SCC120 Fundamentals of Computer Science
Unit 9: Trees (Representations)

Jidong Yuan
yuanjd@bjtu.edu.cn

Representations of Trees

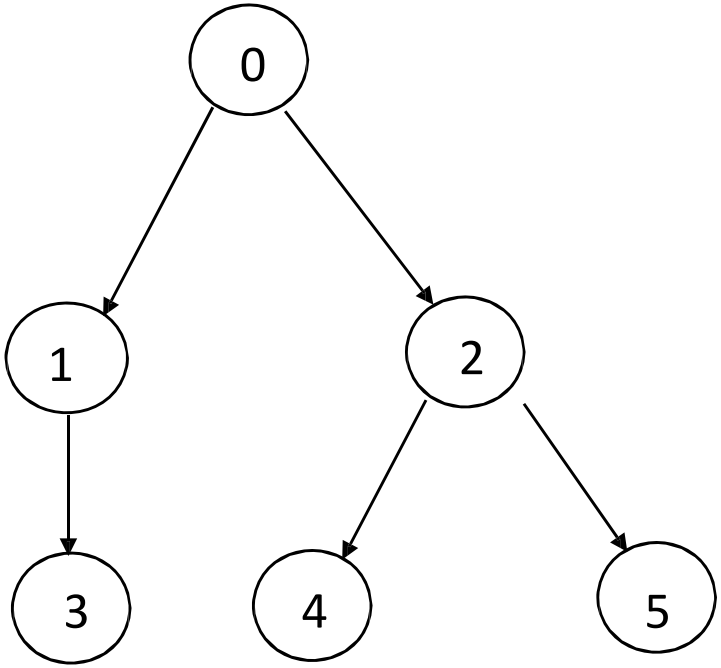
- There are many ways to implement trees
  - Different ones for different types of use
- The methods introduced for graphs can be adapted for trees (but there are also others just for trees)
- We look at three methods:
  (1) Adjacency Matrix
  (2) Parent Vector
  (3) String Representations

(1) An Adjacency Matrix

- This method uses a 2D array of binary (or boolean) values
  - Each element indicates the presence or absence of an edge
  - Node values can be stored in a separate array

An Adjacency Matrix

	0	1	2	3	4	5
0	0	1	1	0	0	0
1	0	0	0	1	0	0
2	0	0	0	0	1	1
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0



An Adjacency Matrix

A preorder traversal method for this representation:

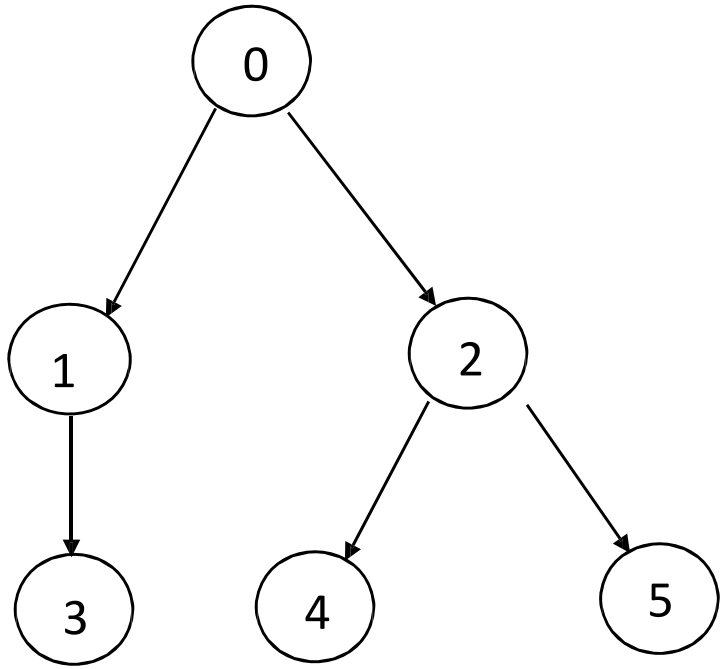
```
public void depthFirstTraversal(Tree T, Node N)
{
   visitNode(N);
   for (int X=0; X<w; X++)
      if (A[N,X])
         depthFirstTraversal(T, X);
}
```

(2) A Parent Vector

- In the adjacency matrix, each column is:
  - Either all zeros (for the root node)
  - Or contains exactly one value
- We can collapse all the rows together
- Giving a linear array P in which each element holds the index of the node's parent
- This is sometimes called a *parent vector*

A Parent Vector

0	-1
1	0
2	0
3	1
4	2
5	2



A Parent Vector

A preorder traversal method for this representation:

