Stacks and Queues IV

Lecture 9

Prof. Eric Polizzi
Summary previous lectures

- Stacks and Queues are **abstract data structures**; Complexity O(1).

### Stack (LIFO)

**Basic operations using a Stack**
- push(item); pop(); peek()
- isEmpty(); isFull(); size()

### Queue (FIFO)

**Basic operations using a Queue**
- enqueue(item); dequeue(); peek()
- isEmpty(); isFull(); size()
Priority Queues

- It is a more specialized data structure than a simple queue
- Each item has a “priority” associated with it (ordered by key values)
- Item with lowest or highest key is at the front
- Items are inserted in proper position to maintain order

Useful as programming tool; Examples:
- Multitasking operating system (highest priority program/job must be placed in priority queue)
- Minimum spanning tree for weighted graph (see later in class)
Priority Queue - implementation

- We choose a model where removal/dequeue at the front is easy

  - $\text{maxSize}$ is the capacity of the queue
  - $\text{rear/font}$ indexes of the item at the rear/front ("front at the top" makes removal easier)
  - $A, B, D, E$ are primitive data (int, long, double, etc.) or objects

- After $C$ is added (enqueue/insert)

- After $A$ leaves (dequeue/remove)
Priority Queue- implementation

Remarks:
- 'rear' is always at 0 and 'front' is always at (nItems-1)
- Both indexes are unnecessary for ”ordered array” implementation

Problem: dequeue is O(1) but enqueue is O(N).... slow insertion

Priority Queue is often implemented using a heap to improve insertion time

To do: Test Java applet PriorityQ.html
Example using array of integer (lowest key value in front)

```java
class PriorityQ {
    // variables
    private int maxSize; // PriorityQ capacity
    private int nItems; // #items in PriorityQ
    private int[] array; // array in sorted order that holds the items

    // constructors
    public PriorityQ(int maxSize) {
        this.maxSize = maxSize; // set array size
        array = new int[maxSize]; // create array
        nItems = 0; // no item yet
    }
}
```
Example using array of integer (lowest key value in front)

```java
class PriorityQ {
    // variables
    // constructors
    // methods

    public boolean isEmpty() {
        return (nItems==0); //true if PriorityQ is empty
    }

    public boolean isFull() {
        return (nItems==maxSize); //true if full
    }

    public int size() {
        return (nItems); // return current number of items
    }
}
```
```java
public int peekFront() { //peek front item
    if (isEmpty())
        throw new IllegalStateException("PQ is empty");
    return array[nItems-1]; //return front item
}

public int dequeue() { //remove item
    if (isEmpty())
        throw new IllegalStateException("PQ is empty");
    nItems--;
    return array[nItems+1];
}

public void enqueue(int item) { //insert item
    if (isFull())
        throw new IllegalStateException("PQ is full");
    int i=nItems-1;
    while(i>=0 && array[i]<=item) {
        array[i+1] = array[i]; //shift up
        i--;
    } //go down one position
    array[i+1]=item; //insert item
    nItems++; //increment #items
}
```
Priority Queue- Example

- can you guess the result?

```java
class PriorityQApp1 {
    public static void main(String[] args) {
        PriorityQ myqueue = new PriorityQ(5);
        myqueue.enqueue(30);
        myqueue.enqueue(50);
        myqueue.enqueue(10);
        myqueue.enqueue(40);
        myqueue.enqueue(20);
        while(!myqueue.isEmpty()){
            System.out.print(myqueue.dequeue() + " ");
        }
    }
}
```

Result is 10 20 30 40 50
Priority Queues: A Model using Multiple Queues

- Your preferred airline has only one check-in service at the airport
- Two waiting lines for passengers:
  - One is the coach class
  - One is the business class
- Passengers in business class have higher priority to check in
- Two queues: one with high priority and one with low priority
Priority Queues: A Model using Multiple Queues

- **Pseudocode**

```java
Queue lowQueue = new Queue(5); // low priority queue (coach)
Queue highQueue = new Queue(5); // high priority queue (buisness)

// Passenger arrival (at the rear of respective queues)
if (newPassengerCome){
    if (isBuisness)
        highQueue.enqueue(newPassenger);
    else
        lowQueue.enqueue(newPassenger);
}

// Check-in Service
if (!highQueue.isEmpty())
    highQueue.dequeue(); // Serve Buisness passengers in priority
else
    lowQueue.dequeue(); // Serve next Coach passenger
```