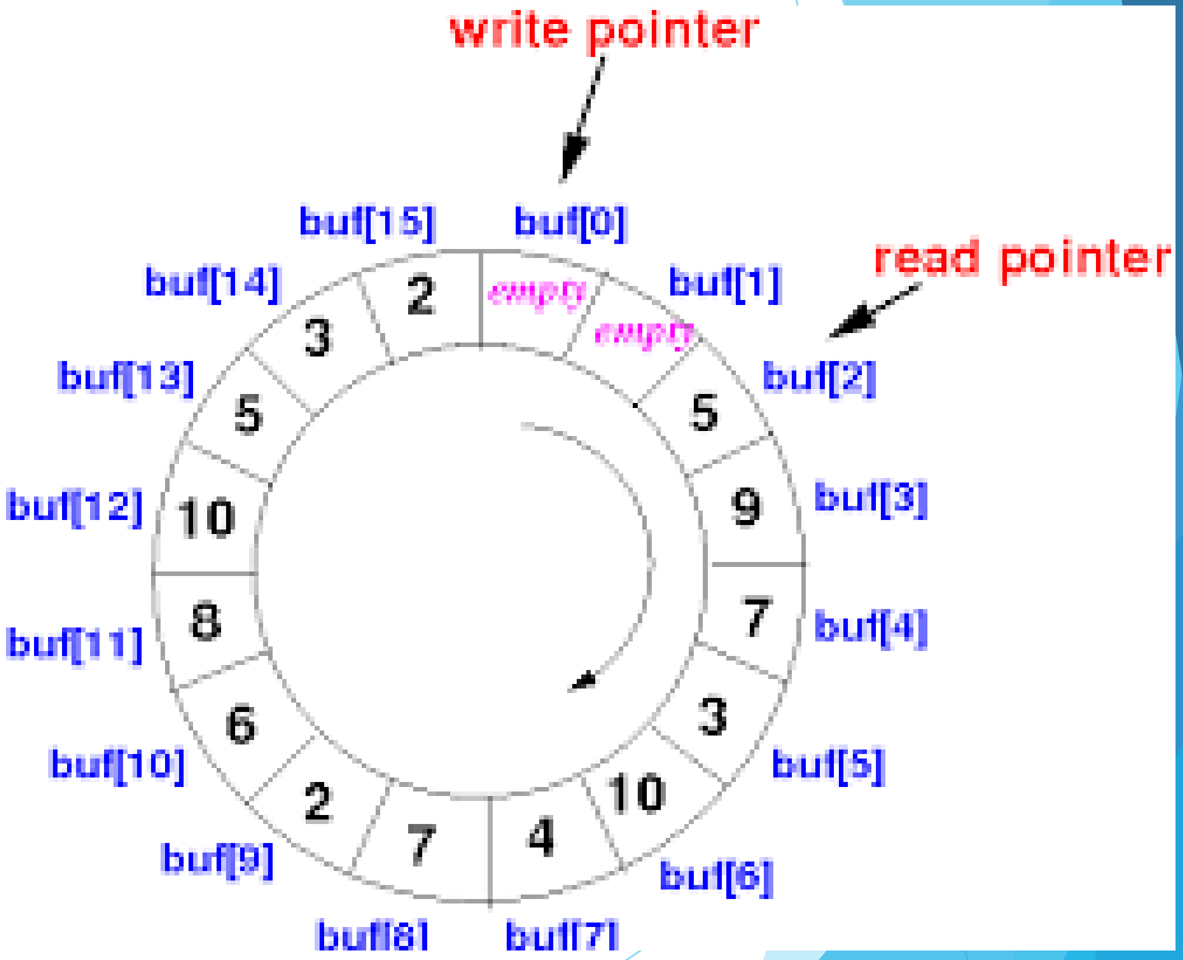
Implementation using a circular array

Array:



Pretend array is a circle:





Circular array implementation of a queue

- Invariant:
  - Array/list `items` large enough to hold the entire queue
  - Variable `capacity` for the fixed size of the array
  - Variable `size` that tells how many items are in the queue
  - So 0 is less than or equal to `size` is less than or equal to `capacity`
  - Variable `head` is the index of the front of the queue
  - tail == (head+size-1)%capacity

- If size > 0, the queue items are at locations `items[head]` through `items[head+size-1)%capacity]`
- If size ==0, head == (tail + 1)%capacity

Simulation using a queue

- Modelling the behavior of a real-word queue - supermarket, ticket line, bank, restaurant, car wash
- Example – small grocery store with only one register. Use a simulation to find out
  - How long customers wait on average? How long does the line get? What is the maximum wait?
- Simulate the check out process.
- Customers arrive with a number of items and are served in the order they arrive.
- There is a randomness to the times at which they arrive and to the number of items they have.
- The time to check out depends on the number of items.

Simulation using a queue

- Some abstractions to simplify
- The simulation will be controlled by “clock ticks” or counting. Think of a “clock tick” as representing a second.
- A customer consists of an arrival time and a number of items. The numbers are generated randomly subject to some conditions and stored in a file.
- See simulation.py for genTestData
- If the store serves 30 customers per hour, then one customer arrive on average every 2 minutes or 120 seconds.
- So each second there is a 1/120 probability of a customer arriving.
- Generate a random number in [0, 1) and if that number is < 1/arrivaltime, create a customer.

Simulation using a queue

- Create a class for Customer
- Read the file into a queue of Customer objects.
- Simulate using CheckerSim object.
- CheckerSim object takes a queue of Customers and an average processing time for one item as parameters and computes a number of statistics.
  - averageWait
  - maximumWait
  - maximumLineLength
- Run method sets the clock ticking- time driven
- At each clock tick, any customer in the queue arriving at that time is move into the checkout line.
- If the checker is processing another Customer, this Customer has to wait to be processed – decrease the serviceTime variable
- Once the serviceTime variable is 0, if there is a Customer waiting, process them and update the statistics