Lecture 02/24/16

Lecturer: Xiaodi Wu

February 24th, 2016
Removal

Remove key $k$

- Search for that key $k$.
- If no such $k$, nothing to remove.
- Otherwise, arrive at an internal node (two cases): (1) with all external nodes (2) otherwise.
- Claim: can always reduce to case (1). Suppose the key is the $i$th item $k_i$ at a node $z$.
  - Find the right-most internal node $v$ in the subtree rooted at the $i$th child of $z$. Claim: $v$ is case (1). Why?
Removal

Remove key $k$

- Search for that key $k$.
- If no such $k$, nothing to remove.
- Otherwise, arrive at an internal node (two cases): (1) with all external nodes (2) otherwise.
- Claim: can always reduce to case (1). Suppose the key is the $i$th item $k_i$ at a node $z$.
  - Find the right-most internal node $v$ in the subtree rooted at the $i$th child of $z$. Claim: $v$ is case (1). Why?
  - Swap $k_i$ and the last item of $v$. Reduce to case (1).
Removal: Cont’d

Remove key $k$: Case 1

- $k$ is stored at a node $v$ with only external children.
- Remove it. The depth property is preserved.
- However, the size property might be violated (i.e., under-flow)
- A generic way to handle the under-flow needed.
Restoration at Underflow!

Find the immediate siblings of $v$

- Note $v$ should be 2-node before removal.
- If there is an immediate sibling $w$ of $v$ (3-node or 4-node), then perform a transfer operation.
Restoration at Underflow!

Find the immediate siblings of $v$

- Note $v$ should be 2-node before removal.
- If there is an immediate sibling $w$ of $v$ (3-node or 4-node), then perform a **transfer** operation.
- Otherwise, perform a **fusion** operation with an immediate sibling $w$ of $v$. (2-node in this case).
Transfer operation

\( v \) node to remove keys, \( w \) 3-node or 4-node and \( v \)'s immediate sibling, \( u \) 3-node or 4-node and \( u \)'s parent and the key \( k \) that separates \( v, w \).

- Assume \( w \) is after \( v \), similarly for the other case.
- Let \( k_w \) be the first key in \( w \) and \( T_w \) the first subtree of \( w \).
- Move \( k_w \) to \( u \), replacing the position of \( k \). Move \( k \) to \( v \) as the last key.
- Move \( T_w \) to be the last subtree of \( v \).
Transfer operation

\( \nu \) : node to remove keys, \( \omega \) : 3-node or 4-node and \( \nu \) ’s immediate sibling, \( \omega \) : \( \omega \), \( \nu \) ’s parent and the key \( k \) that separates \( \nu \), \( \omega \).

- assume \( \omega \) is after \( \nu \), similarly for the other case.
- Let \( k_\omega \) be the first key in \( \omega \) and \( T_\omega \) the first subtree of \( \omega \).
- Move \( k_\omega \) to \( \omega \), replacing the position of \( k \). Move \( k \) to \( \nu \) as the last key.
- Move \( T_\omega \) to be the last subtree of \( \nu \).

Correctness & Implementation

- **Transfer** preserves the depth-property and the multi-way search tree property.
- Restore the size-property of \( \nu \). Preserve the size-property of the rest nodes.
- Implementation: \( O(1) \).
Fusion operation

$v$: node to remove keys, $w$: 2-node and $v$’s immediate sibling, $u$: $w$, $v$’s parent and the key $k$ that separates $v$, $w$.

- Combine $v$ and $w$ to get a new node $v'$. 

Correctness & Implementation

- Fusion preserves the depth-property and the multi-way search tree property.
- Establish the size-property of $v'$. 
- However, $u$ might violate the size-property. Repeat either the transfer or fusion on $u$ again. At most repeat $O(h) = O(\log(n))$ times.
Fusion operation

\( v \): node to remove keys, \( w \): 2-node and \( v \)’s immediate sibling, \( u \): \( w \), \( v \)’s parent and the key \( k \) that separates \( v \), \( w \).

- Combine \( v \) and \( w \) to get a new node \( v' \).
- Move \( k \) into the new node \( v' \). Keep subtrees from both \( v \), \( w \) and put them in the right order.
Fusion operation

\(v\): node to remove keys, \(w\): 2-node and \(v\)'s immediate sibling, \(u\): \(w\), \(v\)'s parent and the key \(k\) that separates \(v\), \(w\).

▶ Combine \(v\) and \(w\) to get a new node \(v'\).
▶ Move \(k\) into the new node \(v'\). Keep subtrees from both \(v\), \(w\) and put them in the right order.

Correctness & Implementation

▶ **Fusion** preserves the depth-property and the multi-way search tree property.
Fusion operation

$v$: node to remove keys, $w$: 2-node and $v$'s immediate sibling, $u$: $w$, $v$’s parent and the key $k$ that separates $v$, $w$.

- Combine $v$ and $w$ to get a new node $v'$.
- Move $k$ into the new node $v'$. Keep subtrees from both $v$, $w$ and put them in the right order.

Correctness & Implementation

- **Fusion** preserves the depth-property and the multi-way search tree property.
- Establish the size-property of $v'$.
Fusion operation

\(v\): node to remove keys, \(w\): 2-node and \(v\)'s immediate sibling, \(u\): \(w\), \(v\)'s parent and the key \(k\) that separates \(v, w\).

- Combine \(v\) and \(w\) to get a new node \(v'\).
- Move \(k\) into the new node \(v'\). Keep subtrees from both \(v, w\) and put them in the right order.

Correctness & Implementation

- **Fusion** preserves the depth-property and the multi-way search tree property.
- Establish the size-property of \(v'\).
- However, \(u\) might violate the size-property. Repeat either the transfer or fusion on \(u\) again. At most repeat \(O(h) = O(\log(n))\) times.
(2,4) Tree insertion: Remove 12, fusion
(2,4) Tree insertion: Remove 12, fusion
(2,4) Tree insertion: Remove 13, no transfer or fusion
(2,4) Tree insertion: Remove 13, no transfer or fusion
(2,4) Tree insertion: Remove 14, fusion twice
Tree insertion: Remove 14, fusion twice
(2,4) Tree insertion: Remove 14, fusion twice
(2,4) Tree insertion: Remove 14, fusion twice
(2,4) Tree insertion: Remove 14, fusion twice