Balancer — Analysis

Jared Dyreson

January 10, 2022

Contents

1 Introduction **2**

2 Use Cases **3**

3 Events **3**

4 Algorithm and Mathematical Analysis **4**
4.1 Reordering Algorithm — make_heap 4
4.1.1 Pseudocode 4
4.2 Mathematical Analysis 4

Abstract

Current implementations of priority queues in the C++ Standard Template Library do not allow for on demand reordering. Container objects are protected member attributes and cannot explicitly be modified. This is a safety mechanism to not allow the user of the data structure to unintentionally break it. However, since the goal is to bypass this, Balancer is a class that inherits from the base priority queue structure and provides this interface. To still retain the original safety feature contained in the STL as only classes that are friends can request the data structure to reorder. This also ensures that the structure cannot erratically change its state during the runtime of the program.

1 Introduction

This solution was provided by here by a user that suggested to inherit from the base std::priority_queue data structure.

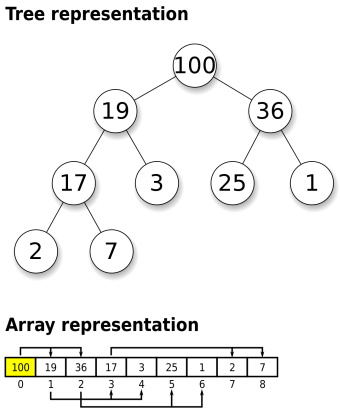

Figure 1: Example of Priority Heap

Priority queues with a special comparator function can influence how it's structured. These functions can be customised to specific objects that are stored in the container. In most use cases, node objects should have a time stamp object that indicates how old it is. Older objects are generally assigned higher priority to avoid starvation.

2 Use Cases

Various types of programs can benefit from a reordering on demand paradigm. CPU schedulers, fast food preparation and reservation systems could see an efficiency increase by using such a data structure. All of these applications have some aspect of time that renders the tree stale.

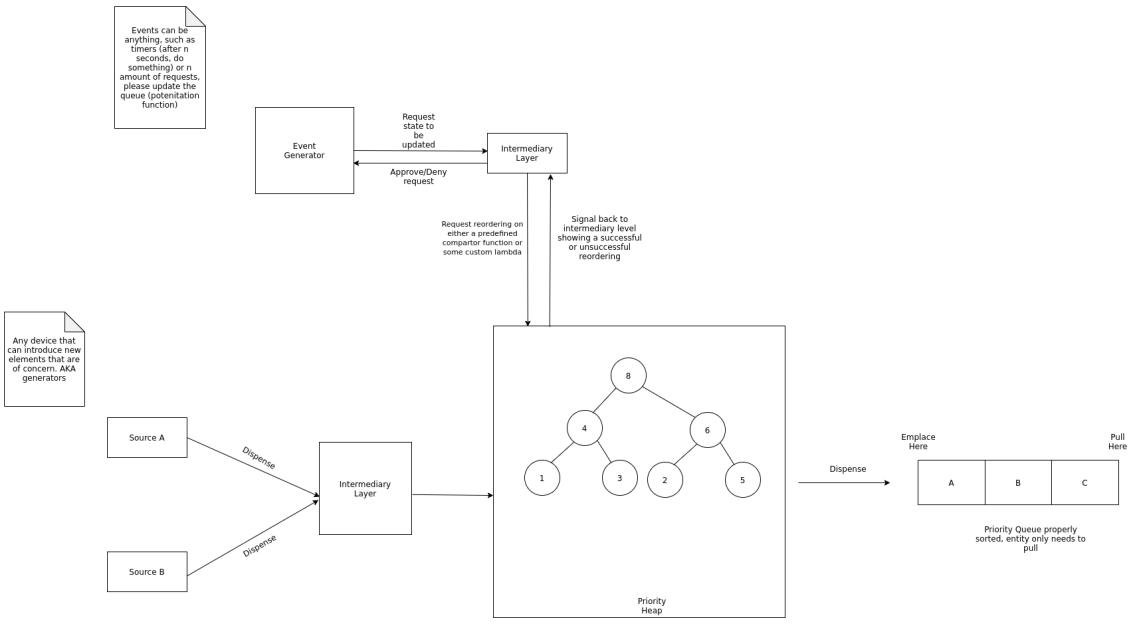

Figure 2: Example system

This sketch shows how the newly designed priority heap in action with respect to generalized inputs and outputs. An arbitrary number of requests are spawned, which are then fed into another intermediary level that will funnel into the queue. Upon insertion, the queue will reorder itself based on it's custom comparator function. This still retains functionality from the original priority queue in STL, however also checks for the element's presence. Some implementations can use a std::vector, however this checking process is $O(n)$ time, as all the elements would need to be evaluated. Please refer to the mathematical and algorithm analysis sections further down to get a more detailed explanation.

3 Events

Events are the only means of updating the state of the structure and are strictly outside influences to the system.

4 Algorithm and Mathematical Analysis

4.1 Reordering Algorithm — make_heap

4.1.1 Pseudocode

```
function reorder {
      make_heap(underlying_container.begin,
                underlying_container.end,
                comparator)
}
```

4.2 Mathematical Analysis