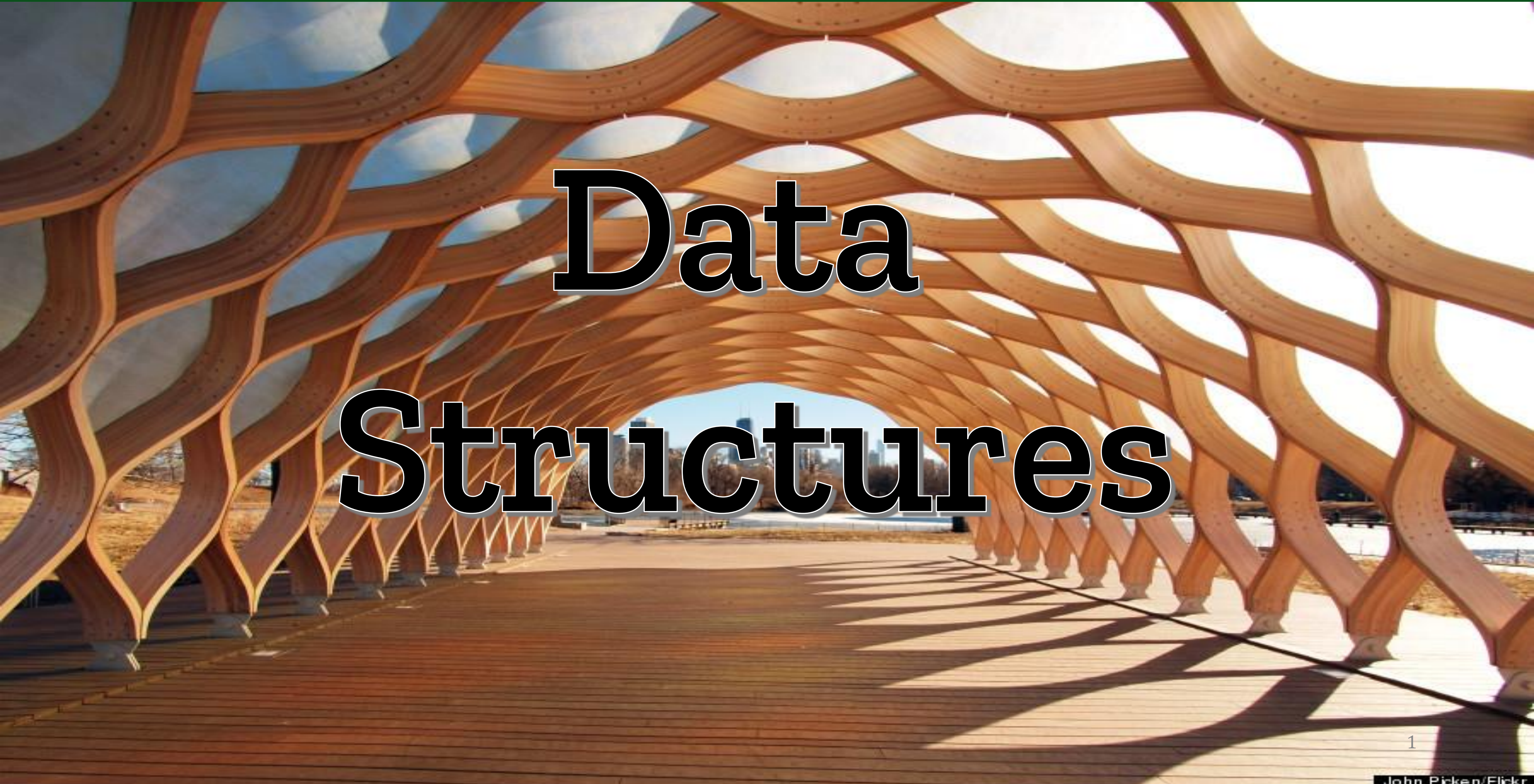
Data Structures



John Picken/Flickr

What is a Data Structure

- A method of organizing data to enable problem solving
- a collection of data values, the relationships among them, and the functions or operations that can be applied to the data.

Wegner, Peter; Reilly, Edwin D. (2003-08-29). Encyclopedia of Computer Science. Chichester, UK: John Wiley and Sons. pp. 507–512. ISBN 978-0470864128.

