CSE 221: Algorithms

Balanced trees

Mumit Khan
Fatema Tuz Zohora

Computer Science and Engineering
BRAC University

References

1. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Second Edition*. The MIT Press, September 2001.
2. Erik Demaine and Charles Leiserson, *6.046J Introduction to Algorithms*. MIT OpenCourseWare, Fall 2005. Available from: `ocw.mit.edu/OcwWeb/Electrical-Engineering-and-Computer-Science/6-046JFall-2005/CourseHome/index.htm`
3. Robert Sedgewick, *Left-Leaning Red-Black Trees*. 2008.

Last modified: February 9, 2013

Contents

Contents

Need for balanced trees

- The lookup and insertion time in a binary search tree is $O(h)$:

  Best case when the tree is balanced, $h = \lfloor \lg n \rfloor = O(\lg n)$
  Worst case when the tree is *linear*, then $h = O(n)$

Need for balanced trees

- The lookup and insertion time in a binary search tree is $O(h)$:

  Best case when the tree is balanced, $h = \lfloor \lg n \rfloor = O(\lg n)$

  Worst case when the tree is *linear*, then $h = O(n)$

Need for balanced trees

- The lookup and insertion time in a binary search tree is $O(h)$:

  Best case when the tree is balanced, $h = \lfloor \lg n \rfloor = O(\lg n)$
  Worst case when the tree is *linear*, then $h = O(n)$

Need for balanced trees

- The lookup and insertion time in a binary search tree is $O(h)$:

  Best case when the tree is balanced, $h = \lfloor \lg n \rfloor = O(\lg n)$

  Worst case when the tree is *linear*, then $h = O(n)$

- So how can we guarantee $O(\lg n)$ performance in a binary search tree?

Need for balanced trees

- The lookup and insertion time in a binary search tree is $O(h)$:

  Best case when the tree is balanced, $h = \lfloor \lg n \rfloor = O(\lg n)$
  Worst case when the tree is *linear*, then $h = O(n)$

- So how can we guarantee $O(\lg n)$ performance in a binary search tree? Keep it *balanced* of course!

Need for balanced trees

- The lookup and insertion time in a binary search tree is $O(h)$:

  Best case when the tree is balanced, $h = \lfloor \lg n \rfloor = O(\lg n)$
  Worst case when the tree is *linear*, then $h = O(n)$

- So how can we guarantee $O(\lg n)$ performance in a binary search tree? Keep it *balanced* of course!

How do we balance a tree?

Need for balanced trees

- The lookup and insertion time in a binary search tree is $O(h)$:

  Best case when the tree is balanced, $h = \lfloor \lg n \rfloor = O(\lg n)$
  Worst case when the tree is *linear*, then $h = O(n)$

- So how can we guarantee $O(\lg n)$ performance in a binary search tree? Keep it *balanced* of course!

How do we balance a tree?

1. Self-balancing binary search trees – Red-Black, AVL, etc. trees.

Need for balanced trees

- The lookup and insertion time in a binary search tree is $O(h)$:

  Best case when the tree is balanced, $h = \lfloor \lg n \rfloor = O(\lg n)$

  Worst case when the tree is *linear*, then $h = O(n)$

- So how can we guarantee $O(\lg n)$ performance in a binary search tree? Keep it *balanced* of course!

How do we balance a tree?

1. Self-balancing binary search trees – Red-Black, AVL, etc. trees.
2. Bounded depth n-ary trees – 2-3-4, B, etc. trees.

Contents

2-3-4 trees

Definition (2-3-4 tree)

Generalize binary search tree to allow multiple keys per node, and ensure that all the leaves are at the same depth.

2-3-4 trees

Definition (2-3-4 tree)

Generalize binary search tree to allow multiple keys per node, and ensure that all the leaves are at the same depth.

Result: perfectly balanced tree

2-3-4 trees

Definition (2-3-4 tree)

Generalize binary search tree to allow multiple keys per node, and ensure that all the leaves are at the same depth.

Result: perfectly balanced tree

- 2-node: one key, two children (just like in a BST)
- 3-node: two keys, three children
- 4-node: three keys, four children

2-3-4 trees

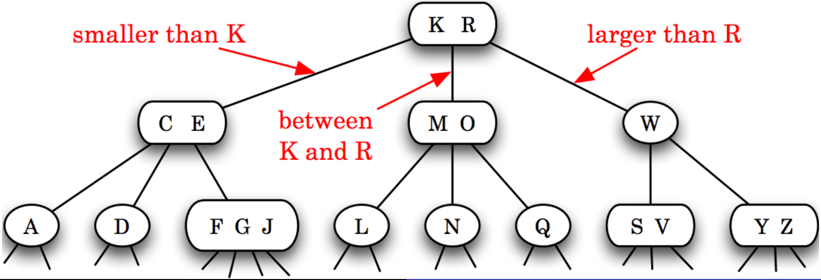
Definition (2-3-4 tree)

Generalize binary search tree to allow multiple keys per node, and ensure that all the leaves are at the same depth.

Result: perfectly balanced tree

- 2-node: one key, two children (just like in a BST)
- 3-node: two keys, three children
- 4-node: three keys, four children

Courtesy of Robert Sedgewick `http://www.cs.princeton.edu/~rs/talks/LLRB/RedBlack.pdf`



Searching in a 2-3-4 tree

- Compare search key against keys in a node.

Searching in a 2-3-4 tree

- Compare search key against keys in a node.
- Find interval containing associated search key.

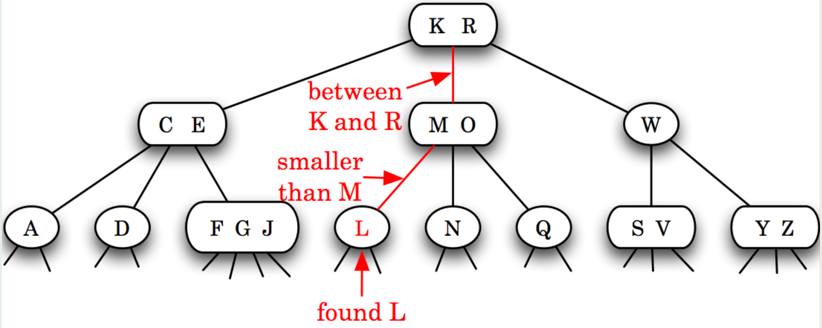
Searching in a 2-3-4 tree

- Compare search key against keys in a node.
- Find interval containing associated search key.
- Recursively follow associated link.

Searching in a 2-3-4 tree

- Compare search key against keys in a node.
- Find interval containing associated search key.
- Recursively follow associated link.