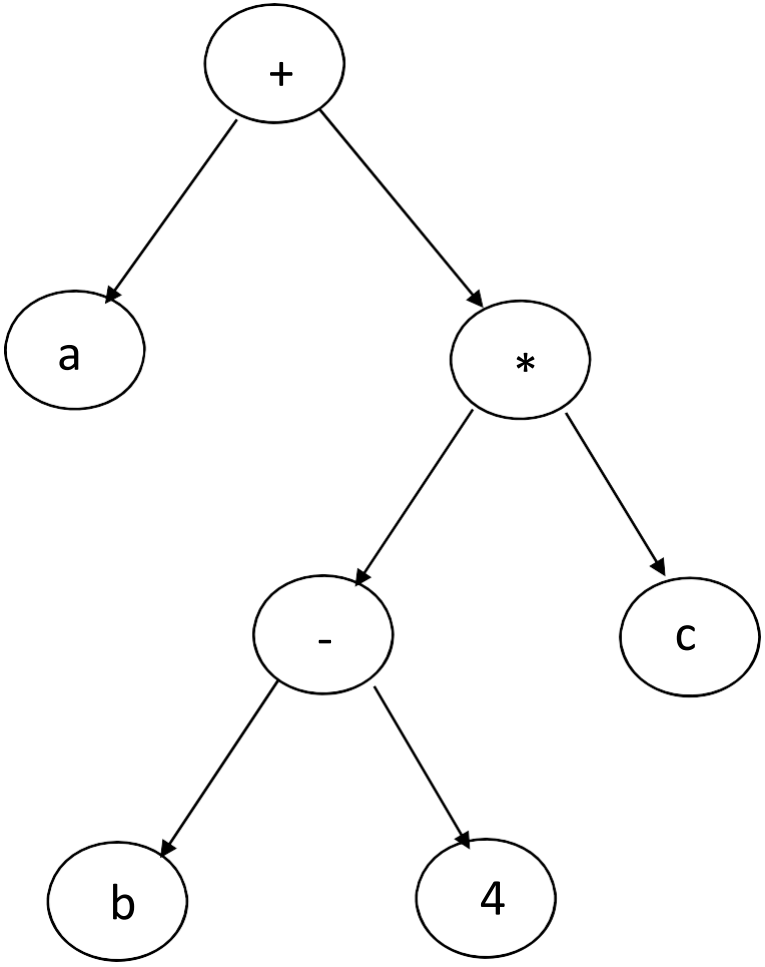
```
public void depthFirstTraversal(Tree T, Node N)
{
   visitNode(N);
   for (int X=0; X<w; X++)
      if (P[X] == N)
         depthFirstTraversal(T, X);
}
```

(3) String Representations

- From tree to string: We can use preorder or postorder traversals to give an (unambiguous) string representation for a tree

- From string to tree: From +a*-b4c we can generate this tree

  This works because we can distinguish *operands* (leaves) from *operators* (non-leaves)



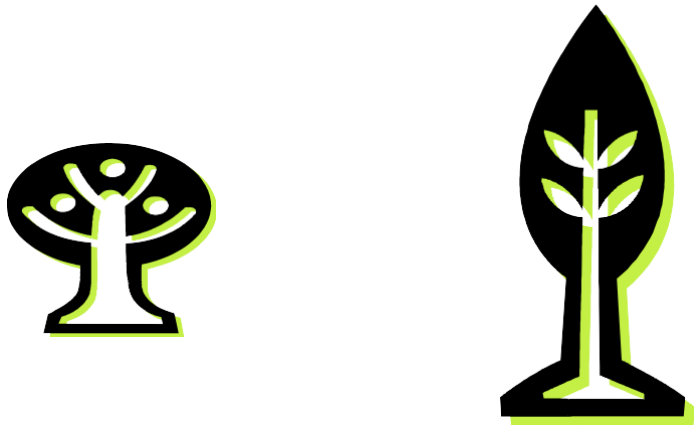
- I’ve picked a number from 1 to 100. You can only ask me yes or no questions (i.e. I only reply yes or no) to find the number. What questions would you ask to find the number as quickly as possible?
- If the number is from 1 to N, how many questions in big O notation do you need to ask?

Searching a Tree

- If you are searching a tree, it is more efficient if every comparison of what you are searching for with an element in the tree is as “useful” as possible
- When is it as “useful” as possible?

