YEAH!

Priority Queue

Sahil Chopra - 2.16.2016

Adapted from SLs Brendon Go & Rishi Bedi

What’s a Priority Queue?

- Queue where elements are enqueued w/ priority rating
- Elements dequeued according to priority rating
- Element w/ priority 0 is higher priority than elements w/ priority 1, 2, 3, … 100, etc.

- Example: Routing patients at hospital ER
  - Regardless of arrival time, some cases are more urgent & will be handled more quickly (higher priority)

3 Implementations

- Abstraction: Enqueue & Dequeue via “Sorted” Queue
- Implementation:
    - ArrayPriorityQueue
    - LinkedListPriorityQueue
    - HeapPriorityQueue
- Sorted Order:
    - According to priority value
    - Remember smaller integer = greater priority
    - Break priority ties by alphabetical ordering

Methods To Implement

```
pq.enqueue(value, priority);
pq.peek();
pq.changePriority(value, newPriority);
pq.size();
out << pq
PriorityyQueue()
```

```
pq.dequeue();
pq.peekPriority();
pq.isEmpty();
pq.clear();
~PriorityQueue()
```

Array PQ

- Unsorted Array for internal data storage
- Private Member Variables (Restricted To):
  - Pointer to internal array of elements
  - Integer for array’s actual capacity
  - Integer for pq’s size
- Enqueue: Add elements to end of array
  - O(N) = ?
- Dequeue: Search array for value w/ greatest priority
  - O(N) = ?

Array PQ

- Enqueue: Add elements to end of array
  - O(N) = 1
- Dequeue: Search array for value w/ greatest priority
  - O(N) = N → Inefficient

```
index     0       1       2       3       4       5       6       7       8       9
       +-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+
 value | "x":5 | "b":4 | "a":8 | "m":5 | "q":5 | "t":2 |       |       |       |       |
       +-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+
```

- PQEntry: Class w/ Integer priority and char value

Array PQ

- `enqueue(value, priority)`
  - add to end of array
  - what do we do when we run out of space (see vector class ex.)
- `peek, peekPriority, dequeue`
  - go through all elements to find minimum
  - don’t forget to erase when you dequeue
  - what do we do with the gap?
- `isEmpty(), size()`
  - what values should you consult?
- `clear ()`
  - do we need to free memory? are other other options?
- `changePriority()`
  - just change the priority value

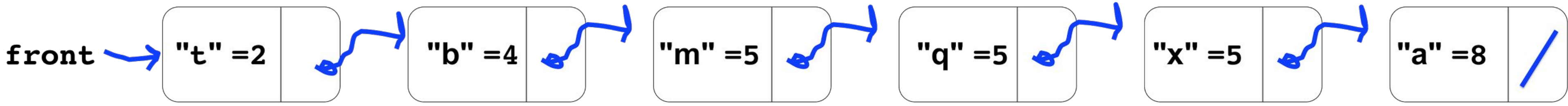
Linked List PQ

- Linked List for internal data storage sorted by priority val
- Private Member Variables (Restricted To):
    - Pointer to front of list
- Enqueue: Find appropriate place in sorted linked list
    - O(N) = ?
- Dequeue: Remove from the front of the linked list
    - O(N) = ?

