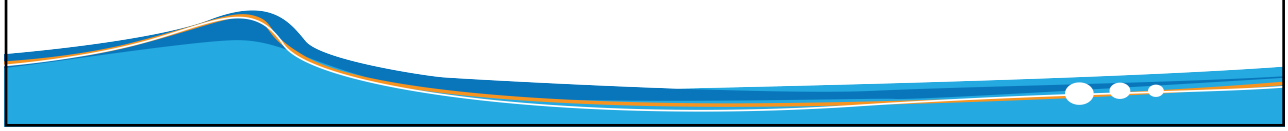
fit@hcmus

Tree Structures

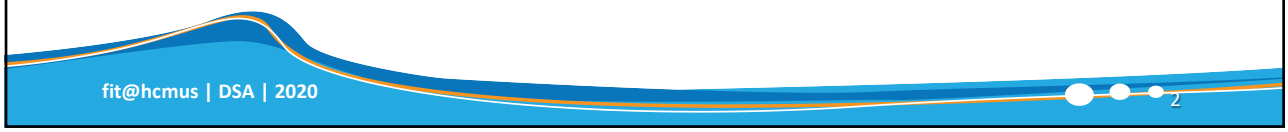


1

fit@hcmus

Contents

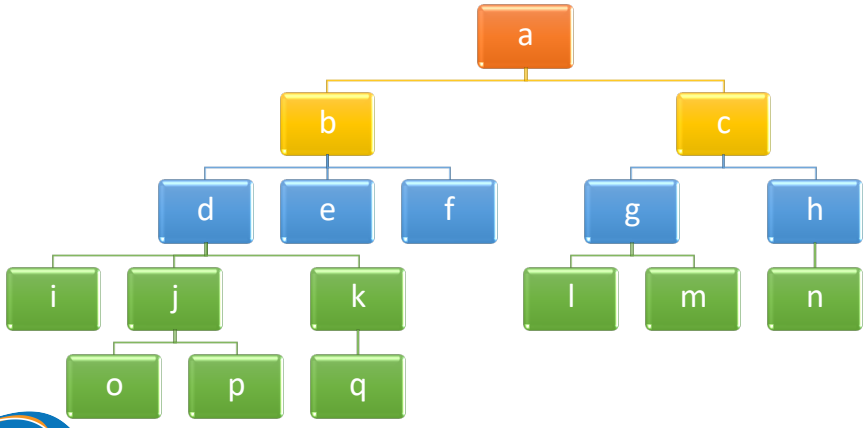
- Terminologies
- Tree traversals
- Tree representation
- Binary tree
- Binary search tree
- AVL tree



