The Art of Data Structures

Queues

Alan Beadle
CSC 162: The Art of Data Structures
Agenda

• What Is a Queue?
• The Queue Abstract Data Type
• Implementing a Queue in Python
• Simulation: Hot Potato
• Simulation: Printing Tasks
Queues
Queues

- Heavily used for things that take turns "first in, first out" -- FIFO

- Examples:
  - “Ready” list for processes in your computer
  - Incoming messages from the Internet (e.g. to web server)
  - Pending requests to the disk drive
Queues

A Queue of Python Data Objects
Implementation
Implementation

Queue Operations

• `Queue()` creates a new queue that is empty; it needs no parameters and returns an empty queue.

• `enqueue(item)` adds a new item to the rear of the queue; it needs the item and returns nothing.

• `dequeue()` removes the front item from the queue; it needs no parameters, returns the item and the queue is modified.
Implementation

Queue Operations

- `is_empty()` tests to see whether the queue is empty; it needs no parameters and returns a boolean value
- `size()` returns the number of items on the queue; it needs no parameters and returns an integer
# Implementation

## Queue Operations

<table>
<thead>
<tr>
<th>Queue Operation</th>
<th>Queue Contents</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>q.is_empty()</td>
<td>[]</td>
<td>TRUE</td>
</tr>
<tr>
<td>q.enqueue(4)</td>
<td>[4]</td>
<td></td>
</tr>
<tr>
<td>q.enqueue('dog')</td>
<td>['dog',4]</td>
<td></td>
</tr>
<tr>
<td>q.enqueue(True)</td>
<td>[True,'dog',4]</td>
<td></td>
</tr>
<tr>
<td>q.size()</td>
<td>[True,'dog',4]</td>
<td>3</td>
</tr>
<tr>
<td>q isempty()</td>
<td>[True,'dog',4]</td>
<td>FALSE</td>
</tr>
<tr>
<td>q.enqueue(8.4)</td>
<td>[8.4,True,'dog',4]</td>
<td></td>
</tr>
<tr>
<td>q.dequeue()</td>
<td>[8.4,True,'dog']</td>
<td>4</td>
</tr>
<tr>
<td>q.dequeue()</td>
<td>[8.4,True]</td>
<td>dog'</td>
</tr>
<tr>
<td>q.size()</td>
<td>[8.4,True]</td>
<td>2</td>
</tr>
</tbody>
</table>
Queue Implementation in Python

class Queue:
    def __init__(self):
        self.items = []

    def is_empty(self):
        return self.items == []

    def enqueue(self, item):
        self.items.insert(0, item)

    def dequeue(self):
        return self.items.pop()

    def size(self):
        return len(self.items)
Simulation: Hot Potato
Simulation: Hot Potato

A Six Person Game of Hot Potato

Bill

- pass to next person

Brad

- after 5 passes, Brad is eliminated
- until predefined counting constant

Kent

David

- and so on

Susan

Jane
Simulation: Hot Potato

A Six Person Game of Hot Potato

- Whoever has the "potato" when the "music" stops drops out simulation:
  - person at head of queue "has the potato"
  - dequeue and enqueue to simulate passing it to one person
Simulation: Hot Potato

A Queue Implementation of Hot Potato

enqueue

Go to the rear
(Pass the potato)

dequeue
Simulation: Hot Potato

A Python Implementation

def hot_potato(namelist, num):
    simqueue = Queue()

    for name in namelist:
        simqueue.enqueue(name)

    while simqueue.size() > 1:
        for i in range(num):
            simqueue.enqueue(simqueue.dequeue())

    simqueue.dequeue()

    return simqueue.dequeue()
Simulation: Hot Potato

A Python Implementation

Round 1: cycles 7 times, producing:
["David", "Susan", "Jane", "Kent", "Brad", "Bill"]
then deletes David

Round 2: cycles 7 times, producing
["Kent", "Brad", "Bill", "Susan", "Jane"]
then deletes Kent

Round 3: cycles 7 times, producing
["Jane", "Brad", "Bill", "Susan"]
then deletes Jane

Round 4: cycles 7 times, producing
["Bill", "Susan", "Brad"]
then deletes Bill

Round 5: cycles 7 times, producing
["Brad", "Susan"]
then deletes Brad
leaving Susan
Simulation: Hot Potato

A Python Implementation

• It might make more sense just to have a list, cycle a cursor through it, and eliminate the "person" it points at...
Simulation: Printing Tasks
Simulation: Printing Tasks

*Computer Science Lab Printing Queue*
class Printer:
    def __init__(self, ppm):
        self.pagerate = ppm
        self.currentTask = None
        self.timeRemaining = 0

    def tick(self):
        if self.currentTask != None:
            self.timeRemaining = self.timeRemaining - 1
        if self.timeRemaining <= 0:
            self.currentTask = None

    def busy(self):
        if self.currentTask != None:
            return True
        else:
            return False

    def startNext(self, newtask):
        self.currentTask = newtask
        self.timeRemaining = newtask.getPages() * 60/self.pagerate
import random

class Task:
    def __init__(self, time):
        self.timestamp = time
        self.pages = random.randint(1, 21)

    def getStamp(self):
        return self.timestamp

    def getPages(self):
        return self.pages

    def waitTime(self, currenttime):
        return currenttime - self.timestamp
import random

def simulation(numSeconds, pagesPerMinute):
    labprinter = Printer(pagesPerMinute)
    printQueue = Queue()
    waitingtimes = []

    for currentSecond in range(numSeconds):
        if newPrintTask():
            task = Task(currentSecond)
            printQueue.enqueue(task)

        if (not labprinter.busy()) and (not printQueue.isEmpty()):
            nexttask = printQueue.dequeue()
            waitingtimes.append(nexttask.waitTime(currentSecond))
            labprinter.startNext(nexttask)

        labprinter.tick()

    averageWait = sum(waitingtimes) / len(waitingtimes)
    print("Average Wait %6.2f secs %3d tasks remaining." % (averageWait, printQueue.size()))

Simulation: Printing Tasks
Printer Queue Simulation: Printer Class
def newPrintTask():
    num = random.randrange(1,181)
    if num == 180:
        return True
    else:
        return False

for i in range(10):
    simulation(3600,5)
This simulation could be used to answer questions such as:

- Will the available printer keep up if enrollment increases?
- What if the size of the average print task decreases since Python is such a powerful language and programs tend to be much shorter?
However, it is important to remember that a simulation is only as good as the assumptions that are used to build it.

Real data about the number of print tasks per hour and the number of students per hour is necessary to construct a robust simulation.

Also, the printer might jam sometimes.
Questions?