CS 106X, Lecture 18
Arrays and Trees

reading:

Programming Abstractions in C++, Chapters 11.3, 16.1

Plan For Today

- Arrays
- Announcements
- Trees

Learning Goals

- Understand the difference between Vectors and arrays
- Understand how trees use pointers to represent data in useful ways

Plan For Today

- Arrays
- Announcements
- Trees

Arrays

2	12	15	6	-1	4

(on the stack)

Arrays

2	12	15	6	-1	4

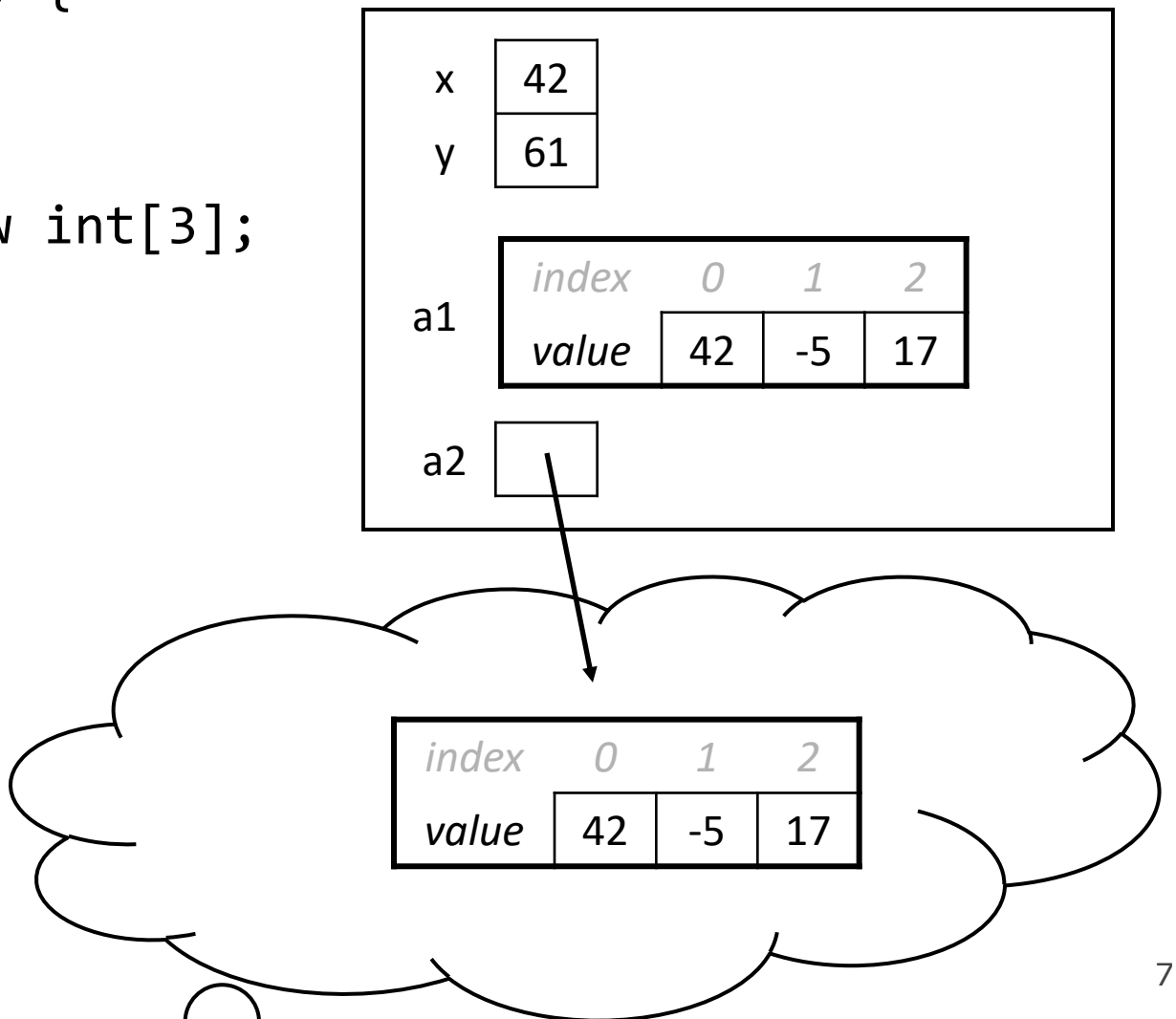
type name type length

```
int *myArray = new int[6];
```

(on the heap)

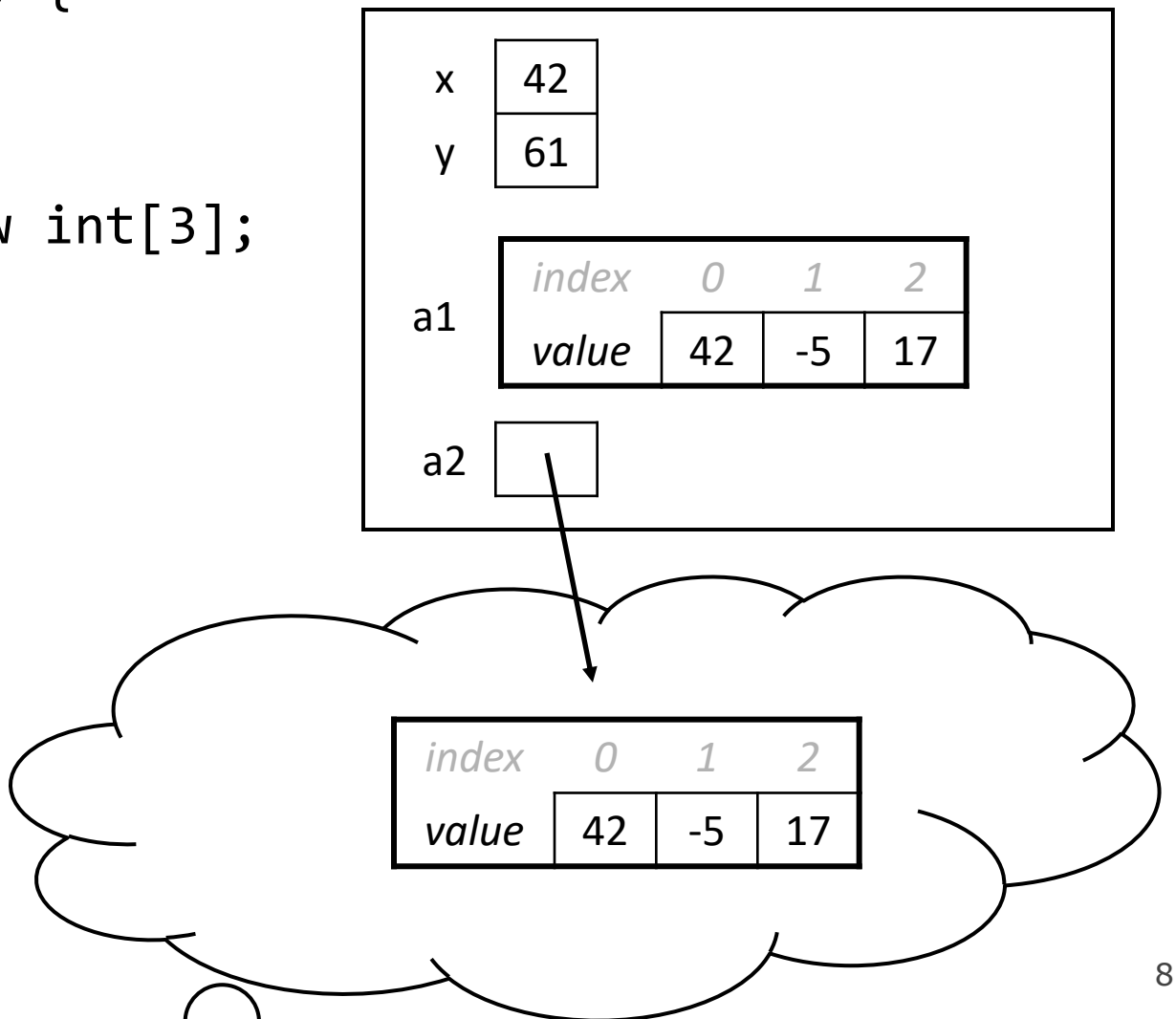
Stack and Heap

```
void makeArrays() {
    int x = 42;
    int y = 61;
    int a1[3];
    int* a2 = new int[3];
    ...

}
```

