Binary Search Tree

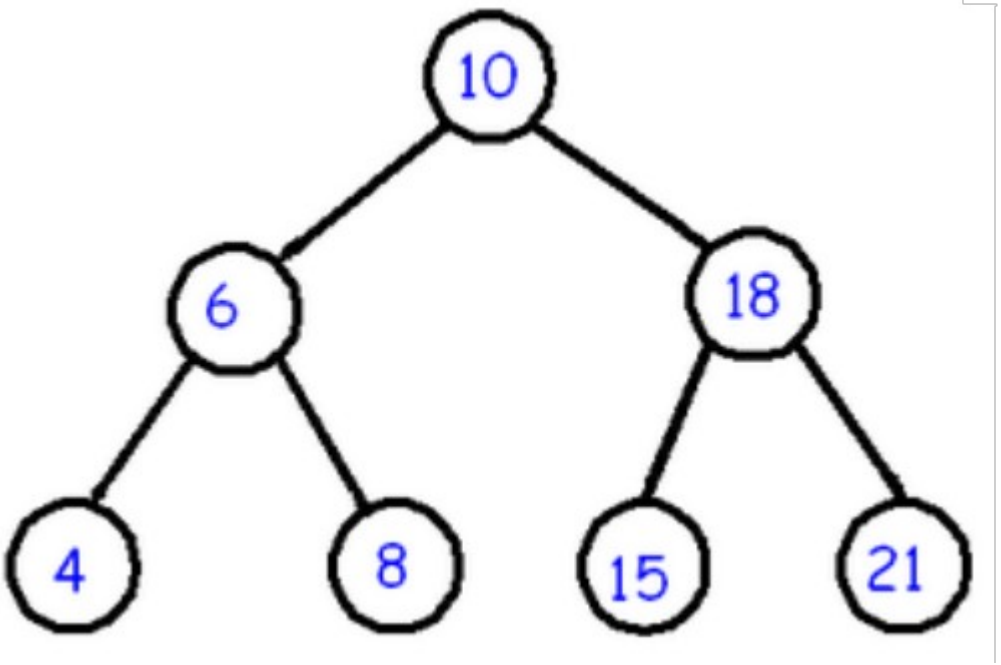
Prof. Siddharth Shah

Binary Search Tree (BST)

- The basic idea behind this data structure is to have such a storing repository that provides the efficient way of data sorting, searching and retrieving.
- A BST is a binary tree where nodes are ordered in the following way:
  - Each node contains one key (also known as data)
  - The keys in the left subtree are less than the key in its parent node, in short L < P;
  - The keys in the right subtree are greater than the key in its parent node, in short P < R;
  - Duplicate keys are not allowed.

Example - BST

- In the following tree
  - all nodes in the left subtree of 10 have keys < 10
  - all nodes in the right subtree > 10
- Because both the left and right subtrees of a BST are again search trees.



Insert a node into BST

Procedure: BSTINSERT(HEAD, KEY)

- Given a lexically ordered binary tree this procedure inserts the node whose information field is equal to **KEY**.
- **PARENT** is a pointer variable which denotes the address of the parent of the node to be inserted.
- **CURRENT** denotes the address of the node in focus.
- Also, the tree assumed to have a list head whose address is given by **HEAD**.

Insert a node into BST

Procedure: BSTINSERT(HEAD, KEY)

1. [Intialize]

If LPTR(HEAD) = HEAD
Then NEW <= NODE
DATA(NEW) ← KEY
LPTR(HEAD) ← NEW
Return
Else CURRENT ← LPTR(HEAD)
PARENT ← HEAD

2. [Traverse tree to find Parent node of key]

Repeat while CURRENT ≠ NULL
PARENT ← CURRENT
If KEY < DATA(CURRENT)
Then CURRENT ← LPTR(CURRENT)
Else CURRENT ← RPTR(CURRENT)

3. [Create a node with key]

If KEY < DATA(PARENT)
Then LPTR(PARENT) <= NODE
DATA(LPTR(PARENT)) ← KEY
Else RPTR(PARENT) <= NODE
DATA(RPTR(PARENT)) ← KEY

4. [Finished]

Return

Delete a node from BST

Procedure: BSTDELETE(HEAD, KEY)

