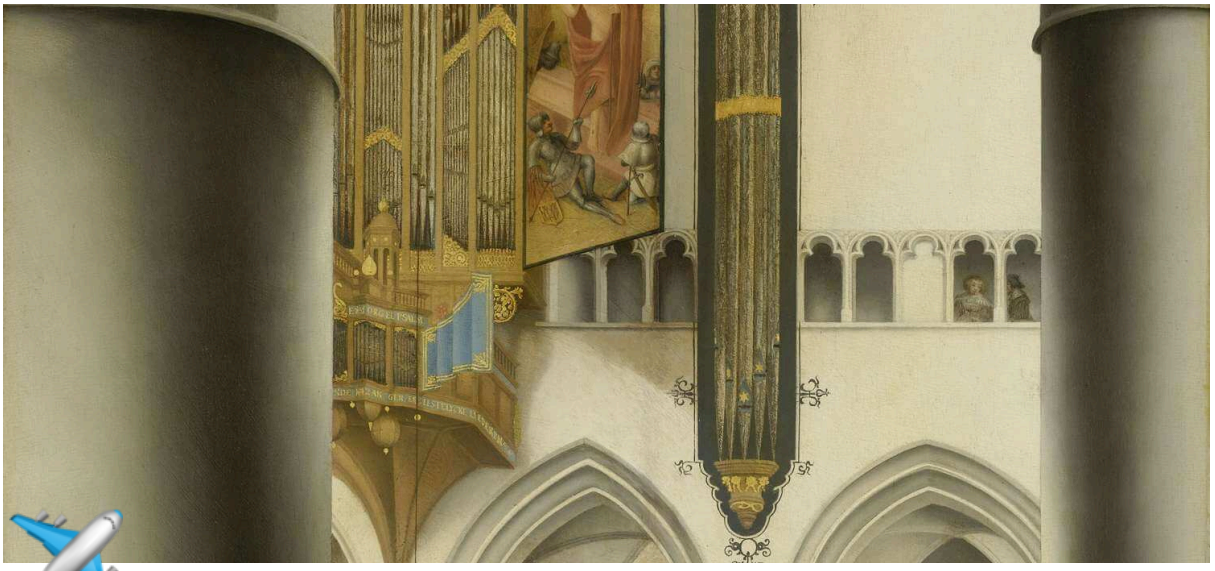
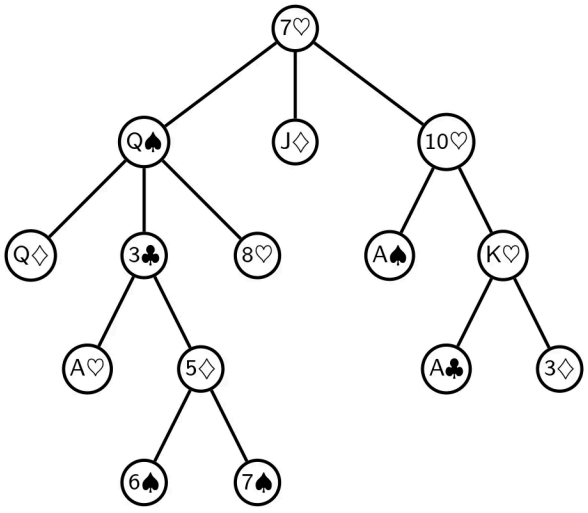


Trees



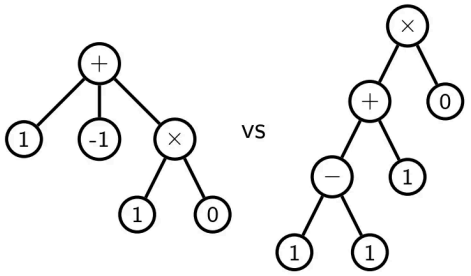
In computer science, a tree is a very general and powerful (data) structure. It consists of a set of linked nodes in a connected *graph*, in which each node has at most one parent node, and zero or more children nodes with a specific order.