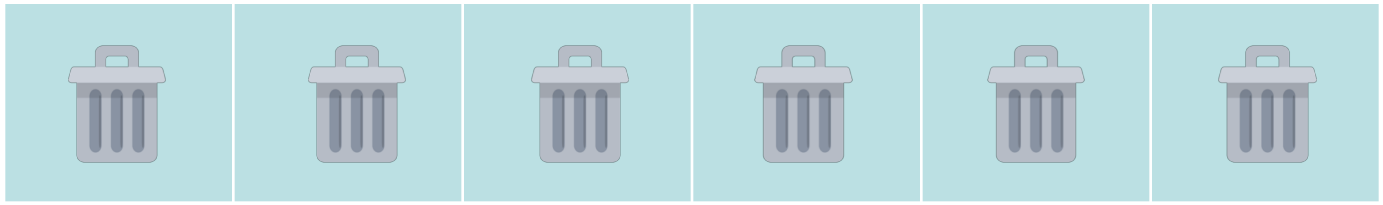


Stack and Heap

```
void makeArrays() {
    int x = 42;
    int y = 61;
    int a1[3];
    int* a2 = new int[3];
    delete[] a2;
    ...
}
```

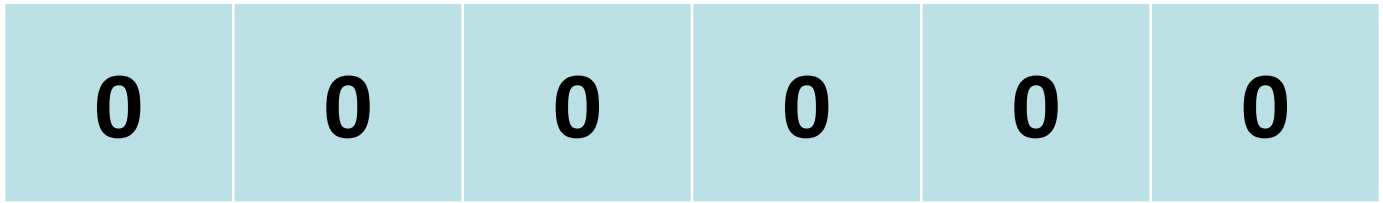


Heap Initialization



```
int *myArray = new int[6];
```

Heap Initialization



slower!

```
int *myArray = new int[6]();
```

Arrays As Parameters

```
void myFunction(int arr[]) {
      ...
}

int main() {
     int myArray[6];
     myFunction(myArray);
     return 0;
}
```

Array Length

- C++ arrays have no members!
  - No **`size`** field
  - No helpful methods like **`indexOf`**

```
int arraySize = 6;
int myArray[arraySize];

for int i = 0; i < arraySize; i++) {
    myArray[i] = ...
}
```

Array Length

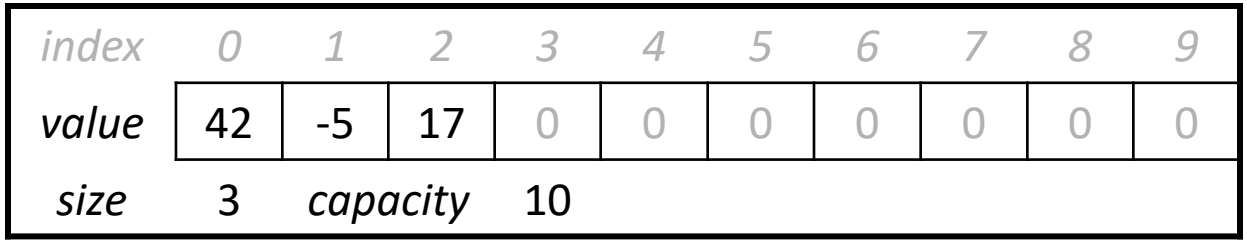
- C++ arrays have no members!
  - No **`size`** field
  - No helpful methods like **`indexOf`**

```
void myFunction(int arr[], int size) {
      ...
}

int main() {
    int arraySize = ...
    int myArray[arraySize];
    myFunction(myArray,  arraySize);
}
```

How Vector/Stack works

- Vectors and Stacks contain internal **arrays** to store elements.
- The array is created with some extra space (it is an "unfilled array"), and is replaced with a larger array when space runs out.

```
Vector<int> v;
v.add(42);
v.add(-5);
v.add(17);
```




```
Stack<int> s;
s.push(42);
s.push(-5);
s.push(17);
```