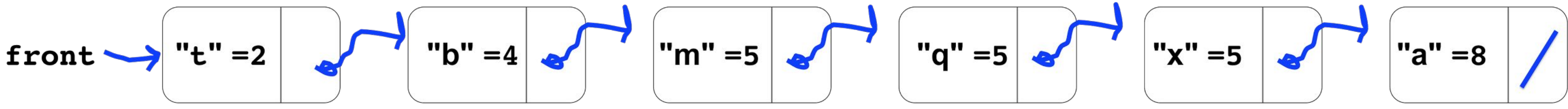
Linked List PQ

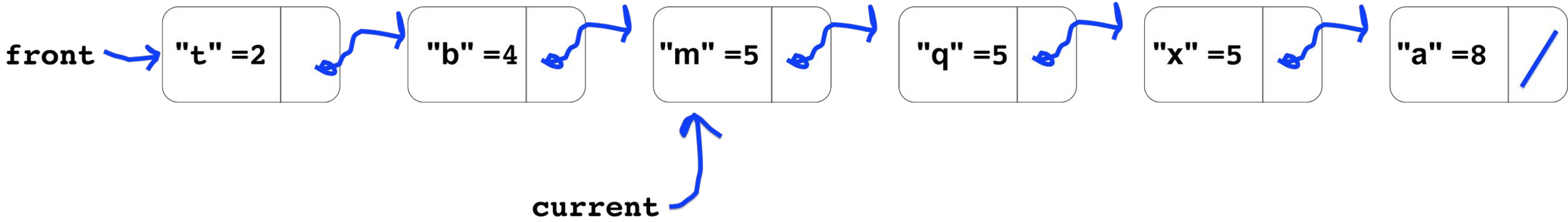
- Enqueue: Find appropriate place in sorted linked list
    - O(N) = N
- Dequeue: Remove from the front of the linked list
    - O(N) = 1

