Data Structures
Queues and Iterators

CS 225
September 11, 2023

Brad Solomon & G Carl Evans

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
Department of Computer Science

Learning Objectives

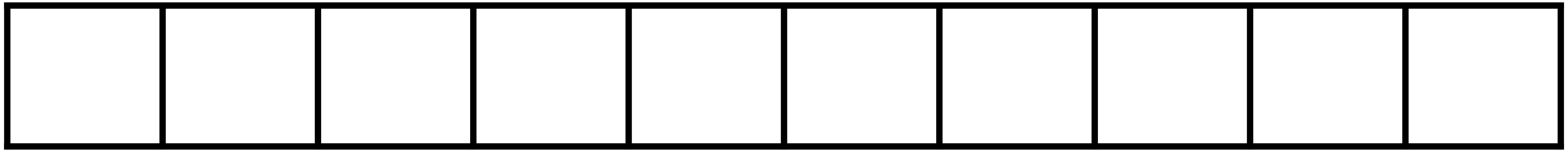
Review the queue data structure

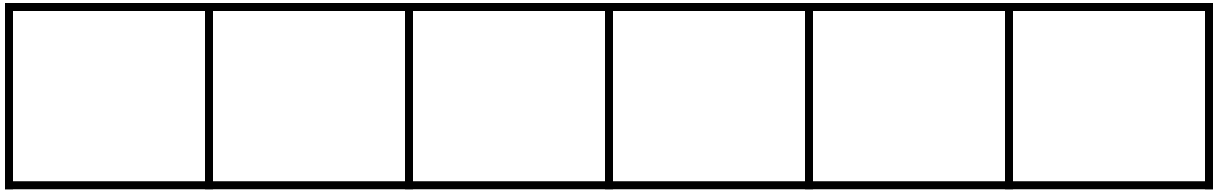
Introduce and explore iterators

Introduce trees

Queue Data Structure

What do we need to track to maintain a queue with an array list?





```
Queue<int> q;
q.enqueue(3);
q.enqueue(8);
q.enqueue(4);
q.dequeue();
q.enqueue(7);
q.dequeue();
q.dequeue();
q.enqueue(2);
q.enqueue(1);
q.enqueue(3);
q.enqueue(5);
q.dequeue();
q.enqueue(9);
```

Size:

Front:

Capacity:

Queue Data Structure: Resizing

		m	o	n	

```
Queue<char> q;
...
q.enqueue(m);
q.enqueue(o);
q.enqueue(n);
...
q.enqueue(d);
q.enqueue(a);
q.enqueue(y);
q.enqueue(i);
q.enqueue(s);
```

Queue ADT

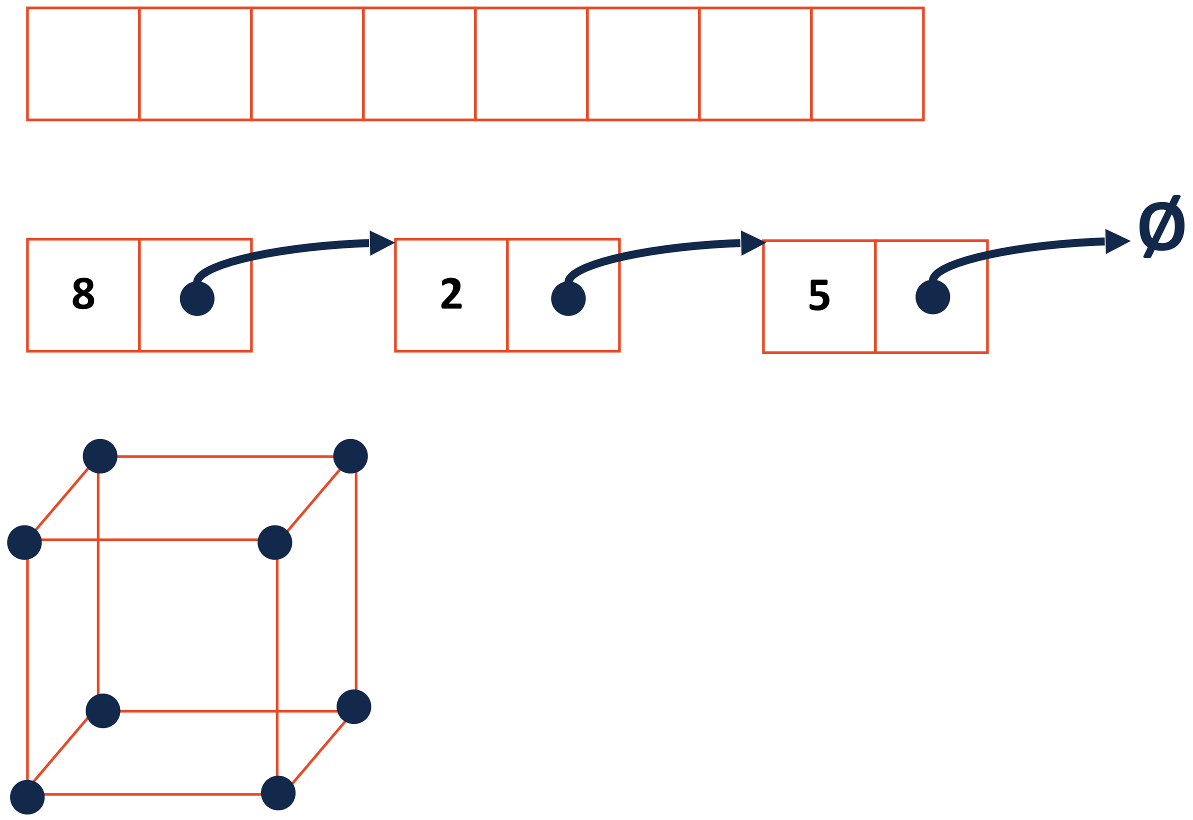
- [Order]:

- [Implementation]:

- [Runtime]:

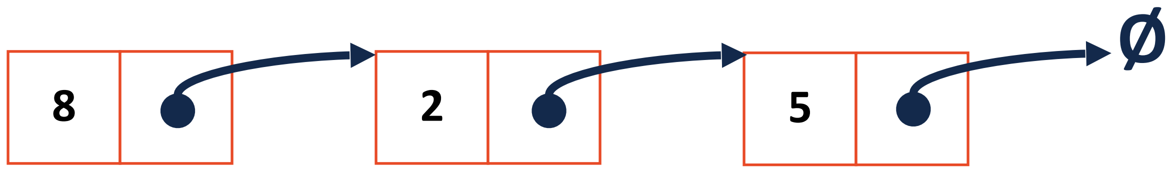
Iterators

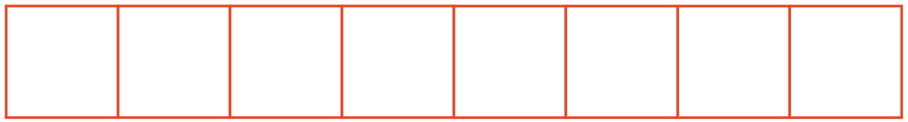
We want to be able to loop through all elements for any underlying implementation in a systematic way

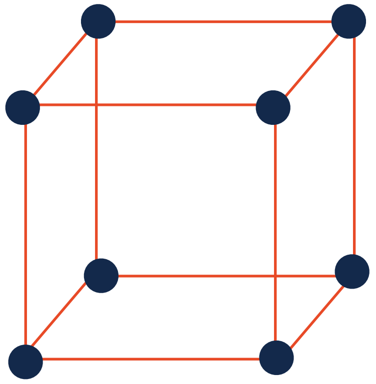


Iterators

We want to be able to loop through all elements for any underlying implementation in a systematic way







Cur. Location	Cur. Data	Next
`ListNode *`		
`index`		
`(x, y, z)`		

Iterators

For a class to implement an iterator, it needs two functions:

```
Iterator begin()
```

```
Iterator end()
```

Iterators

The actual iterator is defined as a class **inside** the outer class:

1. It must be of base class **`std::iterator`**

2. It must implement at least the following operations:

```
Iterator& operator ++()

const T & operator *()

bool operator !=(const Iterator &)
```

Iterators

Future assignments will have you write custom iterators:

```
template <class T>
class List {

    class ListIterator : public std::iterator<std::bidirectional_iterator_tag, T> {
      public:

        ListIterator& operator++();

        ListIterator& operator--()

        bool operator!=(const ListIterator& rhs);

        const T& operator*();
    };

    ListIterator begin() const;

    ListIterator end() const;
};
```

stlList.cpp

```
#include <list>
#include <string>
#include <iostream>

struct Animal {
  std::string name, food;
  bool big;
  Animal(std::string name = "blob", std::string food = "you", bool big = true) :
    name(name), food(food), big(big) { /* nothing */ }
};

int main() {
  Animal g("giraffe", "leaves", true), p("penguin", "fish", false), b("bear");
  std::vector<Animal> zoo;

  zoo.push_back(g);
  zoo.push_back(p);   // std::vector’s insertAtEnd
  zoo.push_back(b);

  for ( std::vector<Animal>::iterator it = zoo.begin(); it != zoo.end(); ++it ) {
    std::cout << (*it).name << " " << (*it).food << std::endl;
  }

  return 0;
}
```

```
std::vector<Animal> zoo;


/* Full text snippet */

  for ( std::vector<Animal>::iterator it = zoo.begin(); it != zoo.end(); ++it ) {
    std::cout << (*it).name << " " << (*it).food << std::endl;
  }


/* Auto Snippet */

  for ( auto it = zoo.begin(); it != zoo.end; ++it ) {
    std::cout << animal.name << " " << animal.food << std::endl;
  }

/* For Each Snippet */

  for ( const Animal & animal : zoo ) {
    std::cout << animal.name << " " << animal.food << std::endl;
  }
```

