Modeling and Sim Hw2

Toby Harvey

May 1, 2020

Notes about program:

The program can be run with: Python3 tandum_queue_waits.py
The only depedence it has is numpy to generate exponential and uniform samples.
numpy can be installed with the python3 package manager pip3 (assuming you are familiar with it).

(1.1) output can be found at the bottom or in tandum_queue.out

(1.2) The event list I believe can be implemented as either a length 2 or a length 3 array for the case with travel times. We can treat the travel times as seperate events and keep track of them and write a whole function for an travel event. I elected not to this as it seems as though we can get by with only a lenght 2 list. Instead of designating a whole new event, one can just schedule an arrival at the next queue with an offset of time drawn from the uniform(0,2), and then increment the number of people in transit. Similarly we will have to deciment the number of people in transit when an arrivel happens. See lines 112 and 155-160 in code. This is because travel is completely indepedent of any other customers, and therefore arrivals at the next queue can be schedules immediately.

(2.2) and (2.3) I just used a sorted list to implement the event list. This has the advantage that when we need to get the next event it is immediatly avaliable to us at the first index of the list. The disadvatage is that when we recieve an event we may have to look all the way through the list to find this events place (i.e. we must iterate through the list untill the time of the event we are trying to add is less than the events already contained in the list). This means that getting is $O(1)$, and adding is $O(n)$ where n is the length of the list.

If we know that we will only recieve a relatively small number of customers $O(n)$ is not as far away from $O(\log n)$ which we could get from a tree implementation. So with relatively large arrival times compared to the total time, a list implementation may not be so bad. Alternatively if we knew that arrival times were getting more frequent over our simulation, then the times going into the event list would be closer to the front, meaning we would not have to iterate as far into the list to place them. It is a bit of an odd situation, but a list implementation may prove to be not so bad in this case too.

(2.3) Splay trees are a type of binary tree, but balances its self through “splaying”, which also keeps the most searched for elements at the top. When an element is accessed it is moved to the root of the tree through a series of zig-zig and zig-zags, in which the node is moved up the tree and its children and parents are rotated into their correct places. Splay trees have $O(\log n)$ insert and search times, because we only need to consider where the node lies at each level of the tree, and each level of the tree has $(2)^l$ where l is the level number nodes on it. In a priority queue we are removing nodes with highest priority, since splay trees do a splay operation on the parent of the child that is removed, this means that when the parent is removed it will be at the top of the tree. I believe this makes the average empirical deletion time quite low.