

ACOL106: Data Structures and Algorithms

Ragesh Jaiswal, IIT Delhi

Data Structures: **Balanced** Binary Search Trees

Data Structures

Balanced Binary Search Trees → AVL Trees

Algorithm

// p denotes the node that is inserted.

BalanceAfterPut(Node p)

- While going up from p , let z denote the first node for which the height balance property is not satisfied.
- Let y be the child of z with greater height.
- Let x be the child of y with greater height.
- Perform a tri-node restructuring w.r.t. nodes x, y, z .

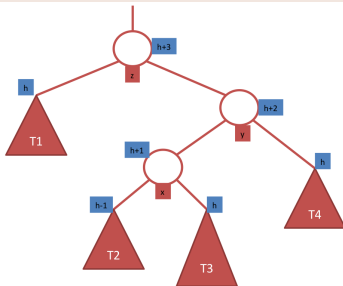


Figure: The height of the nodes after inserting the new node are as shown above.

Data Structures

Balanced Binary Search Trees $\rightarrow$ AVL Trees

Algorithm

// p denotes the node that is inserted.

BalanceAfterPut(Node p)

- While going up from p , let z denote the first node for which the height balance property is not satisfied.
- Let y be the child of z with greater height.
- Let x be the child of y with greater height.
- Perform a tri-node restructuring w.r.t. nodes x, y, z .

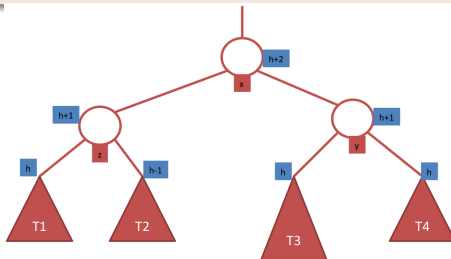
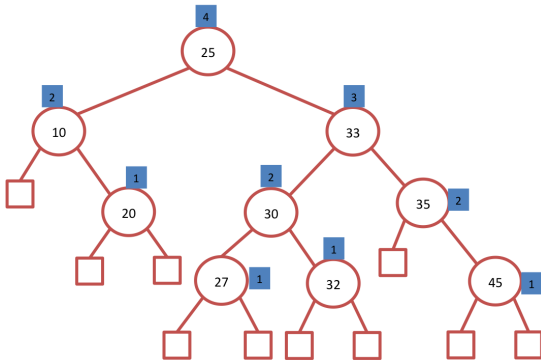


Figure: The height of the nodes after inserting and performing tri-node restructuring.

Data Structures

Balanced Binary Search Trees → AVL Trees

- Question: How do we perform `remove(k)` operation on an AVL tree?
- Same as in BST. However, you also have to make sure that after deletion, the height balance property is maintained.
- Consider deleting the entry with key 20 in the Tree below.



Data Structures

Balanced Binary Search Trees → AVL Trees

- Question: How do we perform `remove(k)` operation on an AVL tree?
- Same as in BST. However, you also have to make sure that after deletion, the height balance property is maintained.
- Consider deleting the entry with key 20 in the Tree below.

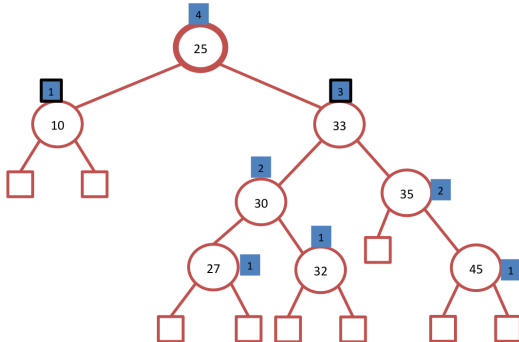


Figure: The tree needs to be balanced.

Data Structures

Balanced Binary Search Trees → AVL Trees

- Question: How do we perform `remove(k)` operation on an AVL tree?

