Lecture 10/19/16

Lecturer: Xiaodi Wu

Reading: Chapter 3.1
Binary Search Tree: Insertion

Insert key $k$ into a binary search tree $T$

- First, $w =$ TreeSearch($k$, $T$.root()).
Binary Search Tree: Insertion

Insert key $k$ into a binary search tree $T$

- First, $w = \text{TreeSearch}(k, T.\text{root}())$.
- If $k$ is not in $T$, i.e., $w$ is an external node. We replace $w$ by an internal node storing $(k, e)$ and add two external children to $w$. 

(Typically, $e$ is the element associated with $k$ in $T$.)
Insert key $k$ into a binary search tree $T$

- First, $w = \text{TreeSearch}(k, \text{T.root()})$.
- If $k$ is not in $T$, i.e., $w$ is an external node. We replace $w$ by an internal node storing $(k, e)$ and add two external children to $w$.
- If $k$ is in $T$, i.e., $w$ is an internal node. Call TreeSearch($k$, rightChild($w$)) and apply the above algorithm recursively. (duplicate the key)
Algorithm insertItem\((k, e, v, T)\)
Input: a search key-element \((k, e)\) and a node \(v\) of a binary search tree \(T\).
Output: a updated \(T\).
\(w \leftarrow \text{TreeSearch}(k, v)\)
\textbf{if} \(w\) is external \textbf{then}
    Replace \(w\) by an internal node storing \((k, e)\) with two external children. Return.
\textbf{else}
    insertItem\((k, e, T.\text{rightChild}(w), T)\).
\textbf{end if}
Algorithm insertItem\((k, e, v, T)\)

Input: a search key-element \((k, e)\) and a node \(v\) of a binary search tree \(T\).

Output: a updated \(T\).

\(w \leftarrow TreeSearch(k, v)\)

if \(w\) is external then
  Replace \(w\) by an internal node storing \((k, e)\) with two external children. Return.

else
  insertItem\((k, e, T.rightChild(w), T)\).
end if

- Time: \(O(h)\) could from \(O(\log n)\) to \(O(n)\).
Insertion in binary search trees

**Algorithm** `insertItem(k, e, v, T)`
Input: a search key-element \((k, e)\) and a node \(v\) of a binary search tree \(T\).
Output: a updated \(T\).

\(w \leftarrow \text{TreeSearch}(k, v)\)

**if** \(w\) is external **then**
   Replace \(w\) by an internal node storing \((k, e)\) with two external children. Return.
**else**
   `insertItem(k, e, T.rightChild(w), T)`.
**end if**

- **Time:** \(O(h)\) could from \(O(\log n)\) to \(O(n)\).
- **Correctness:** rely on the correctness of TreeSearch.
Binary Search Trees: insert(30)
Binary Search Trees: insert(30)
Binary Search Trees: insert(29)
Binary Search Trees: insert(29)
Binary Search Trees: insert(29)
Binary Search Tree: Insertion

More Questions

- One can also call TreeSearch(k, leftChild(w)). Why?
More Questions

- One can also call TreeSearch(k, leftChild(w)). Why?
- Alternative way to handle duplication of the key?
More Questions

- One can also call TreeSearch(k, leftChild(w)). Why?
- Alternative way to handle duplication of the key? A counter at each node!
Remove key \( k \) out of a binary search tree \( T \)

- First, \( w = \text{TreeSearch}(k, T.\text{root}()) \).
Binary Search Tree: Removal

Remove key $k$ out of a binary search tree $T$

- First, $w = \text{TreeSearch}(k, T.\text{root}())$.
- If $k$ is not in $T$, i.e., $w$ is an external node. We have nothing to remove. Done!
Remove key \( k \) out of a binary search tree \( T \)

- First, \( w = \text{TreeSearch}(k, T.\text{root}()) \).
- If \( k \) is not in \( T \), i.e., \( w \) is an external node. We have nothing to remove. Done!
- Otherwise, \( w \) is an internal node. We distinguish the following two cases.
  - (1) at least one of the children of \( w \) is an external node.
  - (2) both of the children of \( w \) are internal nodes.
Binary Search Tree: Removal

Case 1

- (1) at least one of the children of \( w \) is an external node.
Case 1

- (1) at least one of the children of \( w \) is an external node.
- Let \( z \) be the external child. Let \( y \) be the other child.
- Remove \( z, w \) and connect \( y \) to \( w \)'s parent replacing \( w \)'s position.

