Binary Search Trees

Ch. 5.7

Dictionary

- A collection of pairs (*key*, *value*)

- Operations:
  - Create(*max_size*)
  - IsEmpty()
  - Search(*key*)
  - Delete(*key*)
  - Insert(k*ey*, *value*)

To perform search, insert, and delete:

A **binary search tree** has a good performance than other data structures.

Definition of binary search tree

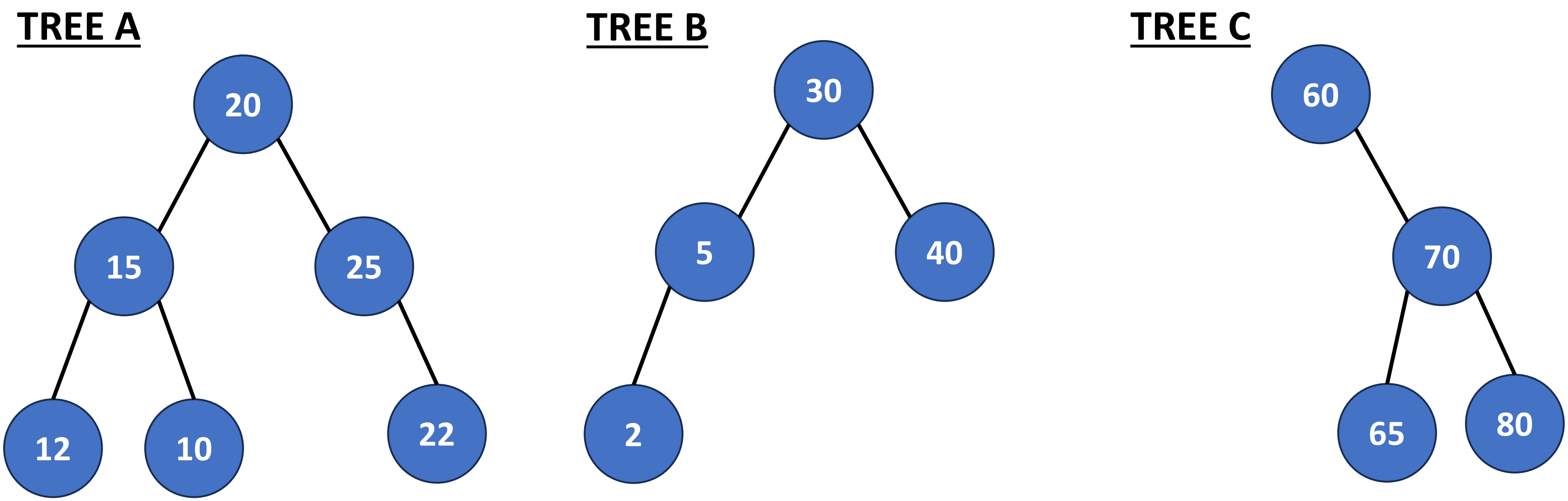
- A binary tree
- Each node has a (key, value) pair.
- Keys in the tree are distinct.
- For every node *x*,

  - all keys in the left subtree are smaller than that in *x*.

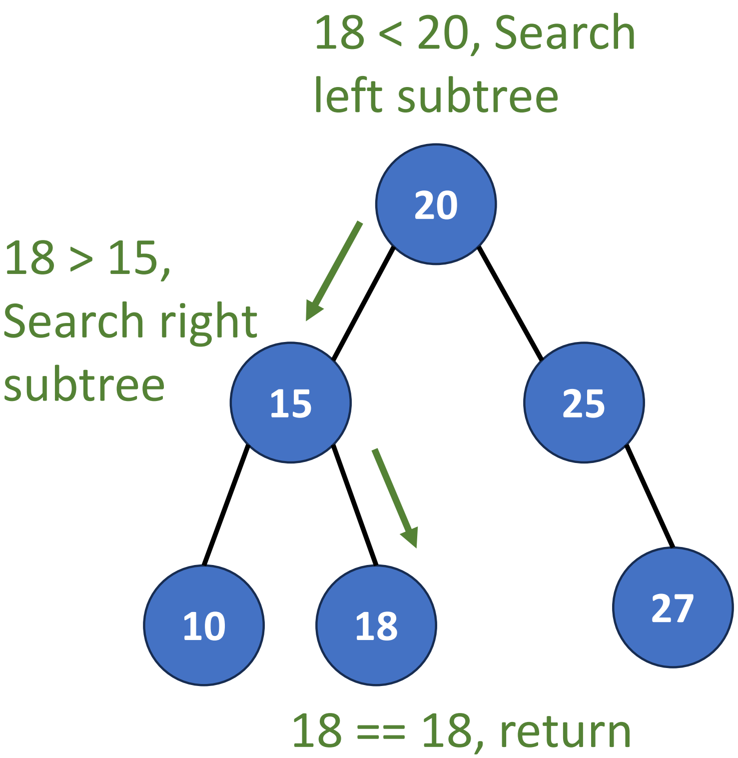
  - all keys in the right subtree are larger than that in *x*.
- The subtrees are also binary search tree.