Algorithm Sketch

//Initially p denotes the parent of the removed node

`BalanceAfterRemove(Node  $p$ )`

- Let z be the first unbalanced node going up from p
- If no such z exists then return
- Let y be the child of z of greater height.
- Let x be the child of y defined as follows:
 - If one child of y is taller than the other then x is the taller child, otherwise x is the child of y with the same side as y is of z .
- Perform Tri-node restructuring w.r.t. x, y, z
- Let b denote the tallest node (among the nodes involved in restructuring) after the restructuring.
- If b is not the root, then `BalanceAfterRemove(b.parent)`

Data Structures

Balanced Binary Search Trees → AVL Trees

Algorithm Sketch

//Initially p denotes the parent of the removed node

BalanceAfterRemove(Node p)

- Let z be the first unbalanced node going up from p
- If no such z exists then return
- Let y be the child of z of greater height.
- Let x be the child of y defined as follows:
If one child of y is taller than the other then x is the taller child, otherwise x is the child of y with the same side as y is of z .
- Perform Tri-node restructuring w.r.t. x, y, z
- Let b denote the tallest node (among the nodes involved in restructuring) after the restructuring.
- If b is not the root, then BalanceAfterRemove($b.parent$)

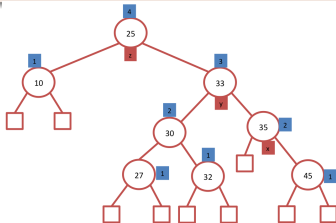


Figure: The tree needs to be balanced.

Data Structures

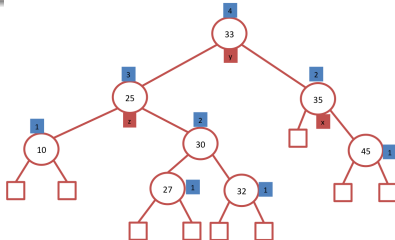
Balanced Binary Search Trees → AVL Trees

Algorithm Sketch

//Initially p denotes the parent of the removed node

BalanceAfterRemove(Node p)

- Let z be the first unbalanced node going up from p
- If no such z exists then return
- Let y be the child of z of greater height.
- Let x be the child of y defined as follows:
If one child of y is taller than the other then x is the taller child, otherwise x is the child of y with the same side as y is of z .
- Perform Tri-node restructuring w.r.t. x, y, z
- Let b denote the tallest node (among the nodes involved in restructuring) after the restructuring.
- If b is not the root, then BalanceAfterRemove($b.parent$)



Data Structures

Balanced Binary Search Trees $\rightarrow$ AVL Trees

Algorithm Sketch

//Initially p denotes the parent of the removed node

BalanceAfterRemove(Node p)

- Let z be the first unbalanced node going up from p
- If no such z exists then return
- Let y be the child of z of greater height.
- Let x be the child of y defined as follows:
If one child of y is taller than the other then x is the taller child, otherwise x is the child of y with the same side as y is of z .
- Perform Tri-node restructuring w.r.t. x, y, z
- Let b denote the tallest node (among the nodes involved in restructuring) after the restructuring.
- If b is not the root, then BalanceAfterRemove($b.parent$)

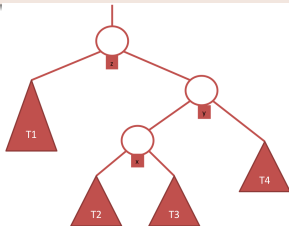


Figure: Suppose a node is deleted from $T1$.

Data Structures

Balanced Binary Search Trees → AVL Trees

Algorithm Sketch

//Initially p denotes the parent of the removed node

BalanceAfterRemove(Node p)

- Let z be the first unbalanced node going up from p
- If no such z exists then return
- Let y be the child of z of greater height.
- Let x be the child of y defined as follows:
If one child of y is taller than the other then x is the taller child, otherwise x is the child of y with the same side as y is of z .
- Perform Tri-node restructuring w.r.t. x, y, z
- Let b denote the tallest node (among the nodes involved in restructuring) after the restructuring.
- If b is not the root, then BalanceAfterRemove($b.parent$)

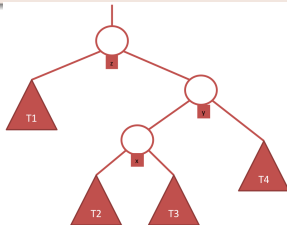


Figure: Suppose a node is deleted from $T1$.

