The World Evolves, Priorities Change, and So Do You[1]

[1] Marilu Henner

Robin Dawes

November 24, 2021

Can you prevent low-priority tasks from being trapped forever at the back of a Priority Queue?

.

Never Getting There

There is a potential problem with priority queues: low priority items may never reach the front of the queue. That is not necessarily a bad thing if we are modeling a system where the goal is to always be working on the most valuable or important task. For example, this may be true for those engaged in creative arts - such a person might have a number of projects to choose from but no necessity of completing all of them. On the other hand, if the priority queue is being used to determine the order in which airplanes should land at a busy airport, it is essential that all the planes be permitted to land eventually.

Fortunately there is a simple solution: we can periodically increase the priority of items already in the priority queue. In this assignment you will experiment with this idea.

Setting the Scene

Suppose we have a set of items with four priority levels. Let the set of priorities be $\{10, 20, 30, 40\}$, with 10 being the lowest priority and 40 being the highest.

The items are held in a priority queue. When a new item is created, its priority is selected according to this distribution:

$$\begin{aligned} \text{probability}(\text{priority} = 10) &= 0.4 \\ \text{probability}(\text{priority} = 20) &= 0.3 \\ \text{probability}(\text{priority} = 30) &= 0.2 \\ \text{probability}(\text{priority} = 40) &= 0.1 \end{aligned}$$

and then the new item is added to the priority queue.

Since most items will have priority > 10, the probability is very high that there will always be at least one item with priority > 10 in the queue. Thus an item with priority 10 will very probably never reach the front of the queue.

We address this problem by increasing the priority of all items already in the queue before adding a new item.

To simplify the situation we will assume that the priority queue has a constant size of 20: when the item with highest priority is removed from the priority queue, the sequence of actions is this:

1. remove the item with highest priority
2. increment the priority of all items still in the priority queue
3. create a new item (priority 10, 20, 30 or 40) and add it to the priority queue

If we increment the priority of all items in the queue by a constant amount, there are two benefits:

- items maintain their position in the queue - we don't need to move them around
- items with original priority 10 are guaranteed to eventually have priority > 40, so they will reach the head of the queue in advance of all new items added after that point

So What is Left to Explore?

We need to consider the size of the increment, and how it affects the behaviour of the system.

Example: Suppose we set our increment to be 100. After the first removal, all items still in the queue will have their priorities incremented to 110, 120, 130, or 140. Any new arrival will have initial priority <= 40, and *it will never catch up*. Thus all the items in the queue prior to the first arrival will be removed before any items that arrive after that point. In fact, after the items that were in the queue prior to the first arrival are all removed, the queue behaves just like a simple first-in-first-out queue: the increment is so large that later arrivals can never reach the head before earlier arrivals.

This is great news for the priority 10 items - their low priority does not affect their time in the queue.

Example: On the other hand, suppose we set our increment to be 0.1. After the first removal, all items still in the queue will have their priorities incremented to 10.1, 20.1, 30.1, or 40.1. After 100 more removals, the items with original priority = 10 will have priority = 20.1 ... so they will beat new arrivals with priority 20. After 200 more removals, they will finally have priority = 40.1 and will reach the head of the queue before any new arrivals after that point. Any item arriving with initial priority 40 will take precedence over all existing items with initial priority $k < 40$ unless they have been in the queue for $> 10 * (40 - k)$ iterations. This makes the queue behave a lot like a pure priority queue: initial priority values are much more important than arrival order.

This is great for the priority 40 items ... but the priority 10 items may have to wait a long, long time before reaching the head of the queue (even though they **are** guaranteed to get there eventually).

Our goal is to find a reasonable balance between these extremes. One way to quantify this is to compare the "wait time" for items arriving with initial priority = 40.

Consider the situation without any increments to the priorities. Since the probability of having priority 40 is 0.1, the probability of having more than one priority 40 item in the queue is low (remember, if the queue contains any priority 40 items, one of them is immediately removed). This means that when a priority 40 item arrives, it usually goes straight to the head of the queue. We can call this situation "wait

time = 0" - the item arrives and is immediately removed.

With increment = 100, we saw that the queue eventually becomes just first-in-first-out, so a new arrival with priority 40 will have to wait for 20 other items to be removed before it reaches the head of the queue ... so this is "wait time = 20"

With increment = 0.1 ... it's harder to analyze. But experimental data can be used to estimate the expected wait time.

I haven't done this experiment!

Let's set our goal to be "expected wait time for priority 40 items is ≤ 5". In other words, priority 40 items should, on average, have ≤ 5 items leave the queue before they do.

And the question, finally, is :

What is the largest increment value such that expected wait time for priority 40 items is ≤ 5?

An approximate answer is satisfactory.

What Do I Have to Do?

Part 1

Implement a priority queue using a Max-Heap.

Part 2

Instantiate the system described above:

- 20 items in queue
- initial priorities in $\{10, 20, 30, 40\}$)
- priorities incremented after each removal
- new item added after priorities are incremented

Part 3

Conduct experiments to answer the question posed above.

In each experiment you should:

- choose an increment value
- populate the priority queue with 20 items
- iterate through the "remove item, increment priorities, add new item" sequence many times
- determine the average wait time for priority 40 items

A practical question: how do you determine the wait time for a priority 40 item?

A practical answer: when you remove a priority 40 item, you can determine how long it spent in the queue by looking at its final priority. If your increment value is 7 and the item's final priority is 96, then it was incremented $\frac{96-40}{7} = 8$ times.

What Do I Submit?

Please submit your source code for the all parts of the assignment, a summary of your experimental results, and your answer to the question.

Due Date

The assignment is due at 11:59 PM on 20211206. Note that this is not the originally planned due date.