Are they binary search trees?

- Note: Only the keys are shown.



Operation: Search(*root*,*k*)



Let's find node with key 18.

<u>Three conditions:</u>

- k == root's key: terminate
- k < root's key: check left subtree
- k > root's key: check right subtree

<u>Time Complexity:</u>

O(height) = O(n)

- n is the number of nodes.

Operation: Search(*root*,*k*)

Recursive

Node

leftChild	data =(key,value)	rightChild

```
if (!root) return NULL;
if (k == root->data.key)
  return root->data;
if (k < root->data.key)
  return search(root->leftChild, k);
return search(root->rightChild,k)
```

Variable space requirement: O(height) = O(n)

n is the number of nodes.

Operation: Search(*tree*,*k*)

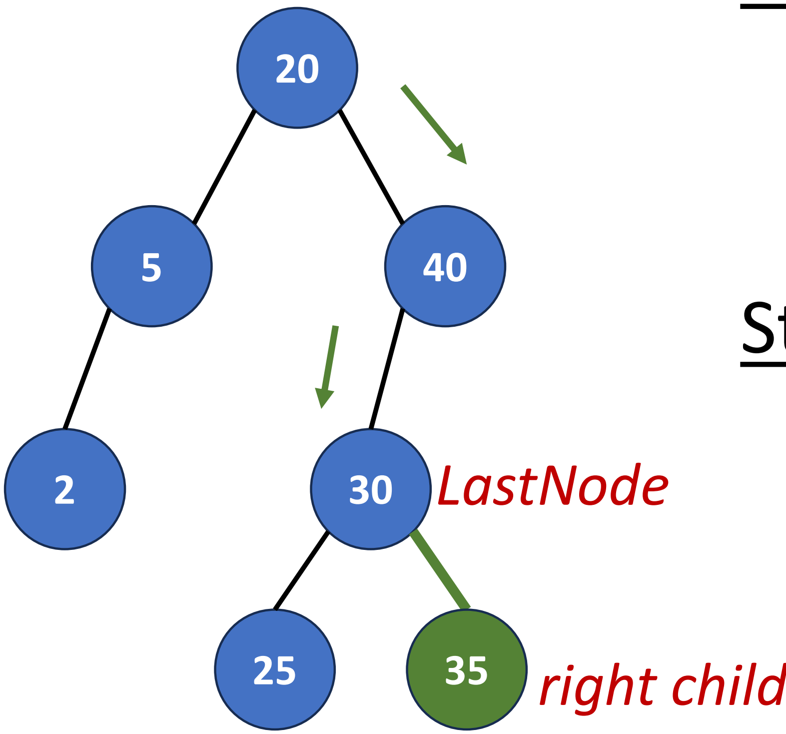
Iterative

```
while(tree);
  if (k == tree->data.key)
    return tree->data;
  if (k < tree->data.key)
    tree = tree->leftChild;
  else
    tree = tree->rightChild;
return NULL
```

Node

leftChild	data =(key,value)	rightChild

Operation: Insert(*root, key, value*)



Insert a node with key 35.