Data Structures

Balanced Binary Search Trees → AVL Trees

Algorithm Sketch

//Initially p denotes the parent of the removed node

`BalanceAfterRemove(Node  $p$ )`

- Let z be the first unbalanced node going up from p
- If no such z exists then return
- Let y be the child of z of greater height.
- Let x be the child of y defined as follows:
If one child of y is taller than the other then x is the taller child, otherwise x is the child of y with the same side as y is of z .
- Perform Tri-node restructuring w.r.t. x, y, z
- Let b denote the tallest node (among the nodes involved in restructuring) after the restructuring.
- If b is not the root, then `BalanceAfterRemove( $b.parent$ )`

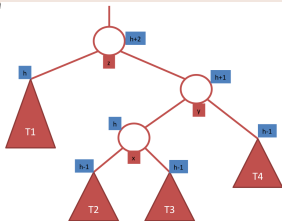


Figure: Suppose a node is deleted from $T1$. One possible scenario for heights before deletion.

Data Structures

Balanced Binary Search Trees → AVL Trees

Algorithm Sketch

//Initially p denotes the parent of the removed node

BalanceAfterRemove(Node p)

- Let z be the first unbalanced node going up from p
- If no such z exists then return
- Let y be the child of z of greater height.
- Let x be the child of y defined as follows:
If one child of y is taller than the other then x is the taller child, otherwise x is the child of y with the same side as y is of z .
- Perform Tri-node restructuring w.r.t. x, y, z
- Let b denote the tallest node (among the nodes involved in restructuring) after the restructuring.
- If b is not the root, then BalanceAfterRemove(b .parent)

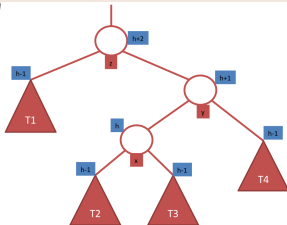


Figure: Suppose a node is deleted from $T1$. Heights after deletion.

Data Structures

Balanced Binary Search Trees → AVL Trees

Algorithm Sketch

//Initially p denotes the parent of the removed node

BalanceAfterRemove(Node p)

- Let z be the first unbalanced node going up from p
- If no such z exists then return
- Let y be the child of z of greater height.
- Let x be the child of y defined as follows:
If one child of y is taller than the other then x is the taller child, otherwise x is the child of y with the same side as y is of z .
- Perform Tri-node restructuring w.r.t. x, y, z
- Let b denote the tallest node (among the nodes involved in restructuring) after the restructuring.
- If b is not the root, then BalanceAfterRemove($b.parent$)

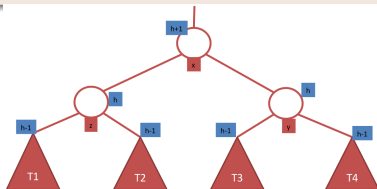


Figure: Suppose a node is deleted from $T1$. Heights after tri-node restructuring.

Data Structures: **Balanced** Binary Search Trees

Data Structures

Binary Search Trees → Multiway Search Trees

- Binary search trees allows storage and access of data in time proportional to the height of the tree.
- The number of children of a node binary search trees is upper bounded by 2.
- Removing this restriction might provide us more flexibility without costing us in terms of performance.
- **Multiway Search Trees** are generalisation of Binary Search Trees where internal nodes are allowed to have more than two children.

Data Structures

Binary Search Trees → Multiway Search Trees

- **Multiway Search Trees** are generalisation of Binary Search Trees where internal nodes are allowed to have more than two children.
- A node in an ordered tree is said to be a d -node iff it has d children.

Definition (Multiway Search Tree)

A Multiway Search Tree is an ordered tree that has the following properties:

- Each internal node is a d -node with $d \geq 2$.
- Each internal d -node with children $c_1, \dots, c_d$ stores an ordered set of $(d - 1)$ key-value pairs $(k_1, v_1), (k_2, v_2), \dots, (k_{d-1}, v_{d-1})$, where $k_1 \leq k_2 \leq \dots \leq k_{d-1}$.
- Consider any d -node w . Let $k_0 = -\infty$ and $k_d = +\infty$ by convention. For every entry (k, v) stored in the subtree rooted at c_i , we have $k_{i-1} \leq k < k_i$.

