Lecture 10/28/16

Lecturer: Xiaodi Wu

Reading: Chapter 2.4, [CLRS] Chap 6
Heap Example: only keys
Heap Example: only keys

[4, 5, 6, 15, 9, 7, 20, 16, 25, 14, 12, 11, 8]
Insertion

Goals

▶ Maintain three properties of heap.
▶ Cost $\sim$ the height of the heap. i.e., $O(h) = O(\log(n))$. 

External Nodes as "place-holders".

Heap-Order Property: for every node $v$ other than the root, its key $\geq$ the key of its parent.

Complete Binary Trees: binary tree with height $h$ and maximum number of nodes in all levels $0, \ldots, h-1$. In level $h-1$, the internal nodes are to the left of the external nodes.

Last Node: as the rightmost internal node on level $h-1$. 

Insertion

Goals

▶ Maintain three properties of heap.
▶ Cost $\sim$ the height of the heap. i.e., $O(h) = O(\log(n))$.
▶ External Nodes as ”place-holders”.
▶ Heap-Order Property: for every node $v$ other than the root, its key $\geq$ the key of its parent.
▶ Complete Binary Trees: binary tree with height $h$ and maximum number of nodes in all levels $0, \cdots, h - 1$. In level $h - 1$, the internal nodes are to the left of the external nodes.
▶ Last Node: as the rightmost internal node on level $h - 1$. 
Insertion: Implementation

\[\text{insertItem}(k, e)\]

- Insert \((k, e)\) after the last node of the heap.
Insertion: Implementation

\text{insertItem}(k, e)

- Insert \((k, e)\) after the last node of the heap.
- Up-Heap Bubbling on node \((k, e)\).
Insertion: Implementation

\textbf{insertItem}(k, e)

\begin{itemize}
\item Insert \((k, e)\) after the last node of the heap.
\item Up-Heap Bubbling on node \((k, e)\).
\item Up-Heap bubbling on any node \(z\) is as follows. If \(z\) is root, stop. Otherwise, let \(u\) be \(z\)'s parent. If \(\text{key}(z) < \text{key}(u)\), then swap the key-element pair stored in node \(z\), \(u\) and continue up-heap bubbling on \(u\). Otherwise, stop!
\end{itemize}
Insertion: Implementation

insertItem(k, e)

- Insert \((k, e)\) after the last node of the heap.
- Up-Heap Bubbling on node \((k, e)\).
- Up-Heap bubbling on any node \(z\) is as follows. If \(z\) is root, stop. Otherwise, let \(u\) be \(z\)’s parent. If \(\text{key}(z) < \text{key}(u)\), then swap the key-element pair stored in node \(z\), \(u\) and continue up-heap bubbling on \(u\). Otherwise, stop!

Array-based Implementation

- \(O(1)\) for inserting after the last node. Update the last node pointer.
- \(O(h)\) for Up-Heap Bubbling. i.e., \(O(\log n)\).
Heap Example: Insertion with key 2

[4, 5, 6, 15, 9, 7, 20, 16, 25, 14, 12, 11, 8]
Heap Example: Insertion with key 2

[4, 5, 6, 15, 9, 7, 20, 16, 25, 14, 12, 11, 8, 2]
Heap Example: Insertion with key 2

[4, 5, 6, 15, 9, 7, 2, 16, 25, 14, 12, 11, 8, 20]
Heap Example: Insertion with key 2

[4, 5, 2, 15, 9, 7, 6, 16, 25, 14, 12, 11, 8, 20]
Heap Example: Insertion with key 2

[2, 5, 4, 15, 9, 7, 6, 16, 25, 14, 12, 11, 8, 20]
Insertion: Correctness

- Insertion after the last node $\Rightarrow$ internal-only storage and complete binary trees.
Insertion: Correctness

- Insertion after the last node $\Rightarrow$ internal-only storage and complete binary trees.
- Up-Heap Bubble: Heap-Order Property. How to prove?
Insertion: Correctness

- Insertion after the last node $\Rightarrow$ internal-only storage and complete binary trees.
- Up-Heap Bubble: Heap-Order Property. How to prove?
- UP-Heap Bubble can start from anywhere as long as its subtree is a valid heap!
Insertion: Correctness

- Insertion after the last node ⇒ internal-only storage and complete binary trees.
- Up-Heap Bubble: Heap-Order Property. How to prove?
- UP-Heap Bubble can start from anywhere as long as its subtree is a valid heap!

Step-by-Step Snapshots of the Array

- [4, 5, 6, 15, 9, 7, 20, 16, 25, 14, 12, 11, 8]
- [4, 5, 6, 15, 9, 7, 20, 16, 25, 14, 12, 11, 8, 2]
- [4, 5, 6, 15, 9, 7, 2, 16, 25, 14, 12, 11, 8, 20]
- [4, 5, 2, 15, 9, 7, 6, 16, 25, 14, 12, 11, 8, 20]
- [2, 5, 4, 15, 9, 7, 6, 16, 25, 14, 12, 11, 8, 20]
Removal: Implementation

removeMin()

- remove the root of the heap.
Removal: Implementation

removeMin()

▶ remove the root of the heap.
▶ Move the last node to the root.