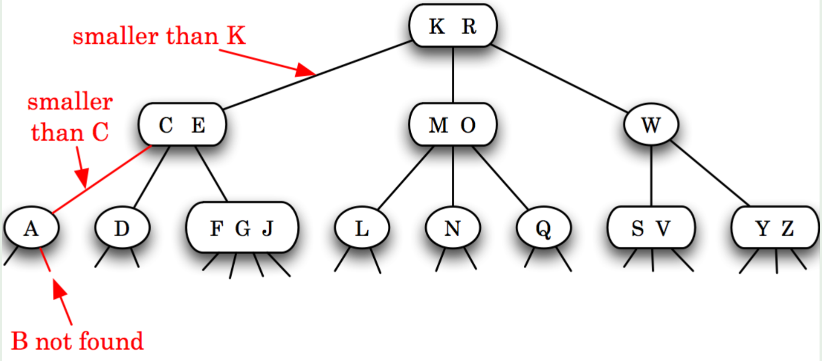
Example (Searching for L)



Inserting into a 2-3-4 tree

- Search to bottom for insertion position of key B.
- 2-node at bottom: convert to 3-node

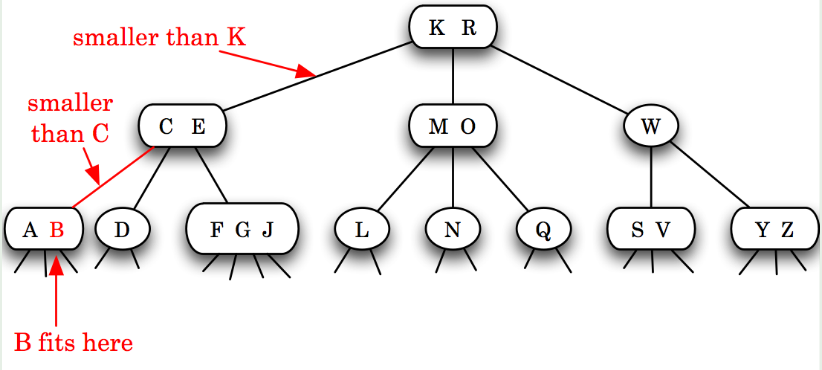
Example (Insert B)



Inserting into a 2-3-4 tree

- Search to bottom for insertion position of key B.
- 2-node at bottom: convert to 3-node

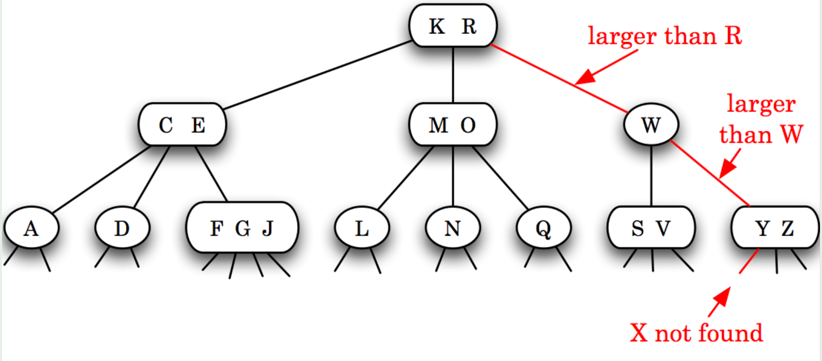
Example (Insert B)



Inserting into a 2-3-4 tree

- Search to bottom for insertion position of key X.
- 3-node at bottom: convert to 4-node

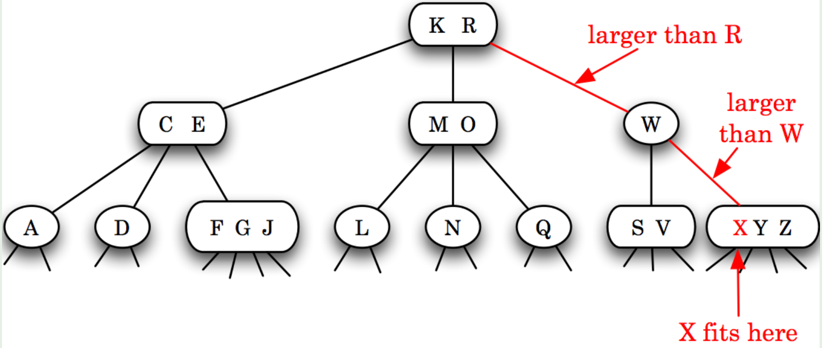
Example (Insert X)



Inserting into a 2-3-4 tree

- Search to bottom for insertion position of key X.
- 3-node at bottom: convert to 4-node

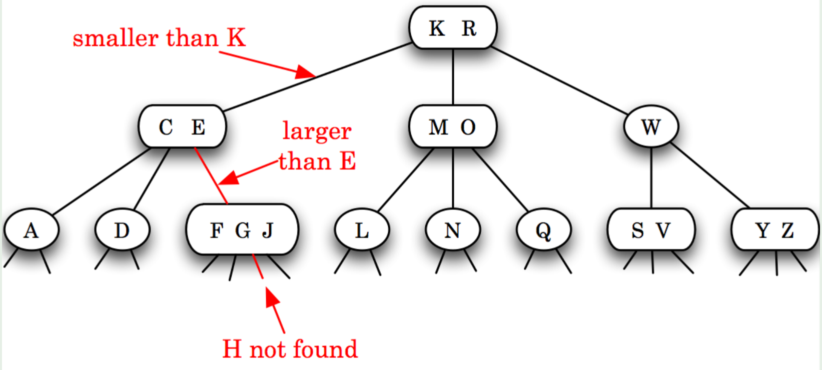
Example (Insert X)



Inserting into a 2-3-4 tree

- Search to bottom for insertion position of key H.
- 4-node at bottom: no room for key!
- Must split node to make room for new key.

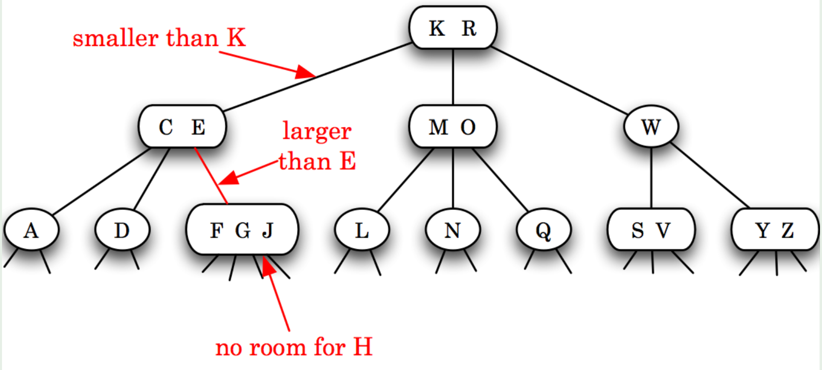
Example (Insert H)



Inserting into a 2-3-4 tree

- Search to bottom for insertion position of key H.
- 4-node at bottom: no room for key!
- Must split node to make room for new key.

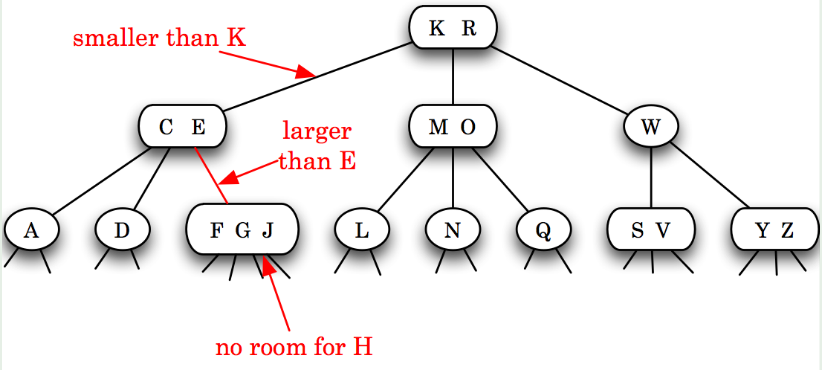
Example (Insert H)



Inserting into a 2-3-4 tree

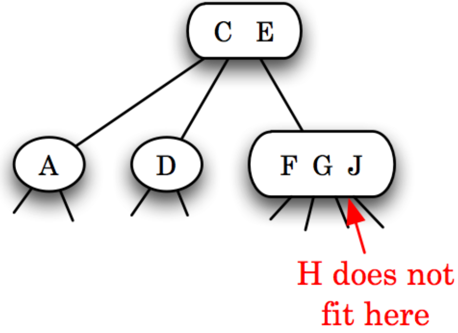
- Search to bottom for insertion position of key H.
- 4-node at bottom: no room for key!
- Must split node to make room for new key.

