CS310 – Advanced Algorithms and Data Structures

Class 10 Priority Queues and Event-Driven Simulation
Java Interfaces:

Consider the Bag API, page 121

public class Bag<Item> implements Iterable<Item>
Bag() -- constructor, not part of Java interface
void add(Item item)
boolean isEmpty()
int size()

Here is an Interface with all the methods of the API:
public interface Bag<Item> implements Iterable<Item>
{
    void add(Item item);
    boolean isEmpty();
    int size();
}
Java Interfaces can describe a subset of the methods of a class

public class Bag<Item> implements Iterable<Item>   Bag API, page 121
Bag()            <-- constructor, not part of Java interface
void add(Item item)
boolean isEmpty()
int size()

Interface with some of the methods of the Bag API:
public interface Bag<Item> implements Iterable<Item> {  
void add(Item item);
boolean isEmpty();
}
• Note that a class needs to implement all the methods of the interface to have “implements Bag<Item>” in its class definition, so more classes will qualify if the interface is simpler.
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.
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.
Classic 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
# 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>
import java.util.PriorityQueue;

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

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]
// 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);
}
Priority queue example

What output is expected here?
Student with gpa 2.3
Student with gpa 3.1
Student with gpa 4.0

Note that duplicates are OK here: If we add another 3.1, we just get two 3.1s back:
Student with gpa 2.3
Student with gpa 3.1
Student with gpa 3.1
Student with gpa 3.1
Student with gpa 4.0
Priority queue implementation

- It is possible to implement a PQ with TreeMap.
- We can use the domain set (the map keys) to hold the PQ elements, and the range element (map value) to hold the duplicate count for this key.
- We can supply a Comparator to the TreeMap constructor to order the map by the priority.
- Then insert works as usual, and deletemin finds the first key in keySet via an iterator, then uses get() for its multiplicity, and either decreases the count by one or removes the key if the count reaches 0.
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 simulation itself 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.
Event-Driven Simulation: Particle Simulation in S&W

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.
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. If yes, 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.
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
Event time

Call Duration, here 5, so inserts HANG_UP at t=1+5=6

Find lowest-time entry for deleteMin

Priority queue states

<table>
<thead>
<tr>
<th>Event</th>
<th>Time</th>
<th>Call</th>
<th>Duration, h</th>
</tr>
</thead>
<tbody>
<tr>
<td>HANG_UP</td>
<td>1</td>
<td>User 0, Len 1</td>
<td></td>
</tr>
<tr>
<td>DIAL_IN</td>
<td>1</td>
<td>User 1, Len 5</td>
<td></td>
</tr>
<tr>
<td>DIAL_IN</td>
<td>2</td>
<td>User 2, Len 4</td>
<td></td>
</tr>
<tr>
<td>HANG_UP</td>
<td>3</td>
<td>User 3, Len 11</td>
<td></td>
</tr>
<tr>
<td>DIAL_IN</td>
<td>4</td>
<td>User 4, Len 7</td>
<td></td>
</tr>
<tr>
<td>HANG_UP</td>
<td>5</td>
<td>User 5, Len 7</td>
<td></td>
</tr>
<tr>
<td>HANG_UP</td>
<td>6</td>
<td>User 6, Len 7</td>
<td></td>
</tr>
<tr>
<td>DIAL_IN</td>
<td>7</td>
<td>User 7, Len 8</td>
<td></td>
</tr>
</tbody>
</table>

No. of operators that are free
Call Simulator

- The simulation uses the **Poisson distribution**, to specify the call duration \( x \) :

\[
P(x) = a^x \cdot e^{-a} / x!
\]

Where \( a \) is the average

Here \( a = \lambda \)

From wikipedia
The Event Class

private static class Event implements Comparable<Event>
{
    static final int DIAL_IN = 1;
    static final int HANG_UP = 2;

    public Event() {
        this(0, 0, DIAL_IN);
    }

    public Event(int name, int tm, int type) {
        who = name;
        time = tm;
        what = type;
    }

    public int compareTo(Event rhs) {
        return time - rhs.time;
    }

    int who; // the number of the user
    int time; // when the event will occur
    int what; // DIAL_IN or HANG_UP

    Note compareTo based on time
public class CallSim
{
    private class Event implements Comparable<Event>
    {... }
    public CallSim( int operators, double avgLen, int callIntrvl )
    {... }
    private Random r;
    private PriorityQueue<Event> eventSet;  // Pending events

    // Basic parameters of the simulation
    private int availableOperators;   // Number of avail. operators
    private double avgCallLen;        // Length of a call
    private int freqOfCalls;          // Interval between calls

    // Used by nextCall only
    private int userNum = 0;
    private int nextCallTime = 0;

    private void nextCall( int delta )
    { ... }
    public void runSim( long stoppingTime )
    { ... }
}
/**
 * Constructor.
 * @param operator number of operators.
 * @param avgLen average length of a call.
 * @param callIntrvl the average time between calls.
 */

public CallSim( int operators, double avgLen, int callIntrvl )
{
    eventSet = new PriorityQueue<Event>( );
    availableOperators = operators;
    avgCallLen = avgLen;
    freqOfCalls = callIntrvl;
    r = new Random( );
    nextCall( freqOfCalls ); // Schedule first call
nextCall Method

private int userNum = 0;
private int nextCallTime = 0;

/**
 * Place a new DIAL_IN event into the event queue.
 * Then advance the time when next DIAL_IN event will occur.
 * In practice, we would use a random number to set the time.
 */
private void nextCall( int delta )
{
    Event ev = new Event( userNum++, nextCallTime, Event.DIAL_IN );
    eventSet.insert( ev );
    nextCallTime += delta;
}
runSim method, part 1

```java
public void runSim( long stoppingTime )
{
    Event e = null;
    int howLong;

    while( !eventSet.isEmpty() )
    {
        e = eventSet.remove();

        if( e.time > stoppingTime )
            break;

        if( e.what == Event.HANG_UP ) // HANG_UP
        {
            availableOperators++;  
            System.out.println( "User " + e.who + 
                " hangs up at time " + e.time );
        }
```
else  // DIAL_IN
{
    System.out.print("User "+e.who+
                  " dials in at time "+e.time+
                      ");
    if( availableOperators > 0 )
    {
        availableOperators--;
        howLong = nextPoisson( avgCallLen );
        System.out.println("and connects for "+howLong+" minutes");
        e.time += howLong;
        e.what = Event.HANG_UP;
        eventSet.add( e );
    }
    else
        System.out.println("but gets busy signal");

    nextCall( freqOfCalls );
}
Random Number Generation Based on the Poisson Distribution

Not in JDK. Listed in Fig 9.5 on p. 404. We use $r$.nextDouble() for nextDouble(), where $r$ is a JDK Random.

```java
/**
 * Return an int using a Poisson distribution, and
 * change the internal state.
 * @param expectedValue the mean of the distribution.
 * @return the pseudorandom int.
 */
public int nextPoisson( double expectedValue )
{
    double limit = -expectedValue;
    double product = Math.log( nextDouble() );
    int count;

    for( count = 0; product > limit; count++ )
        product += Math.log( nextDouble() );

    return count;
}
```
Comments

• Poisson distribution describes the probability of events occurring over a period of time, not their duration, but the distribution is good enough.
• In real life the connecting clients don't call every minute – a random process has to be simulated (this could be a Poisson process).
• Average call time in the simulation described in the book – 5.6 minutes (good enough).
• CompareTo in Event class would be better if used Integer.CompareTo on the time field.
• Code using JDK random is avail at our website.