Outgrowing Our Shell

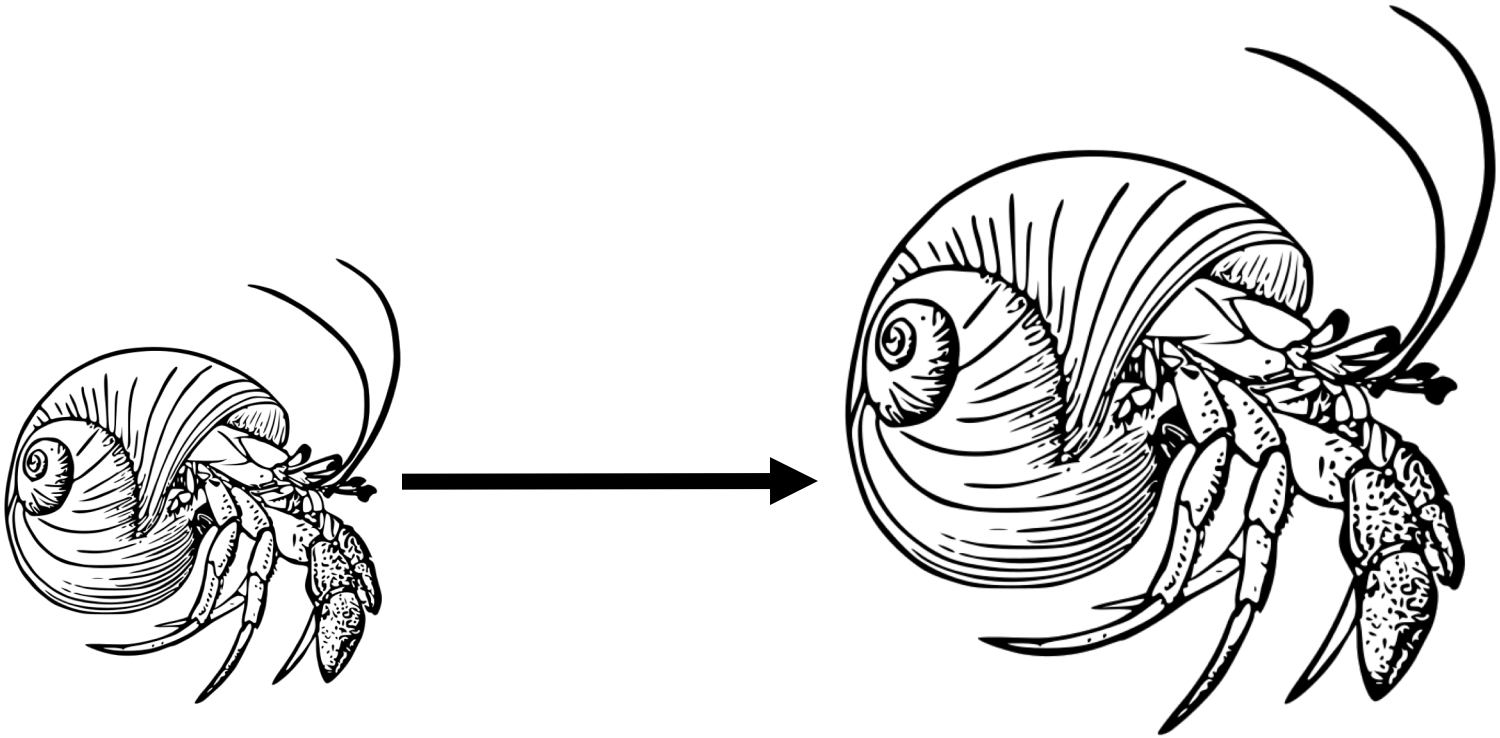


Image from Freshwater and Marine Image Bank: https://commons.wikimedia.org/wiki/File:FMIB_52669_Pagurus_bernhardus,_the_hermit-crab-retouched.svg

Exercise

- Let's implement a **stack** using an unfilled array.
  - We'll call it **`ArrayStack`**. It will be very similar to the C++ `Stack`.
  - its behavior:
    - `push(`***`value`***`)`
    - `pop()`
    - `peek()`
    - `isEmpty()`
    - `toString()`
    - `operator <<`
  - The stack's *size* will be the number of elements added to it so far.
    - The actual array length ("capacity") in the object may be larger.
      We'll start with an array of **length 10** by default.

ArrayStack.h

```
class ArrayStack {
public:
...

private:
      int elements[INITIAL_CAPACITY];
      ...
};
```

ArrayStack.h

```
class ArrayStack {
public:
...

private:
      int *elements;
      ...
};
```

ArrayStack.cpp

```
ArrayStack::ArrayStack() {
    elements = new int[INITIAL_CAPACITY]();
    ...
}
```

Plan For Today

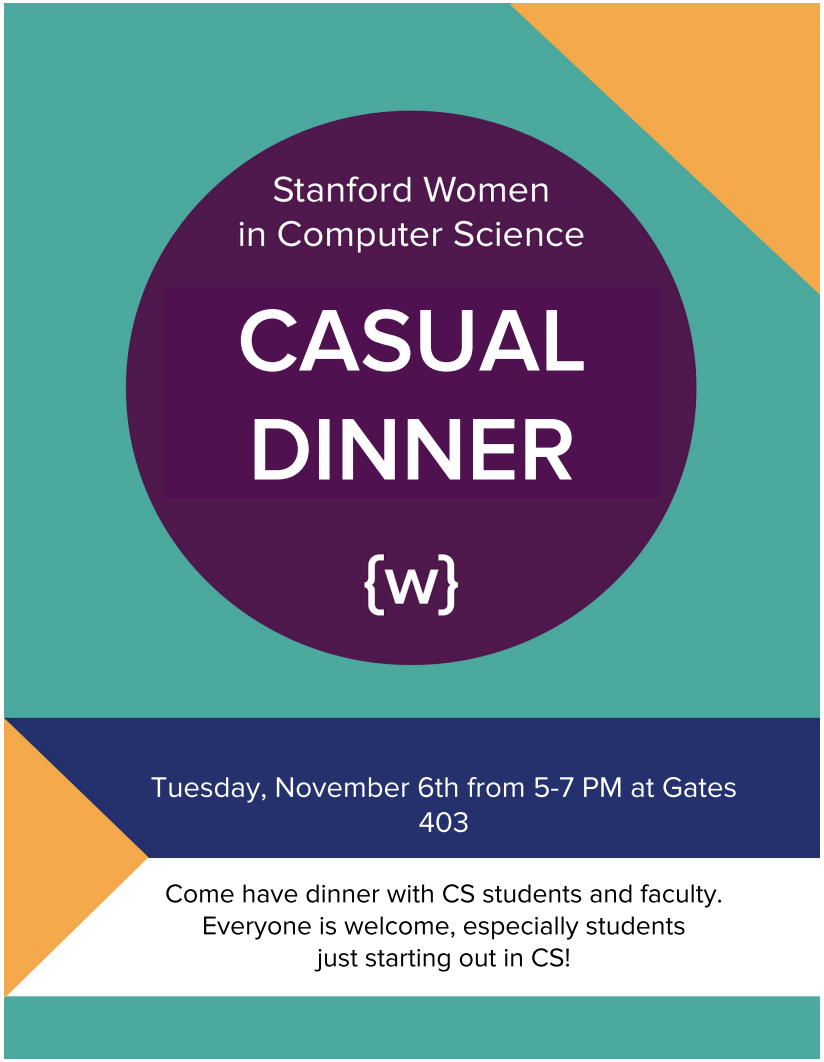
- Arrays
- Announcements
- Trees

Announcements

- Congratulations on finishing the midterm!
- Midterms will be graded and handed back by Monday
- Section Leading Application due tonight at midnight! See cs198.stanford.edu

