CS310 – Advanced Algorithms and Data Structures

Spring, 2021

Priority Queues and Event-Driven Simulation
Priority queues

- A priority queue holds elements that have a “priority” value, and we can new elements, or get back out the one with minimum priority value.

[Diagram showing Insert and DeleteMin operations]
Priority queues

• For example, suppose incoming messages to an email support center each have a priority (lowest number is highest priority.)
• Unanswered messages are held in a priority queue, and new messages are added to it.
• When a support engineer becomes free, they take out the message of highest priority to work on.
• It might be a recently received message or an old message—this is not a normally-ordered queue.
Simple Priority queue example using element’s compareTo

```java
import java.util.PriorityQueue; ← JDK class

public class PriorityQueueDemo
{
    public static void main( String [ ] args )
    {
        PriorityQueue<Integer> minPQ = new PriorityQueue<Integer>( );
        minPQ.add( 4 );
        minPQ.add( 3 );
        minPQ.add( 5 );
        while (!minPQ.isEmpty()) {
            System.out.println(minPQ.poll()+",");
        }
    }
}

Output: MinPQ:3, 4, 5,
Note that duplicates are OK here: If we add another 3, we just get two 3s back:
MinPQ:3, 3, 4, 5,
Often data are processed in a specific order, the priority

Examples

- In OS process scheduling, a process comes with a priority, and high priority processes are executed earlier than low priority ones
- In network routing, high priority traffic (VoIP, IPTV) are delivered before other traffic

We need to dynamically maintain the pending jobs
- New jobs keep coming in
- When a server becomes available, the highest-priority job is removed from PQ and serviced

Convention: non-negative, and a smaller value has a higher priority
There is a Queue interface in the JDK, a subinterface of Collection

The JDK docs list various implementing concrete classes including ArrayDeque, previously listed here, and PriorityQueue

So a PriorityQueue ISA Collection, so it has a “wide” interface

Thus a PriorityQueue ISA Queue ISA Collection

The element class needs to implement compareTo to compare priorities of elements, or, for the JDK class, a Comparator can be provided.
## Operations of a Priority Queue

- **insert(item)**: Insert into proper place in PQ
- **FindMin()**: Return min element by priority
- **DeleteMin()**: Delete and return min element by priority

**isEmpty()** and/or **size()** also needed

<table>
<thead>
<tr>
<th>S&amp;W MinPQ(p. 309)</th>
<th>JDK PriorityQueue&lt;E&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Insert</strong></td>
<td><strong>insert(item)</strong></td>
</tr>
<tr>
<td><strong>FindMin</strong></td>
<td><strong>Key min()</strong></td>
</tr>
<tr>
<td><strong>DeleteMin</strong></td>
<td><strong>Key delMin()</strong></td>
</tr>
</tbody>
</table>
## Performance of Operations of a Priority Queue

- Implementation by different data structures, worst case runtime:

<table>
<thead>
<tr>
<th></th>
<th>Unsorted array</th>
<th>Sorted array</th>
<th>Linked List</th>
<th>Balanced tree</th>
<th>JDK/Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>FindMin</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>DeleteMin</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
</tbody>
</table>

- Note that since Insert and DeleteMin are $O(\log n)$, can sort $n$ items in $O(n \log n)$ time by Inserts, then DeleteMins. That’s a good sort!
Recall SimpleStudent, with equals, hashCode, and compareTo based on id, an integer (see next slide for code)

We could add students to a priority queue, then loop over DeleteMins

The first one out would be the smallest id, then the next, ...

But this could be done by a TreeSet, or an array sort, not that interesting

Suppose we add GPA to SimpleStudent, a new instance variable, and want students in GPA order
SimpleStudent Example: another Set-ready object

```java
import java.lang.Integer;  // and adding gpa for Student...

class SimpleStudent implements Comparable<SimpleStudent> {
    private int id; private String name;  //add private double gpa;

    public SimpleStudent(String n, int i) { name=n; id =i; }
    // getters and setters for name and id go here  //and getGpa()

    public boolean equals(Object rhs) {  //more compact code, but still correct
        if (rhs ==null || getClass()  != rhs.getClass())
            return false;
        SimpleStudent other =(SimpleStudent) rhs;
        return id ==other.id;
    }

    public int compareTo(SimpleStudent other)
    { return 
        Integer.valueOf(id).compareTo(Integer.valueOf(other.id));
    }

    public int hashCode()
    { return Integer.valueOf(id).hashCode(); }
}
```

2/25/2021
PriorityQueue Example: Students in GPA order

- With GPA added to SimpleStudent, it’s not so simple, call the result Student
- Student still has equals, hashCode, and compareTo based on id
- It has a new instance variable, gpa, and we want students in GPA order
- We can set up a JDK PriorityQueue with priority given by the student’s GPA
- The JDK PriorityQueue allows us to provide a Comparator (comparing two elements by GPA) to do this
- Comparators are discussed in S&W pp. 338-339.
- Comparator\(<T>\) has method int compare(T o1, T o2)
Java Comparators, then and now

• Previously, Java Comparators were hard to use.

• We had to write a whole class to create one.
• See GeeksForGeeks article

• Java 8+ is willing to create a Comparator based on any specified getter of a class, for example:

• Create a Comparator to compare Students by gpa:

```java
Comparator<Student> byGpa = 
    Comparator.comparing(Student::getGpa);
```

• Note the new Java syntax to specify which getter to use
public static void main(String[] args) {
    // new Java 8 features--easy Comparators!
    // Java8 Comparator method "comparing" builds a specific-field comparator, here on GPA--
    Comparator<Student> byGpa = Comparator.comparing(Student::getGpa);
    // or use a lambda function--see code online
    PriorityQueue<Student> pq =
        new PriorityQueue<Student> (byGpa);
    pq.add(new Student("Ann", 100, 2.3));
    pq.add(new Student("Ling", 104, 4.0));
    pq.add(new Student("Dave", 104, 3.1));
    System.out.println(pq); // note duplicates (equal id)
    // Use DeleteMin to pull them out in GPA order--
    while (!pq.isEmpty()) {
        Student x = pq.poll();
        System.out.println(x);
    }
}
// Java8 Comparator method "comparing" builds a specific-field Comparator, here on GPA--
Comparator<Student> byGpa =
    Comparator.comparing(Student::getGpa);
PriorityQueue<Student> pq =
    new PriorityQueue<Student>(byGpa);
    pq.add(new Student("Ann",100, 2.3));
    pq.add(new Student("Ling", 104, 4.0));
    pq.add(new Student("Dave",104, 3.1));
System.out.println(pq); // note duplicates (equal id)
// Use DeleteMin to pull them out in GPA order--
while (!pq.isEmpty()) {
    Student x = pq.poll();
    System.out.println(x);
}

-------------
[(Ann 100 2.3), (Ling 104 4.0), (Dave 104 3.1)]
(Ann 100 2.3)            <-- Student.toString() output
(Dave 104 3.1)
(Ling 104 4.0)
Comparators can be used in many places:

- To specify priority order for a priority queue, as we just saw.
- To construct a TreeSet<E> based on any ordering
- To do sorts in a specified order (we’ll review sorting later)
```java
public static void main(String[] args)
{
    // Java8 Comparator method "comparing" builds a specific-field comparator, here on GPA--
    Comparator<Student> byGpa =
        Comparator.comparing(Student::getGpa);
    // use comparator in TreeSet constructor to
    // specify order--
    Set<Student> pqset =
        new TreeSet<Student> (byGpa);
    pqset.add(new Student("Ann", 100, 2.3));
    pqset.add(new Student("Ling", 104, 4.0));
    pqset.add(new Student("Dave", 104, 3.1));
    System.out.println(pqset); // note dupes (equal id)
    // Use iterator to access them in GPA order--
    for (Student x: pqset) {
        System.out.println(x);
    }
}
```
So why bother with PriorityQueue??

Well, a PQ is used where new items are coming in and old items are being thrown away: it fits this scenario.

We can handle the deleteMin with a TreeSet: create an Iterator, do a next() on it and then remove().

So we can use a TreeSet to *implement* a PriorityQueue, but the PriorityQueue is still a useful abstraction with a simpler API.
More capable PriorityQueues, with changeKey

• What if student of id 100 *in* the PQ has changed GPA?

• To fix this, we need to scan the PQ to find the student, then delete that student and reinsert them. Pretty expensive.

• BTW, we can’t just change the GPA value for that student (once found) without breaking the TreeSet data structure.*

• There is a more capable version of PriorityQueue implemented in S&W, the IndexMinPQ, API on page 320. It has a method changeKey that would be more efficient (log n) for the above scenario. This data structure, which involves 3 arrays (see page 333), is used in the book’s implementation of the famous Dijkstra’s algorithm of graphs we’ll study later (page 655).

*Ref: Set interface Javadoc “Great care must be exercised if mutable objects are used ...”

2/25/2021
## Operations of a Priority Queue: APIs

<table>
<thead>
<tr>
<th>Classic Setup</th>
<th>S&amp;W MinPQ&lt;Key&gt; (p. 309)</th>
<th>JDK PriorityQueue&lt;Key&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert</td>
<td>insert(Key item)</td>
<td>add(Key item), offer(Key item), same</td>
</tr>
<tr>
<td>FindMin</td>
<td>Key min()</td>
<td>Key peek()</td>
</tr>
<tr>
<td>DeleteMin</td>
<td>Key delMin()</td>
<td>Key poll()</td>
</tr>
</tbody>
</table>
Event driven simulations

• Consider a time-line, with points representing events.
• Each event fires off future events, adding points further to the right on the line, sometimes near, sometimes further, past other events.
• How do we keep track of which event is next? Answer: Put the events in a priority queue with time = priority.
• Note that a simulation itself takes hardly any time. It processes events in time order, typically handling a whole day in a second or less.
