Trees

Carlos Delgado Kloos
Mª Carmen Fernández Panadero
Raquel M. Crespo García
Dep. Ingeniería Telemática
Univ. Carlos III de Madrid

cdk@it.uc3m.es

Contents

rcrespo@it.uc3m.es

Quote

- *"The structure of concepts is formally called a **hierarchy** and since ancient times has been a basic structure for all western knowledge. Kingdoms, empires, churches, armies have all been structured into hierarchies. Tables of contents of reference material are so structured, mechanical assemblies, computer software, all scientific and technical knowledge is so structured..."*

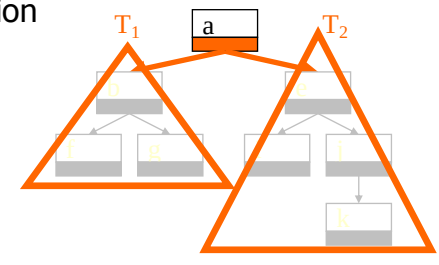
  -- Robert M. Pirsig: *Zen and the Art of Motorcycle Maintenance*

cdk@it.uc3m.es

Tress

Concept and characteristics

- A **tree** is a non-linear data structure that stores the elements **hyerarchically**
- Trees can be defined in two ways:
  - Non-recursive definition
  - Recursive definition



Recursive definition

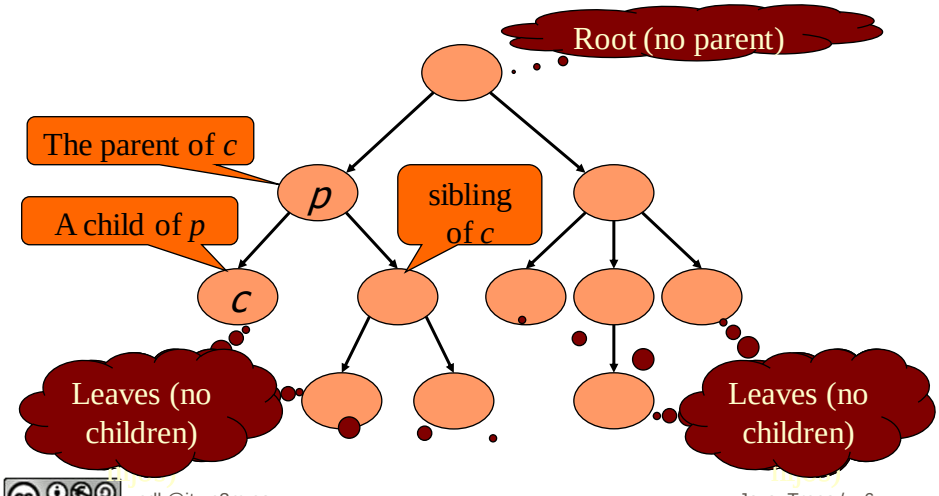
mcfp@it.uc3m.es

Non-recursive definition

- A tree consists of a set of **nodes** and a set of **edges**, such that:
  - There is a special node called **root**
  - For each node *c*, except for the root, there is one edge from another node *p* (*p* is **parent** of *c*, *c* is one of the children of *p*)
  - For each node there is a **unique path** (sequence of edges) from the root