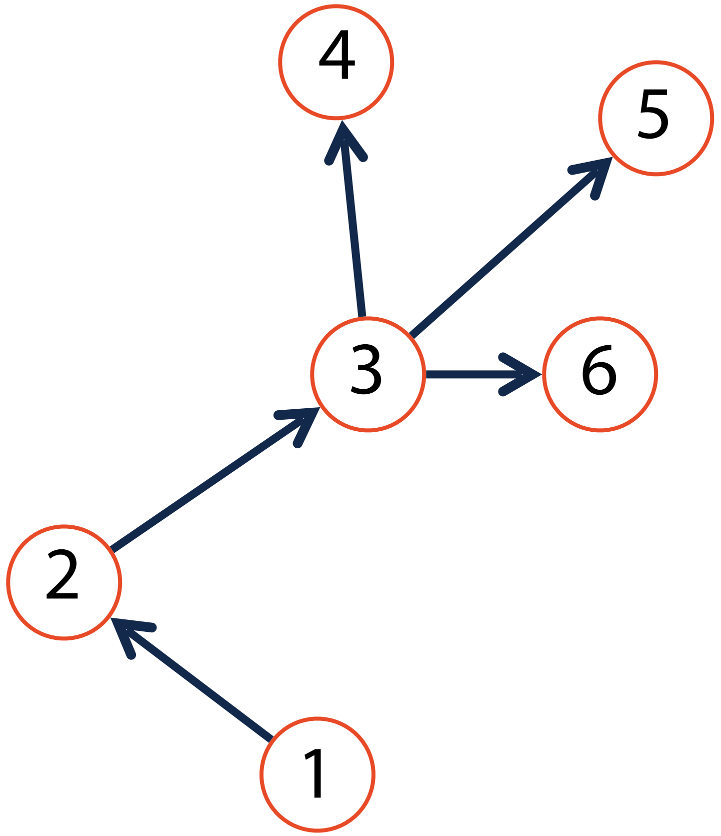
Trees

A non-linear data structure defined recursively as a collection of nodes where each node contains a value and zero or more connected nodes.

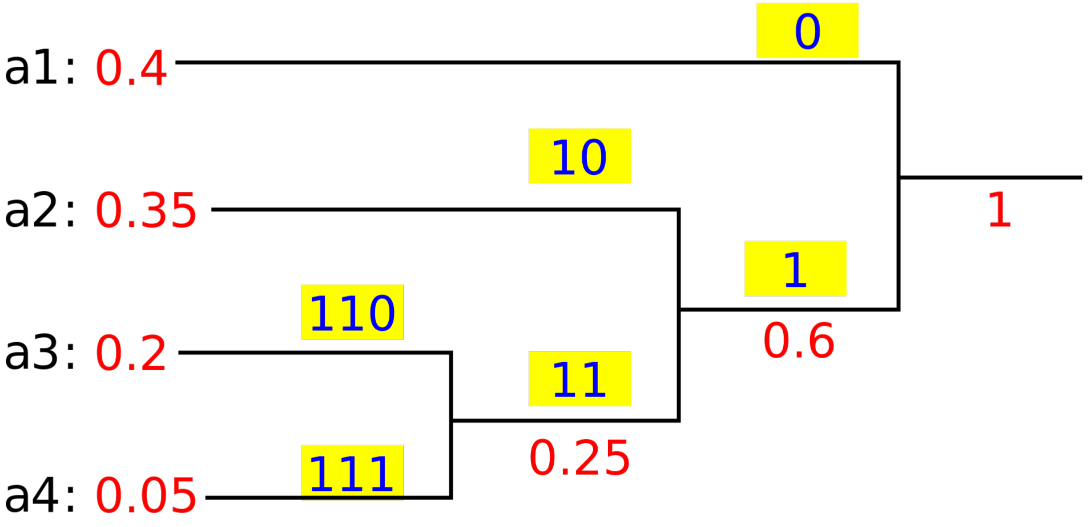
[In CS 225] a tree is also:

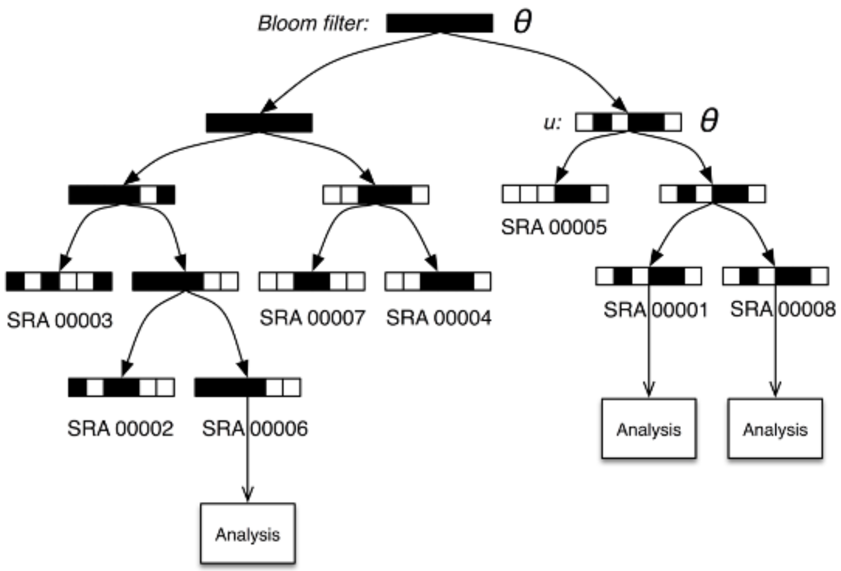
1)

