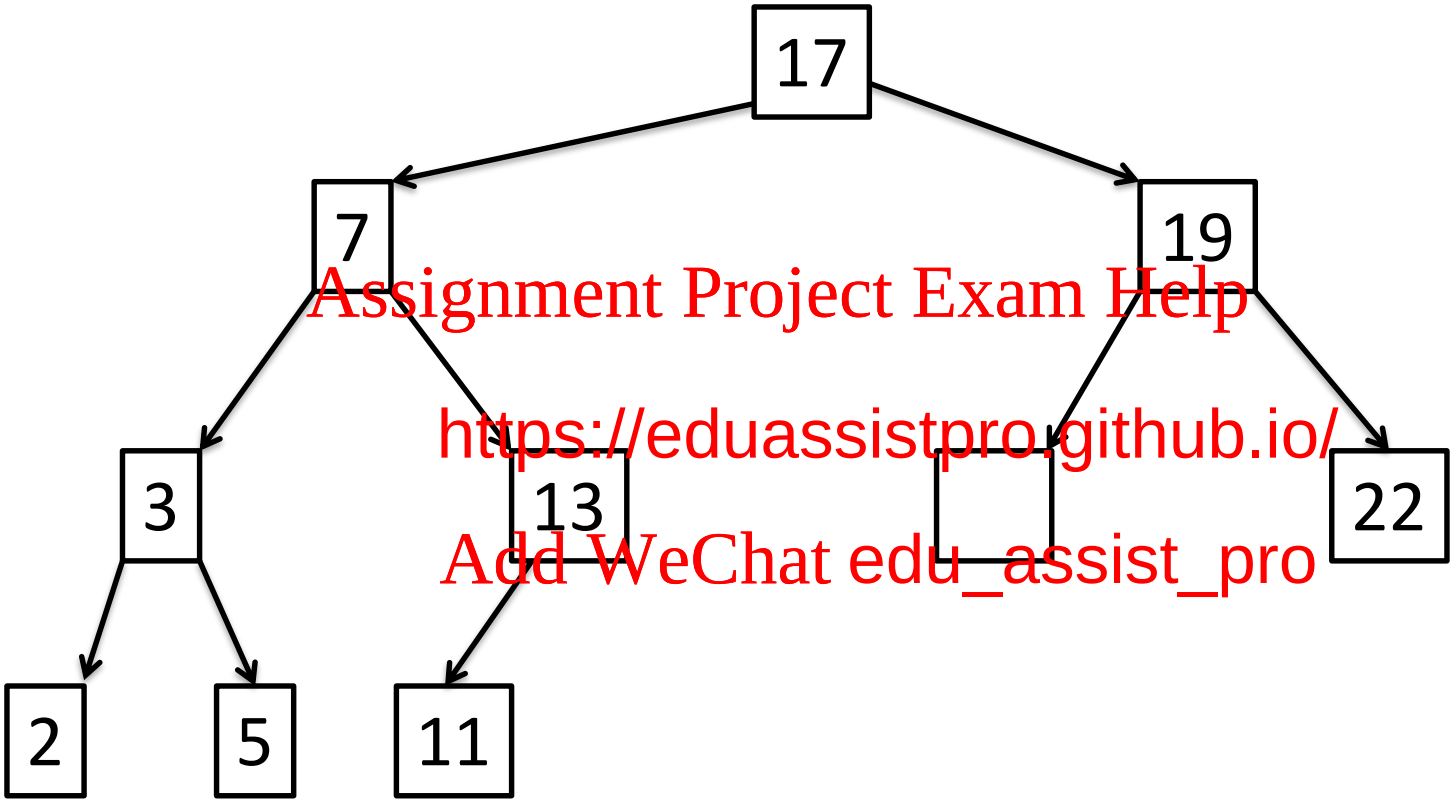
Data Structures and Algorithms

Lecture 10: Binary es

Overview

- Trees
- Sorted Sequences
- Binary Sear

Tree



Representing Trees

- Heaps are special trees (can be stored efficiently in an array)
- Want to ha epresentation of trees
- Each node stores
  - an element
  - has a pointer to the left subtree (might be empty)
  - has a point to the right subtree (might be empty)

TreeItem

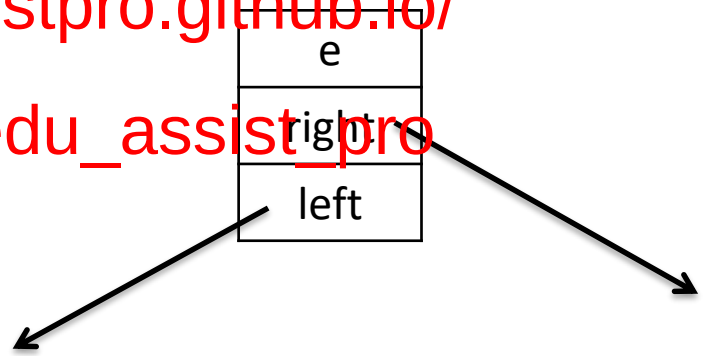
Class Handle = **Pointer to** TreeItem

Class TreeIte

e: Element

right: Handle

left: Handle



Tree Traversal

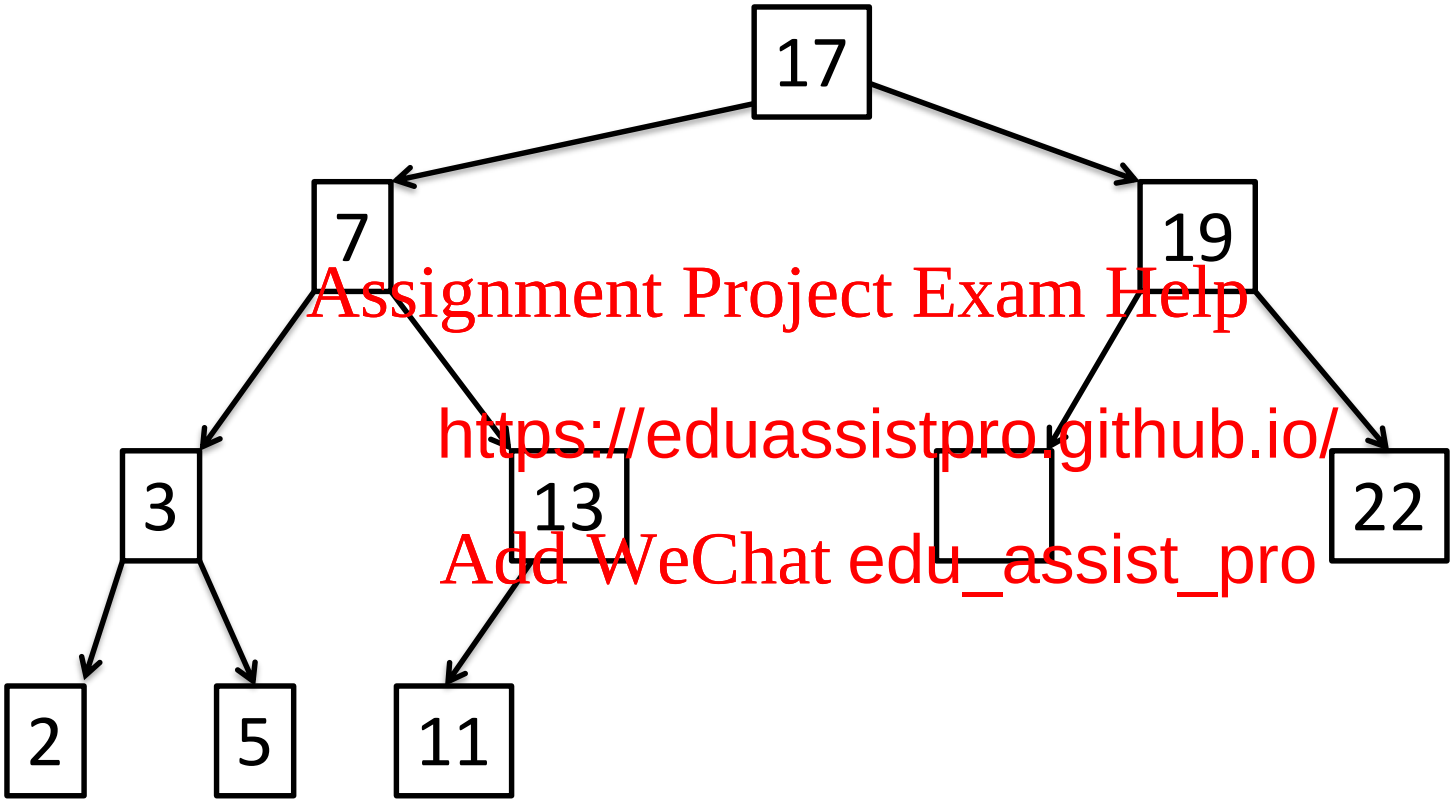
- Want to visit every node in the tree (and print out the elements).
- Recursive f traversal

Preorder Traversal

Preorder(Tree T)