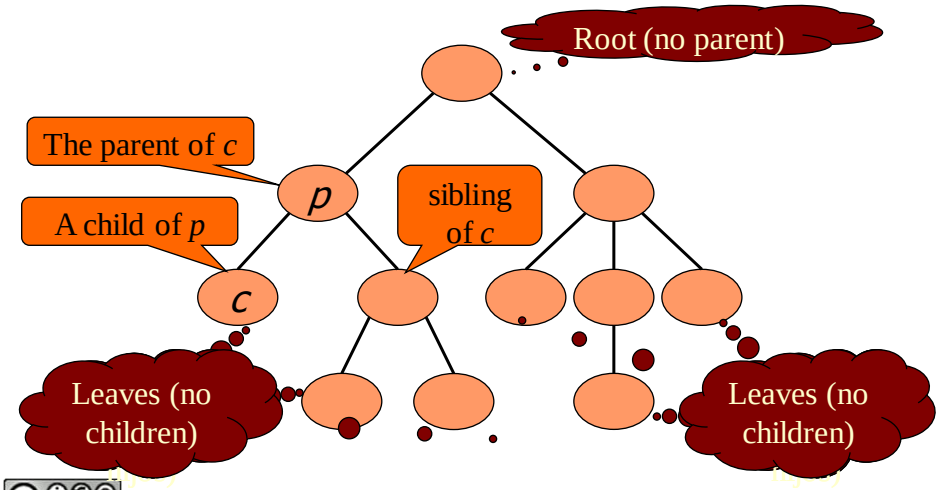
cdk@it.uc3m.es

Example



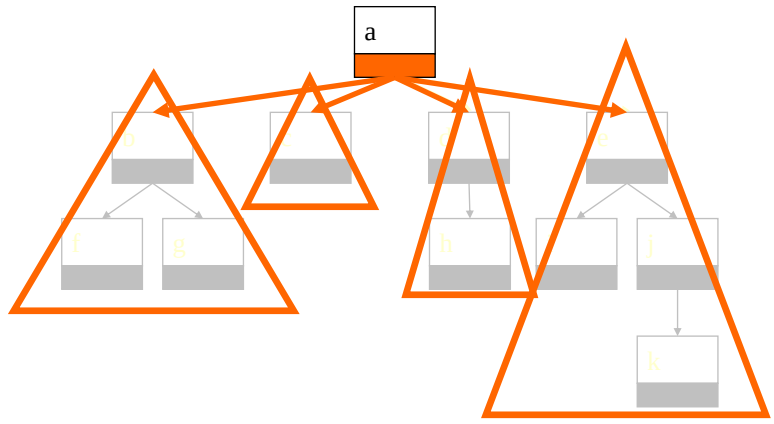
cdk@it.uc3m.es

Example

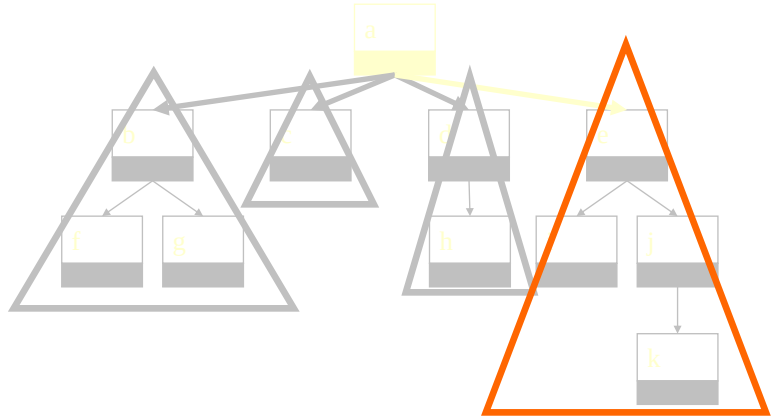


Trees

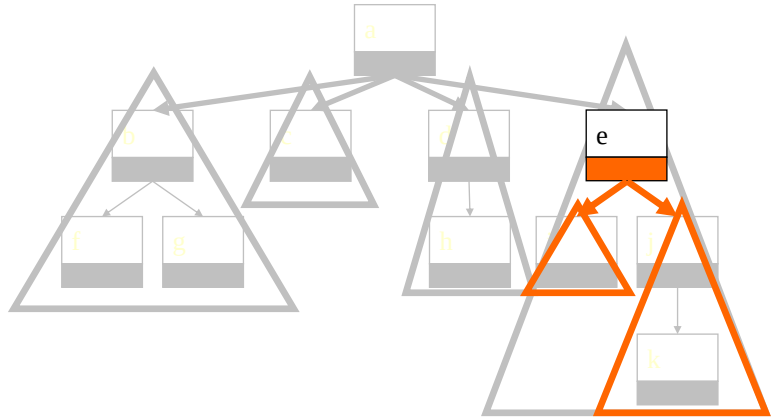
Recursive definition



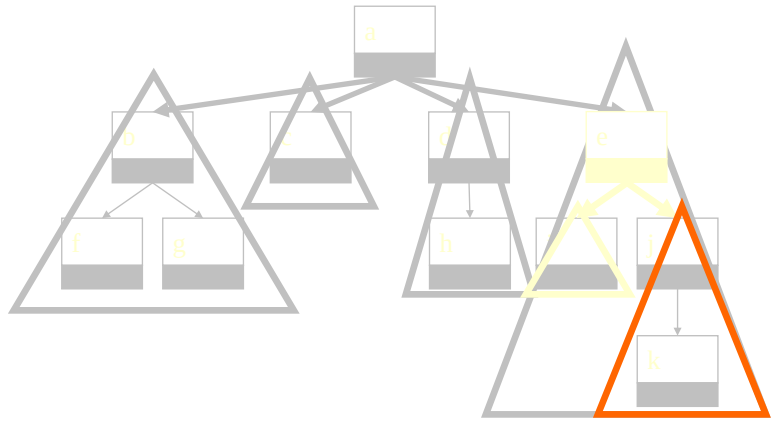
Trees
Recursive definition



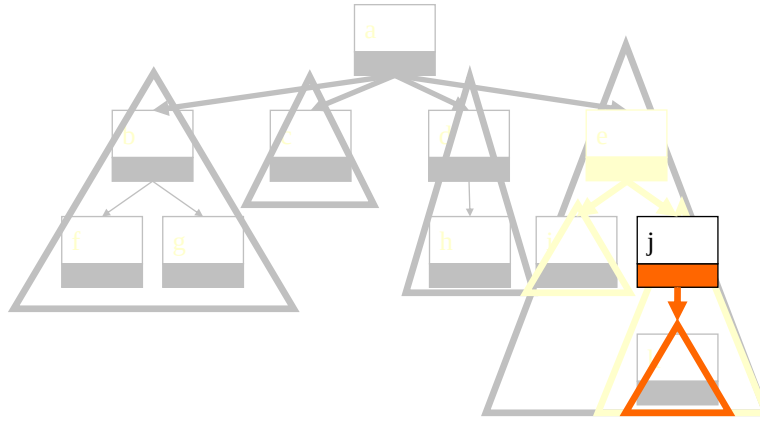
Trees
Recursive definition



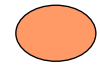
Trees
Recursive definition



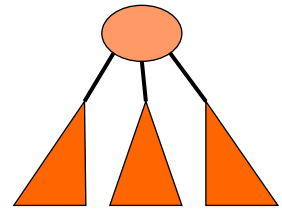
Trees
Recursive definition



Recursive definition (1)

- A tree is
  - A node 
  - or a node and subtrees connected to the node by means of an edge to its root