Example (Insert H)



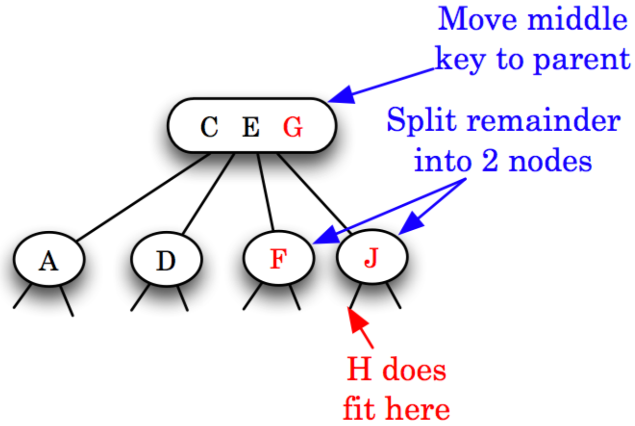
Splitting a 4-node in a 2-3-4 tree

Idea is to move the middle element to the parent, making room for one more key.



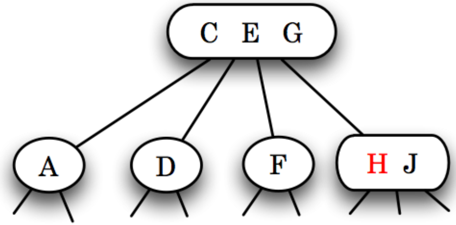
Splitting a 4-node in a 2-3-4 tree

Idea is to move the middle element to the parent, making room for one more key.



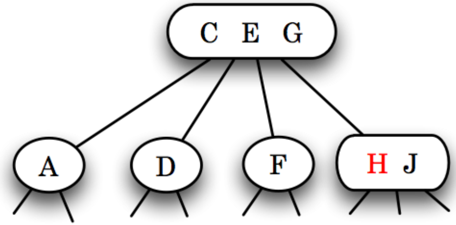
Splitting a 4-node in a 2-3-4 tree

Idea is to move the middle element to the parent, making room for one more key.



Splitting a 4-node in a 2-3-4 tree

Idea is to move the middle element to the parent, making room for one more key.

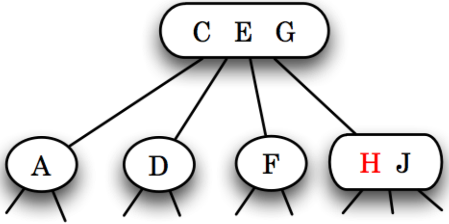


Question

What if the parent is a 4-node too!

Splitting a 4-node in a 2-3-4 tree

Idea is to move the middle element to the parent, making room for one more key.



Question

What if the parent is a 4-node too!

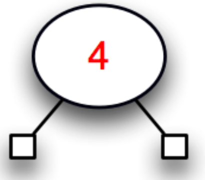
Solution: Split the parent too, potentially creating a new root.