Event-Driven Simulation - examples

• Bank Teller Problem
  – Customers arrive and wait in line until one of the \( k \) tellers is available.
  – How long on the average a customer has to wait?
  – What % of the time tellers are actually servicing requests?
  – Applications – determine ideal number of tellers; statistics on average waiting time etc.

• Event-driven simulation
  – Start a simulation clock at zero ticks and advance the clock to the next event time at each stage.
  – The event is serviced and statistics are collected
Event-Driven Simulation: Particle Simulation in S&W

- S&W pp. 856-865
- Circular discs/balls glide across flat horizontal surface, collide with each other and the walls (in a rectangle)
- Starting from one point in time and one ball, we figure out when it will hit a wall or another ball, that collision is an event, add it to PQ, etc.
- Example video
- The S&W simulation only calculates the collisions. To make a video like this, we would also have to work out a lot of intermediate points on the straight lines between collisions.
FYI: Event-Driven Simulation: Particle Simulation in S&W, code

Main loop, pg. 864:

- Get an event from the PQ (next event in time)
- Make the particles move based on their velocities for the time interval since the last event processed
- Process the event: change one or two particles’ velocity based on the collision
- Recalculate future events (collisions) for participant(s) in the collision that just got processed.
FYI: Call Bank Simulation, from Weiss

- Setup: Customer service is accessed by dialing one telephone number
- If any one of the operators (customer service reps) in the bank is available, the user is connected.
- If all the operators are busy, the phone will give a busy signal (simple case)
- The variables are:
  - Number of operators in the bank
  - Probability distribution that governs dial-in attempts
  - Probability distribution that governs connect time
  - How long to run the simulation
Types of events in call bank simulation

1. Customer arrival event
Check for an available operator. If none, signal busy. Otherwise, process the customer: compute simulated departure time and add the departure event (hangup) to the set of events waiting to happen.

2. Customer departure event
Gather statistics for the customer. Make the operator available for another call.
Call Simulator

- One call is coming in every minute, that is, each call event generates another call event one minute later.
- The calls each last a random length of time, with average of 5 minutes.
  - Each call generates a hangup event in the future in x minutes, where x is random with average 5 min.
  - It also generate a next call in 1 min
- How do we keep track of which event is next?
  - Answer: use a priority queue with time = priority.
- Code: CallSim.java is available at [Weiss website]
Sample Trace for the Call Bank

1. User 0 dials in at time 0 and connects for 1 minute
2. User 0 hangs up at time 1
3. User 1 dials in at time 1 and connects for 5 minutes
4. User 2 dials in at time 2 and connects for 4 minutes
5. User 3 dials in at time 3 and connects for 11 minutes
6. User 4 dials in at time 4 but gets busy signal
7. User 5 dials in at time 5 but gets busy signal
8. User 6 dials in at time 6 but gets busy signal
9. User 1 hangs up at time 6
10. User 2 hangs up at time 6
11. User 7 dials in at time 7 and connects for 8 minutes
12. User 8 dials in at time 8 and connects for 6 minutes
13. User 9 dials in at time 9 but gets busy signal
14. User 10 dials in at time 10 but gets busy signal
15. User 11 dials in at time 11 but gets busy signal
16. User 12 dials in at time 12 but gets busy signal
17. User 13 dials in at time 13 but gets busy signal
18. User 3 hangs up at time 14
19. User 14 dials in at time 14 and connects for 6 minutes
20. User 8 hangs up at time 14
21. User 15 dials in at time 15 and connects for 3 minutes
22. User 7 hangs up at time 15
23. User 16 dials in at time 16 and connects for 5 minutes
24. User 17 dials in at time 17 but gets busy signal
25. User 15 hangs up at time 18
26. User 18 dials in at time 18 and connects for 7 minutes

One call comes in every minute

All 3 operators are busy

2 operators are now free
Priority queue states

Event time (priority)
Call Duration, here 5, so inserts HANG_UP at t=1+5=6
Find lowest-priority entry for deleteMin (_________)

No. of operators that are free