Topic 16 Queues

"FISH queue: n.

[acronym, by analogy with FIFO (First In, First Out)] 'First In, Still Here'. A joking way of pointing out that processing of a particular sequence of events or requests has stopped dead. Also FISH mode and FISHnet; the latter may be applied to any network that is running really slowly or exhibiting extreme flakiness."

-The Jargon File 4.4.7

Queues

- A sharp tool, like stacks
- A line

–In England people don't "get in line" they "queue up".





Queue Properties

- Queues are a first in first out data structure
 - FIFO (or LILO, but I guess that sounds a bit silly)
- Add items to the end of the queue
- Access and remove from the front
 - Access to the element that has been in the structure the *longest* amount of time
- Used extensively in operating systems
 - Queues of processes, I/O requests, and much more
 CS314





Queues in Operating Systems

- On a computer with N cores on the CPU, but more than N processes, how many processes can actually be executing at one time?
- One job of OS, schedule the processes for the CPU

			System M	Ionitor				
<u>M</u> onitor	<u>E</u> dit <u>V</u> iew <u>H</u>	<u>H</u> elp						
Processes	s Resources	File Syster	ns					
Load ave	erages for the	last 1, 5, 1	L5 minutes	: 1.28, 1.4	9,1.40			
Process Name 🔹		Status	% CPU	Nice	ID	Memory		
at	at-spi-registryd		Sleeping	0	0	3683	2.4 MiB	
bo	onobo-activati	on-server	Sleeping	0	0	3158	332.0 KiB	
bt	-applet		Sleeping	0	0	3179	212.0 KiB	
🖗 cl	ock-applet		Sleeping	0	0	3241	676.0 KiB	
dt	ous-daemon		Sleeping	0	0	3118	164.0 KiB	
dt	ous-launch		Sleeping	0	0	3120	0 bytes	
eç	ggcups		Sleeping	0	0	3165	1.6 MiB	
es	cd		Sleeping	0	0	3200	44.0 KiB	
🥘 fir	efox		Sleeping	0	0	15110	0 bytes	
🧕 🥹 fir	efox-bin		Sleeping	0	0	15126	143.5 MiB	
ga	am_server		Sleeping	0	0	3220	156.0 KiB	
go	confd-2		Sleeping	0	0	3126	504.0 KiB	
🖂 ge	edit		Sleeping	0	0	1464	6.9 MiB	•
							End Pro	ocess

(Q- Filter			Windowed Processes				
it Process II	nspect Filte	r	Show				
Process ID	Process Name	User	% CPU	# Threads	Real Memory	VSIZE	
361	📫 Finder	steve	0.00	4	21.05 MB	238.26 MB	
366	🇑 Safari	steve	0.00	6	23.53 MB	252.85 MB	
346	😡 loginwindow	steve	0.00	2	3.80 MB	185.66 MB	
367	🕼 iTunes	steve	4.00	10	22.03 MB	239.66 MB	
371	Activity Monitor	steve	2.80	2	20.11 MB	246.64 MB	
368	🜠 iPhoto	steve	0.00	3	33.39 MB	281.90 MB	
374	Terminal	steve	0.00	4	12.98 MB	244.08 MB	
360	😡 SystemUlServer	steve	0.00	2	5.35 MB	227.74 MB	
359	🔝 Dock	steve	0.00	2	6.43 MB	200.11 MB	
	CPU System Me	mory Disk A	ctivity D	isk Usage	Network)	
				(CPU Usage	r	
% User: 4.50		Threads:	213				
% Syst	tem: 4.00	Processes:	65				
	Nice: 0.00						
%	Idle: 91.50						

Queue operations

- void enqueue(E item)
 - -a.k.a. add (E item)
- ► front()
 - **a.k.a.** E peek()
- ▶ E dequeue()
 - a.k.a. E remove()
- boolean isEmpty()
- Specify methods in an interface, allow multiple implementations.

Queue interface, version 1

public interface Queue314<E> {
 //place item at back of this queue
 public void enqueue(E item);

//access item at front of this queue
//pre: !isEmpty()
public E front();

//remove item at front of this queue
//pre: !isEmpty()
public E dequeue();

```
public boolean isEmpty();
```

}

Implementing a Queue

Given the internal storage container and choice for front and back of queue what are the Big O of the queue operations?