Fat/Thin Trees

- It is best if each comparison divides the remaining possibilities roughly in half
  - Reject half and keep half for further investigation
- It is best if the trees are fat and short, rather than thin trees
  - Or if the tree has smaller height (for the same number of total nodes in the tree)
  - Or if the tree is as close as possible to *uniform*
  - Or if the tree is more balanced

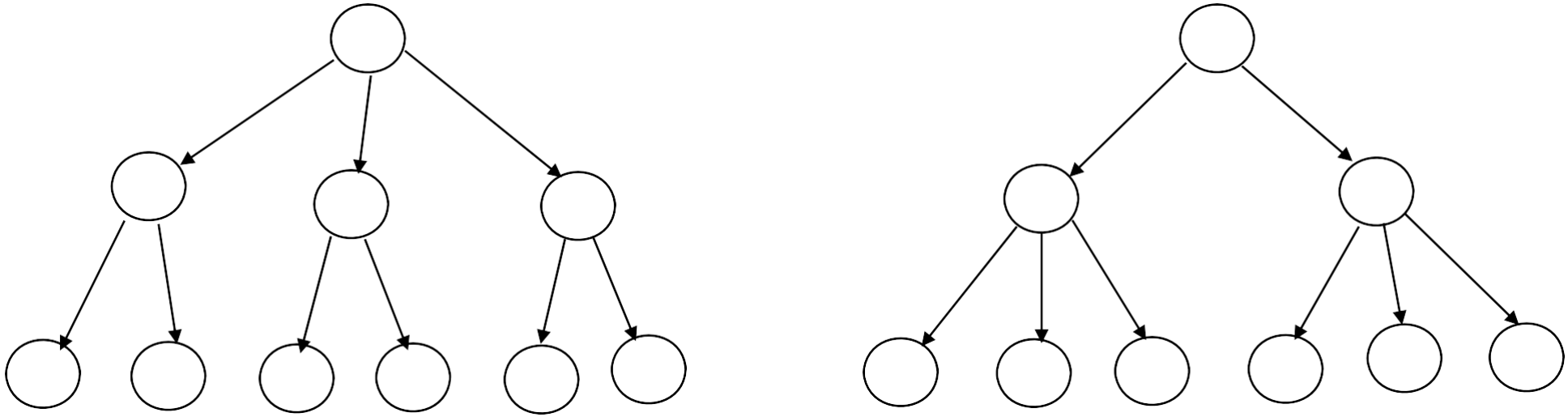


Motivation of Balancing a Tree

- If we are continually adding and deleting nodes from the tree, it may not be optimally efficient for searching
- So we may want to “re-balance” the tree every so often, after a number of additions and deletions, so that searching can be fast

Example of Balanced Trees: 2-3 Trees

- Each non-leaf node has two or three child nodes
- All paths from the root node to a leaf node are the same length

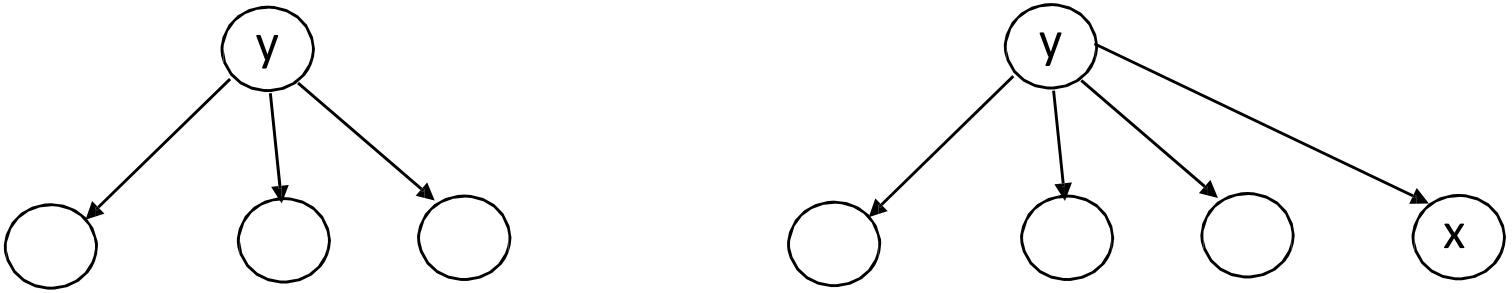


Maintaining 2-3 Trees

- "Add node" operation for 2-3 trees
  - Start with (a part of a) 2-3 tree on the left
  - Add node x as the child node of node y
  - If y already has two child nodes, this new node x becomes the third child node, and it is still a 2-3 tree

