Trees

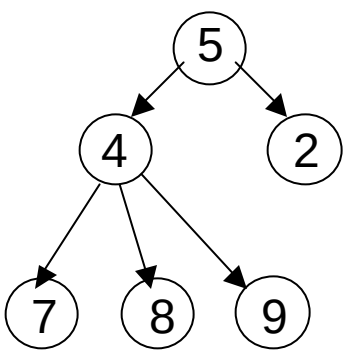
Weiss sec. 7.3.5 (preview)
ch. 18 (sort of)

Data Structures So Far

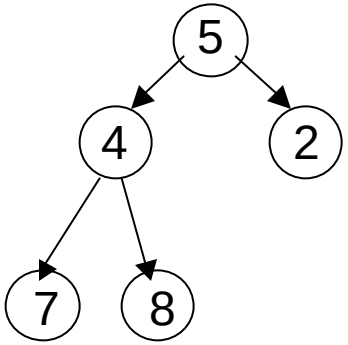
- Arrays
  - random access, fixed size, hard to insert/delete
- java.util.ArrayList
  - random access, dynamic resize, hard to insert/delete
- Linked Lists
  - sequential access, dynamic resize, easy to insert/delete
- Something new:
  - semi-random access, dynamic resize, semi-easy to insert/delete

Trees

- Iterative description:
  - a set of *cells*
  - each cell has 0 or more *successors* (children)
  - one cell is called the *root*
  - each cell has 1 *predecessor* (parent), except the root which has no parent

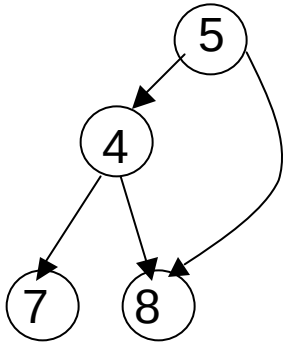


General tree

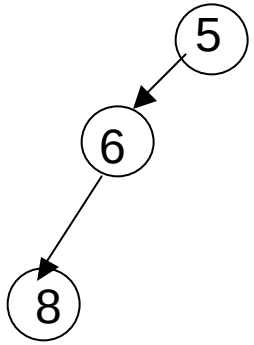


Binary tree

- Recursive description:
  - a tree can be empty
  - a tree can be a cell with 0 or more non-empty trees as children
- Binary tree:
  - a tree where each cell has at most 2 children



