Day 8: Binary Heaps & Disjoint-Set

Tinghai Zhang

January 8, 2025

Contents

1 Binary Heaps

1.1 Priority Queue ADT

Requirements

There are some items with ***priority***, and we need an ADT which supports:

- The next item to access or remove is the ***highest-priority*** item.
- New items may be added ***any time***.

One of common use cases: hospital emergency department.

Two Basic Implementation

- (Unsorted) array
  - Enqueue: add new item at the end of the array, $\mathcal{O}(1)$.
  - Dequeue: scan the array to find the highest-priority item, $\mathcal{O}(n)$.
- Sorted array
  - Enqueue: scan the array to find the right position for the new item, $\mathcal{O}(n)$.
  - Dequeue: remove the last item, $\mathcal{O}(1)$.

Entirely unsorted is too chaotic, but entirely sorted is unnecessary. A compromise is to use a ***heap***.

1.2 Binary Heaps

Binary heaps store items ***partially sorted***. All the items are stored in a ***binary tree***, which satisfies:

- The tree is ***complete***, i.e. nodes in it are filled left-to-right on each level (row) of the tree.
  - The tree is complete if and only if the array representation of the tree is filled from index 1 to n.
- The tree is ***heap-ordered***, i.e. the value of each node is ***greater than or equal to*** the values of its children. We call the property ***max-heap*** property. The ***min-heap*** property is defined similarly.

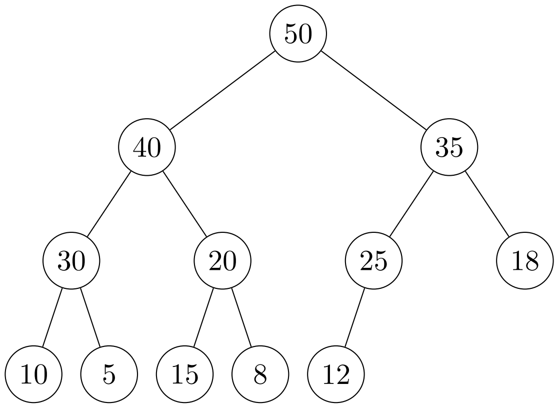


Figure 1 A max-heap binary tree

1.3 Implementation

Find Parent and Child Nodes

The algorithm to find the parent and child nodes of a node at index i in the array representation of a binary heap is shown in Algorithm 1.

Algorithm 1: Find parent and child nodes

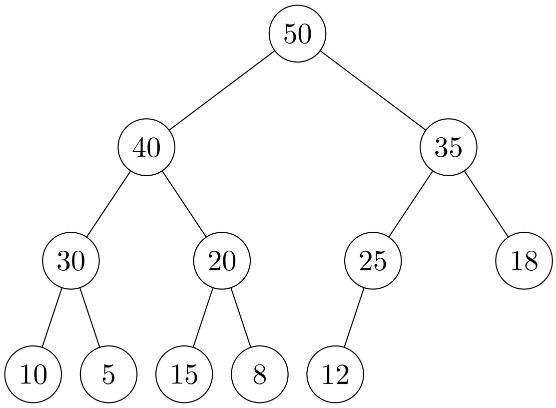
Function PARENT(i)
  return $\lfloor i/2 \rfloor$
end

Function LEFT-CHILD(i)
  return $2i$
end

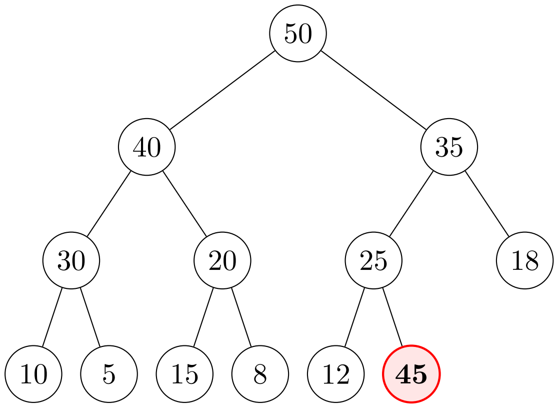
Function RIGHT-CHILD(i)
  return $2i + 1$
end

Insert a New Item

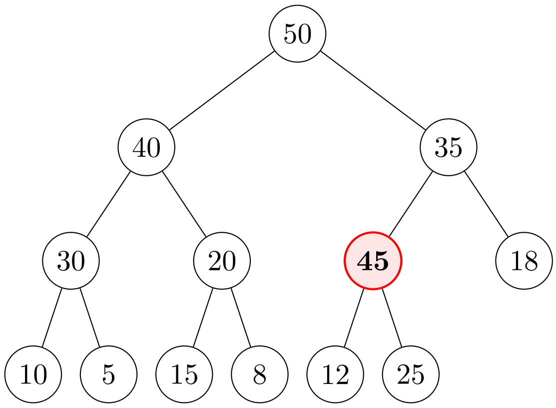
When inserting a new item into a max-heap, we add it to the end of the array and then ***float*** it up to keep the max-heap property. An example to insert a new item is shown in Figure 2.