2

fit@hcmus

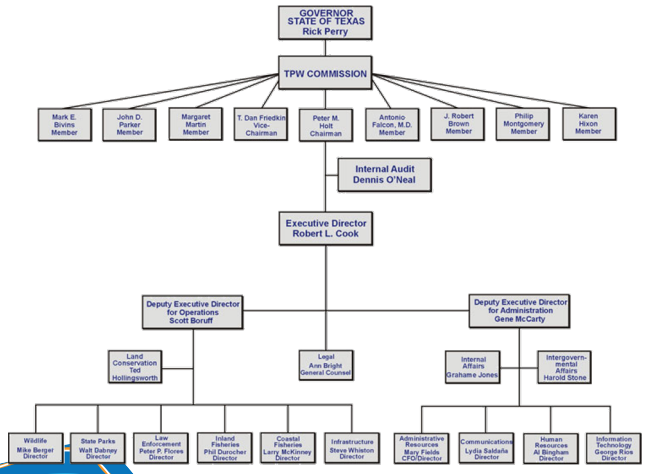
Some Examples



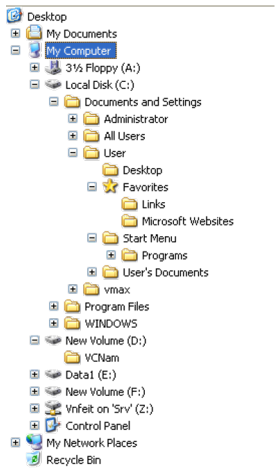
4

fit@hcmus

Some Examples



Organizational chart



Directory tree

5

fit@hcmus

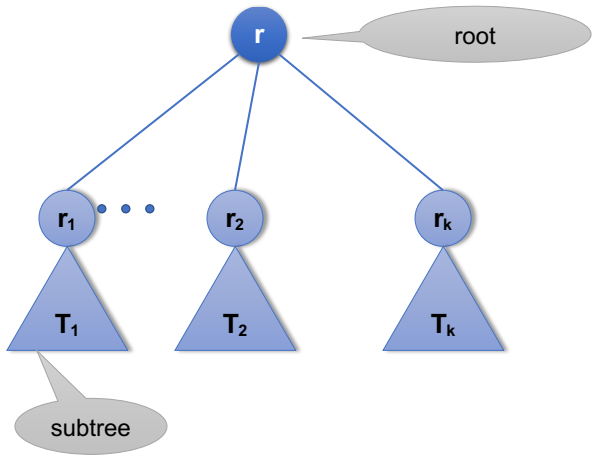
Trees

- Used to represent **relationships**
- ***hierarchical*** in nature
  - "Parent-child" relationship exists between nodes in tree.
  - Generalized to ancestor and descendant
  - Lines between the **nodes** are called **edges**
- A **subtree** in a tree is any node in the tree together with all of its descendants

6

fit@hcmus

Trees



7

fit@hcmus

Terminologies

- node: an item/element in a tree.
- parent (of node *n*): The node **directly above** node *n* in the tree.
- child (of node *n*): The node **directly below** node *n* in the tree.
- root: The only node in the tree with no parent.
- leaf: A node with no children.
- path: A sequence of nodes and edges connecting a nodes with the nodes below it.

8

fit@hcmus

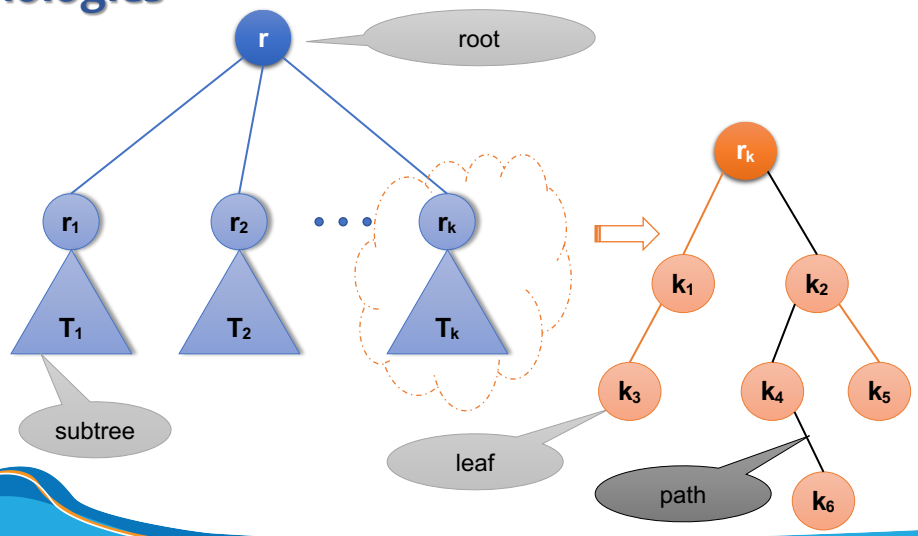
Terminologies

- siblings: Nodes with common parent.
- ancestor (of node *n*): a node on the path from the root to *n*.
- descendant (of node *n*): a node on the path from node *n* to a leaf.
- subtree (of node *n*): A tree that consists of a child (if any) of *n* and the child's descendants.

9

fit@hcmus

Terminologies



10

fit@hcmus

Terminologies

- degree/order
  - Order of node *n*: number of children of node *n*.
  - Order of a tree: the maximum order of nodes in that tree.
- depth/level (of node *n)*
  - If *n* is the root of *T* , it is at level 1.
  - If *n* is not the root of *T* , its level is 1 greater than the level of its parent.

```
if node n is root:
    level(n) = 1
Otherwise:
   level(n) = 1 + level(parent(n))
```


11

fit@hcmus

Terminologies

- Height of tree: number of nodes in the longest path from the root to a leaf.

- Height of a tree T in terms of the levels of its nodes
  - If T is empty, its height is 0.
  - If T is not empty, its height is equal to the maximum level of its nodes.

