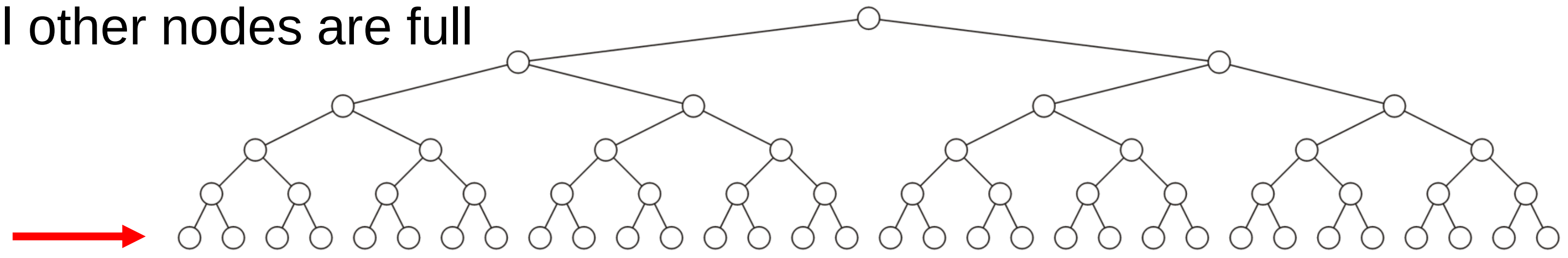
Perfect binary trees

Definition

Standard definition:

- A perfect binary tree of height h is a binary tree where
  - All leaf nodes have the same depth h
  - All other nodes are full



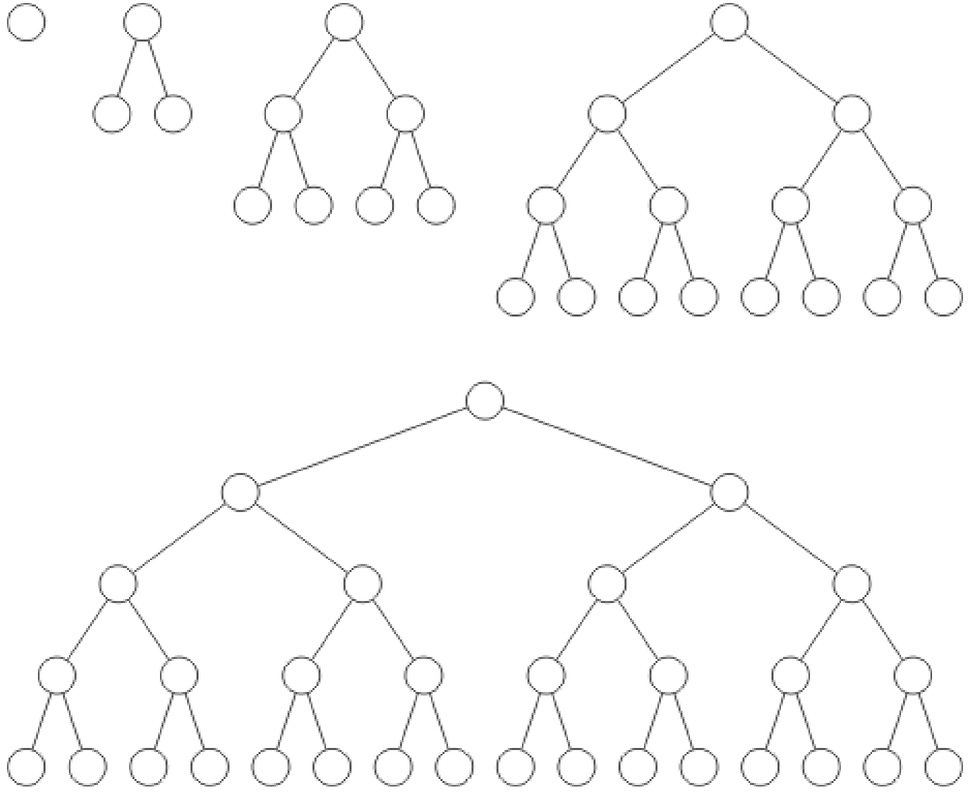
Definition

Recursive definition:

- A binary tree of height $h = 0$ is perfect
- A binary tree with height $h > 0$ is a perfect if both sub-trees are prefect binary trees of height $h - 1$

Examples

Perfect binary trees of height $h = 0, 1, 2, 3$ and 4



Theorems

We will now look at four theorems that describe the properties of perfect binary trees:

- A perfect tree has $2^{h+1} - 1$
- The height is $\Theta(\ln(n))$
- There are 2^h leaf nodes
- The average depth of a node is $\Theta(\ln(n))$

The results of these theorems will allow us to determine the optimal run-time properties of operations on binary trees

Applications

Perfect binary trees are considered to be the *ideal* case

- The height and average depth are both $\Theta(\ln(n))$

We will attempt to find trees which are as close as possible to perfect binary trees

Complete binary trees

Background

A perfect binary tree has ideal properties but restricted in the number of nodes: $n = 2^{h+1} - 1$ for $h = 0, 1, \ldots$

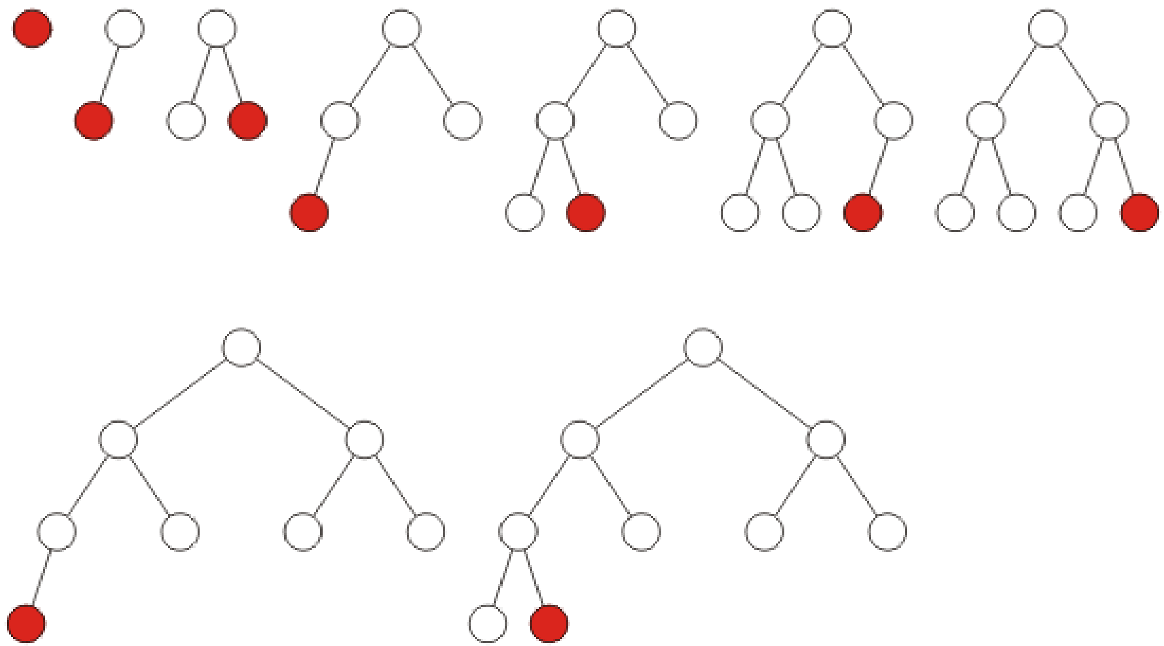
$$1, 3, 7, 15, 31, 63, 127, 255, 511, 1023, ...$$