Announcements

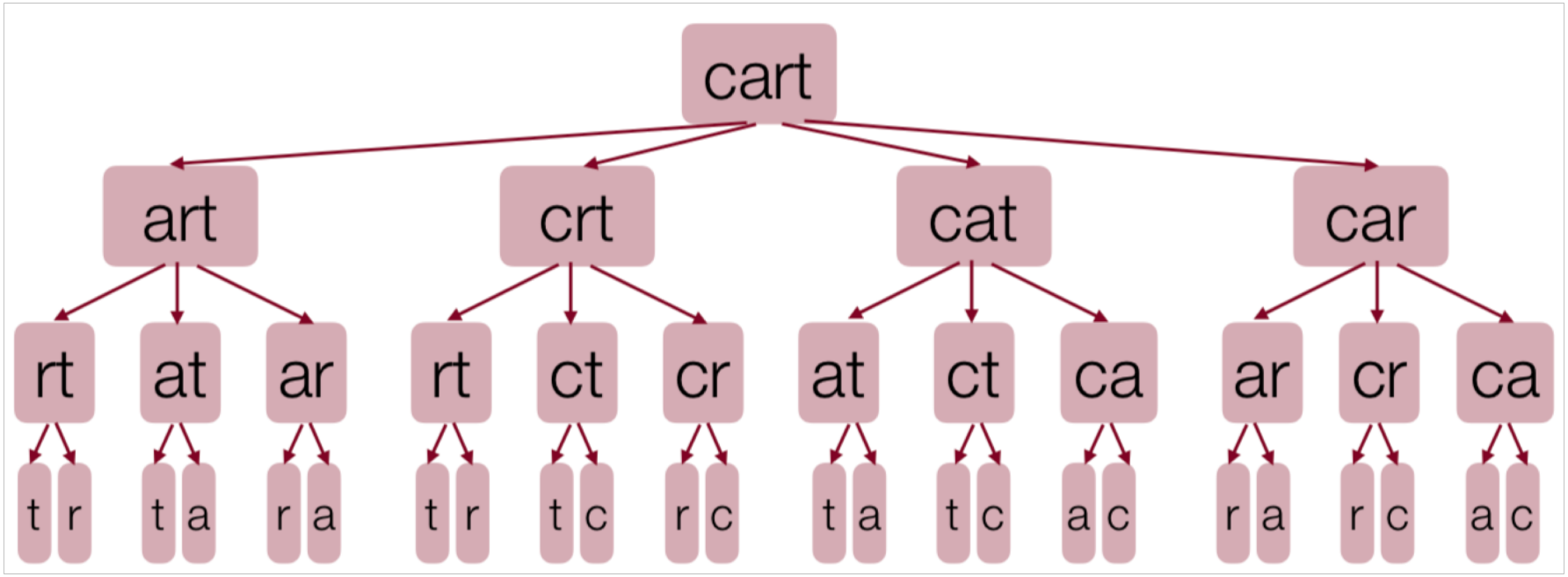
- WiCS Casual Dinner **Tues. 11/6 5-7 in Gates 403**! Join me!



