Clojure Heap

Grace, Yanting Zhong

Abstract

This project aims to enable the application of heap in Clojure, which can be used to force the order of the result file in Clojask.

Introduction

A heap is a tree-based data structure that satisfies the heap property, which is argued as one maximally efficient implementation of a priority queue. There are two different types of heaps:

- In a min-heap: for any given node C, if P is a parent node of C, then the key of P is less than or equal to the key of C. Mathematically, heap[k] <= heap[2k+1] and heap[k] <= heap[2k+2], for all k, counting from zero.
- In a max-heap: for any given node C, if P is a parent node of C, then the key of P is greater than or equal to the key of C. Mathematically, heap[k] >= heap[2k+1] and heap[k] >= heap[2k+2], for all k, counting from zero.

While a heap is partly ordered, it is a helpful when the root, i.e., the object with the highest (or lowest) priority, is repeated removed from the collection, and it realizes a complexity of logN.

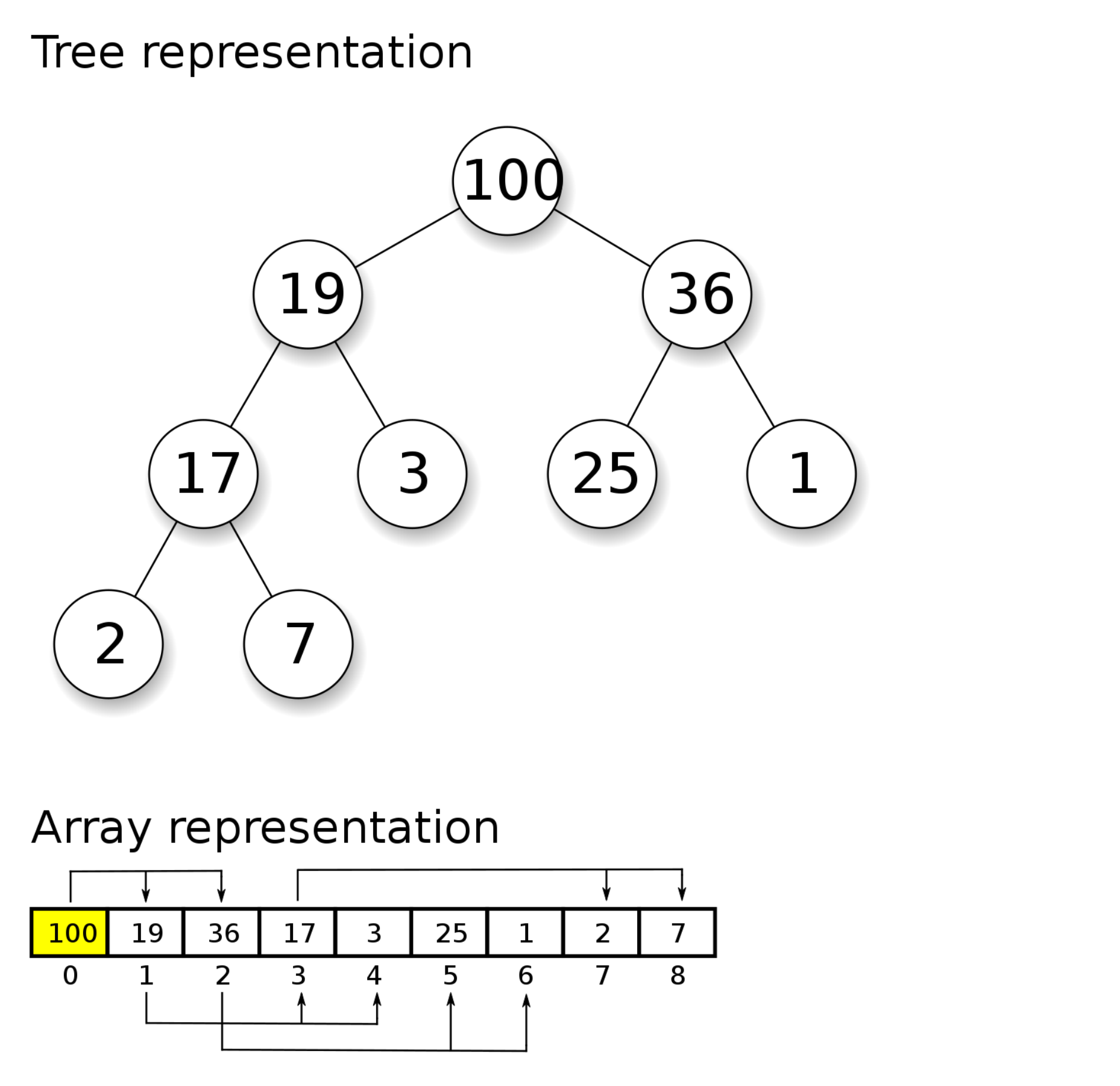


Figure 1. Examples of tree representation and array representation

Methodology

The project delivers a Clojure wrapper of java.util.PriorityQueue.

