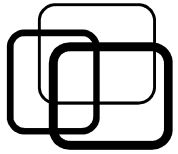
Binary Search Tree

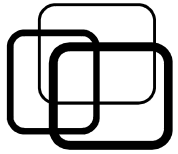
Inst. Nguyễn Minh Huy

Contents



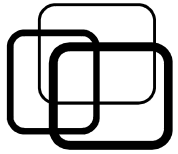
- Concepts.
- Implementation.

Contents

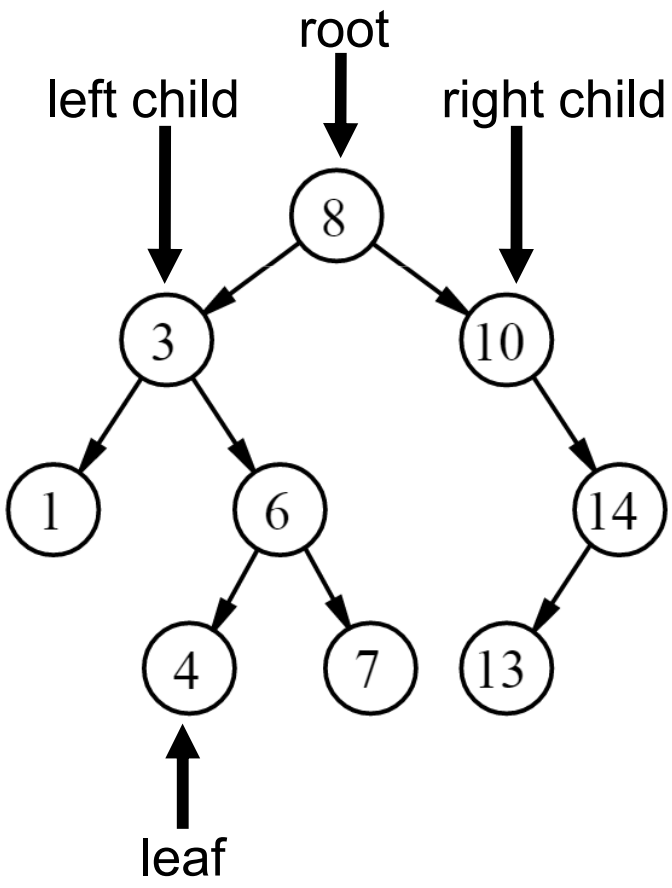


- **Concepts.**
- Implementation.