Doesn't include the empty tree



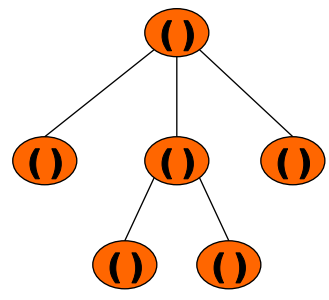
Recursive definition (2)

- A tree is
  - empty
  - or a node and zero or more non-empty subtrees connected to the node by means of an edge to its root

Examples

- File system
- Structure of a book or a document
- Decision tree
- Arithmetic expressions

Example
Expressions

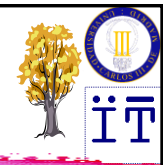


(()(()())())

Terminology

- A node is **external**, if it doesn't have children (it is a ***leaf***)
- A node is **internal**, if it has one or more children
- A node is **ancestor** of another one, if it is its parent or an ancestor of its parent
- A node is **descendent** of another one, if the latter is ancestor of the former
- The descendents of a node determine a **subtree** where this node acts as the root

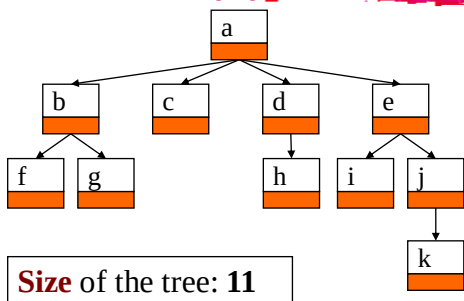
Terminology



- A **path** from one node to another one, is a sequence of consecutive edges between the nodes.
  - Its *length* is the number of edges it is composed of.
- The **depth** of a node is the length of the path from the root to this node.
- The **height** of a tree is the depth of the deepest node.
- The **size** of a tree is the number of nodes.

Example

Terminology and properties

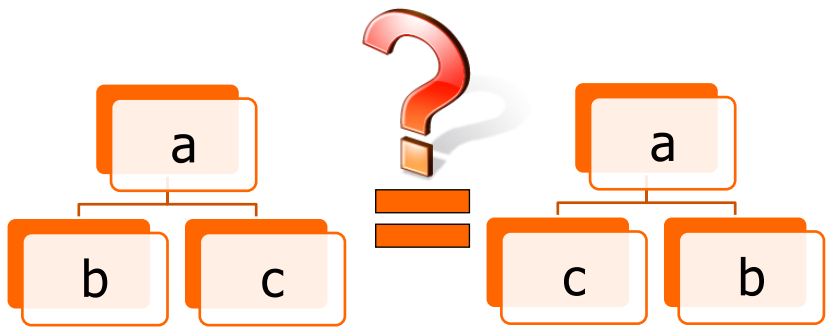


Size of the tree: **11**
Height of the tree: **3**

Node	Height	Depth	Size	Internal / External
a	**3**	0	**11**	Internal
b	1	1	3	Internal
c	0	1	1	External
d	1	1	2	Internal
e	2	1	4	Internal
f	0	2	1	External
g	0	2	1	External
h	0	2	1	External
i	0	2	1	External
j	1	2	2	Internal
k	0	**3**	1	External

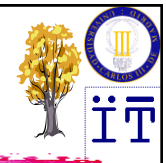
Terminology

Ordered tree



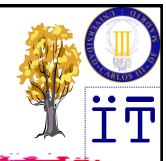
- A tree is **ordered**, if for each node there exists a linear ordering for its children.

Terminology
Binary tree



- A **binary** tree is an ordered tree, where each node has 0, ~~1~~ or 2 children (the left and the right child).

Basic algorithms

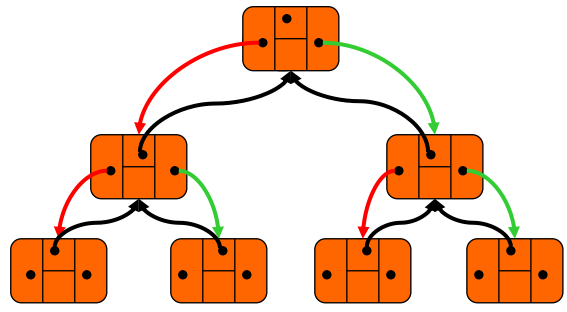


- Size (number of nodes)
- Depth of a node
- Height
- Traversals
  - Euler
  - Pre-, in- and post-order
- *To simplify, we assume binary trees*

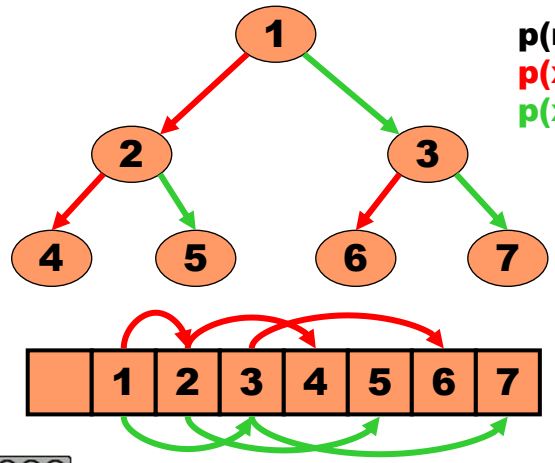
Implementations

- Based on a linked structure
- Sequence-based

Implementation based on a linked structure



Sequence-based implementation



p(rot)=1
p(x.left)=2*p(x)
p(x.right)=2*p(x)+1

Class Binary tree...

```
public class BTree {
  protected BNode root;

  BTree() {
    root = null;
  }

  BTree(Object info){
    root = new Bnode(info);
  }
```