12

fit@hcmus

Terminologies

- Height of tree *T*:

```
if T is empty:
   height(T) = 0
Otherwise:
   height (T) = max{level(Ni)}, Ni∈T
```

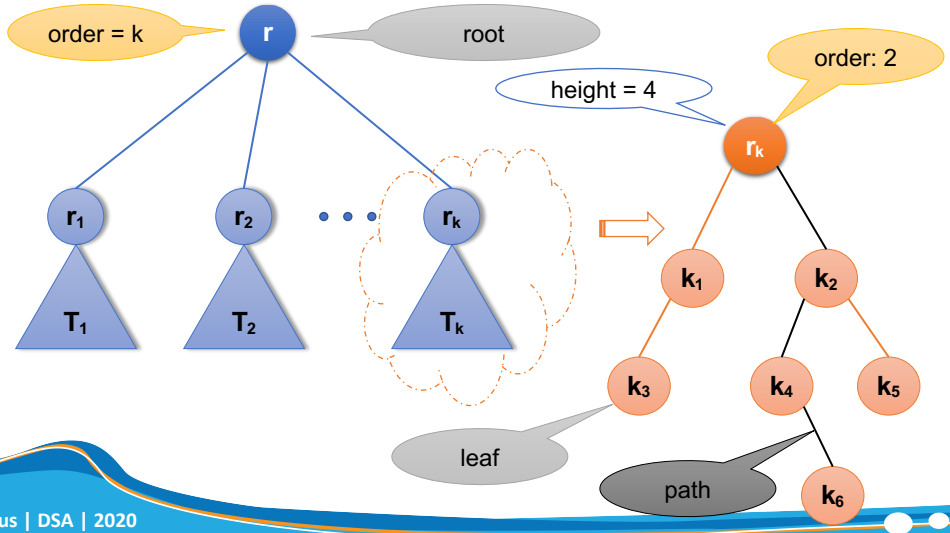
- Height of tree *T*:

```
if T is empty:
   height(T) = 0
Otherwise:
   height(T) = 1 + max{height(Ti)}, Ti is a subtree of T
```


13

fit@hcmus

Terminologies



14

fit@hcmus

Kinds of Trees

15

fit@hcmus

General Tree

- Set T of one or more nodes such that T is partitioned into disjoint subsets
  - A single node r , the root
  - Sets that are general trees, called subtrees of r

16

fit@hcmus

n-ary tree

- set T of nodes that is either empty or partitioned into disjoint subsets:
  - A single node r , the root
  - n possibly empty sets that are n-ary subtrees of r

17

fit@hcmus

Binary tree

- Set T of nodes that is either empty or partitioned into disjoint subsets
  - Single node r , the root
  - Two possibly empty sets that are binary trees, called left and right subtrees of r

18

fit@hcmus

Traversals

19

fit@hcmus

Traversal

- Visit each node in a tree **exactly once**.
- Many operations need using tree traversals.
- The basic tree traversals:
  - Pre-order
  - In-order
  - Post-order

20

fit@hcmus

Pre-order Traversal

```
PreOrder(root)
{
  if root is empty
     Do_nothing;
  Visit root; //Print, Add, ...
  //Traverse every Child_i.
  PreOrder(Child_0);
  PreOrder(Child_1);
  …
  PreOrder(Child_k-1);
}
```


21

fit@hcmus

Post-order Traversal

```
PostOrder(root)
{
  if root is empty
     Do_nothing;
  //Traverse every Child_i
  PostOrder(Child_0);
  PostOrder(Child_1);
  …
  PostOrder(Child_k-1);
  Visit at root; //Print, Add, ...
}
```


22

fit@hcmus

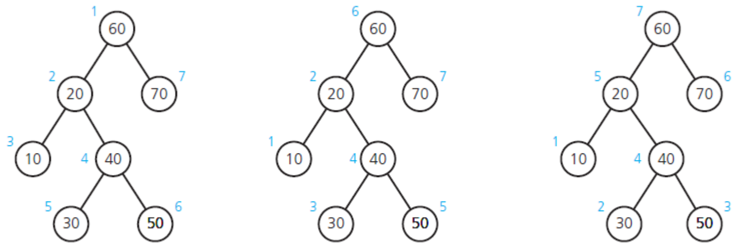
In-order Traversal

```
InOrder(root)
{
  if root is empty
     Do_nothing;
  //Traverse the child at the first position
  InOrder(Child_0);
  Visit at root;
  //Traverse other children
  InOrder(Child_1);
  InOrder(Child_2);
  …
  InOrder(Child_k-1);
}
```


23

fit@hcmus

Traversals



(a) Preorder: 60, 20, 10, 40, 30, 50, 70 (b) Inorder: 10, 20, 30, 40, 50, 60, 70 (c) Postorder: 10, 30, 50, 40, 20, 70, 60

(Numbers beside nodes indicate traversal order.)