Given that the default implementation of Priority Queue in Java is a min-heap, there introduces a comparator to enable the implementation of a max-heap in Clojure.

Usage

Initialization of an empty heap and heapifying an existing collection of integers are both enabled. More detailed API usage descriptions are as follows.

The way to initiate an empty min-heap: `init-empty`

```
user=> (def hmin (init-empty))
#'clojure-heap.core/hmin
user=> hmin
#object[java.util.PriorityQueue 0x62c1a9c2 "[]"]
```

The way to initiate an empty max-heap: `init-empty`

```
user=> (def hmax (init-empty >))
#'clojure-heap.core/hmax
user=> hmax
#object[java.util.PriorityQueue 0x5a34485d "[]"]
```

The way to min-heapify a collection: `init`

```
user=> (def hpmin (init [6 8 7 0 0 1 4 3 2]))
#'clojure-heap.core/hpmin
user=> hpmin
#object[java.util.PriorityQueue 0x2448f167 "[0, 0, 1, 2, 6,
7, 4, 8, 3]"]
```

The way to max-heapify a collection: `init`

```
user=> (def hpmax (init [6 8 7 0 0 1 4 3 2] >))
#'clojure-heap.core/hpmax
user=> hpmax
#object[java.util.PriorityQueue 0x3f84b171 "[8, 6, 7, 3, 0,
1, 4, 0, 2]"]
```

The way to add a new element to the heap: `add`

```
user=> (add hpmin 0)
#object[java.util.PriorityQueue 0x2448f167 "[0, 0, 1, 2, 0,
7, 4, 8, 3, 6]"]
user=> (add hpmax 1)
#object[java.util.PriorityQueue 0x3f84b171 "[8, 6, 7, 3, 1,
1, 4, 0, 2, 0]"]
```

The way to remove an element from the heap: `remove`

```
user=> (remove hpmin 8)
#object[java.util.PriorityQueue 0x2448f167 "[0, 0, 1, 2, 0,
7, 4, 6, 3]"]
user=> (remove hpmax 3)
#object[java.util.PriorityQueue 0x3f84b171 "[8, 6, 7, 2, 1,
1, 4, 0, 0]"]
```

The way to return the root of the heap without changing the heap: `pop`

```
user=> (pop hpmin)
0
user=> hpmin
#object[java.util.PriorityQueue 0x2448f167 "[0, 0, 1, 2, 0,
7, 4, 6, 3]"]
user=> (pop hpmax)
8
user=> hpmax
#object[java.util.PriorityQueue 0x3f84b171 "[8, 6, 7, 2, 1,
1, 4, 0, 0]"]
```

