Lecture 02/13/17

Lecturer: Xiaodi Wu

Reading: Chapter 6.5, 6.2
Heap Example: only keys

```
[4, 5, 6, 15, 9, 7, 20, 16, 25, 14, 12, 11, 8]
```
Heap Example: only keys

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

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

Insertion
Insertion

Goals

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

- Heap-Order Property:
  - (min-heap) for every node $v$ other than the root, its key $\geq$ the key of its parent.
  - (max-heap) for every node $v$ other than the root, its key $\leq$ the key of its parent.
Insertion

Goals

- Maintain properties of heap.
- Cost \(\sim\) the height of the heap. i.e., \(\Theta(h) = \Theta(\log(n))\).

- Heap-Order Property:
  - (min-heap) for every node \(v\) other than the root, its key \(\geq\) the key of its parent.
  - (max-heap) for every node \(v\) other than the root, its key \(\leq\) 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.
Take Min-Heap as an example

Note: the textbook uses Max-Heap as an example. Our slides complement the story.

Basic Operation: A the array storing the heap

Take Min-Heap as an example

**Note**: the textbook uses Max-Heap as an example. Our slides complement the story.

**Basic Operation**: A the array storing the heap

- Heap-Maximum (A): easy solution?
Take Min-Heap as an example

**Note**: the textbook uses Max-Heap as an example. Our slides complement the story.

**Basic Operation**: A the array storing the heap

- Heap-Maximum (A): easy solution?
- Heap-Increase-Key (A, x, k): increase the element x’s key to the new value k. How to implement?
Take Min-Heap as an example

Note: the textbook uses Max-Heap as an example. Our slides complement the story.

Basic Operation: A the array storing the heap

- Heap-Maximum (A): easy solution?
- Heap-Increase-Key (A, x, k): increase the element x’s key to the new value k. How to implement?
- Heap-Decrease-Key (A, x, k): decrease the element x’s key to the new value k. How to implement?
Decrease-Key and Insertion in the Min-heap

Decrease-Key

- Update the key value of element $x$ to $k$. $O(1)$. 

Min-Heap-Insert:

- Attach a new element to the end of the array with key $+$ $\infty$.
- Decrease-key the new element to the right key $k$ (then up-heap bubbles).
Decrease-Key and Insertion in the Min-heap

Decrease-Key

- Update the key value of element $x$ to $k$. $O(1)$.
- Increase/Decrease of the key might violate the heap-order property. Why?
Decrease-Key and Insertion in the Min-heap

Decrease-Key

- Update the key value of element \( x \) to \( k \). \( O(1) \).
- Increase/Decrease of the key might violate the heap-order property. Why?
- Decrease-Key: Up-Heap Bubbling on the element \( x \).
Decrease-Key and Insertion in the Min-heap

Decrease-Key

- Update the key value of element $x$ to $k$. $O(1)$.
- Increase/Decrease of the key might violate the heap-order property. Why?
- Decrease-Key: Up-Heap Bubbling on the element $x$.
- 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!
Decrease-Key and Insertion in the Min-heap

Decrease-Key

- Update the key value of element $x$ to $k$. $O(1)$.
- Increase/Decrease of the key might violate the heap-order property. Why?
- Decrease-Key: Up-Heap Bubbling on the element $x$.
- Up-Heap bubbling on any node $z$ is as follows. If $z$ is root, stop. Otherwise, let $u$ be $z$’s parent. If $key(z) < key(u)$, then swap the key-element pair stored in node $z$, $u$ and continue up-heap bubbling on $u$. Otherwise, stop!
- $O(h)$ for Up-Heap Bubbling. i.e., $O(\log n)$. (array-based)
Decrease-Key and Insertion in the Min-heap

Decrease-Key

- Update the key value of element \(x\) to \(k\). \(O(1)\).
- Increase/Decrease of the key might violate the heap-order property. Why?
- Decrease-Key: Up-Heap Bubbling on the element \(x\).
- 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!
- \(O(h)\) for Up-Heap Bubbling. i.e., \(O(\log n)\).(array-based)

Min-Heap-Insert: \(O(\log(n))\)

- Attach a new element to the end of the array with key \(+\infty\).
- Decrease-key the new element to the right key \(k\) (then up-heap bubbles).
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 $\Rightarrow$ complete binary trees.
Insertion: Correctness

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

- Insertion after the last node $\Rightarrow$ 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 $\Rightarrow$ 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

