ECE430.217 Data Structures

# Parental trees

Weiss Book Chapter 4.1

**Byoungyoung Lee**
[https://compsec.snu.ac.kr](https://compsec.snu.ac.kr)
**byoungyoung@snu.ac.kr**

Lecture slides were prepared based on Douglas W. Harder's notes (**dwharder@alumni.uwaterloo.ca).**

## Outline

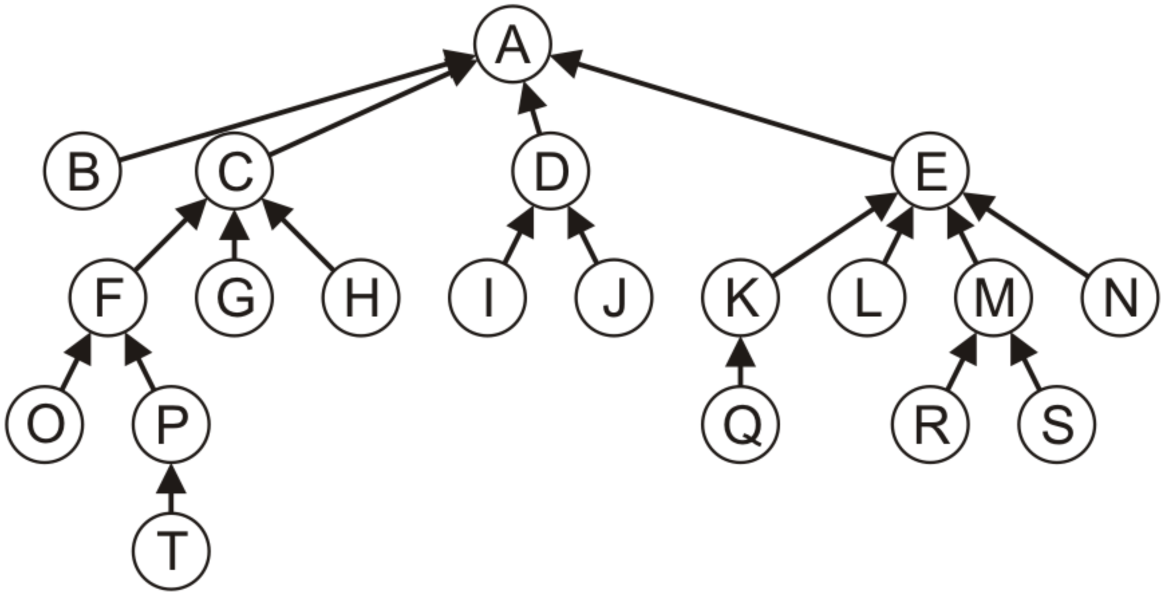
In this topic, we will

- Define a parental tree
- Consider an efficient implementation
- Converting a parental tree to a node-based tree

## Definition

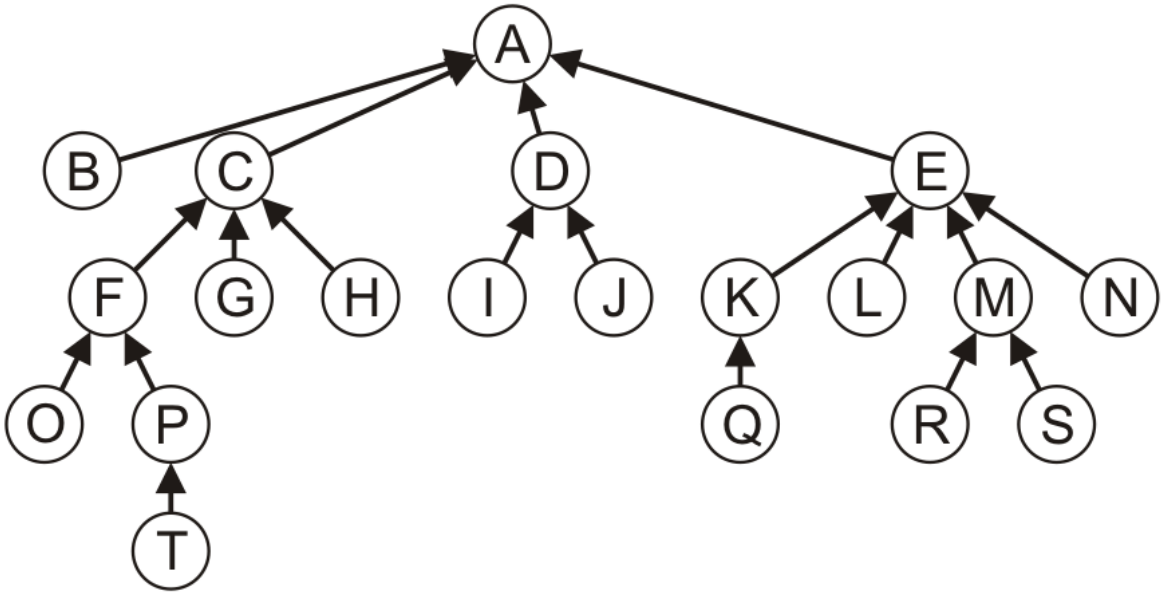
A *parental tree* is a tree where each node only keeps a reference to its parent node