fit@hcmus

Examples

Pre-order

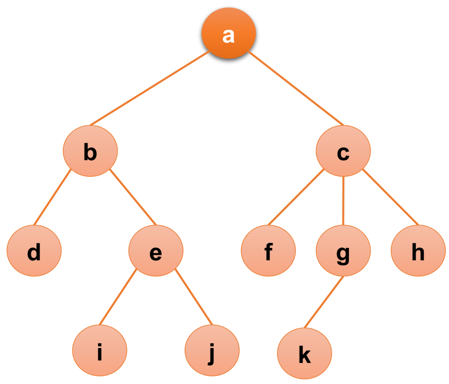
- *a b d e i j c f g k h*

In-order

- d b i e j a f c k g h

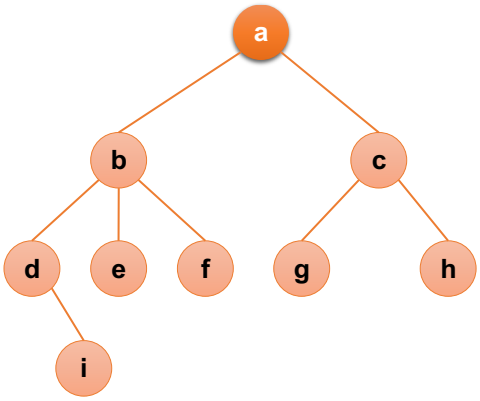
Post-order

- *d i j e b f k g h c a*



fit@hcmus

Examples



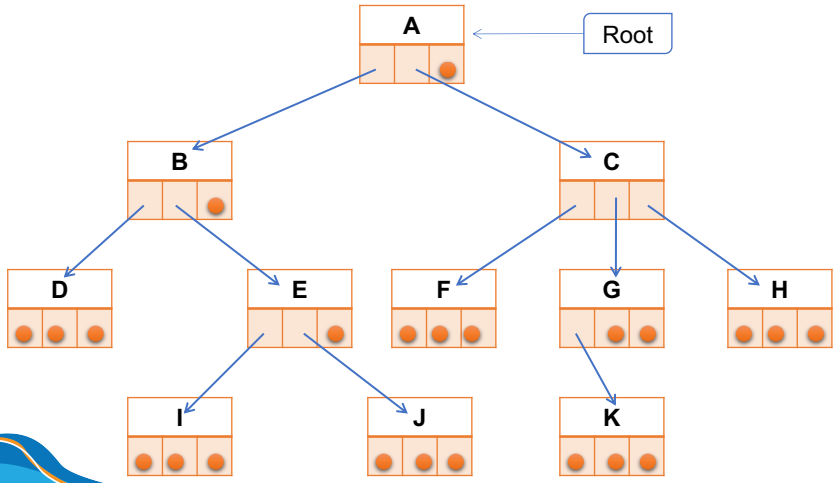
27

fit@hcmus

Tree Representation

28

Tree Representation



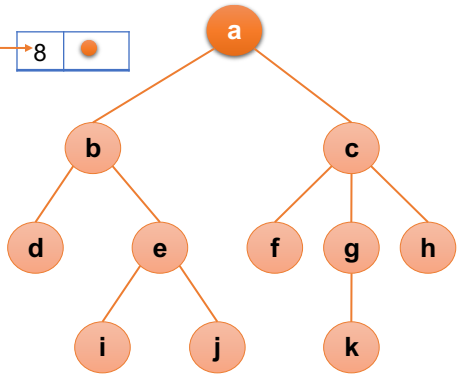
29

Tree Representation

	info	child
1	a	
2	b	
3	c	
4	d	
5	e	
6	f	
7	g	
8	h	
9	i	
10	j	
11	k	

id	next
2	
4	
6	
9	
11	

3	
5	
7	
10	
8	

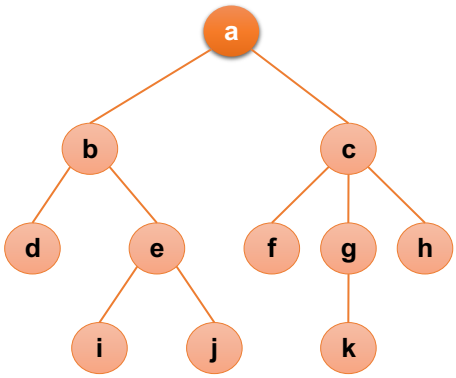


30

fit@hcmus

Tree Representation

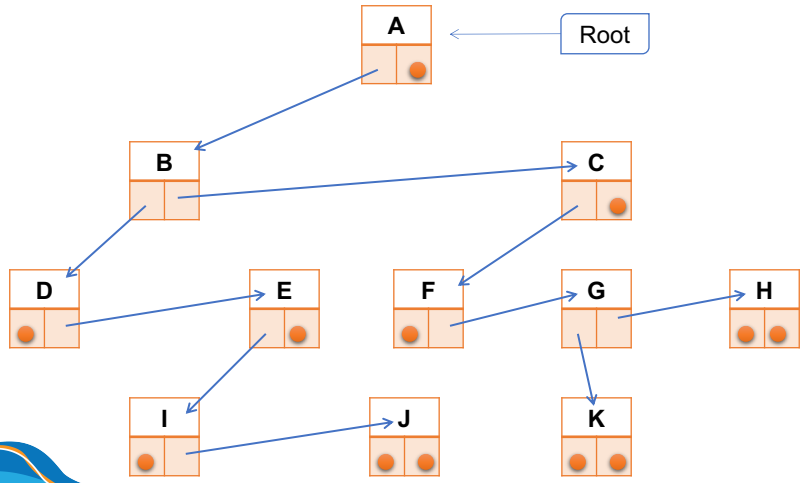
	Info	Eldest Child	Next Sibling
1	a	2	0
2	b	4	3
3	c	6	0
4	d	0	5
5	e	9	0
6	f	0	7
7	g	11	8
8	h	0	0
9	i	0	10
10	j	0	0
11	k	0	0



31

fit@hcmus

Tree Representation

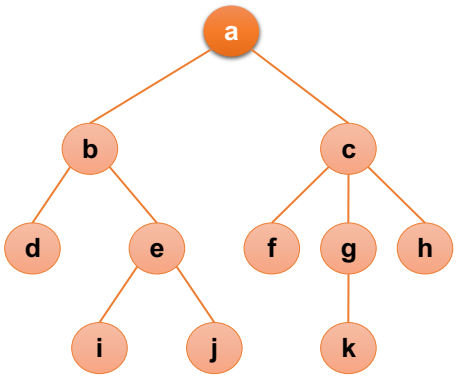


32

fit@hcmus

Tree Representation

	Info	Parent
1	a	0
2	b	1
3	c	1
4	d	2
5	e	2
6	f	3
7	g	3
8	h	3
9	i	5
10	j	5
11	k	7



33

fit@hcmus

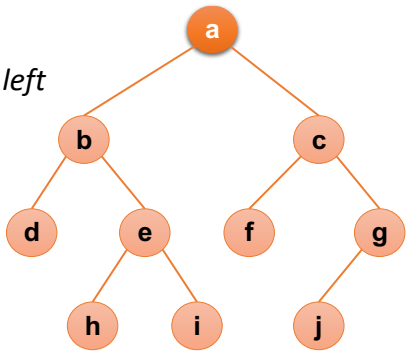
Binary Tree

34

fit@hcmus

Binary Tree

- Set T of nodes that is either empty or partitioned into disjoint subsets.
  - Single node *r*, the root
  - Two possibly empty sets that are binary trees, called *left* and *right* subtrees of *r*.
