Notes lecture 12

Binary tree (continued)

Binary search tree

*Binary tree representations:*

* using an array (similar to a binary heap)
  + elements represented on an array in a level-order way

A diagram of a diagram

Description automatically generated

* Linked representation
  + with dynamic allocation
    - node
      * info
      * left
      * right
  + on an array (like the linked-list on array)

A white sheet with black text

Description automatically generated with medium confidence

*Binary tree traversals:*

* Pre-order = root + left + right

A diagram of a tree

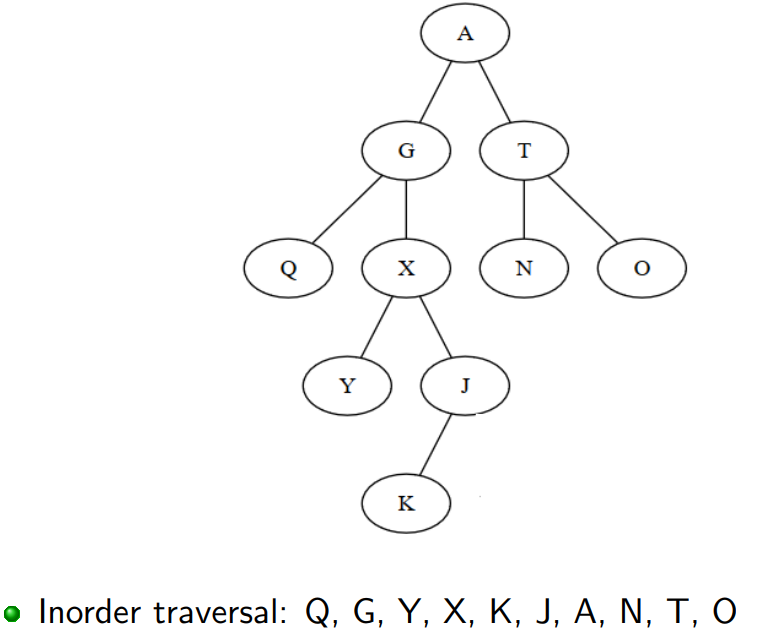
Description automatically generated

* + Iterative implementation:

A screenshot of a computer code

Description automatically generated

* In-order = left + root + right



* + Iterative implementation:

A screenshot of a computer code

Description automatically generated

* Post-order = left + right + root

A diagram of a tree

Description automatically generated

* Level-order = BFS

A diagram of a tree

Description automatically generated

*Binary search trees:*

* Binary trees with the property that for any node x:
  + left\_subtree(x)
  + right\_subtree(x)
* If we do an in-order traversal of a binary search tree, we will get the elements in increasing order

A diagram of a tree

Description automatically generated

* Used as representation for sorted containers

A close-up of a white box

Description automatically generated

|  |
| --- |
| * **Function** successor(tree, node) **is:**   + If [node].right is different from NIL     - “leftest”, lowest descendent of [node].right     - [node].right if it has no left descendant   + Else     - lowest ancestor that has node on its left branch     - NIL if no ancestor has node on its left branch |

|  |
| --- |
| * **Function** remove(tree, node) **is:**   + The node to be removed has no descendant     - Set the corresponding child of the parent to NIL   + The node to be removed has one descendant     - Replace it with its child   + The node to be removed has two descendants     - Replace it with the maximum of the left subtree/minimum of the right subtree |

* To implement a SortedList on a BST keep
  + for each node the number of left descendants
  + this gives us the position of the root in the tree
  + when we have operations based on positions, we use these values to decide if we go left or right