B-Trees

Saikrishna Arcot
(edits by M. Hudachek-Buswell)

May 30, 2017

B-Trees

- B-trees are similar to BSTs and AVL trees (in that smaller items are to the left of a data item and larger items are to the right), but one major difference is that each node can have multiple data items.

B-Trees

- B-trees are similar to BSTs and AVL trees (in that smaller items are to the left of a data item and larger items are to the right), but one major difference is that each node can have multiple data items.
- The **order** of a B-tree describes how many data items each node have have, and how many children each node can have.

B-Trees

- B-trees are similar to BSTs and AVL trees (in that smaller items are to the left of a data item and larger items are to the right), but one major difference is that each node can have multiple data items.
- The **order** of a B-tree describes how many data items each node have have, and how many children each node can have.
- For example, if the order of a B-tree is 5, each node can have 5 children, and each node can have $5 - 1 = 4$ data items.

B-Trees

- B-trees are similar to BSTs and AVL trees (in that smaller items are to the left of a data item and larger items are to the right), but one major difference is that each node can have multiple data items.
- The **order** of a B-tree describes how many data items each node have have, and how many children each node can have.
- For example, if the order of a B-tree is 5, each node can have 5 children, and each node can have $5 - 1 = 4$ data items.
- The slides will focus on 2-4 trees, which are B-trees with an order of 4.

Node

- Each node in a 2-4 tree can hold up to 3 data items and can have up to 4 children.

Node

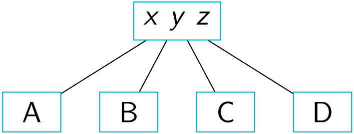
- Each node in a 2-4 tree can hold up to 3 data items and can have up to 4 children.
- The data items in each node is stored in ascending order.

Node

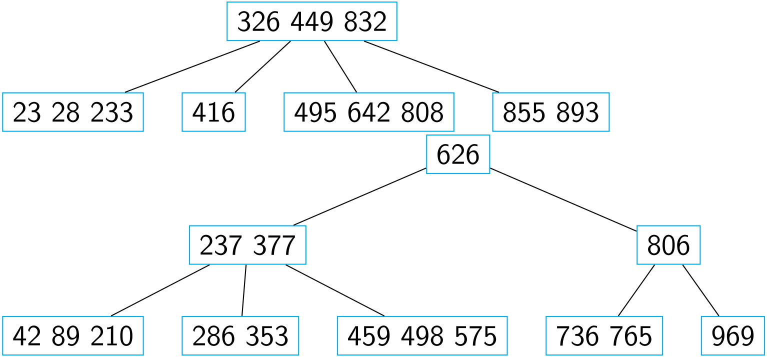
- Each node in a 2-4 tree can hold up to 3 data items and can have up to 4 children.
- The data items in each node is stored in ascending order.
- The first child is less than the first data item, the second child is between the first data item and the second data item, the third child is between the second data item and third data item, and the fourth child is greater than the third data item.

Node

In the node below, everything in node A is less than x, everything in node B is between x and y, everything in node C is between y and z, and everything in node D is greater than z.



Example 2-4 Tree



Properties

- In a B-tree, the depth of all of the leaf nodes are the same.

Properties

- In a B-tree, the depth of all of the leaf nodes are the same.
- If there are n data items in a node, there must be either 0 or $n + 1$ children.

Properties

- In a B-tree, the depth of all of the leaf nodes are the same.
- If there are n data items in a node, there must be either 0 or $n + 1$ children.
- Because of these rules, a B-tree will always be balanced, and searching, adding to, or removing from a B-tree will be worst case $O(\log n)$.

Searching

- Start at the root node.

Searching

- Start at the root node.
- Compare the first data item with what you're searching for.

Searching

- Start at the root node.
- Compare the first data item with what you're searching for.
  - If the data you're looking for is less than the data in the node, then go to the n^{th} child, where n is the position of the data in the node.

Searching

- Start at the root node.
- Compare the first data item with what you're searching for.
  - If the data you're looking for is less than the data in the node, then go to the n^{th} child, where n is the position of the data in the node.
  - If the data you're looking for is equal to the data in the node, then you've found the data.

Searching

- Start at the root node.
- Compare the first data item with what you're searching for.
  - If the data you're looking for is less than the data in the node, then go to the n^{th} child, where n is the position of the data in the node.
  - If the data you're looking for is equal to the data in the node, then you've found the data.
  - If the data you're looking for is greater than the data in the node, then go to the next data item in the node.

Searching

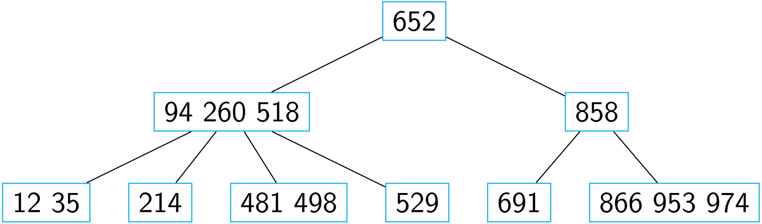
- Start at the root node.
- Compare the first data item with what you're searching for.
  - If the data you're looking for is less than the data in the node, then go to the n^{th} child, where n is the position of the data in the node.
  - If the data you're looking for is equal to the data in the node, then you've found the data.
  - If the data you're looking for is greater than the data in the node, then go to the next data item in the node.
  - If the data you're looking for is greater than all of the data items in the node, then go to the last child.