- Note, this definition is restricted to this course
- Also known as a *parent-pointer tree*



## Definition

This requires significantly less memory than our general tree structure, as no data structure is required to track the children



## Implementation

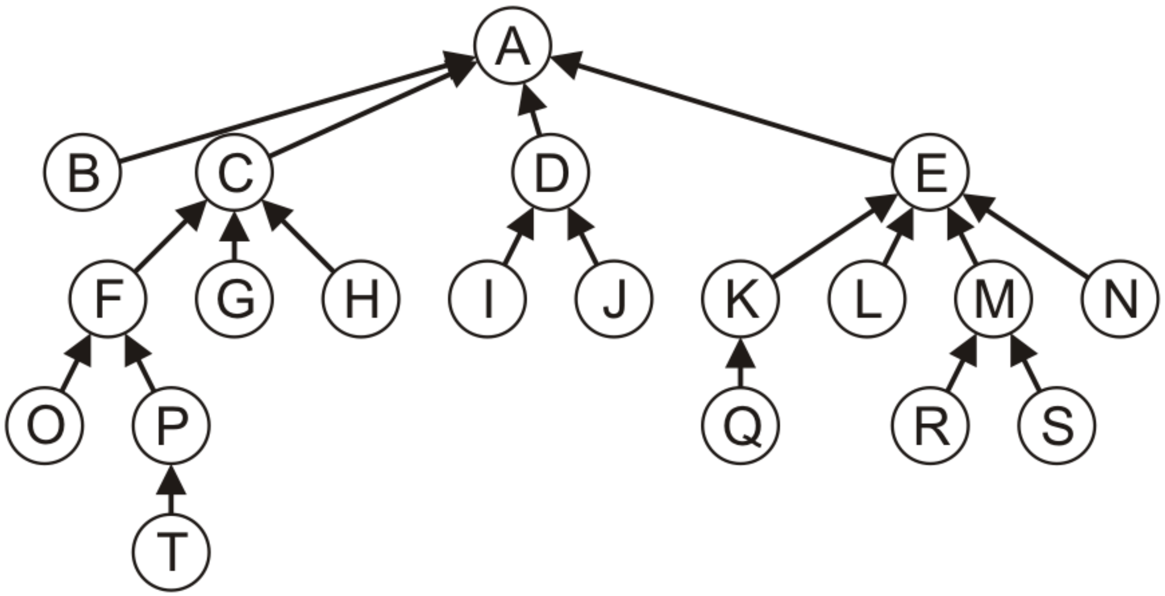
A naïve implementation may also be node based:

```
template <typename Type>
class Parental_tree {
    private:
        Type element;
        Parental_tree *parent;
    public:
        // ...
};
```

## Implementation

Instead, generate an array of size $n$ and associate each entry with a node in the tree