- Given a lexically ordered binary tree this procedure deletes the node whose information field is equal to **KEY**.
- **PARENT** is a pointer variable which denotes the address of the parent of the node marked for deletion.
- **CURRENT** denotes the address of the node to be deleted.
- **PRED** and **SUC** are pointer variables used to find the inorder successor of CURRENT.
- **Q** contains the address of the node to which either the left or right link of the parent of KEY must be assigned in order to complete the deletion.
- Finally, **D** contains the direction from the parent node to the node marked for deletion.
- Also, the tree assumed to have a list head whose address is given by **HEAD**.
- **FOUND** is a boolean variable which indicates whether the node marked for deletion has been found.

Delete a node from BST

1. [Intialize]

```
If      LPTR(HEAD) ≠ HEAD
Then    CURRENT← LPTR(HEAD)
        PARENT ← HEAD
        D ← 'L'
Else    Write ('Node not found')
        Return
```

2. [Search for the node marked for deletion]

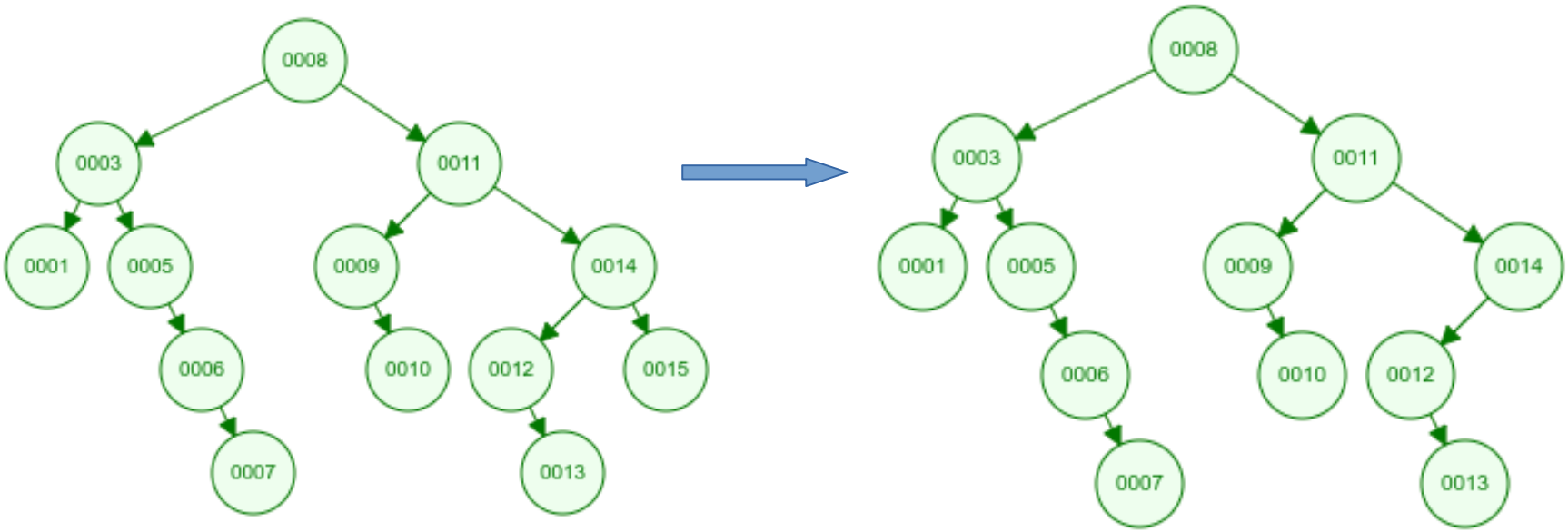
```
FOUND ← false
    Repeat while not FOUND and CURRENT ≠ NULL
        If      DATA(CURRENT) = KEY
        Then    FOUND ← true
        Else    If      KEY < DATA(CURRENT)
                Then    PARENT ←  CURRENT      /* branch left
                        CURRENT ← LPTR(CURRENT)
                        D ← 'L'
                Else    PARENT ←  CURRENT      /* branch righ
                        CURRENT ← RPTR(CURRENT)
                        D ← 'R'
    If  FOUND = false
    Then        Write ('Node not found')
                Return
```

