7SENG010W Data Structres & Algorithms

Week 8 Lecture

Heaps – Priority Queues

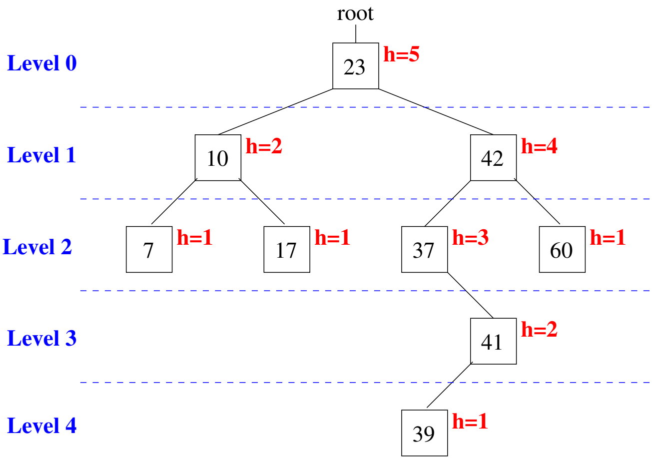
Overview of Lecture 8: Heaps – Priority Queues

- Recap of *Tree Properties*
- *Priority Queues & Heaps*
- *.NET 6 Generic Priority Queue Class*
  - `PriorityQueue<TElement,TPriority>` class

PART I

Recap – Tree Properties

Recap of Binary Tree & Node Properties



A tree and node properties: **nodes levels**, **height** of the tree with it as the root.

PART II

Priority Queues (or Heaps)

Priority Queues

- A *priority queue* is a data structure that stores a collection of *(key, data)* values either in ascending or descending *key* order.
- Its main purpose is to allow the fast & efficient *removal* of the item with the *"highest"* priority in the queue, i.e. the first item in the priority queue.
- That is either the *minimum key*'s data or *maximum key*'s data depending on the queue's *key* ordering.
- The other main operation is the *insertion* of *(key, data)* values into the priority queue into its appropriate position in the queue, depending on:
  - its *key* value,
  - the queue's *key* ordering, i.e. ascending or descending.

Example of a Priority Queue: Planets (1/2)

As an example consider a *priority queue* of *planets*, where the "*priority*" is the average distance of the planet from Earth in *Astronomical Units* (AU)[1].

So here the *key* is the *"planet's distance from Earth"*; the *value* is *"the planet"* & the *key's priority* is in *ascending ordered* using "$\leq$".

(1) Inserting $(4.2,\ Jupiter)$:

$$Planets = \langle\ (4.2,\ Jupiter)\ \rangle$$

(2) Inserting $(0.52,\ Mars)$, priorities: $0.52 \leq 4.2$:

$$Planets = \langle\ (0.52,\ Mars),\ (4.2,\ Jupiter)\ \rangle$$

(3) Inserting $(0.61,\ Mercury)$, priorities: $0.52 \leq 0.61 \leq 4.2$:

$$Planets = \langle\ (0.52,\ Mars),\ (0.61,\ Mercury),\ (4.2,\ Jupiter)\ \rangle$$

(4) Inserting $(29.09,\ Neptune)$, priorities: $4.2 \leq 29.09$:

$$Planets = \langle\ (0.52,\ Mars),\ (0.61,\ Mercury),\ (4.2,\ Jupiter),\ (29.09,\ Neptune)\ \rangle$$

[1] 1 AU is the distance from the Sun to Earth, which is 149,600,000 km.