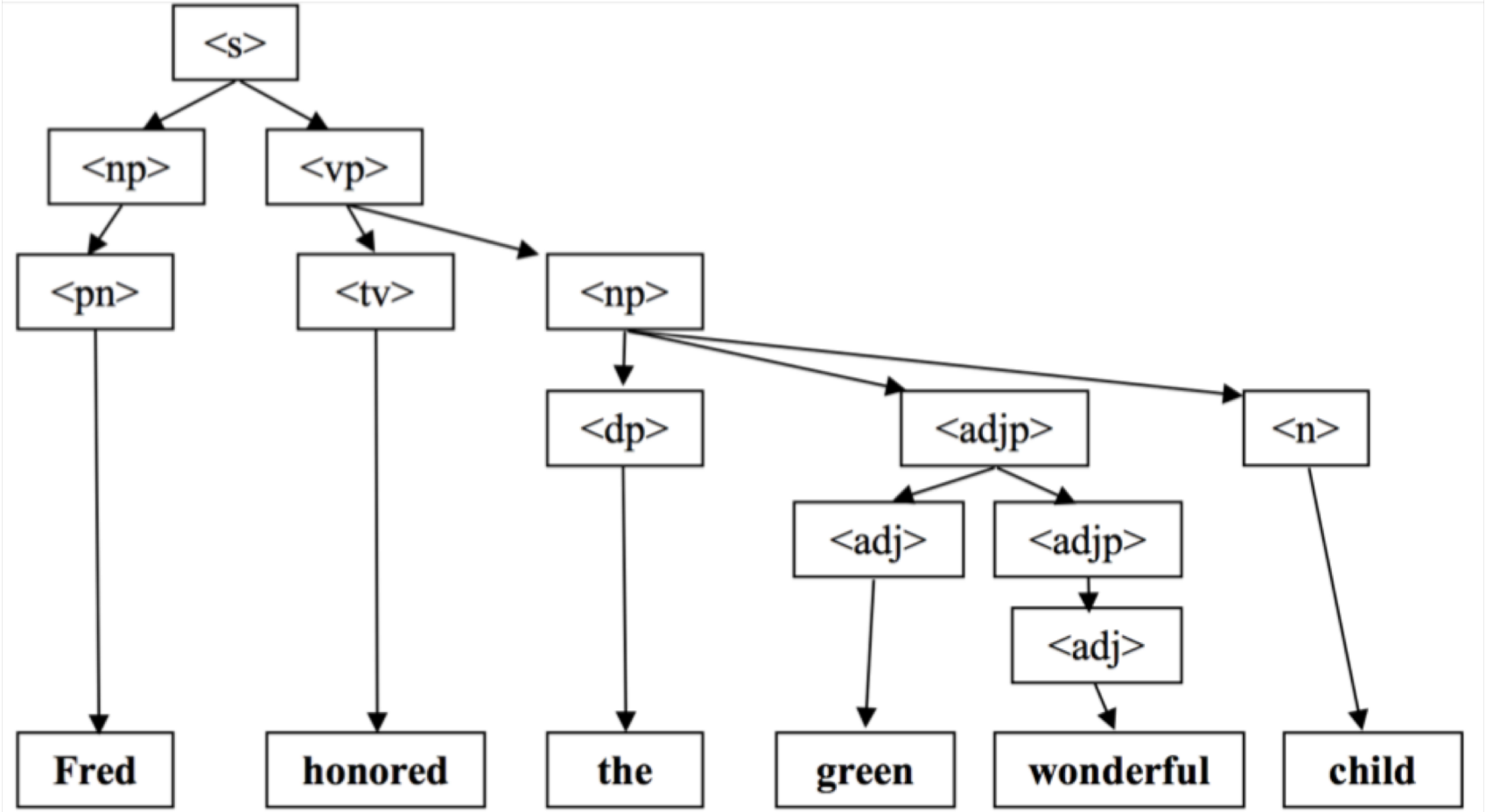
Plan For Today

- Arrays
- Announcements
- Trees

Trees

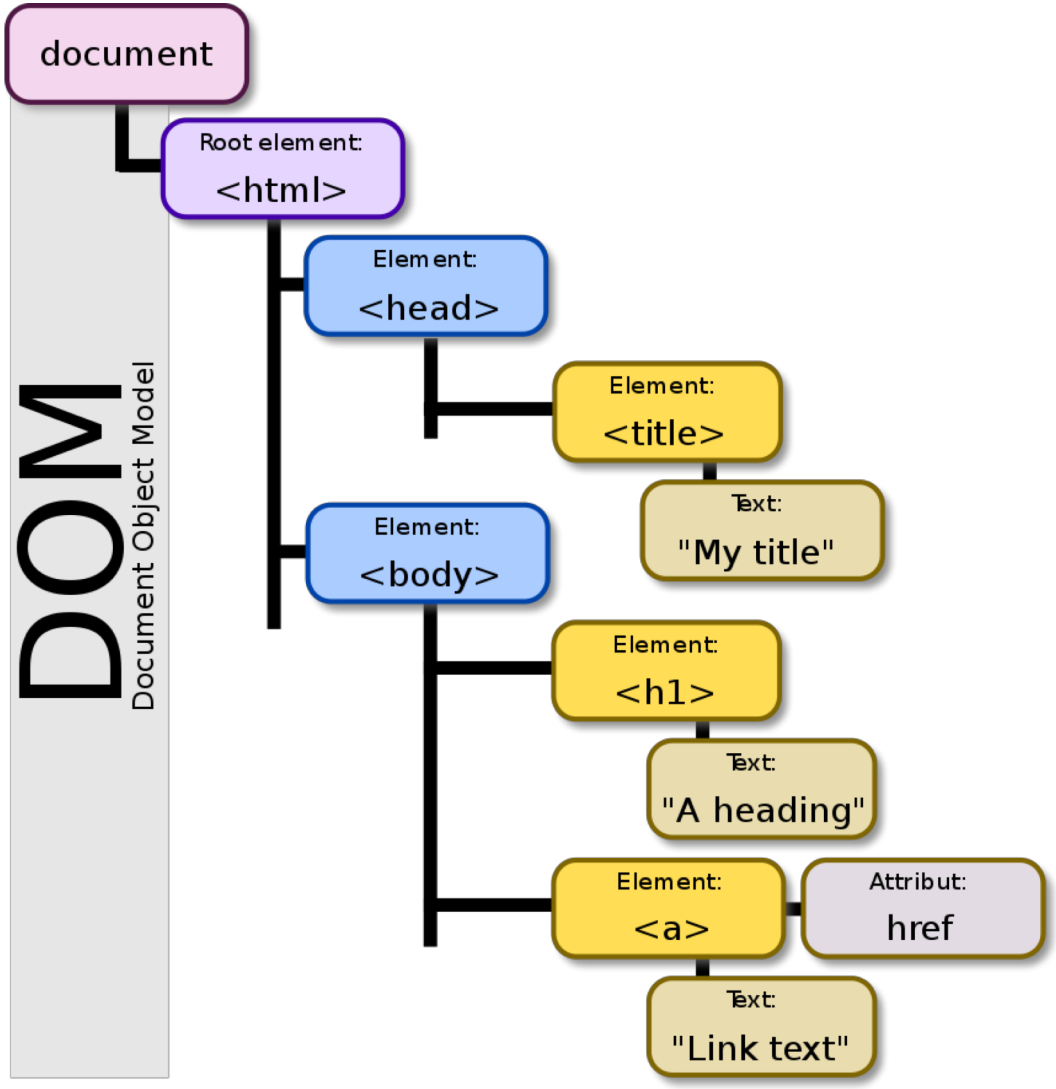

cart
art
crt
cat
car
rt
at
ar
rt
ct
cr
at
ct
ca
ar
cr
ca
t r
t a
r a
t r
t c
r c
t a
t c
a c
r a
r c
a c

Trees



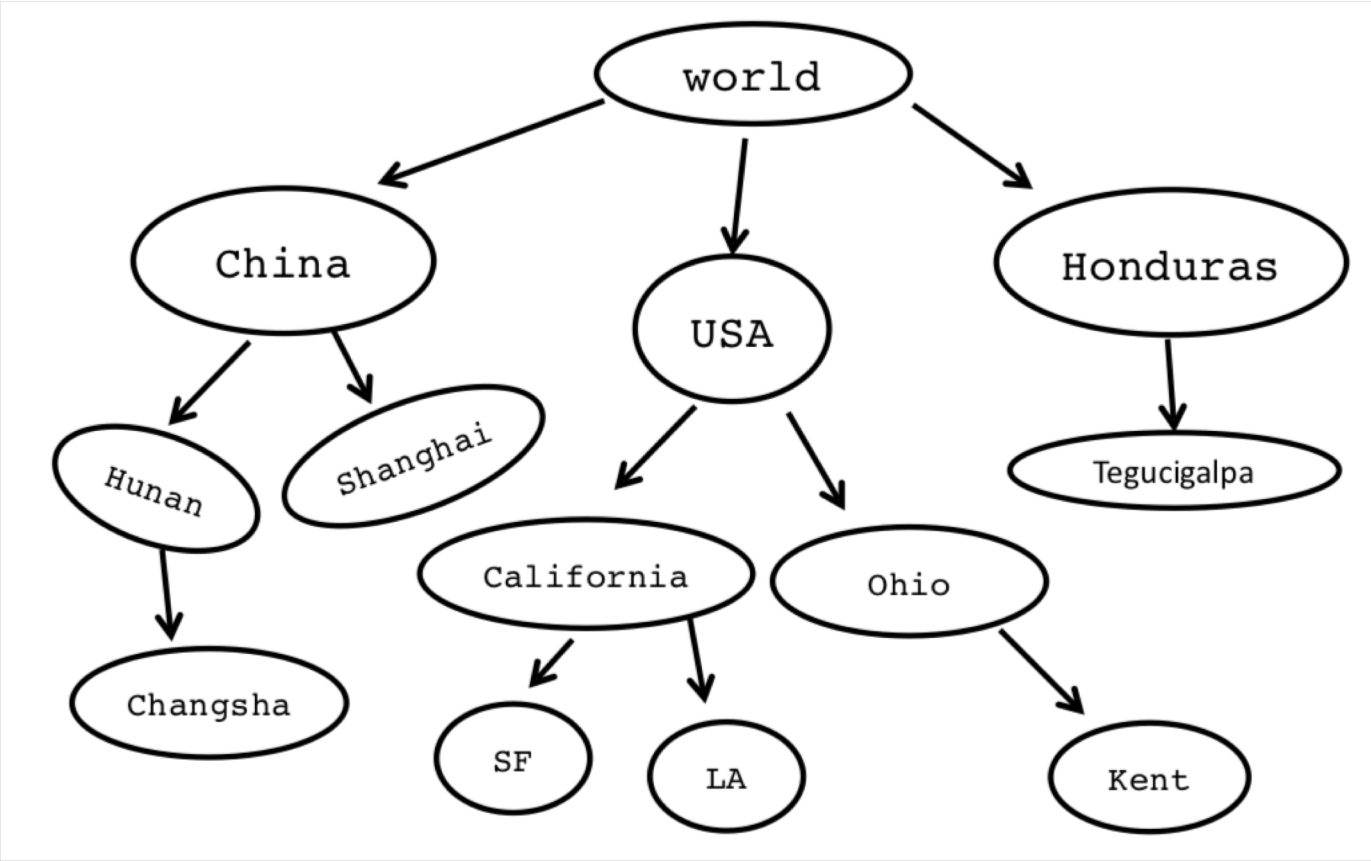
Random expansion from `sentence.txt` *grammar for symbol* `"<s>"`