Correctness: maintain the binary search tree property.

Time: \( O(h) \).
Binary Search Tree: Removal

Case 1

- (1) at least one of the children of \( w \) is an external node.
- Let \( z \) be the external child. Let \( y \) be the other child.
- Remove \( z, w \) and connect \( y \) to \( w \)'s parent replacing \( w \)'s position.
- Correctness: maintain the binary search tree property.
Case 1

- (1) at least one of the children of \( w \) is an external node.
- Let \( z \) be the external child. Let \( y \) be the other child.
- Remove \( z, w \) and connect \( y \) to \( w \)'s parent replacing \( w \)'s position.
- Correctness: maintain the binary search tree property.
- Time: \( O(h) \).
Binary Search Trees: Remove(32)
Binary Search Trees: Remove(32)
Binary Search Trees: Remove(32)
Binary Search Tree: Removal

Case 2

- (2) both of the children of $w$ are internal nodes.
Case 2

- (2) both of the children of $w$ are internal nodes.
- Find $y$: the first internal node that follows $w$ in an inorder traversal. How?
Binary Search Tree: Removal

Case 2

- (2) both of the children of $w$ are internal nodes.
- Find $y$: the first internal node that follows $w$ in an inorder traversal. How?
- Such $y$ must have an external left child. Why?
Binary Search Tree: Removal

Case 2

- (2) both of the children of \( w \) are internal nodes.
- Find \( y \): the first internal node that follows \( w \) in an inorder traversal. How?
- Such \( y \) must have an external left child. Why?
- Two Steps: (a) replace \( w \) by \( y \). (b) Remove(\( y \)).
Case 2

- (2) both of the children of \( w \) are internal nodes.
- Find \( y \): the first internal node that follows \( w \) in an inorder traversal. How?
- Such \( y \) must have an external left child. Why?
- Two Steps: (a) replace \( w \) by \( y \). (b) Remove(y).
- Correctness: Step (a)?
Binary Search Tree: Removal

Case 2

- (2) both of the children of $w$ are internal nodes.
- Find $y$: the first internal node that follows $w$ in an inorder traversal. How?
- Such $y$ must have an external left child. Why?
- Two Steps: (a) replace $w$ by $y$. (b) Remove($y$).
- Correctness: Step (a)? by the inorder property.
Case 2

- (2) both of the children of $w$ are internal nodes.
- Find $y$: the first internal node that follows $w$ in an inorder traversal. How?
- Such $y$ must have an external left child. Why?
- Two Steps: (a) replace $w$ by $y$. (b) Remove($y$).
- Correctness: Step (a)? by the inorder property.
- Step (b)?
Case 2

- (2) both of the children of \( w \) are internal nodes.
- Find \( y \): the first internal node that follows \( w \) in an inorder traversal. How?
- Such \( y \) must have an external left child. Why?
- Two Steps: (a) replace \( w \) by \( y \). (b) Remove(\( y \)).
- Correctness: Step (a)? by the inorder property.
- Step (b)? by the analysis in Case 1
Binary Search Tree: Removal

Case 2

- (2) both of the children of \( w \) are internal nodes.
- Find \( y \): the first internal node that follows \( w \) in an inorder traversal. How?
- Such \( y \) must have an external left child. Why?
- Two Steps: (a) replace \( w \) by \( y \). (b) Remove(\( y \)).
- Correctness: Step (a)? by the inorder property.
- Step (b)? by the analysis in Case 1.
- Time: \( O(h) \).
Binary Search Trees: Remove(65)
Binary Search Trees: Remove(65)
Binary Search Trees: Remove(65)
Describe an algorithm that checks whether $T$ is a valid binary search tree. Analyze the worst-case complexity of your algorithm.

Assume $T$ is a binary search tree and let $k$ be another input. Describe an algorithm that finds one of the closest-to-$k$ keys in the binary tree $T$. Analyze the worst-case complexity of your algorithm. (Assume all the keys are integers and the distance between two keys is the absolute value of their difference.)