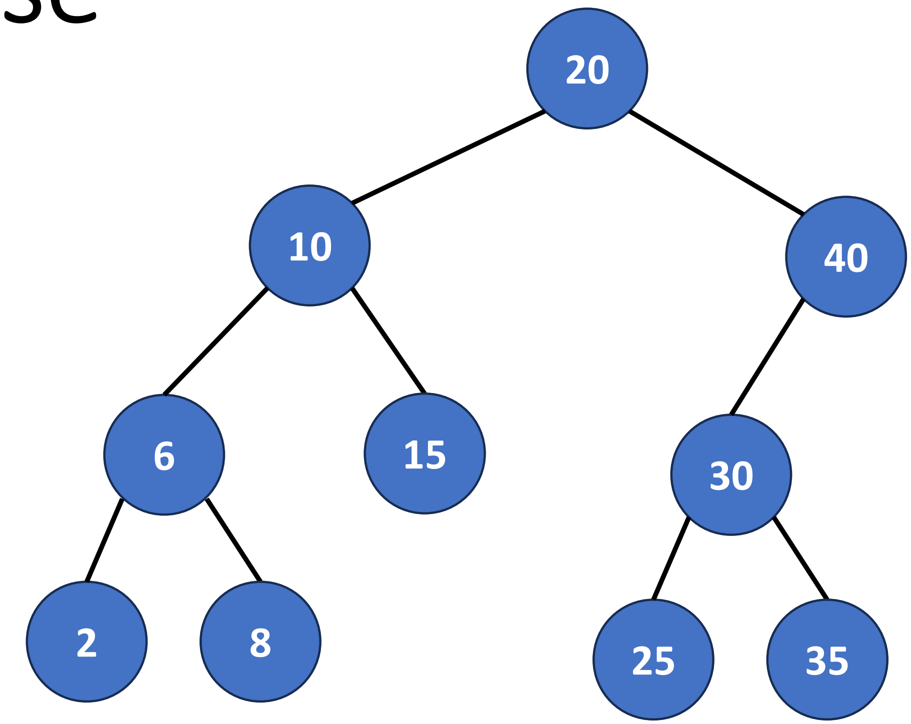
Step 1: Search the tree

- Matched: Do nothing
- No match: Obtain *LastNode,* which is the last node during the search.

Step 2: Add a new node

- Create a new node with (key, value)
- if key > the key of *LastNode*, add the new node as right child.
- if key < key of *LastNode*, add the new node as left child.

Exercise



Please reply your answers of Q5 to Q7 via the following link:

Group members: 1~3 people

Q5: Where will you insert a pair whose key is 7?

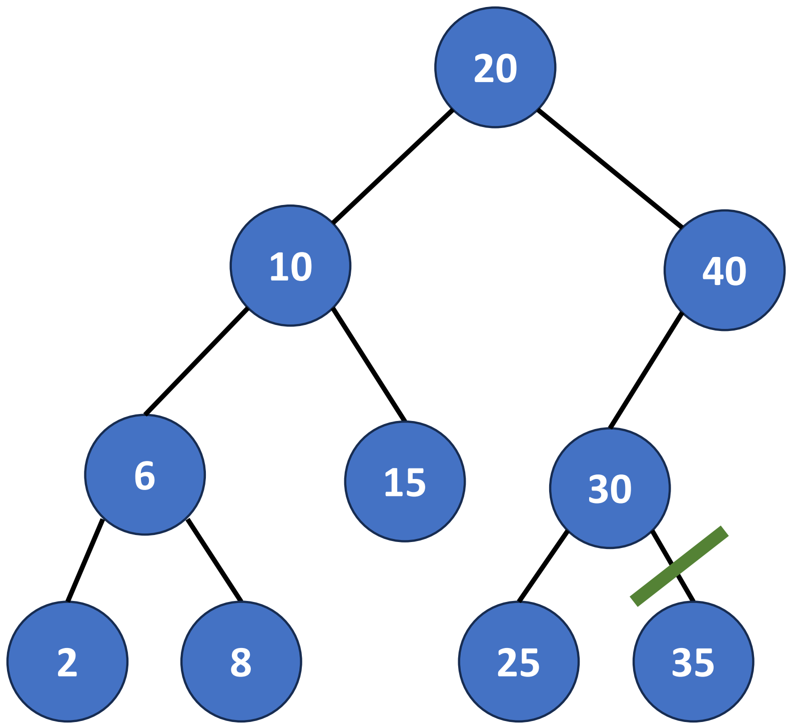
Q6: Where will you insert a pair whose key is 18?

