CS310 - Advanced Data Structures and Algorithms

Spring 2014 – Class 10

March 6, 2014

Announcement

- The midterm exam will take place on Thursday, March 27 (that's 3 weeks from today).
- The Tuesday, March 25 class will partly be a review class.
- Weight – 30% of your final grade.
- If you can't make it for a really good (documented) reason contact me ASAP for a make-up exam.
- 5 questions, answer on paper, even if code is required.
- Please write clearly or you will have to read it to me in person during my office hour (I mean it!).

Midterm Exam

- you can bring the book, class notes, hw's, pa's, solutions and anything written or printed you think may help you.
- Do not bring computers, ipods, friends or any other electronic or human means of communications.
- Work should be strictly individual.
- Cell phones must be silent and only used to show the time.

Covered Material

- Anything we learned and was covered in hw1-3 and pa1-2 including but not limited to:
- Chapter 1-4 – pre-requisites.
- Runtime analysis – big-O notation, recursive runtime analysis (T(n)), logarithms, rules of sum and product of operations (chapter 5).
- Collections – Linked/Array lists, Tree/Hash sets, Tree/Hash maps (chapter 6).
- Implementation – Hash tables (chapter 20), set operations.
- Event driven simulation and beyond – not included.

Comparison of Different Set Methods

	Hashing	Bitmaps (IntSet only)	Tree	List
Construct	O(1)	O(N)	O(1)	O(1)
Contains, get, add, remove	O(1)	O(1)	O(log N)	O(N)
Union, intersection, difference	O(N)	O(N)	O(N)	$O(N^2)$
Next, nextKey	O(1)	O(1)	O(log N)	O(1)
Set Min, Max	O(N)	O(N)	O(log N)	O(N)

Limitations, Disadvantages, etc.

- Hashing: Assumes "good" hashing function, and rehashing, so the table can start small. Need to code hashCode for user-defined elements.
- When rehashing occurs, the add that caused it takes much longer than the other adds: the O(1) for add is an average.
- BitMap: Only for Sets of moderate-sized ints.
- "Fast" O(n) because 32 elements are processed together, small constant in T(n) formula.
- Trees: Requires the elements or domain elements have an ordering.Min/Max for Set is very fast, but that speed applies only to the ordering used in building the tree.

BitMap Implementation to More Cases

- another use of computed mappings to 0..n-1
- Since the bitmap gives us the best-looking union, intersection, and difference performance results, we would like to use it over more cases if possible.
- If we have a one-one computed mapping from an element type to the ints 0 to N-1, and its inverse as well, then we can map from e, an element, to its i, an int, use that as a bit number, and determine membership.
- To return an element, we use the inverse mapping to reconstruct e from i.
- Now we have enough info to do a performance analysis of a Set app.

BitMap Implementation to More Cases

- Recall Map of string to object movie (qoh, subs)
- subs = Set of all movies this one can substitute for, assume O(1):

```
for all movie in M.keySet():
    M.get(movie) to get (qoh, subs) :  O(1)
    Set<String> availSubs =
        cover-set INTERSECT avail
    if empty, gapcount++
```

- N movies, O(1) in each subs, INTERSECT with avail: needs O(1) contains, so O(1)
- isEmpty: O(1), so O(1) per movie, O(N) in all.
- So T(N) = O(N), assuming O(1) in cover-sets

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.

Priority Queue in Java

- PriorityQueue$< T >$ exists in the JDK starting 5, as a class, not a separate interface.
- It extends the Queue interface with "element" for get-front-element and "T remove()" for dequeue/deleteMin.
- add() is used for enqueue/insert.
- insert (add), isEmpty, and deleteMin (remove) and findMin (element) are the basic PriorityQueue methods, that are useful in ordinary apps like the pool of incoming support messages.

Priority Queue Example

```
public class PriorityQueueDemo {
 public static <AnyType extends Comparable<?  super AnyType>>
 void dumpPQ( String msg, PriorityQueue<AnyType> pq )
 {
   System.out.println( msg + ":" );
   while( !pq.isEmpty( ) )
     System.out.println( pq.remove( ) );
 }
 // Do some inserts and removes (done in dumpPQ).
 public static void main( String [ ] args )
 {
   PriorityQueue<Integer> minPQ = new PriorityQueue<Integer>( );
   minPQ.add( 4 );
   minPQ.add( 3 );
   minPQ.add( 5 );
   dumpPQ( "minPQ", minPQ );
 }
}
```

Priority Queue Example

What output is expected here?
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

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 its occurrence rate.
- Then insert works like using a map for counting, 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 (Chap. 13.2)

- 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: Simulate using 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 – Example

- 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 is collected

Modem Bank Simulation

- A modem is accessed by dialing one telephone number
- If any one of the modems in the bank is available, the user is connected.
- If all the modems are in use, the phone will give a busy signal (no line or queue).
- The variables are:
  - Number of modems 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 Modem Bank Simulation

1. **Customer arrival event:** Check for an available teller. If none, place the arrival in the line. If yes, process the customer, compute departure time and add the departure event to the set of events waiting to happen.
2. **Customer departure event:** Gather statistics for the customer and check the line to see whether another customer is waiting. If so – process that next customer.