We require binary trees which are

- Similar to perfect binary trees, but
- Defined for all n

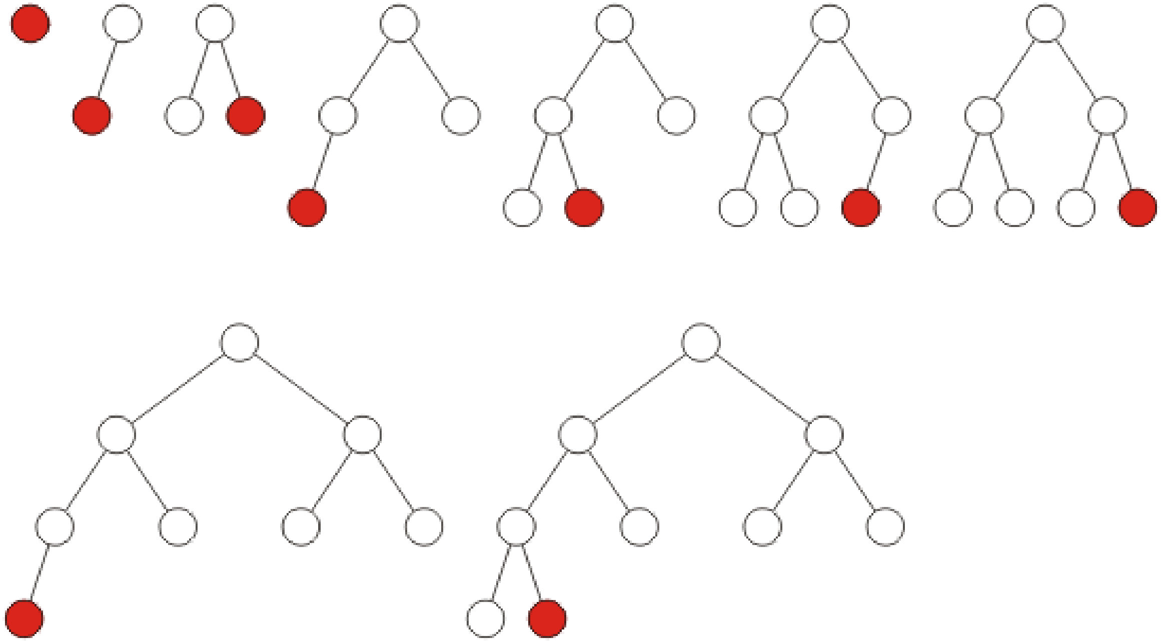
Definition

A complete binary tree filled at each depth from left to right:



Definition

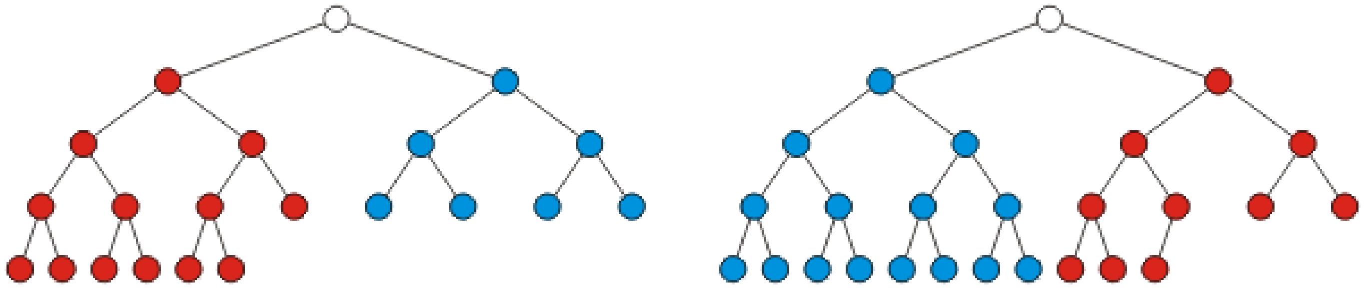
The order is identical to that of a breadth-first traversal



Recursive Definition

Recursive definition: a binary tree with a single node is a complete binary tree of height $h = 0$ and a complete binary tree of height h is a tree where either:

- The left sub-tree is a **complete tree** of height $h - 1$ and the right sub-tree is a **perfect tree** of height $h - 2$, or
- The left sub-tree is **perfect tree** with height $h - 1$ and the right sub-tree is **complete tree** with height $h - 1$



Height

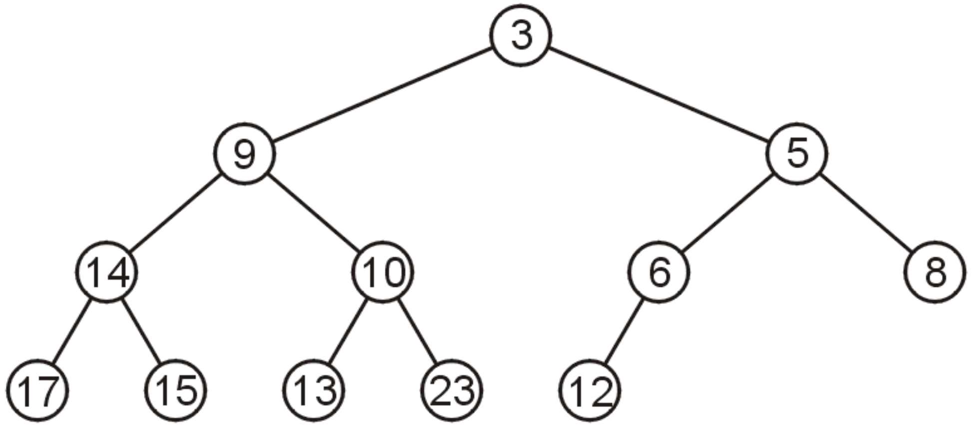
Theorem

The height of a complete binary tree with n nodes is $h = \lfloor \lg(n) \rfloor$

Array storage

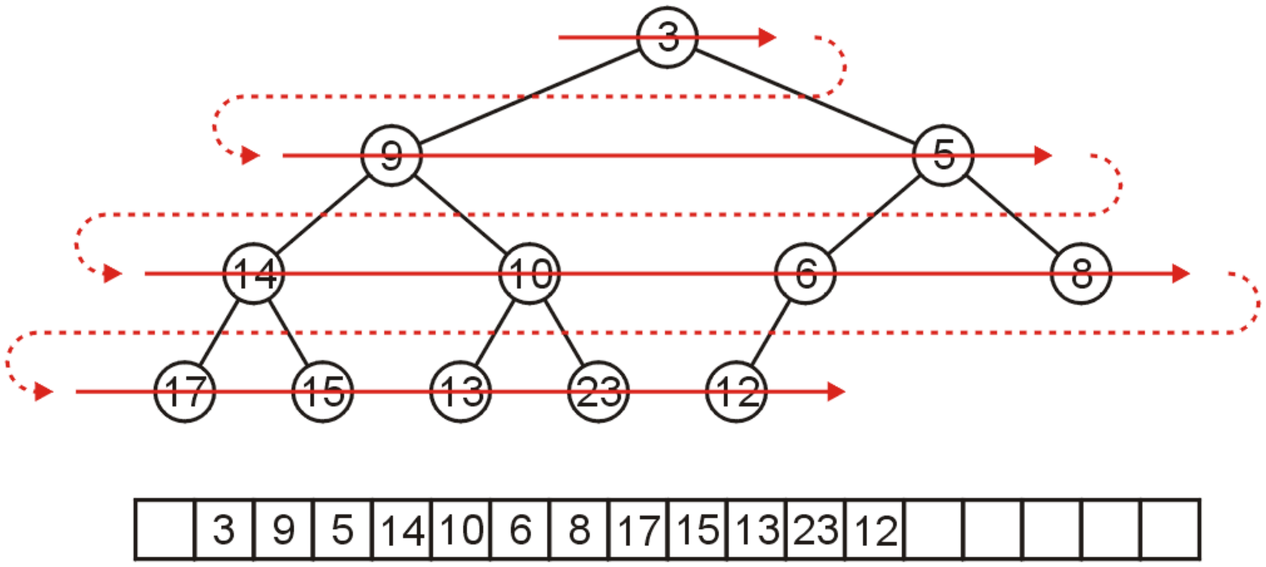
We are able to store a complete tree as an array

- Traverse the tree in breadth-first order, placing the entries into the array



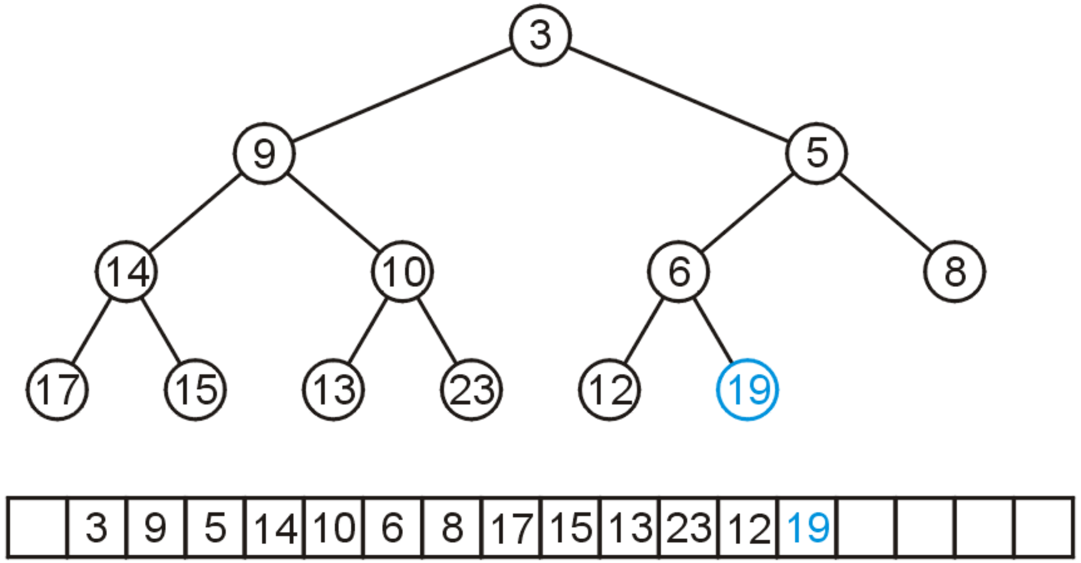
Array storage

We can store this in an array after a quick traversal:



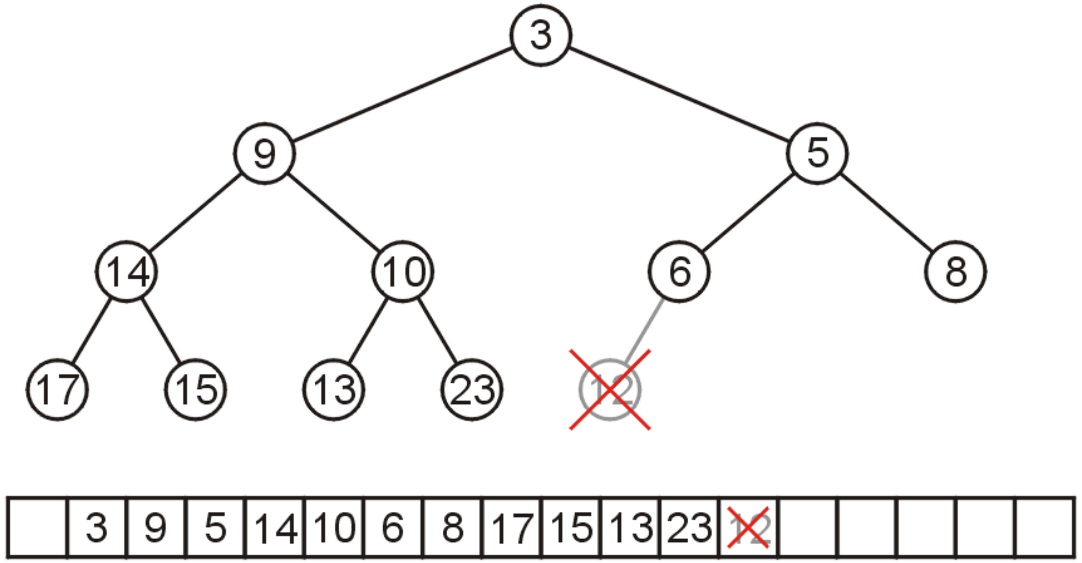
Array storage

To insert another node while maintaining the complete-binary-tree structure, we must insert into the next array location



Array storage

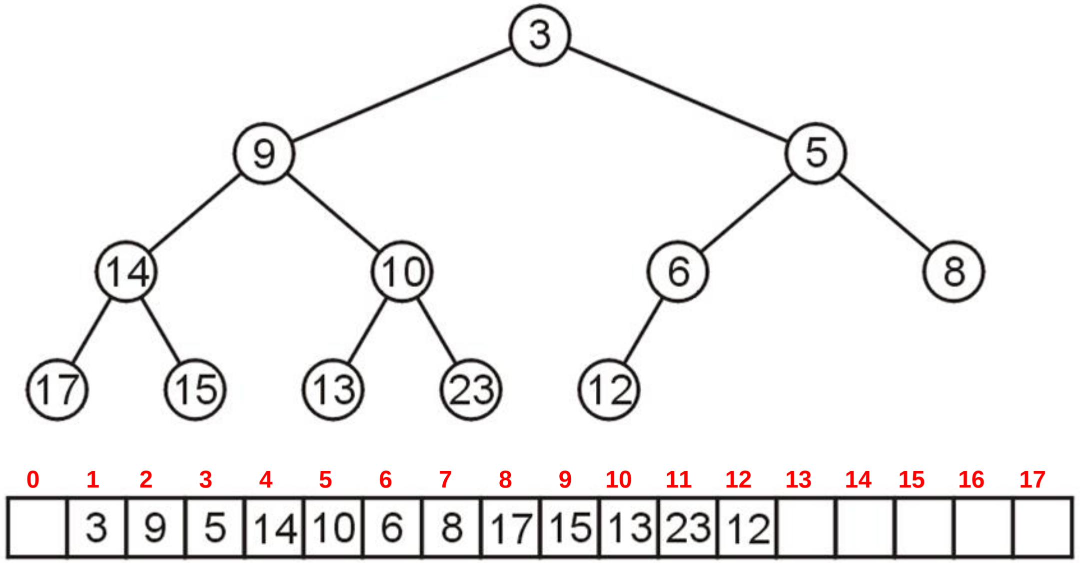
To remove a node while keeping the complete-tree structure, we must remove the last element in the array



Array storage

Leaving the first entry blank yields a bonus:

- The children of the node with index k are in $2k$ and $2k + 1$
- The parent of node with index k is in $k \div 2$

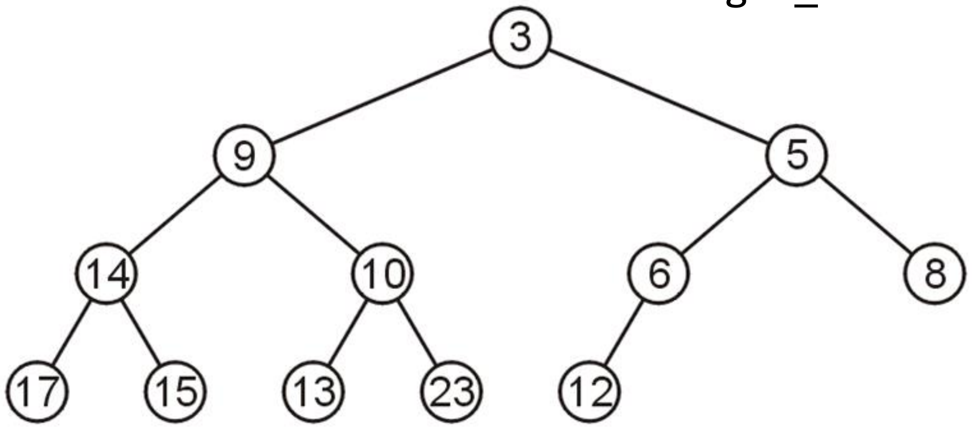


Array storage

Leaving the first entry blank yields a bonus:

– In C++, this simplifies the calculations:

```
parent = k >> 1;
left_child = k << 1;
right_child = left_child | 1;
```

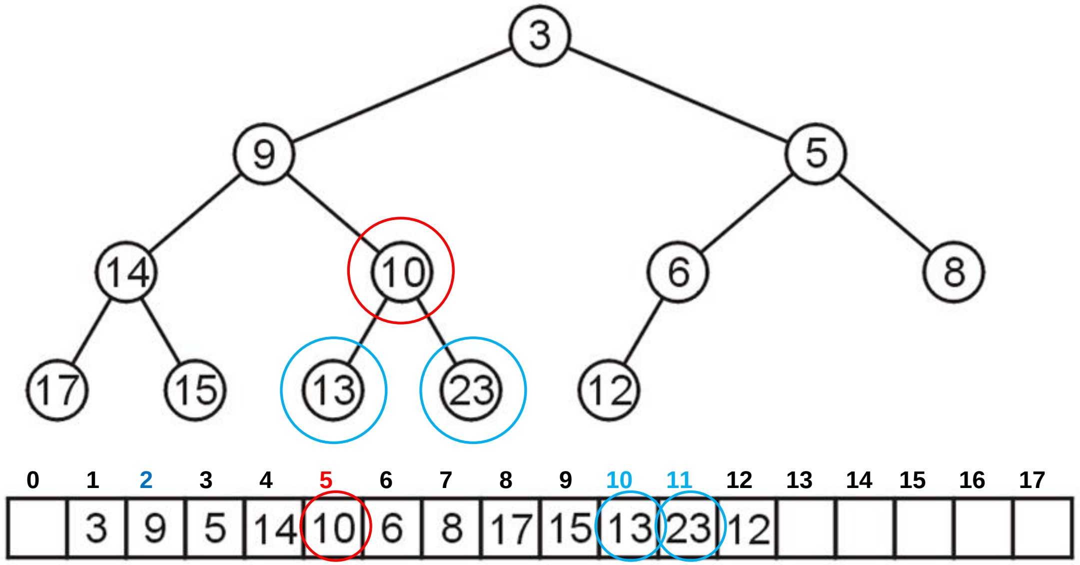


0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	3	9	5	14	10	6	8	17	15	13	23	12					