Data Structures

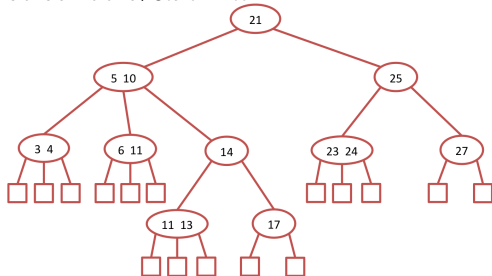
Binary Search Trees → Multiway Search Trees

Definition (Multiway Search Tree)

A Multiway Search Tree is an ordered tree that has the following properties:

- Each internal node is a d -node with $d \geq 2$.
- Each internal d -node with children $c_1, \dots, c_d$ stores an ordered set of $(d - 1)$ key-value pairs $(k_1, v_1), (k_2, v_2), \dots, (k_{d-1}, v_{d-1})$, where $k_1 \leq k_2 \leq \dots \leq k_{d-1}$.
- Consider any d -node w . Let $k_0 = -\infty$ and $k_d = +\infty$ by convention. For every entry (k, v) stored in the subtree rooted at c_i , we have $k_{i-1} \leq k < k_i$.

- Is this a Multiway Search Tree?



Data Structures

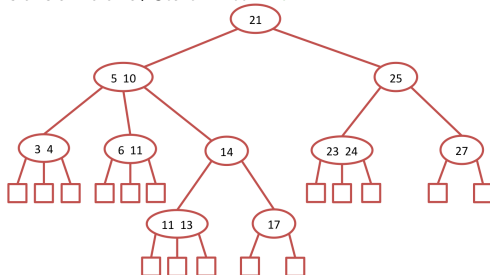
Binary Search Trees → Multiway Search Trees

Definition (Multiway Search Tree)

A Multiway Search Tree is an ordered tree that has the following properties:

- Each internal node is a d -node with $d \geq 2$.
- Each internal d -node with children $c_1, \dots, c_d$ stores an ordered set of $(d - 1)$ key-value pairs $(k_1, v_1), (k_2, v_2), \dots, (k_{d-1}, v_{d-1})$, where $k_1 \leq k_2 \leq \dots \leq k_{d-1}$.
- Consider any d -node w . Let $k_0 = -\infty$ and $k_d = +\infty$ by convention. For every entry (k, v) stored in the subtree rooted at c_i , we have $k_{i-1} \leq k < k_i$.

- Is this a Multiway Search Tree? No



Data Structures

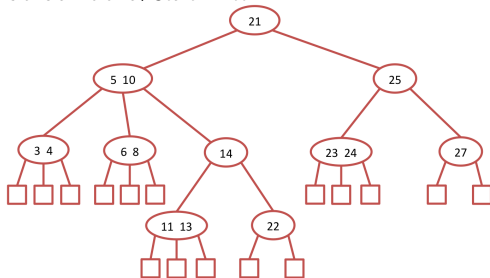
Binary Search Trees → Multiway Search Trees

Definition (Multiway Search Tree)

A Multiway Search Tree is an ordered tree that has the following properties:

- Each internal node is a d -node with $d \geq 2$.
- Each internal d -node with children $c_1, \dots, c_d$ stores an ordered set of $(d - 1)$ key-value pairs $(k_1, v_1), (k_2, v_2), \dots, (k_{d-1}, v_{d-1})$, where $k_1 \leq k_2 \leq \dots \leq k_{d-1}$.
- Consider any d -node w . Let $k_0 = -\infty$ and $k_d = +\infty$ by convention. For every entry (k, v) stored in the subtree rooted at c_i , we have $k_{i-1} \leq k < k_i$.

- Is this a Multiway Search Tree?



Data Structures

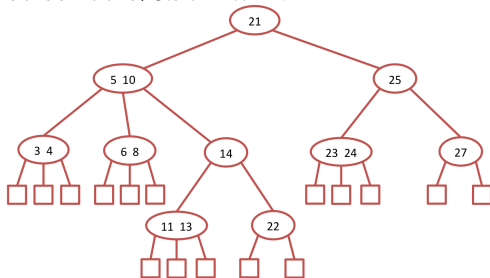
Binary Search Trees → Multiway Search Trees

Definition (Multiway Search Tree)

A Multiway Search Tree is an ordered tree that has the following properties:

- Each internal node is a d -node with $d \geq 2$.
- Each internal d -node with children $c_1, \dots, c_d$ stores an ordered set of $(d - 1)$ key-value pairs $(k_1, v_1), (k_2, v_2), \dots, (k_{d-1}, v_{d-1})$, where $k_1 \leq k_2 \leq \dots \leq k_{d-1}$.
- Consider any d -node w . Let $k_0 = -\infty$ and $k_d = +\infty$ by convention. For every entry (k, v) stored in the subtree rooted at c_i , we have $k_{i-1} \leq k < k_i$.