Q7: The time complexity of insertion in binary search trees.

Operation: Delete(*key*)

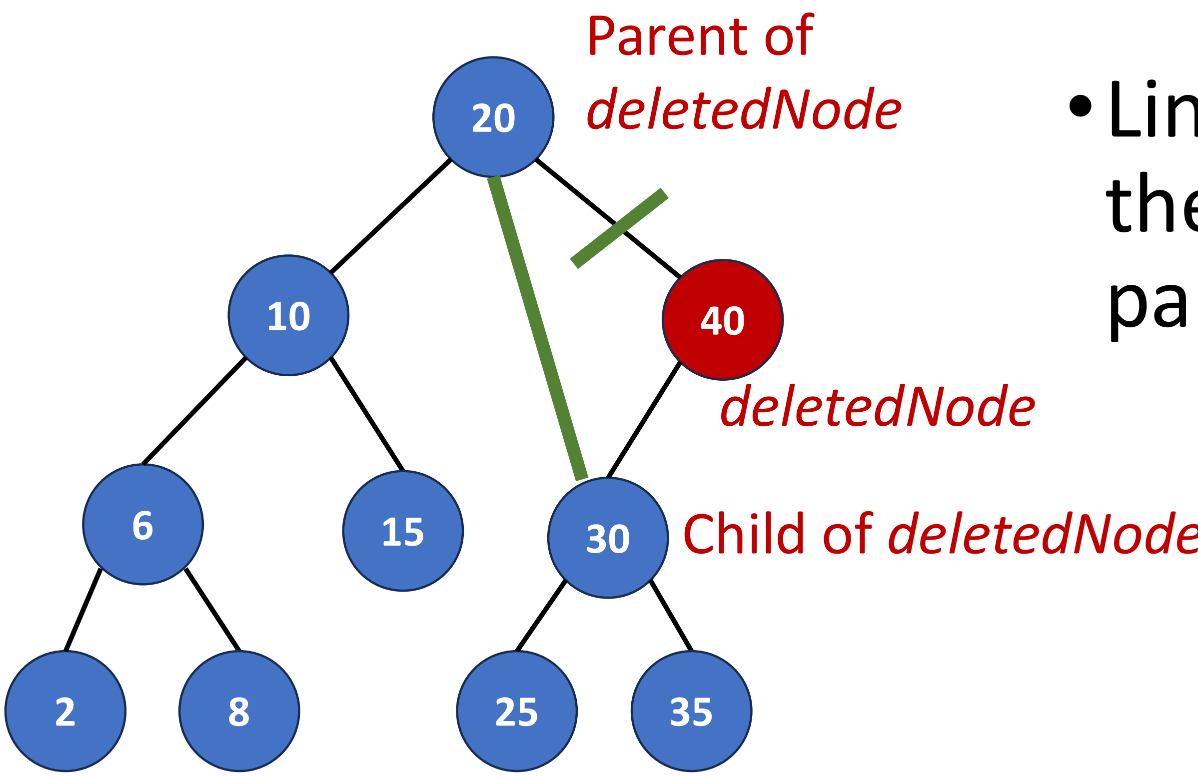
- Four cases:
  - No element with delete key.
  - Element is in a leaf.
  - Element is in a degree-1 node.
  - Element is in a degree-2 node.

Delete a leaf



Insert the node whose key is 35.