Insertion in action

Insert 4 into an empty 2-3-4 tree

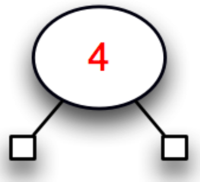
Insertion in action

Insert 4 into an empty 2-3-4 tree – done



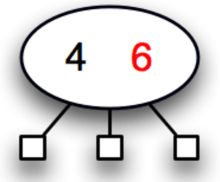
Insertion in action

Insert 6



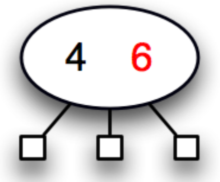
Insertion in action

Insert 6 – done



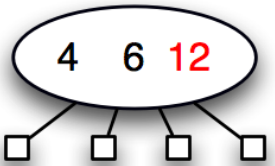
Insertion in action

Insert 12



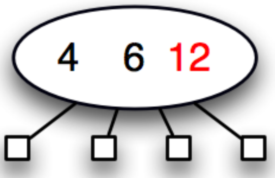
Insertion in action

Insert 12 – done



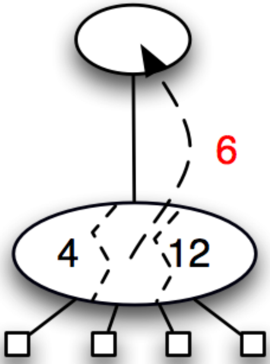
Insertion in action

Insert 15



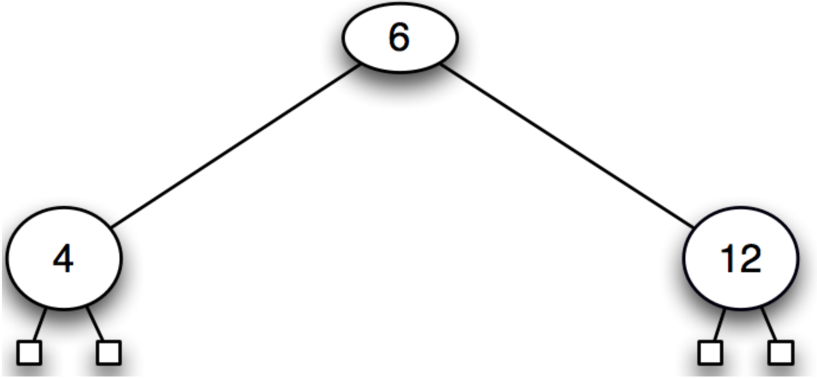
Insertion in action

Insert 15: No room, so split node



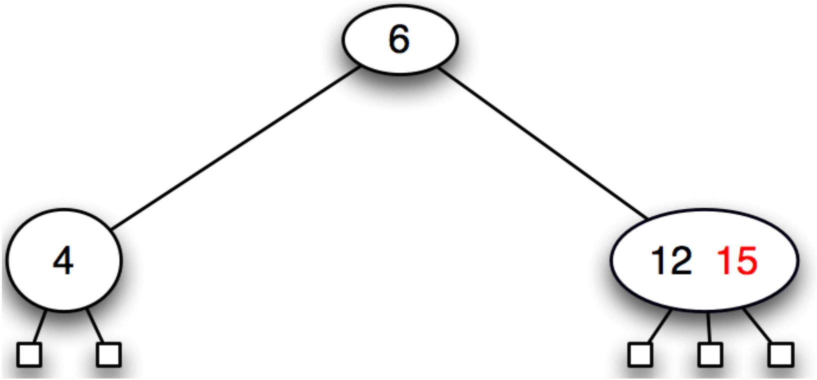
Insertion in action

Insert 15: Room available after splitting



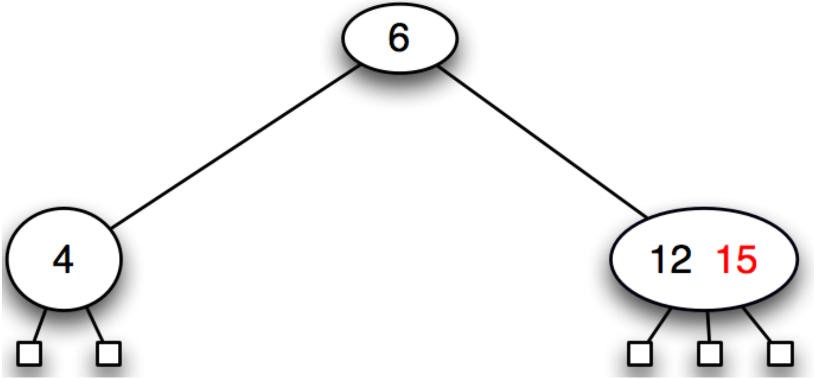
Insertion in action

Insert 15 – done



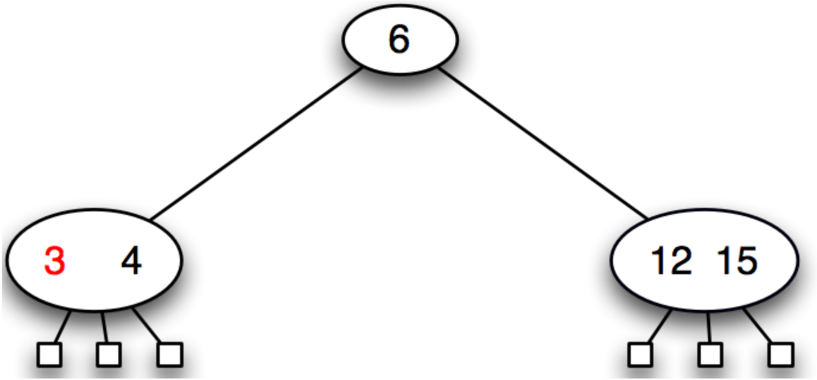
Insertion in action

Insert 3



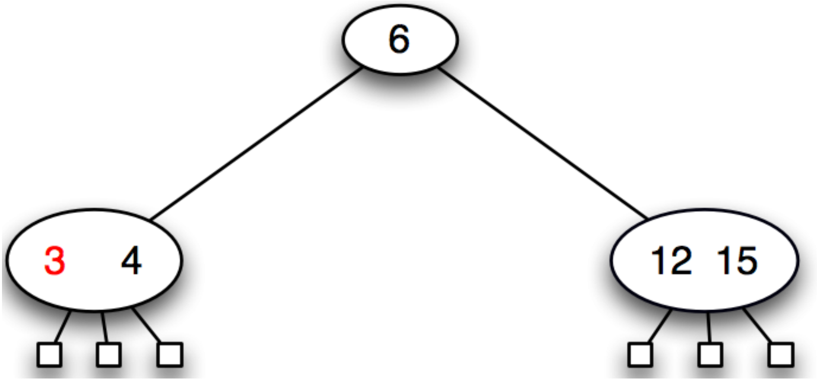
Insertion in action

Insert 3 – done



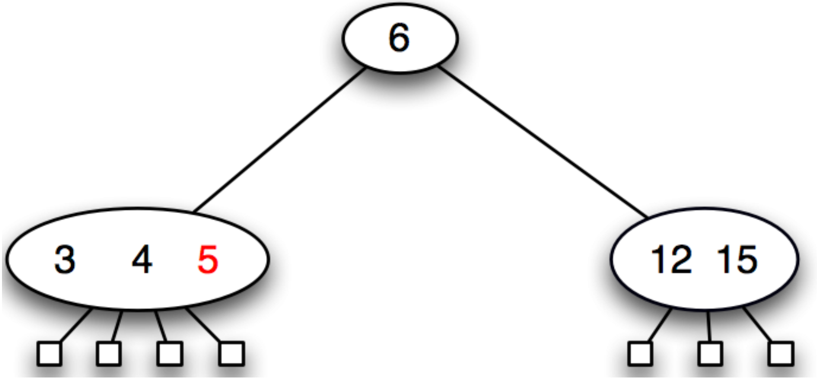
Insertion in action

Insert 5



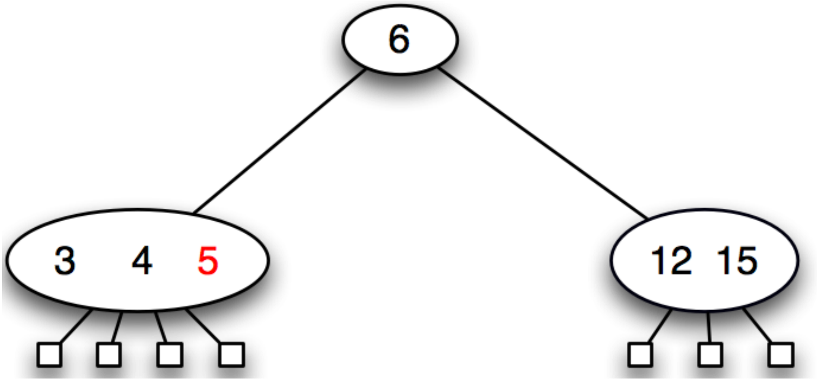
Insertion in action

Insert 5 – done



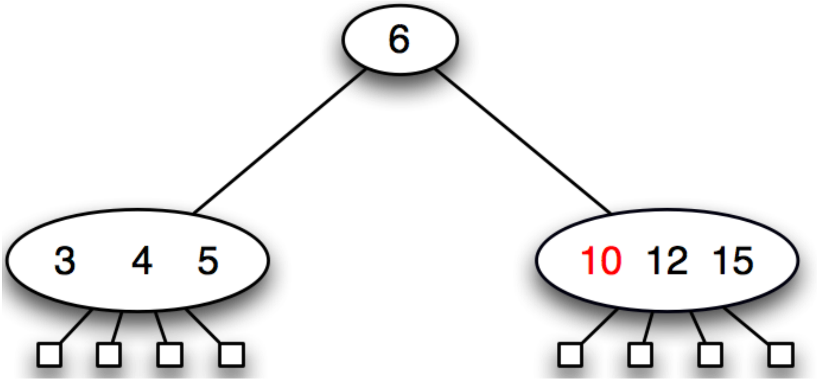
Insertion in action

Insert 10



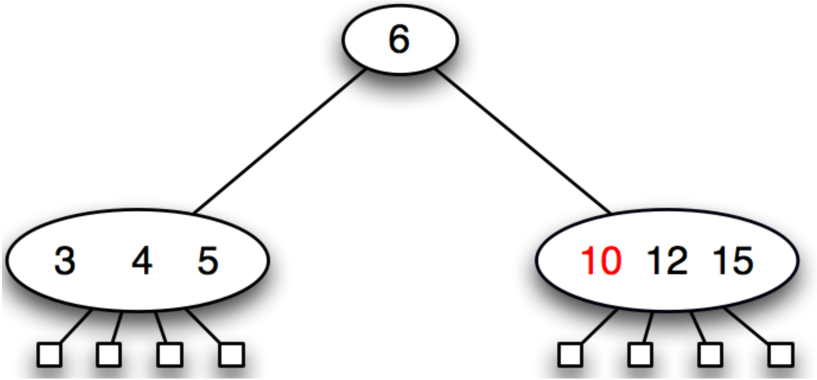
Insertion in action

Insert 10 – done



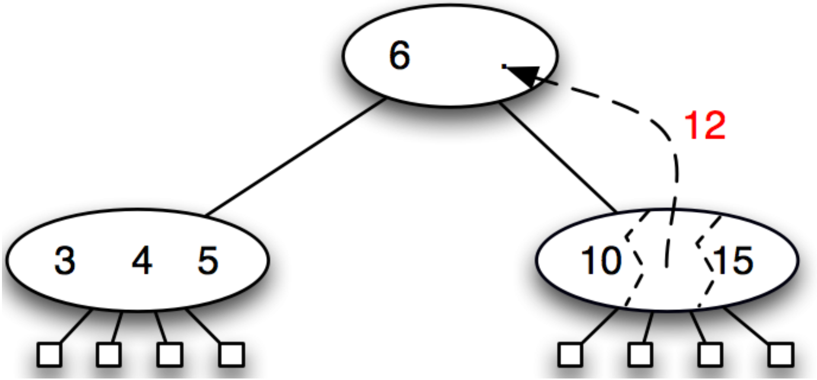
Insertion in action

Insert 8



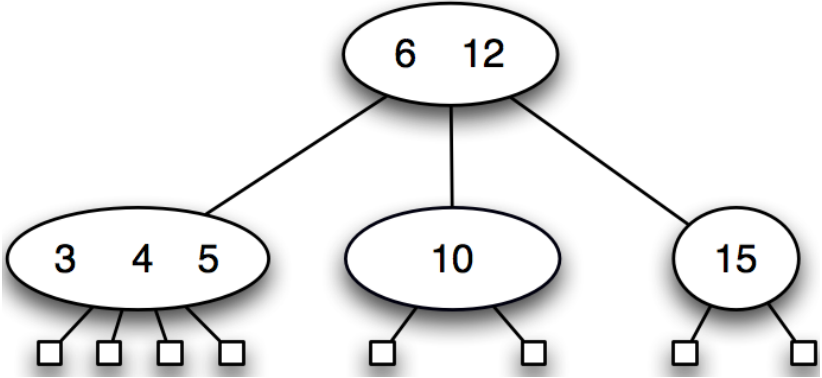
Insertion in action

Insert 8: No room, split node



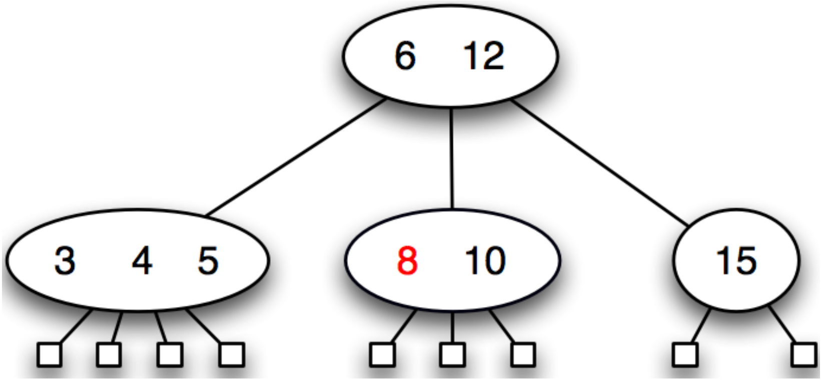
Insertion in action

Insert 8: Room available after splitting

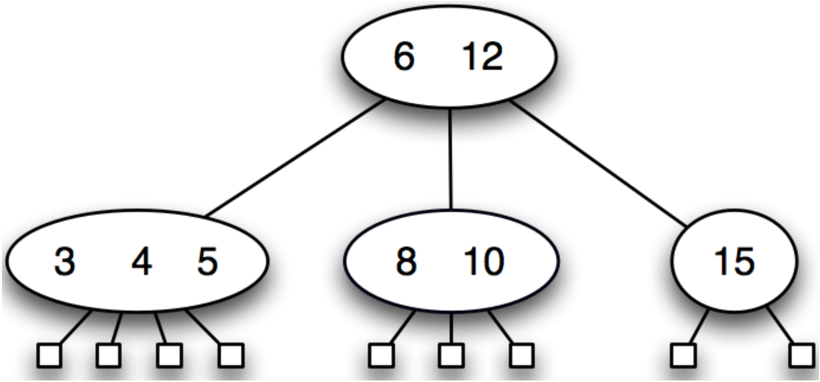


Insertion in action

Insert 8 – done



Insertion in action



Analysis of 2-3-4 tree

- Search and insert operations on a 2-3-4 tree is bounded by the height of the tree, so $O(h)$.

Analysis of 2-3-4 tree