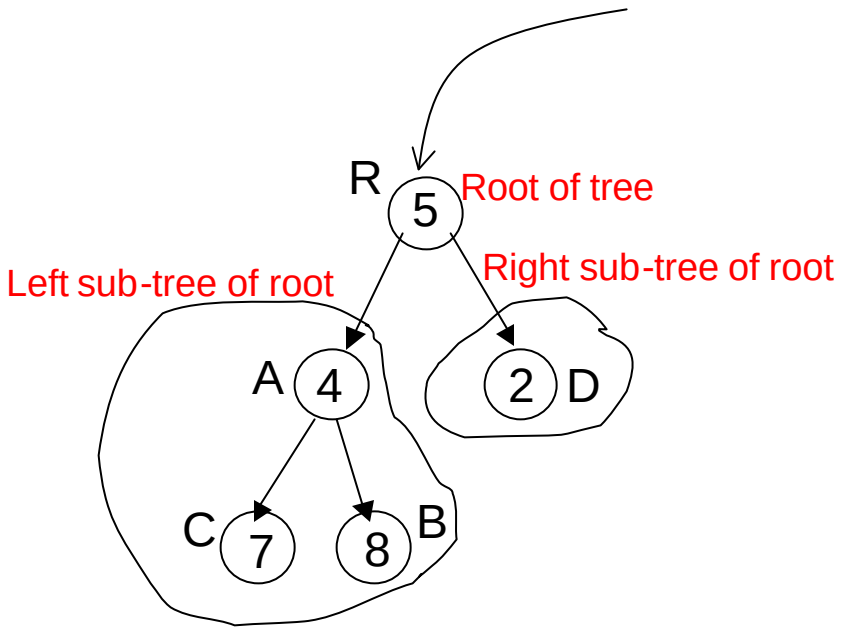
Not a tree



List-like tree

Terminology

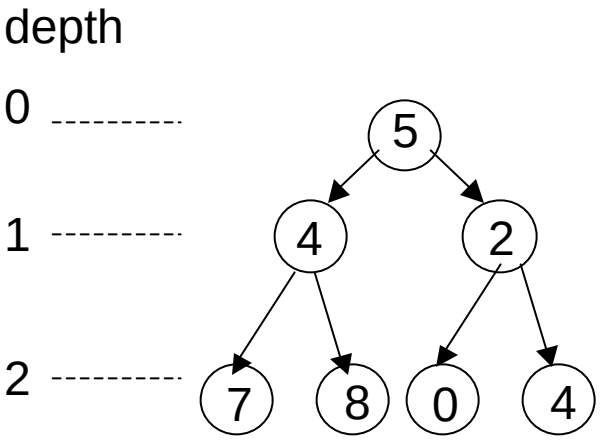
- *edge* A→B
  - A is *parent* of B
  - B is *child* of A
- *path* R → A → B
  - R and A are *ancestors* of B
  - A and B are *descendants* of R
- *leaf node* has no descendants
- *depth of node* is length of path from root to the node
  - depth(A) = 1
  - depth(B) = 2
- *height of node*: length of longest path from node to leaf
  - height(A) = 1
  - height(B) = 0
- *height of tree* = height of root
  - in example, height of tree = 2



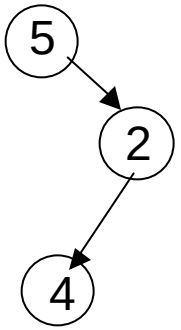
Binary tree

Binary Tree Trivia

- At most 2^d cells at depth d
- In tree of height h:
  - at least h+1 elements
  - at most $2^{h+1} - 1$ elements



Height 2,
maximum number of nodes



Height 2,
minimum number of nodes

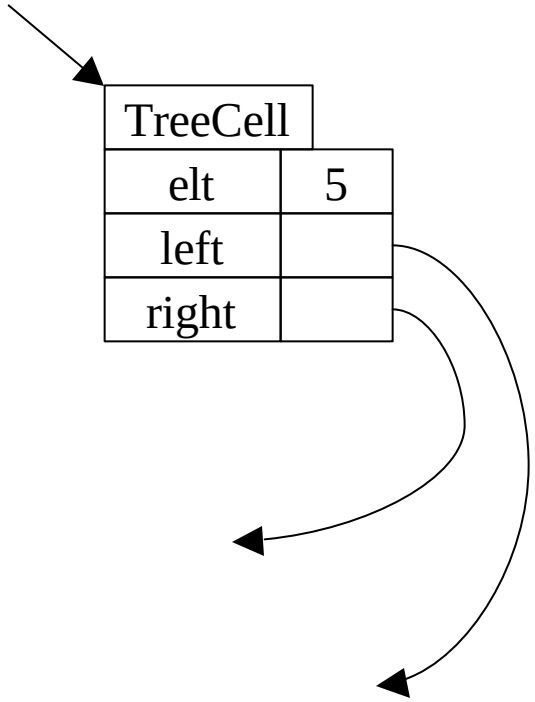
Code

```
public class TreeCell {
  private Object elt;
  private TreeCell left;
  private TreeCell right;

  // Construct a tree with only a single cell
  public TreeCell(Object elt) {
    this(elt, null, null); // defer to three-argument constructor
  }

  // Construct a tree with children
  public TreeCell (Object elt, TreeCell left, TreeCell right) {
    this.elt = elt;
    this.left = left;
    this.right = right;
  }

  /* getters and setters: getElt, getLeft, getRight, setElt, … */
}
```



General Trees: GTreeCell

- One approach: array of children

```
public class GTreeCell {
  protected Object elt;
  protected GTreeCell [ ] children;
  …
}
```