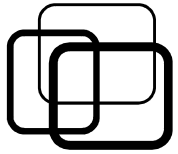
Concepts



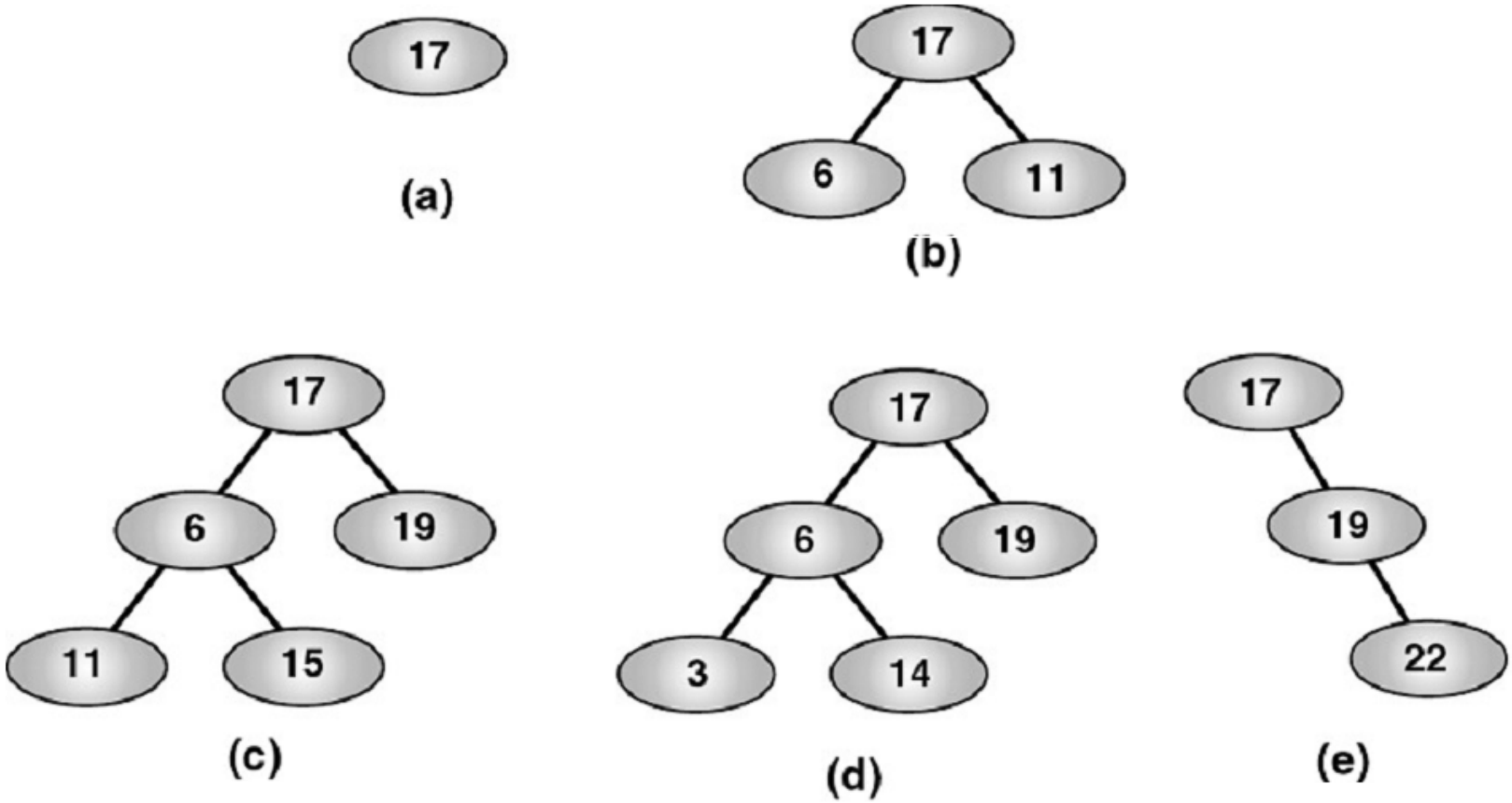
- Binary search tree:
  - A binary linked data structure.
  - Linear linked vs. binary linked.
  - Each node has:
    - At most one parent.
    - At most two children.
    - **All left < node <= all right.**
  - Root: node has no parent.
  - Leaf: node has no children.



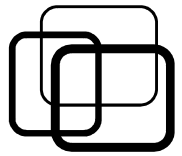
Concepts



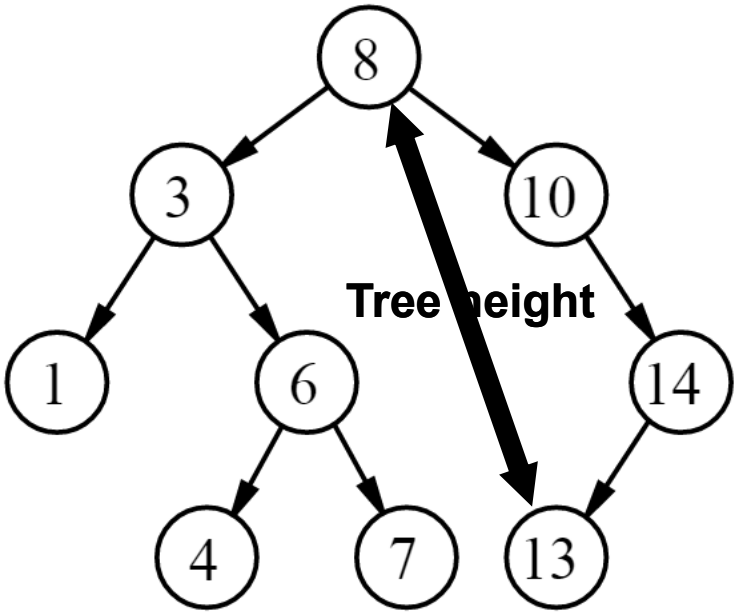
- Example:
  - Which tree is binary search tree?