The events are inserted to the queue by the time in which they occur and processes by their time stamp.

Modem 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 Customer departure event 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 Outputs for the Modem Bank

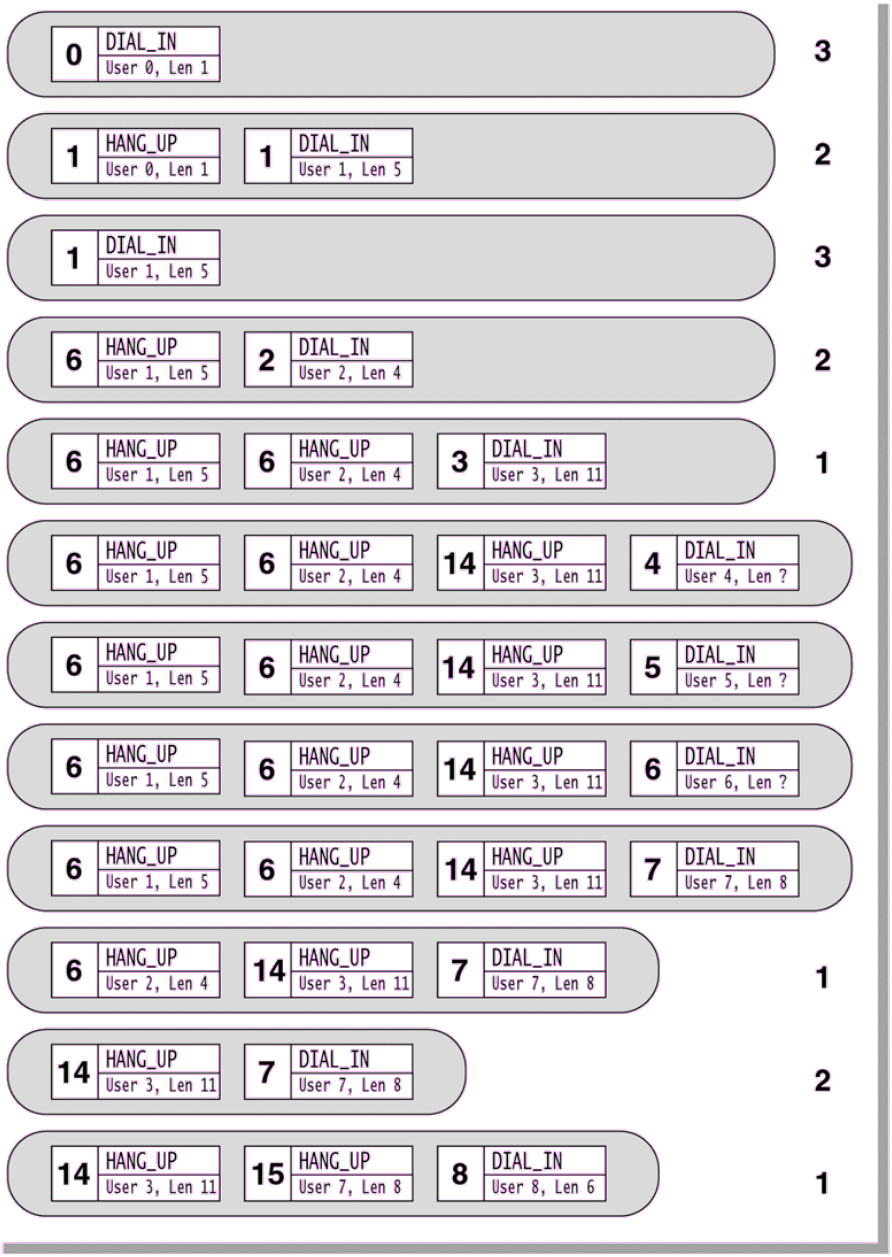
One call comes in every minute

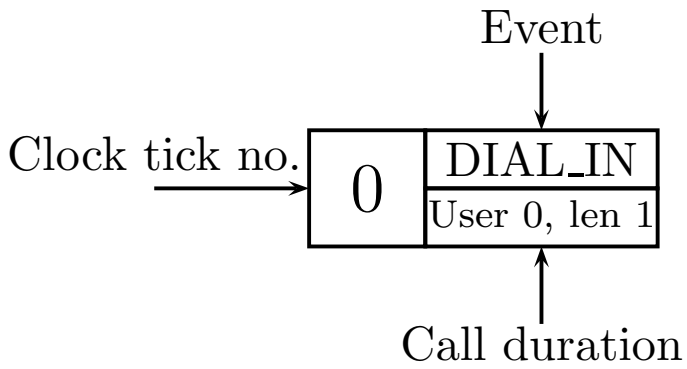
```
1  User 0 dials in at time 0 and connects for 1 minute
2  User 0 hangs up at time 1
3  User 1 dials in at time 1 and connects for 5 minutes