Examples of trees

- We have seen some trees while sorting (slides 10, 14, 22, etc.)
- The formula

$$1 - 1 + 1 \times 0$$

- UoB Sustainability Webpage

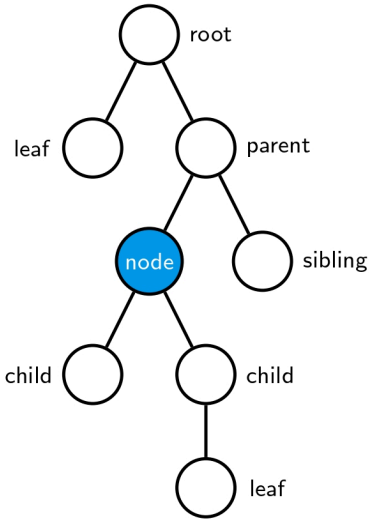


🔥 ***The left tree I'd interpret as*** $(1 - 1) + (1 * 0)$

The right tree I'd interpret as $((1 - 1) + 1) * 0$

🔥 ***The original formula gave no indication of order of priority. We as humans by default use BIDMAS to answer these questions with set universally agreed rules. But a computer doesn't know this. And trees provide a representation with a specific order of priority depending on how the tree is constructed***

Tree Terminology



root the unique node at the base

parent a node immediately *above*; each node (with the exception of the root) has exactly one parent.

child a node immediately *below*.

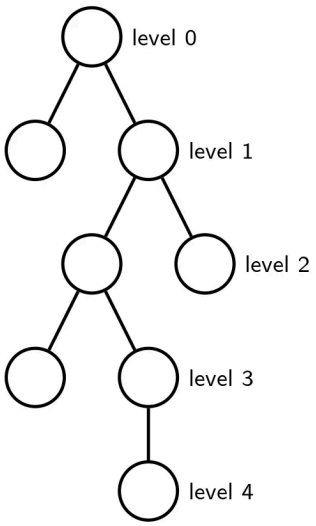
leaf a node with no children. The tree on the left has 4 leafs.

sibling a node (on the same level) that shares the same parent.

ancestor parent or any node on the unique path to the root.

descendent any node below, i.e., children, children of childrens, ...

3



level of a node (or depth) is the length of the path from the node to the *root* (*root* has *level* 0).

height of a tree is the length of the longest path from the *root* to a *leaf*. Equivalently, the maximal depth of a node.

size of a tree is the number of nodes in the tree.

A tree with one node has *size* 1 and *height* 0.
The empty tree (with no nodes) has *size* 0 and, by convention, a *height* of -1.
The tree on the left has *size* 8 and *height* 5.

More general trees

Depending on the number of child nodes that each node has, we can have:

- Unary trees (0 or 1 children) = Linked Lists,
- Binary trees (up to 2 children),
- Ternary trees (up to 3 children),
- Quad trees (up to 4 children), ...

You might also have a tree with unspecified maximal number of children (e.g., storing the pointers to children in an array).

```
class Node {
  int val;
  Node[] children;
}
```

8

Quad Trees:

Quad Trees

A *Quad Tree* is a particular kind of tree that differs from the binary tree that we have looked at so far in two respects:

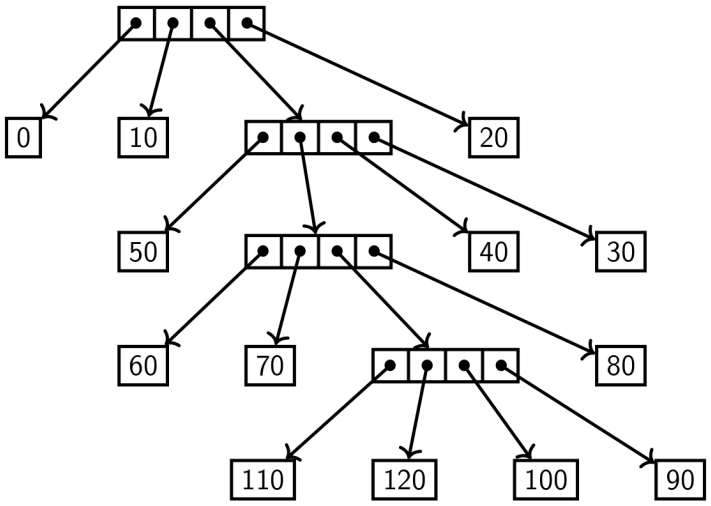
- Each internal (non-leaf) node has **exactly 4 children**.
- Often, only leafs store values, i.e., internal nodes have no values.

This data structure is particularly useful in representing and manipulating 2-dimensional data such as images. A typical application is in image compression, or maps.