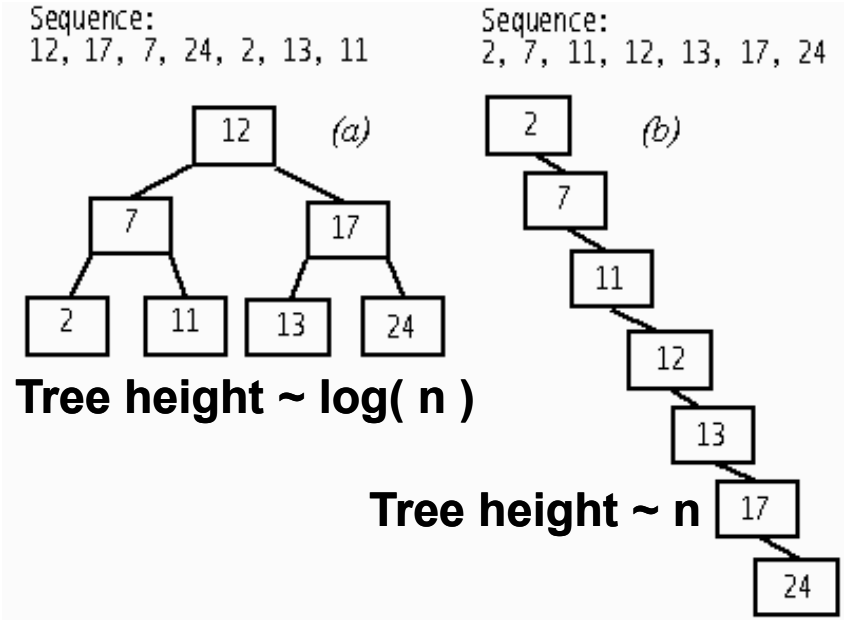


Concepts

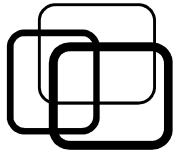


- Tree height:
  - Height of node: longest path from node to leaf.
  - Tree height = height of root.
  - Important factor of performance:
    - For a n-node tree: log(n) <= tree height <= n.



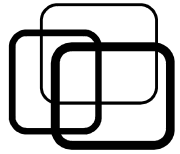


Contents



- Concepts.
- **Implementation.**

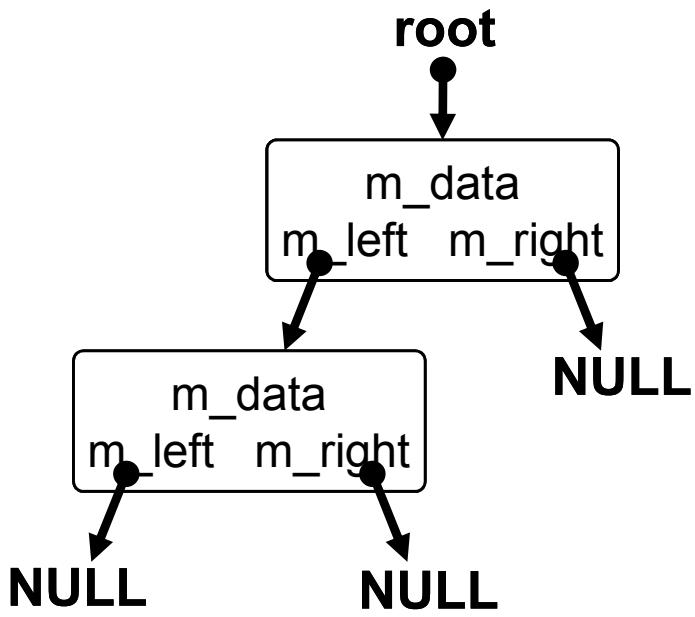
Implementation



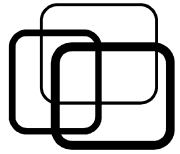
- ADT Binary search tree:
  - Values:

```
struct TreeNode {
        int          m_data;
        TreeNode  *m_left;
        TreeNode  *m_right;
};

class BSTree {
private:
      TreeNode* m_root;
};
```

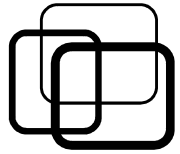


Implementation



- ADT Binary search tree:
  - Operations:
    - Initialization.
    - Check empty.
    - Find a key.
    - Add a key.
    - Remove a key.
    - Visit.