- Search and insert operations on a 2-3-4 tree is bounded by the height of the tree, so $O(h)$.
- Maximum height occurs when all nodes are 2-nodes, so for a tree with n keys, we have $n + 1 \geq 2^h$, since there are $n + 1$ external nodes at height h.

Analysis of 2-3-4 tree

- Search and insert operations on a 2-3-4 tree is bounded by the height of the tree, so $O(h)$.
- Maximum height occurs when all nodes are 2-nodes, so for a tree with n keys, we have $n+1 \geq 2^h$, since there are $n+1$ external nodes at height h.
- Minimum height occurs when all nodes are 4-nodes, so for a tree with n keys: we have $n+1 \leq 4^h$. So, $n+1 \leq 4^h = 2^{2h}$.

Analysis of 2-3-4 tree

- Search and insert operations on a 2-3-4 tree is bounded by the height of the tree, so $O(h)$.
- Maximum height occurs when all nodes are 2-nodes, so for a tree with n keys, we have $n+1 \geq 2^h$, since there are $n+1$ external nodes at height h.
- Minimum height occurs when all nodes are 4-nodes, so for a tree with n keys: we have $n+1 \leq 4^h$. So, $n+1 \leq 4^h = 2^{2h}$.
- This provides bounds on n. Taking logarithms of both sides:

$$h \leq \lg(n+1) \leq 2h$$

This proves that $h = \Theta(\lg n)$.

Analysis of 2-3-4 tree

- Search and insert operations on a 2-3-4 tree is bounded by the height of the tree, so $O(h)$.
- Maximum height occurs when all nodes are 2-nodes, so for a tree with n keys, we have $n+1 \geq 2^h$, since there are $n+1$ external nodes at height h.
- Minimum height occurs when all nodes are 4-nodes, so for a tree with n keys: we have $n+1 \leq 4^h$. So, $n+1 \leq 4^h = 2^{2h}$.
- This provides bounds on n. Taking logarithms of both sides:

$$h \leq \lg(n+1) \leq 2h$$

  This proves that $h = \Theta(\lg n)$.
- The bounded depth property guarantees that all operations are $O(h) = O(\lg n)$ in a 2-3-4 tree.