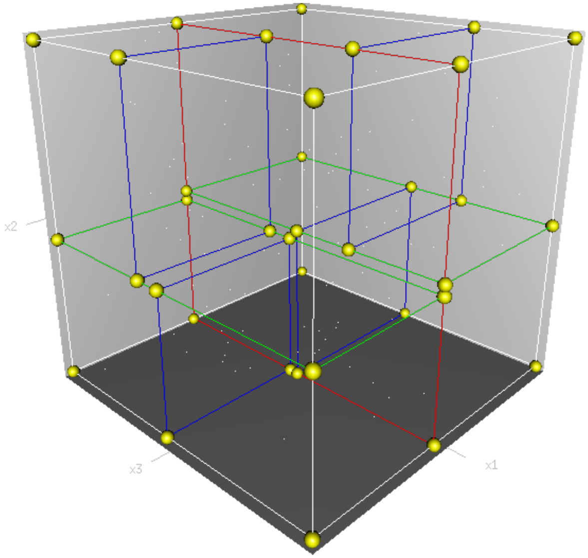
2)

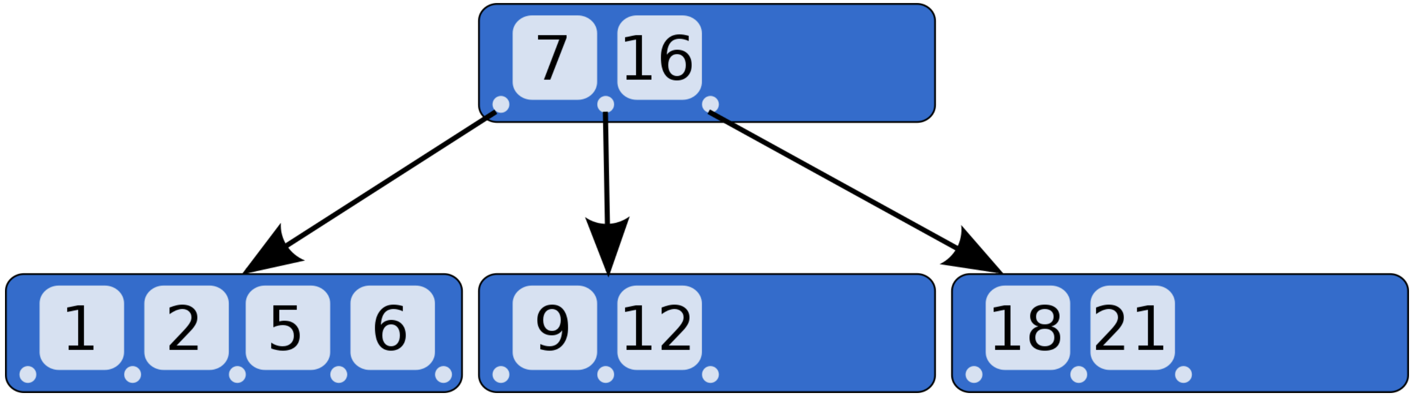


There are many *types* of trees

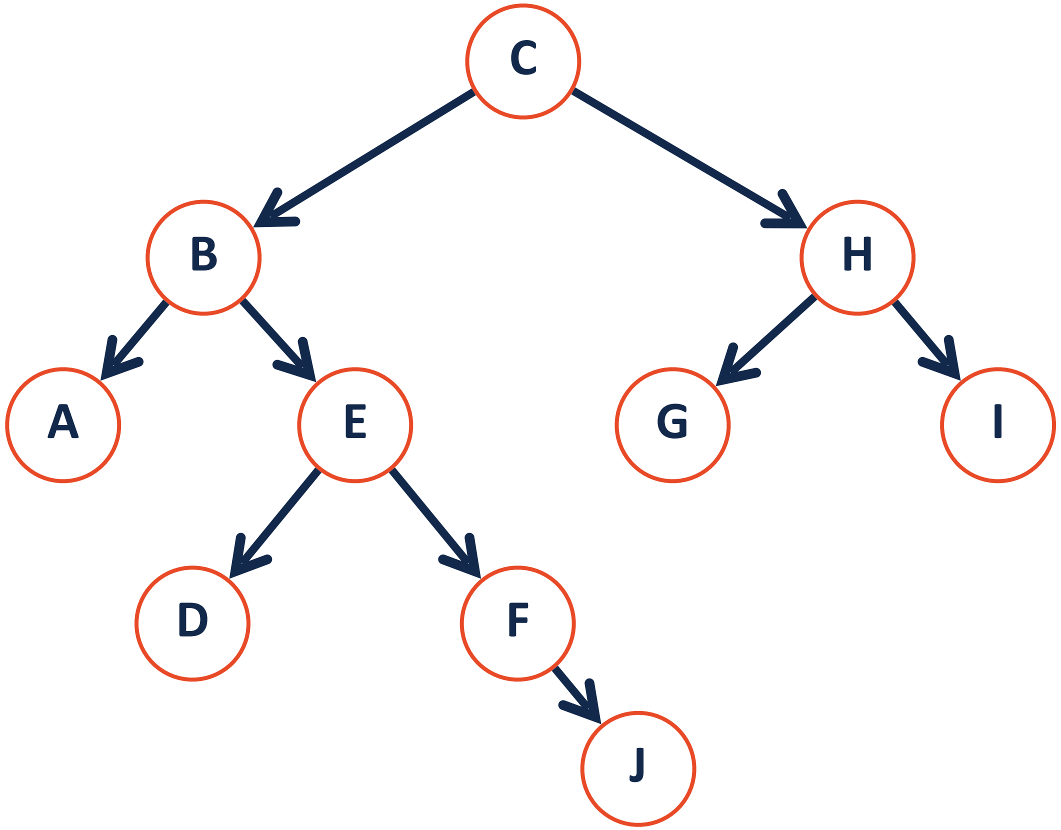








Tree Terminology

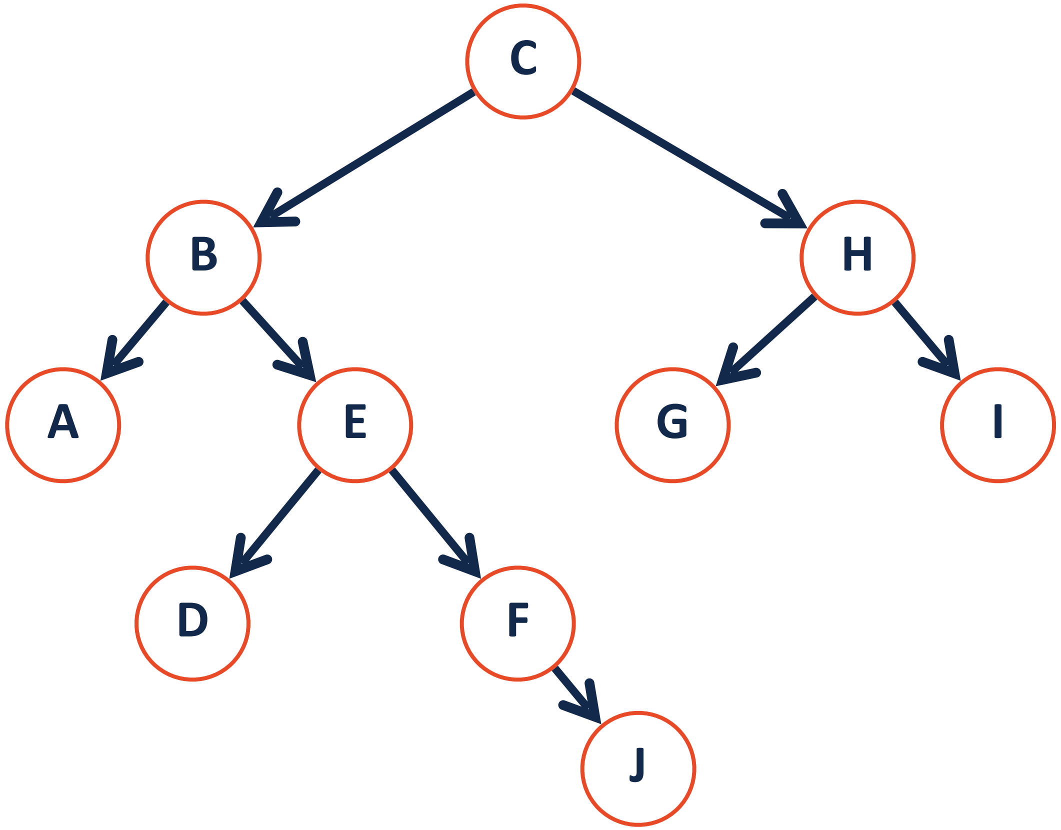


Node: The vertex of a tree

