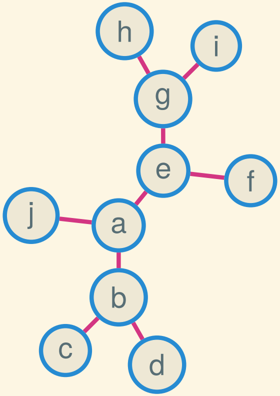
Trees and Tree Walks

EECS 214, Fall 2017

Let’s talk trees

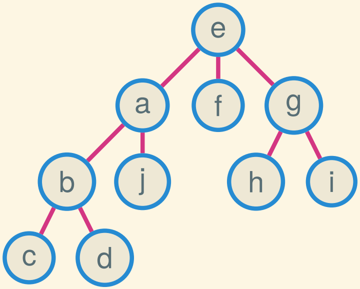
Definition

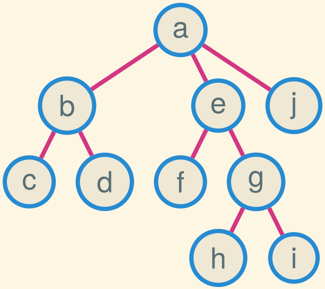
A tree is a graph with no cycles:



Rooted trees

We can *root* a tree by choosing one vertex to be the root:

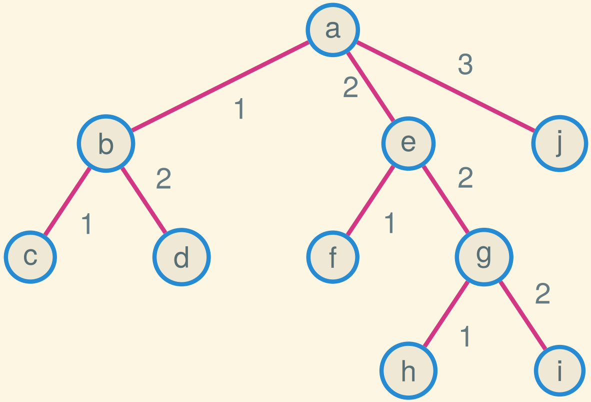




This lets us talk about *children* and *subtrees*

Rooted, ordered trees

An *ordered* tree assigns an order to the children of each node:

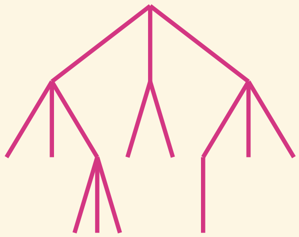


Now we can refer to the 1st child, 2nd child, etc.

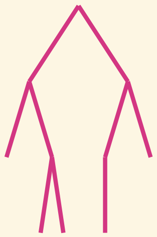
k-ary trees

In a *k*-ary tree, each node has at most *k* children:

a 3-ary tree

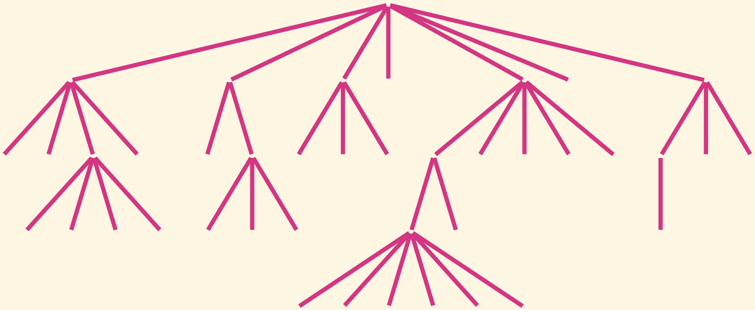


a 2-ary tree



Rose tree

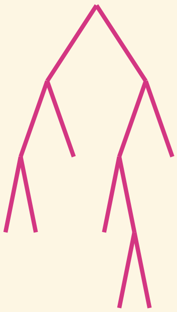
A rose tree is an ∞-ary tree:



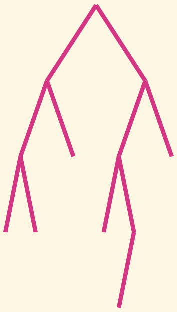
Full trees

A *k*-ary tree is full if every non-leaf node has *k* children:

full binary tree



not full

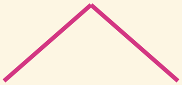


Complete trees

A tree is complete if every level is full of nodes except the last, which must be filled from the left:

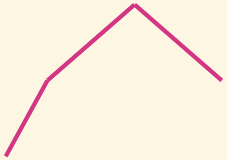
Complete trees

A tree is complete if every level is full of nodes except the last, which must be filled from the left:



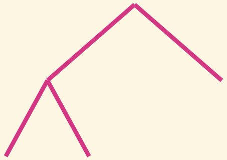
Complete trees

A tree is complete if every level is full of nodes except the last, which must be filled from the left:



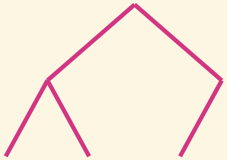
Complete trees

A tree is complete if every level is full of nodes except the last, which must be filled from the left:



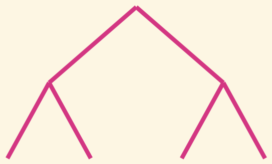
Complete trees

A tree is complete if every level is full of nodes except the last, which must be filled from the left:



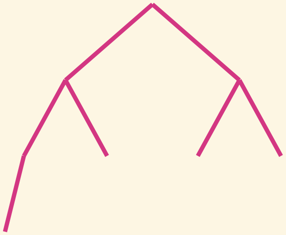
Complete trees

A tree is complete if every level is full of nodes except the last, which must be filled from the left:



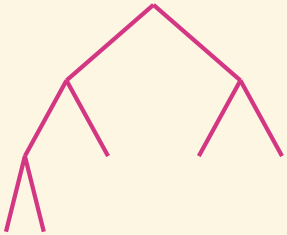
Complete trees

A tree is complete if every level is full of nodes except the last, which must be filled from the left:



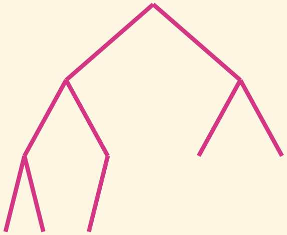
Complete trees

A tree is complete if every level is full of nodes except the last, which must be filled from the left:



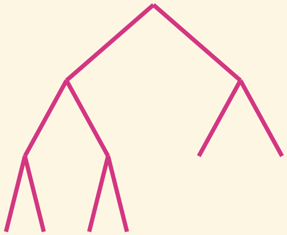
Complete trees

A tree is complete if every level is full of nodes except the last, which must be filled from the left:



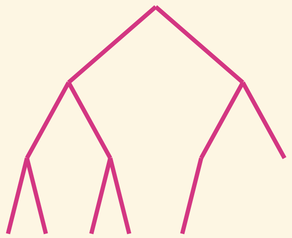
Complete trees

A tree is complete if every level is full of nodes except the last, which must be filled from the left:



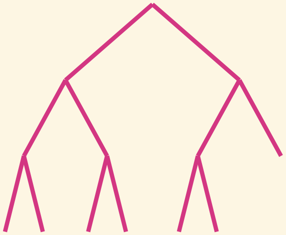
Complete trees

A tree is complete if every level is full of nodes except the last, which must be filled from the left:



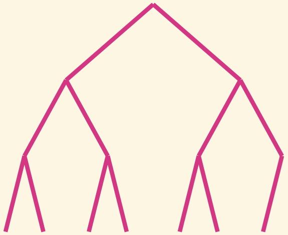
Complete trees

A tree is complete if every level is full of nodes except the last, which must be filled from the left:



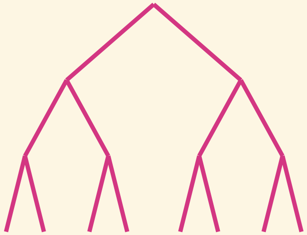
Complete trees

A tree is complete if every level is full of nodes except the last, which must be filled from the left:



Complete trees

A tree is complete if every level is full of nodes except the last, which must be filled from the left:

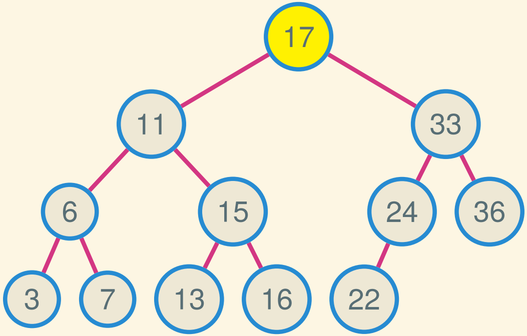


What’s a tree walk?