3. [Perform the indicated deletion and restructure the tree]

```
If   LPTR(CURRENT) = NULL
Then   Q ← RPTR(CURRENT)      /* empty left subtree */
Else   If      RPTR(CURRENT) = NULL
       Then    Q ← LPTR(CURRENT)      /* empty right subtree */
       Else    (check right child for successor)
               SUC ← RPTR(CURRENT)
               If      LPTR(SUC) = NULL
               Then    LPTR(SUC) ← LPTR(CURRENT)
                       Q ← SUC
               Else    (search for successor of CURRENT)
                       PRED ← RPTR(CURRENT)
                       SUC ← LPTR(PRED)
                       Repeat while LPTR(SUC) ≠ NULL
                               PRED ← SUC
                               SUC ← LPTR(PRED)
                       (Connect Successor)
                       LPTR(PRED) ← RPTR(SUC)
                       LPTR(SUC) ← LPTR(CURRENT)
                       RPTR(SUC) ← RPTR(CURRENT)
                       Q ← SUC
(Connect parent of KEY to its replacement)
If  D = 'L'
Then  LPTR(PARENT) ← Q
Else  RPTR(PARENT) ← Q
Return
```

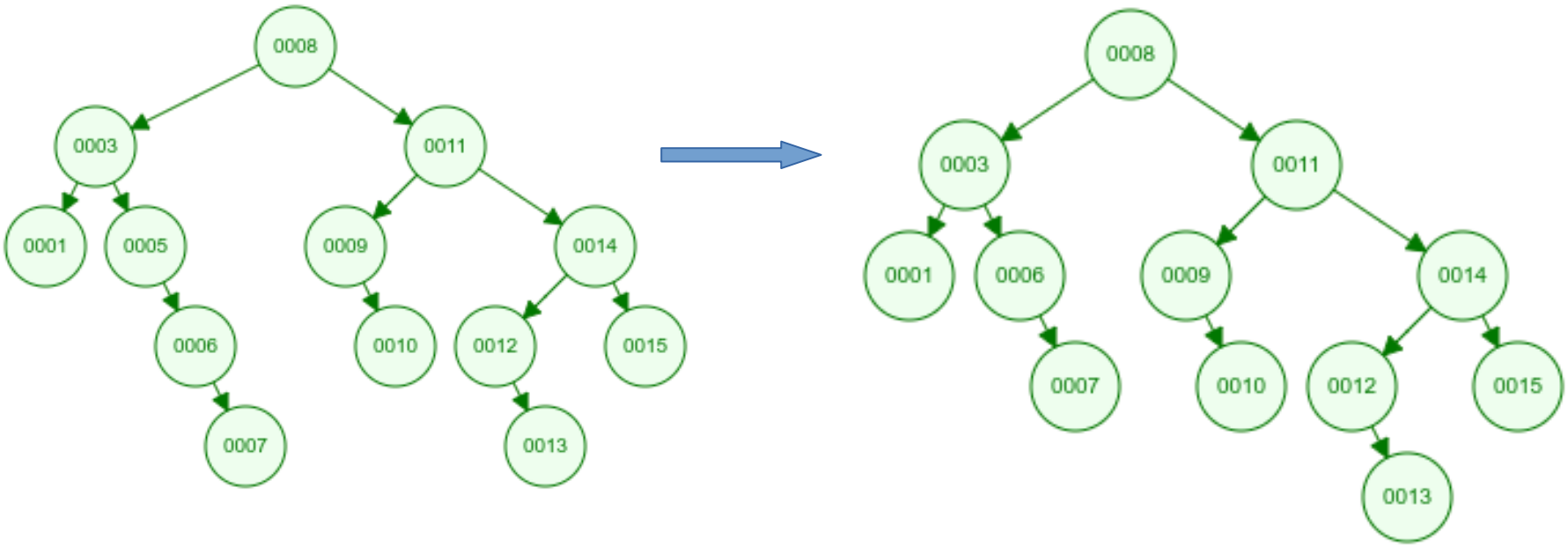
Example

Delete a node with key 15



Example

Delete a node with key 5



Example

Delete a node with key 11