- Arguably, the key organizing factor in software design

Object / Class as a Data Structure

- Creates a "relationship" between the fields in a single object
  - All fields describe the same object

- Define the ways of accessing and manipulating that data through methods

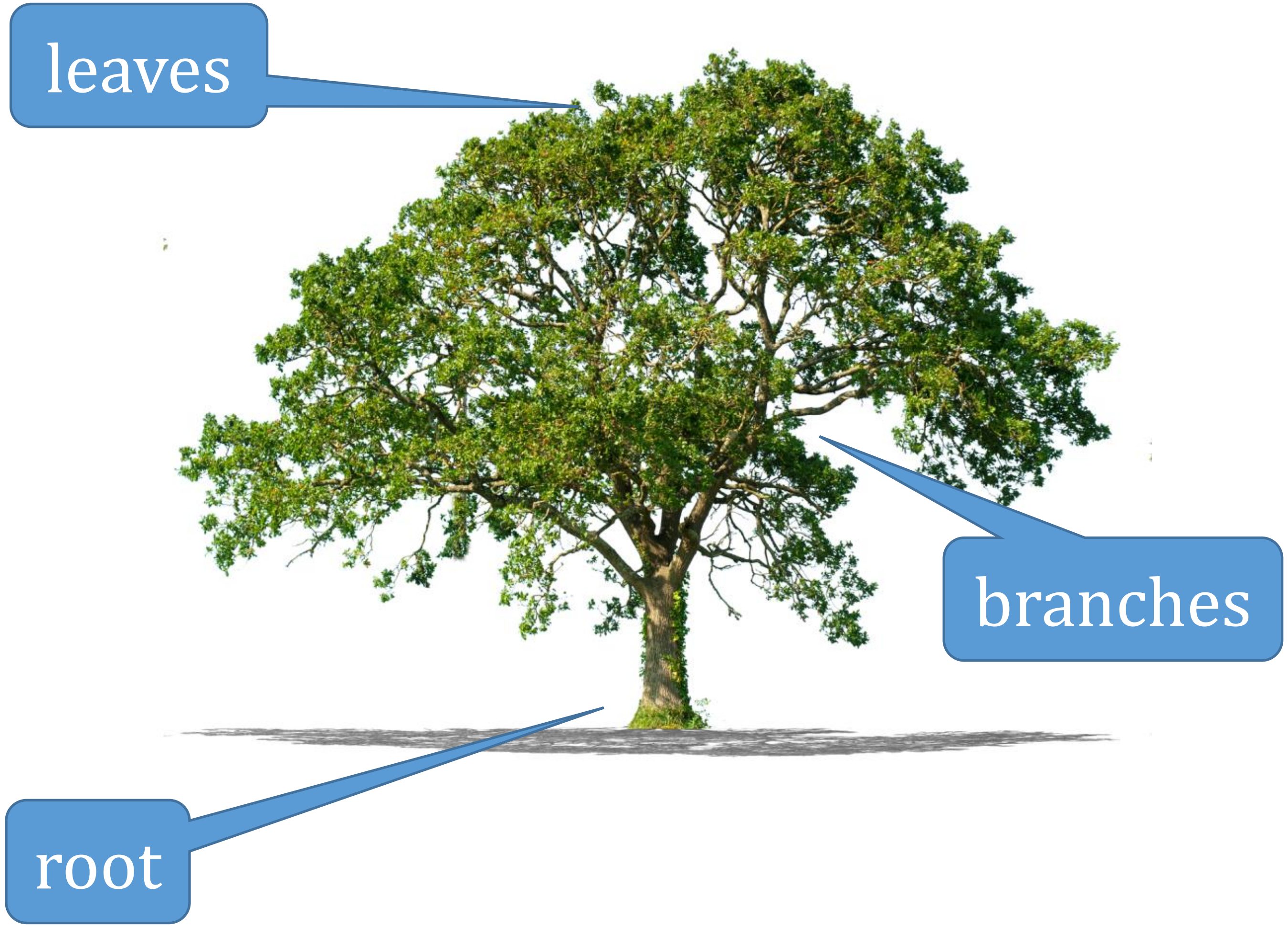
Singly Linked List

- Fast insertion if you know where to insert
- Easy to grow and shrink
- Low "overhead" (but more than arrays)
- Always starts at "head"
- Easy to move forward, hard to move backward!

Doubly Linked List

- Each node contains a "prev" reference to it's predecessor
  - as well as a "next" pointer to what comes after
- Usually track both head and tail, so we can start from either end
- Almost the same as a singly linked list, but more overhead and book-keeping traded off for better performance in some applications.
- See the details in the example code