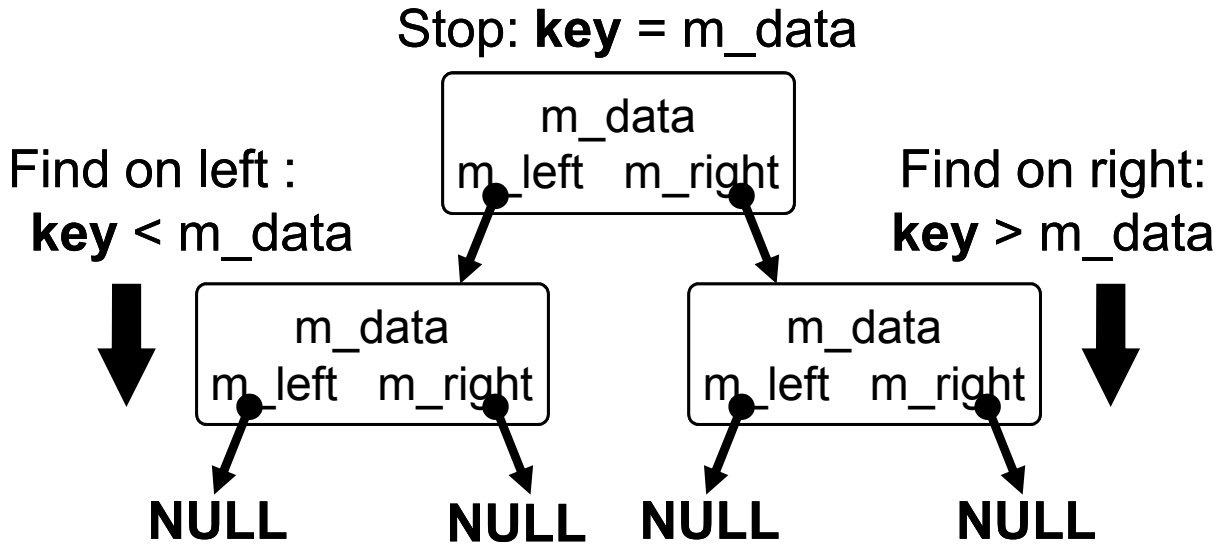
Implementation



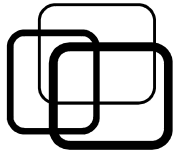
- Find a key:

Find key at a node {
    if (**key** < m_data)
        Find key on left child;
    if (**key** > m_data)
        Find key on right child;