...Class Binary tree...

```
public int size() {
  int size = 0;
  if (root != null)
    size=root.size();
  return size;
}
```

```
public int height(){
  int h = -1;
  if (root != null)
    h=root.height();
  return h;
}
```


...Class Binary tree

```
public void preorder() {
  if (root!=null) root.preorder();
}
public void inorder() {
  if (root!=null) root.inorder();
}
public void postorder() {
  if (root!=null) root.postorder();
}
}
```

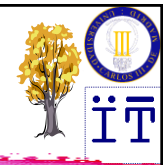
Class Binary node...

```
class BNode {
  private Object info;
  private BNode left;
  private BNode right;

  BNode() {
    this(null);
  }
  BNode(Object info) {
    this(info,null,null);
  }
  BNode(Object info, BNode l, BNode r) {
    this.info=info;
    left=l;
    right=r;
  }
```


...Class Binary node...

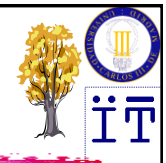
```
  int size (){...;}
  int height (){...;}

  void preorder (){...;}
  void inorder (){...;}
  void postorder (){...;}
}
```

BNode Class: Size



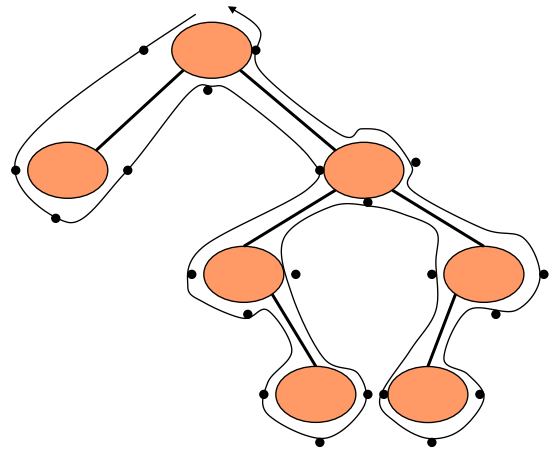
```
int size (){
  int size = 1;
  if (left != null)
    size += left.size();
  if (right != null)
    size += right.size();
  return size;
}
```


BNode Class: Height



```
int height() {
 int hl = -1;
 int hr = -1;
 if (left != null)
   hl = left.height();
 if (right != null)
   hr = right.height();
 return 1 + Math.max(hl, hr);
}
```

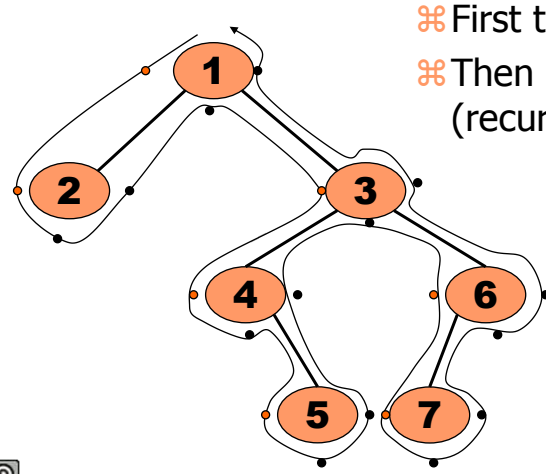
Euler traversal



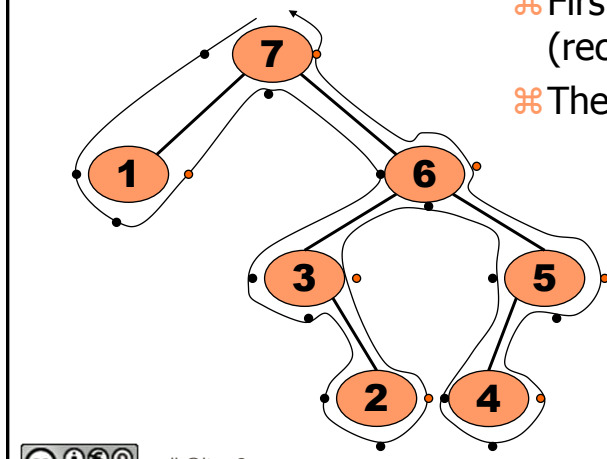
Also applicable to non-binary trees!

Preorder traversal

- First the node
- Then its children (recursively)

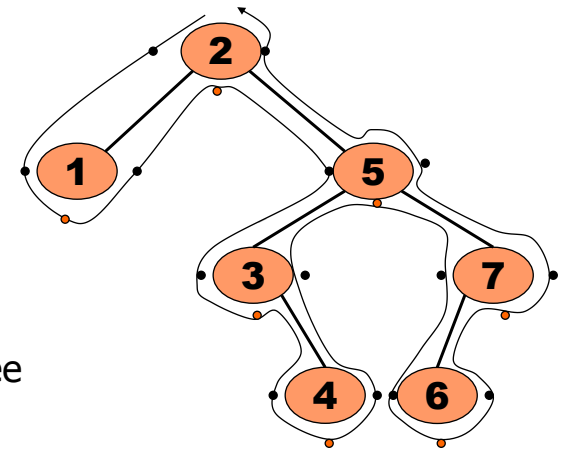


Postorder traversal



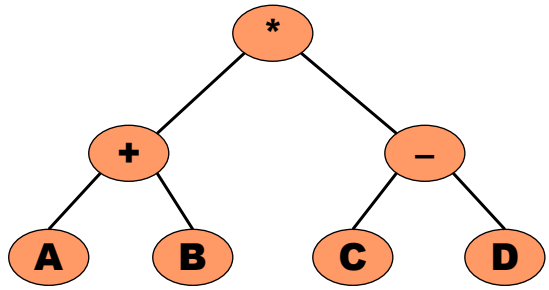
- First the children trees (recursively)
- Then the node