ArrayList

LinkedList (Singly Linked) LinkedList (Doubly Linked)

enqueue

front

dequeue

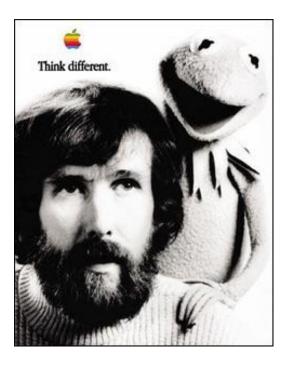
isEmpty

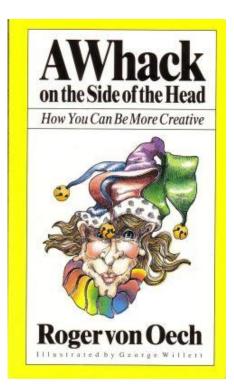
Clicker 1

- If implementing a queue with a singly linked list with references to the first and last nodes (head and tail) which end of the list should be the front of the queue in order to have all queue operations O(1)?
- A. The front of the list should be the front of the queue.
- B. The back of the list should be the front of the queue.
- C. Either end will work to make all ops O(1).
- D. Neither end will allow all ops to be O(1).

Alternate Implementation

- How about implementing a Queue with a native array?
 - Seems like a step backwards





Application of Queues

Radix Sort

- radix is a synonym for *base*. base 10, base 2

- Multi pass sorting algorithm that only looks at individual digits during each pass
- Use queues as *buckets* to store elements
- Create an array of 10 queues
- Starting with the least significant digit place value in queue that matches digit
- empty queues back into array
- repeat, moving to next least significant digit

Radix Sort in Action: 1s place

- original values in array
 9, 113, 70, 86, 12, 93, 37, 40, 252, 7, 79, 12
- Look at ones place
 <u>9</u>, 11<u>3</u>, 7<u>0</u>, 8<u>6</u>, 1<u>2</u>, 9<u>3</u>, 3<u>7</u>, 4<u>0</u>, 25<u>2</u>, <u>7</u>, 7<u>9</u>, 1<u>2</u>

8

9

Array of Queues (all empty initially):

0 5

- 1 6 2 7
- 3

4

Radix Sort in Action: 1s

- original values in array
 9, 113, 70, 86, 12, 93, 37, 40, 252, 7, 79, 12
- Look at ones place
 <u>9</u>, 11<u>3</u>, 7<u>0</u>, 8<u>6</u>, 1<u>2</u>, 9<u>3</u>, 3<u>7</u>, 4<u>0</u>, 25<u>2</u>, <u>7</u>, 7<u>9</u>, 1<u>2</u>
- Queues:
 - 0 7<u>0</u>, 4<u>0</u> 5 1 6 86
 - 2 1<u>2</u>, 25<u>2</u>, 1<u>2</u>
 - 3 11<u>3</u>, 9<u>3</u>

4

7 37, 7

9 <u>9</u>, 7<u>9</u>

8

Radix Sort in Action: 10s

Empty queues in order from 0 to 9 back into array

70, 40, 12, 252, 12, 113, 93, 86, 37, 7, 9, 79

- Now look at 10's place
 <u>7</u>0, <u>4</u>0, <u>1</u>2, 2<u>5</u>2, <u>1</u>2, 1<u>1</u>3, <u>9</u>3, <u>8</u>6, <u>3</u>7, <u>7</u>, <u>9</u>, <u>7</u>9
- Queues:
 - 0 $_7, _9$ 5 252

 1 12, 12, 113 6

 2
 7 70, 79

 3 37 8 86

 4 40 9 93

Radix Sort in Action: 100s

- Empty queues in order from 0 to 9 back into array
 7, 9, 12, 12, 113, 37, 40, 252, 70, 79, 86, 93
- Now look at 100's place

___7, __9, _12, _12, <u>1</u>13, _37, _40, <u>2</u>52, _70, _79, _86, _93

• Queues:

0 _7, _9, _12, _12, _37, _40, _70, _79, _86, _93

 1
 <u>1</u>13
 6

 2
 <u>2</u>52
 7

 3
 8
 8

 4
 9

5

Radix Sort in Action: Final Step

- Empty queues in order from 0 to 9 back into array
 - 7, 9, 12, 12, 40, 70, 79, 86, 93, 113, 252

Radix Sort Code

```
public static void sort(int[] data) {
      ArrayList<Queue<Integer>> queues
           = new ArrayList<Queue<Integer>>();
      for (int i = 0; i < 10; i++)
          queues.add(new LinkedList<Integer>());
      int passes = numDigits(getMax(data));// helper methods
      for(int i = 0; i < passes; i++) {</pre>
          for (int j = 0; j < data.length; j++) {
               int digit = valueOfDigit(data[j], i);
              queues.get(digit).add(data[j]);
          int pos = 0;
          for(Queue<Integer> q : queues) {
              while(!q.isEmpty())
                   data[pos++] = q.remove();
          }
  ւ CS314
                                                         16
```