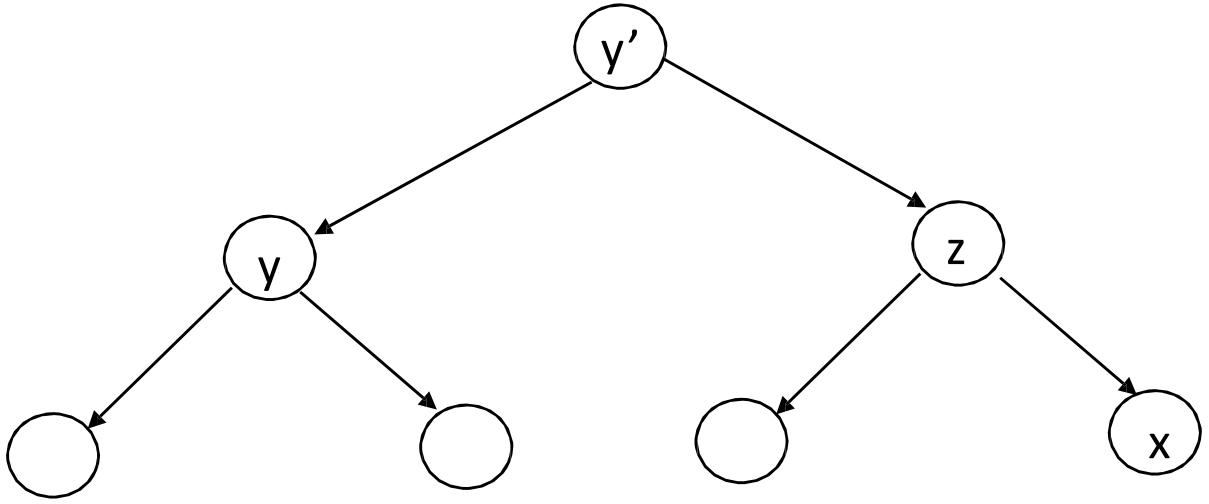
Maintaining 2-3 Trees

– If y originally has three child nodes, it now has four child nodes



Maintaining 2-3 Trees

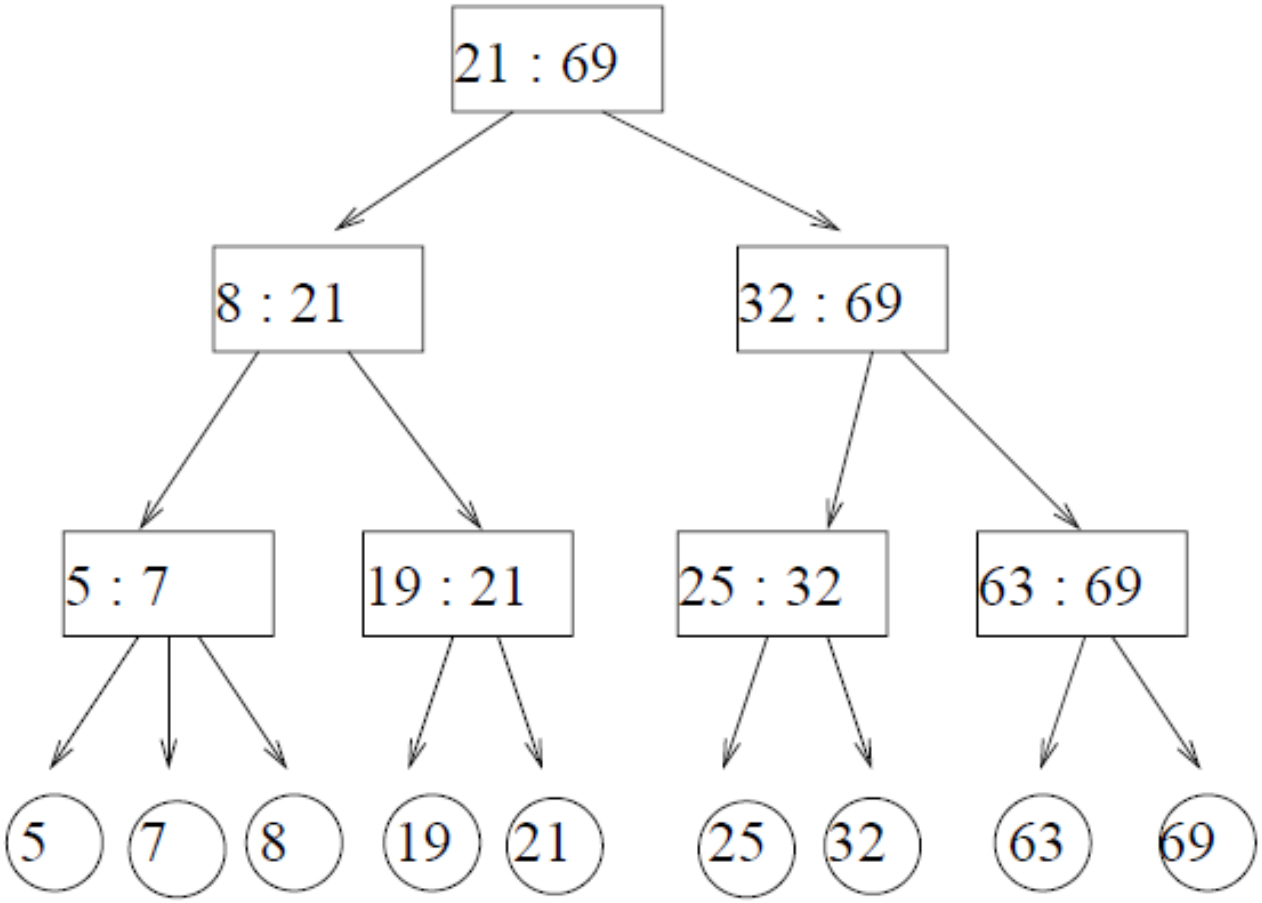
- We make a new node z
- We keep two of the four child nodes for y, and let the other two be the child nodes of z
- Then we make y and z sibling nodes by making z a child node of y's parent node y' (if y was the root of the tree, then we add node y')