Inorder (symmetric) traversal



- First the left tree (recursively)
- Then the node
- Finally, the right tree (recursively)

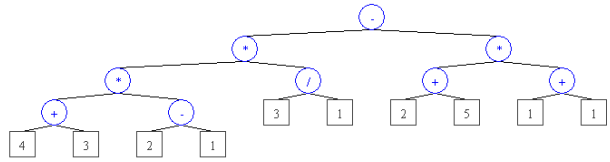
(A+B)*(C–D)



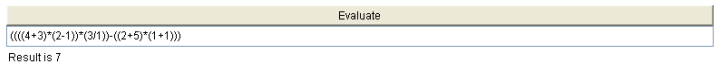
Example

Infix	Prefix	Postfix
A+B	+AB	AB+
A+B–C	–+ABC	AB+C–
(A+B)*(C–D)	*+AB–CD	AB+CD–*

Activity

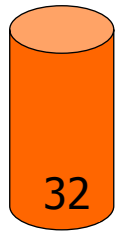


- Visualize expressions as trees
  http://www.cs.jhu.edu/~goodrich/dsa/05trees/Demo1/

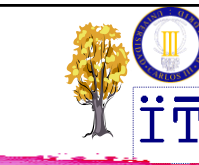


Postfix notation

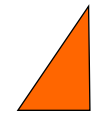
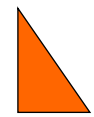
- HP calculators, RPN=Reverse Polish Not.
- Stack to store objects
- Eg: ~~3 5 + 6 2 - *~~



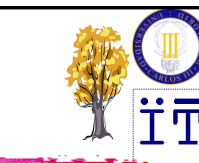
Class BNode: preorder



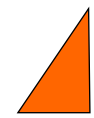
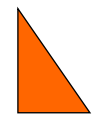
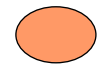
```
void preorder (){
  System.out.println(info);
  if (left != null)
    left.preorder();
  if (right != null)
    right.preorder();
}
```

Class BNode: postorder

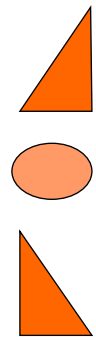


```
void postorder (){
  if (left != null)
    left.postorder();
  if (right != null)
    right.postorder();
  System.out.println(info);
}
```

Class BNode: inorder

```
void inorder (){
  if (left != null)
    left.inorder();
  System.out.println(info);
  if (right != null)
    right.inorder();
}
```



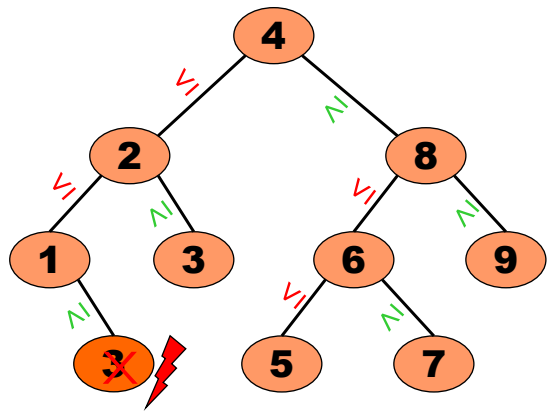
Properties of binary trees

- Let
  - E=Number of external nodes
  - I=Number of internal nodes
  - N=Size=E+I
  - H=Height
- then
  - $E=I+1$
  - $H+1 \leq E \leq 2^H$ $H \leq I \leq 2^H-1$ $2*H+1 \leq N \leq 2^{H+1}-1$
  - $\log_2(N+1)-1 \leq H \leq (N-1)/2$

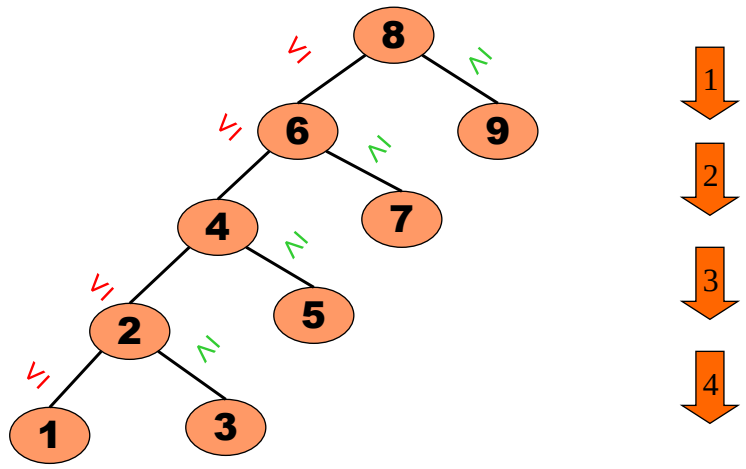
Binary search trees

- A **binary search tree** is a binary tree where for each node *n*,
  - all the keys in the **left** subtree are **smaller** (or equal) than the key of *n*
  - and all those of the **right** subtree **larger** (or equal)