General Trees: GTreeCell

- Second approach: linked list of children

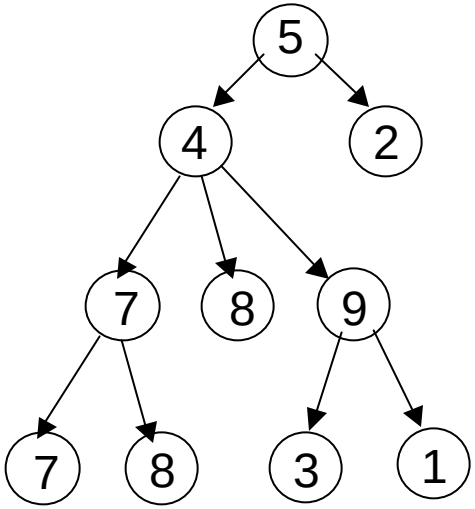
```
public class GTreeCell {
  protected Object elt;
  protected ListCell children;
  …
}
```

General Trees: GTreeCell

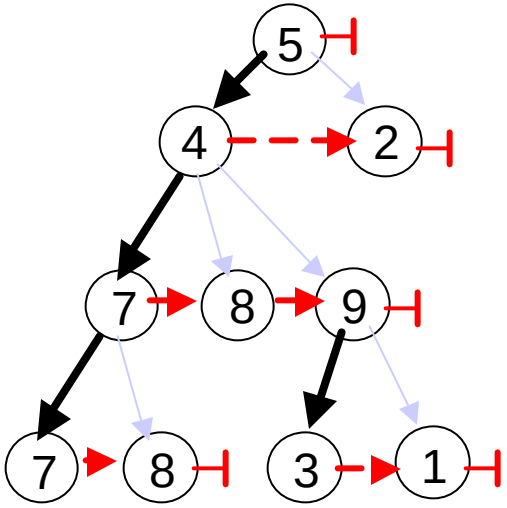
- Third approach: TreeCell doubles as a ListCell
  - Store the *next* pointer from ListCell in the TreeCell object
  - Each cell maintains link to **left child** and **right sibling**

```
public class GTreeCell {
  protected Object elt;
  protected GTreeCell left;
  protected GTreeCell sibling; // essentially ListCell.next
```

Q: How to enumerate children?



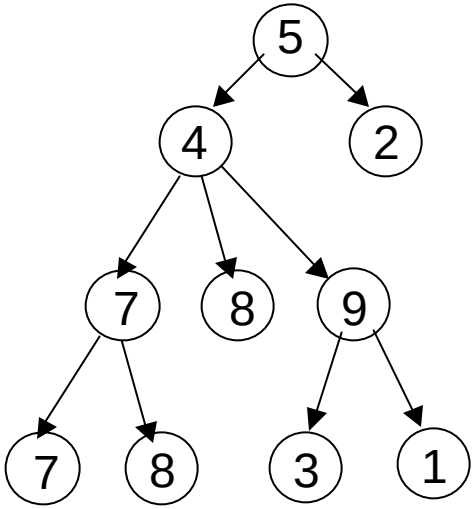
General tree



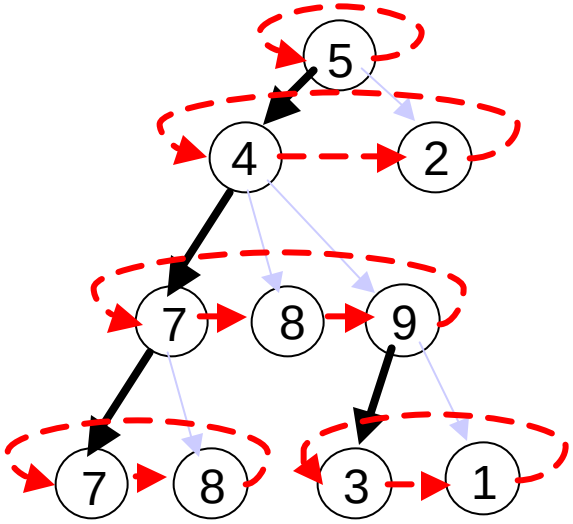
Tree represented using GTreeCell

General Trees: GTreeCell

- Last approach: Circular sibling list
  - Let the sibling list “wrap around”
  - Helps with some operations