4  User 2 dials in at time 2 and connects for 4 minutes
5  User 3 dials in at time 3 and connects for 11 minutes
6  User 4 dials in at time 4 but gets busy signal
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
```

figure 13.4

Sample output for the modem bank simulation involving three modems: A dial-in is attempted every minute; the average connect time is 5 minutes; and the simulation is run for 18 minutes

Priority Queue Example



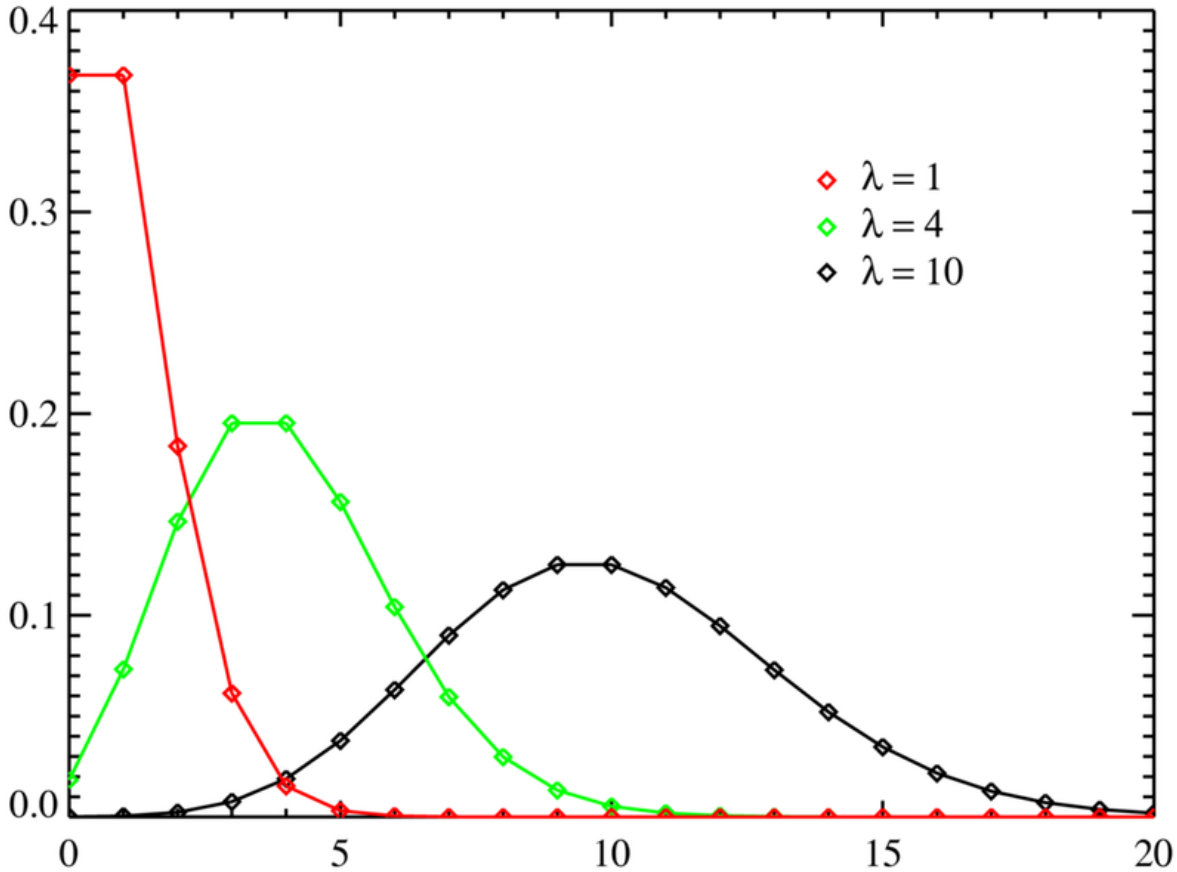


To the right, the number of free modems.