Trees



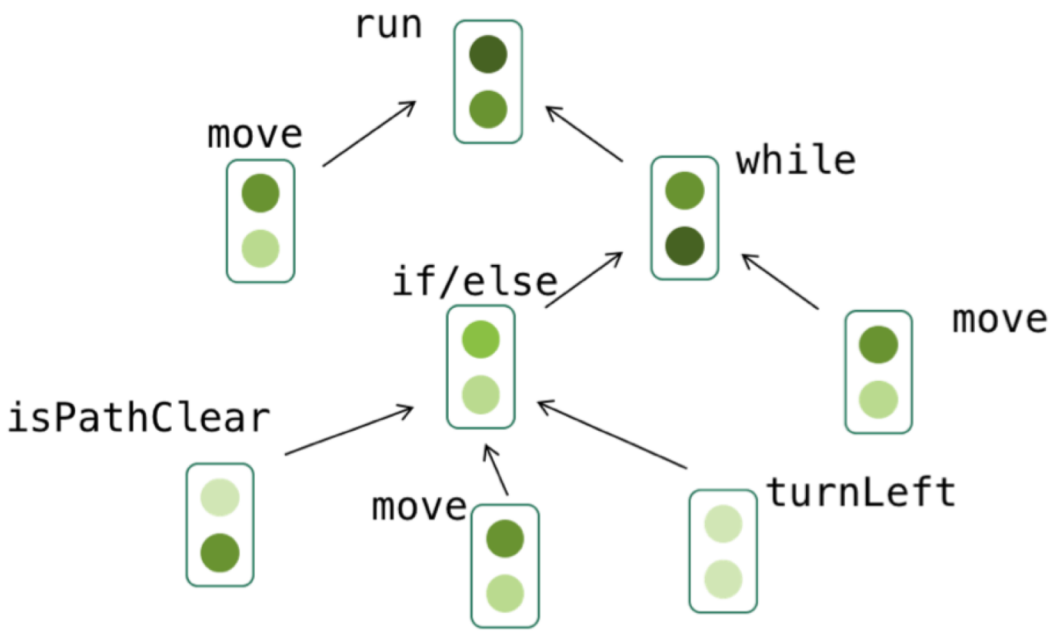
By Birger Eriksson : https://en.wikipedia.org/wiki/File:DOM-model.svg

Trees


world
China
USA
Honduras
Hunan
Shanghai
Tegucigalpa
California
Ohio
Changsha
SF
LA
Kent

Trees

```
// Example student solution
function run() {
    // move then loop
    move();
    // the condition is fixed
    while (notFinished()) {
        if (isPathClear()) {
            move();
        } else {
            turnLeft();
        }
        // redundant
        move();
    }
}
```

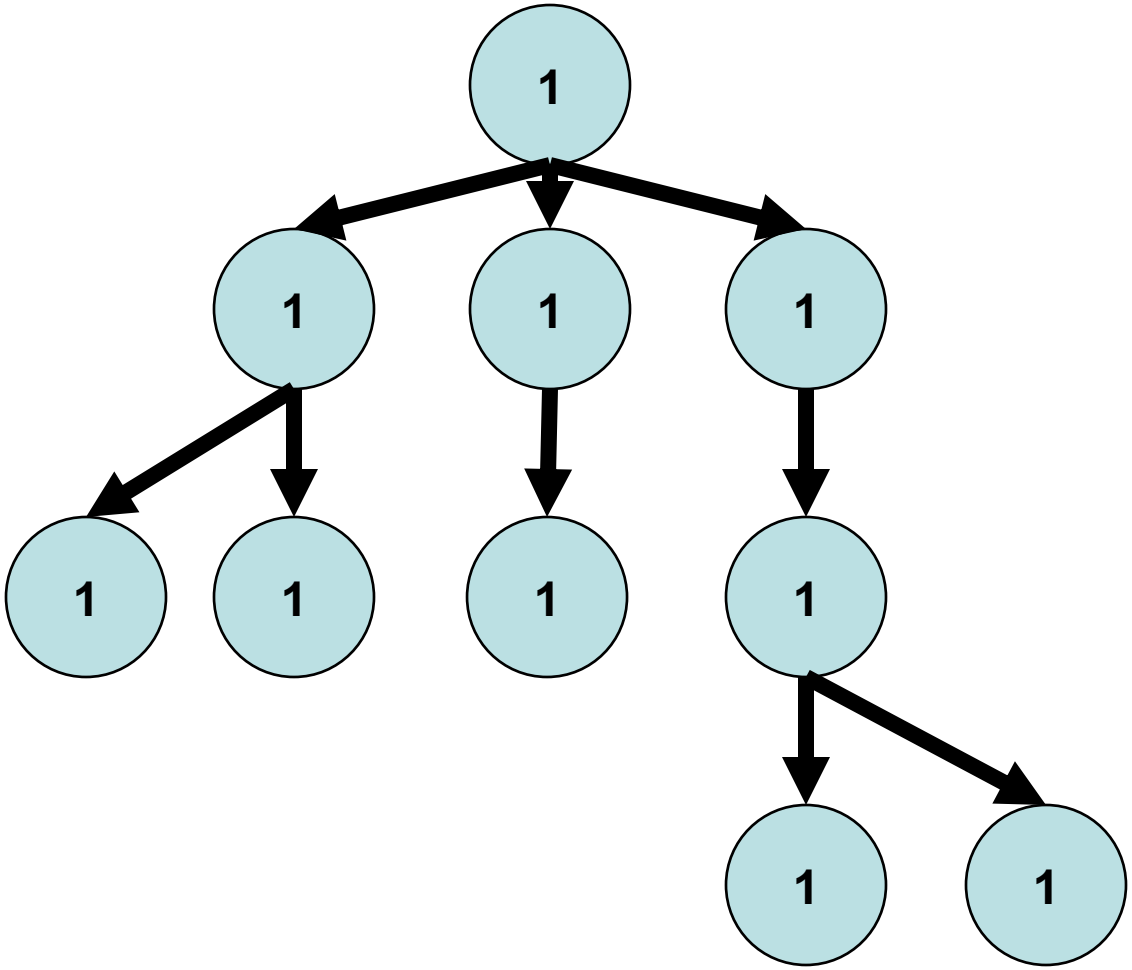


This is a figure in an academic paper written by a recent CS106 student!