Summary of 2-3-4 trees

Summary of 2-3-4 trees

Positives

1. All leaves are the same depth – *bounded depth*.

Summary of 2-3-4 trees

Positives

1. All leaves are the same depth – *bounded depth*.
2. Search and insert operations are $O(\lg n)$ in the worst case.

Summary of 2-3-4 trees

Positives

1. All leaves are the same depth – *bounded depth*.
2. Search and insert operations are $O(\lg n)$ in the worst case.

Negatives Different types of nodes in the tree

Summary of 2-3-4 trees

Positives

1. All leaves are the same depth – *bounded depth*.
2. Search and insert operations are $O(\lg n)$ in the worst case.

Negatives Different types of nodes in the tree – complicates the data structures needed.

Summary of 2-3-4 trees

Positives

1. All leaves are the same depth – *bounded depth*.
2. Search and insert operations are $O(\lg n)$ in the worst case.

Negatives Different types of nodes in the tree – complicates the data structures needed.

Key question

Is there something that provides $O(\lg n)$ performance with the same advantages of binary tree format?

Summary of 2-3-4 trees

Positives

1. All leaves are the same depth – *bounded depth*.
2. Search and insert operations are $O(\lg n)$ in the worst case.

Negatives Different types of nodes in the tree – complicates the data structures needed.

Key question

Is there something that provides $O(\lg n)$ performance with the same advantages of binary tree format? YES – Red-Black trees!

Contents

Red-Black tree

Definition

Red-Black tree Red-Black tree is a binary search tree with the following properties:

Red-Black tree

Definition

Red-Black tree Red-Black tree is a binary search tree with the following properties:

1. Every node is either red or black.

Red-Black tree

Definition

Red-Black tree Red-Black tree is a binary search tree with the following properties:

1. Every node is either red or black.
2. The root and external nodes (leaves) are black.

Red-Black tree

Definition

Red-Black tree Red-Black tree is a binary search tree with the following properties:

1. Every node is either red or black.
2. The root and external nodes (leaves) are black.
3. If a node is red, then its parent is black.

Red-Black tree

Definition

Red-Black tree Red-Black tree is a binary search tree with the following properties:

1. Every node is either red or black.
2. The root and external nodes (leaves) are black.
3. If a node is red, then its parent is black.
4. All simple paths from any node x to a descendant external node or leaf have the same number of black nodes.

Red-Black tree

Definition

Red-Black tree Red-Black tree is a binary search tree with the following properties:

1. Every node is either red or black.
2. The root and external nodes (leaves) are black.
3. If a node is red, then its parent is black.
4. All simple paths from any node x to a descendant external node or leaf have the same number of black nodes. This number is called the black-height(x).

Red-Black tree

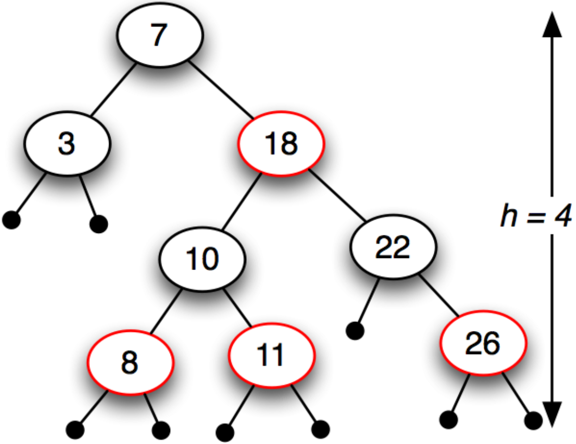
Definition

Red-Black tree Red-Black tree is a binary search tree with the following properties:

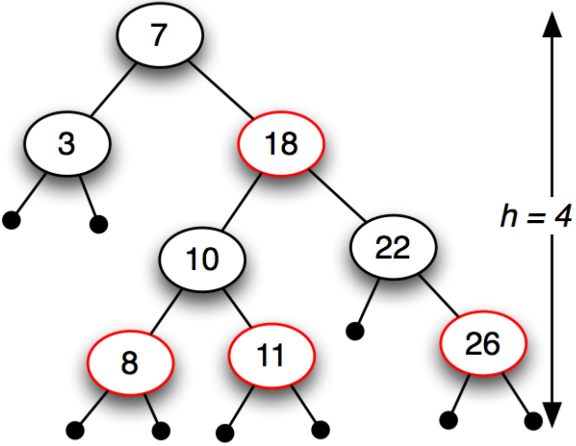
1. Every node is either red or black.
2. The root and external nodes (leaves) are black.
3. If a node is red, then its parent is black.
4. All simple paths from any node x to a descendant external node or leaf have the same number of black nodes. This number is called the black-height(x).

The data structure needed for a Red-Black tree is a binary search tree with an extra color bit for each node.

Example of red-black tree

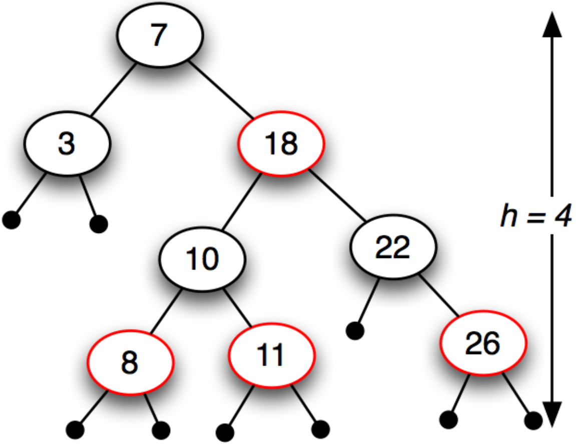


Example of red-black tree



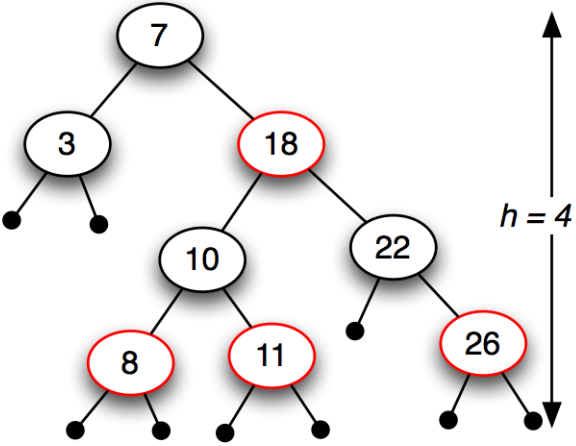
1. Every node is either red or black.

Example of red-black tree



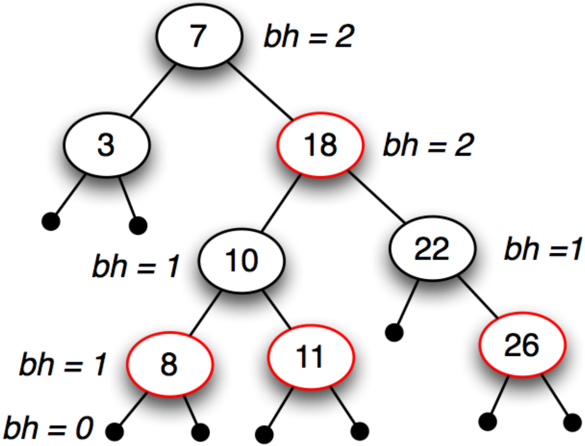
2. The root and external nodes (leaves) are black.

Example of red-black tree



3. If a node is red, then its parent is black.

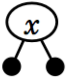
Example of red-black tree



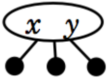
4. All simple paths from any node x to a descendant external node or leaf have the same number of black nodes $=$ black-height(x).

