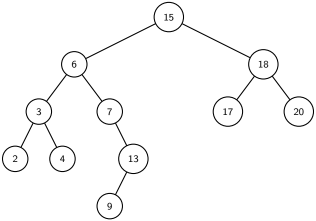
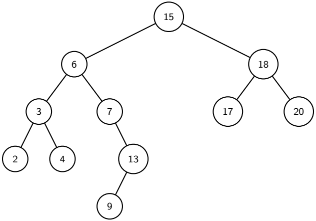
Successor

- The structure of BST allows us to determine the successor of a node without ever comparing keys!



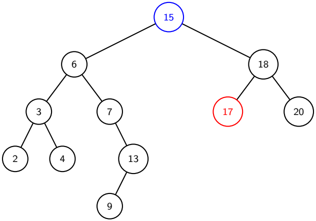
Successor

- The structure of BST allows us to determine the successor of a node without ever comparing keys!
- Two cases may arise:
  - **Case 1:** The right sub-tree of a node x is non-empty.
    - **Successor of x:** The leftmost node in the right sub-tree.
    - $\therefore$ call $\textsc{Tree-Minimum}(\textit{right}[x])$.



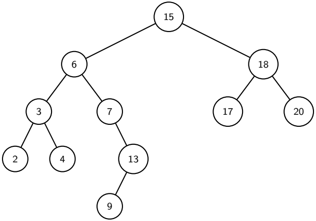
Successor

- The structure of BST allows us to determine the successor of a node without ever comparing keys!
- Two cases may arise:
    - **Case 1:** The right sub-tree of a node x is non-empty.
        - **Successor of x:** The leftmost node in the right sub-tree.
        - $\therefore$ call $\textsc{Tree-Minimum}(\mathit{right}[x])$.
        - **Example:** Successor of 15 is 17.



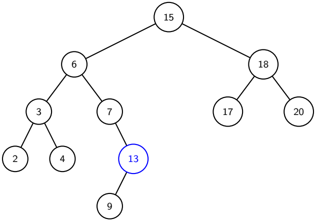
Successor

- The structure of BST allows us to determine the successor of a node without ever comparing keys!
- Two cases may arise:
  - **Case 2:** The right sub-tree of a node x is empty.
    - **Successor of x:** The lowest ancestor of x whose left child is also an ancestor of x.



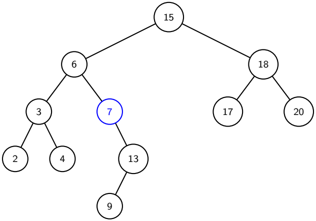
Successor

- The structure of BST allows us to determine the successor of a node without ever comparing keys!
- Two cases may arise:
  - **Case 2:** The right sub-tree of a node x is empty.
    - **Successor of x:** The lowest ancestor of x whose left child is also an ancestor of x.
    - Go up the tree from x until we encounter a node that is the left child of its parent.
    - **Example:** Consider the node 13.



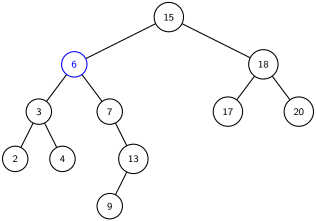
Successor

- The structure of BST allows us to determine the successor of a node without ever comparing keys!
- Two cases may arise:
  - **Case 2:** The right sub-tree of a node x is empty.
    - **Successor of x:** The lowest ancestor of x whose left child is also an ancestor of x.
    - Go up the tree from x until we encounter a node that is the left child of its parent.
    - **Example:** Consider the node 13.



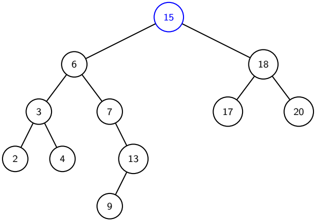
Successor

- The structure of BST allows us to determine the successor of a node without ever comparing keys!
- Two cases may arise:
  - **Case 2:** The right sub-tree of a node x is empty.
    - **Successor of x:** The lowest ancestor of x whose left child is also an ancestor of x.
    - Go up the tree from x until we encounter a node that is the left child of its parent.
    - **Example:** Consider the node 13.



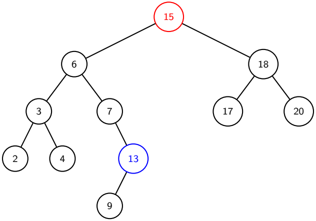
Successor

- The structure of BST allows us to determine the successor of a node without ever comparing keys!
- Two cases may arise:
  - **Case 2:** The right sub-tree of a node x is empty.
    - **Successor of x:** The lowest ancestor of x whose left child is also an ancestor of x.
    - Go up the tree from x until we encounter a node that is the left child of its parent.
    - **Example:** Consider the node 13.



Successor

- The structure of BST allows us to determine the successor of a node without ever comparing keys!
- Two cases may arise:
  - **Case 2:** The right sub-tree of a node x is empty.
    - **Successor of x:** The lowest ancestor of x whose left child is also an ancestor of x.
    - Go up the tree from x until we encounter a node that is the left child of its parent.
    - Then this parent is the successor.
    - **Example:** The successor of 13 is 15.