Delete a degree-1 node

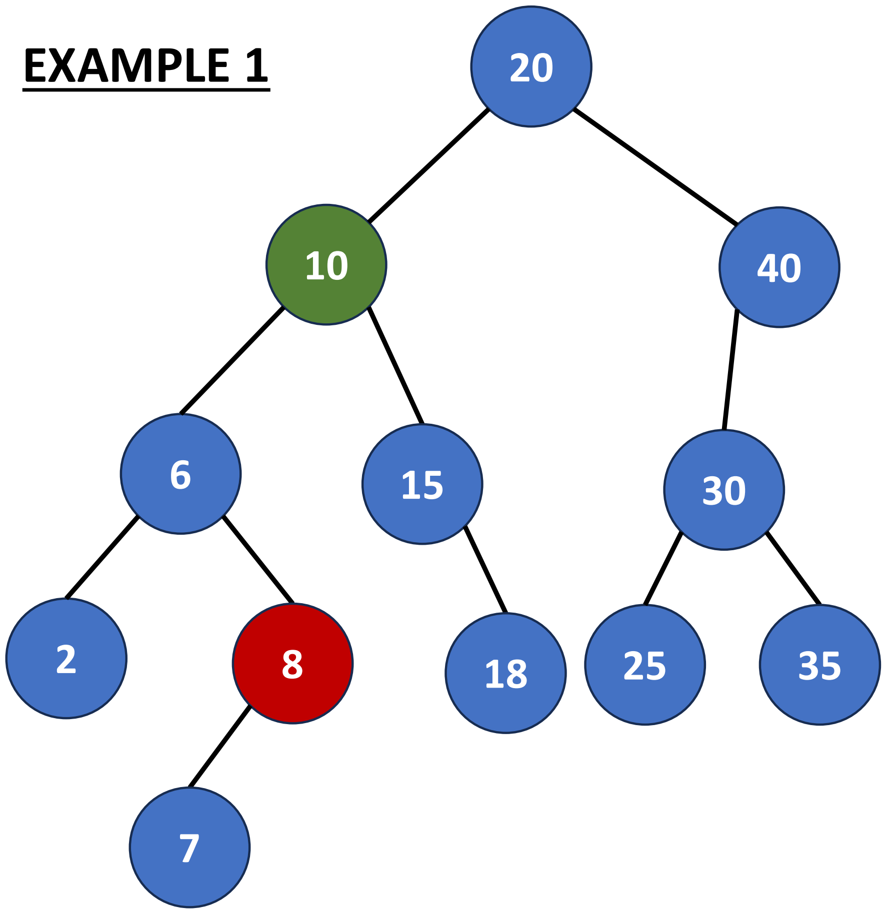


Delete the node whose key is 40.