- Other definition: A rooted binary tree has a root node and every node has at most two children.



35

fit@hcmus

Types of Binary Tree

- Complete binary tree
- Full binary tree
- Perfect binary tree
- Heap

36

fit@hcmus

Perfect Binary Tree

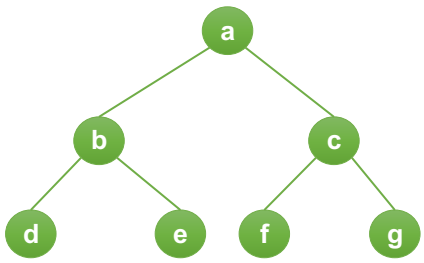
- A **perfect binary tree** is a binary tree in which
  - all interior nodes have two children
  - and all leaves have the same depth or same level.
- In a perfect binary tree of height *h*, all nodes that are at a level less than *h* have two children each.

37

fit@hcmus

Perfect Binary Tree

- If *T* is empty, *T* is a perfect binary tree of height 0.
- If *T* is not empty and has height $h > 0$, T is a perfect binary tree if its root's subtrees are both perfect binary trees of height $h - 1$.



38

fit@hcmus

Complete Binary Tree

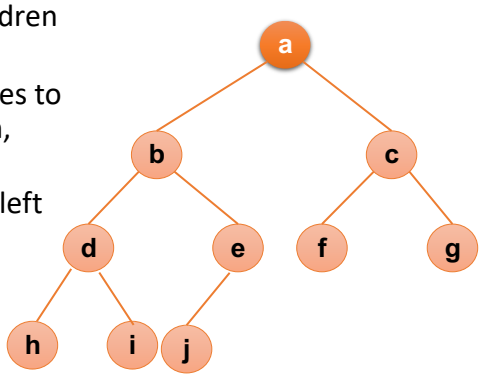
- A **complete binary tree** of height *h* is a binary tree that is **perfect** down to level *h* – 1, with level *h* filled in from left to right.
- In a complete binary tree every level, except possibly the last, is completely filled, and all nodes in the last level are as far left as possible.
- Other definition: A **complete binary tree** is a perfect binary tree whose rightmost leaves (perhaps all) have been removed.