Array storage

For example, node 10 has index 5:

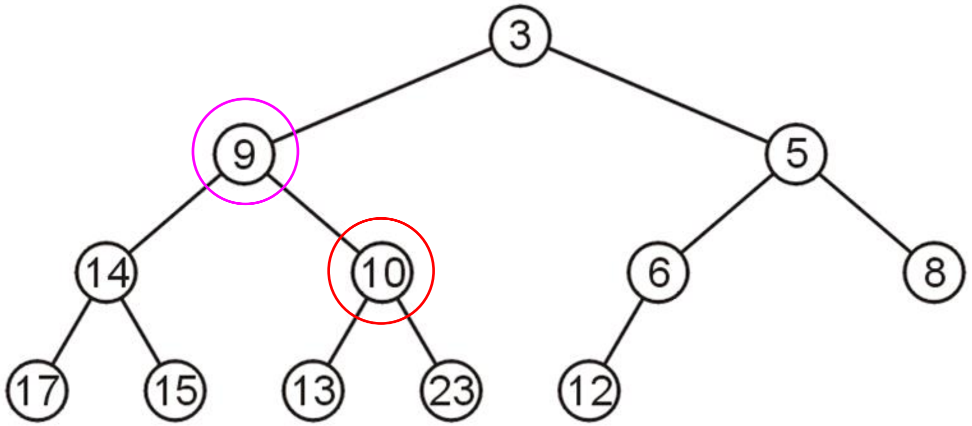
- Its children 13 and 23 have indices 10 and 11, respectively



Array storage

For example, node 10 has index 5:

- Its children 13 and 23 have indices 10 and 11, respectively
- Its parent is node 9 with index 5/2 = **2**



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	3	9	5	14	10	6	8	17	15	13	23	12					

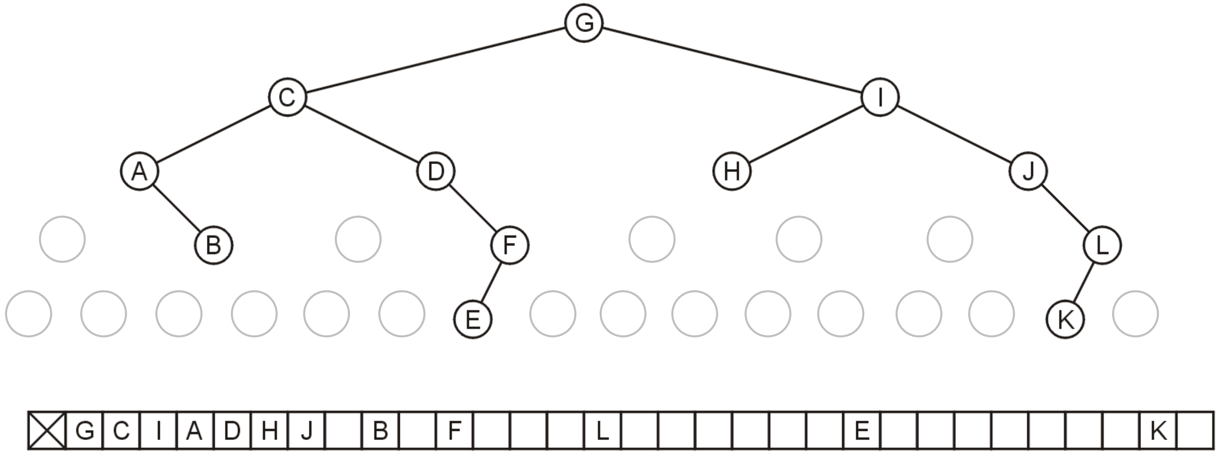
Array storage

Question: why not store any tree as an array using breadth-first traversals?

- There is a significant potential for a lot of wasted memory

Consider this tree with 12 nodes would require an array of size 32

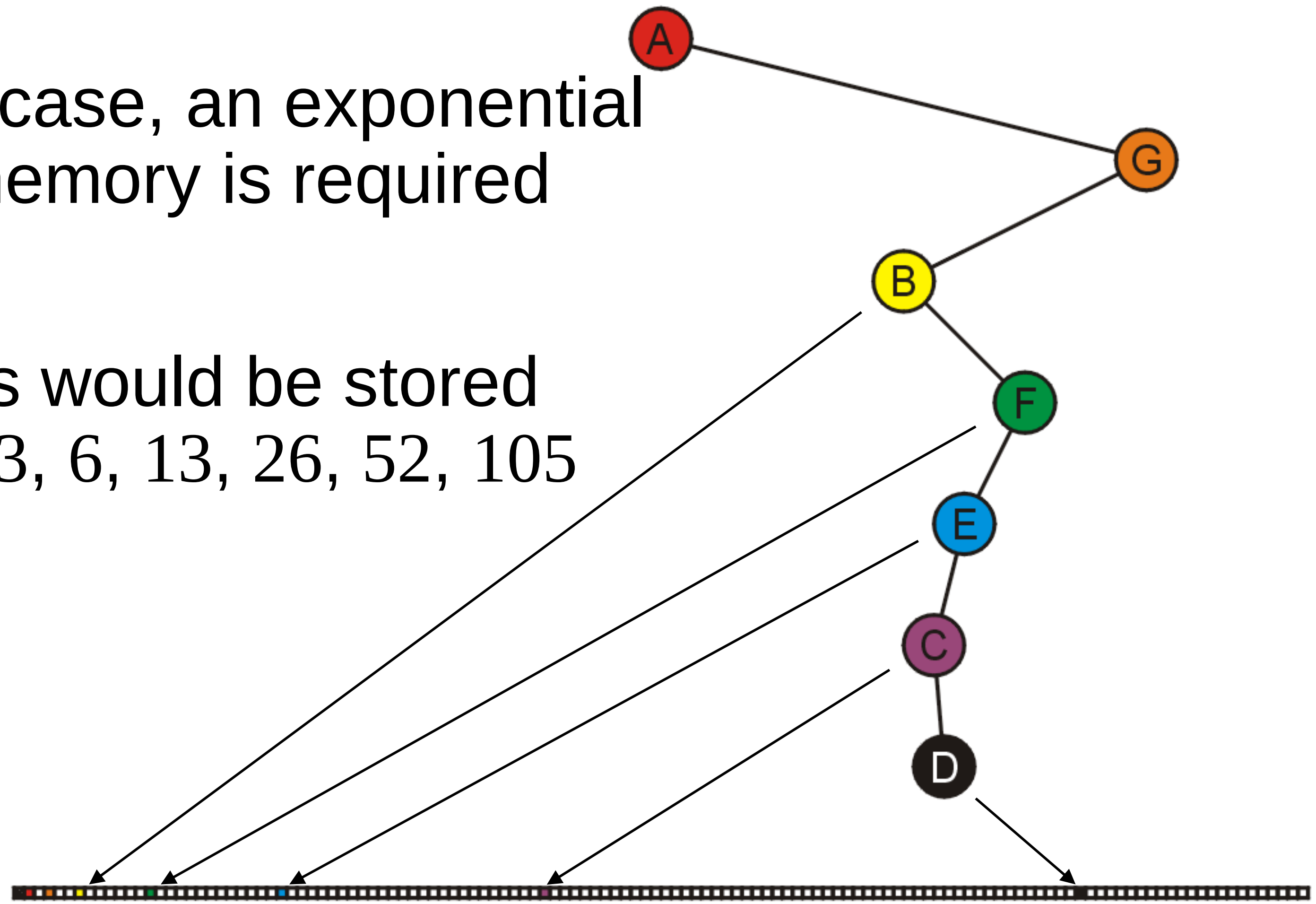
- Adding a child to node K doubles the required memory