1. Visit the root (and print out the element)
2. If (T->left ! eft)
3. If (T->right t)

Preorder Traversal



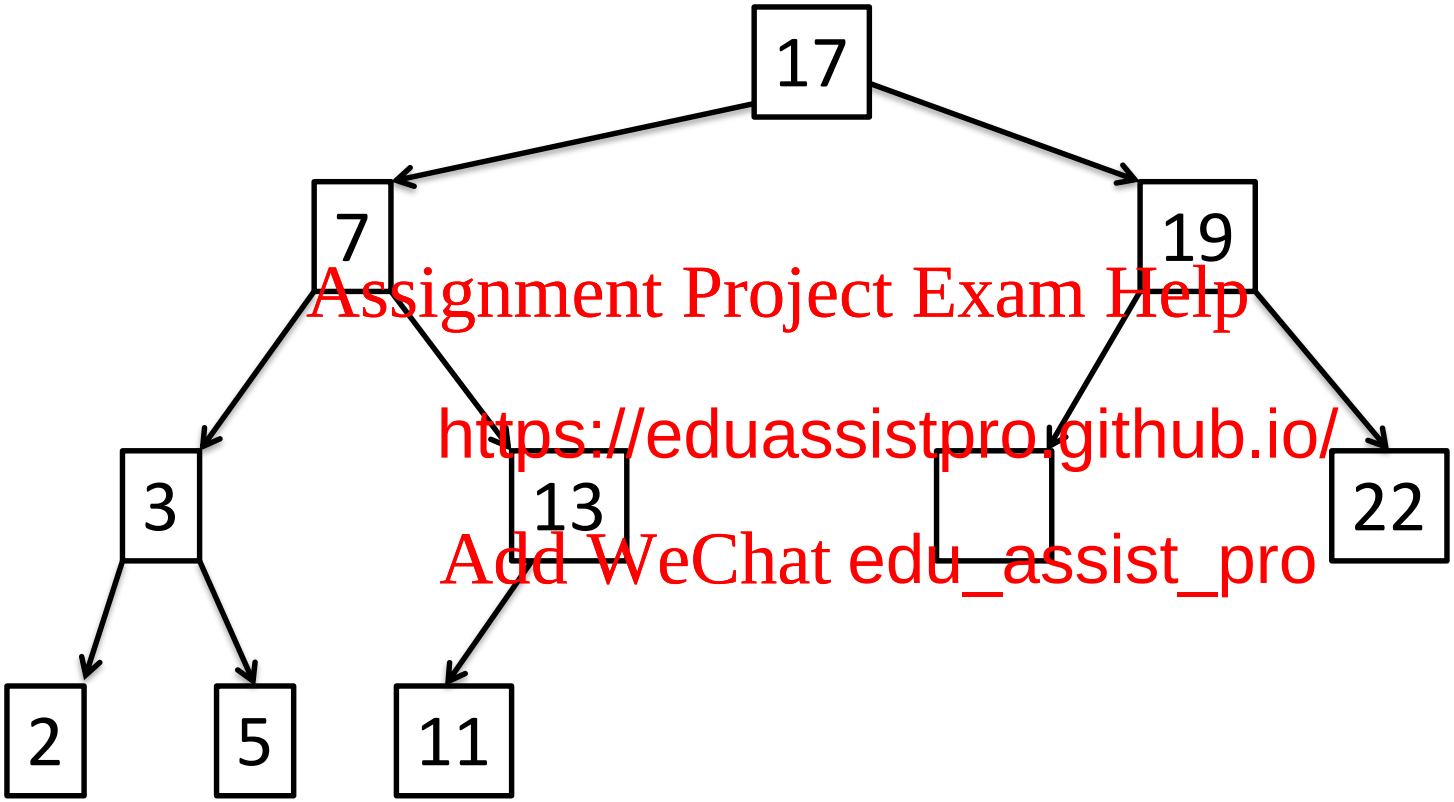
Order nodes are visited: 17, 7, 3, 2, 5, 13, 11, 19, 18, 22

Postorder Traversal

Postorder(Tree T)

1. If (T->left !=null) Postorder(T->left)
2. If (T->right >right)
3. Visit the ro e element)

Postorder Traversal



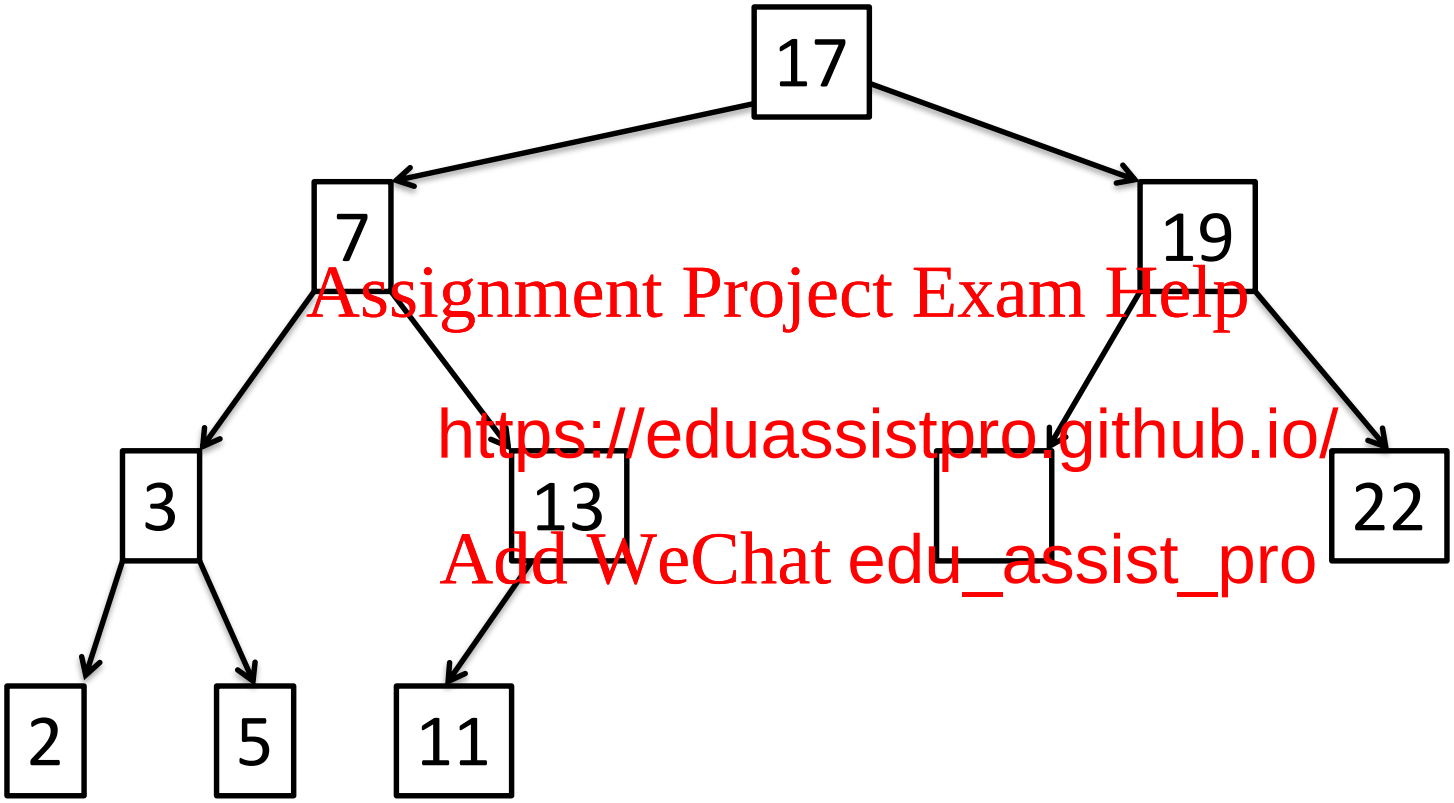
Order nodes are visited: 2, 5, 3, 11, 13, 7, 18, 22, 19, 17

Inorder Traversal

Inorder(Tree T)