Searching

- Start at the root node.
- Compare the first data item with what you're searching for.
  - If the data you're looking for is less than the data in the node, then go to the n^{th} child, where n is the position of the data in the node.
  - If the data you're looking for is equal to the data in the node, then you've found the data.
  - If the data you're looking for is greater than the data in the node, then go to the next data item in the node.
  - If the data you're looking for is greater than all of the data items in the node, then go to the last child.
- Repeat the previous step, until you find the data in the tree or go off of the tree, in which case the data you're looking for isn't in the tree.

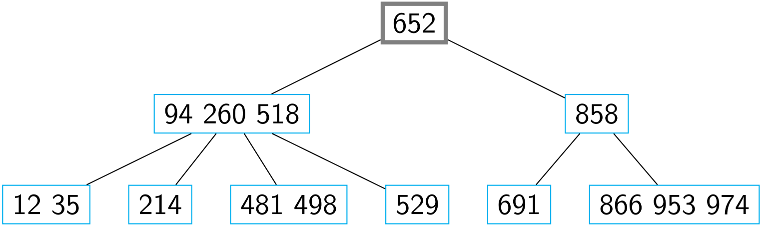
Searching

For example, if you were searching for 481:



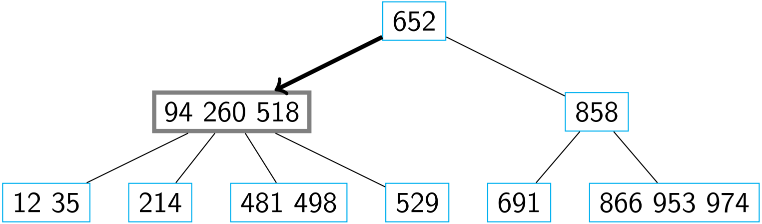
Searching

For example, if you were searching for 481:



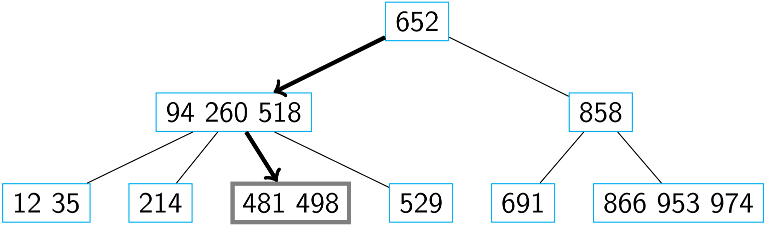
Searching

For example, if you were searching for 481:



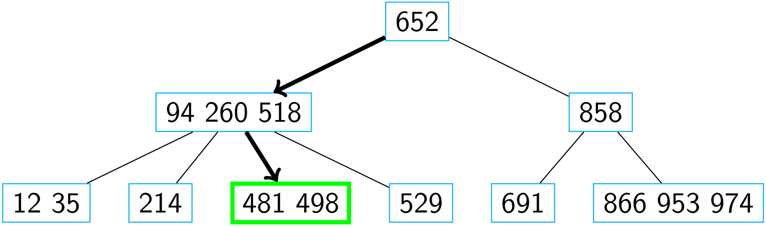
Searching

For example, if you were searching for 481:



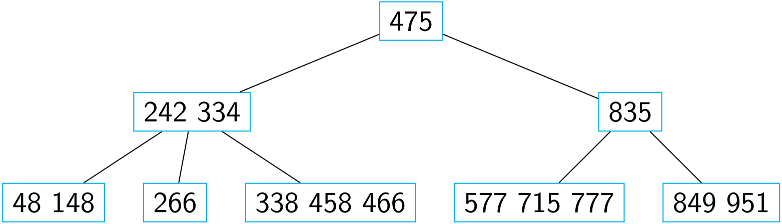
Searching

For example, if you were searching for 481:



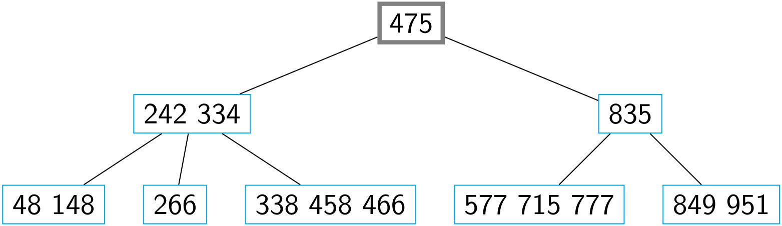
Searching

For example, if you were searching for 951:



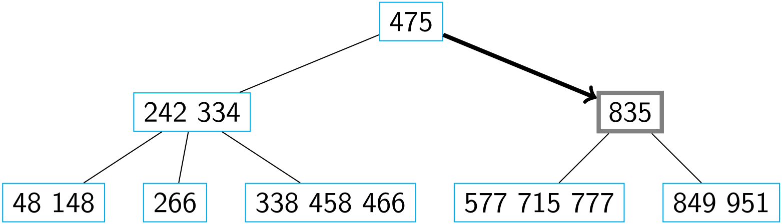
Searching

For example, if you were searching for 951:



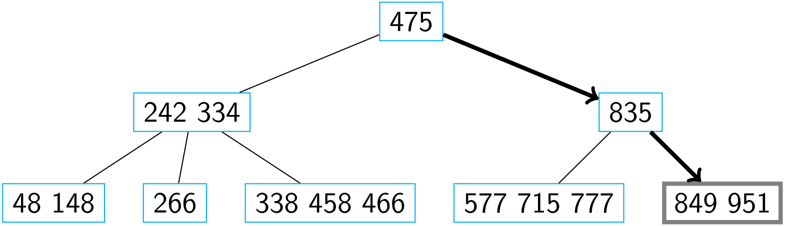
Searching

For example, if you were searching for 951:



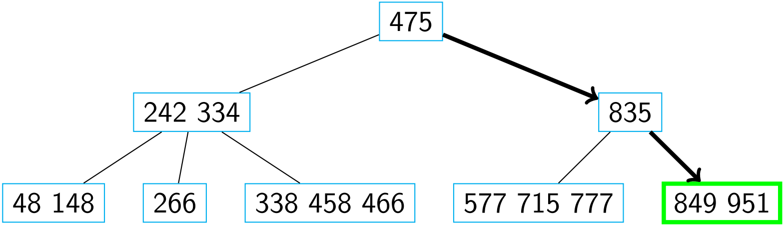
Searching

For example, if you were searching for 951:



Searching

For example, if you were searching for 951:



Searching

```
procedure SEARCH(data, node)
    if node is not valid then
        return FALSE
    else
        for i ← 1, n do
            if data = node.data[i] then
                return TRUE
            else if data < node.data[i] then
                child ← i^th child node
                return SEARCH(data, child)
            end if
        end for
```

Searching

child ← last child node
return SEARCH(*data*, *child*)
end if
end procedure

Adding

- When adding to a B-tree, the new data item is always added to a leaf node, not to an internal node.

Adding

- When adding to a B-tree, the new data item is always added to a leaf node, not to an internal node.
- Follow the same steps as searching until you find the data in the tree or reach a leaf node.

Adding

- When adding to a B-tree, the new data item is always added to a leaf node, not to an internal node.
- Follow the same steps as searching until you find the data in the tree or reach a leaf node.
- If the data is not in the tree (i.e. you are at a leaf node), add the data to the leaf node.

Adding

- When adding to a B-tree, the new data item is always added to a leaf node, not to an internal node.
- Follow the same steps as searching until you find the data in the tree or reach a leaf node.
- If the data is not in the tree (i.e. you are at a leaf node), add the data to the leaf node.
- If the data is in the tree, then what happens is implementation-defined. Some implementations may do nothing, some implementations may update the data item, and some implementations may add a duplicate.

Overflow

- After the data is added to a leaf node, the node may have too many data items for the type of tree. (For example, in the case of a 2-4 tree, the node could have 4 data items instead of the maximum of 3 items.)

Overflow

- After the data is added to a leaf node, the node may have too many data items for the type of tree. (For example, in the case of a 2-4 tree, the node could have 4 data items instead of the maximum of 3 items.)
- In such a case, the middle data item (either the second or third item in a 2-4 tree) is promoted to the parent node (if a parent doesn't exist, a parent node is created), and this node is split into two nodes.

Overflow

- After the data is added to a leaf node, the node may have too many data items for the type of tree. (For example, in the case of a 2-4 tree, the node could have 4 data items instead of the maximum of 3 items.)
- In such a case, the middle data item (either the second or third item in a 2-4 tree) is promoted to the parent node (if a parent doesn't exist, a parent node is created), and this node is split into two nodes.
- Items smaller than the promoted item go to the left child of the promoted item, and items larger than the promoted item go to the right child of the promoted item.

Overflow

- After the data is added to a leaf node, the node may have too many data items for the type of tree. (For example, in the case of a 2-4 tree, the node could have 4 data items instead of the maximum of 3 items.)
- In such a case, the middle data item (either the second or third item in a 2-4 tree) is promoted to the parent node (if a parent doesn't exist, a parent node is created), and this node is split into two nodes.
- Items smaller than the promoted item go to the left child of the promoted item, and items larger than the promoted item go to the right child of the promoted item.
- These slides will promote the third item in a 2-4 tree node.

Adding

For example, if you were adding 742:

$\boxed{293\ 481\ 984}$

Adding

For example, if you were adding 742:

$\boxed{293\ 481\ 984}$

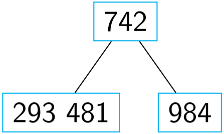
Adding

For example, if you were adding 742:

$\boxed{293\ 481\ 742\ 984}$

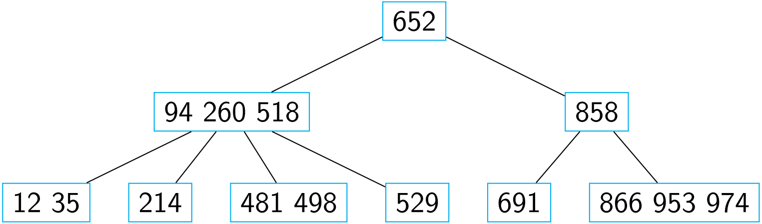
Adding

For example, if you were adding 742:



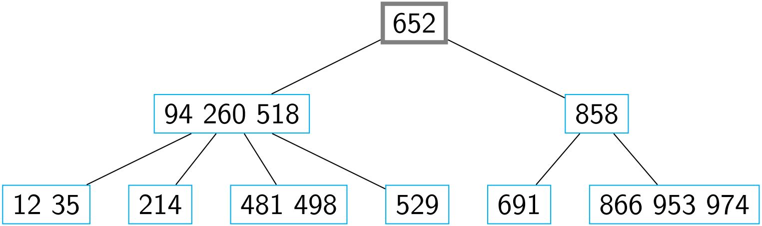
Adding

For example, if you were adding 917:



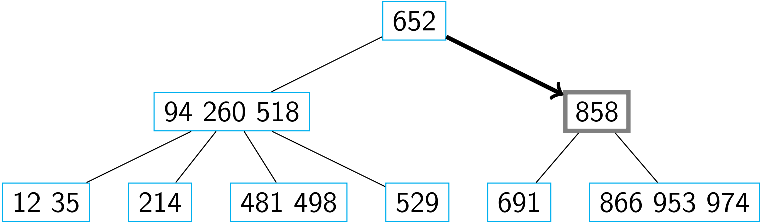
Adding

For example, if you were adding 917:



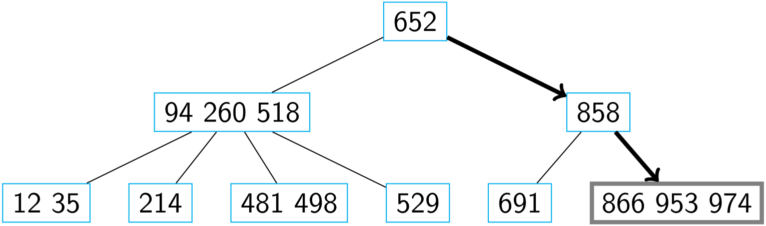
Adding

For example, if you were adding 917:



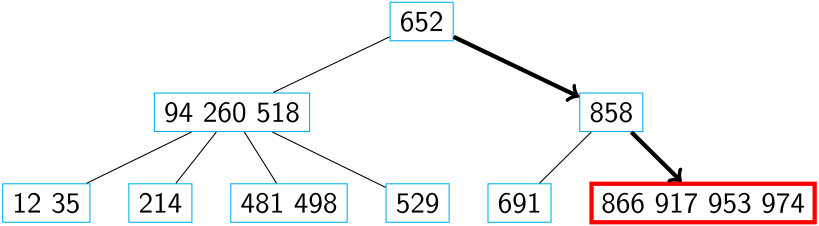
Adding

For example, if you were adding 917:



Adding

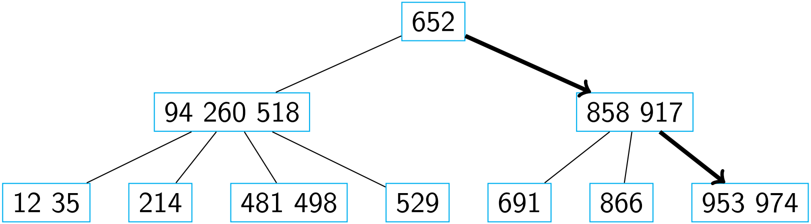
For example, if you were adding 917:



Note how the node has 4 items instead of the maximum of 3 items.

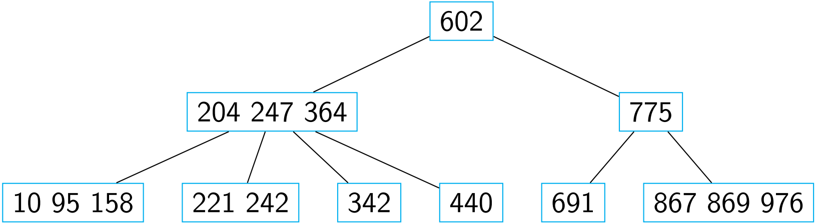
Adding

For example, if you were adding 917:



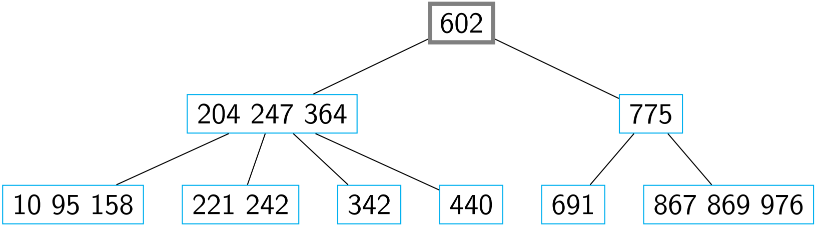
Adding

For example, if you were adding 179:



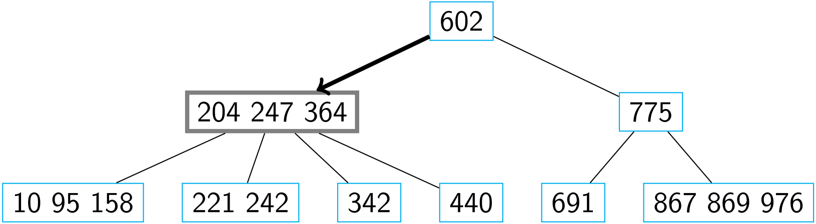
Adding

For example, if you were adding 179:



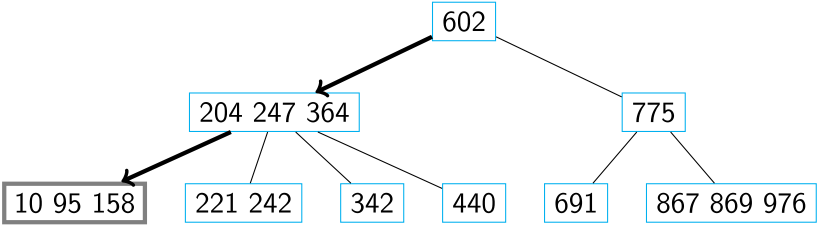
Adding

For example, if you were adding 179:



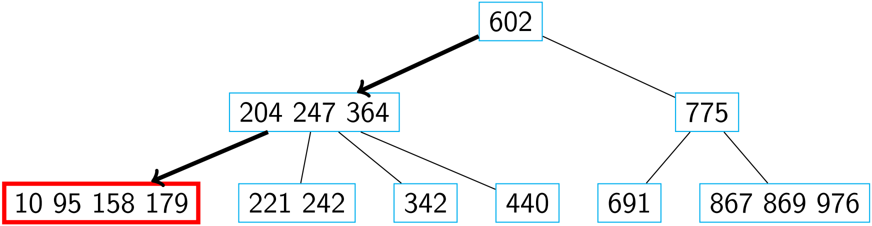
Adding

For example, if you were adding 179:



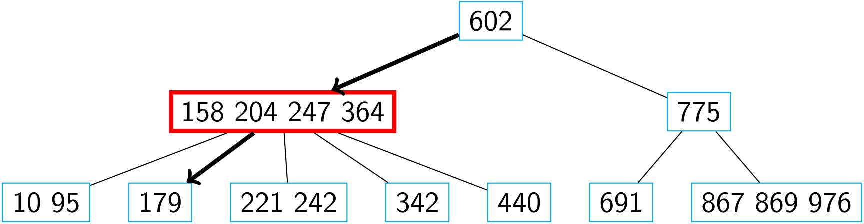
Adding

For example, if you were adding 179:



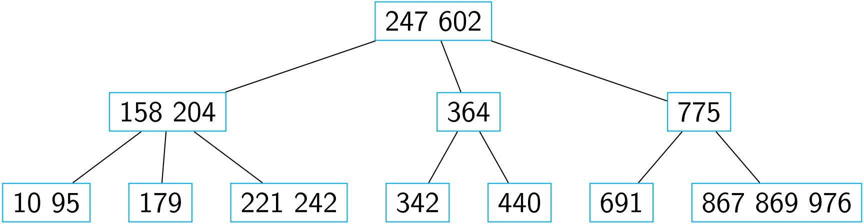
Adding

For example, if you were adding 179:



Adding

For example, if you were adding 179:



Adding

procedure $\textsc{Add}(data, node)$
  for $i \leftarrow 1, n$ **do**
    if $data = node.data[i]$ **then**
      impementation-defined behavior
    else if $data < node.data[i]$ **then**
      if *node* has any children **then**
        $child \leftarrow i^{th}$ child node
        $\textsc{Add}(data, child)$
        if *child* has too many items **then**
          Break *child* up into two nodes, promoting the middle item
        end if
        return

Adding

```
        else
            Add data into this node in the i^th position
            return
        end if
    end if
end for
if node has any children then
    child ← last child node
    ADD(data, child)
    if child has too many items then
        Break child up into two nodes, promoting the middle
item
    end if
```

Adding

else
    Add *data* into the last slot of this node
end if
end procedure

Removing

- Follow the same steps as searching until you find the data in the tree or go off of the tree (in which case the data is not in the tree).

Removing

- Follow the same steps as searching until you find the data in the tree or go off of the tree (in which case the data is not in the tree).
- Once you find the node that contains the data you want to remove, remove the data from the node, and:

Removing

1. If the node has no child nodes and there is at least one other data item in the node, then you're done.

Removing

1. If the node has no child nodes and there is at least one other data item in the node, then you're done.
2. If the node has child nodes, then move either the predecessor or the successor into this node. If there is now an empty node where the predecessor/successor used to be, continue to the next step to fix the empty node.

Removing

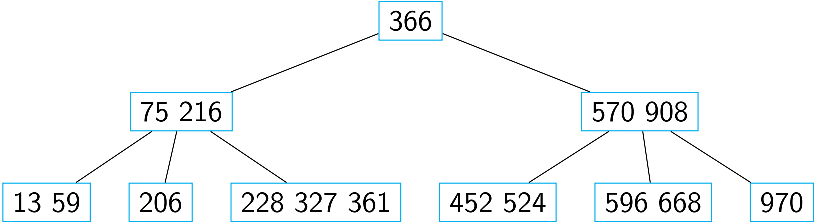
3. If any of the empty node's "sibling" nodes (other nodes that have the same parent) have more than one data item in the node, then "rotate" the data to fill in this node (similar to single rotations done on AVL trees).