- Is this a Multiway Search Tree? No



Data Structures

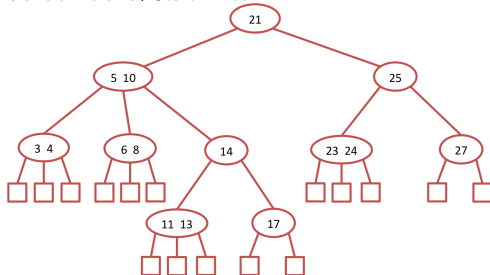
Binary Search Trees $\rightarrow$ Multiway Search Trees

Definition (Multiway Search Tree)

A Multiway Search Tree is an ordered tree that has the following properties:

- Each internal node is a d -node with $d \geq 2$.
- Each internal d -node with children $c_1, \dots, c_d$ stores an ordered set of $(d - 1)$ key-value pairs $(k_1, v_1), (k_2, v_2), \dots, (k_{d-1}, v_{d-1})$, where $k_1 \leq k_2 \leq \dots \leq k_{d-1}$.
- Consider any d -node w . Let $k_0 = -\infty$ and $k_d = +\infty$ by convention. For every entry (k, v) stored in the subtree rooted at c_i , we have $k_{i-1} \leq k < k_i$.

- Is this a Multiway Search Tree?



Data Structures

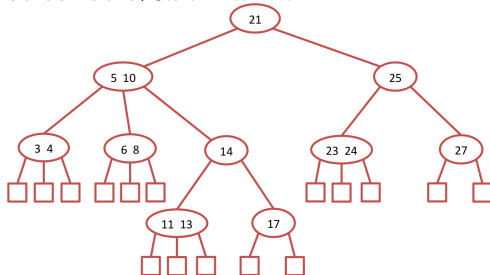
Binary Search Trees $\rightarrow$ Multiway Search Trees

Definition (Multiway Search Tree)

A Multiway Search Tree is an ordered tree that has the following properties:

- Each internal node is a d -node with $d \geq 2$.
- Each internal d -node with children $c_1, \dots, c_d$ stores an ordered set of $(d - 1)$ key-value pairs $(k_1, v_1), (k_2, v_2), \dots, (k_{d-1}, v_{d-1})$, where $k_1 \leq k_2 \leq \dots \leq k_{d-1}$.
- Consider any d -node w . Let $k_0 = -\infty$ and $k_d = +\infty$ by convention. For every entry (k, v) stored in the subtree rooted at c_i , we have $k_{i-1} \leq k < k_i$.

- Is this a Multiway Search Tree? **Yes**



Data Structures

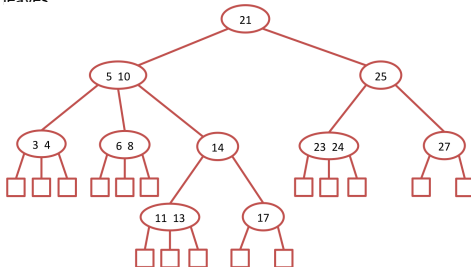
Binary Search Trees → Multiway Search Trees

Definition (Multiway Search Tree)

A Multiway Search Tree is an ordered tree that has the following properties:

- Each internal node is a d -node with $d \geq 2$.
- Each internal d -node with children $c_1, \dots, c_d$ stores an ordered set of $(d - 1)$ key-value pairs $(k_1, v_1), (k_2, v_2), \dots, (k_{d-1}, v_{d-1})$, where $k_1 \leq k_2 \leq \dots \leq k_{d-1}$.
- Consider any d -node w . Let $k_0 = -\infty$ and $k_d = +\infty$ by convention. For every entry (k, v) stored in the subtree rooted at c_i , we have $k_{i-1} \leq k < k_i$.

- Claim 1: Any multiway search tree containing n entries has $n + 1$ leaves



End