1. If (T->left !=null) Inorder(T->left)
2. Visit the ro e element)
3. If (T->right

Inorder Traversal



Order nodes are visited: 2, 3, 5, 7, 11, 13, 17, 18, 19, 22

Observation: This sequence is sorted

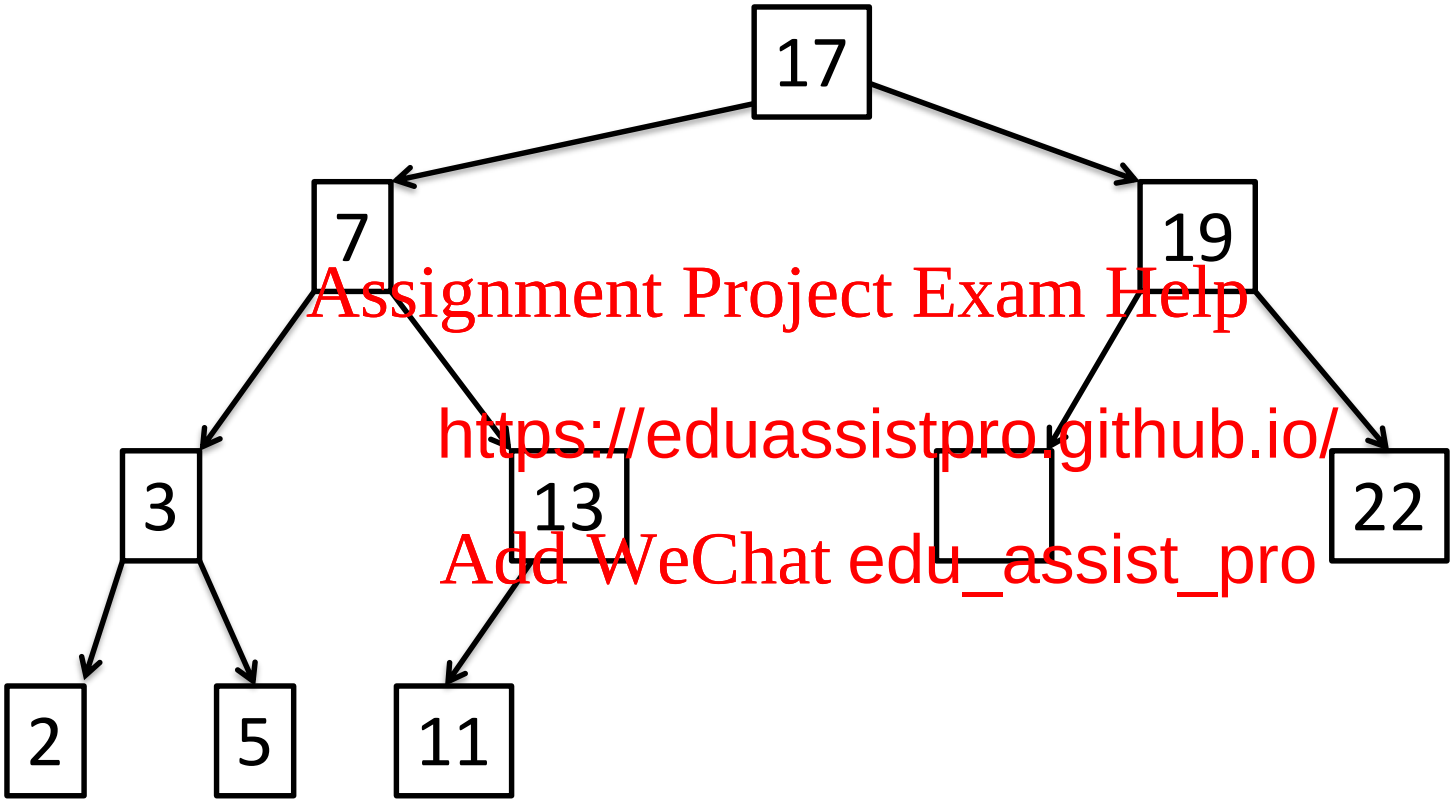
Sorted Sequences

Operations for Sorted Sequences

- Find an element e in the sorted sequence
- Insert an ele ted sequence
- Delete an el orted sequence.

Want to have all these operations implemented in time O(log n).

Binary Search Tree



Sorted sequence by Inorder Traversal: 2, 3, 5, 7, 11, 13, 17, 18, 19, 22

Properties of Binary Search Trees

- All elements in the left subtree of a node k have value smaller than k.
- All element e of a node k have value l

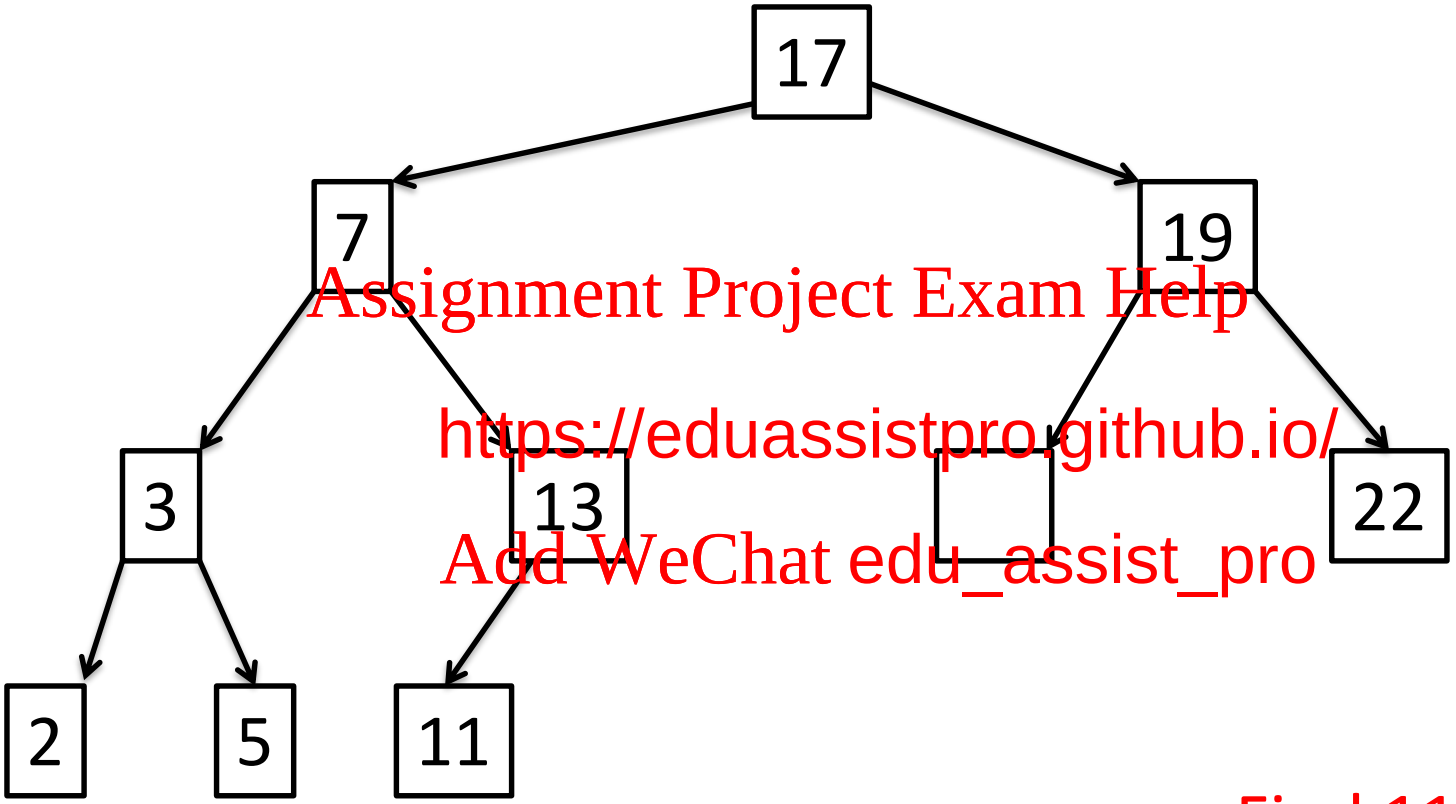
Method find

find(k)