Edge: The connecting path between nodes

Path: A list of the edges (or nodes) traversed to go from node `start` to node `end`

Tree Terminology



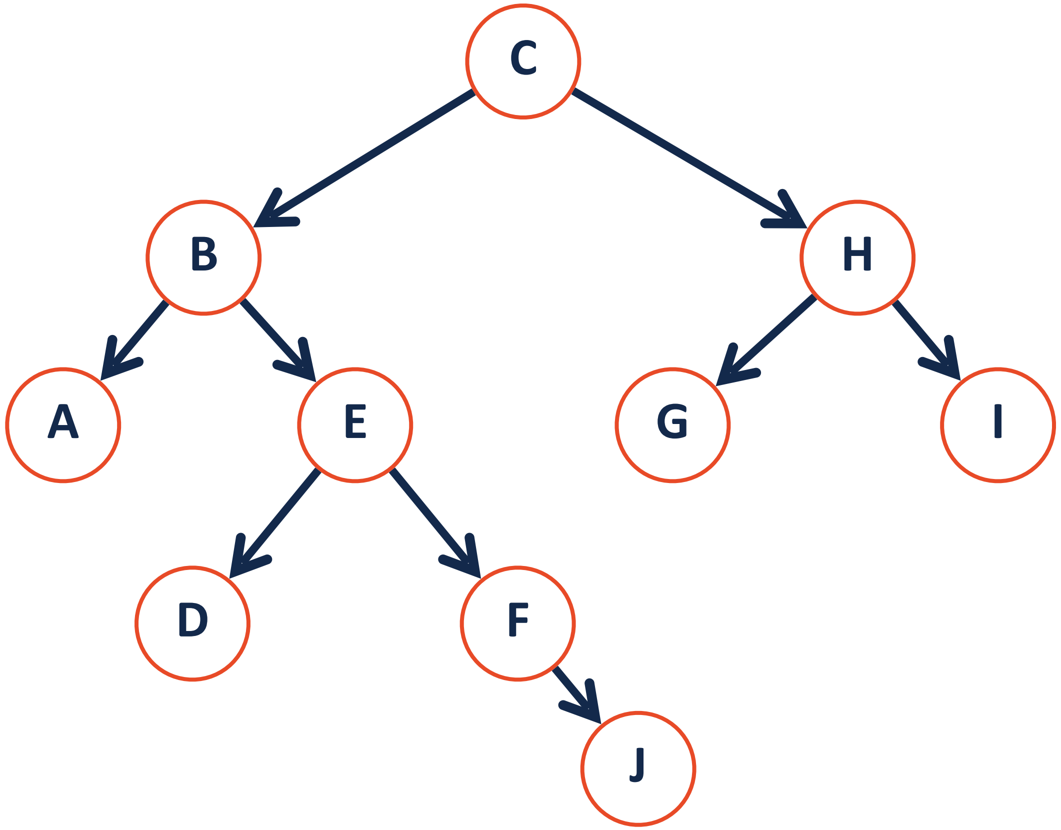
Parent: The precursor node to the current node is the 'parent'