- Link the single child of the *deletedNode* to the parent of *deletedNode*

Delete a degree-2 node

EXAMPLE 1



- Replace by the
  - largest pair in its left subtree

  or

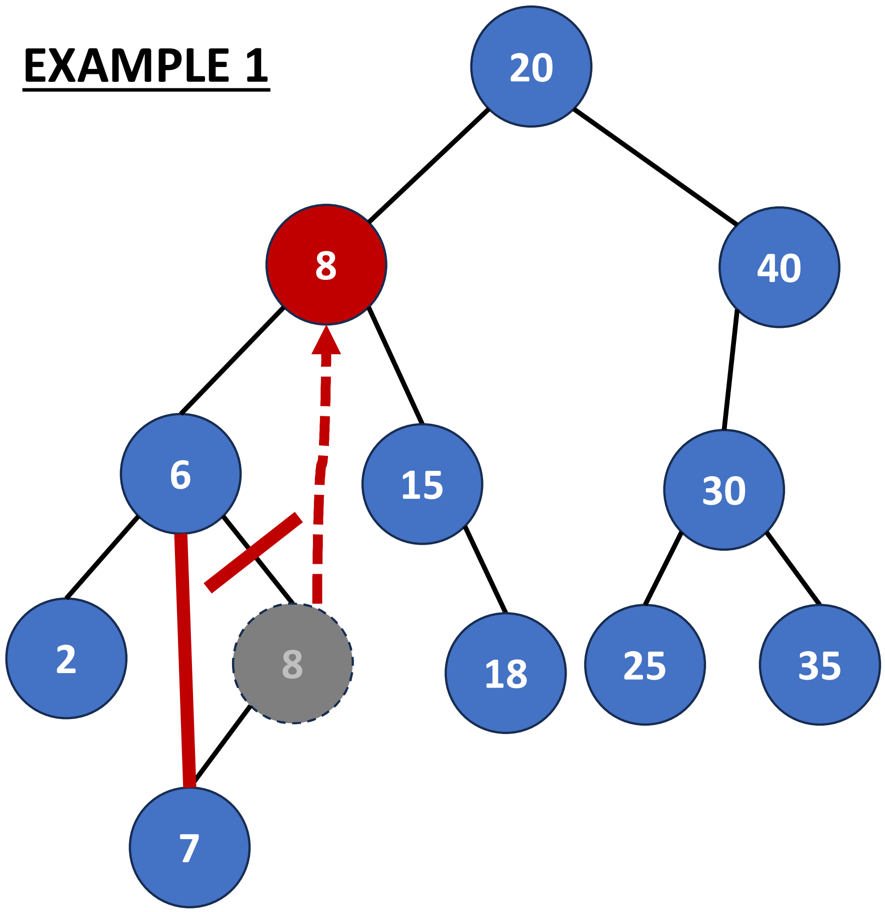
  - smallest pair in its right subtree

These two pairs must be leaf nodes or degree-one nodes. **Why?**

Delete the node whose key is 10.

Delete a degree-2 node

EXAMPLE 1



- Replace by the
  - largest pair in its left subtree

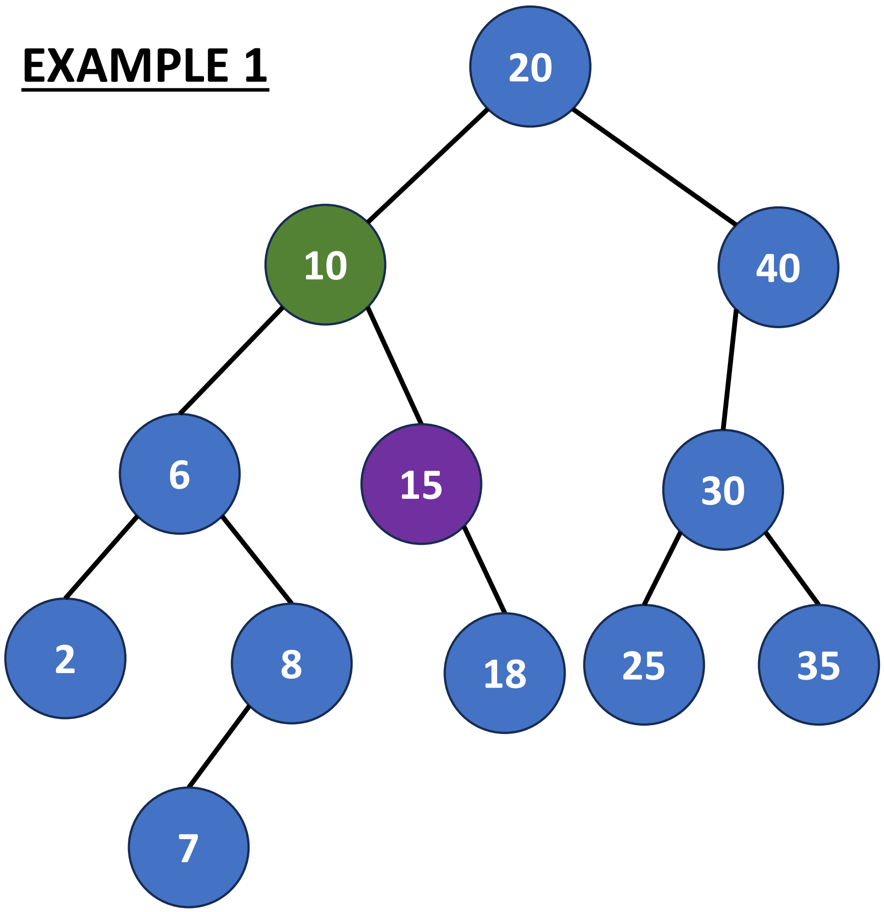
  or

  - smallest pair in its right subtree

Delete the node whose key is 10.