- Start at the root.
- At a node x,

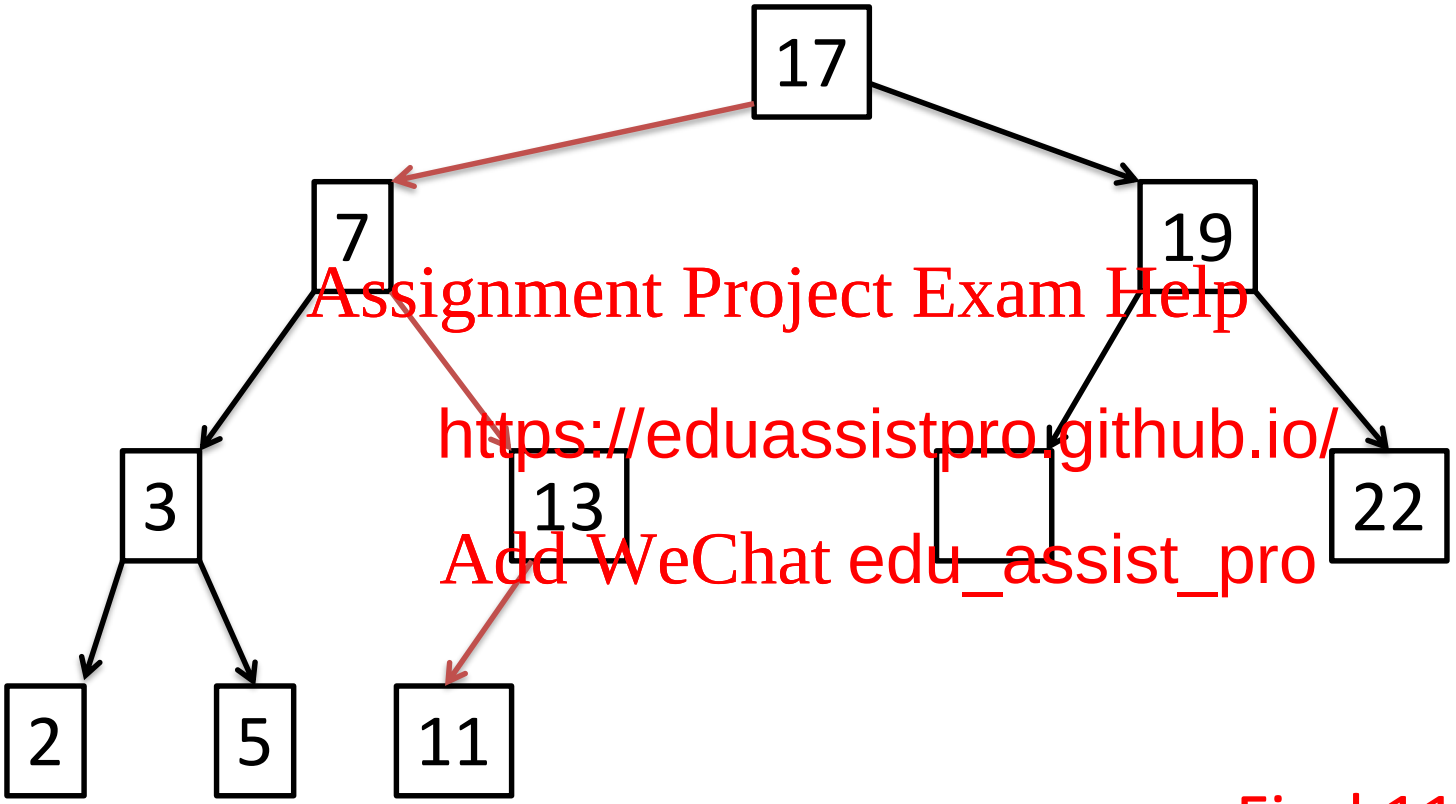
1. If k=x, then
2. If k < x, search in the le e of x. If subtree does not exist return not found.
3. If k > x, search in the right subtree of x. If subtree does not exist, return not found

Find



Find 11

Find



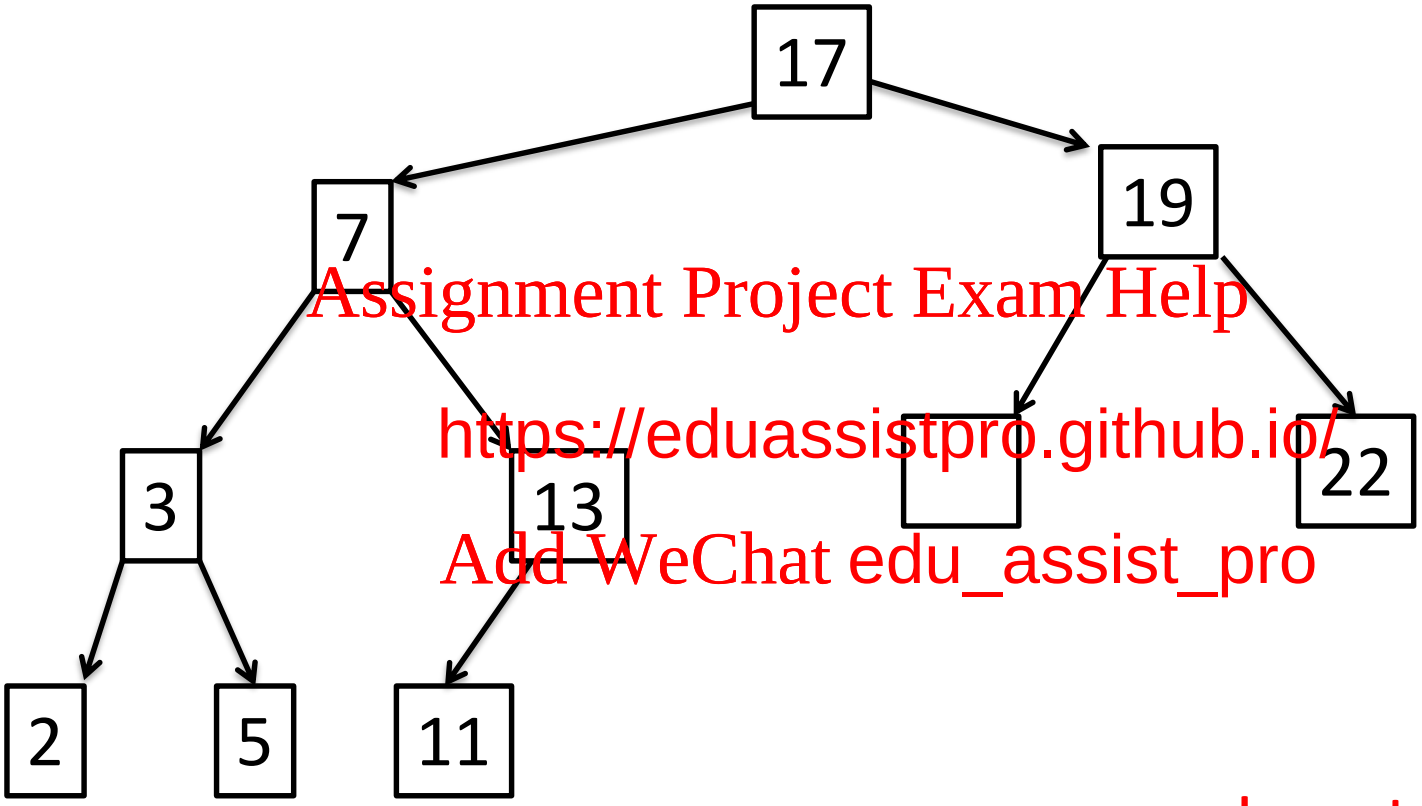
Find 11

Insertion

insert(k)

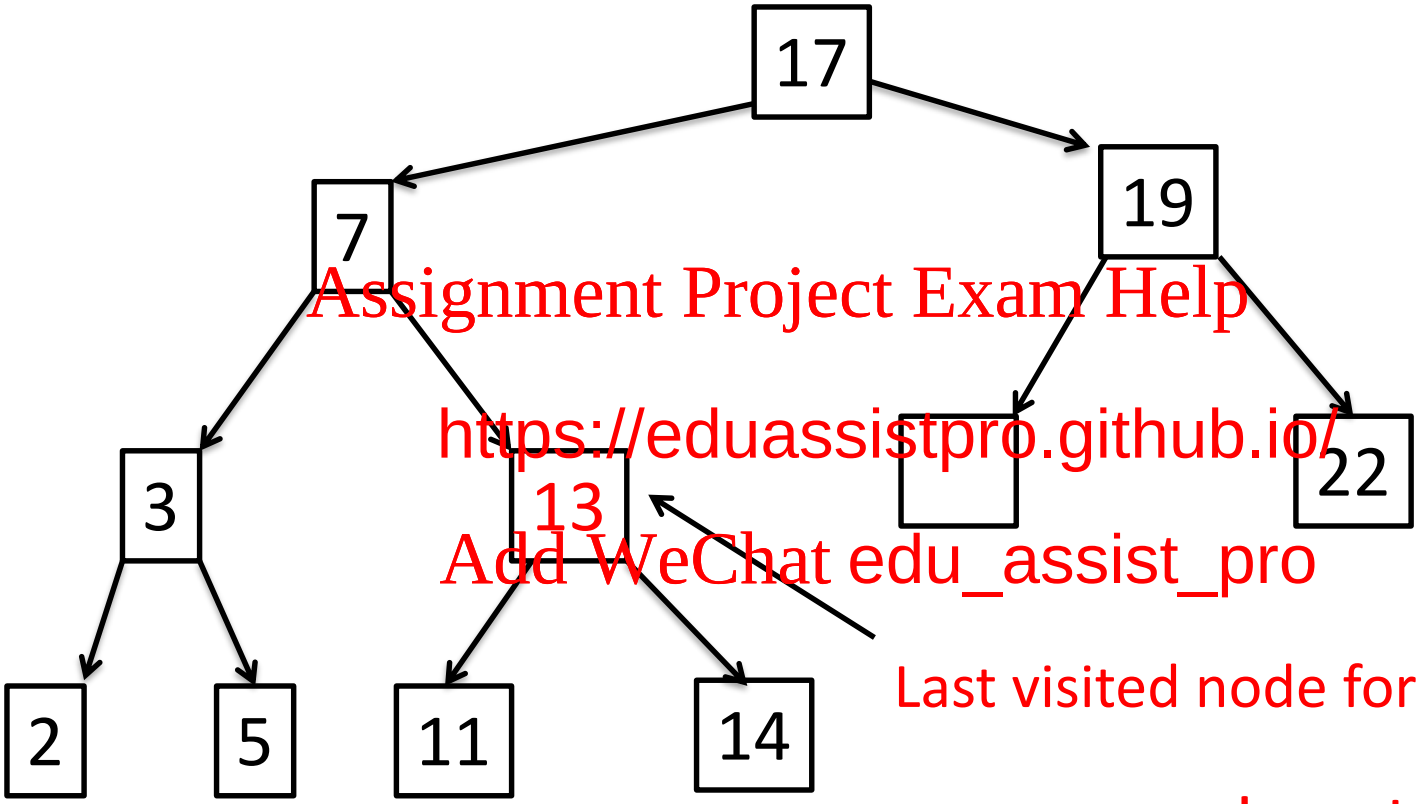
1. find(k)
2. If not foun k becomes child of the f find(k).