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T |

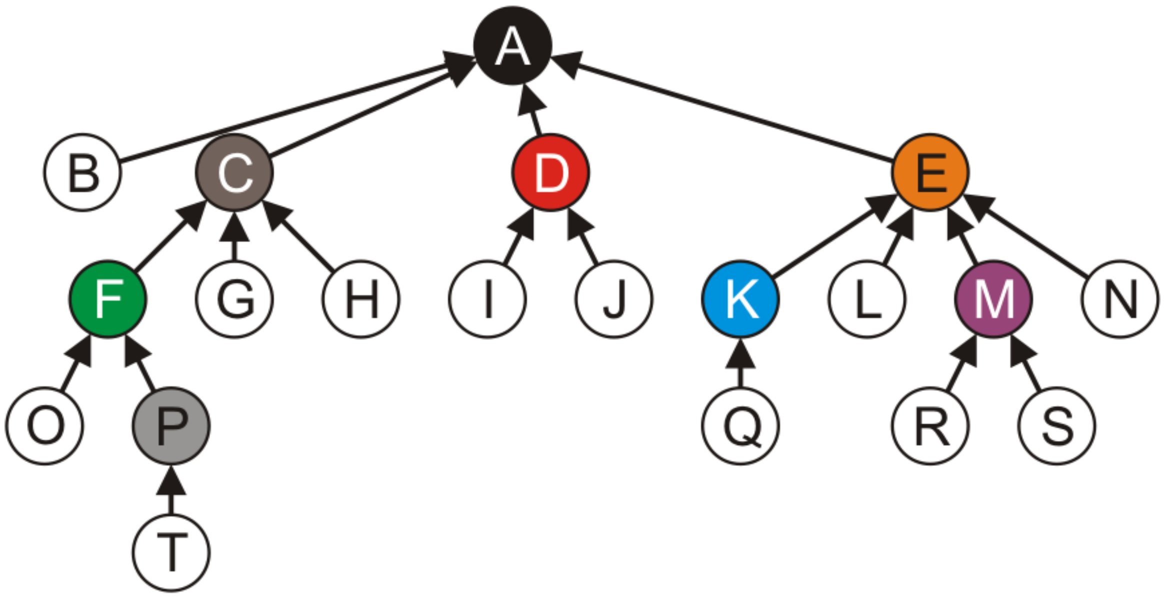


# Implementation

Store the index of the parent in each node

– The root node, wherever it is, points to itself

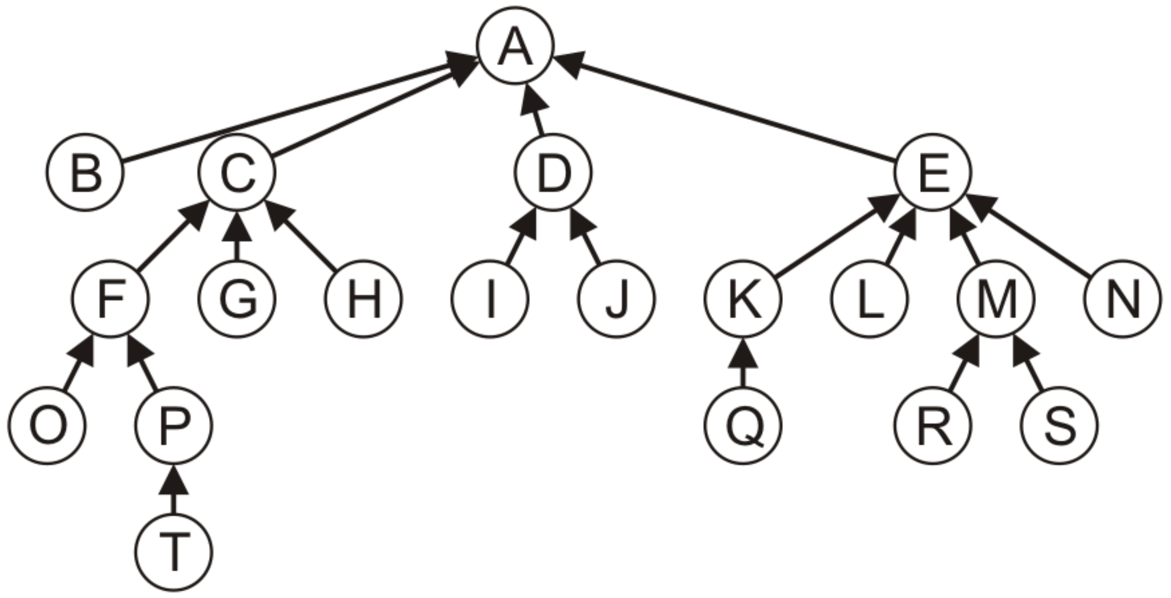
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 2 | 2 | 2 | 3 | 3 | 4 | 4 | 4 | 4 | 5 | 5 | 10 | 12 | 12 | 15 |
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T |