Child: The nodes linked by the current node are it's 'children'

Neighbor: Parent or child

Degree: The number of children for a given node

Tree Terminology



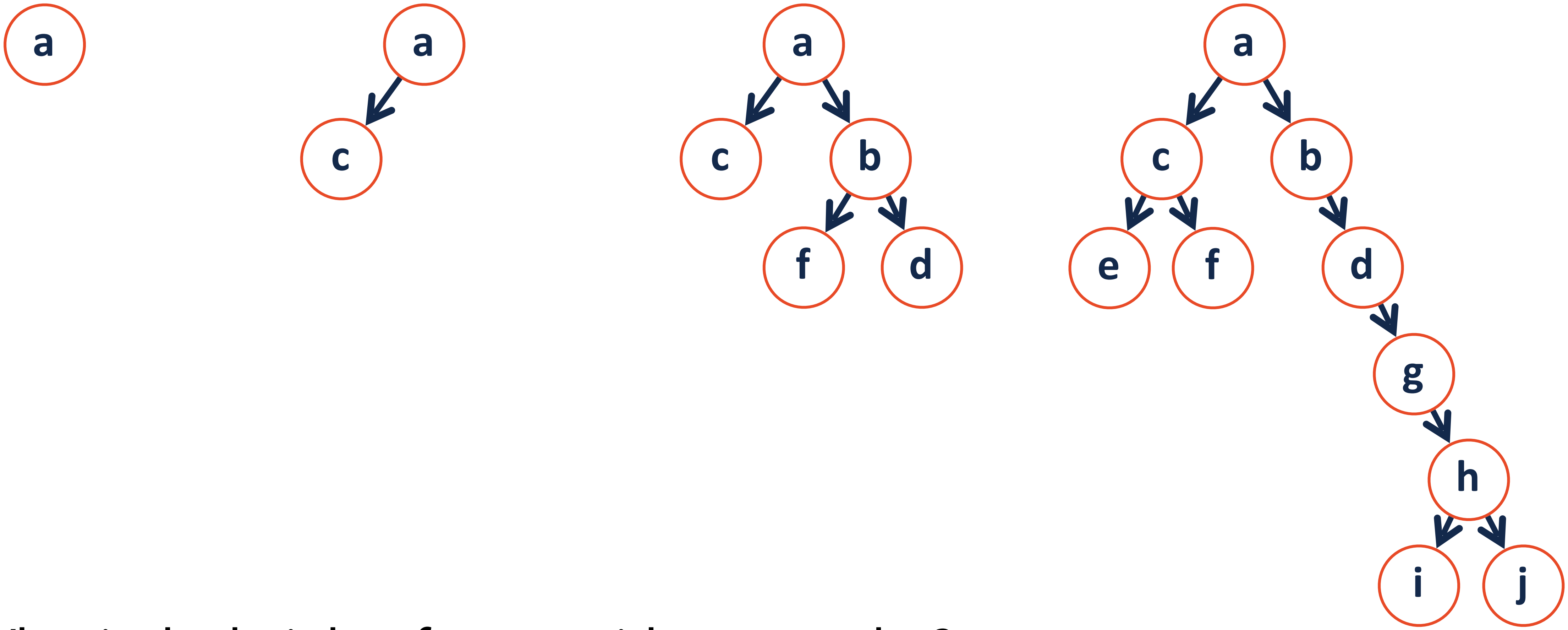
Root: The start of a tree (the only node with no parent).

Leaf: The terminating nodes of a tree (have no children)

Internal: A node with at least one child

Tree Terminology

Height: the length of the longest path from the root to a leaf



What is the height of a tree with ***zero*** nodes?