Example



Example



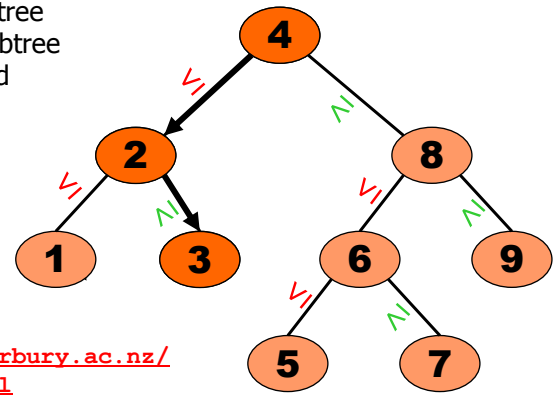
Operations

- Search
- Insertion
- Extraction

Search

Searching "3":

- 3<4: go to left subtree
- 3>2: go to right subtree
- 3=3: element found

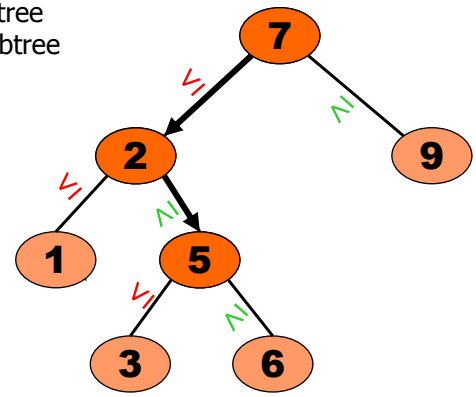


http://www.cosc.canterbury.ac.nz/mukundan/dsal/BST.html

Insertion

Inserting "6":

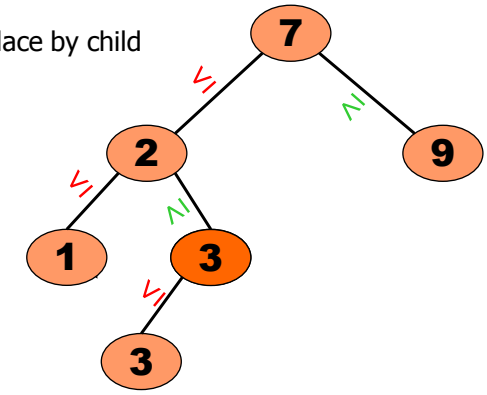
- 6<7: go to left subtree
- 6>2: go to right subtree
- when hole: insert



Extraction (1)

Extracting "5":

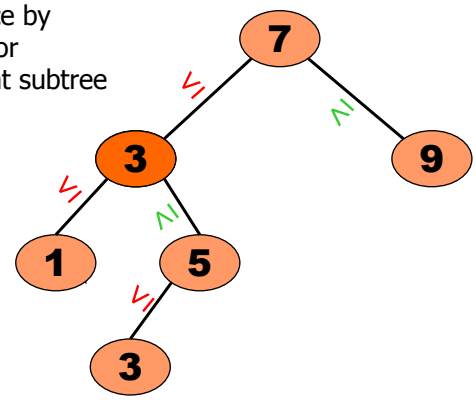
- if leaf: extract
- if degenerate: replace by child



Extraction (2)

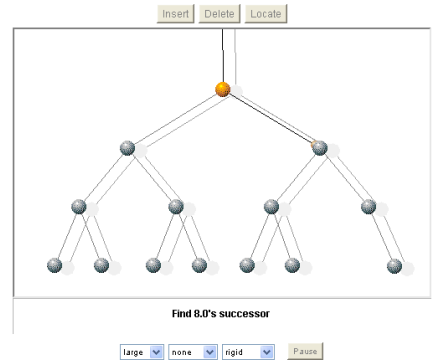
Extracting "2":

- if 2 children: replace by
  - largest at left, or
  - smallest at right subtree



Activity

- See animation of binary search trees
  http://www.ibr.cs.tu-bs.de/courses/ss98/audii/applets/BST/BST-Example.html



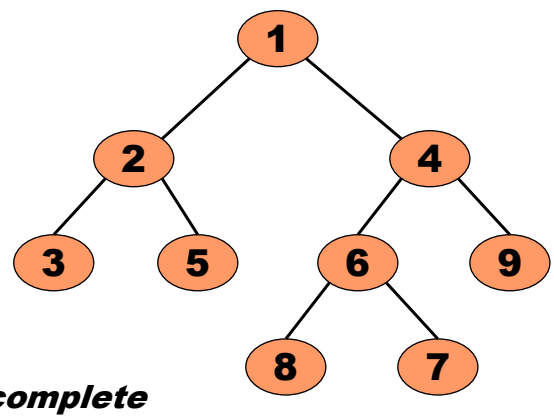
Heaps

- A binary **heap** is a complete binary tree where every node has a key greater(*) than or equal to the key of its parent.
  - Usually, heaps refer to binary heaps
  - * It could also be defined as less or equal (the order criteria is arbitrary)
- Utitity
  - Priority queues
  - Sorting algorithm