    return **current node**;
}



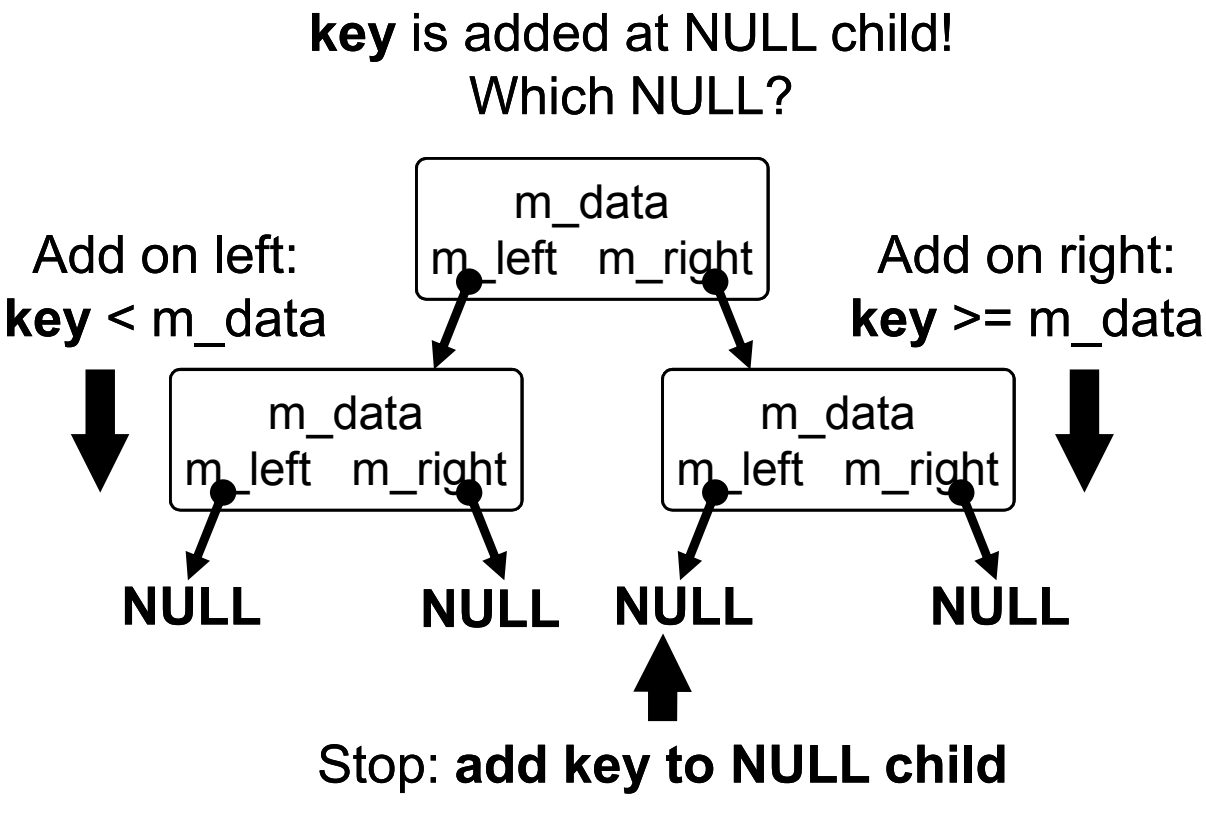
Implementation



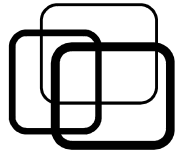
- Add a key (keep tree condition):

Add key at a node {
    if **NULL** node
        Make node with key
        Stop

    if (**key** < m_data)
        Add key on left child
    else
        Add key on right child
}