Delete a degree-2 node

EXAMPLE 1



- Replace by the

  - largest pair in its left subtree

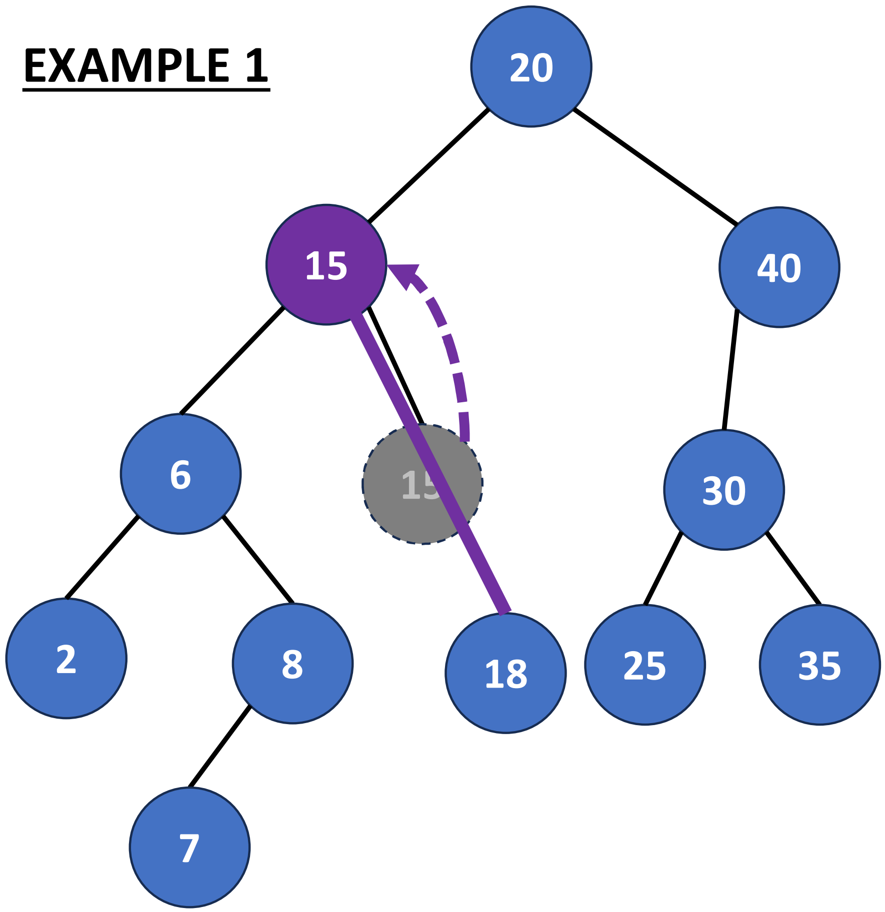
  or

  - smallest pair in its right subtree

Delete the node whose key is 10.

Delete a degree-2 node

EXAMPLE 1



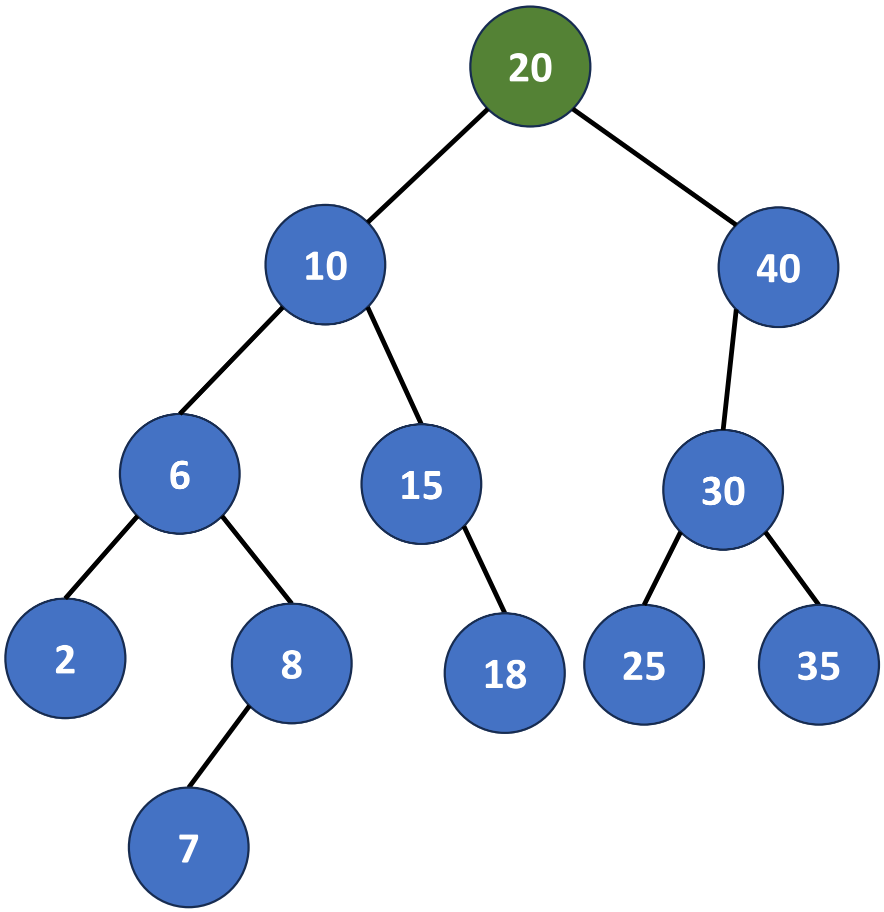
- Replace by the
  - largest pair in its left subtree