Maintaining 2-3 Trees

  - If y’ now has three child nodes, we are done
  - If y’ now has four child nodes, we repeat this procedure on y’, working up towards the root
  - Eventually the whole tree is a 2-3 tree again
- “Delete node” operation for 2-3 trees is similar
- This is not the whole story about re-balancing, but it illustrates the idea

2-3 Trees: Example



2-3 Trees: Example

2-3 Trees

- leaf nodes
  - contain data item
  - all leaves on same level
- nonleaf node
  - has 2 or 3 children
  - two search values
    - largest item in left subtree
    - largest item in middle subtree
- smallest 2-3 tree
  - only one node (leaf node)
- balanced, ordered tree

2-3 Trees: Example

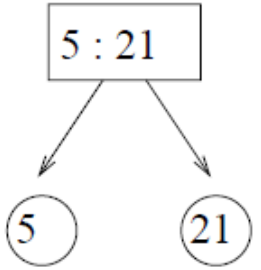
- How to build a 2-3 tree?

- Let us assume that the sequence to be inserted is “5, 21, 8, 63, 69, 32, 7, 19, 25”

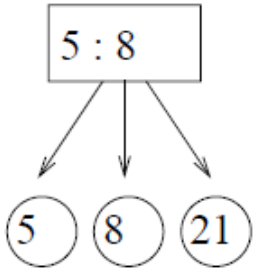
2-3 Trees: Example

Insert 5

Insert 21

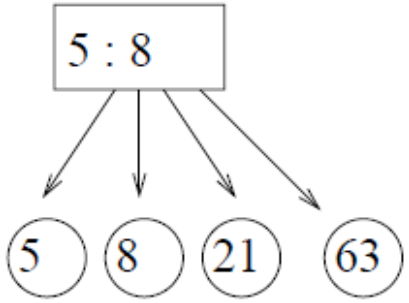


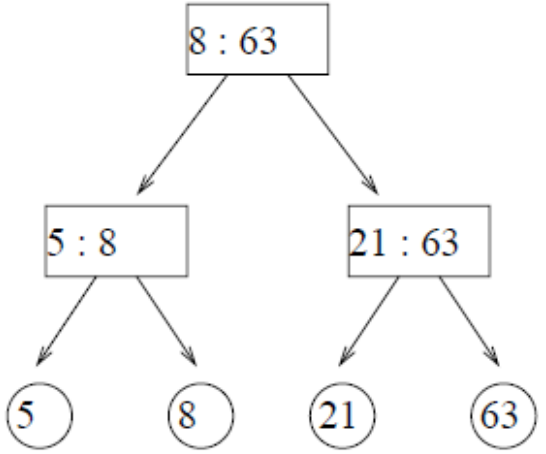
Insert 8



2-3 Trees: Example

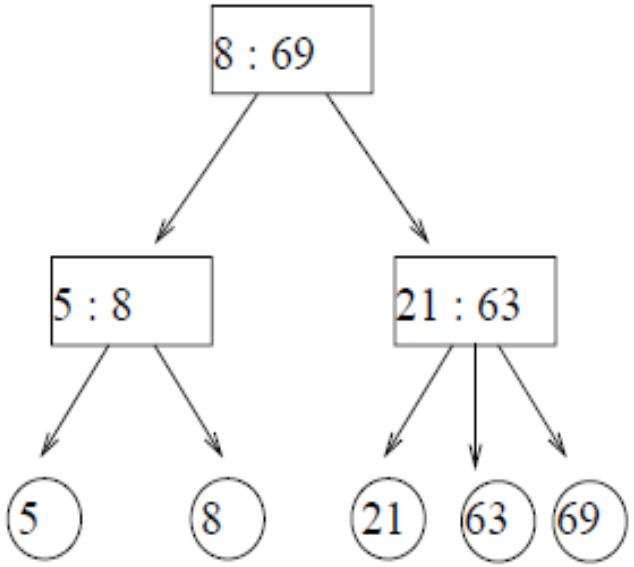
Insert 63





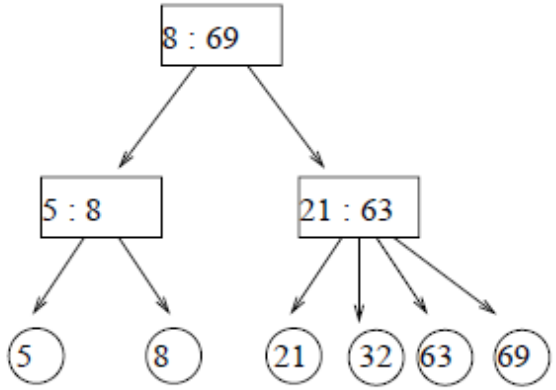
2-3 Trees: Example