Equivalence of red-black tree and a 2-3-4 tree

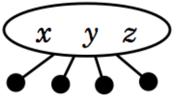
2-node



3-node

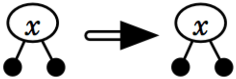


4-node

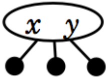


Equivalence of red-black tree and a 2-3-4 tree

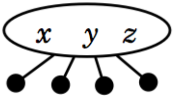
2-node



3-node

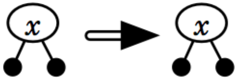


4-node

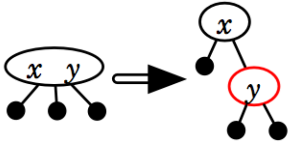


Equivalence of red-black tree and a 2-3-4 tree

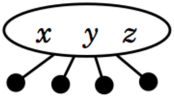
2-node



3-node

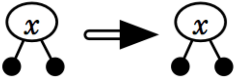


4-node

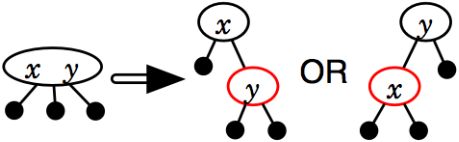


Equivalence of red-black tree and a 2-3-4 tree

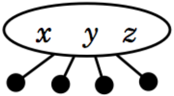
2-node



3-node

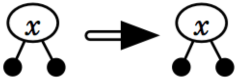


4-node

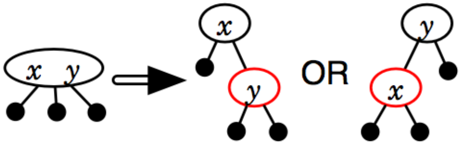


Equivalence of red-black tree and a 2-3-4 tree

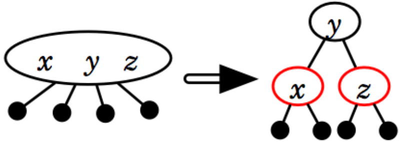
2-node



3-node

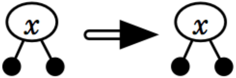


4-node

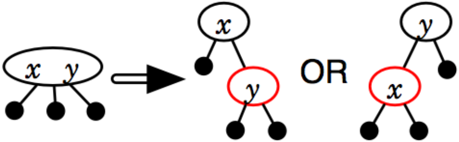


Equivalence of red-black tree and a 2-3-4 tree

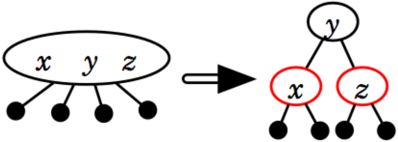
2-node



3-node



4-node

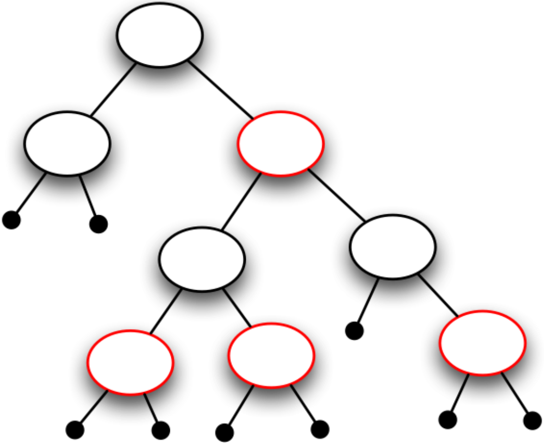


Key observation

Red-black tree is simply another way of representing a 2-3-4 tree!

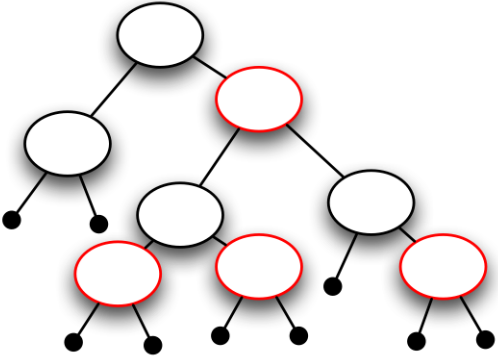
Height of a red-black tree

- Merge the red nodes into their black parents.



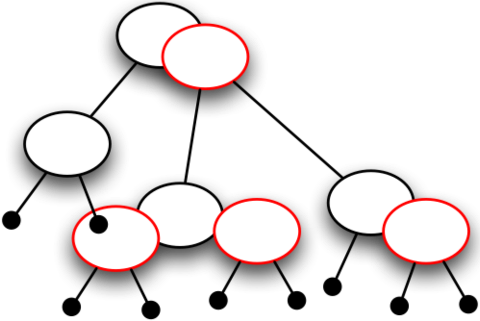
Height of a red-black tree

- Merge the red nodes into their black parents.



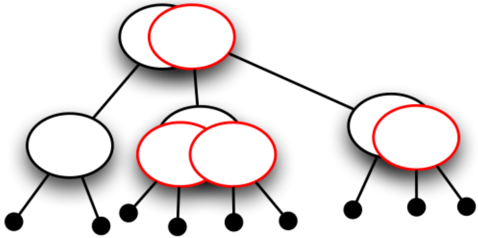
Height of a red-black tree

- Merge the red nodes into their black parents.



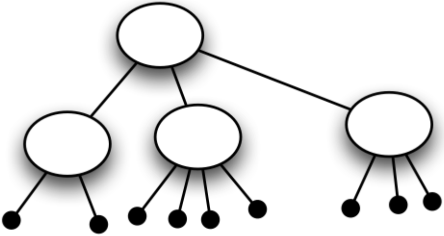
Height of a red-black tree

- Merge the red nodes into their black parents.



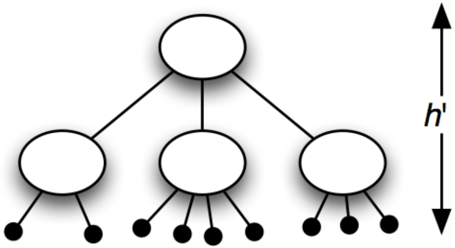
Height of a red-black tree

- Merge the red nodes into their black parents.



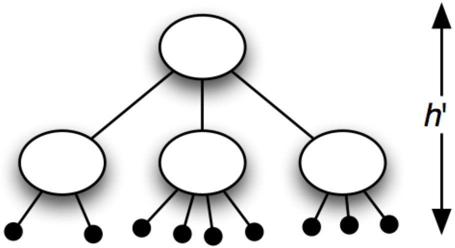
Height of a red-black tree

- Merge the red nodes into their black parents.

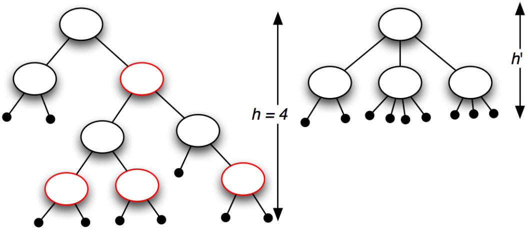


Height of a red-black tree

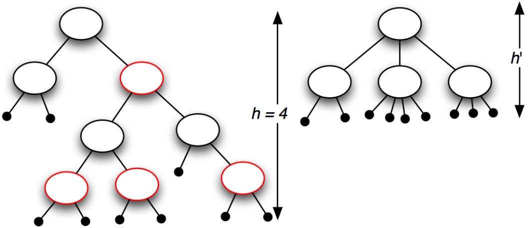
- Merge the red nodes into their black parents.
- Produces a 2-3-4 tree with height h'.