39

fit@hcmus

Complete Binary Tree

- A binary tree is complete if
  - All nodes at level *h* – 2 and above have two children each, and
  - When a node at level *h* – 1 has children, all nodes to its left at the same level have two children each, and
  - When a node at level *h* – 1 has one child, it is a left child



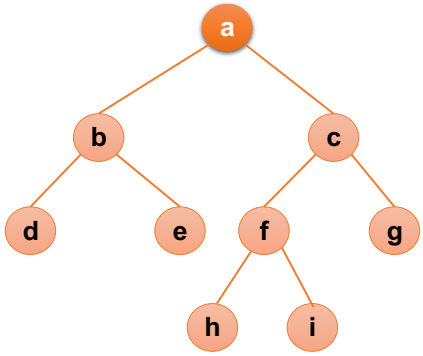
40

fit@hcmus

Full Binary Tree

- A full binary tree (sometimes referred to as a **proper binary tree** or **a plane binary tree**) is a binary tree in which ***every node has either 0 or 2 children***.

- A full binary tree is either:
  - A single vertex.
  - A tree whose root node has two subtrees, both of which are full binary trees.

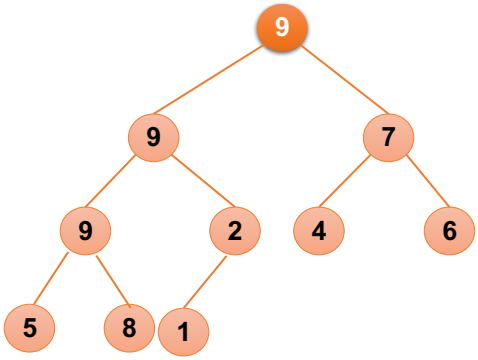


41

fit@hcmus

Heap

- A heap is a ***complete binary tree*** that either is empty or
  - Its root
    - (Max-heap): Contains a value greater than or equal to the value in each of its children, and
    - (Min-heap): Contains a value less than or equal to the value in each of its children, and
    - Has heaps as its subtrees



42

fit@hcmus

Number of Nodes

- Given a binary tree *T* height of *h*.
- What is the maximum number of nodes?
- What is the minimum number of nodes?

43

fit@hcmus

Height of Tree

- Given a binary tree *T* with *n* nodes.
- What is the maximum height of that tree?
- What is the minimum height of that tree?

44

fit@hcmus

Binary Search Tree

46

fit@hcmus

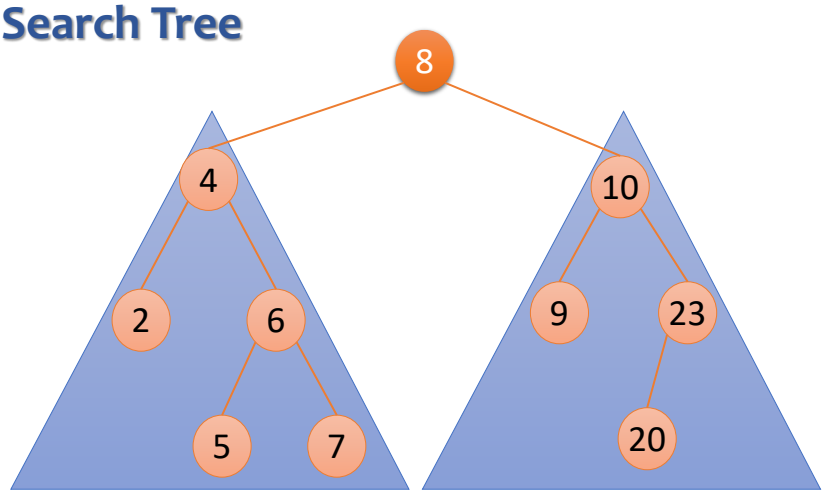
Binary Search Tree

- A binary search tree is a binary tree in which each node *n* has properties:
  - *n* 's value greater than all values in left subtree T_L
  - *n* 's value less than all values in right subtree T_R
  - Both T_R and T_L are binary search trees.