Insert



Insert 14

Insert



Last visited node for find(14)

Insert 14

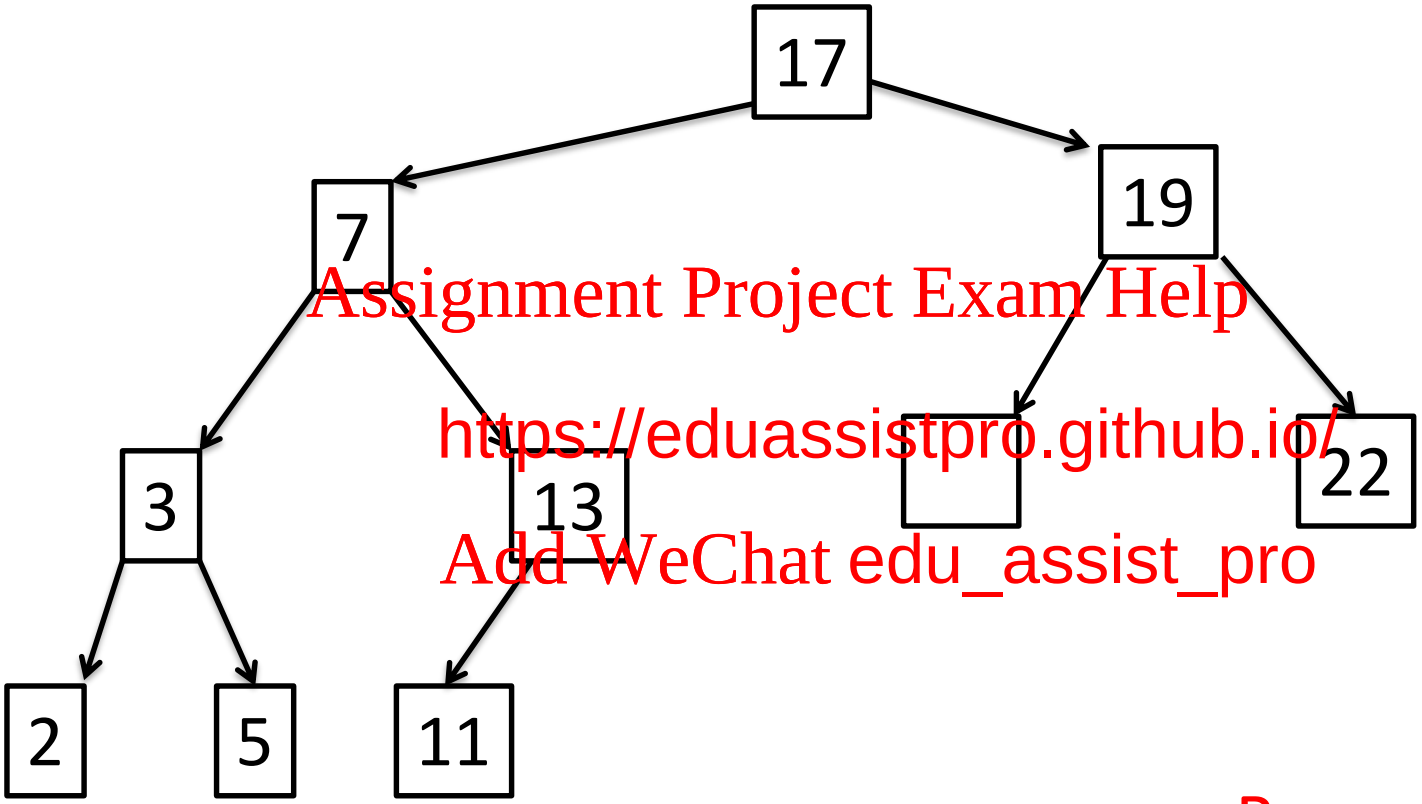
Remove

remove(k)

1. find(k)

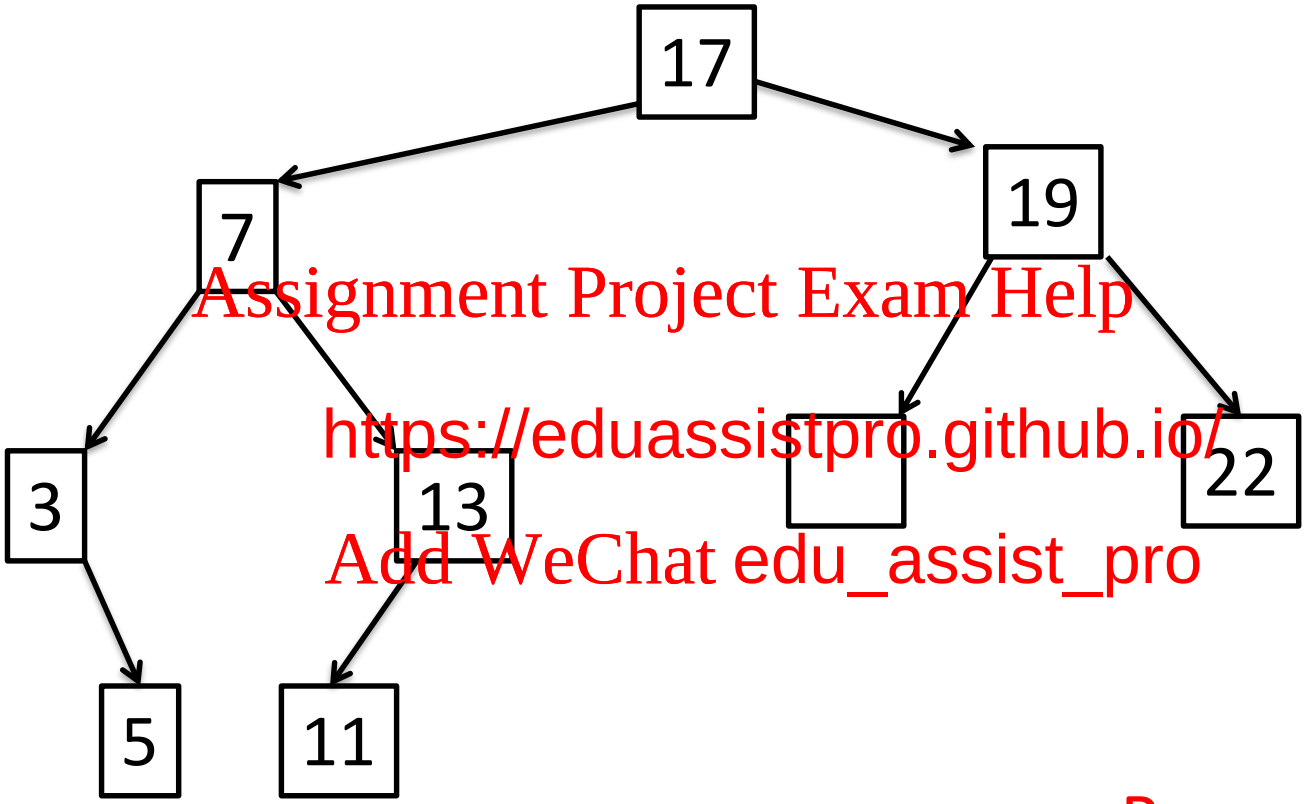
- If k is stored at a leaf, delete this leaf and the incoming ed
- If k has one c er pointing to k to x and delete k
- If k has two children
  - search in the tree for the largest element x smaller than k.
  - swap x and k and delete(k)