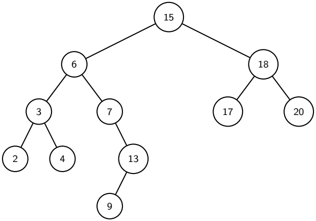


Tree-Successor

```
Tree-Successor($x$)
  I/P: A node $x$ whose successor we need to find.
  O/P: The successor of $x$.
  Begin
    if ($right[x] \neq \textbf{nil}$)
      return Tree-Minimum($right[x]$);
    $y \leftarrow parent[x]$;
    while ($y \neq \textbf{nil}$) and ($x = right[y]$)
      $x \leftarrow y$;
      $y \leftarrow parent[y]$;
    return $y$;
  End
```

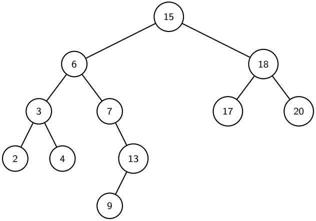
Predecessor

- The structure of BST allows us to determine the Predecessor of a node without ever comparing keys!



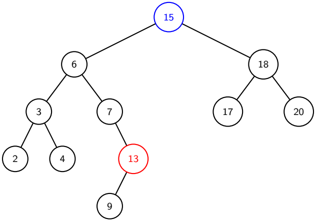
Predecessor

- The structure of BST allows us to determine the Predecessor of a node without ever comparing keys!
- Two cases may arise:
  - **Case 1:** The left sub-tree of a node x is non-empty.
    - **Predecessor of x:** The rightmost node in the left sub-tree.
    - $\therefore$ call $\textsc{Tree-Maximum}(\textit{left}[x])$.



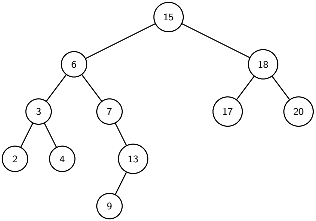
Predecessor

- The structure of BST allows us to determine the Predecessor of a node without ever comparing keys!
- Two cases may arise:
  - **Case 1:** The left sub-tree of a node x is non-empty.
    - **Predecessor of x:** The rightmost node in the left sub-tree.
    - $\therefore$ call $\textsc{Tree-Maximum}(\mathit{left}[x])$.
    - **Example:** Predecessor of 15 is 13.



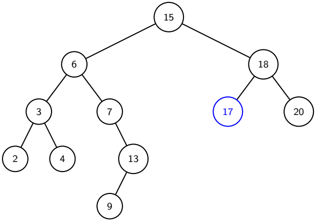
Predecessor

- The structure of BST allows us to determine the Predecessor of a node without ever comparing keys!
- Two cases may arise:
  - **Case 2:** The left sub-tree of a node x is empty.
    - **Predecessor of x:** The lowest ancestor of x whose right child is also an ancestor of x.



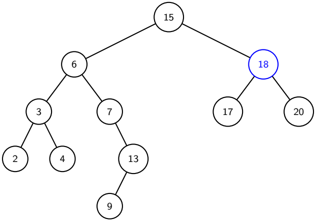
Predecessor

- The structure of BST allows us to determine the Predecessor of a node without ever comparing keys!
- Two cases may arise:
  - **Case 2:** The left sub-tree of a node x is empty.
    - **Predecessor of x:** The lowest ancestor of x whose right child is also an ancestor of x.
    - Go up the tree from x until we encounter a node that is the right child of its parent.
    - **Example:** Consider the node 17.



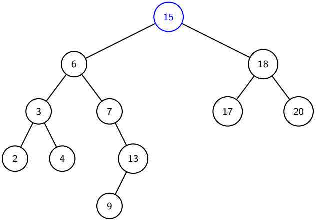
Predecessor

- The structure of BST allows us to determine the Predecessor of a node without ever comparing keys!
- Two cases may arise:
  - **Case 2:** The left sub-tree of a node x is empty.
    - **Predecessor of x:** The lowest ancestor of x whose right child is also an ancestor of x.
    - Go up the tree from x until we encounter a node that is the right child of its parent.
    - **Example:** Consider the node 17.



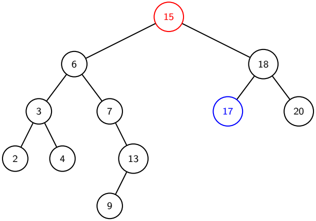
Predecessor

- The structure of BST allows us to determine the Predecessor of a node without ever comparing keys!
- Two cases may arise:
  - **Case 2:** The left sub-tree of a node x is empty.
    - **Predecessor of x:** The lowest ancestor of x whose right child is also an ancestor of x.
    - Go up the tree from x until we encounter a node that is the right child of its parent.
    - **Example:** Consider the node 17.



Predecessor

- The structure of BST allows us to determine the Predecessor of a node without ever comparing keys!
- Two cases may arise:
  - **Case 2:** The left sub-tree of a node x is empty.
    - **Predecessor of x:** The lowest ancestor of x whose right child is also an ancestor of x.
    - Go up the tree from x until we encounter a node that is the right child of its parent.
    - Then this parent is the predecessor.
    - **Example:** The Predecessor of 17 is 15.



Tree-Predecessor

```
Tree-Predecessor($x$)
    I/P: A node $x$ whose predecessor we need to find.
    O/P: The predecessor of $x$.
    Begin
        if ($left[x] \neq$ nil)
            return Tree-Maximum($left[x]$);
        $y \leftarrow parent[x]$;
        while ($y \neq$ nil) and ($x = left[y]$)
            $x \leftarrow y$;
            $y \leftarrow parent[y]$;
        return $y$;
    End
```