  or

  - smallest pair in its right subtree

Delete the node whose key is 10.

Exercise

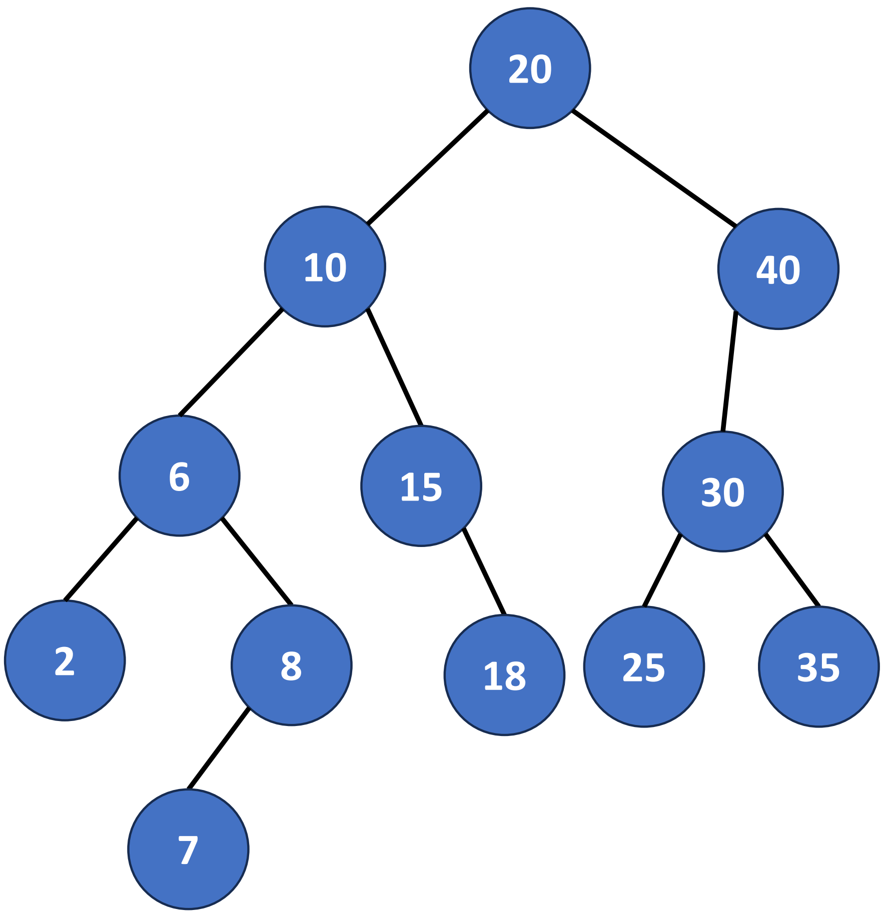


Please reply your answers of Q8 via the following link:

Group members: 1~3 people

- Q8: Describe the steps to <u>delete</u> the node containing 20.

Rank of a node in a binary search tree



Rank of node *x*:

- The number of the nodes whose values are smaller than *x*
- Position of x in inorder

  [2,6,7,8,10,15,18,20,25,30,35,40]

Example:

rank(2) = 0

rank(15) = 5

Summary

- Definition of binary search trees
- Operations of binary search trees
  - Search
  - Insert
  - Delete: 4 cases
- Rank of nodes in binary search trees