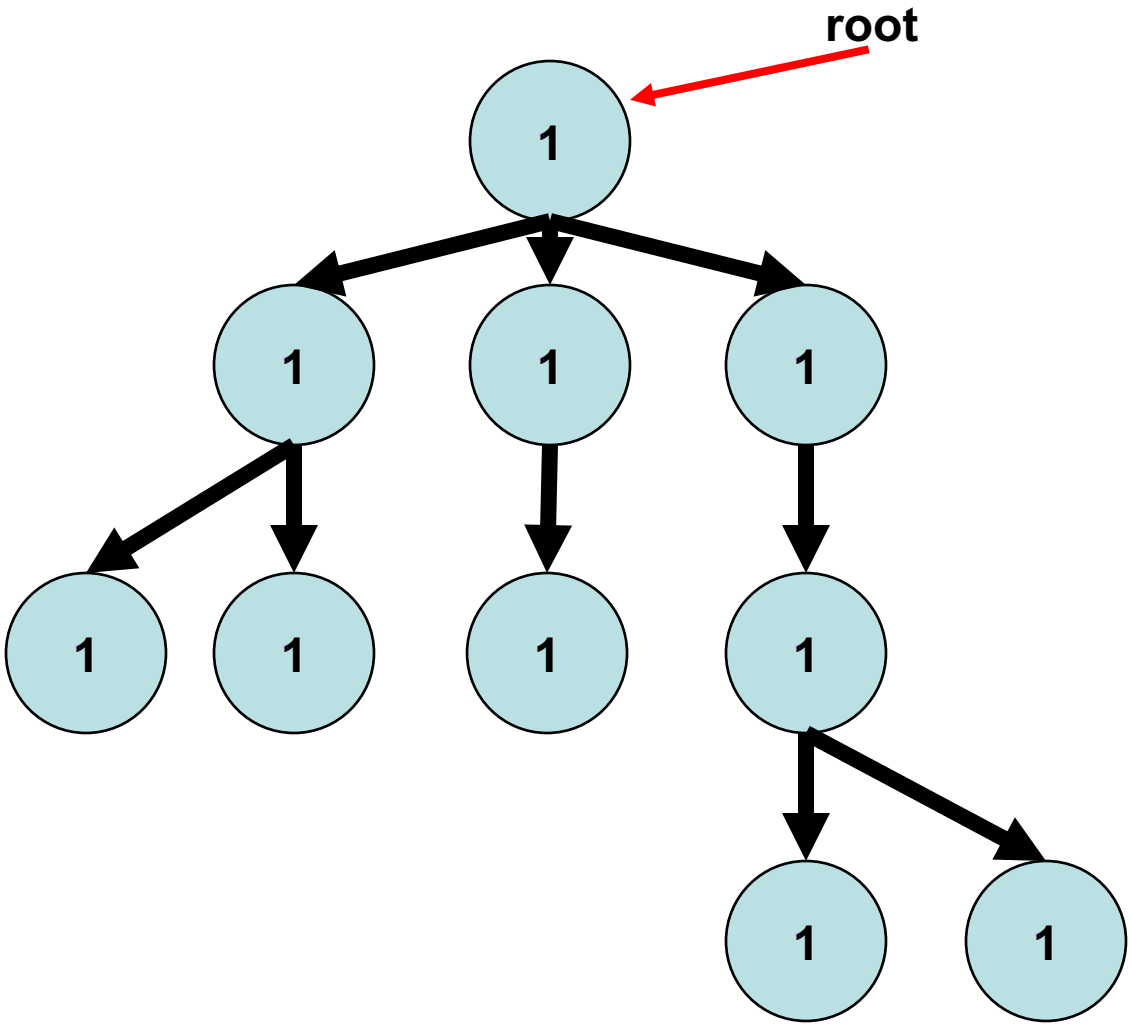
Trees

- **Tree:** a directed, acyclic structure of linked nodes
  - ***Directed:*** has one-way links between nodes
  - ***Acyclic:*** No path wraps back around to the same node twice (aka only one unique path from a node to any other node.
- Recursive!
  - A tree is either empty, or
  - A root node that contains **data** and zero or more **subtrees**.