47

Binary Search Tree

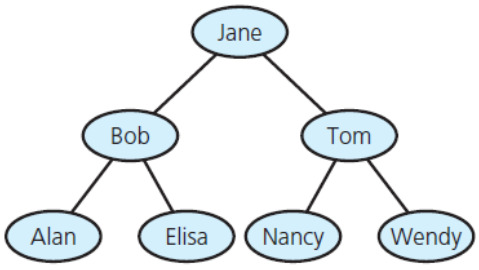
fit@hcmus

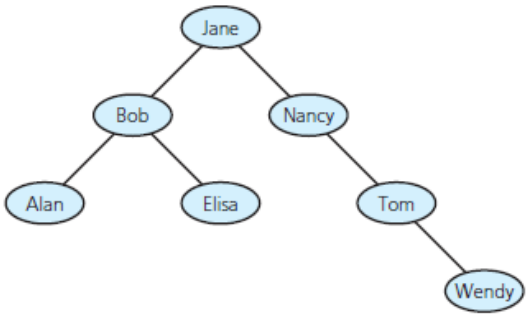


48

Binary Search Tree

fit@hcmus





49

fit@hcmus

Operations

- Insert (a key)
- Search (a key)
- Remove (a key)
- Traverse
- Sort (based on key value)
- Rotate (Left rotation, Right rotation)

50

fit@hcmus

Insertion

```
Insert (root, Data)
{
   if (root is NULL) {
         Create a new_Node containing Data
         This new_Node becomes root of the tree
   }
   //Compare root's key with Key
   if root's Key is less than Data's Key
         Insert Key to the root's RIGHT subtree
   else if root's Key is greater than Data's Key
         Insert Key to the root's LEFT subtree
   else
         Do nothing //Explain why?
}
```


51

fit@hcmus

Insertion

- Beginning with an empty binary search tree, what binary search tree is formed when you insert the following values in the order given?

15, 5, 12, 8, 23, 1, 17, 21

52

fit@hcmus

Insertion

- Beginning with an empty binary search tree, what binary search tree is formed when you insert the following values in the order given?
  - W, T, N, J, E, B, A
  - W, T, N, A, B, E, J
  - A, B, W, J, N, T, E
  - B, T, E, A, N, W, J

53

fit@hcmus

Search

```
Search (root, Data)
{
    if (root is NULL) {
          return NOT_FOUND;
    }
    //Compare root's key with Key
    if root's Key is less than Data's Key
          Search Data in the root's RIGHT subtree
    else if root's Key is greater than Data's Key
          Search Data in the root's LEFT subtree
    else
          return FOUND //Explain why?
}
```


54

fit@hcmus

Deletion

- When we delete a node, three possibilities arise.
- Node to be deleted:
  - **is leaf:**
    - Simply remove from the tree.
  - has **only one child:**
    - Copy the child to the node and delete the child
  - has **two children:**
    - Find in-order successor (predecessor) ***S_Node*** of the node.
    - Copy contents of ***S_Node*** to the node and delete the ***S_Node***.

55

fit@hcmus

Traversals

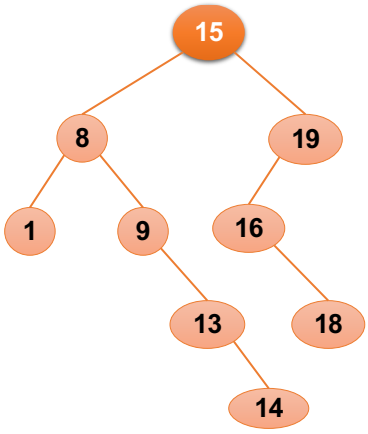
- Pre-order:
  Node - Left - Right
- In-order:
  Left - Node - Right
- Post-order:
  Left - Right - Node

56

fit@hcmus

Traversals

- What are the pre-order, in-order and post-order traversals of this binary search tree?

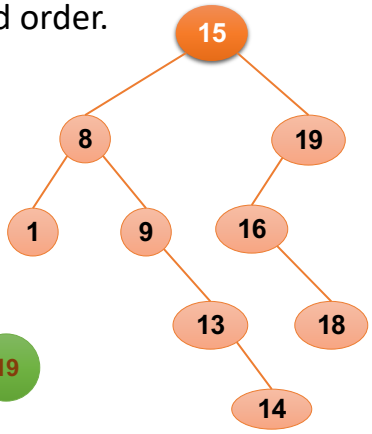


57

fit@hcmus

Sort

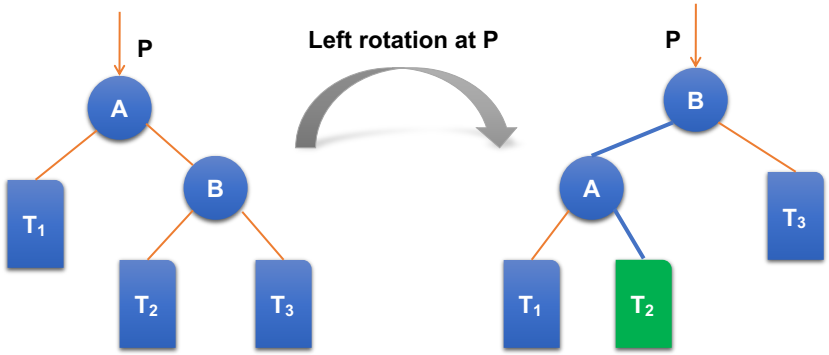
- In-order traversal gives the key values in sorted order.