Linked List PQ - Enqueue “O” w/ Priority 5

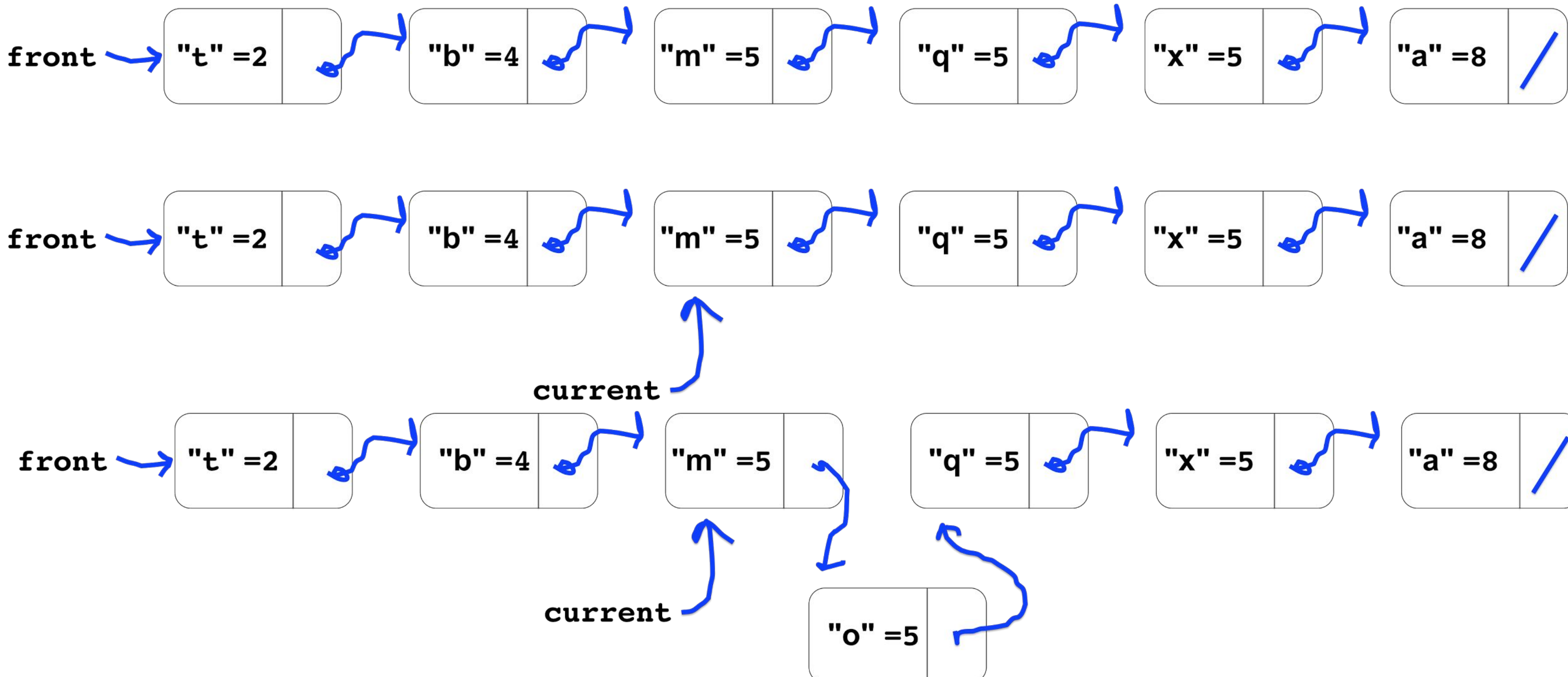


Linked List PQ - Enqueue “O” w/ Priority 5





Linked List PQ - Enqueue “O” w/ Priority 5



Linked List PQ

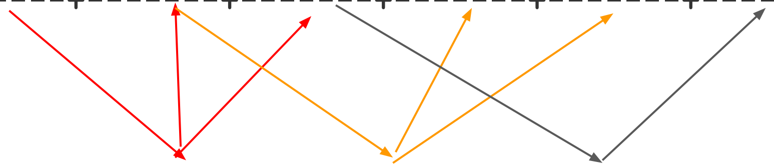
- `enqueue(value, priority)`
  - traverse linked list until insertion point is located
- `peek, peekPriority, dequeue`
  - examine front of linked list
- `isEmpty(), size()`
  - what does “front” equal if linked list is empty?
  - how do we determine size w/out variable?
- `clear ()`
  - do we need to free memory?
- `changePriority()`
  - find and remove list node
  - enqueue list node w/ new priority value

Heap PQ

- Binary Heap sorted by priority val
- Private Member Variables (Restricted To):
  - Pointer to array: `*PQEntry[]`
  - Int - Array Capacity & Int - PQ Size
- Enqueue: Place at end of array and “bubble up”
  - O(N) = ?
- Dequeue: Remove from front of array, select last element, place at the front, and “bubble down”
  - O(N) = ?

Heap PQ

```
index       0       1       2       3       4       5       6       7       8       9
      +-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+
value |       | "t":2 | "m":5 | "b":4 | "x":5 | "q":5 | "a":8 |       |       |       |
      +-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+
size = 6
capacity = 10
```



Remember:

- element at index *i* has two children at *2*i* and *2*i+1*
- parent has higher priority (smaller value) than children
- skip element at index 0 to make math easier

Heap PQ - Enqueue “y” w/ Val 3

```
index     0       1       2       3       4       5       6       7       8       9
      +-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+
value |       | "t":2 | "m":5 | "b":4 | "x":5 | "q":5 | "a":8 |       |       |       |
      +-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+
size = 6
capacity = 10
```

- Add y:3 at index at 7, examine parent at index 3
- b:4 < y:3 so swap elements at index 3 and 7
- Examine parent at index 1, y:3 < t:2 so stop

```
index     0       1       2       3       4       5       6       7       8       9
      +-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+
value |       | "t":2 | "m":5 | "y":3 | "x":5 | "q":5 | "a":8 | "b":4 |       |       |
      +-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+
size = 7
capacity = 11
```

Heap PQ - Dequeue

```
index     0       1       2       3       4       5       6       7       8       9
      +-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+
value |       | "t":2 | "m":5 | "y":3 | "x":5 | "q":5 | "a":8 | "b":4 |       |       |
      +-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+
size = 7
capacity = 11
```

- Remove t:2 from index 1, place b:4 from index 7 at index 1 to replace t:2

```
index     0       1       2       3       4       5       6       7       8       9
      +-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+
value |       | "b":4 | "m":5 | "y":3 | "x":5 | "q":5 | "a":8 |       |       |       |
      +-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+
size = 6
capacity = 10
```

Heap PQ - Dequeue

```
index     0       1       2       3       4       5       6       7       8       9
      +-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+
value |       | "b":4 | "m":5 | "y":3 | "x":5 | "q":5 | "a":8 |       |       |       |
      +-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+
size = 6
capacity = 10
```

- Examine children at indices 3 and 4, b:4 < y:3 so swap them
- Examine children at index 6, b:4 > a:8 so stop

```
index     0       1       2       3       4       5       6       7       8       9
      +-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+
value |       | "y":3 | "m":5 | "b":4 | "x":5 | "q":5 | "a":8 |       |       |       |
      +-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+
size = 6
capacity = 10
```

Heap PQ

- `enqueue(value, priority)`
  - “bubble up” = compare w/ parent & swap if necessary (iterative or recursive?)
  - remember to resize the array if all slots are filled
- `dequeue()`
  - remove at index 1, move last element to index 1
  - “bubble down” = compare w/ child & swap if necessary (iterative or recursive?)
- `peek, peekPriority()`
  - look at index 1 in array
- `isEmpty(), size()`
  - examine capacity and size variables
- `clear ()`
  - do we need to free memory? if not, where should we free the memory?
- `changePriority()`
  - change priority value & then bubble up (can only make items more urgent)