Tree Height

`height(T) =`

Base Case:

Recursive Step:

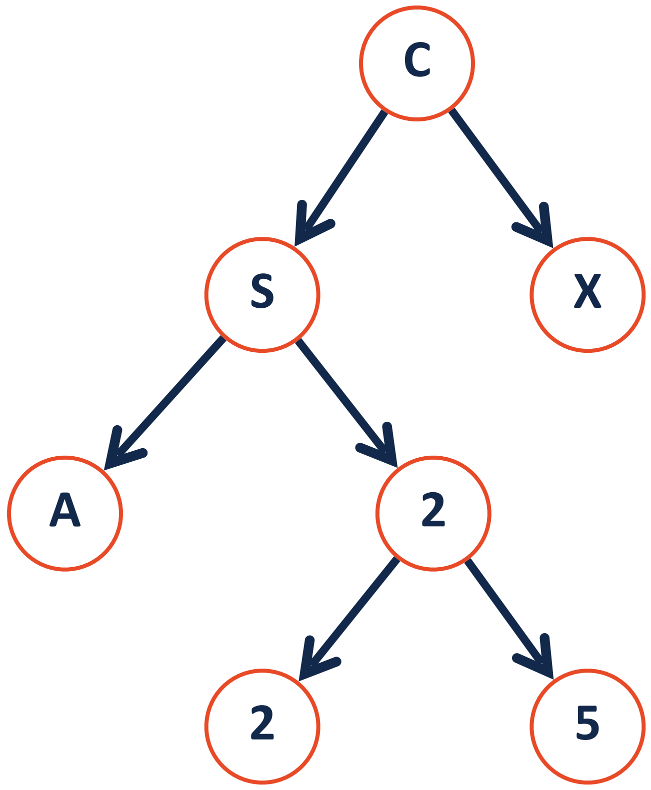
Combining:

Binary Tree

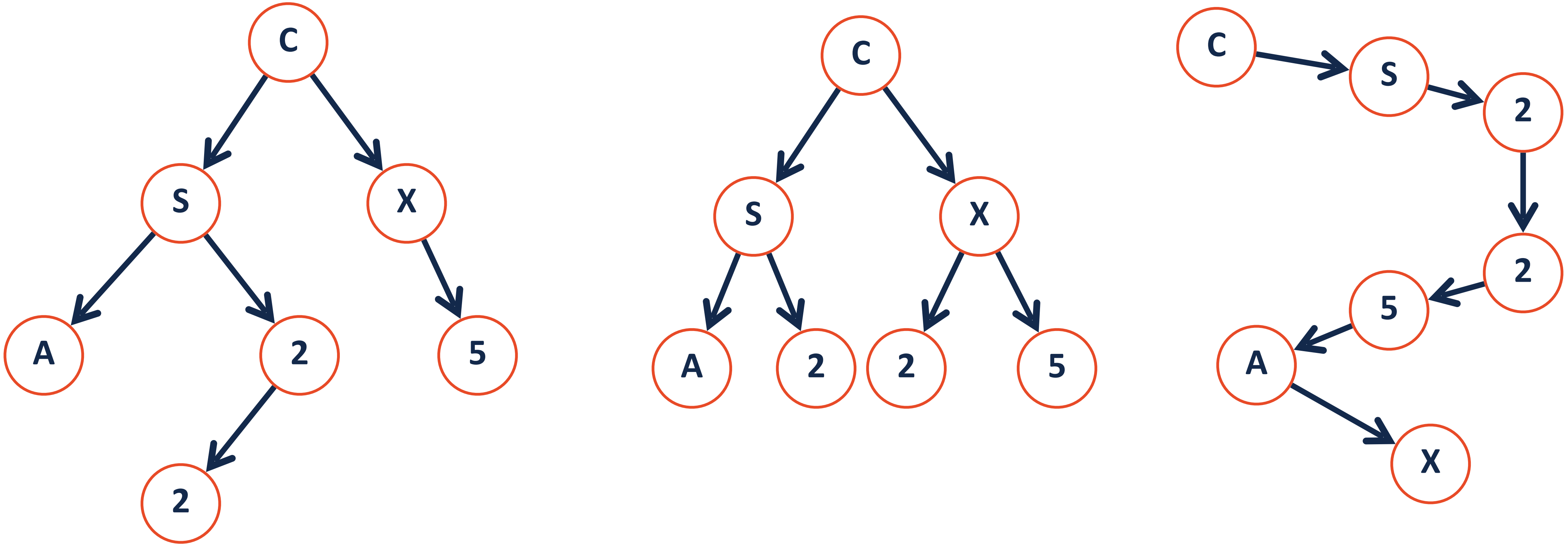
A **binary tree** is a tree T such that:

1.

2.



Which of the following are binary trees?



Binary Tree

Lets define additional terminology for different **types** of binary trees!

1.

2.

3.

Binary Tree: full

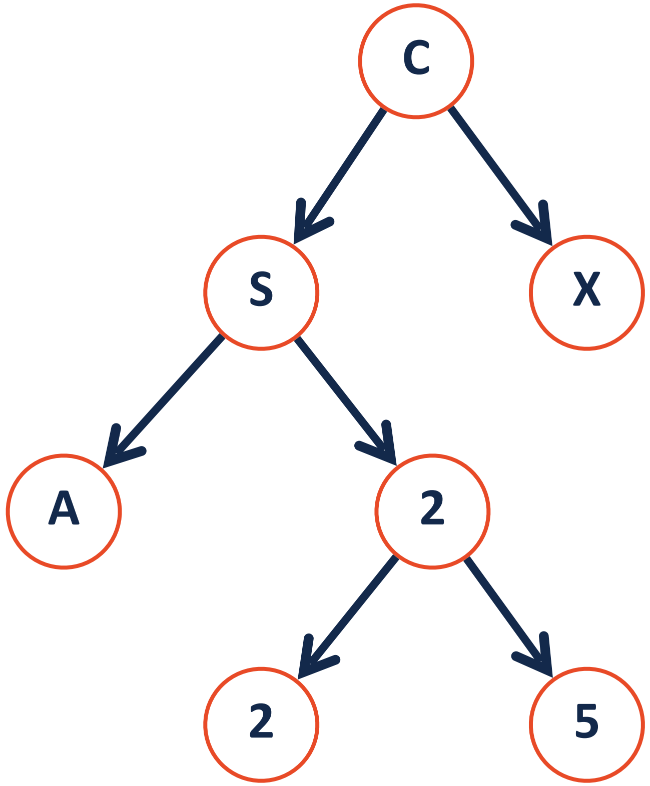
A **full tree** is a binary tree where every node has either 0 or 2 children