Removing

3. If any of the empty node's "sibling" nodes (other nodes that have the same parent) have more than one data item in the node, then "rotate" the data to fill in this node (similar to single rotations done on AVL trees).
4. If all of the empty node's "sibling" nodes have only one data item in each node, then bring down a (reasonable) data item from the parent node and merge this node with either the sibling to the left or the sibling to the right. If this results in an empty/invalid parent node, continue from the third step to fix the parent node.

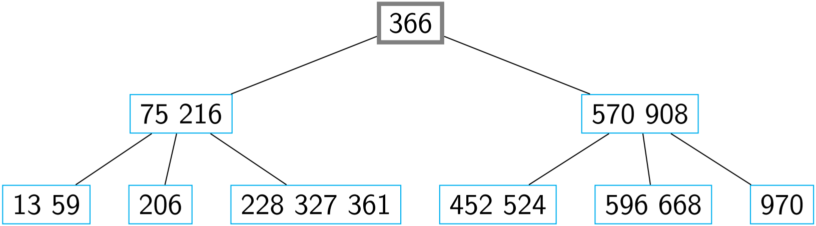
Removing

For example, if you were removing 596:



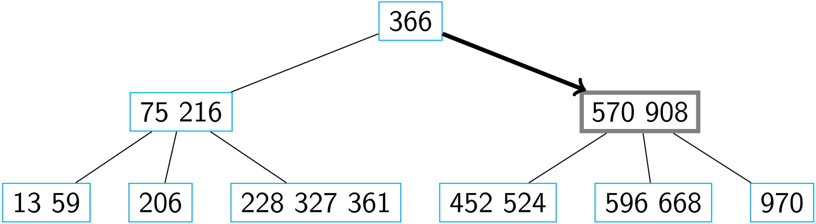
Removing

For example, if you were removing 596:



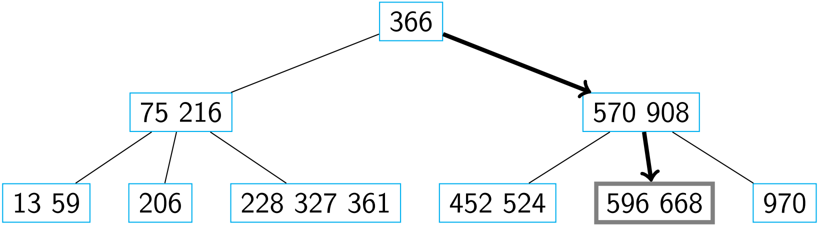
Removing

For example, if you were removing 596:



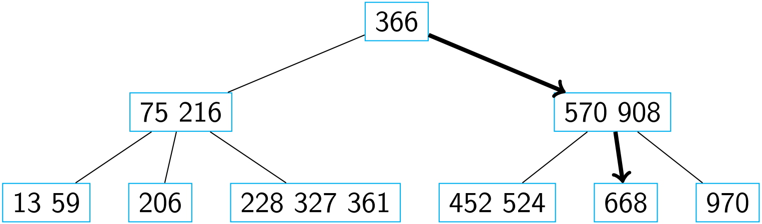
Removing

For example, if you were removing 596:



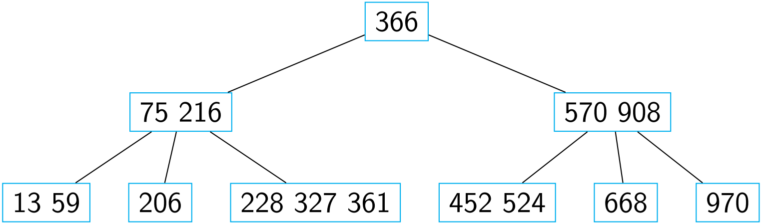
Removing

For example, if you were removing 596:



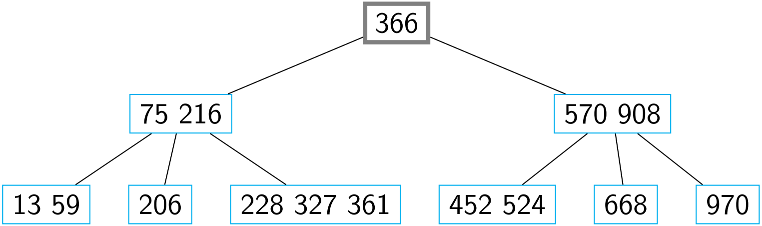
Removing

For example, if you were removing 570 (using the successor):



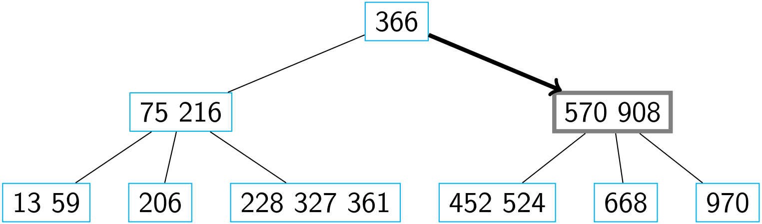
Removing

For example, if you were removing 570 (using the successor):