The way to return the root and take it out of an existing heap: `poll`

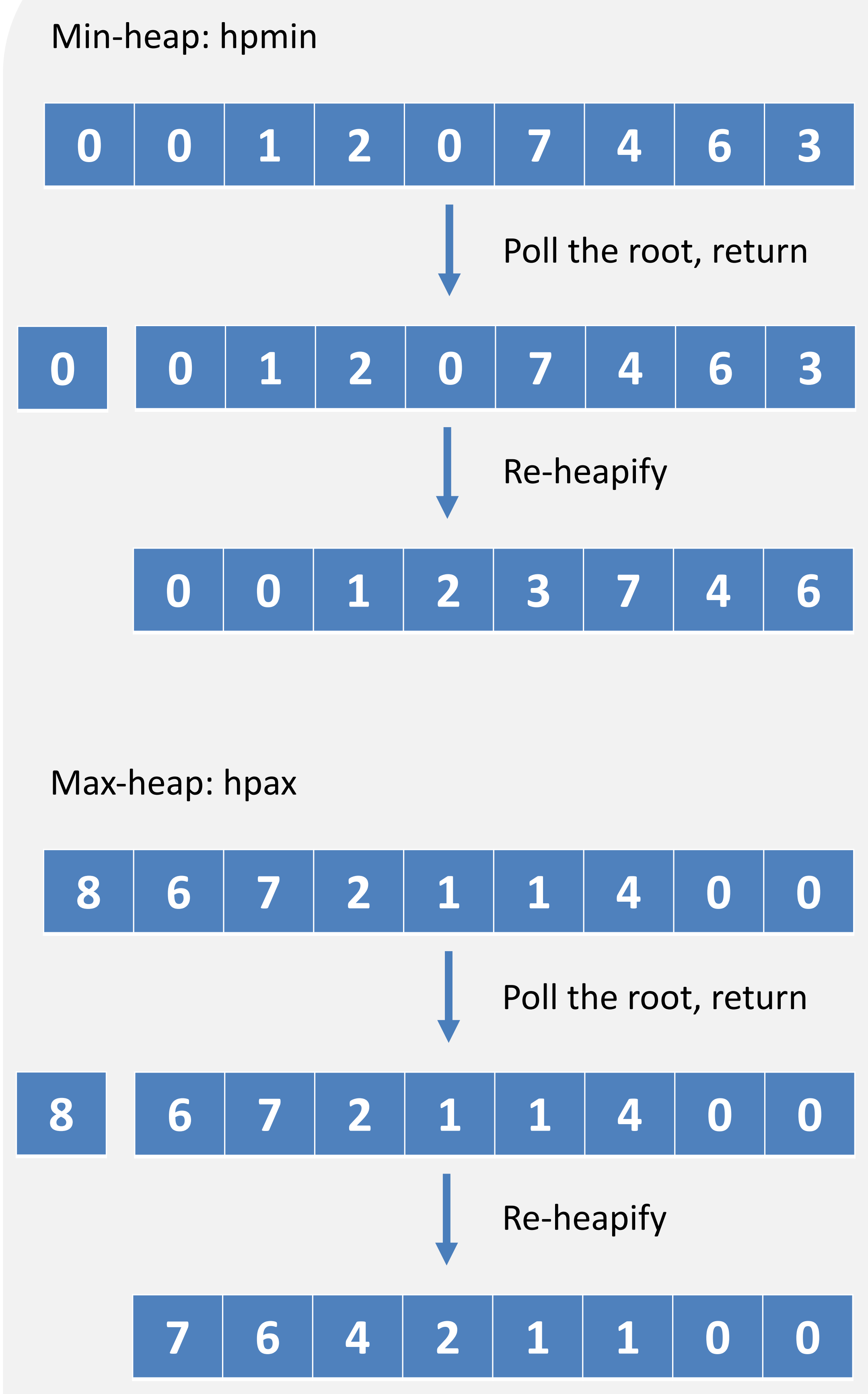
```
user=> (poll hpmin)
0
user=> hpmin
#object[java.util.PriorityQueue 0x2448f167 "[0, 0, 1, 2, 3,
7, 4, 6]"]
user=> (poll hpmax)
8
user=> hpmax
#object[java.util.PriorityQueue 0x3f84b171 "[7, 6, 4, 2, 1,
1, 0, 0]"]
```

The way to check if the heap contains an element: `contains`

```
user=> (contains hpmin 0)
true
user=> (contains hpmax 5)
false
```

The way to get the size of the heap: `size`

```
user=> (size hpmin)
8
user=> (size hpmax)
8
```



Visualization 1. Poll operation on min- and max- heap

Limitation

While the project is focused on a Clojure wrapper of java.util.PriorityQueue, a native implementation of Clojure may also be worth exploration. In this case, a verification test on logN complexity may be necessary.

Also, the inclusiveness of multi data types inside the working collection, e.g., maps, may be another direction that requires further work. The expansion could be helpful in the application in Clojask.

Conclusion

The project realizes the implementation of min- and max- heap in Clojure through an empty or existing collection and a comparator. Fundamental operations including add, remove, pop, poll, contains, and size are all enabled in this framework.

Contact

Grace, Yanting Zhong
University of Hong Kong
Email: gracez@connect.hku.hk
GitHub: https://github.com/clojure-finance/clojure-heap2

References

1. Black (ed.), Paul E. (2004, December 14). Entry for heap in Dictionary of Algorithms and Data Structures. Online version. U.S. National Institute of Standards and Technology, 14 December 2004. Retrieved March 23, 2022, from https://xlinux.nist.gov/dads/HTML/heap.html
2. PriorityQueue (Java Platform SE 7). Retrieved March 23, 2022, from https://docs.oracle.com/javase/7/docs/api/java/util/PriorityQueue.html
3. The Python Standard Library, 8.4. heapq — Heap queue algorithm. Retrieved March 23, 2022, from https://docs.python.org/2/library/heapq.html#heapq
4. Wikimedia Foundation. (2022, March 11). Heap (data structure). Wikipedia. Retrieved March 23, 2022, from https://en.wikipedia.org/wiki/Heap_(data_structure)#cite_note-2