Remove leaf



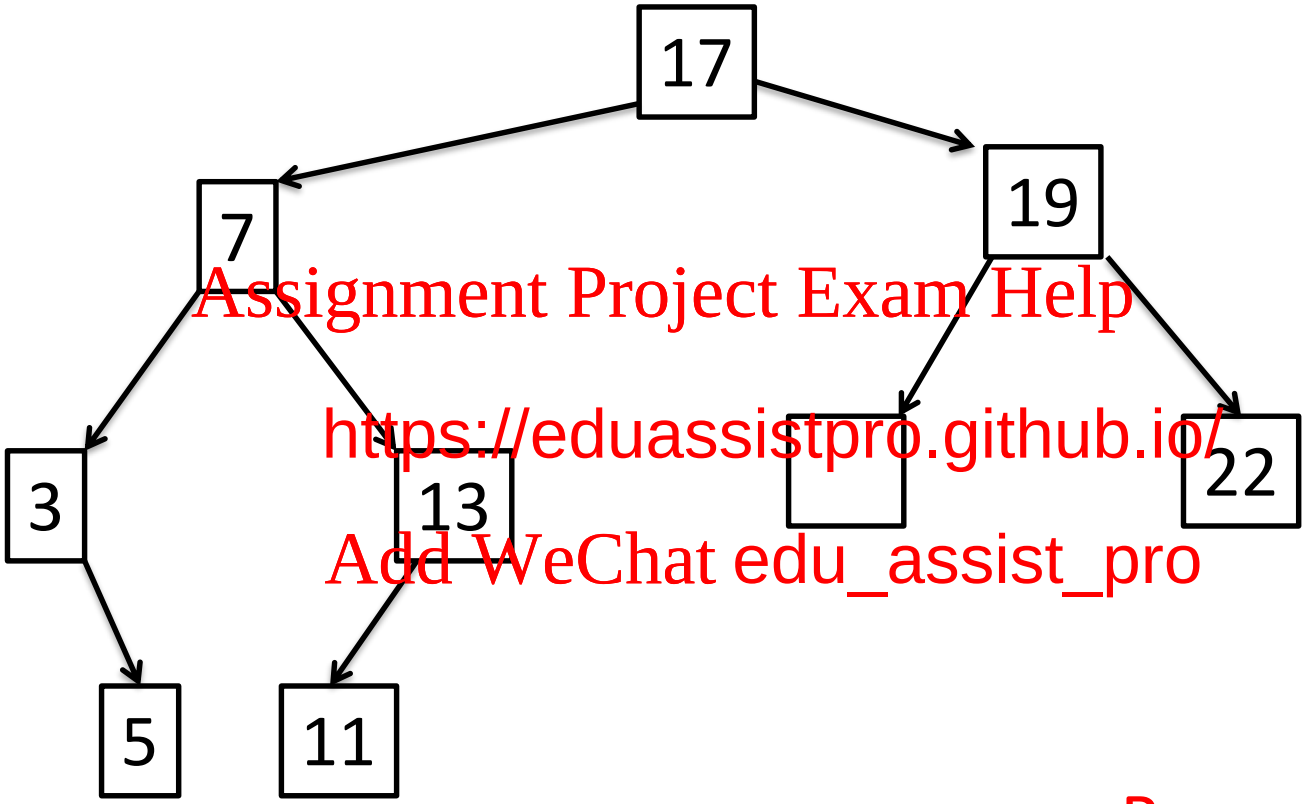
Remove 2

Remove leaf



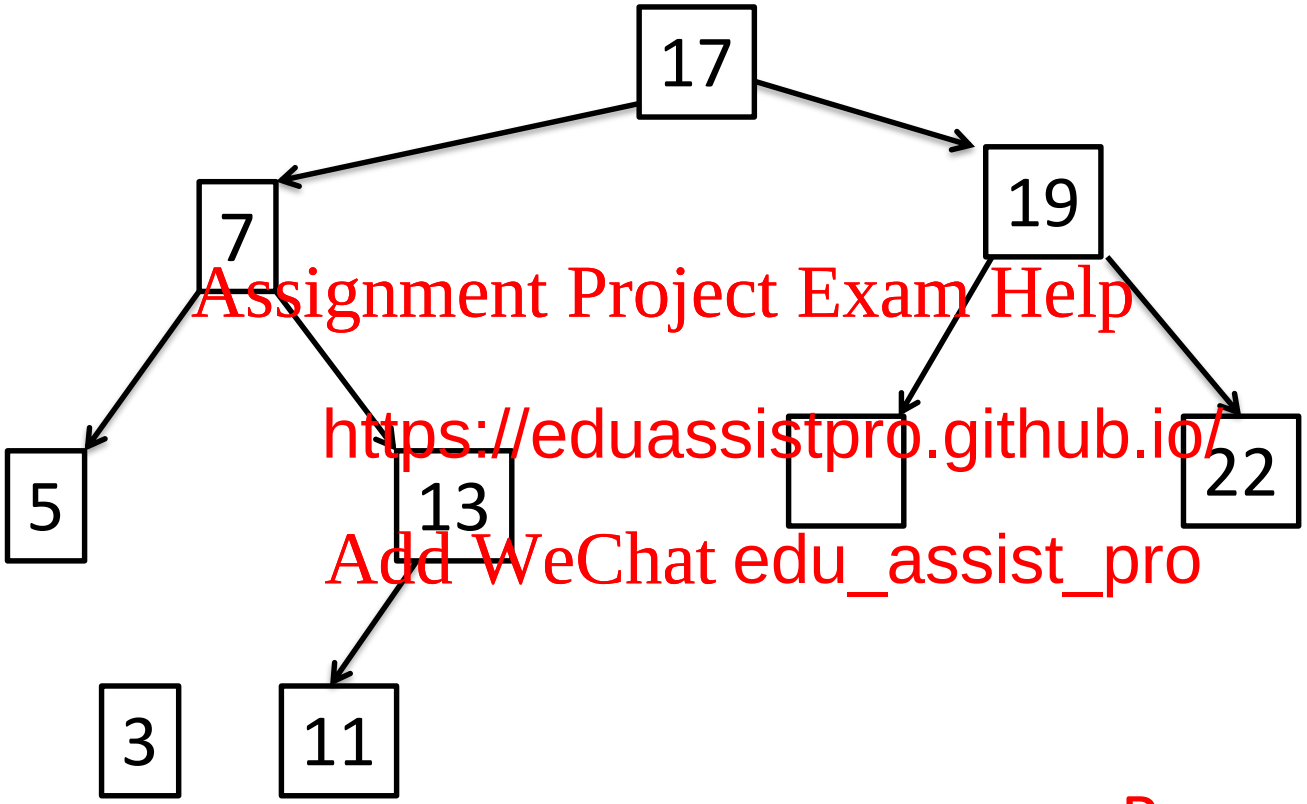
Remove 2

Remove node with 1 child



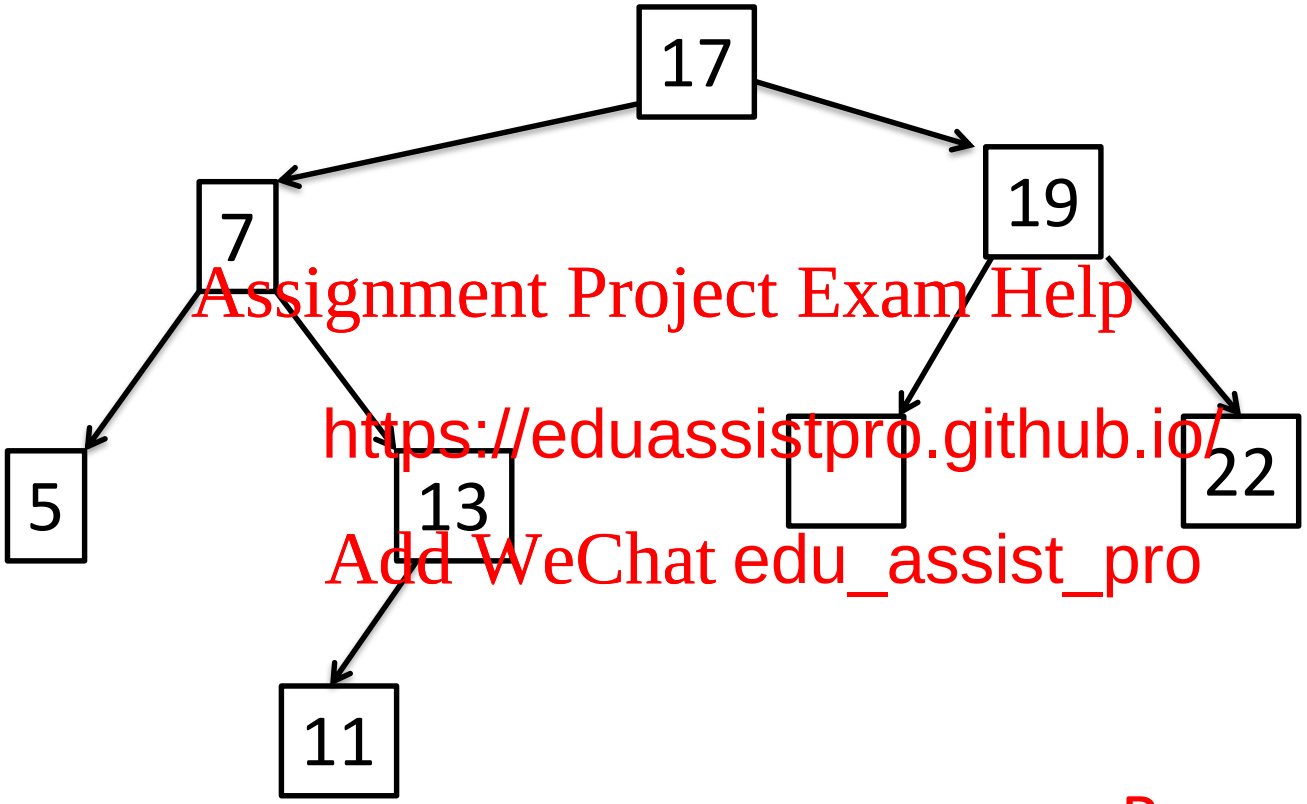
Remove 3

Remove node with 1 child



Remove 3

Remove node with 1 child