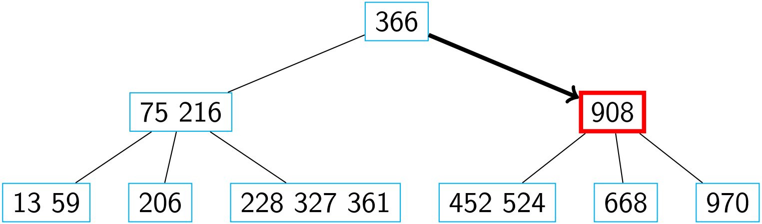
Removing

For example, if you were removing 570 (using the successor):



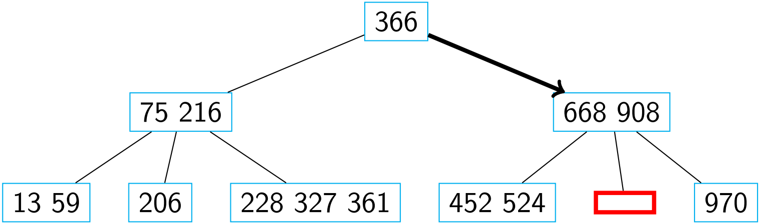
Removing

For example, if you were removing 570 (using the successor):



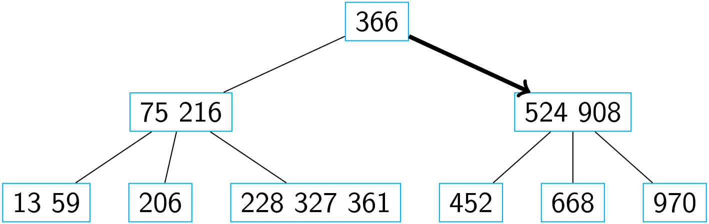
Removing

For example, if you were removing 570 (using the successor):



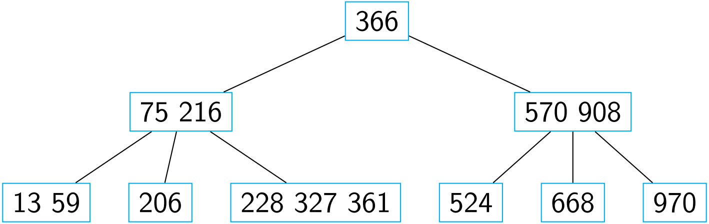
Removing

For example, if you were removing 570 (using the successor):



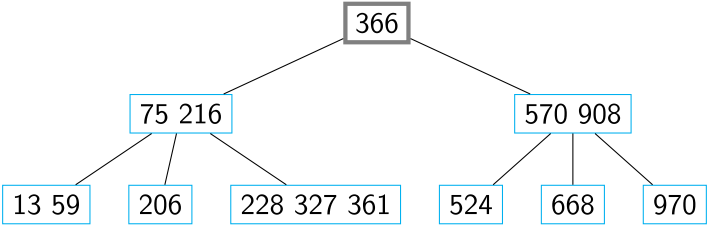
Removing

For example, if you were removing 970:



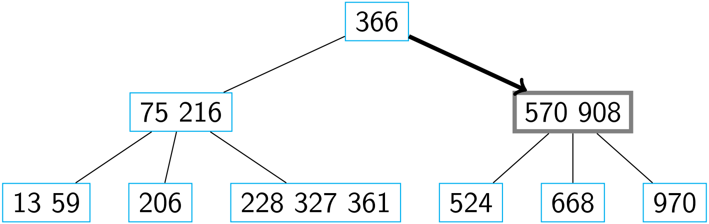
Removing

For example, if you were removing 970:



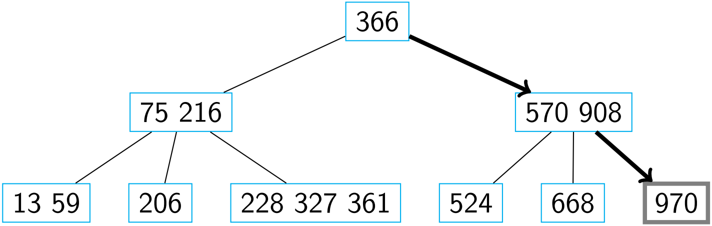
Removing

For example, if you were removing 970:



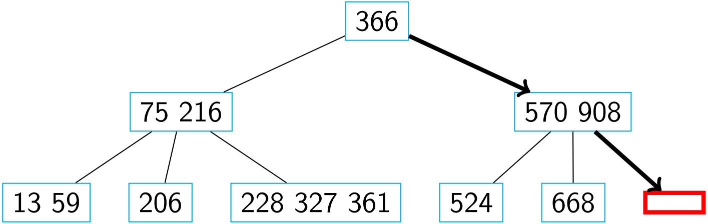
Removing

For example, if you were removing 970:



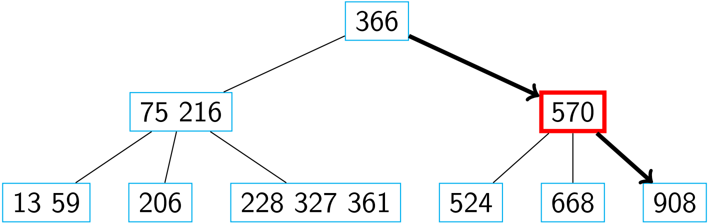
Removing

For example, if you were removing 970:



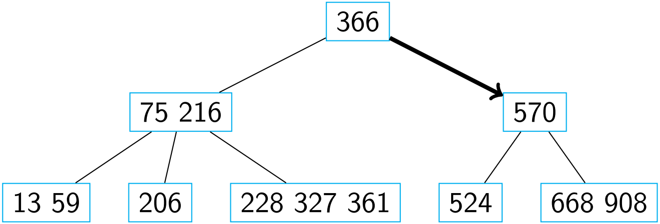
Removing

For example, if you were removing 970:



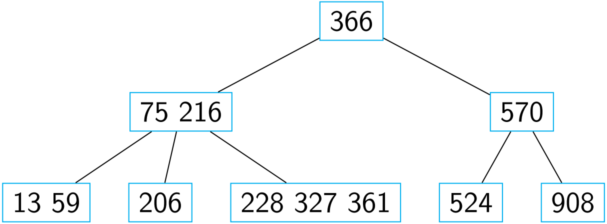
Removing

For example, if you were removing 970:



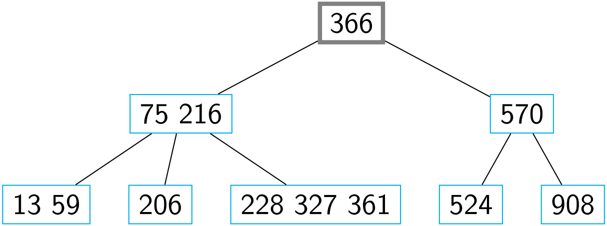
Removing

For example, if you were removing 908:



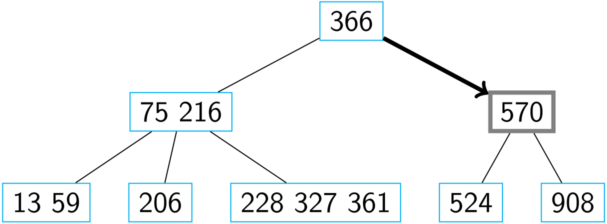
Removing

For example, if you were removing 908:



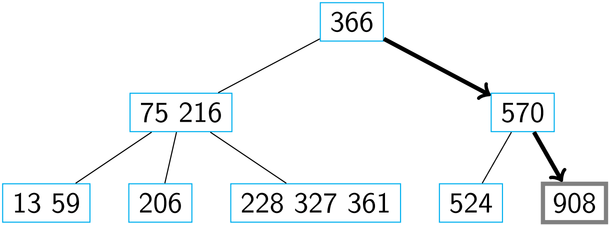
Removing

For example, if you were removing 908:



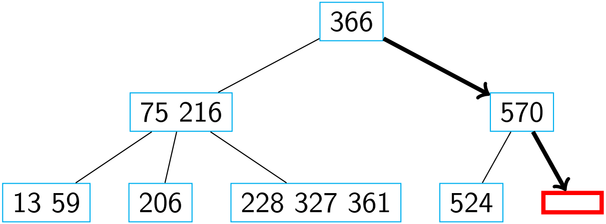
Removing

For example, if you were removing 908:



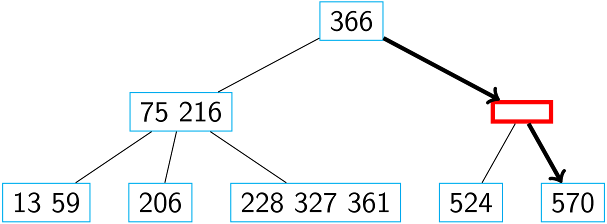
Removing

For example, if you were removing 908:



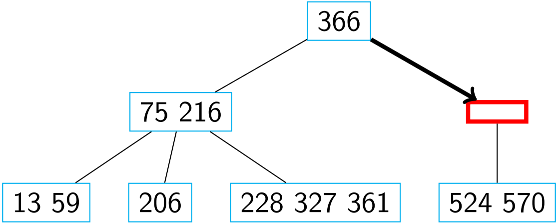
Removing

For example, if you were removing 908:



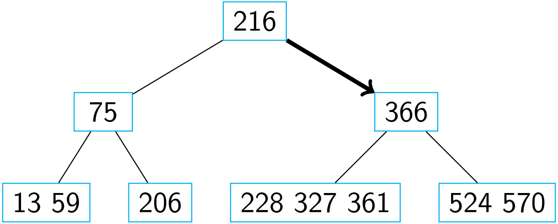
Removing

For example, if you were removing 908:



Removing

For example, if you were removing 908:



Performance

- Because of how items are added into a B-tree (in particular, the depth of all leaf nodes are the same), insertion, search, and deletion are all $O(\log n)$ for average and worst case.

Performance

- Because of how items are added into a B-tree (in particular, the depth of all leaf nodes are the same), insertion, search, and deletion are all $O(\log n)$ for average and worst case.
- In the best case, searching a B-tree is $O(1)$ if the data being searched for is in the root node.