58

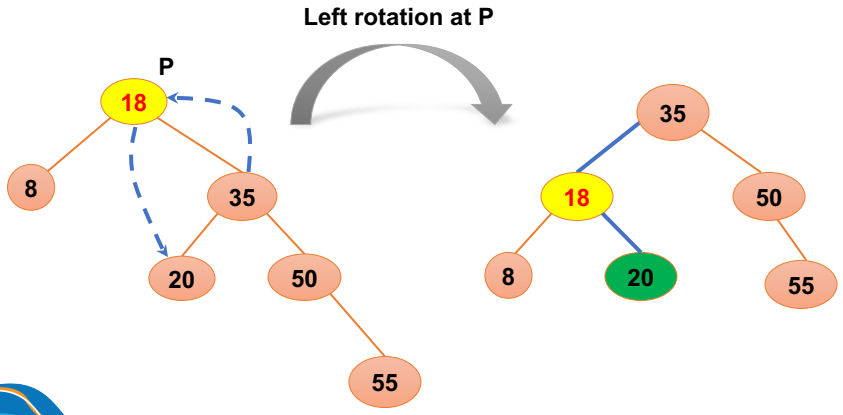
fit@hcmus

Left Rotation



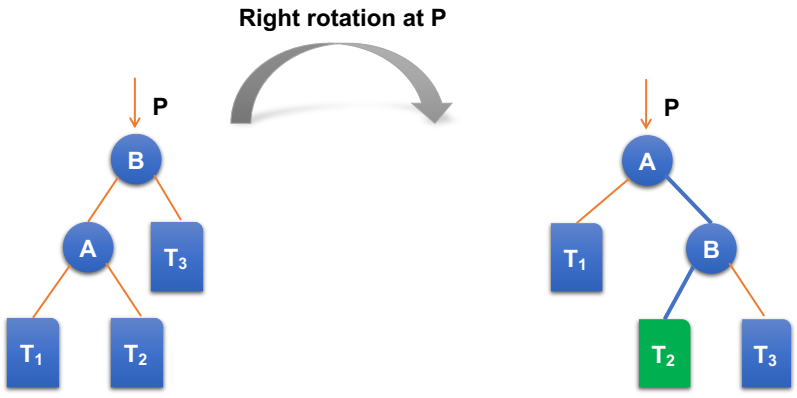
59

Left Rotation



60

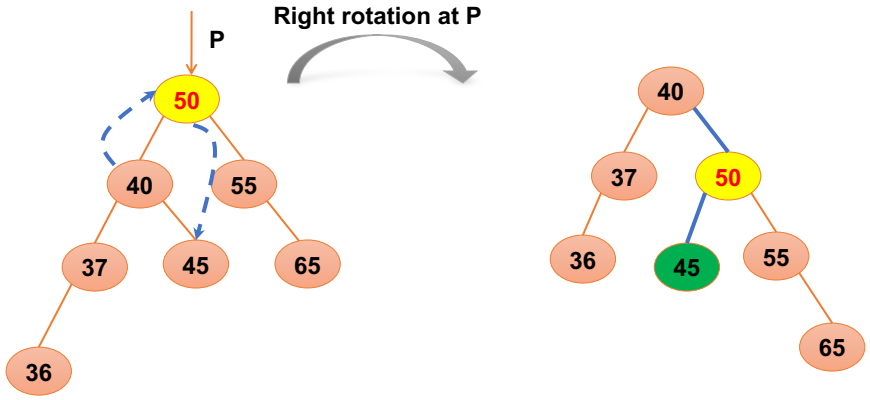
Right Rotation



61

fit@hcmus

Right Rotation



62

fit@hcmus

Efficiency of Binary Search Tree Operations

Operation	Average case	Worst case
Retrieval	O(log *n*)	O(*n*)
Insertion	O(log *n*)	O(*n*)
Removal	O(log *n*)	O(*n*)
Traversal	O(*n*)	O(*n*)

63

fit@hcmus

Very Bad Binary Search Tree

- Beginning with an empty binary search tree, what binary search tree is formed when inserting the following values in the order given?

2, 4, 6, 8, 10, 12, 14, 18, 20

64