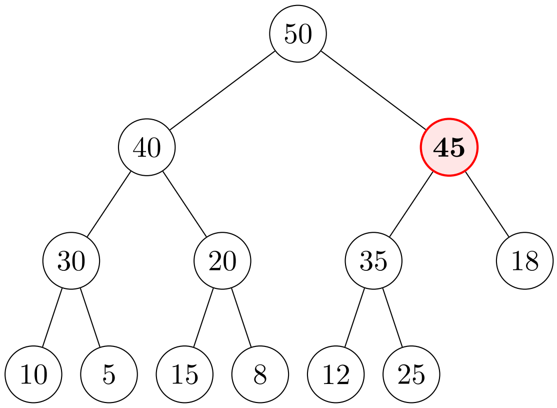
(a) The original max-heap binary tree



(b) Insert the new item 45 at the end of the array



(c) Swap 45 with its parent 25



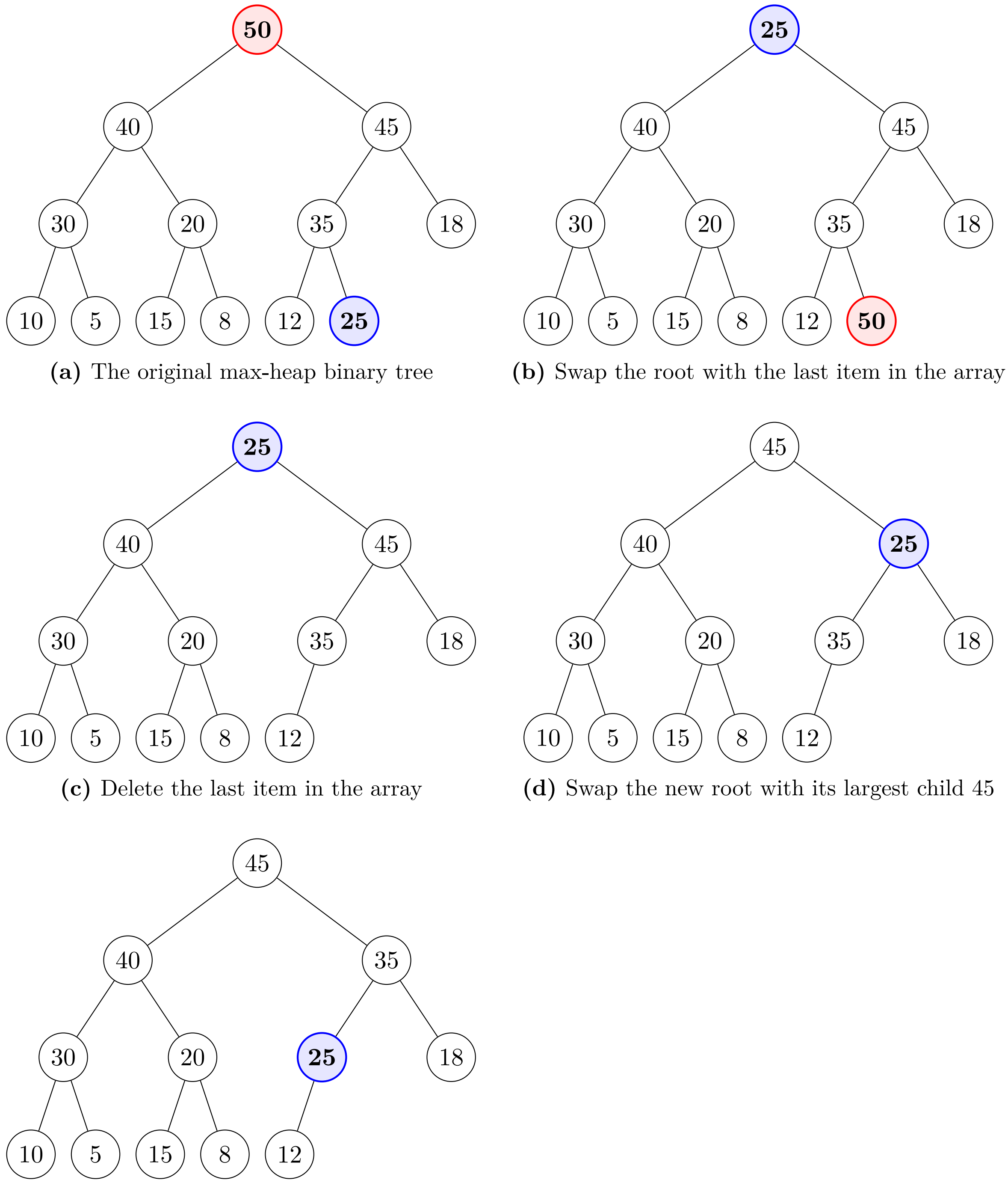
(d) Swap 45 with its parent 35

Figure 2 An example of inserting a new item into a max-heap binary tree

The algorithm to insert a new item into a max-heap is shown in Algorithm 2.

1.4 Delete the Highest-Priority Item

When deleting the highest-priority item from a max-heap, we first swap the root with the last item in the array, then ***sink*** the new root down to keep the max-heap property. An example to delete the highest-priority item is shown in Figure 3.



(a) The original max-heap binary tree

(b) Swap the root with the last item in the array

(c) Delete the last item in the array

(d) Swap the new root with its largest child 45

(e) Swap 25 with its largest child 35

Figure 3 An example of deleting the highest-priority item from a max-heap binary tree

Algorithm 2: Add a new item to a max-heap

```
Function Insert(A, x)
    A.Push-back(x)
    A.size ← A.size + 1
    Float-Up(A.size)
end
Function Float-Up(i)
    while i > 1 and A[i] > A[Parent(i)] do
        Swap A[i] and A[Parent(i)]
        i ← Parent(i)
    end
end
```