Tree



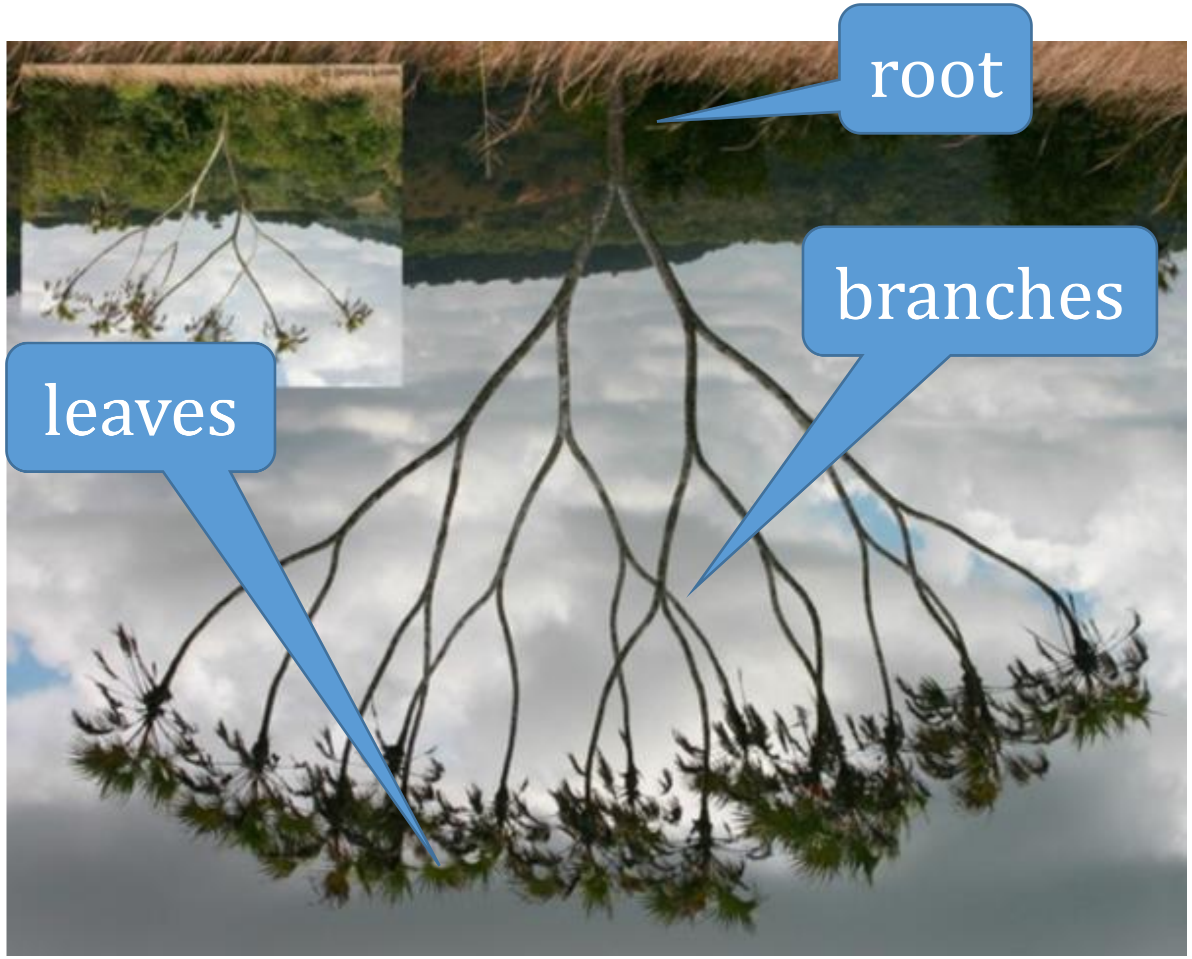
leaves

branches

root

Binary Tree

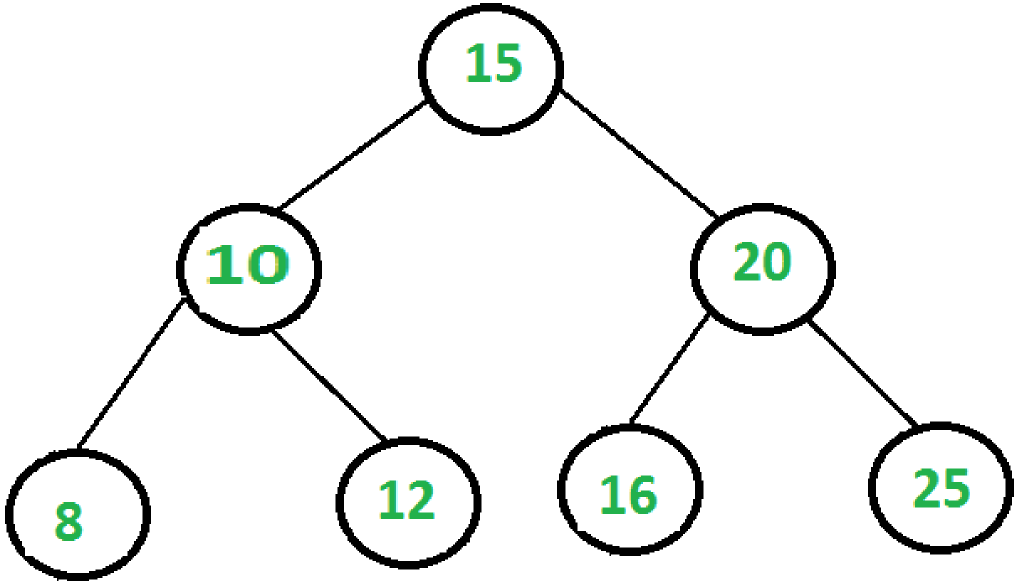
Computer Science Binary Tree



root

branches

leaves



A tree node class….

```
private class TreeNode {
        private int payload;
        private TreeNode left;
        private TreeNode right;

        // Standard constructor, getters and setters
        // and toString
```