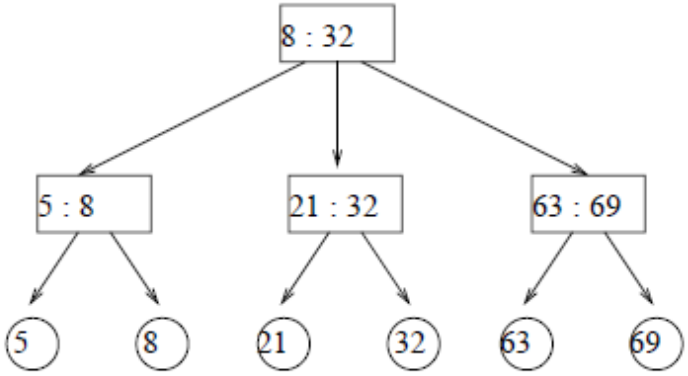
Insert 69



2-3 Trees: Example

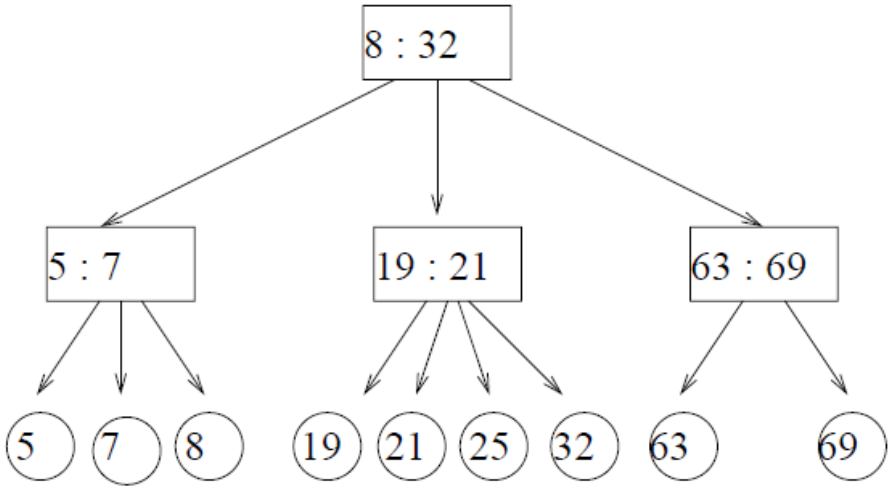
Insert 32

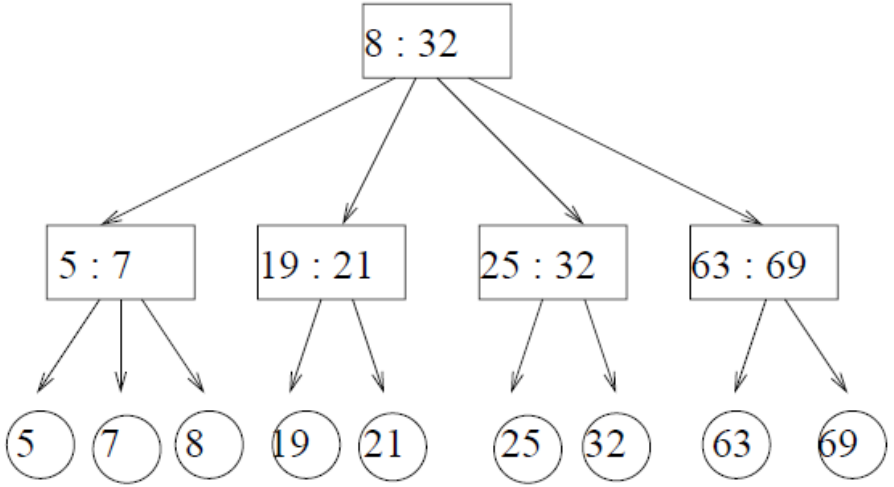




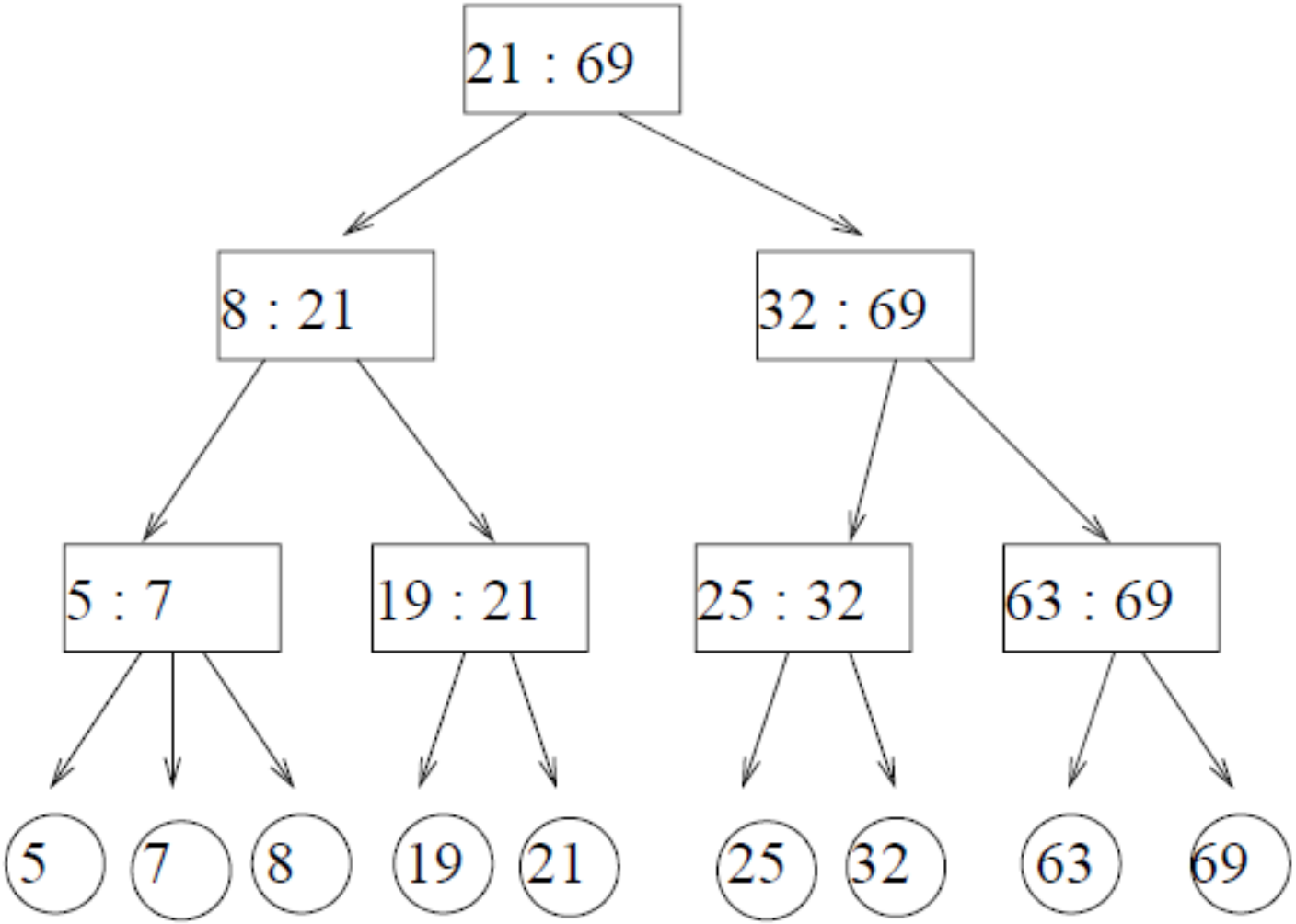
2-3 Trees: Example

Insert 7, 19, 25

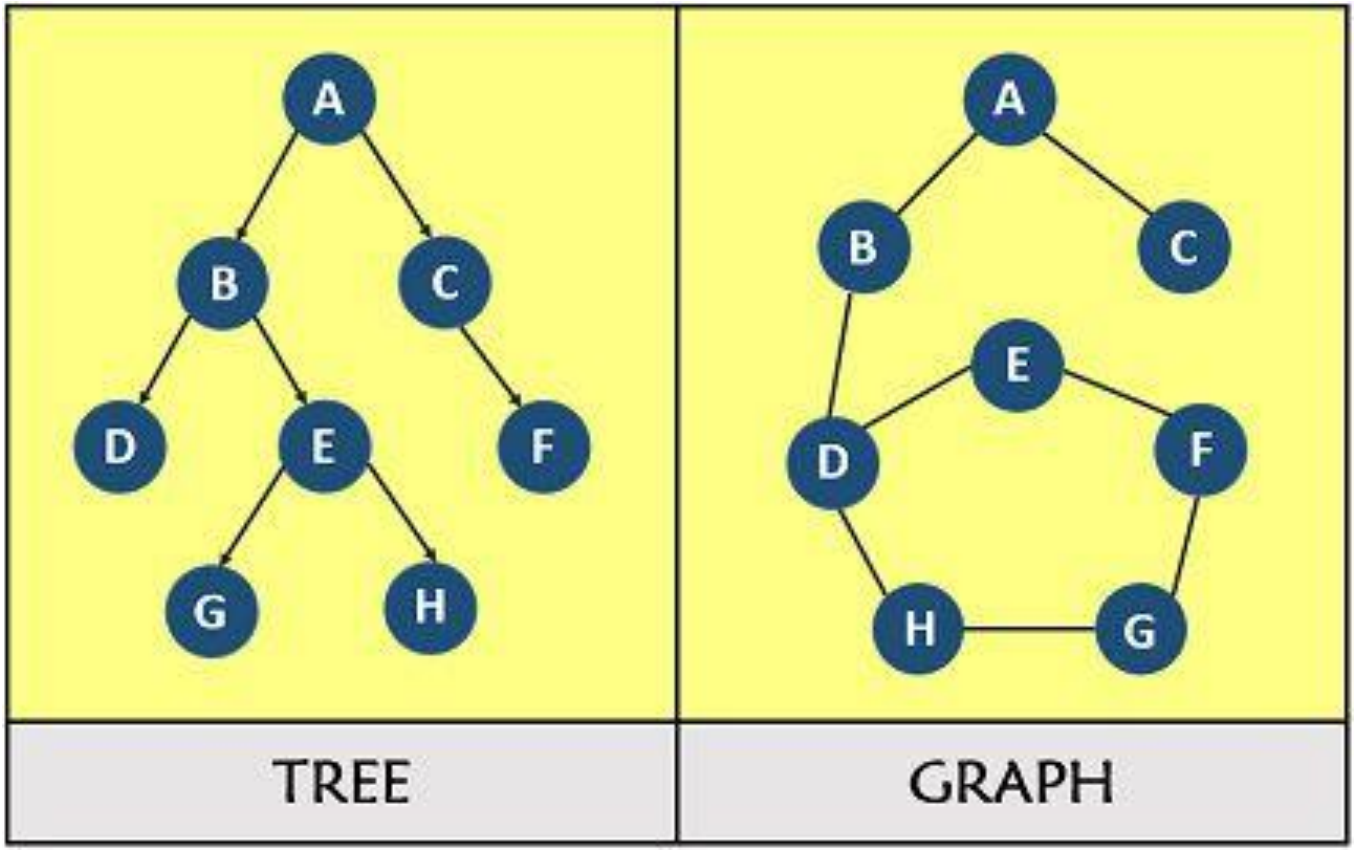




2-3 Trees: Example



Difference Between Tree and Graph



Difference Between Tree and Graph

BASIS FOR COMPARISON	TREE	GRAPH
Path	Only one between two vertices.	More than one path is allowed.
Root node	It has exactly one root node.	Graph doesn't have a root node.
Loops	No loops are permitted.	Graph can have loops.
Complexity	Less complex	More complex comparatively
Traversal techniques	Pre-order, In-order and Post-order.	Breadth-first search and depth-first search.
Number of edges	n-1 (where n is the number of nodes)	Not defined
Model type	Hierarchical	Network

SCC120 ADT (weeks 7-13)

- Week 7 Abstractions; Set
  Stack
- Week 8 Queues Priority
  Queues
- Week 9 Graphs (Terminology)
  Graphs (Traversals)
- Week 10 Graphs (Traversals)
  Graphs (Representations)
- Week 11 Trees (Terminology)
  Trees (Traversals)
- *Week 13* *Trees (Representations)*
  Recap