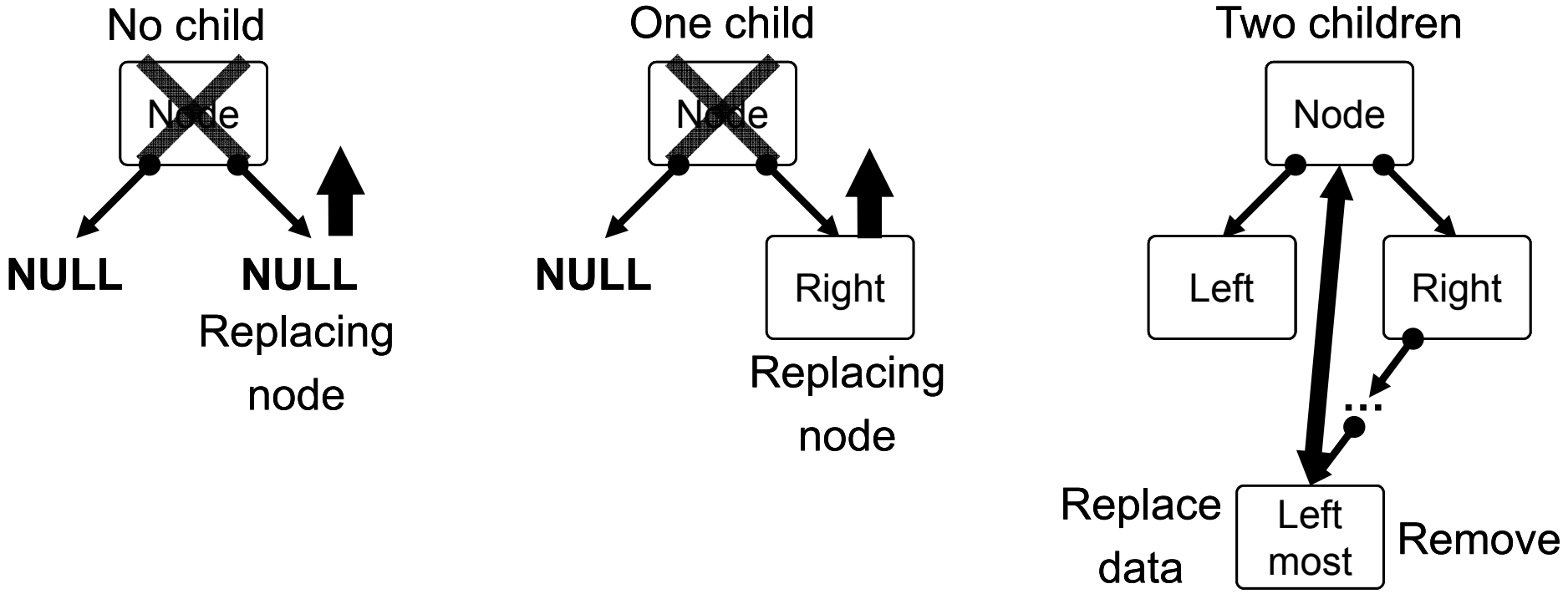


Implementation

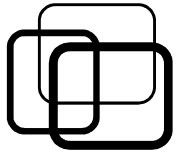


- Remove a key (keep tree condition):

Remove key at a node {
    Find key at node
    if found
        Replace node
}



Implementation

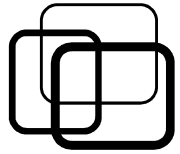


- Remove a key (keep tree condition):

Replace a node {
    if (**no left child**)
        replace = m_right;
        delete current node;
    else if (**no right child**)
        replace = m_left;
        delete current node;
    else
        replace = current node;
        leftMost = **find left most on right**;
        m_data = **leftMost**->m_data;
        Remove leftMost->m_data at **leftMost**;

    current node = **replace**;
}

Implementation



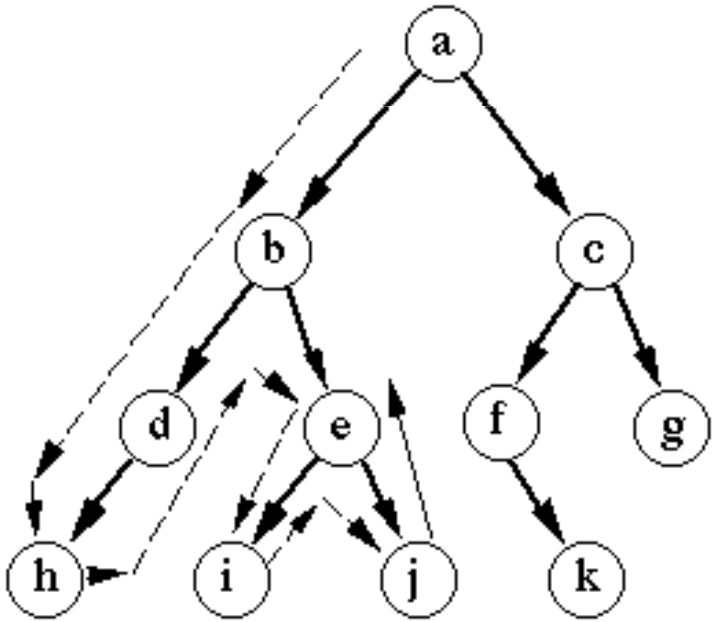
- Visit a node:

Depth-first visit:

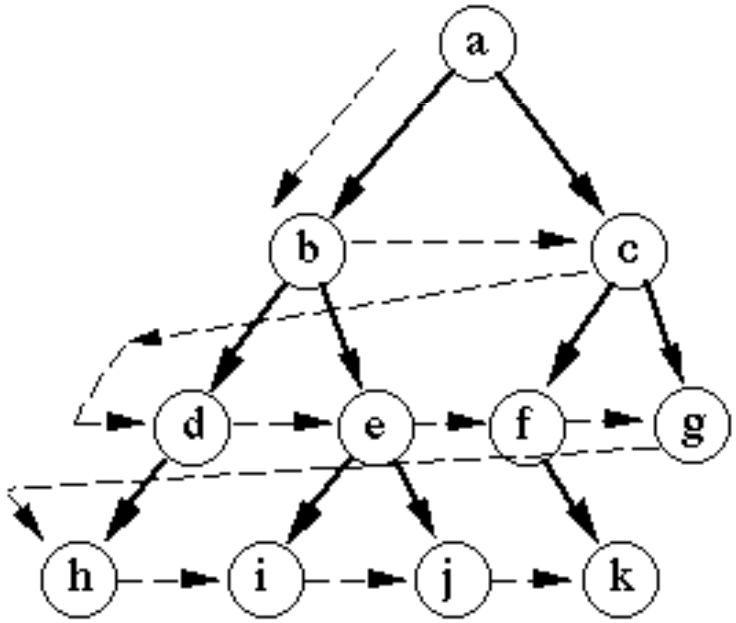
- Visit left branch first.
- Then visit right branch.

Breadth-first visit:

- Visit 1st generation children first.
- Then visit 2nd generation children.

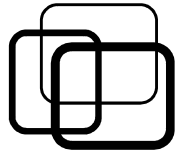


Depth-first search



Breadth-first search

Implementation



- Visit a node:

Visit depth NLR {
    Do something on node…
    Visit depth left child
    Visit depth right child
}

Visit depth LNR {
    Visit depth left child
    Do something on node.
    Visit depth right child
}

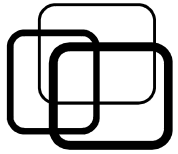
Visit depth LRN {
    Visit depth left child
    Visit depth right child
    Do something on node…
}

Visit breadth {
    push node to **QUEUE**;
    loop **QUEUE is not empty** {
        node = **pop QUEUE**;

        Do something on node…

        if (node **has left child**)
            push left child to **QUEUE**;
        if (node **has right child**)
            push right child to **QUEUE**;
    }
}

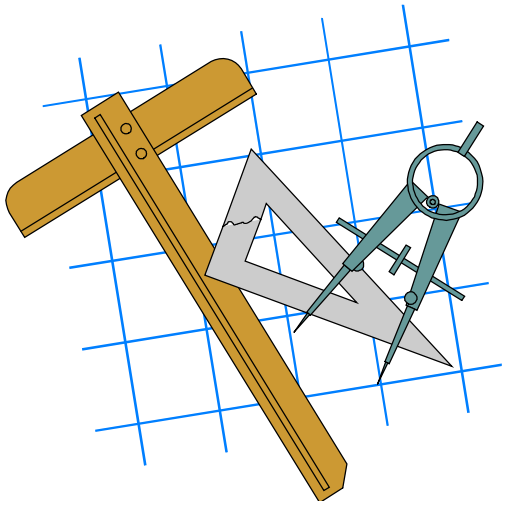
Practice



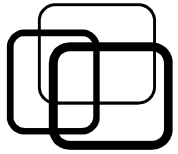
- Practice 8.1:

  Construct class **BSTree** has the following methods:

  - Initialize.
  - Check empty.
  - Count nodes.
  - Measure height.
  - Find key.
  - Add key.
  - Remove key.
  - Visit depth NLR.
  - Visit breadth.



Practice



- Practice 8.2:

  Provide class **BSTree** with the following methods:

  - Construct tree from an array of integers.
  - Export tree to an array depth first NLR.
  - Export tree to an array breadth first.

  Example:

```
int  a [ ] = { 5, 2, 4, 1, 3 };
int  size = 5;
BSTree t( a, size );
t.exportDepthFirst( a, size );
t.exportBreadthFirst( a, size );
```