A tree **F** is **full** if and only if:

1.

2.

3.



Binary Tree: perfect

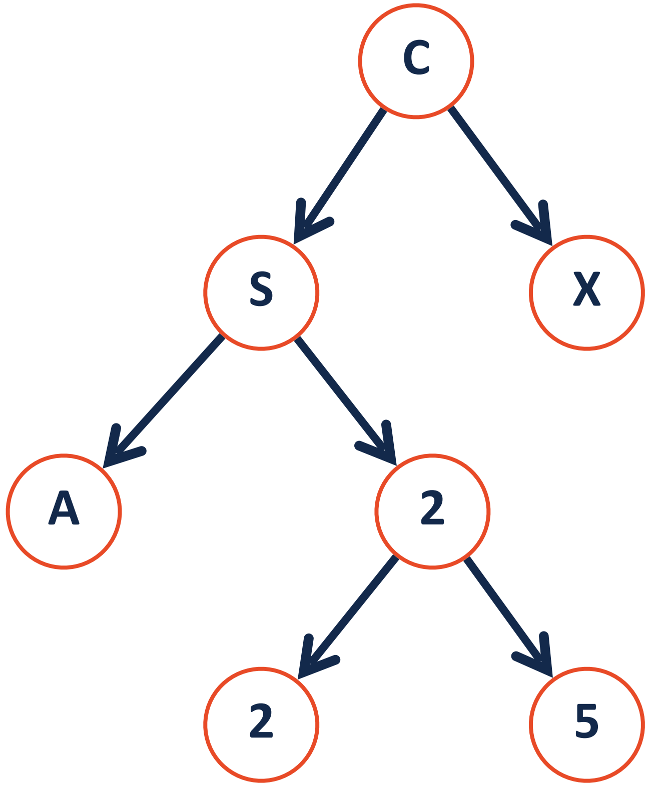
A **perfect tree** is a binary tree where…

Every internal node has 2 children and all leaves are at the same level.

A tree **P** is **perfect** if and only if:

1.

2.



Binary Tree: complete

A **complete tree** is a B.T. where…

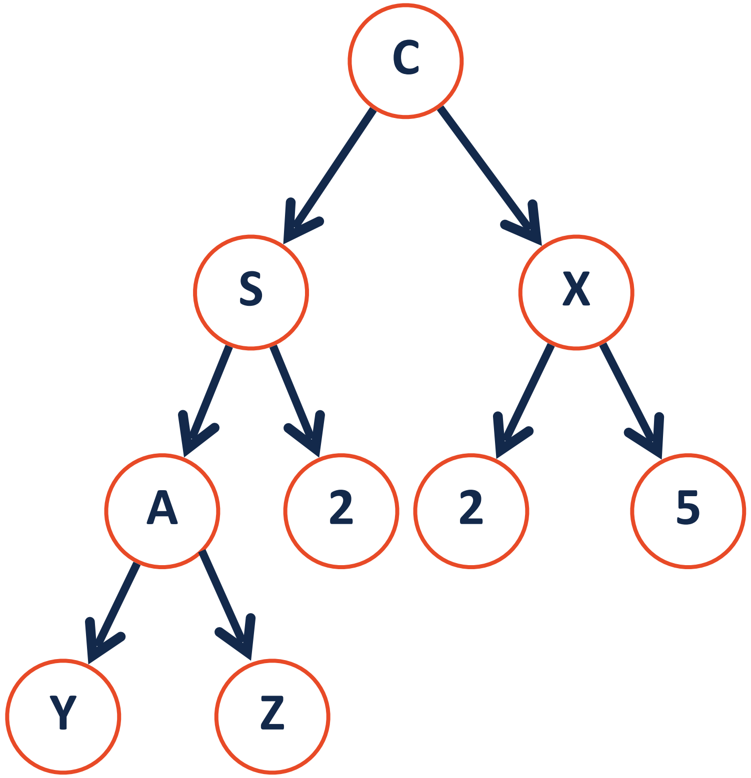
All levels are completely filled except the last (which is pushed to left)

A tree **C** is **complete** if and only if:

1.

2.

3.



Binary Tree

Why do we care?

1. Terminology instantly defines a particular tree structure

2. Understanding how to think ‘recursively’ is very important.

Binary Tree: Thinking with Types

Is every **full** tree **complete**?

Is every **complete** tree **full**?

For next time: Tree ADT and BinaryTree implementation