General tree



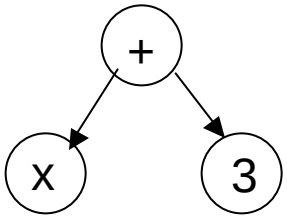
Tree represented using GTreeCell

Applications

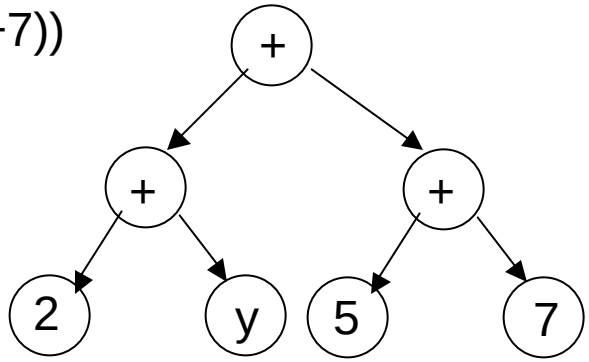
- Trees can represent structure of a language as an *abstract syntax tree* (AST)
- Example grammar:

  E : (E + E)

  E : integer | variable
- What can we do with AST?
  - integration, differentiation, etc.
  - constant-expression elimination
  - re-arrange expressions

Text	Tree representation
-34	
(x + 3)	

((2+y) + (5+7))



Recursion on Trees

- Similar to recursion on lists or integers
- Base case: empty tree (or one-cell tree)
- Recursive case:
  - solve for subtrees
  - combine these solutions to get solution for whole tree

Search: Recursive

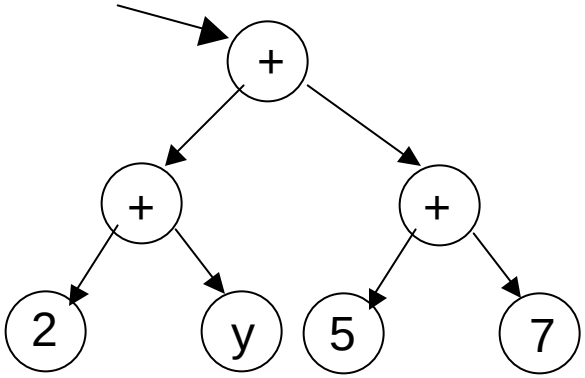
- Find out if object *o* is in the tree:

  static boolean search(TreeCell t, Object o);

- Recursive version is trivial; Try the iterative version at home…

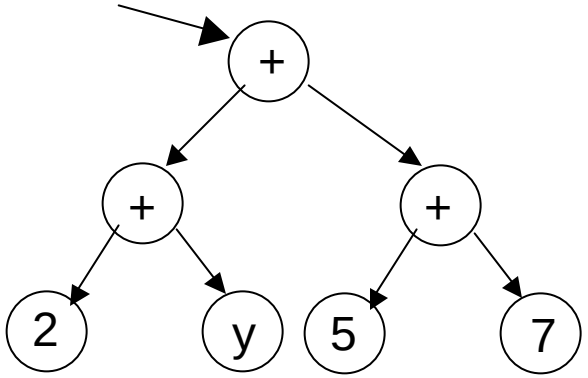
```
public static boolean search(TreeCell t, Object o) {
   if (t == null) return false;
   else return t.getElt().equals(o) | |
               search(t.getLeft(), o) | |
               search(t.getRight(), o);
 }
```

((2+y) + (5+7))



Traversal Ordering

- search( ) traverses tree in following recursive order:
  - first process root node
  - then process left subtree
  - then process right subtree



- *pre-order walk:*
  root, left, right
- *in-order walk:*
  left, root, right
- *post-order walk:*
  left, right, root

Variations

- Write a header class called Tree
  - tree methods can be instance & static methods of Tree
- Maintain reference to parent in each cell
  - analagous to doubly-linked list