Heaps properties

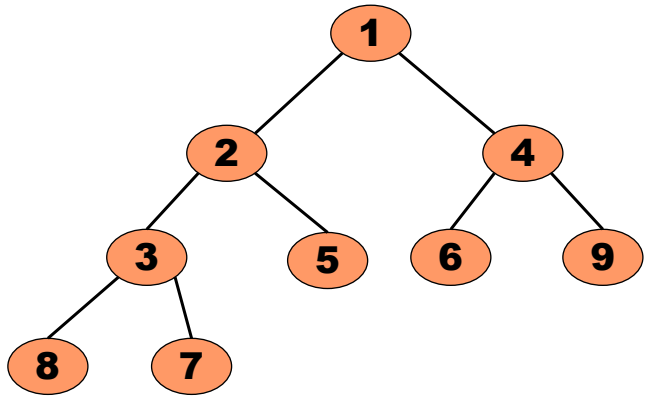
- A heap fulfils two properties:
  - Heap property: for each node *n* (except for the root), its key is larger or equal than the one of its parent.
  - Completeness

Example: heap property

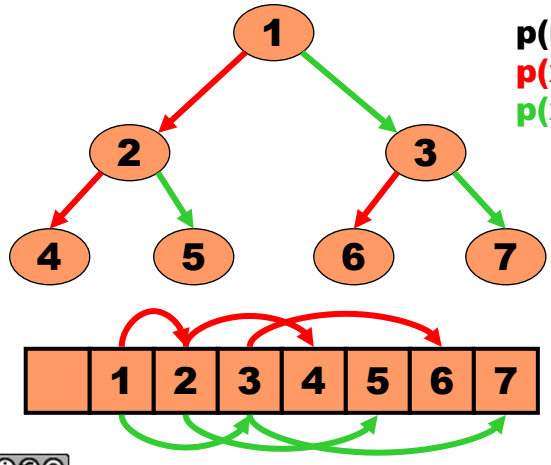


But not complete

Example: complete heap

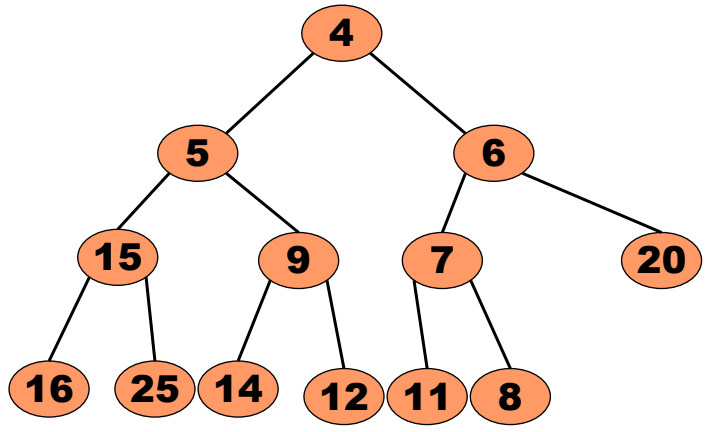


Sequence-based implementation

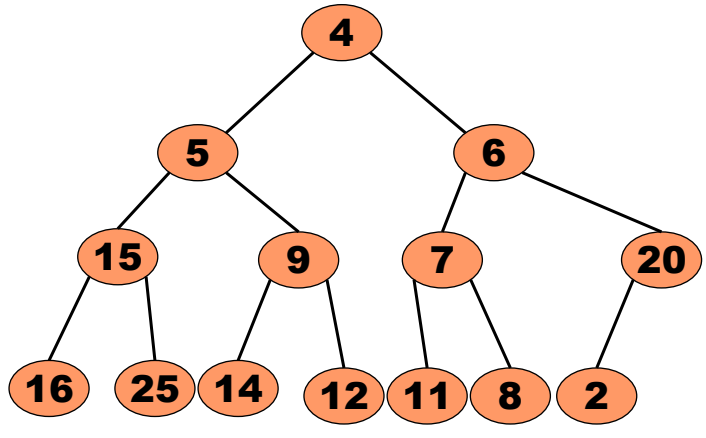


p(root)=1
p(x.left)=2*p(x)
p(x.right)=2*p(x)+1

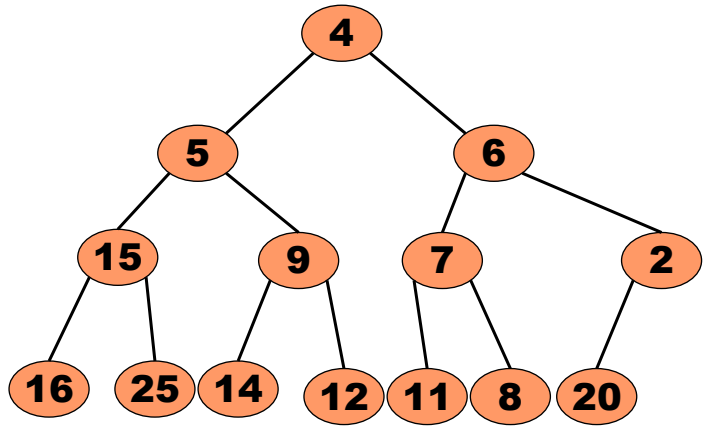