Array-based Implementation

▶ $O(1)$ for removing the root, and moving the last node to the root.
▶ $O(h)$ for Down-Heap Bubbling. i.e., $O(\log n)$. 
Removal: Implementation

removeMin()

- remove the root of the heap.
- Move the last node to the root.
- Down-Heap Bubbling on node \((k, e)\).
Removal: Implementation

removeMin()

- remove the root of the heap.
- Move the last node to the root.
- Down-Heap Bubbling on node \((k, e)\).
- Down-Heap bubbling on any node \(z\) is as follows. If \(z\) and its children satisfy the Head-Order property, stop. Otherwise, let \(u\) be \(z\)'s child with the smallest key. Swap the key-element pair stored in node \(z\), \(u\) and continue down-heap bubbling on \(u\).
Removal: Implementation

removeMin()

- remove the root of the heap.
- Move the last node to the root.
- Down-Heap Bubbling on node \((k, e)\).
- Down-Heap bubbling on any node \(z\) is as follows. If \(z\) and its children satisfy the Head-Order property, stop. Otherwise, let \(u\) be \(z\)'s child with the smallest key. Swap the key-element pair stored in node \(z\), \(u\) and continue down-heap bubbling on \(u\).

Array-based Implementation

- \(O(1)\) for removing the root, and moving the last node to the root.
- \(O(h)\) for Down-Heap Bubbling. i.e., \(O(\log n)\).
Heap Example: removeMin()

[2, 5, 4, 15, 9, 7, 6, 16, 25, 14, 12, 11, 8, 20]
Heap Example: removeMin()
Heap Example: removeMin()

[4, 5, 20, 15, 9, 7, 6, 16, 25, 14, 12, 11, 8]
Heap Example: removeMin()

[4, 5, 6, 15, 9, 7, 20, 16, 25, 14, 12, 11, 8]
RemoveMin(): Correctness

- Remove the root and move the last node to the root $\Rightarrow$ internal-only storage and complete binary trees.
RemoveMin(): Correctness

- Remove the root and move the last node to the root ⇒ internal-only storage and complete binary trees.
- Down-Heap Bubble: Heap-Order Property. How to prove?
RemoveMin(): Correctness

- Remove the root and move the last node to the root \(\Rightarrow\) internal-only storage and complete binary trees.
- Down-Heap Bubble: Heap-Order Property. How to prove?
- Down-Heap Bubble can start from any node as long as anywhere else except its subtree is a valid heap!
RemoveMin(): Correctness

- Remove the root and move the last node to the root $\Rightarrow$ internal-only storage and complete binary trees.
- Down-Heap Bubble: Heap-Order Property. How to prove?
- Down-Heap Bubble can start from any node as long as anywhere else except its subtree is a valid heap!

Step-by-Step Snapshots of the Array

- $[2, 5, 4, 15, 9, 7, 6, 16, 25, 14, 12, 11, 8, 20]$
- $[20, 5, 4, 15, 9, 7, 6, 16, 25, 14, 12, 11, 8]$
- $[4, 5, 20, 15, 9, 7, 6, 16, 25, 14, 12, 11, 8]$
- $[4, 5, 6, 15, 9, 7, 20, 16, 25, 14, 12, 11, 8]$
Heap: Extension

Locator
Assume an abstract object locator $\ell$ that keeps track of the position of each node in heap as well as the key-element pair stored.
Heap: Extension

Locator
Assume an abstract object locator $\ell$ that keeps track of the position of each node in heap as well as the key-element pair stored.

Removeltem($\ell$)

- How? Can we use RemoveMin()?
Locator
Assume an abstract object `locator ℓ` that keeps track of the position of each node in heap as well as the key-element pair stored.

Removeltem(ℓ)

- How? Can we use RemoveMin()?  
- Remove the node at ℓ and move the last node to ℓ.
Heap: Extension

Locator
Assume an abstract object locator ℓ that keeps track of the position of each node in heap as well as the key-element pair stored.

RemoveItem(ℓ)

- How? Can we use RemoveMin()? 
- Remove the node at ℓ and move the last node to ℓ. 
- Up or Down-Heap Bubble on ℓ?
Heap: Extension

Locator
Assume an abstract object locator \( \ell \) that keeps track of the position of each node in heap as well as the key-element pair stored.

\textbf{Removeltem}(\( \ell \))

- How? Can we use RemoveMin()?
- Remove the node at \( \ell \) and move the last node to \( \ell \).
- Up or Down-Heap Bubble on \( \ell \)?

Deal with Max? Deal with both Max and Min?