A tree walk traverses a tree and linearizes the vertices in some order

Pre-order walk

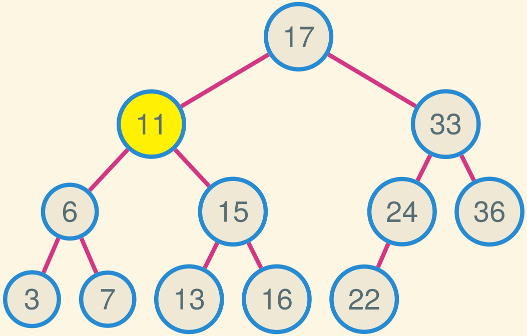
Visit each node *before* its children:



Pre-order: 17

Pre-order walk

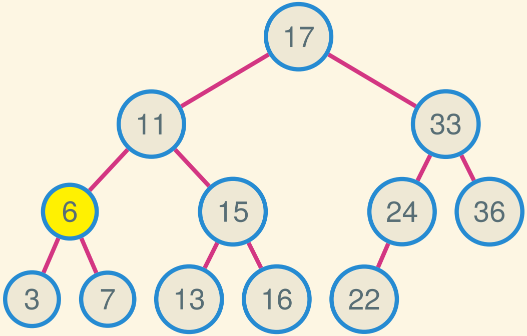
Visit each node *before* its children:



Pre-order: 17, 11

Pre-order walk

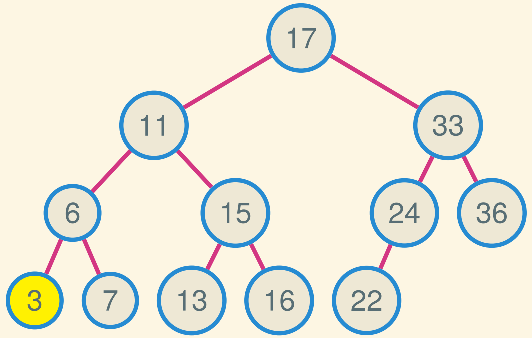
Visit each node *before* its children:



Pre-order: 17, 11, 6

Pre-order walk

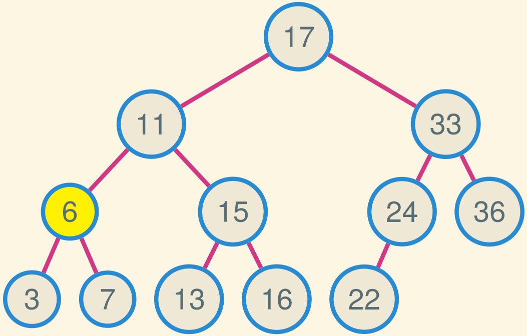
Visit each node *before* its children:



Pre-order: 17, 11, 6, 3

Pre-order walk

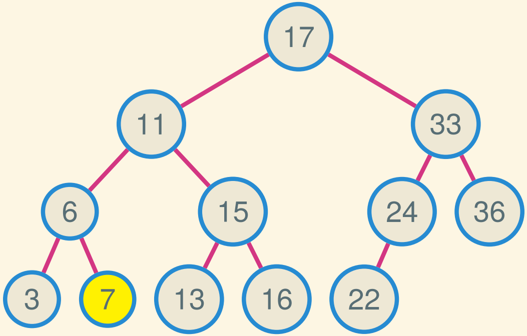
Visit each node *before* its children:



Pre-order: 17, 11, 6, 3

Pre-order walk

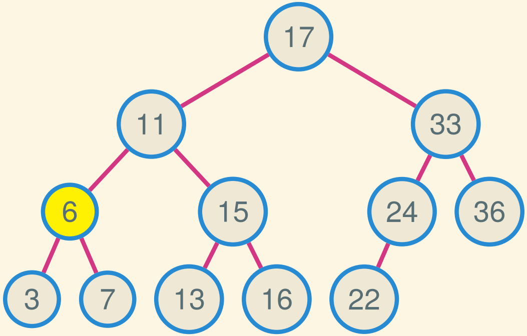
Visit each node *before* its children:



Pre-order: 17, 11, 6, 3, 7

Pre-order walk

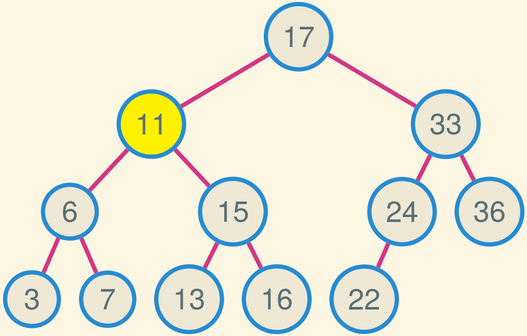
Visit each node *before* its children:



Pre-order: 17, 11, 6, 3, 7

Pre-order walk

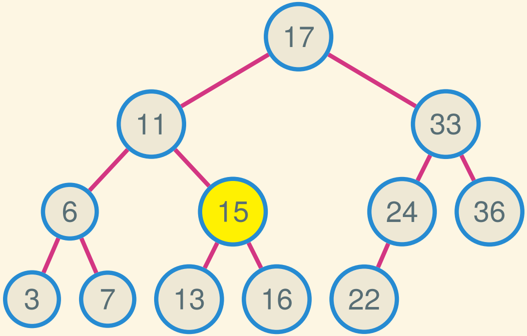
Visit each node *before* its children:



Pre-order: 17, 11, 6, 3, 7

Pre-order walk

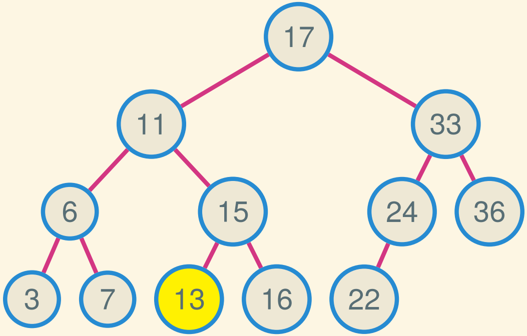
Visit each node *before* its children:



Pre-order: 17, 11, 6, 3, 7, 15

Pre-order walk

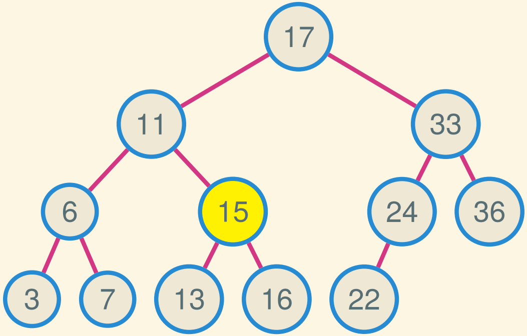
Visit each node *before* its children:



Pre-order: 17, 11, 6, 3, 7, 15, 13

Pre-order walk

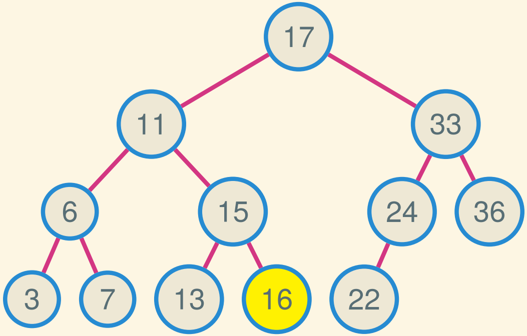
Visit each node *before* its children:



Pre-order: 17, 11, 6, 3, 7, 15, 13

Pre-order walk

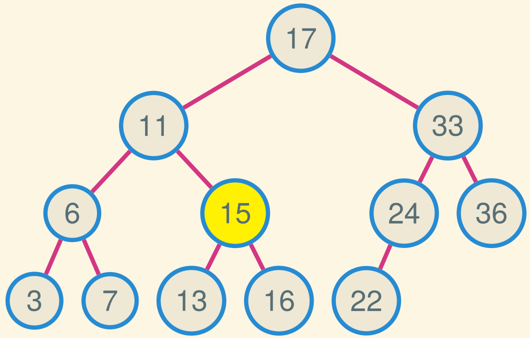
Visit each node *before* its children:



Pre-order: 17, 11, 6, 3, 7, 15, 13, 16

Pre-order walk

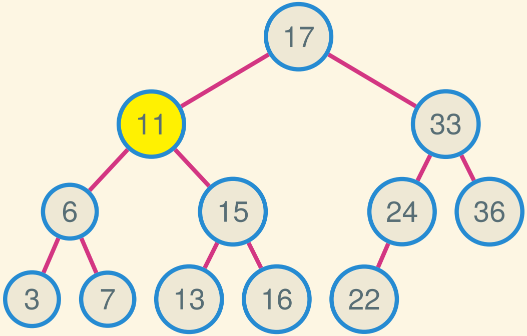
Visit each node *before* its children:



Pre-order: 17, 11, 6, 3, 7, 15, 13, 16

Pre-order walk

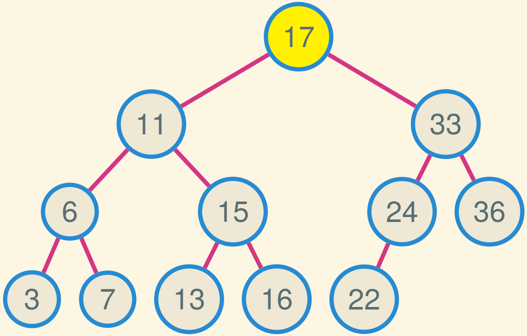
Visit each node *before* its children:



Pre-order: 17, 11, 6, 3, 7, 15, 13, 16

Pre-order walk

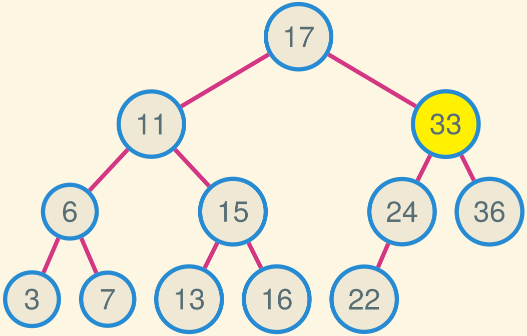
Visit each node *before* its children:



Pre-order: 17, 11, 6, 3, 7, 15, 13, 16

Pre-order walk

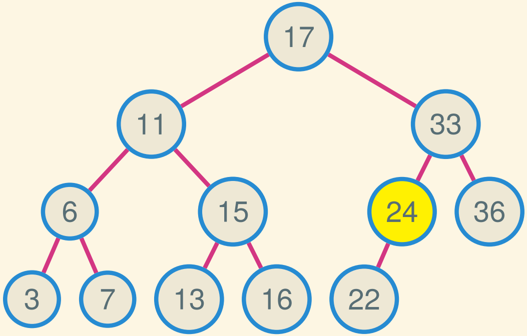
Visit each node *before* its children:



Pre-order: 17, 11, 6, 3, 7, 15, 13, 16, 33

Pre-order walk

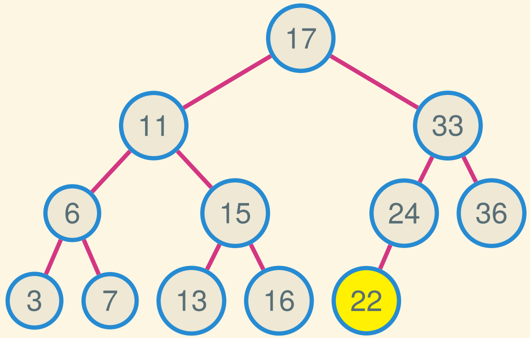
Visit each node *before* its children:



Pre-order: 17, 11, 6, 3, 7, 15, 13, 16, 33, 24

Pre-order walk

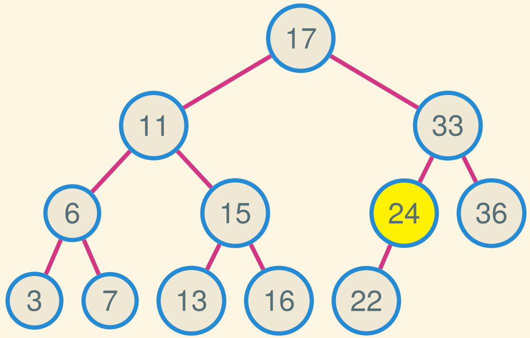
Visit each node *before* its children:



Pre-order: 17, 11, 6, 3, 7, 15, 13, 16, 33, 24, 22

Pre-order walk

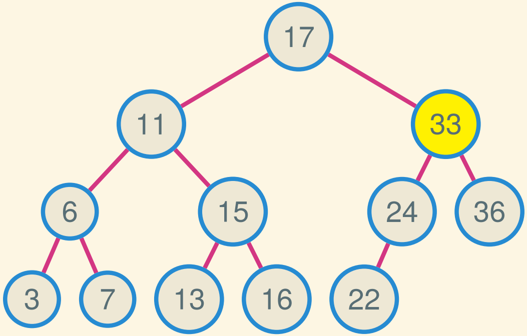
Visit each node *before* its children:



Pre-order: 17, 11, 6, 3, 7, 15, 13, 16, 33, 24, 22

Pre-order walk

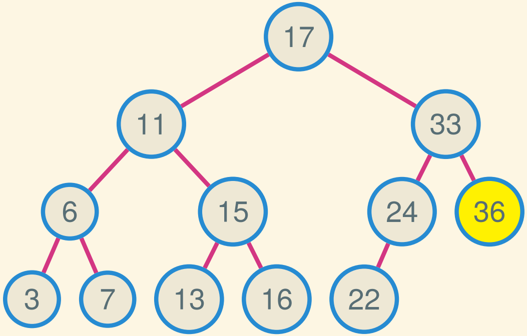
Visit each node *before* its children:



Pre-order: 17, 11, 6, 3, 7, 15, 13, 16, 33, 24, 22

Pre-order walk

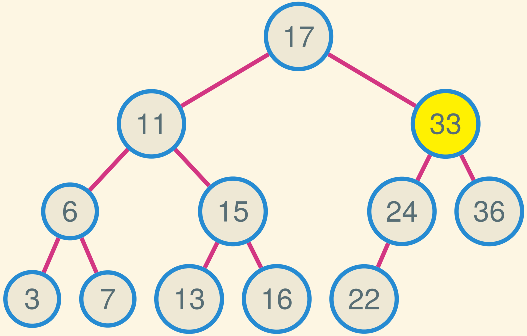
Visit each node *before* its children:



Pre-order: 17, 11, 6, 3, 7, 15, 13, 16, 33, 24, 22, 36

Pre-order walk

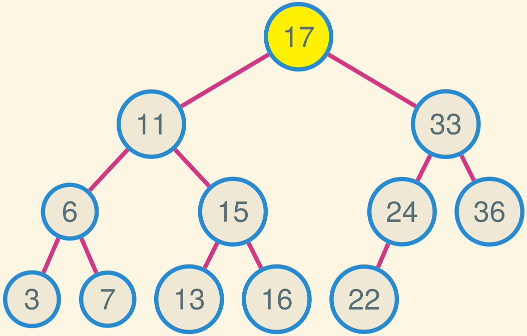
Visit each node *before* its children:



Pre-order: 17, 11, 6, 3, 7, 15, 13, 16, 33, 24, 22, 36

Pre-order walk

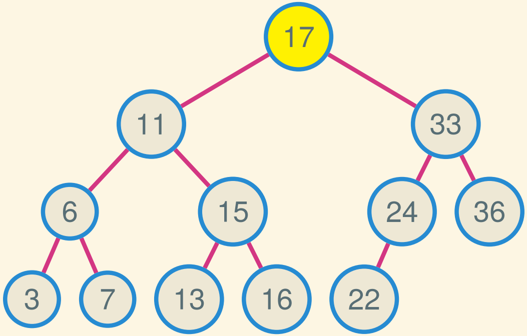
Visit each node *before* its children:



Pre-order: 17, 11, 6, 3, 7, 15, 13, 16, 33, 24, 22, 36

Post-order walk

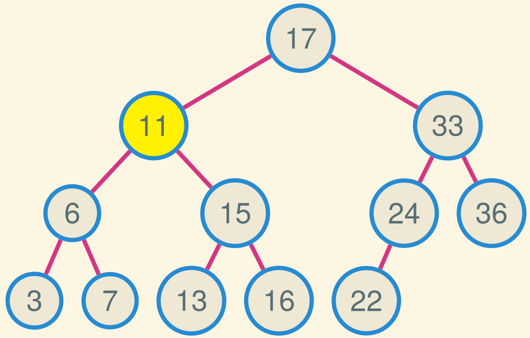
Visit each node *after* its children:



Post-order:

Post-order walk

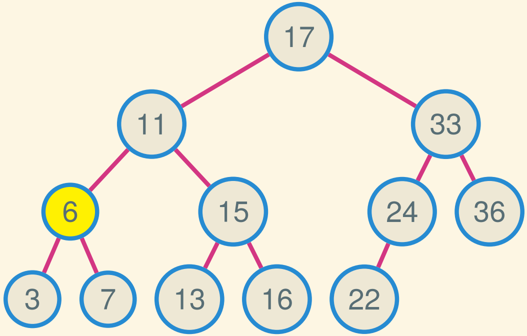
Visit each node *after* its children:



Post-order:

Post-order walk

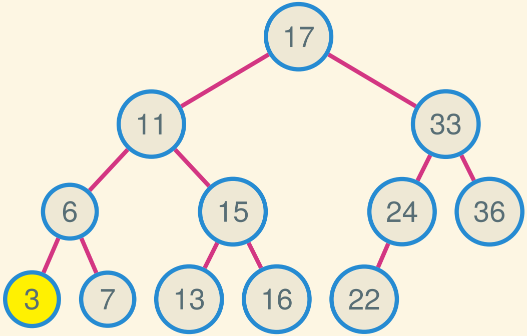
Visit each node *after* its children:



Post-order:

Post-order walk

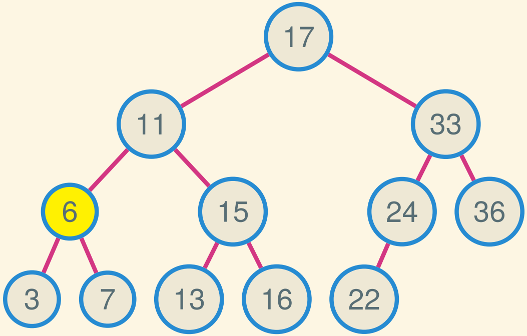
Visit each node *after* its children:



Post-order: 3

Post-order walk

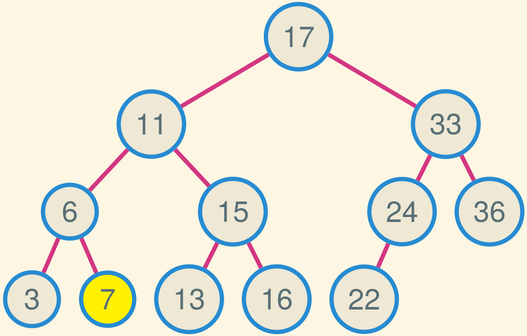
Visit each node *after* its children:



Post-order: 3

Post-order walk

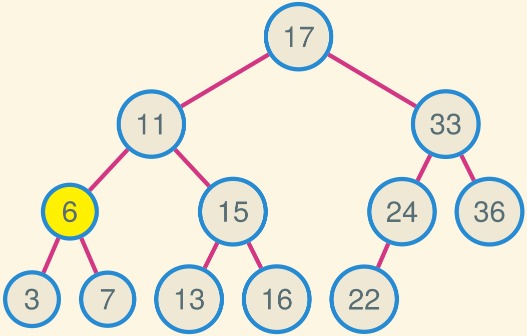
Visit each node *after* its children:



Post-order: 3, 7

Post-order walk

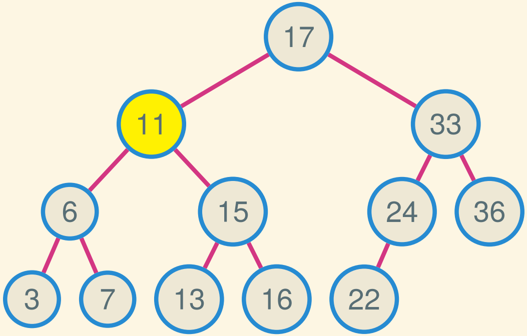
Visit each node *after* its children:



Post-order: 3, 7, 6

Post-order walk

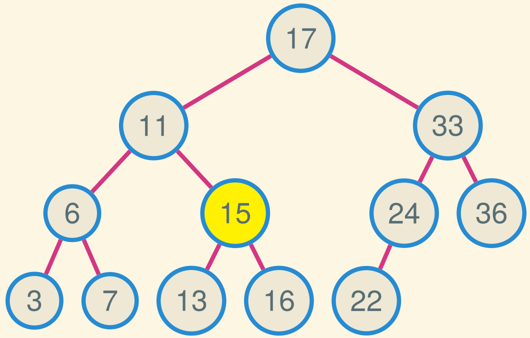
Visit each node *after* its children:



Post-order: 3, 7, 6

Post-order walk

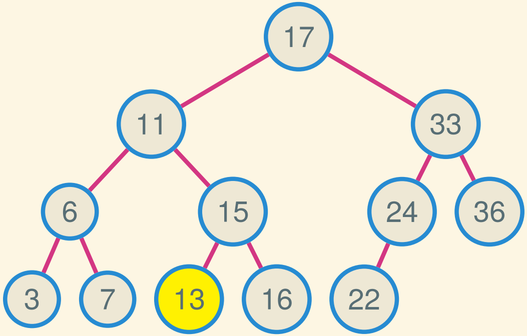
Visit each node *after* its children:



Post-order: 3, 7, 6

Post-order walk

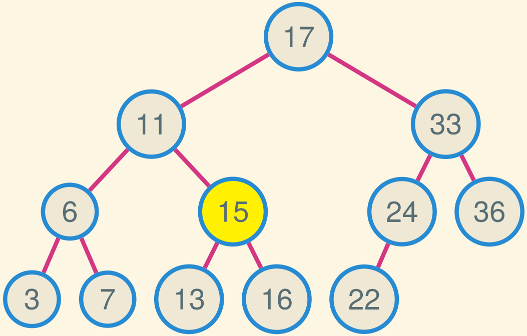
Visit each node *after* its children:



Post-order: 3, 7, 6, 13

Post-order walk

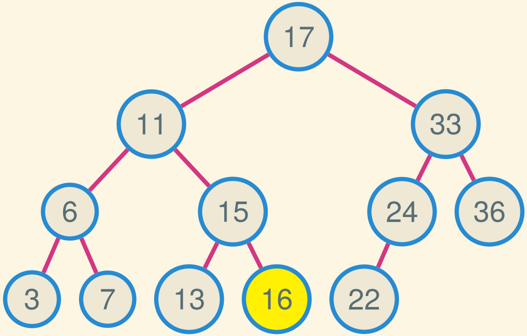
Visit each node *after* its children:



Post-order: 3, 7, 6, 13

Post-order walk

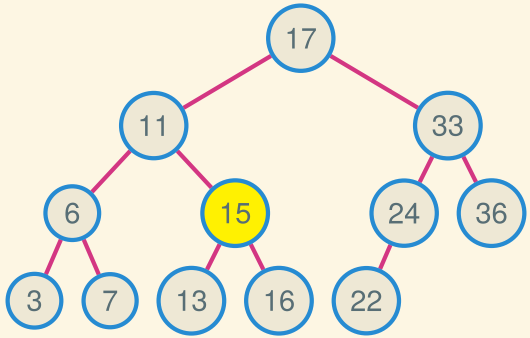
Visit each node *after* its children:



Post-order: 3, 7, 6, 13, 16

Post-order walk

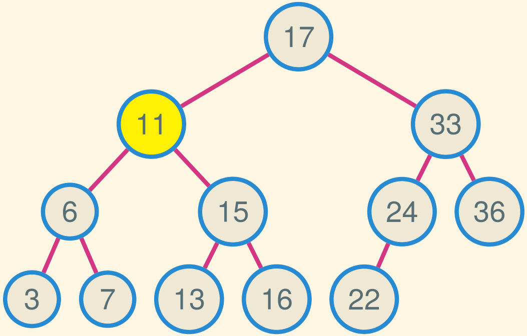
Visit each node *after* its children:



Post-order: 3, 7, 6, 13, 16, 15

Post-order walk

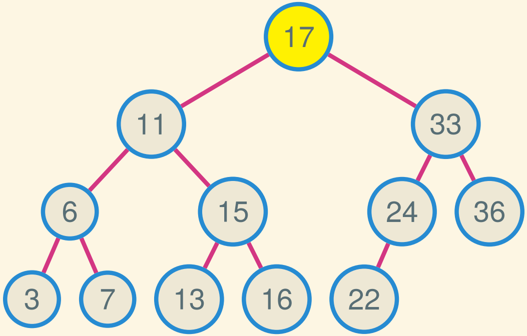
Visit each node *after* its children:



Post-order: 3, 7, 6, 13, 16, 15, 11

Post-order walk

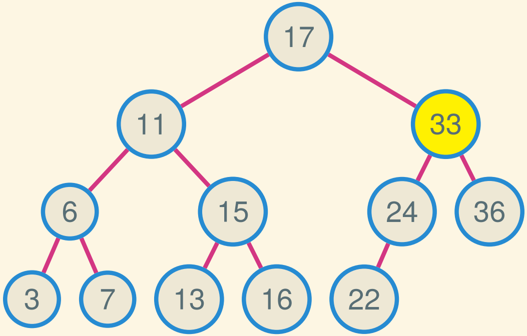
Visit each node *after* its children:



Post-order: 3, 7, 6, 13, 16, 15, 11

Post-order walk

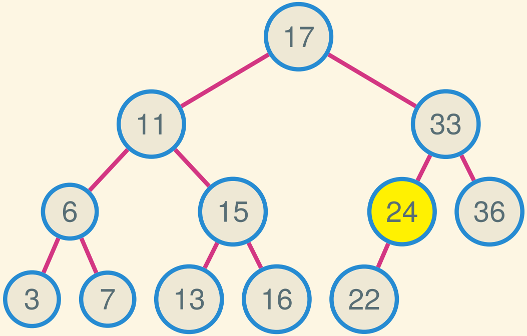
Visit each node *after* its children:



Post-order: 3, 7, 6, 13, 16, 15, 11

Post-order walk

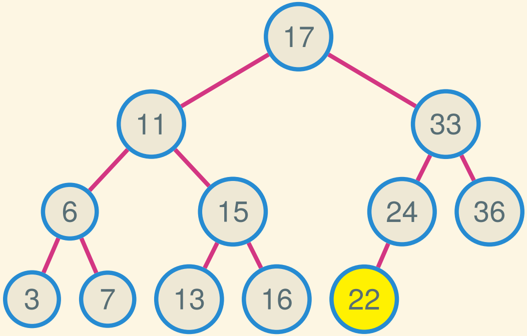
Visit each node *after* its children:



Post-order: 3, 7, 6, 13, 16, 15, 11

Post-order walk

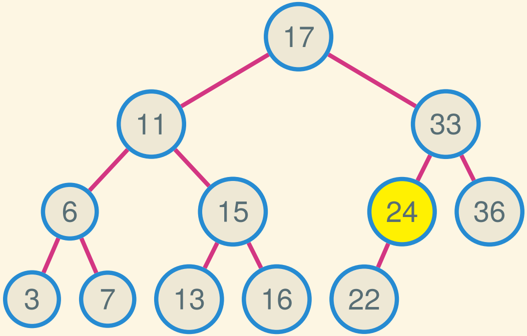
Visit each node *after* its children:



Post-order: 3, 7, 6, 13, 16, 15, 11, 22

Post-order walk

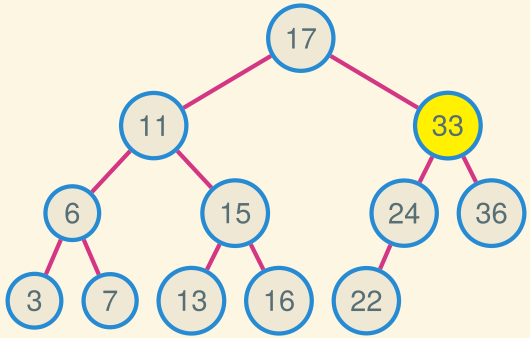
Visit each node *after* its children:



Post-order: 3, 7, 6, 13, 16, 15, 11, 22, 24

Post-order walk

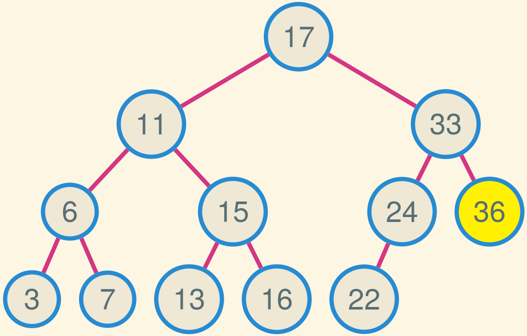
Visit each node *after* its children:



Post-order: 3, 7, 6, 13, 16, 15, 11, 22, 24

Post-order walk

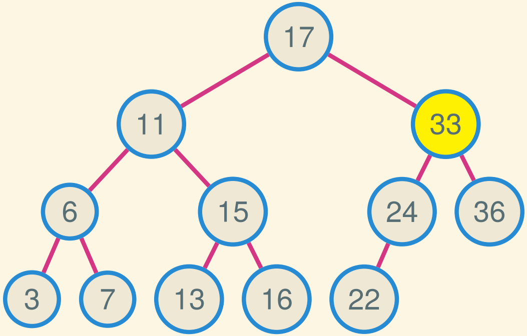
Visit each node *after* its children:



Post-order: 3, 7, 6, 13, 16, 15, 11, 22, 24, 36

Post-order walk

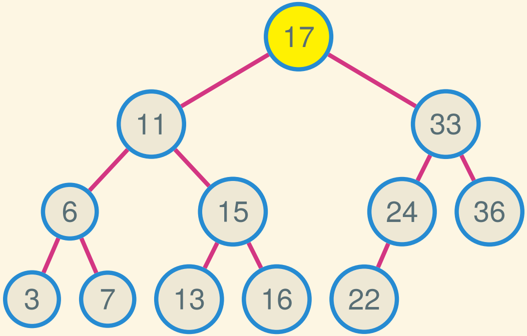
Visit each node *after* its children:



Post-order: 3, 7, 6, 13, 16, 15, 11, 22, 24, 36, 33

Post-order walk

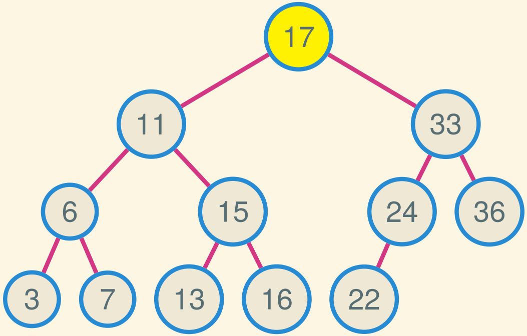
Visit each node *after* its children:



Post-order: 3, 7, 6, 13, 16, 15, 11, 22, 24, 36, 33, 17

In-order walk

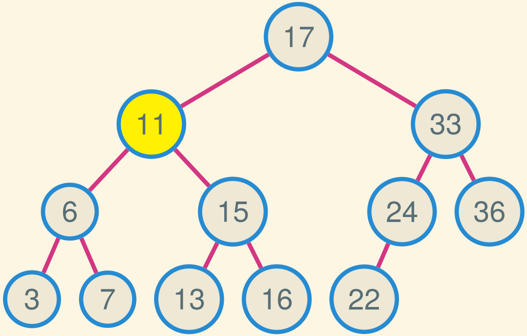
Visit each node *between* its children:



In-order:

In-order walk

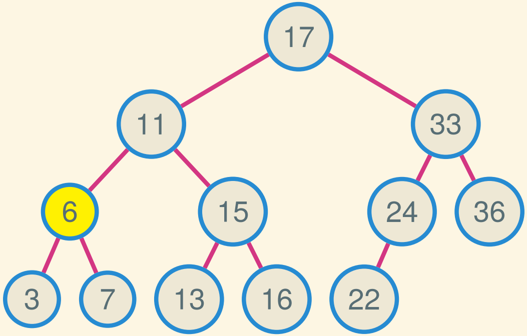
Visit each node *between* its children:



In-order:

In-order walk

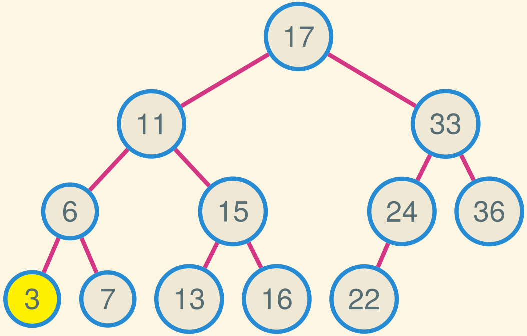
Visit each node *between* its children:



In-order:

In-order walk

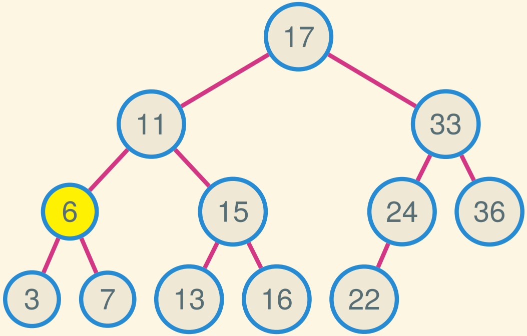
Visit each node *between* its children:



In-order: 3

In-order walk

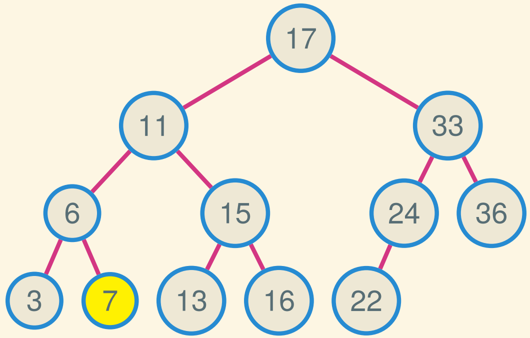
Visit each node *between* its children:



In-order: 3, 6

In-order walk

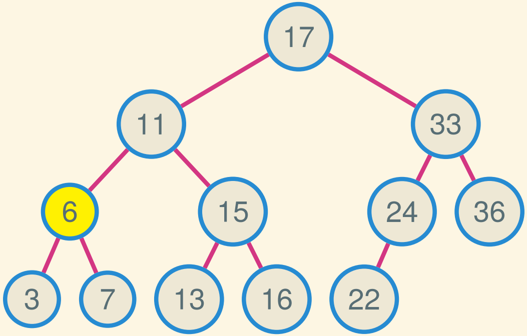
Visit each node *between* its children:



In-order: 3, 6, 7

In-order walk

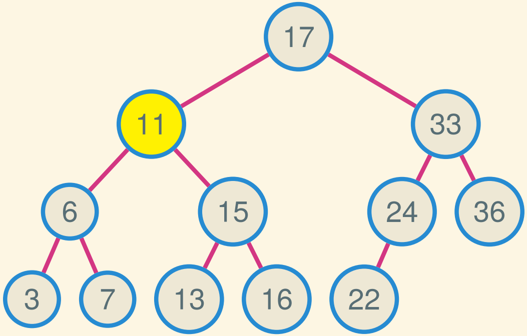
Visit each node *between* its children:



In-order: 3, 6, 7

In-order walk

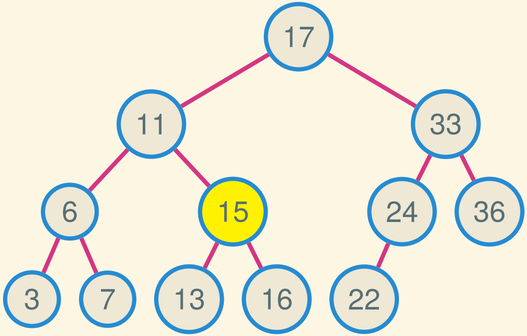
Visit each node *between* its children:



In-order: 3, 6, 7, 11

In-order walk

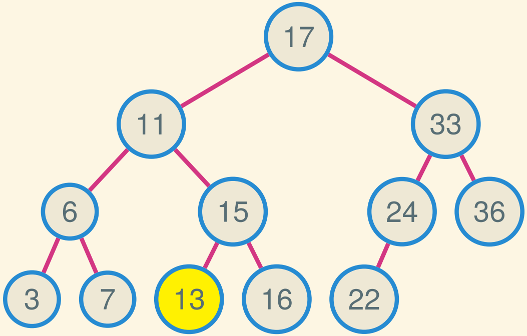
Visit each node *between* its children:



In-order: 3, 6, 7, 11

In-order walk

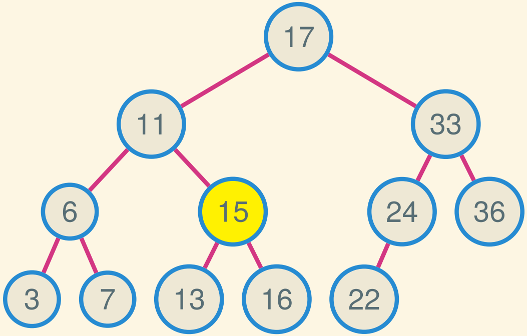
Visit each node *between* its children:



In-order: 3, 6, 7, 11, 13

In-order walk

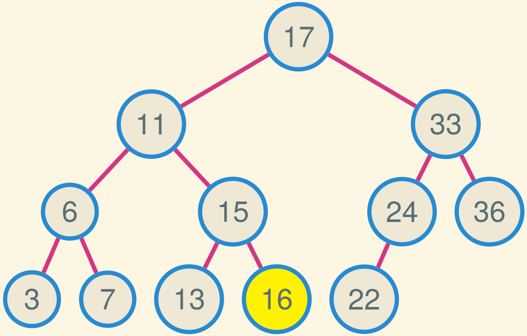
Visit each node *between* its children:



In-order: 3, 6, 7, 11, 13, 15

In-order walk

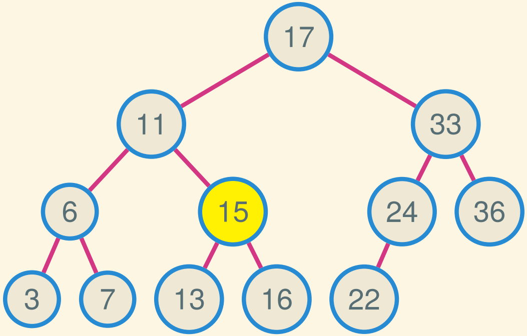
Visit each node *between* its children:



In-order: 3, 6, 7, 11, 13, 15, 16

In-order walk

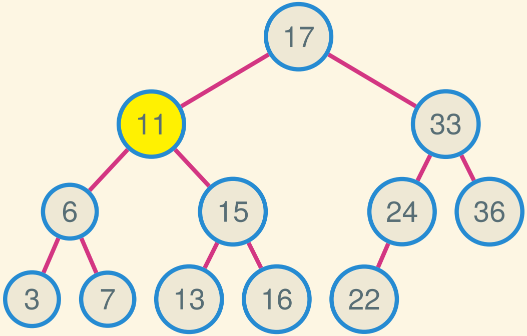
Visit each node *between* its children:



In-order: 3, 6, 7, 11, 13, 15, 16

In-order walk

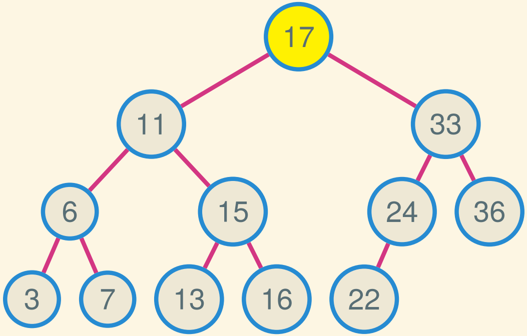
Visit each node *between* its children:



In-order: 3, 6, 7, 11, 13, 15, 16

In-order walk

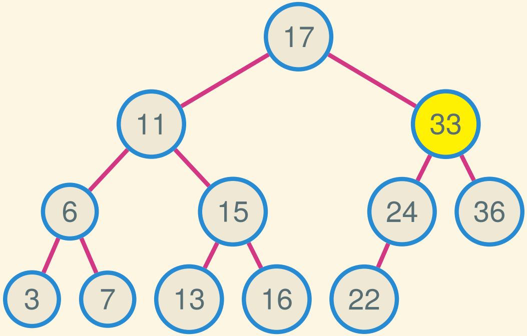
Visit each node *between* its children:



In-order: 3, 6, 7, 11, 13, 15, 16, 17

In-order walk

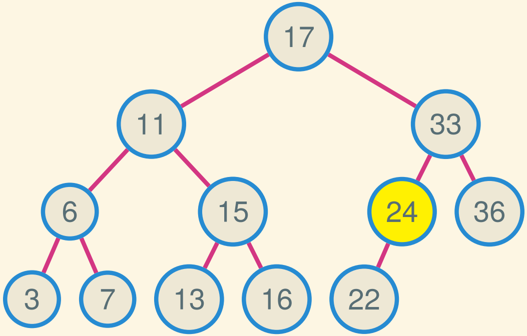
Visit each node *between* its children:



In-order: 3, 6, 7, 11, 13, 15, 16, 17

In-order walk

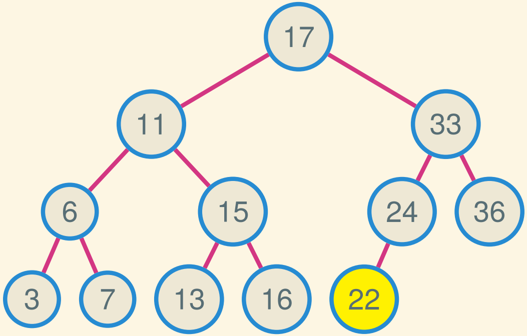
Visit each node *between* its children:



In-order: 3, 6, 7, 11, 13, 15, 16, 17

In-order walk

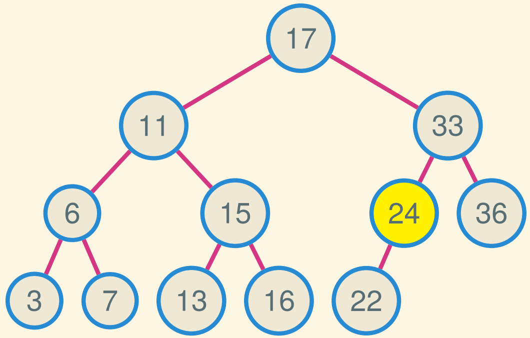
Visit each node *between* its children:



In-order: 3, 6, 7, 11, 13, 15, 16, 17, 22

In-order walk

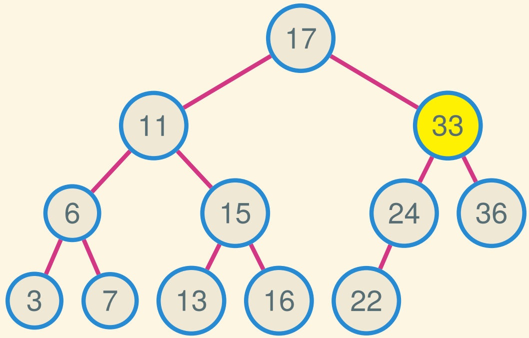
Visit each node *between* its children:



In-order: 3, 6, 7, 11, 13, 15, 16, 17, 22, 24

In-order walk

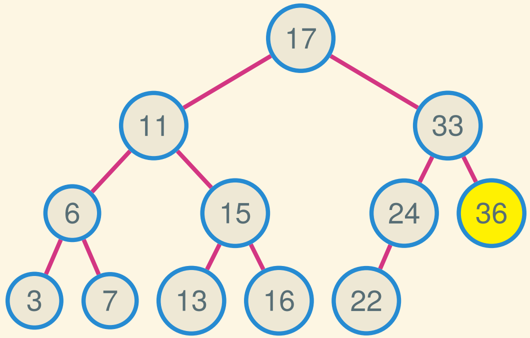
Visit each node *between* its children:



In-order: 3, 6, 7, 11, 13, 15, 16, 17, 22, 24, 33

In-order walk

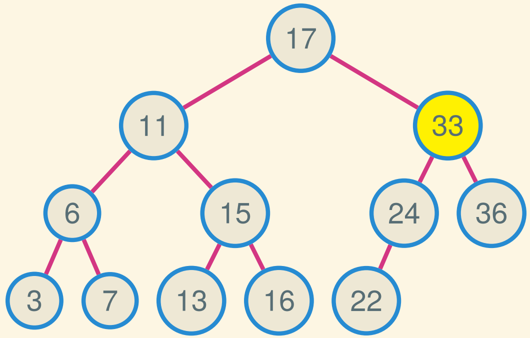
Visit each node *between* its children:



In-order: 3, 6, 7, 11, 13, 15, 16, 17, 22, 24, 33, 36

In-order walk

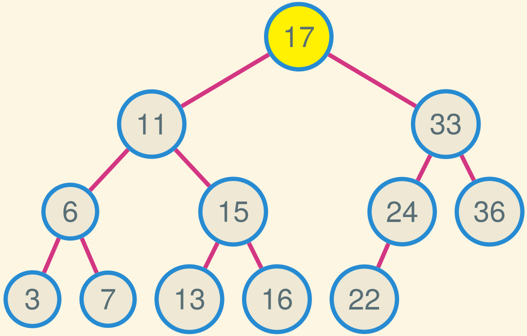
Visit each node *between* its children:



In-order: 3, 6, 7, 11, 13, 15, 16, 17, 22, 24, 33, 36

In-order walk

Visit each node *between* its children:



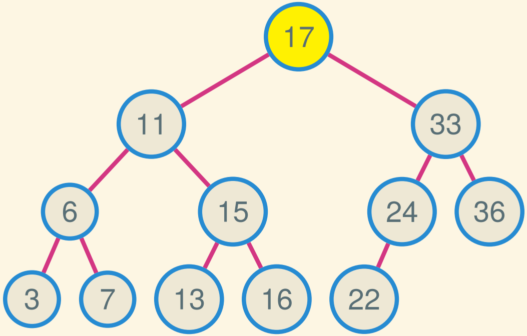
In-order: 3, 6, 7, 11, 13, 15, 16, 17, 22, 24, 33, 36

Tree walk pseudocode

```
Procedure PreOrder(node) is
    if node is not null then
        visit node;
        PreOrder(node.left);
        PreOrder(node.right)
    end
end

Procedure PostOrder(node) is
    if node is not null then
        PostOrder(node.left);
        PostOrder(node.right);
        visit node
    end
end

Procedure InOrder(node) is
    if node is not null then
        InOrder(node.left);
        visit node;
        InOrder(node.right)
    end
end
```

Level-order walk

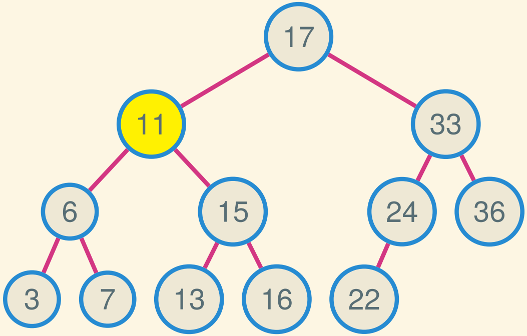
Visit all of each level before the next level:



Level-order: 17

Level-order walk

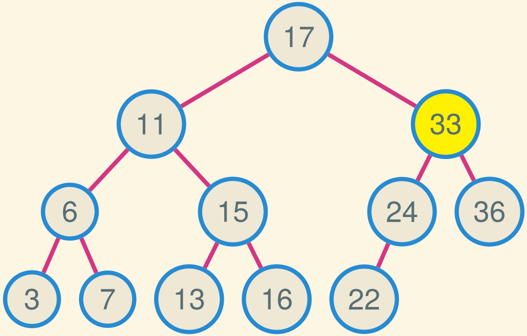
Visit all of each level before the next level:



Level-order: 17, 11

Level-order walk

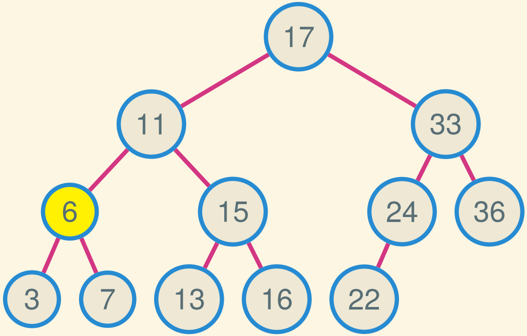
Visit all of each level before the next level:



Level-order: 17, 11, 33

Level-order walk

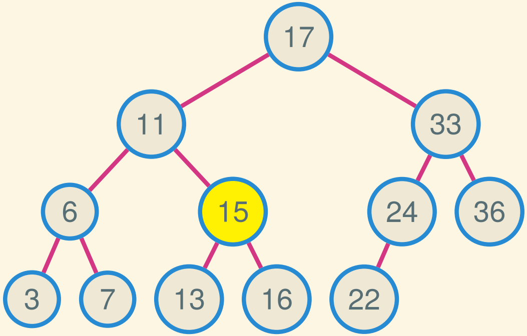
Visit all of each level before the next level:



Level-order: 17, 11, 33, 6

Level-order walk

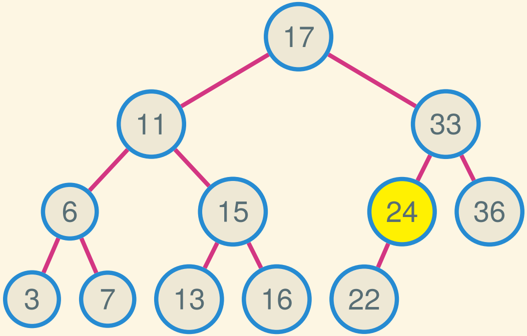
Visit all of each level before the next level:



Level-order: 17, 11, 33, 6, 15

Level-order walk

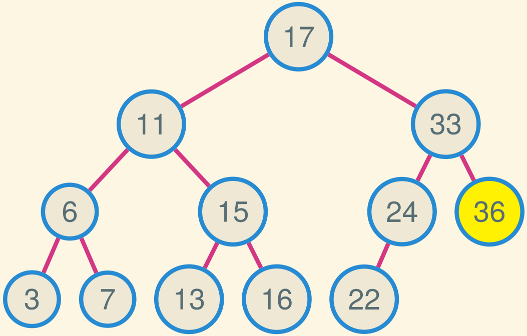
Visit all of each level before the next level:



Level-order: 17, 11, 33, 6, 15, 24

Level-order walk

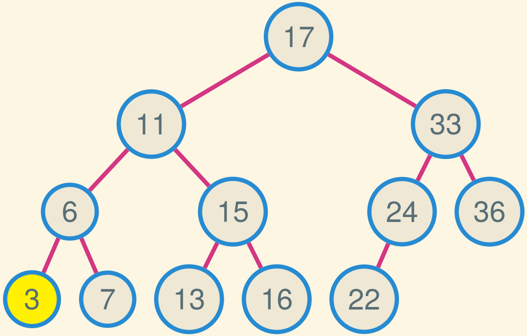
Visit all of each level before the next level:



Level-order: 17, 11, 33, 6, 15, 24, 36

Level-order walk

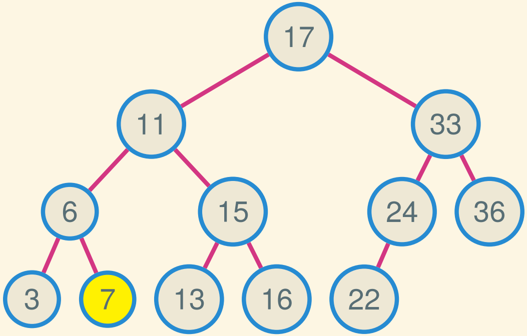
Visit all of each level before the next level:



Level-order: 17, 11, 33, 6, 15, 24, 36, 3

Level-order walk

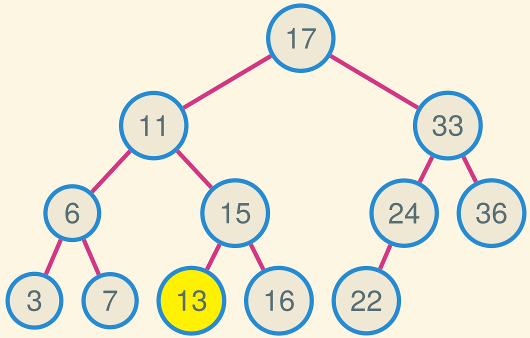
Visit all of each level before the next level:



Level-order: 17, 11, 33, 6, 15, 24, 36, 3, 7

Level-order walk

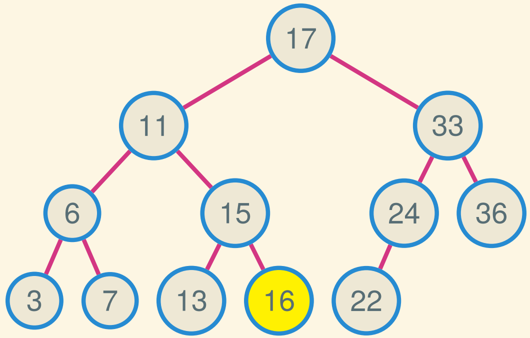
Visit all of each level before the next level:



Level-order: 17, 11, 33, 6, 15, 24, 36, 3, 7, 13

Level-order walk

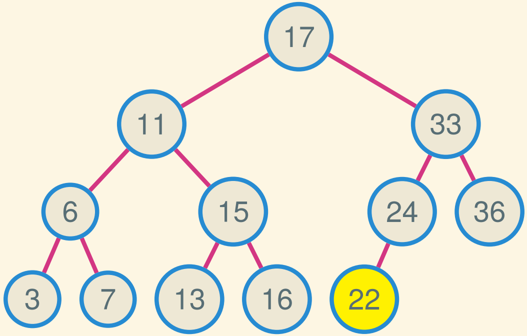
Visit all of each level before the next level:



Level-order: 17, 11, 33, 6, 15, 24, 36, 3, 7, 13, 16

Level-order walk

Visit all of each level before the next level:



Level-order: 17, 11, 33, 6, 15, 24, 36, 3, 7, 13, 16, 22

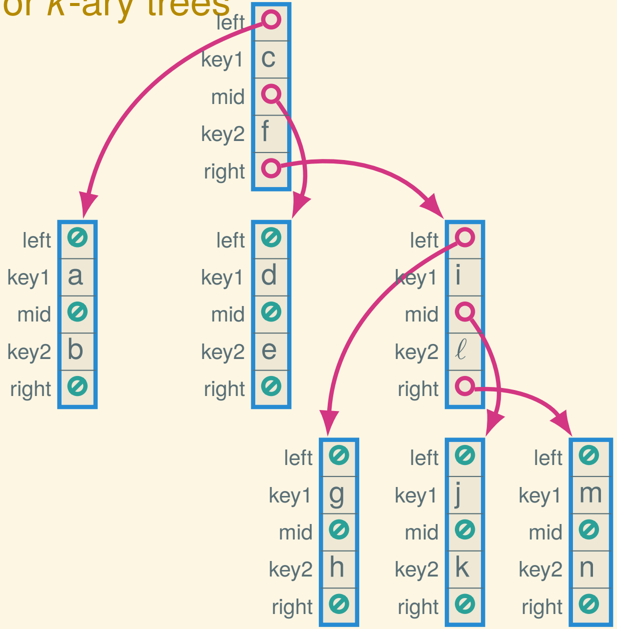
Level-order pseudocode

We use a queue (FIFO) to visit the nodes level-by-level:

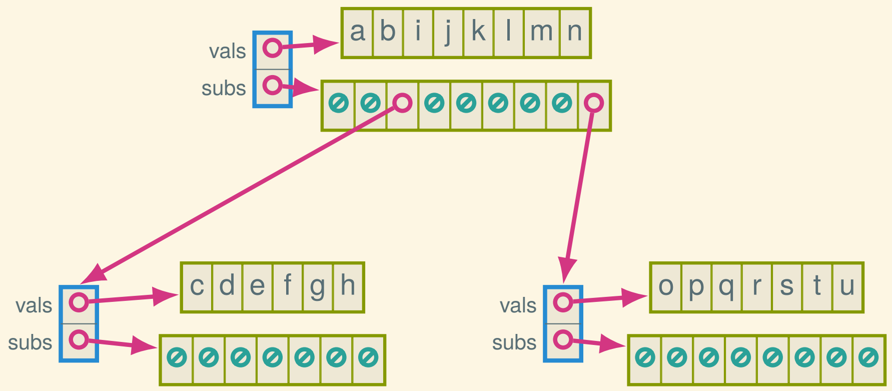
```
Procedure LevelOrder(root) is
    queue ← a new queue;
    Enqueue(queue, root);

    while queue is not empty do
        node ← Dequeue(queue);
        if node is not null then
            visit node;
            Enqueue(queue, node.left);
            Enqueue(queue, node.right)
        end
    end
end
```

Representing trees

Structs for *k*-ary trees

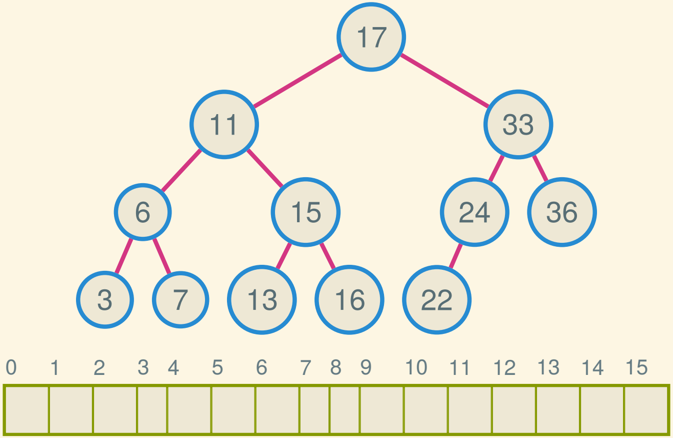


Rose trees using arrays



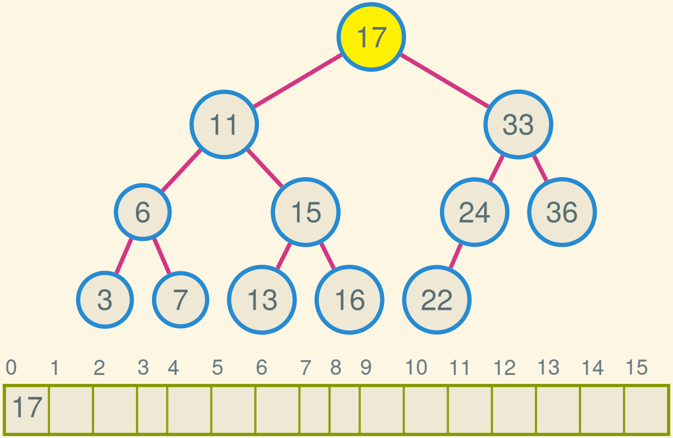
Complete binary trees in level order in an array

A very special case:



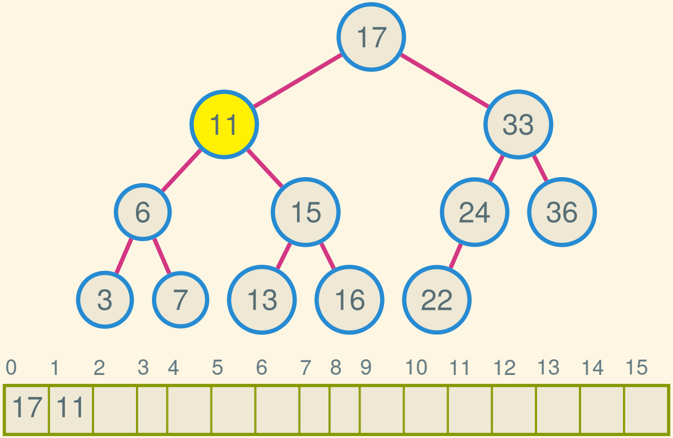
Complete binary trees in level order in an array