Array storage

In the worst case, an exponential amount of memory is required

These nodes would be stored in entries 1, 3, 6, 13, 26, 52, 105



N-ary Trees

N-ary Trees

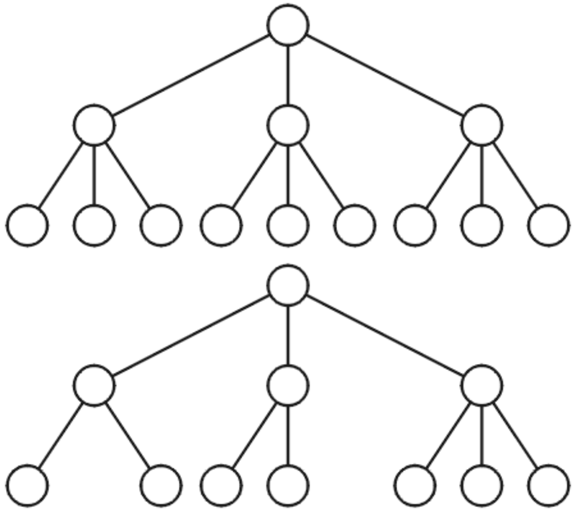
One generalization of binary trees are a class of trees termed N-ary trees:

- A tree where each node had N sub-trees, any of which may be may be empty trees

Ternary Trees

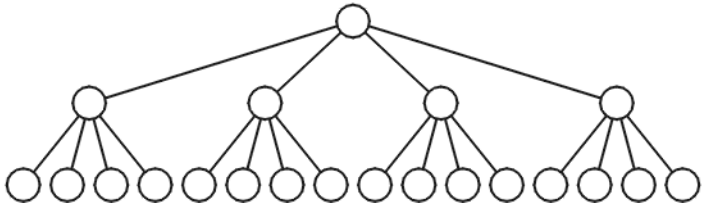
Examples of a ternary (3-ary) trees

- We don’t usually indicate empty sub-trees



Quaternary Trees

Example of a perfect quaternary (4-ary) tree



Implementation of N-ary Trees

The "obvious" implementation of N-ary trees may be something like:

```
# include <algorithm>

template <typename Type>
class Nary_tree {
    private:
        Type node_value;
        int N;
        Nary_tree **children;

    public:
        Nary_tree( Type const &, int = 2 );
        // ...
};
```

```
template <typename Type>
Nary_tree<Type>::Nary_tree( Type const &e, int n ):
node_value( e ),
N( std::max( 2, n ) ),
children( new *Nary_tree[N] ) {
    for ( int i = 0; i < N; ++i ) {
        children[i] = nullptr;
    }
}
```

Implementation of N-ary Trees

Problems with this implementation:

- Requires dynamic memory allocation
- A destructor is required to delete the memory
- No optimizations possible
- Dynamic memory allocation may not always be available (embedded systems)

Solution?

- Specify N at compile time...

N-ary Trees with Template Parameters

```
#include <algorithm>

template <typename Type, int N>
class Nary_tree {
    private:
        Type node_value;
        Nary_tree *children[std::max(N, 2)];   // an array of N children
    public:
        Nary_tree( Type const & = Type() )
        // ...
};

template <typename Type, int N>
Nary_tree<Type, N>::Nary_tree( Type const &e ):node_value( e ) {
    for ( int i = 0; i < N; ++i ) {
        children[i] = nullptr;
    }
}
```

N-ary Trees with Template Parameters

Sample code using this class:

```
Nary_tree<int, 4> i4tree( 1975 );  // create a 4-way tree
std::cout << i4tree.value() << std::endl;
```

N-ary Trees

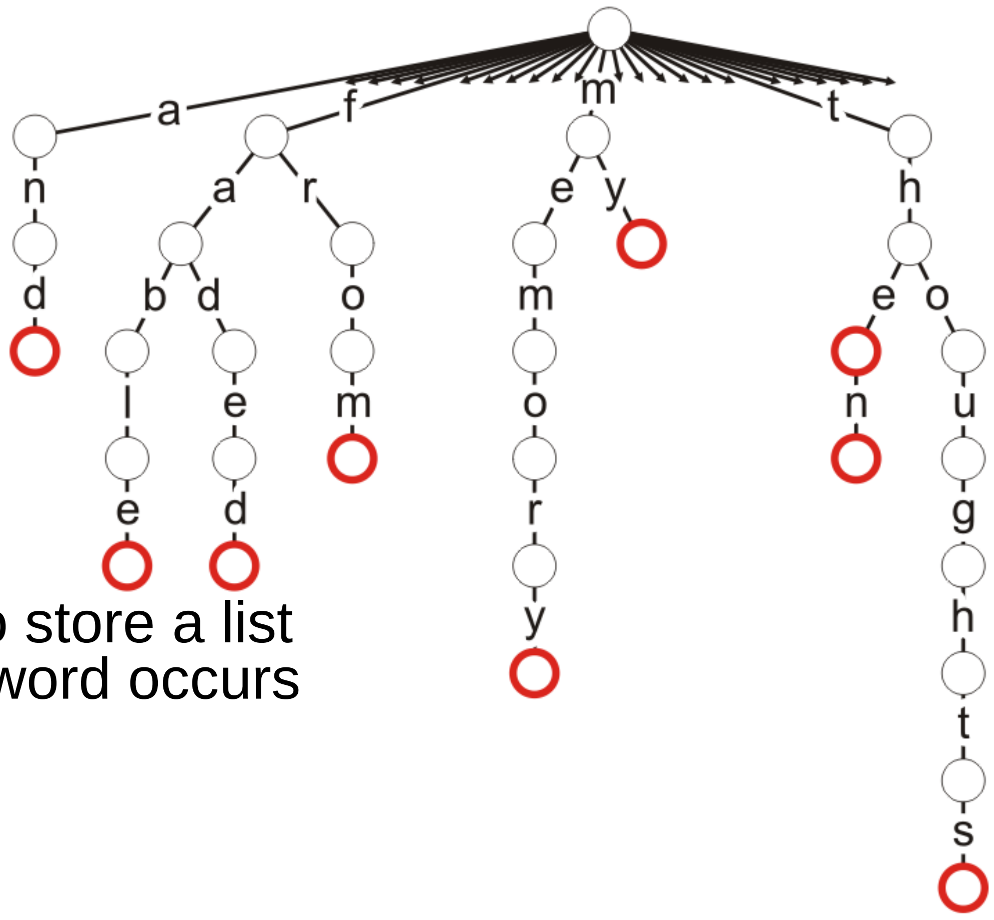
Because the size of the array (the 2nd template parameter) is specified at compile time:

- The compiler can make certain optimizations
- All memory is allocated at once
  - Possibly even on the stack at compile time
- No destructor required

Applications

One application of an 26-ary trees is a *trie* where the root represents the *start* of each valid word, and the different sub-trees represent next letters in valid words

- Consider the words in the phrase
  “The fable then faded from my thoughts and memory.”
- All 26 sub-trees are only shown for the root node, but all nodes have 26 sub-trees
- Some nodes are marked as *terminal* indicating the end of a valid word
- These *terminal* points could be used to store a list of all places in a document where the word occurs
  - Consider the ultimate index to a book



Left-child right-sibling binary tree

Background

Our simple tree data structure is node-based where children are stored as a linked list

- Is it possible to store a general tree as a binary tree?

The Idea

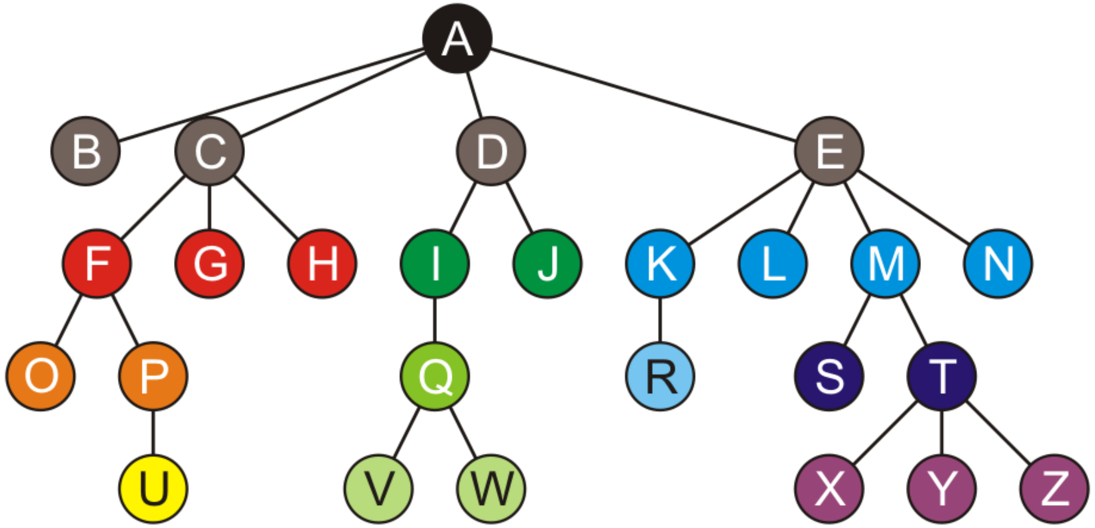
Consider the following:

- The first child of each node is its left sub-tree
- The next sibling of each node is in its right sub-tree

This is called a left-child—right-sibling binary tree

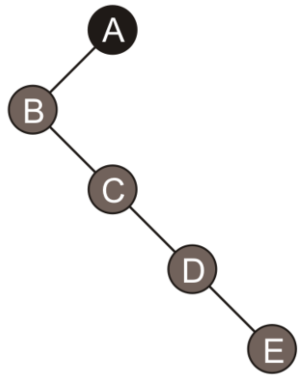
Example

Consider this general tree



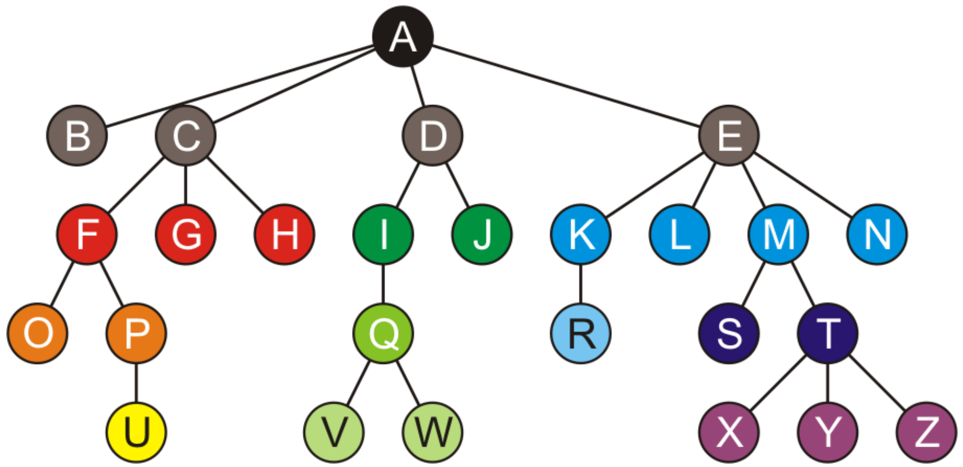
Example

B, the first child of A, is the left child of A



For the three siblings C, D, E:

- C is the right sub-tree of B
- D is the right sub-tree of C
- E is the right sub-tree of D



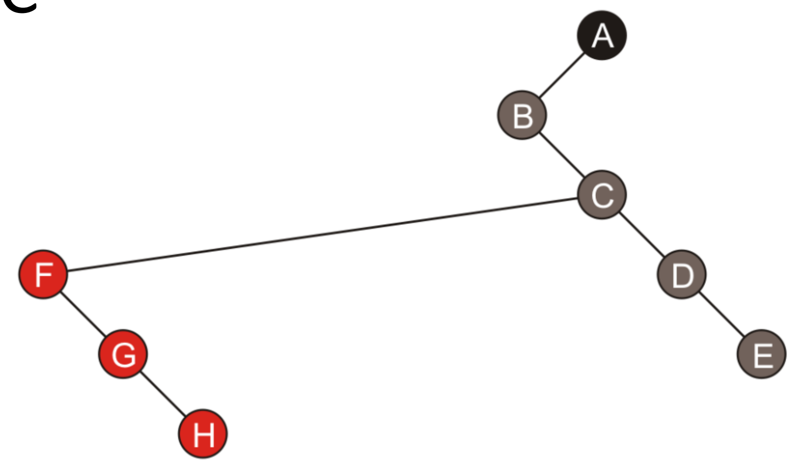
Example

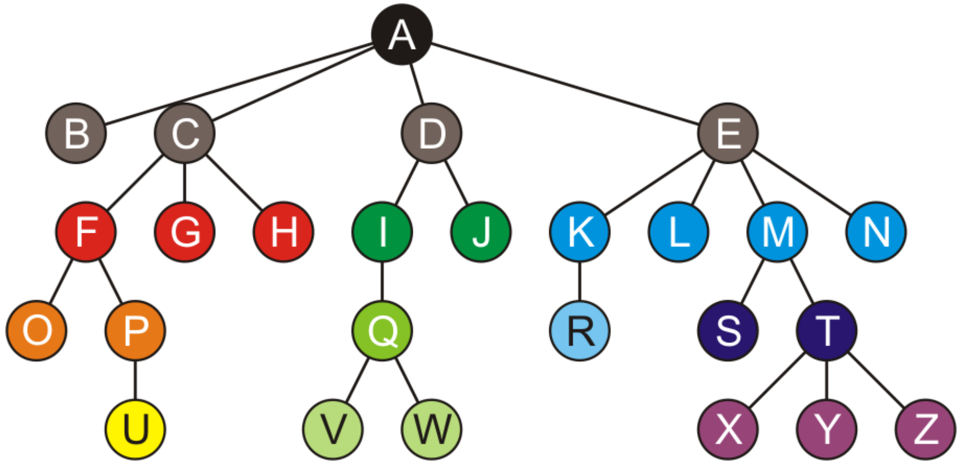
B has no children, so it's left sub-tree is empty

F, the first child of C, is the left sub-tree of C

For the next two siblings:

- G is the right sub-tree of F
- H is the right sub-tree of G

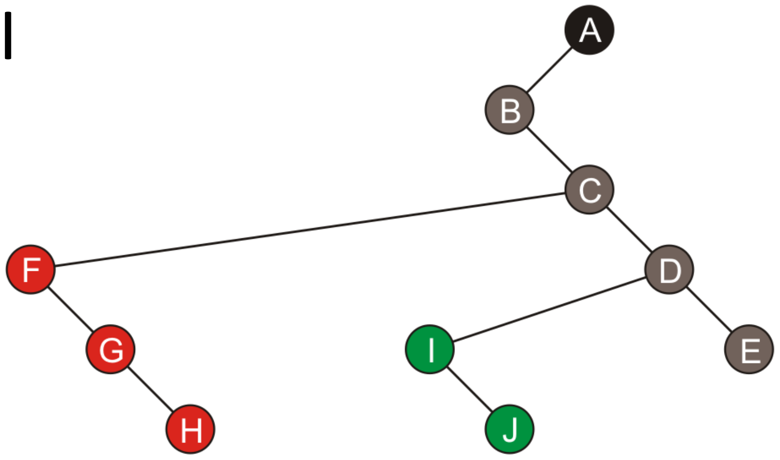


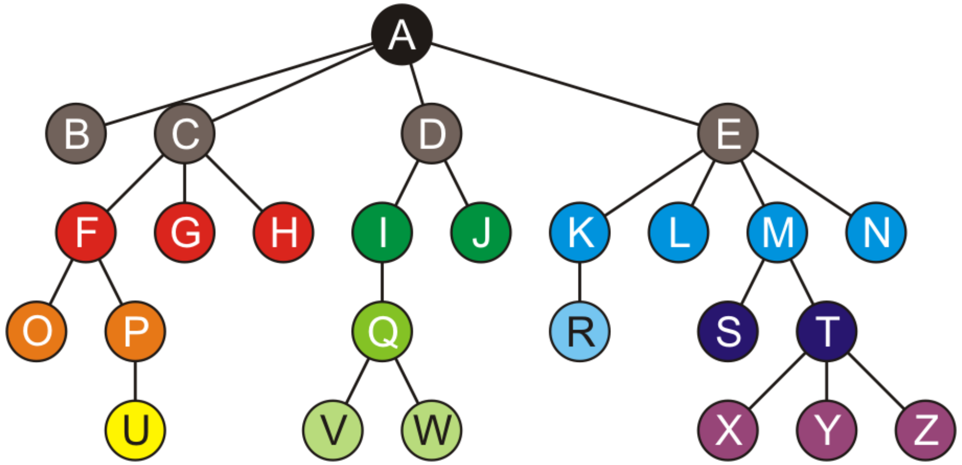


Example

I, the first child of D, is the left child of D

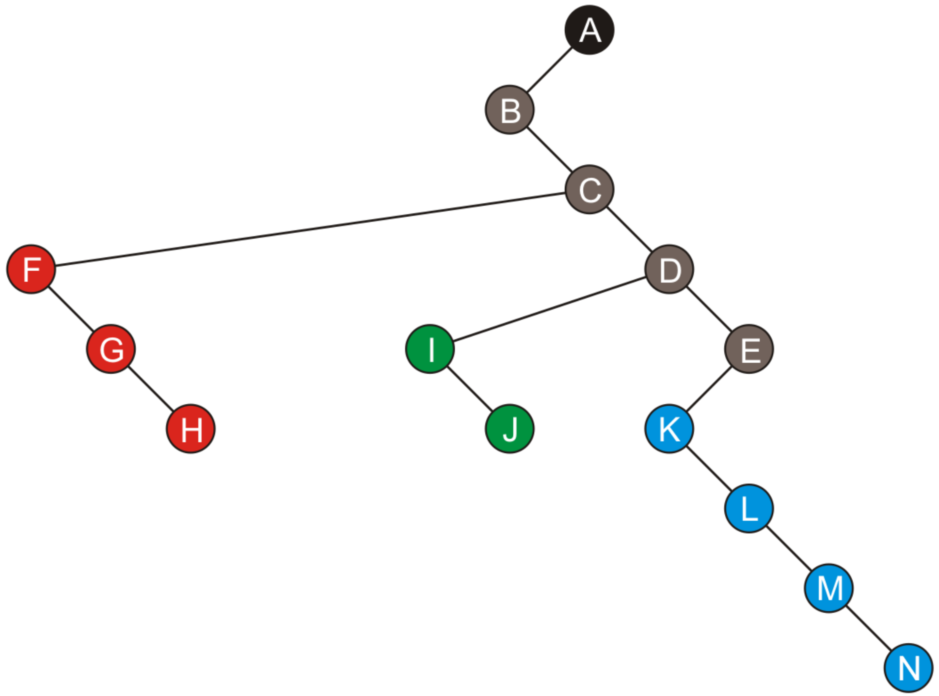
Its sibling J is the right sub-tree of I

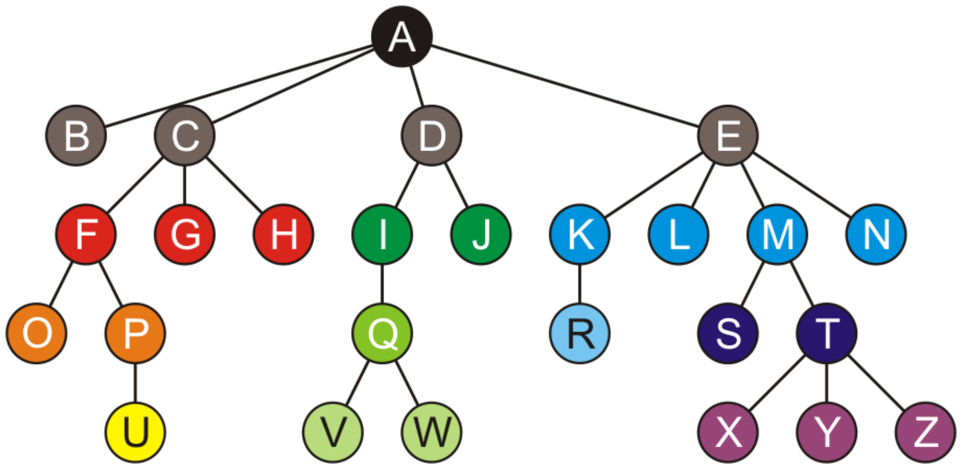




Example

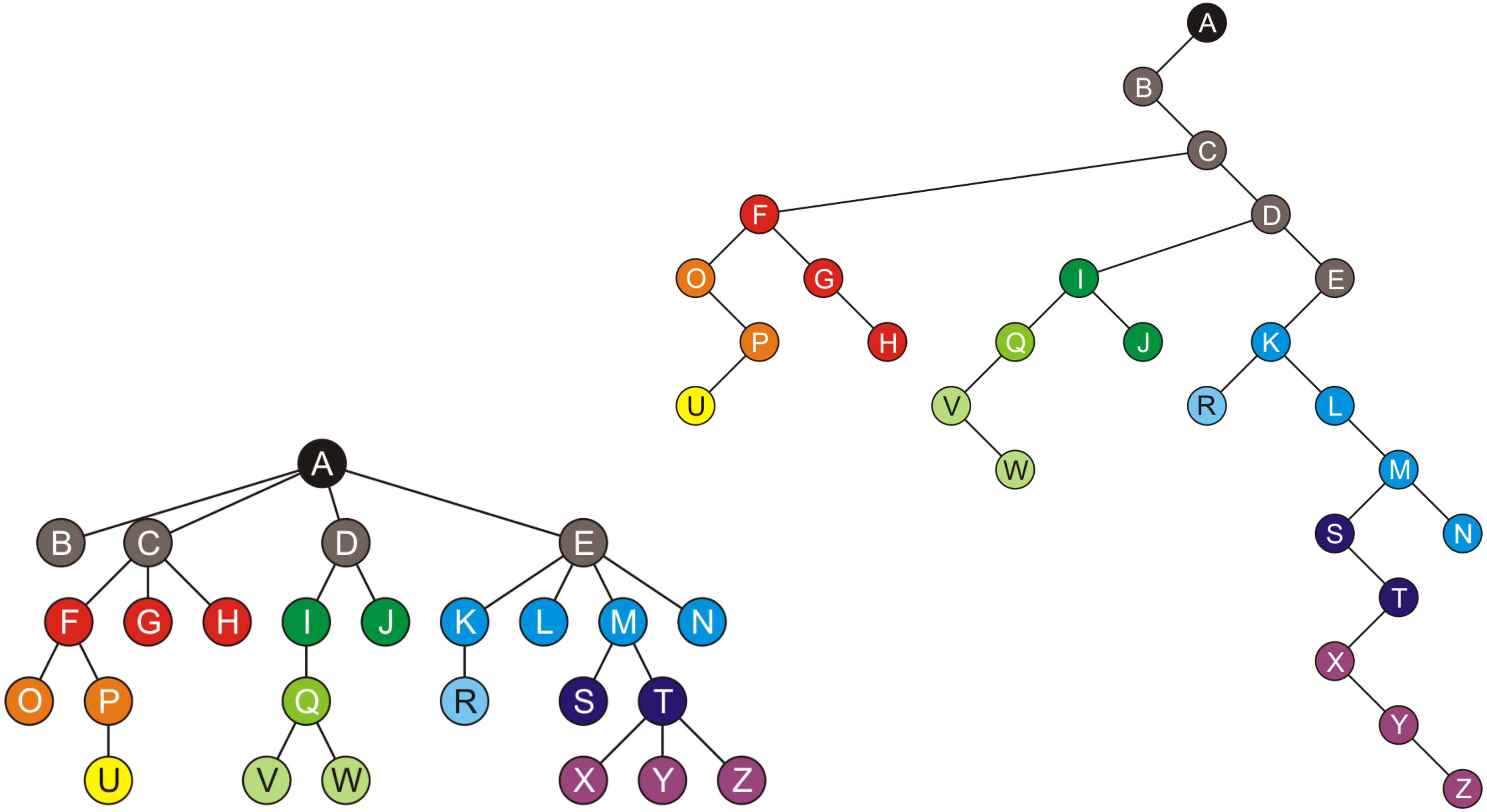
Similarly, the four children of E start with K forming the left sub-tree of E and its three siblings form a chain along the left sub-trees