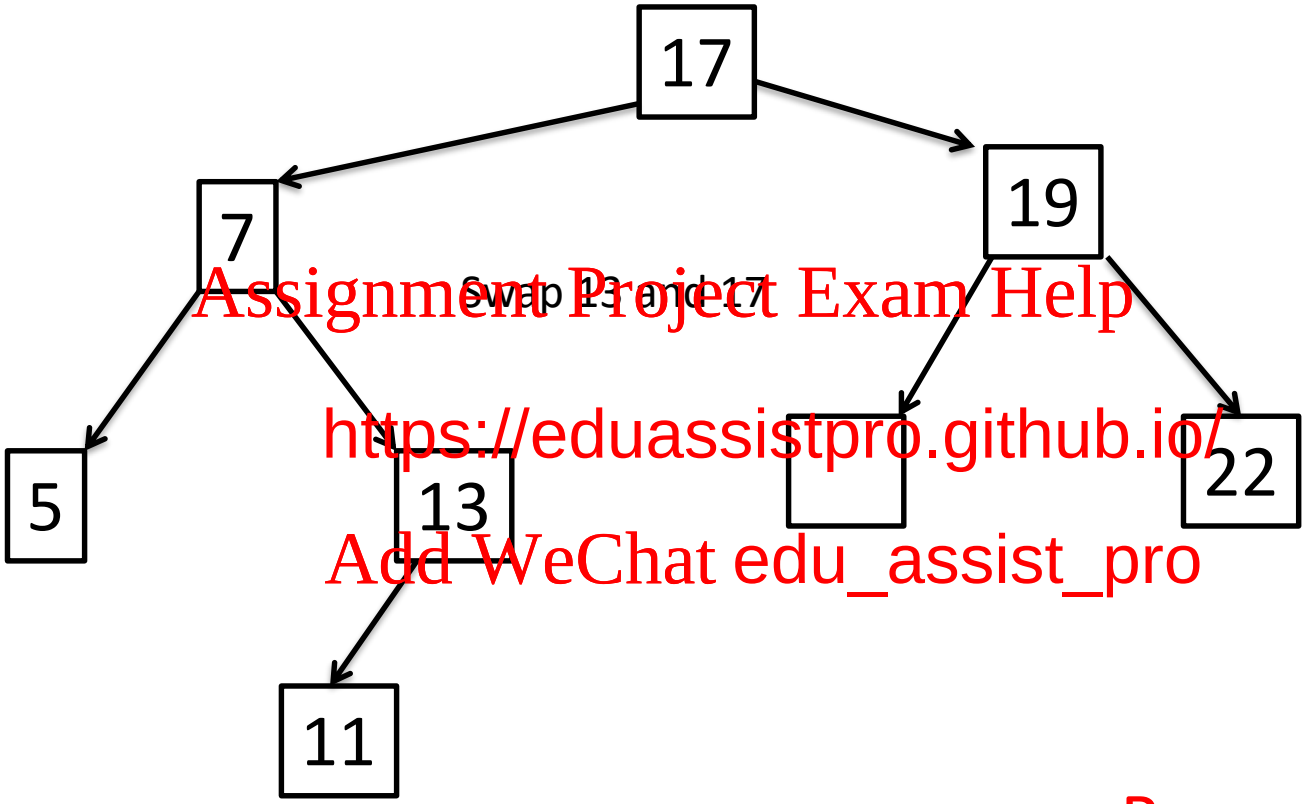
Remove 3

Remove node with 2 children

How to find the largest element smaller than k?

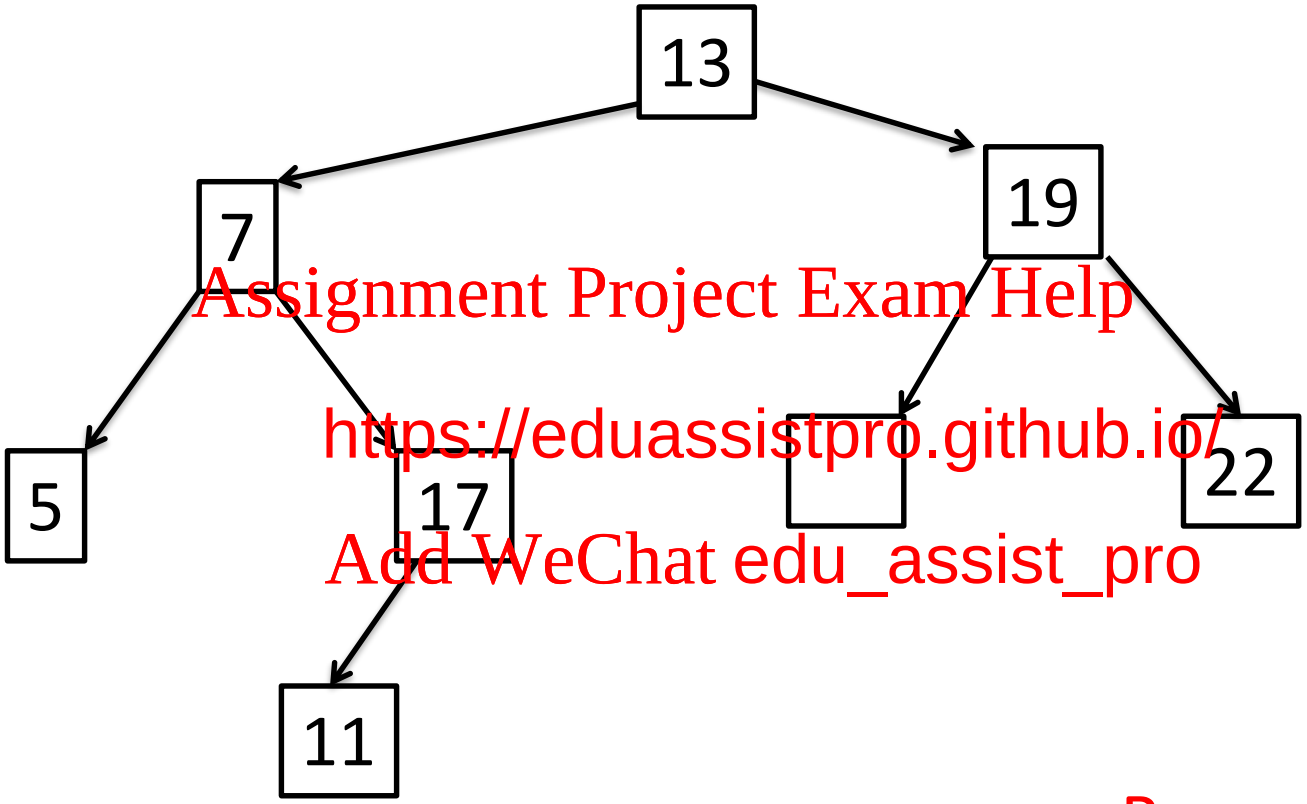
- Go to the left child of k (has to exist as k has two childre
- Follow the as long as possible.