A very special case:



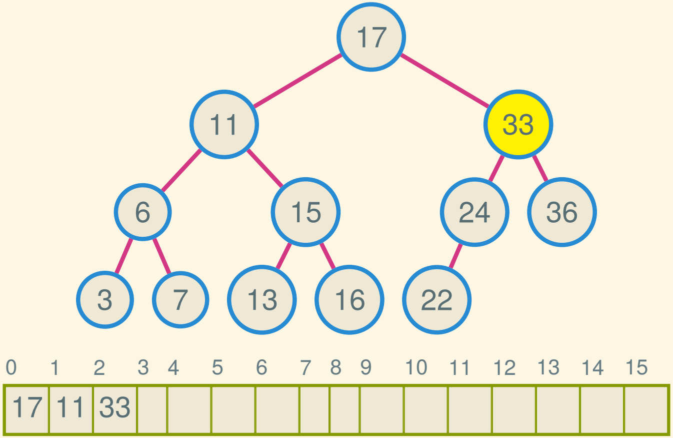
Complete binary trees in level order in an array

A very special case:



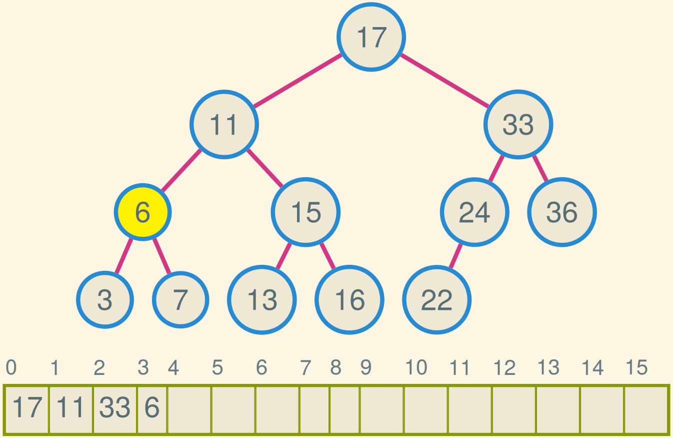
Complete binary trees in level order in an array

A very special case:



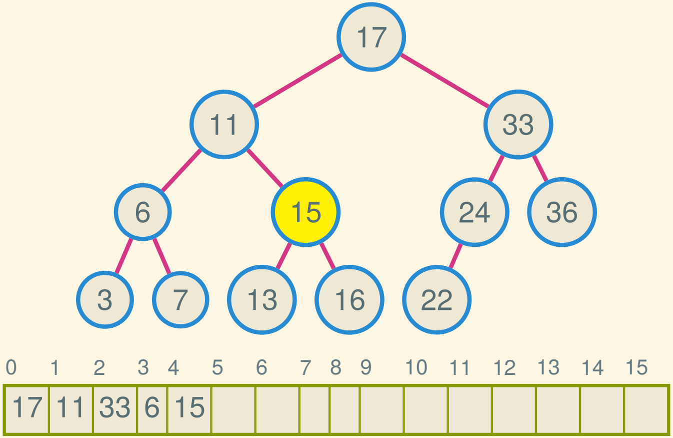
Complete binary trees in level order in an array

A very special case:



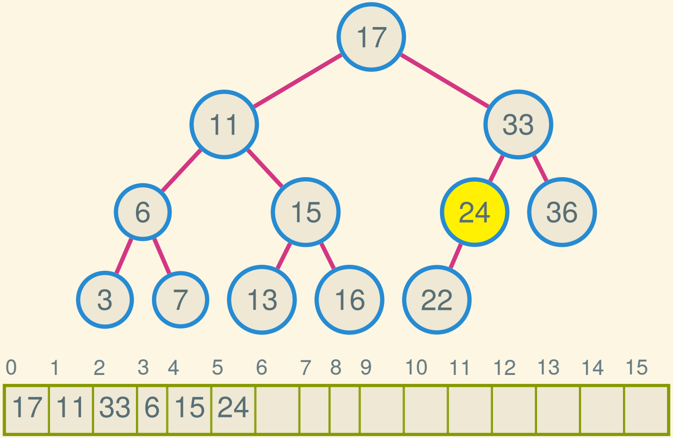
Complete binary trees in level order in an array

A very special case:



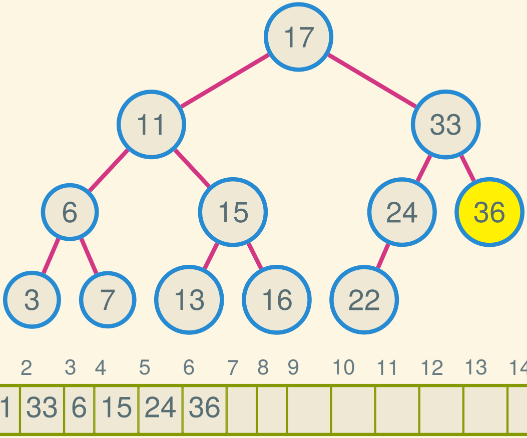
Complete binary trees in level order in an array

A very special case:



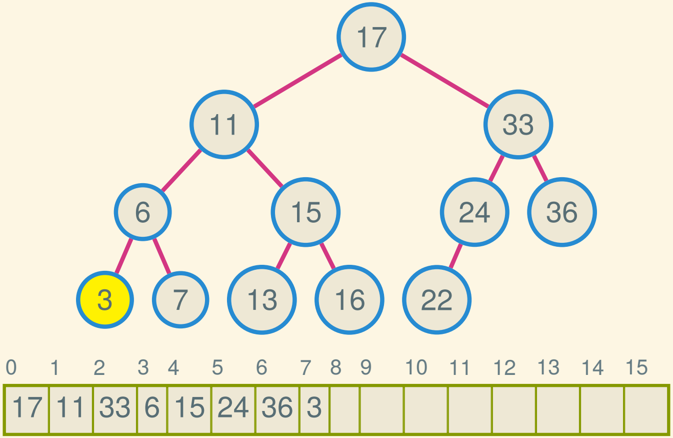
Complete binary trees in level order in an array

A very special case:



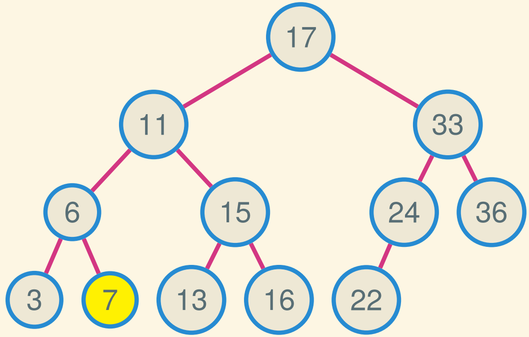
Complete binary trees in level order in an array

A very special case:



Complete binary trees in level order in an array

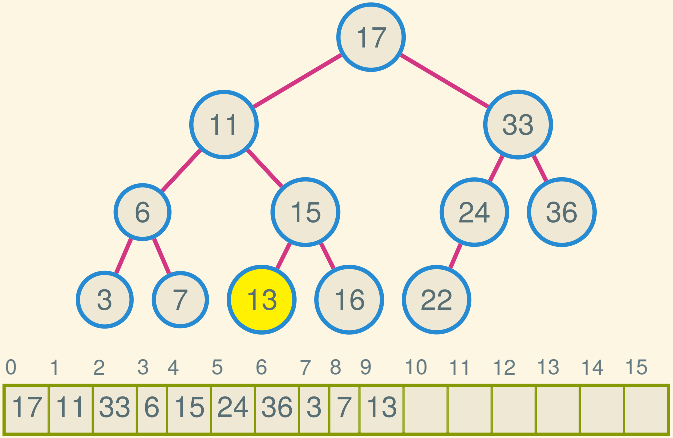
A very special case:



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
17	11	33	6	15	24	36	3	7							

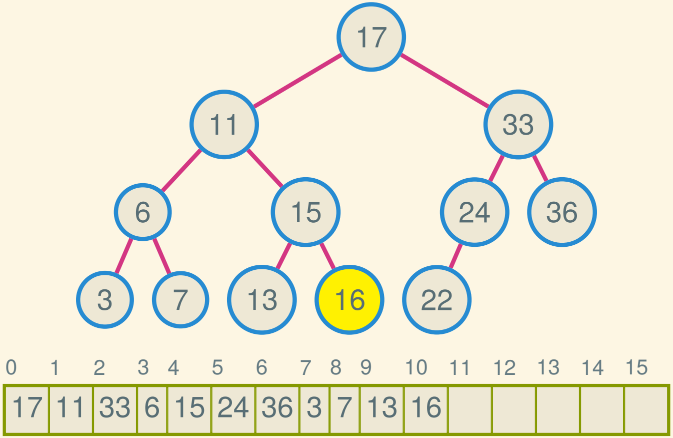
Complete binary trees in level order in an array

A very special case:



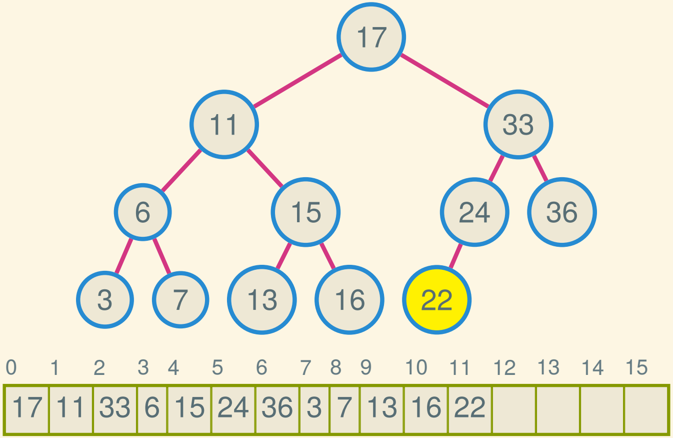
Complete binary trees in level order in an array

A very special case:



Complete binary trees in level order in an array

A very special case:



Next time: graphs