## Implementation

The memory requirements are quite small relative to our node-based implementation

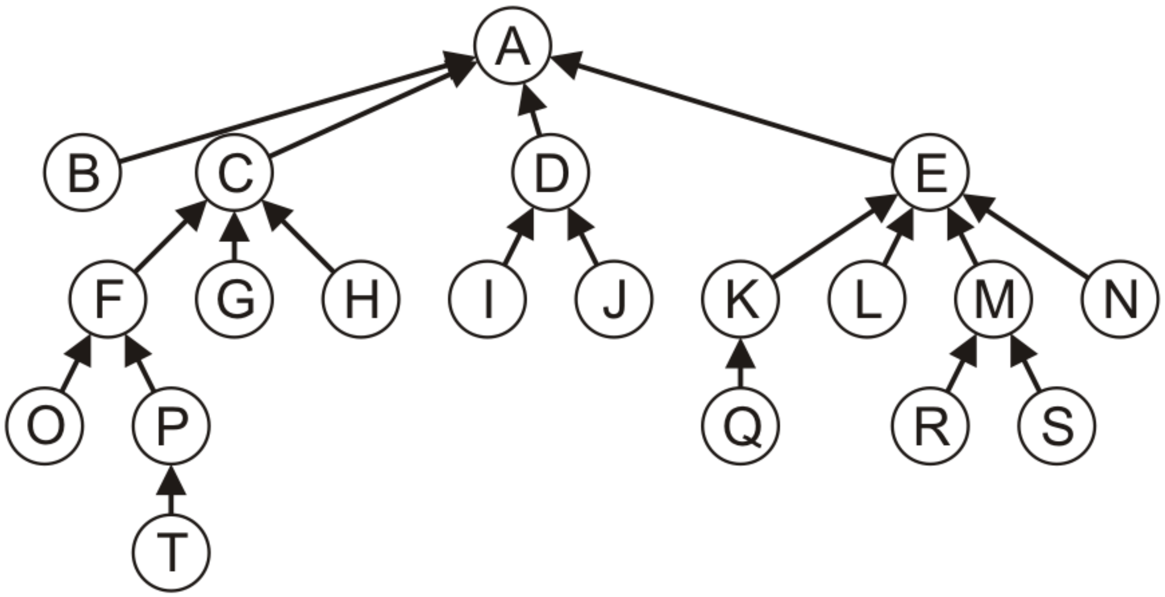
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 2 | 2 | 2 | 3 | 3 | 4 | 4 | 4 | 4 | 5 | 5 | 10 | 12 | 12 | 15 |
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T |



## Implementation

In a tree, only one node will point to itself

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 2 | 2 | 2 | 3 | 3 | 4 | 4 | 4 | 4 | 5 | 5 | 10 | 12 | 12 | 15 |
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T |



## Converting to a `Simple_tree` structure

Converting the array-based parental tree structure back into a node-based general tree structure is relatively straight-forward:

```
int const n = 20;
int parent_array[n] = { 0,  0,  0,  0,  0,  2,  2,  2,  3,  3,
                        4,  4,  4,  4,  5,  5, 10, 12, 12, 15};

Simple_tree<Type> *root_node = nullptr;
Simple_tree<Type> *array = new Simple_tree<Type> *[n];

for ( int i = 0; i < n; ++i ) {
    array[i] = new General_tree<Type>();
}

for ( int i = 0; i < n; ++i ) {
    if ( parent_array[i] == i ) {
        root_node = array[i];
    } else {
        array[parent_array[i]]->attach( array[i] );
    }
}
```

## Looking ahead

The parental tree representation is used in numerous places:

- Storing the critical path for the topological sorting of a directed acyclic graph
- Prim’s algorithm: storing a minimum spanning trees of a weighted graph
- Dijkstra’s algorithm: storing the minimum paths in a weighted graph

## Summary

This topic covered

- The definition of a parental tree
- Considered an efficient implementation
- Considered converting back to a `Simple_tree`-based structure
- Considered various uses