- $[4, 5, 6, 15, 9, 7, 20, 16, 25, 14, 12, 11, 8]$
- $[4, 5, 6, 15, 9, 7, 20, 16, 25, 14, 12, 11, 8, 2]$
- $[4, 5, 6, 15, 9, 7, 2, 16, 25, 14, 12, 11, 8, 20]$
- $[4, 5, 2, 15, 9, 7, 6, 16, 25, 14, 12, 11, 8, 20]$
- $[2, 5, 4, 15, 9, 7, 6, 16, 25, 14, 12, 11, 8, 20]$
Increase-Key and Heap-Extract-Min in the Min-heap

Increase-Key

- Update the key value of element $x$ to $k$. $O(1)$.
Increase-Key and Heap-Extract-Min in the Min-heap

**Increase-Key**

- Update the key value of element $x$ to $k$. $O(1)$.
- Increase-Key: Down-Heap Bubbling on the element $x$. or Min-Heapify($x$).
Increase-Key and Heap-Extract-Min in the Min-heap

Increase-Key

- Update the key value of element $x$ to $k$. $O(1)$.
- Increase-Key: Down-Heap Bubbling on the element $x$. or Min-Heapify($x$).

Min-Heapify($z$) on any node $z$ is as follows. If $z$ and its children satisfy the Min-Head 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 Min-Heapify on $u$. 

Heap-Extract-Min: $O(\log(n))$ for Down-Heap Bubbling, i.e., Min-Heapify.
Increase-Key and Heap-Extract-Min in the Min-heap

Increase-Key

- Update the key value of element \( x \) to \( k \). \( O(1) \).
- Increase-Key: Down-Heap Bubbling on the element \( x \). or Min-Heapify(\( x \)).
- Min-Heapify(\( z \)) on any node \( z \) is as follows. If \( z \) and its children satisfy the Min-Head 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 Min-Heapify on \( u \).
- \( O(\log(n)) \) for Down-Heap Bubbling, i.e., Min-Heapify.
Increase-Key and Heap-Extract-Min in the Min-heap

Increase-Key

- Update the key value of element \( x \) to \( k \). \( O(1) \).
- Increase-Key: Down-Heap Bubbling on the element \( x \). or Min-Heapify(\( x \)).
- Min-Heapify(\( z \)) on any node \( z \) is as follows. If \( z \) and its children satisfy the Min-Head 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 Min-Heapify on \( u \).
- \( O(\log(n)) \) for Down-Heap Bubbling, i.e., Min-Heapify.

Heap-Extract-Min: \( O(\log(n)) \)

- Increase-key \( A[1] \) to the right key \( k \) (then Min-Heapify).
Heap Example: Heap-Extract-Min

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

[20, 5, 4, 15, 9, 7, 6, 16, 25, 14, 12, 11, 8]
Heap Example: Heap-Extract-Min

[4, 5, 20, 15, 9, 7, 6, 16, 25, 14, 12, 11, 8]
Heap Example: Heap-Extract-Min

[4, 5, 6, 15, 9, 7, 20, 16, 25, 14, 12, 11, 8]
Heap-Extract-Min: Correctness

- Remove the root and move the last node to the root ⇒ complete binary trees.
Heap-Extract-Min: Correctness

- Remove the root and move the last node to the root ⇒ complete binary trees.
- Down-Heap Bubble: Heap-Order Property. How to prove?
Heap-Extract-Min: Correctness

- Remove the root and move the last node to the root ⇒ 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!
Heap-Extract-Min: Correctness

- Remove the root and move the last node to the root ⇒ 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

- \[ [2, 5, 4, 15, 9, 7, 6, 16, 25, 14, 12, 11, 8, 20] \]
- \[ [20, 5, 4, 15, 9, 7, 6, 16, 25, 14, 12, 11, 8] \]
- \[ [4, 5, 20, 15, 9, 7, 6, 16, 25, 14, 12, 11, 8] \]
- \[ [4, 5, 6, 15, 9, 7, 20, 16, 25, 14, 12, 11, 8] \]
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. **Can Increase and Decrease key at any location.**
Heap: Up/Down-heap bubbling at any location

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{Can Increase and Decrease key at any location.}

RemoveItem($\ell$)

- How? Can we use Heap-Extract-Min?
Heap: Up/Down-heap bubbling at any location

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. Can Increase and Decrease key at any location.

RemoveItem($\ell$)

- How? Can we use Heap-Extract-Min?
- Remove the node at $\ell$ and move the last node to $\ell$. 
Heap: Up/Down-heap bubbling at any location

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. **Can Increase and Decrease key at any location.**

`Removeltem(\ell)`

- How? Can we use Heap-Extract-Min?
- Remove the node at \( \ell \) and move the last node to \( \ell \).
- Up or Down-Heap Bubble on \( \ell \)?
Heap: Up/Down-heap bubbling at any location

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. Can Increase and Decrease key at any location.

Removeltem(\( \ell \))

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

Deal with Max (textbook Chap 6)?
Deal with both Max and Min?