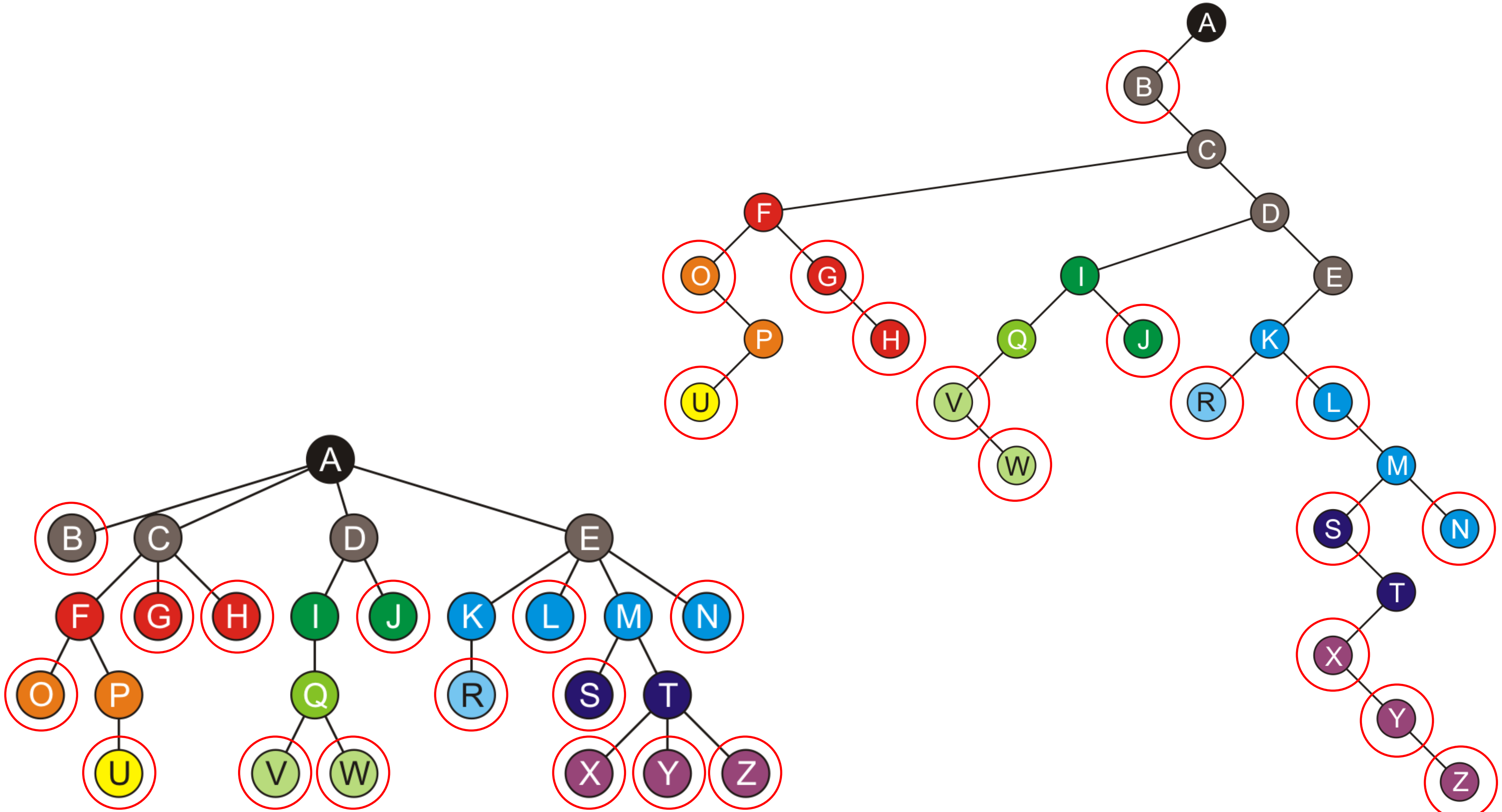
Example

The balance of the nodes in our general tree are shown here



Example

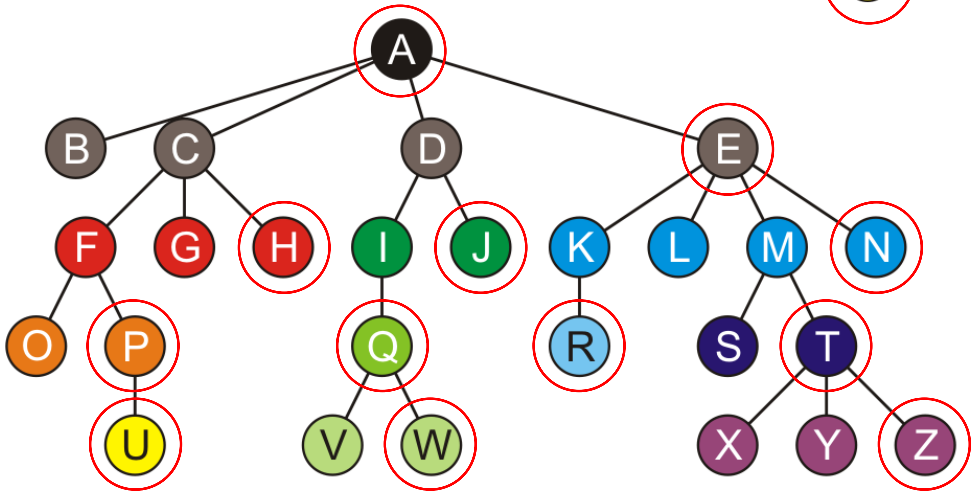
An empty left sub-tree indicates no children

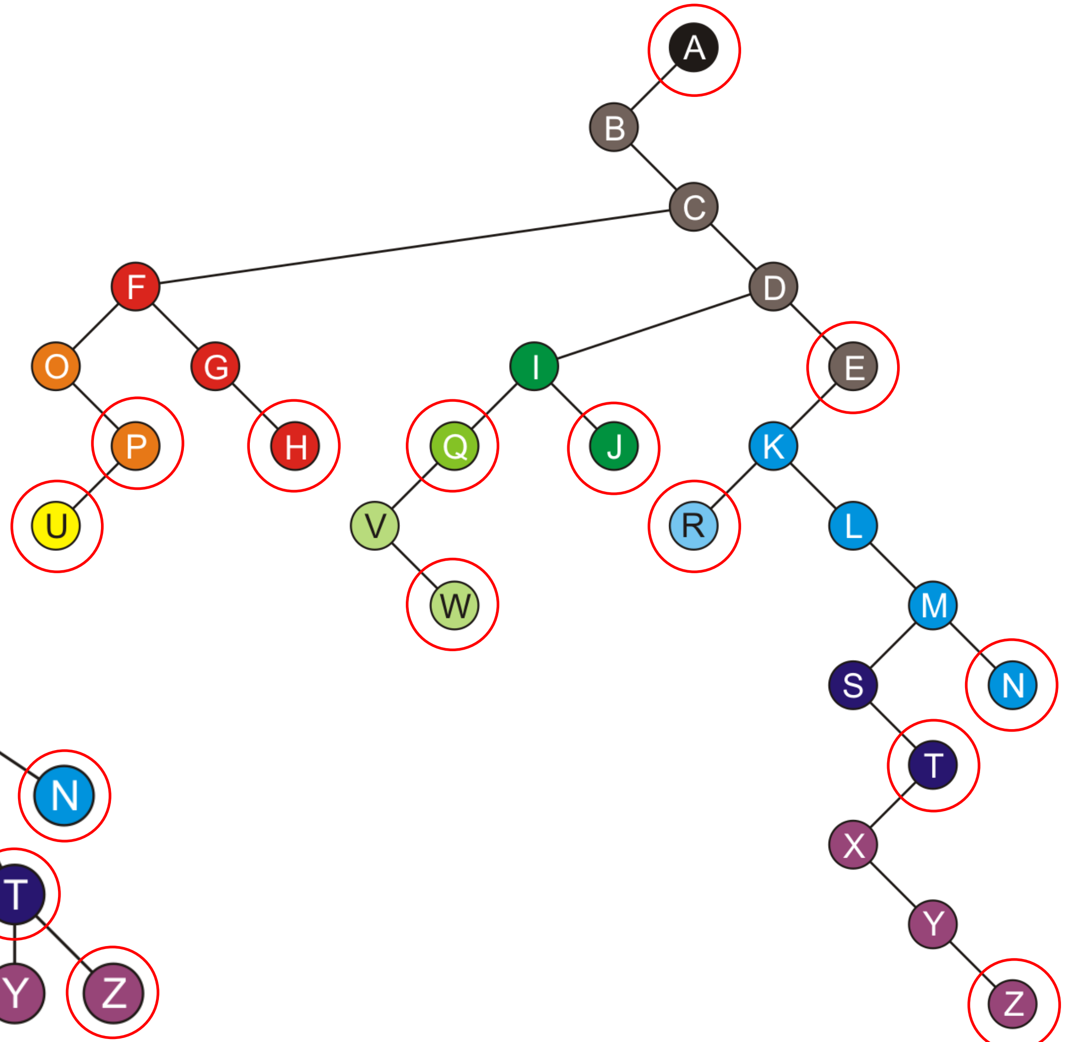


Example

An empty right sub-tree indicates the node is the last of its siblings

- The root node, has no siblings





Forests

A forest, can be stored in this representation as follows:

- Choose one of the roots of the trees as the root of the binary tree
- Let each subsequent root of a tree be a right child of the previous root
- This is the binary-tree representation of this forest
- Think of the roots as siblings of each other