Due Date: 14th of March

Programming Assignment 6

Introduction

One of the relevant applications of the priority queue ADT is discrete event simulation. The heap data structure is quite useful in implementing a priority queue (we will do it in the next class). In this assignment you will build a simple simulation system and use binary heaps to construct the required priority queue.

Discrete Event Simulations

Assume the scenario of our admin staff working processes where you being the customers submit your jobs to be performed by them. For example, you might be requesting for a duplicate copy of identity card that you have lost recently, or asking for a character certificate – all of which require your jobs to be processed through one or more steps. Of course, not every job will require the same processing.

For simplicity, we assume in this problem that there are a fixed number of admin staff members who process your jobs. For identification, we'll call these staff members as M_1, M_2, …, M_n, and assume that *n* (the number of staff members) will never be larger than *10*.

Your (customer) submitted jobs are characterized by the amount of processing time they require on by each staff member. Each staff member is assumed to be identified by a unique identity number. We'll also assume that the jobs are processed by each member, in staff member id-number order. We will allow the processing time by each member to be as small as zero; this will allow us to specify that some jobs don't require processing by some members. Each job has a submission time – that is, the time when the job was presented to the admin department for processing. Additionally, each job has a priority (an integer in the range 1 to 20, with 1 representing the highest priority). We'll identify our jobs as J_1, J_2, …, J_k, and assume that k (the number of jobs) will never be larger than 2000.

The structure of our simulation system may now be easily formulated. For each member there is an associated priority queue which first orders jobs in priority order. Within each group of jobs having the same priority, jobs are further ordered by the time they entered the queue. As each job is submitted for processing, it is placed in the queue for M_1 (even if it requires no processing by M_1). When M_1 becomes idle (which is its initial state), the job at the head of its queue is removed and processed (perhaps for zero units of time) and then placed in the queue for M_2. The processing by M_2 and the other members is handled in a similar manner.

Performing the Simulation

The simulation of admin dept begins with reading the number of members and the characteristics of a set of jobs, and setting the simulated clock to zero. Then, the simulation repeatedly selects the next event (or events) that will occur, advances the simulated clock to that time, and repeats until no more events remain in the system. The only events that will be considered (and their associated processing) are:

- the arrival of a new job – the job is placed in the queue for M_1.
- the completion of a job's processing by M_i – the job M_i just completed is placed in the queue for M_{i+1} (if $i < n$), and the job at the head of member M_i's queue (if any) is removed, and processing begins.

Naturally, if a member is sitting idle at the time a job is placed in its (empty) queue, the job is immediately removed from the queue and its processing is begun. This can be considered part of the processing associated with the completion of a job by the preceding member.

We do not allow the processing of a job by a member to be interrupted for any reason. In particular, if a member is processing a job and a higher priority job is placed in its input queue, the current job it is processing is completed before the higher priority job is started.

The Problem

Assuming we now have some number of staff members and a collection of jobs, how long does it take to process the jobs? How long after submitting a job will a customer receive the finished work? What

portion of the time is each member idle? How would overall processing be affected if we allowed job processing by a member to be preempted if a higher priority job arrived? What if jobs requiring no processing by a member didn't have to wait in the input queue? All of these question can be answered with an event simulation, but in this assignment we only want answers to the following questions:

• For each member, what fractional part of the total simulation time (the time from when the first job is submitted for processing until the time when the entire sets of jobs is complete) is the member busy?
• On the average, what fractional part of each job's total time in the system does it spend waiting for processing by each member? For example, if most jobs require a lot of processing by M_2, but very little processing by M_3, then we'd expect that jobs spend more time in M_2's queue than they spend in M_3's queue.

The Data

The input data for this problem is just a sequence of integers. The first integer in the input specifies the value for n, the number of members in the business. Recall that this will be no larger than 10 (but will certainly be no smaller than 1).

The next integer in the input specifies the total number of jobs to be processed, *k*. As noted earlier, this will be no larger than *2000*.

Finally there will appear k sets of integers, each of which contains *2+n* values. The first of these *2+n* values is the job's arrival time – that is, the time when it was submitted for processing. The next value is the job's priority (in the range *1* to *20*). The remaining *n* values specify the processing time required by each member, in member-id order.

A simple example will clarify this. Consider the following input:

2 3
4 3 5 0
7 2 0 12
5 3 4 8

This input indicates that there are 2 members and 3 jobs. The first job arrives at time 4, has priority 3, requires 5 time units of processing on M_1 and no processing by M_2. The second job arrives at time 7, has priority 2, requires no processing by M_1 and 12 time units of processing by M_2. The third arrives at time 5, has priority 3, and requires 4 time units of processing by M_1 and 8 time units of processing by M_2.

The Output

The output for this assignment can be presented simply, as two columns each containing n numbers (one for each member). Label the first column with the heading "Percent Idle" and the second column with the heading "Wait Percent". The numbers in the each column are to be displayed as percentages with two fractional digits (that is, between 0.00 and 100.00).

Solution Approaches

There are two approaches that you may use in completing this assignment.

1. The first approach requires the use of n+2 priority queues, one for each member, one for the input jobs, and one for job completions by members. The input job queue is ordered in increasing order of arrival time. The queue for each member holds jobs that are waiting for processing; these jobs are ordered first on priority, and then on arrival time in the queue. That is, if there are two jobs with priority 4 in the waiting queue for a member, the one that arrived in the queue first should be processed before the other. The final queue holds entries that are ordered by the completion times by individual members. (When a member begins processing a job, an entry is made in this queue corresponding to the time when its processing will be complete). Determining the next event to process will require that your program examine each of the queues to determine the event with the earliest time.
2. The second approach is more unified, and may prove more manageable. A single queue is maintained, with "event time" and "priority" as the ordering factors. Each entry identifies the event that will occur, the time the event will occur, and the job and member involved. Event types might

include job arrival, and completion of processing by a member. Consider the processing your program might effect. Initially, all the jobs are placed in the queue with their arrival time as the event time (job priority has no effect on when jobs arrive, except among multiple jobs arriving at the same time). Each member has a status variable that indicates if it is busy or idle, and these are all initially set to the idle state. Then, repeatedly, your program will remove the first event from the queue, advance the simulation clock, and perform the processing associated with the event, repeating this cycle until the event queue is empty. For example, when the first event (obviously a job arrival) is removed from the queue, we begin its processing by member M_1 (since it is idle) and add another event to the queue for the time when M_1 will complete its processing of the job. When a completion event occurs, a new event is added to the queue for processing of the job by the next member (if any).

Reference: Assignment adapted from the Data Structure course at UNIVERSITY OF NEBRASKA, OMAHA