Modem Simulator

The book uses the Poisson distribution, to specify the call duration x :

$P(x) = ax * e(-\lambda)/x!$ Where λ is the average



From Wikipedia

The Event Class

```
 7      private static class Event implements Comparable<Event>
 8      {
 9          static final int DIAL_IN = 1;
10          static final int HANG_UP = 2;
11
12          public Event( )
13          {
14              this( 0, 0, DIAL_IN );
15          }
16
17          public Event( int name, int tm, int type )
18          {
19              who  = name;
20              time = tm;
21              what = type;
22          }
23
24          public int compareTo( Event rhs )
25          {
26              return time - rhs.time;
27          }
28
29          int who;         // the number of the user
30          int time;        // when the event will occur
31          int what;        // DIAL_IN or HANG_UP
32      }
```

ModemSim Class

```
11 // void runSim( )       --> Run a simulation
12
13 public class ModemSim
14 {
15     public ModemSim( int modems, double avgLen, int callIntrvl )
16       { /* Figure 13.7 */ }
17
18       // Run the simulation.
19     public void runSim( long stoppingTime )
20       { /* Figure 13.9 */ }
21
22       // Add a call to eventSet at the current time,
23       // and schedule one for delta in the future.
24     private void nextCall( int delta )
25       { /* Figure 13.8 */ }
26
27     private Random r;                          // A random source
28     private PriorityQueue<Event> eventSet;  // Pending events
29
30         // Basic parameters of the simulation
31     private int freeModems;             // Number of modems unused
32     private double avgCallLen;          // Length of a call
33     private int freqOfCalls;            // Interval between calls
34
35     private static class Event implements Comparable<Event>
36       { /* Figure 13.5 */ }
37 }
```

ModemSim Constructor

figure 13.7

The ModemSim constructor

```
1     /**
2      * Constructor.
3      * @param modem number of modems.
4      * @param avgLen averge length of a call.
5      * @param callIntrvl the average time between calls.
6      */
7     public ModemSim( int modems, double avgLen, int callIntrvl )
8     {
9         eventSet    = new PriorityQueue<Event>( );
10        freeModems  = modems;
11        avgCallLen  = avgLen;
12        freqOfCalls = callIntrvl;
13        r           = new Random( );
14        nextCall( freqOfCalls );  // Schedule first call
15    }
```

nextCall Method

figure 13.8

The `nextCall` method places a new `DIAL_IN` event in the event queue and advances the time when the next `DIAL_IN` event will occur

```
1       private int userNum = 0;
2       private int nextCallTime = 0;
3
4       /**
5        * Place a new DIAL_IN event into the event queue.
6        * Then advance the time when next DIAL_IN event will occur.
7        * In practice, we would use a random number to set the time.
8        */
9       private void nextCall( int delta )
10      {
11          Event ev = new Event( userNum++, nextCallTime, Event.DIAL_IN );
12          eventSet.insert( ev );
13          nextCallTime += delta;
14      }
```

runSim Class

```
5       public void runSim( long stoppingTime )
6       {
7           Event e = null;
8           int howLong;
9
10          while( !eventSet.isEmpty( ) )
11          {
12              e = eventSet.remove( );
13
14              if( e.time > stoppingTime )
15                  break;
16
17              if( e.what == Event.HANG_UP )    // HANG_UP
18              {
19                  freeModems++;
20                  System.out.println( "User " + e.who +
21                                      " hangs up at time " + e.time );
22              }
```

runSim Class (cont.)

```
23              else                            // DIAL_IN
24              {
25                  System.out.print(  "User " + e.who +
26                                      " dials in at time " + e.time + " " );
27                  if( freeModems > 0 )
28                  {
29                      freeModems--;
30                      howLong = r.nextPoisson( avgCallLen );
31                      System.out.println(  "and connects for "
32                                         + howLong + " minutes" );
33                      e.time += howLong;
34                      e.what = Event.HANG_UP;
35                      eventSet.add( e );
36                  }
37                  else
38                      System.out.println( "but gets busy signal" );
39
40                  nextCall( freqOfCalls );
41              }
42          }
43      }
```

Random Number Generation Based on the Poisson Distribution

Not in JDK. Listed in Fig 9.5 on p. 404

figure 9.4

Generation of a random number according to the Poisson distribution

```
1     /**
2      * Return an int using a Poisson distribution, and
3      * change the internal state.
4      * @param expectedValue the mean of the distribution.
5      * @return the pseudorandom int.
6      */
7     public int nextPoisson( double expectedValue )
8     {
9         double limit = -expectedValue;
10        double product = Math.log( nextDouble( ) );
11        int count;
12
13        for( count = 0; product > limit; count++ )
14            product += Math.log( nextDouble( ) );
15
16        return count;
17    }
```

Comments

- Poisson distribution describes the probability of events occurring over a period of time, not their duration,
- but the distribution is good enough (can also use Gaussian with avg. 5, cut at 0).
- 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 describe in the book – 5.6 minutes (good enough).
- CompareTo in Event class would be better if used Integer.CompareTo on the time field.