Insert



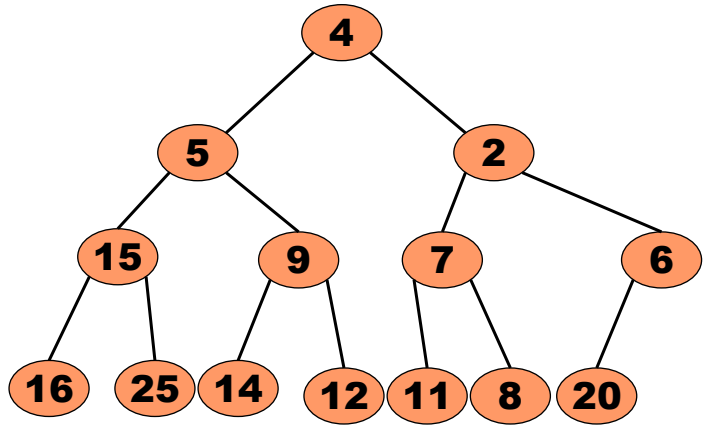
Insert



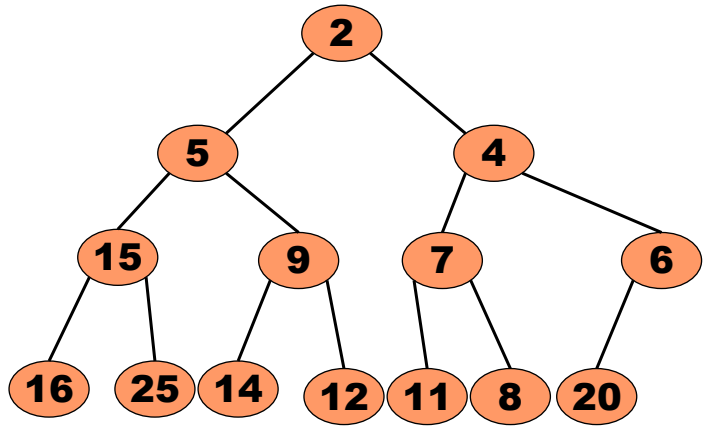
Insert



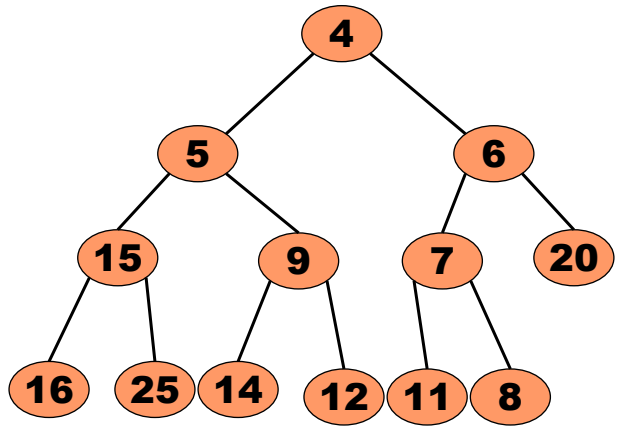
Insert



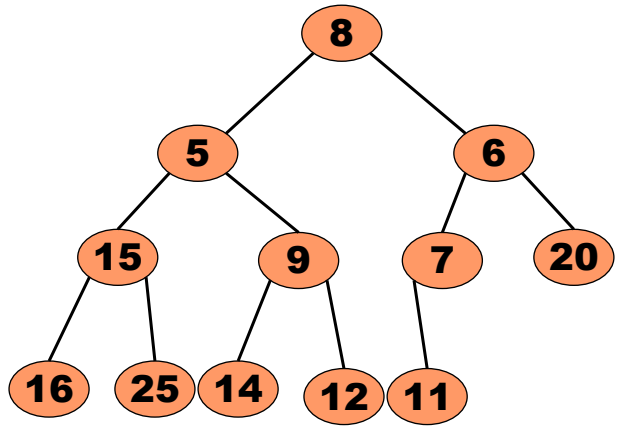
Insert



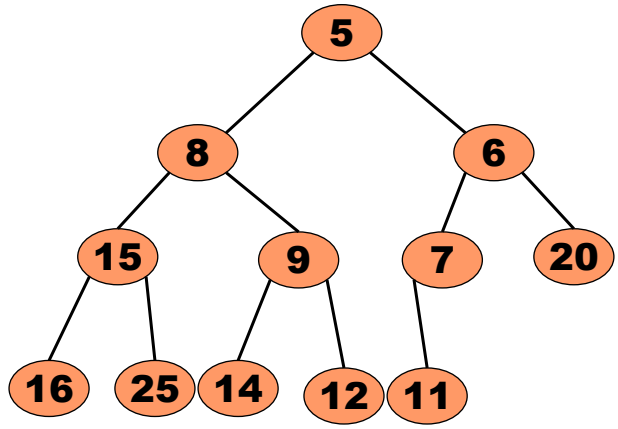
Extract



Extract

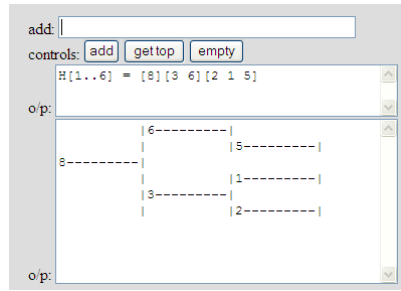


Extract



Activity

- Try out the form in
  http://www.csse.monash.edu.au/~lloyd/tildeAlgDS/Priority-Q/



Activity

- Try out the applet in
  http://www.cosc.canterbury.ac.nz/mukundan/dsal/MinHeapAppl.html