Add a node in "order"….

```
public void pushOrder(int payload) {
    if (root == null) root = new TreeNode(payload);
    else pushOrder(payload, root);
}
private void pushOrder(int payload, TreeNode after) {
    if (payload < after.getPayload()) {
        if (after.getLeft() == null) after.setLeft(new TreeNode(payload));
        else pushOrder(payload, after.getLeft());
    } else {
        if (after.getRight() == null) after.setRight(new TreeNode(payload));
        else pushOrder(payload, after.getRight());
    }
}
```

start at the root

payload<after.payload
insert left…

if there is room, add here

if not, add to the left sub-tree

payload>=after.payload
insert right…

Tree size…

```
public int size() {
    return size(root);
}


private int size(TreeNode from) {
    if (from==null) return 0;
    return 1+size(from.left) +size(from.right);
}
```

start at the root

no nodes in a null reference

this node

number of nodes in left sub-tree

number of nodes in right sub-tree

Tree depth…

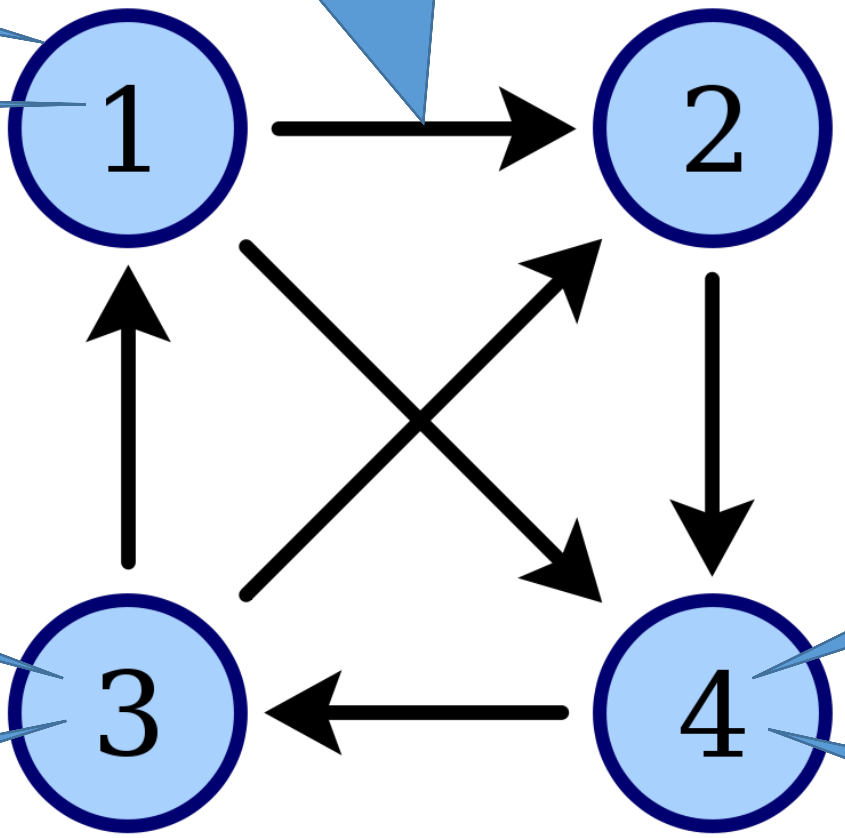
```
public int depth() {
    return depth(root);
}

private int depth(TreeNode from) {
    if (from==null) return 0;
    int dl=depth(from.left);
    int dr=depth(from.right);
    return 1+(dl>dr?dl:dr);
}
```

Depth: The maximum distance from the root to any leaf

this node

maximum of either left or right sub-tree depth

Directed Graph



reference to another node

node

payload

references 2 nodes

referenced by 1 nodes

references 1 node

referenced by 2 nodes

By Booyabazooka - Own work,
Public Domain,
https://commons.wikimedia.org/w/index.php?curid=5275390

Data Structures

- Much more to cover, but that's an intro
  - CS-240 Data Structures and Algorithms
- Java self-references enable easy implementation
- Great examples of method recursion!