Height of a red-black tree (continued)

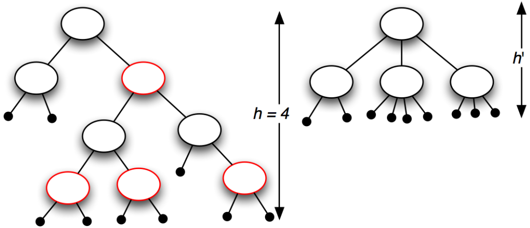


Height of a red-black tree (continued)



- We have $h' \geq h/2$, since at most half the nodes on any path are red.

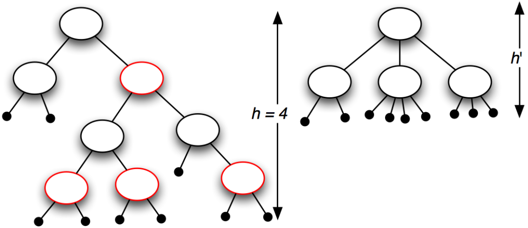
Height of a red-black tree (continued)



- We have $h' \geq h/2$, since at most half the nodes on any path are red.
- Number of external nodes or leaves is $n+1$, so we have:

$$n+1 \geq 2^{h'} \Rightarrow \lg(n+1) \geq h' \geq h/2 \Rightarrow h \leq 2\lg(n+1).$$

Height of a red-black tree (continued)



- We have $h' \geq h/2$, since at most half the nodes on any path are red.
- Number of external nodes or leaves is $n + 1$, so we have:

$$n + 1 \geq 2^{h'} \Rightarrow \lg(n + 1) \geq h' \geq h/2 \Rightarrow h \leq 2\lg(n + 1).$$

Theorem

A red-black tree with n keys has height $h \leq 2\lg(n + 1) = O(\lg n)$.

Summary of red-black trees

Summary of red-black trees

Positives

1. Very simple data structure – a binary search tree with an extra bit for encoding the color.

Summary of red-black trees

Positives

1. Very simple data structure – a binary search tree with an extra bit for encoding the color.
2. Search and insert operations are $O(\lg n)$ in the worst case.

Summary of red-black trees

Positives

1. Very simple data structure – a binary search tree with an extra bit for encoding the color.
2. Search and insert operations are $O(\lg n)$ in the worst case.

Negatives Insert and remove operations require a series of rotations to maintain the black-height property.

Summary of red-black trees

Positives

1. Very simple data structure – a binary search tree with an extra bit for encoding the color.
2. Search and insert operations are $O(\lg n)$ in the worst case.

Negatives Insert and remove operations require a series of rotations to maintain the black-height property.

Key question

How do Red-black trees compare with 2-3-4 trees in terms of performance and data structure complexity?

Conclusion

- Binary search tree guarantees a performance of $O(h)$, but h can vary from $O(\lg n)$ in the best-case to $O(n)$ in the worst-case.

Conclusion

- Binary search tree guarantees a performance of $O(h)$, but h can vary from $O(\lg n)$ in the best-case to $O(n)$ in the worst-case.
- To have $O(\lg n)$ worst-case performance, the solution is use balanced trees.

Conclusion

- Binary search tree guarantees a performance of $O(h)$, but h can vary from $O(\lg n)$ in the best-case to $O(n)$ in the worst-case.
- To have $O(\lg n)$ worst-case performance, the solution is use balanced trees. Two approaches:

  Bounded depth: n-ary tree in which that all the leaves are at the same depth.

Conclusion

- Binary search tree guarantees a performance of $O(h)$, but h can vary from $O(\lg n)$ in the best-case to $O(n)$ in the worst-case.
- To have $O(\lg n)$ worst-case performance, the solution is use balanced trees. Two approaches:

  Bounded depth n-ary tree in which that all the leaves are at the same depth. Example: B-trees, 2-3-4 trees.

Conclusion

- Binary search tree guarantees a performance of $O(h)$, but h can vary from $O(\lg n)$ in the best-case to $O(n)$ in the worst-case.
- To have $O(\lg n)$ worst-case performance, the solution is use balanced trees. Two approaches:

  Bounded depth n-ary tree in which that all the leaves are at the same depth. Example: B-trees, 2-3-4 trees.

  Self-balancing binary trees Binary trees that are self-balancing through a series of transformations (AVL trees), or use pseudo-depth (Red-Black trees).

Conclusion

- Binary search tree guarantees a performance of $O(h)$, but h can vary from $O(\lg n)$ in the best-case to $O(n)$ in the worst-case.
- To have $O(\lg n)$ worst-case performance, the solution is use balanced trees. Two approaches:

  Bounded depth n-ary tree in which that all the leaves are at the same depth. Example: B-trees, 2-3-4 trees.

  Self-balancing binary trees Binary trees that are self-balancing through a series of transformations (AVL trees), or use pseudo-depth (Red-Black trees).

Questions to ask (and remember)

Conclusion

- Binary search tree guarantees a performance of $O(h)$, but h can vary from $O(\lg n)$ in the best-case to $O(n)$ in the worst-case.
- To have $O(\lg n)$ worst-case performance, the solution is use balanced trees. Two approaches:

  Bounded depth n-ary tree in which that all the leaves are at the same depth. Example: B-trees, 2-3-4 trees.

  Self-balancing binary trees Binary trees that are self-balancing through a series of transformations (AVL trees), or use pseudo-depth (Red-Black trees).

Questions to ask (and remember)

- What's the equivalence of a 2-3-4 tree and Red-Black tree?

Conclusion

- Binary search tree guarantees a performance of $O(h)$, but h can vary from $O(\lg n)$ in the best-case to $O(n)$ in the worst-case.
- To have $O(\lg n)$ worst-case performance, the solution is use balanced trees. Two approaches:

  Bounded depth n-ary tree in which that all the leaves are at the same depth. Example: B-trees, 2-3-4 trees.

  Self-balancing binary trees Binary trees that are self-balancing through a series of transformations (AVL trees), or use pseudo-depth (Red-Black trees).

Questions to ask (and remember)

- What's the equivalence of a 2-3-4 tree and Red-Black tree?
- Why is the data structure in implementing a 2-3-4 tree considered complex?

Conclusion

- Binary search tree guarantees a performance of $O(h)$, but h can vary from $O(\lg n)$ in the best-case to $O(n)$ in the worst-case.
- To have $O(\lg n)$ worst-case performance, the solution is use balanced trees. Two approaches:

  Bounded depth n-ary tree in which that all the leaves are at the same depth. Example: B-trees, 2-3-4 trees.

  Self-balancing binary trees Binary trees that are self-balancing through a series of transformations (AVL trees), or use pseudo-depth (Red-Black trees).

Questions to ask (and remember)

- What's the equivalence of a 2-3-4 tree and Red-Black tree?
- Why is the data structure in implementing a 2-3-4 tree considered complex?
- What are some of the disadvantages of a Red-Black tree?