9

It says *often* not always, only the leafs store values. This means the leafs can still store values

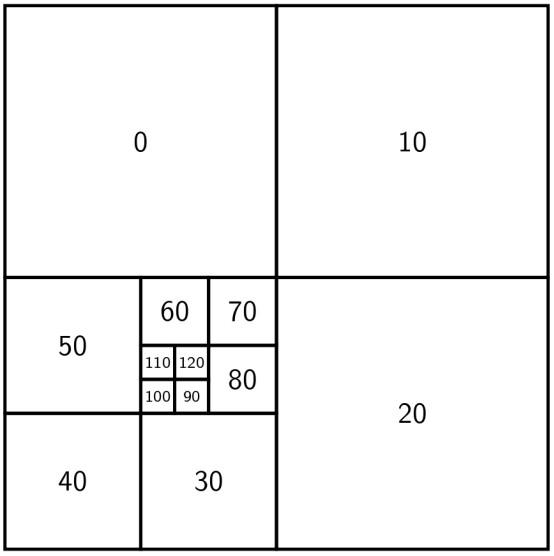
Quad Trees



10

This one only allows 0 or ***4 children***

Quad Trees



11

Exercise

Given a quad tree with the following structure, rotate it counter-clockwise by 90 degrees!

```
class Node {
  int value;
  Node upper_right;
  Node upper_left;
  Node lower_left;
  Node lower_right;
}
```

12

Other ways to represent trees:

Tree Implementation Options

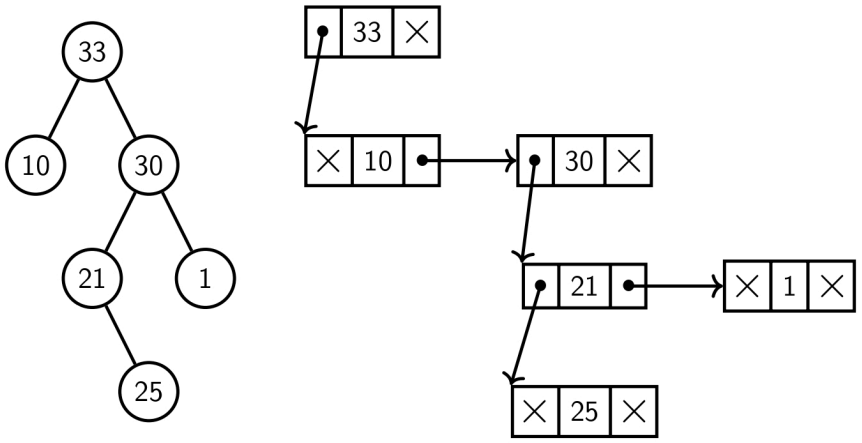
There are 3 common approaches to implementing trees:

Basic: Use nodes like doubly linked list nodes with a value field, and left and right child pointers

Sibling List: Use nodes with a value field, a single *children* pointer, and a pointer to the next sibling. This is good for trees with a variable number of children in each node.

Array: For binary trees, use arrays with a layout based on storing the root at index 1, then the children of the node at index i is stored at index $2 * i$ and $2 * i + 1$.

Tree Implementation Options: Sibling List



14

Structure is :

Left:

Points to child

Middle:

Is current value

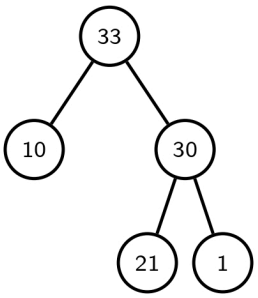
Right:

Points to sibling [if any] in that direction

Tree Implementation Options: Array

Store a binary tree in an array `tree` as follows:

- `tree[0]` is empty,
- `tree[1]` is the root,
- The children of the node `tree[i]` are: `tree[2*i]` and `tree[2*i+1]`.



$$[\times, 33, 10, 30, \times, \times, 21, 1]$$

16

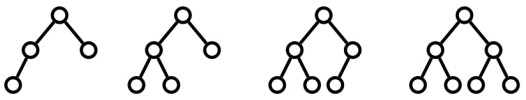
Complete Binary Tree

Complete Binary Tree is a binary tree that can be stored in an array without vasting memory.

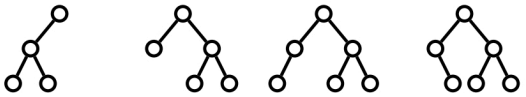
Definition. A binary tree is complete if every level, except possibly the last, is completely filled, and all the leaves on the last level are placed as far to the left as possible.

This simply defines, for a binary tree with a given number of nodes, the *tidiest, most balanced*, and *compact* form possible

Complete:



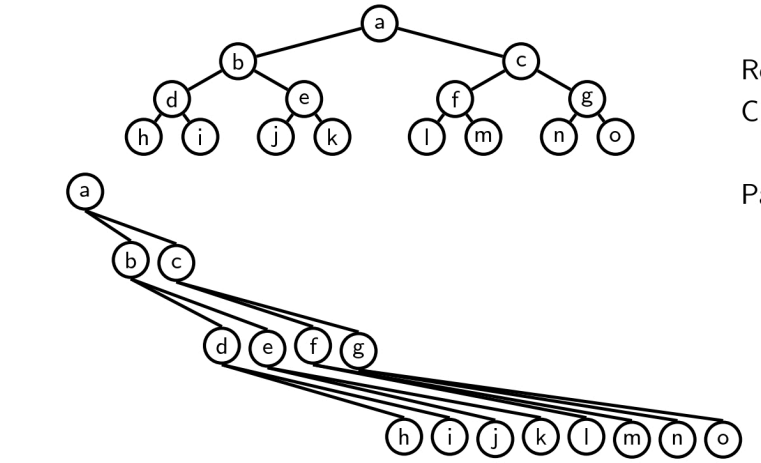
Not Complete:



17

Complete Binary Tree in an Array

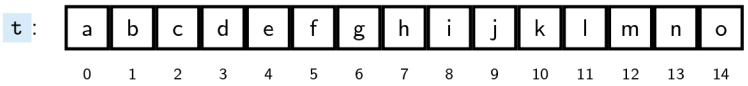
We can store Binary Trees in a simple array structure.



Root: `t[0]`

Children of `i`: `t[2*i+1]` `t[2*i+2]`

Parent of `i`: `t[i/2]`



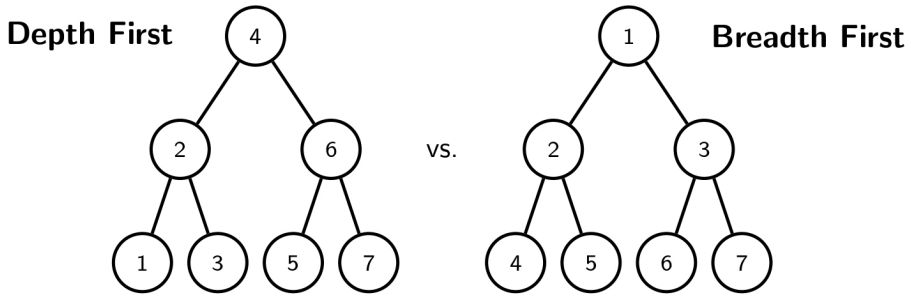
`t`:

18

The formula for the children slightly changed from $t[2 * i], t[(2 * i) + 1]$ ***to*** $t[(2 * i) + 1], t[(2 * i) + 2]$ ***because the root node is*** $t[0]$ ***instead of*** $t[1]$ ***because the its a complete binary tree.***

Search in a tree:

DFS vs. BFS Search



$[1, 2, 3, 4, 5, 6, 7]$ $[4, 2, 5, 1, 6, 3, 7]$

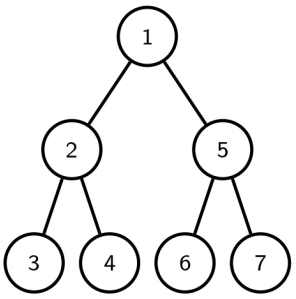
```java
int[] flatten(Node tree) {
  if (tree == null) return new int[] {};
  return flatten(tree.left) + {tree.val} + flatten(tree.right);
}
```

19

Recursively calls the flatten till a leaf is reached then it outputs the leafs value and traces through the tree in order traversal

DFS = Depth First Search

Traversing the tree using *recursion* results in following a branch until its leaf, backtracking if the value is not found.



This is called **depth first search (DFS)**.

```java
bool bfs(Node tree, int value) {
  if (tree == null) {
    return false;
  } else {
    return (tree.val == value
         || bfs(tree.left, value)
         || bfs(tree.right, value))
  }
}
```

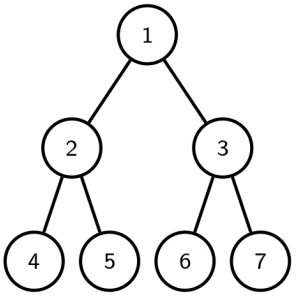
Time complexity: $O(n)$,
Space complexity: $O(\text{depth})$.

20

$[3, 2, 4, 1, 6, 5, 7]$

BFS = Breadth First Search

Traversing the tree by levels, i.e., all children will be visited before any node on level 2.



This is called **breadth first search (BFS)**.

```
1  bool dfs(Node tree, int value) {
2    if (tree == null)
3      return false;
4
5    Queue q = new EmptyQueue;
6    q.enqueue(tree);
7    while (!q.isEmpty()) {
8      node = q.dequeue();
9      if (node.val == value)
10       return true;
11     q.enqueue(node.left);
12     q.enqueue(node.right);
13   }
14 }
```

Time complexity: $O(n)$,
Space complexity: $O(n)$.

21

$[1, 2, 3, 4, 5, 6, 7]$