Example of a Priority Queue: Planets (1/2)

(5) After inserting the remaining planets:

$(8.52,\ Saturn)$, priorities: $4.2 \leq 8.52 \leq 29.09$,
$(18.21,\ Uranus)$, priorities: $8.52 \leq 18.21 \leq 29.09$,
$(0.28,\ Venus)$, priorities: $0.28 \leq 0.52$

$$Planets = \langle\ (0.28,\ Venus),\ (0.52,\ Mars),\ (0.61,\ Mercury),\ (4.2,\ Jupiter),\ (8.52,\ Saturn),\ (18.21,\ Uranus),\ (29.09,\ Neptune)\ \rangle$$

(6) After removing the *lowest priority*, e.g. $(0.28,\ Venus)$:

$$Planets = \langle\ (0.52,\ Mars),\ (0.61,\ Mercury),\ (4.2,\ Jupiter),\ (8.52,\ Saturn),\ (18.21,\ Uranus),\ (29.09,\ Neptune)\ \rangle$$

(7) After removing the *lowest priority*, e.g. $(0.52,\ Mars)$:

$$Planets = \langle\ (0.61,\ Mercury),\ (4.2,\ Jupiter),\ (8.52,\ Saturn),\ (18.21,\ Uranus),\ (29.09,\ Neptune)\ \rangle$$

Implementing Priority Queues – Heaps

The usual approach to implementing a *priority queue* is **not to use an actual queue**, but to use a data structure known as a *Heap*.

A **heap** is a basic data structure and is used to hold *(key, data)* values in either:

- *Ascending* key order, with the *minimum key* value at the front of the queue.
  This is known as a *Min-Heap*.
- *Descending* key order, with *maximum key* value at the front of the queue.
  This is known as a *Max-Heap*.

The *"logical"* view of a heap data structure is that of a *binary tree*, that satisfies two main properties.

However, in reality a heap is rarely implemented as a binary tree, but usually using an *array*.

So in practice a *heap is a linear data structure*.

Heaps – Logical View

A **heap** is a basic data structure and *logically* takes on the form of a *binary tree*, that satisfies two properties on *key ordering* & *tree structure*:

- Tree structure: it is a **complete binary tree**.
  All the levels of the tree are full of nodes, except possibly the highest (final leaf) level, but this must be full from left to right with nodes.

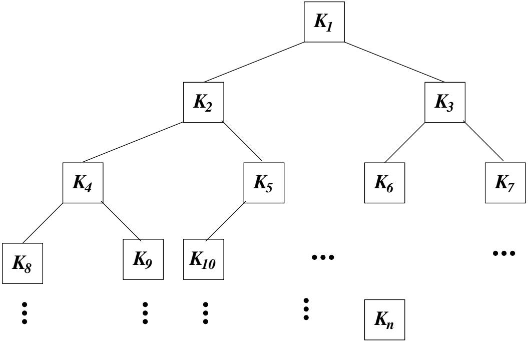
- *Min-Heap* key ordering:
  For all the nodes in the tree, their key values K_p is **less than or equal to** ($\underline{\leq}$) the keys values of its children node's key values K_{lc} and K_{rc}, if they exist. E.g. $K_p \leq K_{lc}$, $K_p \leq K_{rc}$.

- *Max-Heap* key ordering:
  For all the nodes in the tree, their key values K_p is **greater than or equal to** ($\underline{\geq}$) the key values of its children node's key values K_{lc} and K_{rc}, if they exist. E.g. $K_p \geq K_{lc}$, $K_p \geq K_{rc}$.

NOTE: that no direct relationship is defined between a node's left & right children's *key* values, except that they are either both $\leq$ or both $\geq$ that their parent's *key* value, for a Min-heap & Max-heap respectively.

A Max-Heap

The following **heap** with n nodes is **complete** & the node key values: $K_1,\ K_2,\ \ldots,\ K_n$, satisfy the Max-Heap **key ordering** property.



The Max-Heap node **key ordering** property means that the **root's** key K_1 and its **children's** keys $K_2,\ K_3$ satisfy the following:

$$K_1 \ \geq \ K_2, \qquad K_1 \ \geq \ K_3, \qquad [K_2 > K_3,\ K_2 = K_3,\ K_2 < K_3]$$
$$K_2 \ \geq \ K_4, \qquad K_2 \ \geq \ K_5, \qquad [K_4 > K_5,\ K_4 = K_5,\ K_4 < K_5]$$
$$K_3 \ \geq \ K_6, \qquad K_3 \ \geq \ K_7, \qquad [K_6 > K_7,\ K_6 = K_7,\ K_6 < K_7] \qquad \ldots$$