Remove node with 2 children



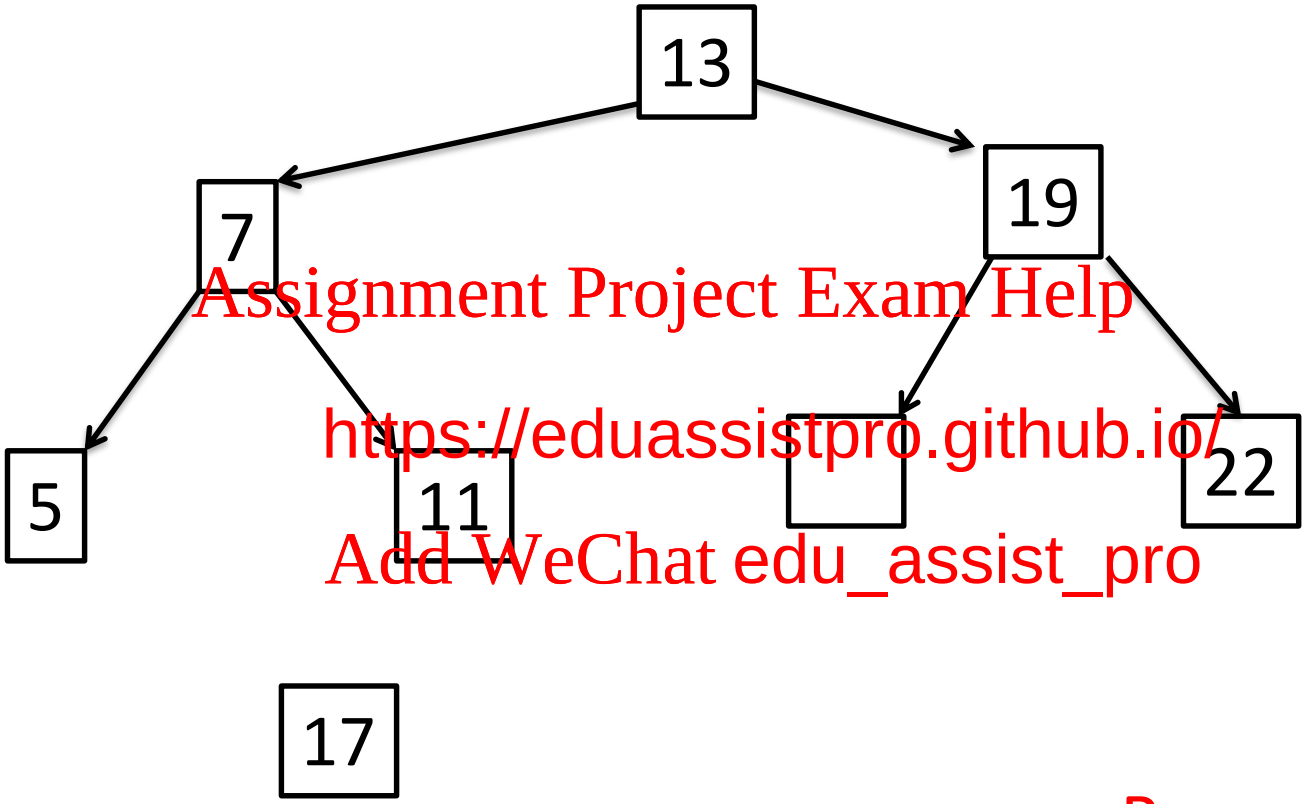
Remove 17

Remove node with 2 children



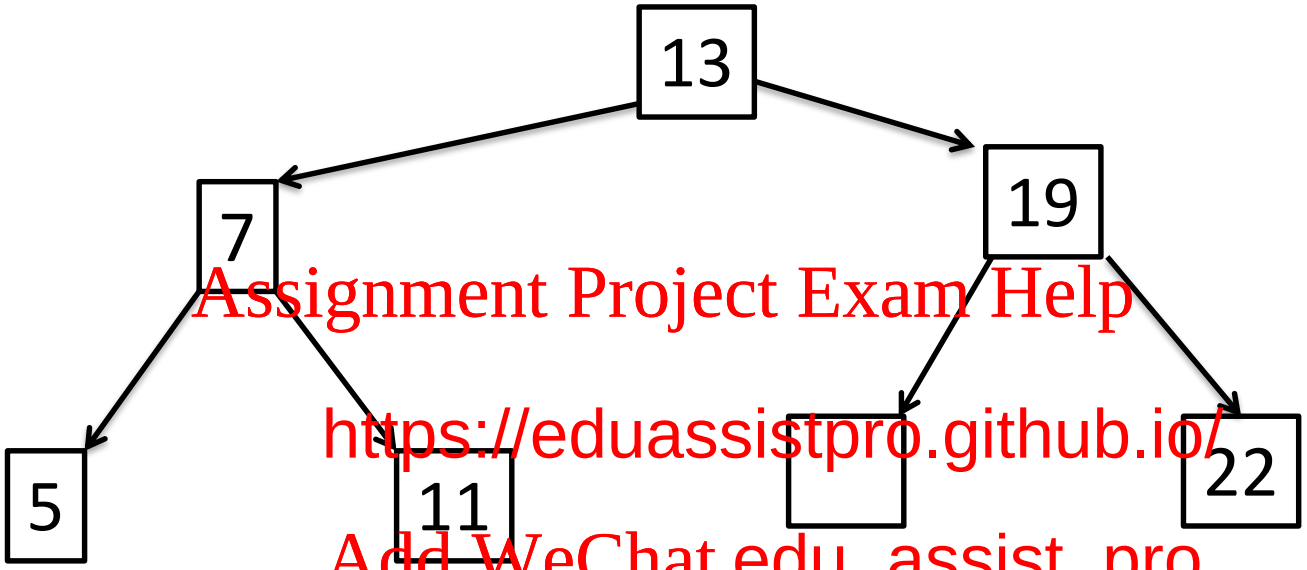
Remove 17

Remove node with 2 children



Remove 17

Remove node with 2 children



Remove 17

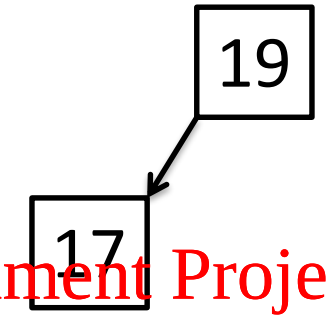
Perfectly Balanced Binary Search Trees

- A binary search tree is perfectly balanced if it has height $\lfloor \log n \rfloor$ (height is the length of the longest pat leaf)

Insertion

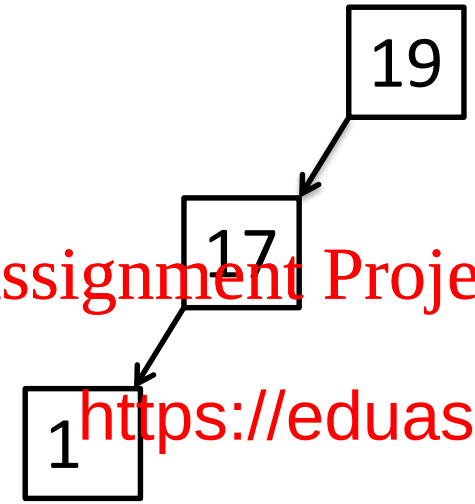
Insert 17

Insertion



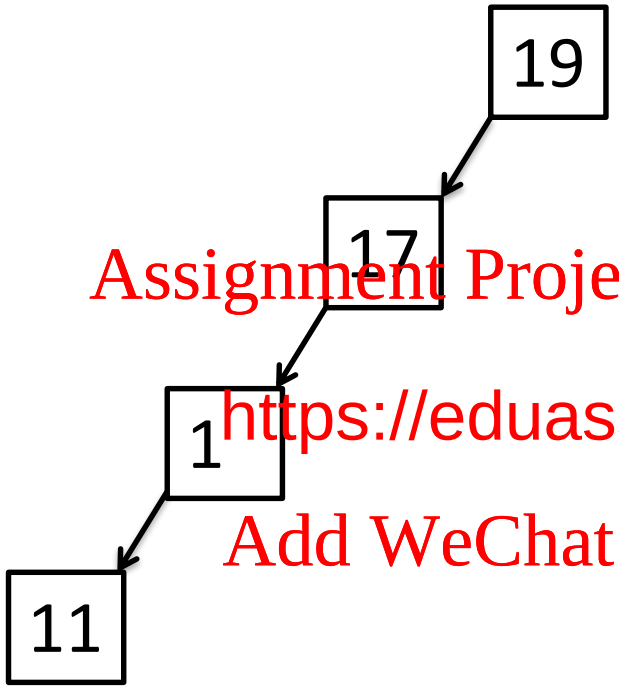
Insert 13

Insertion



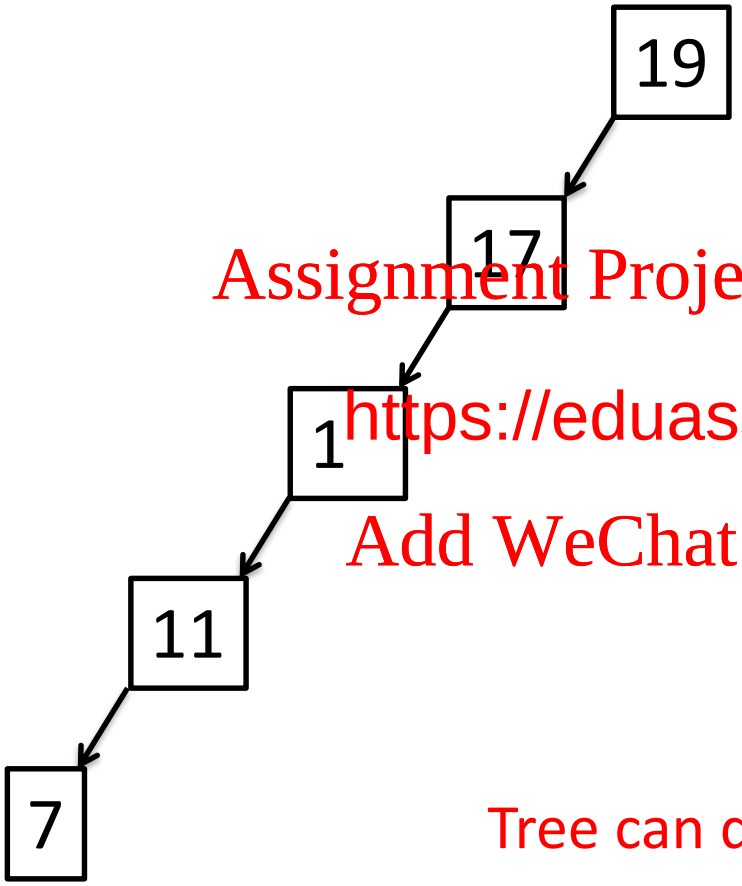
Insert 11

Insertion



Insert 7

Insertion



Tree can degenerate to a list.

…