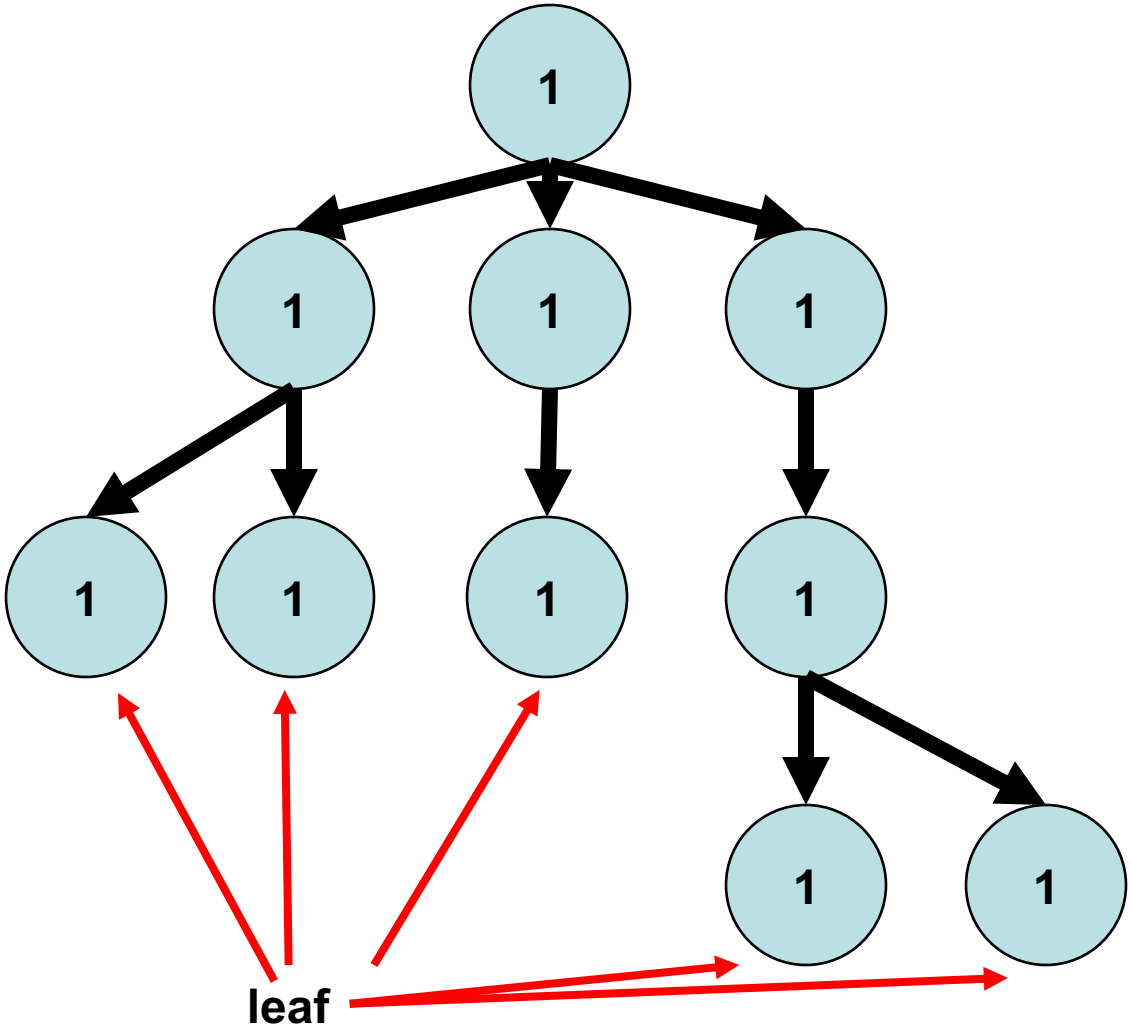
Trees



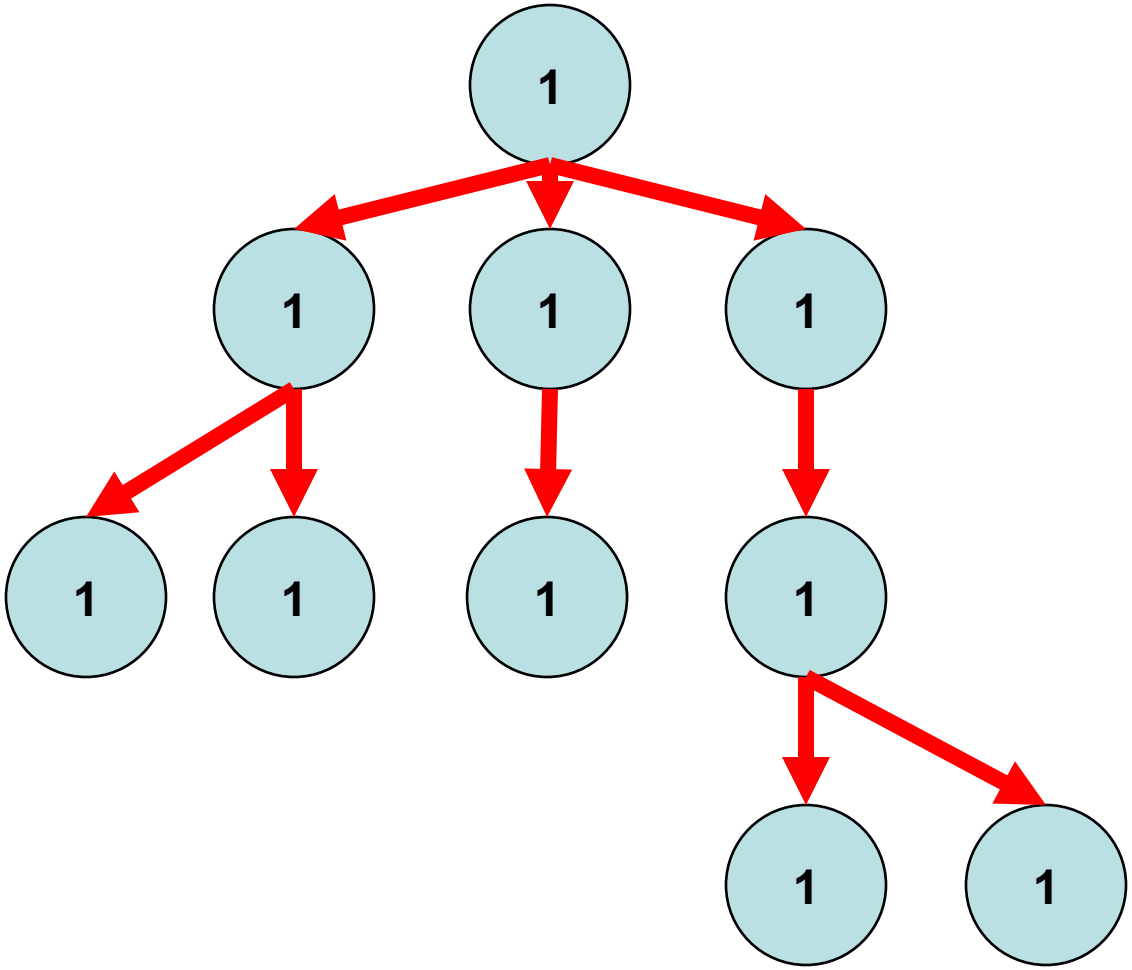
Trees



Trees

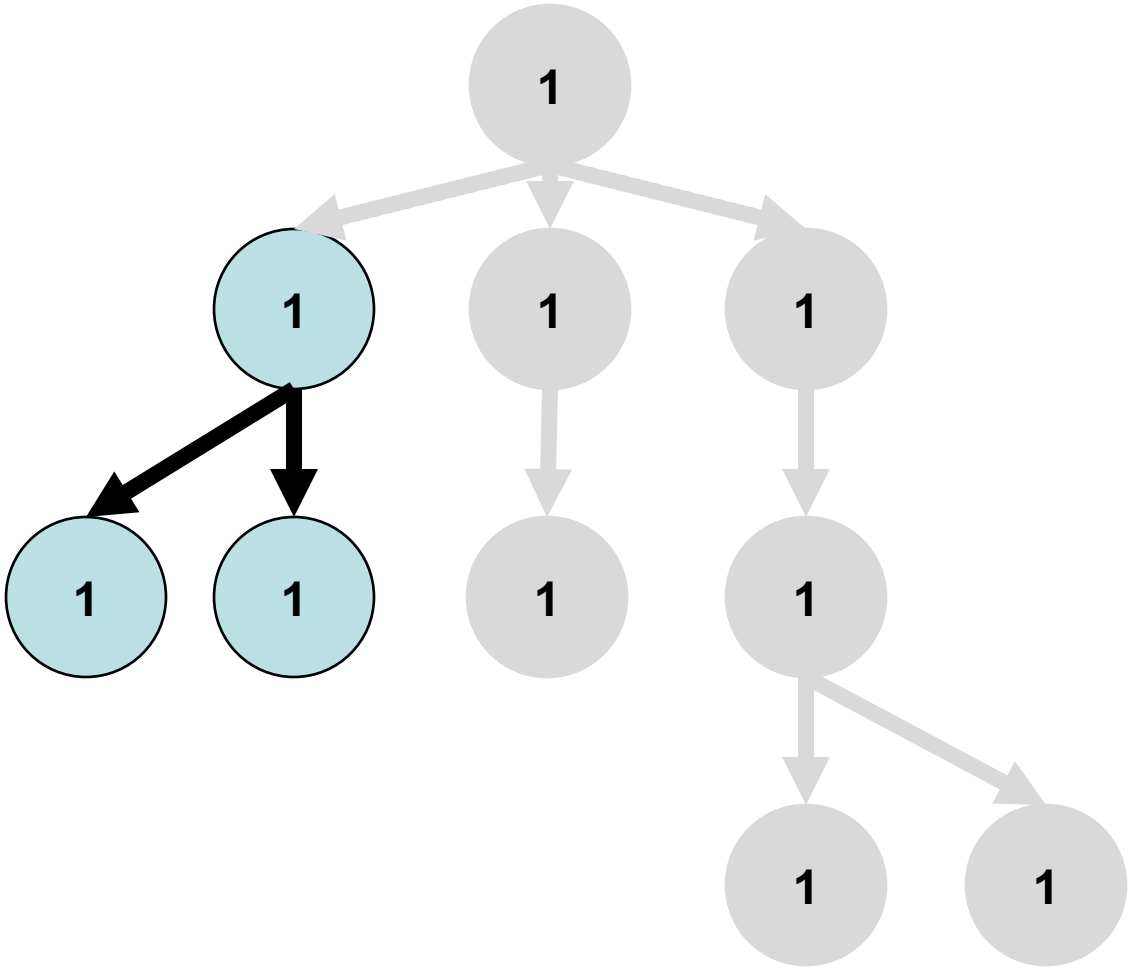


Trees

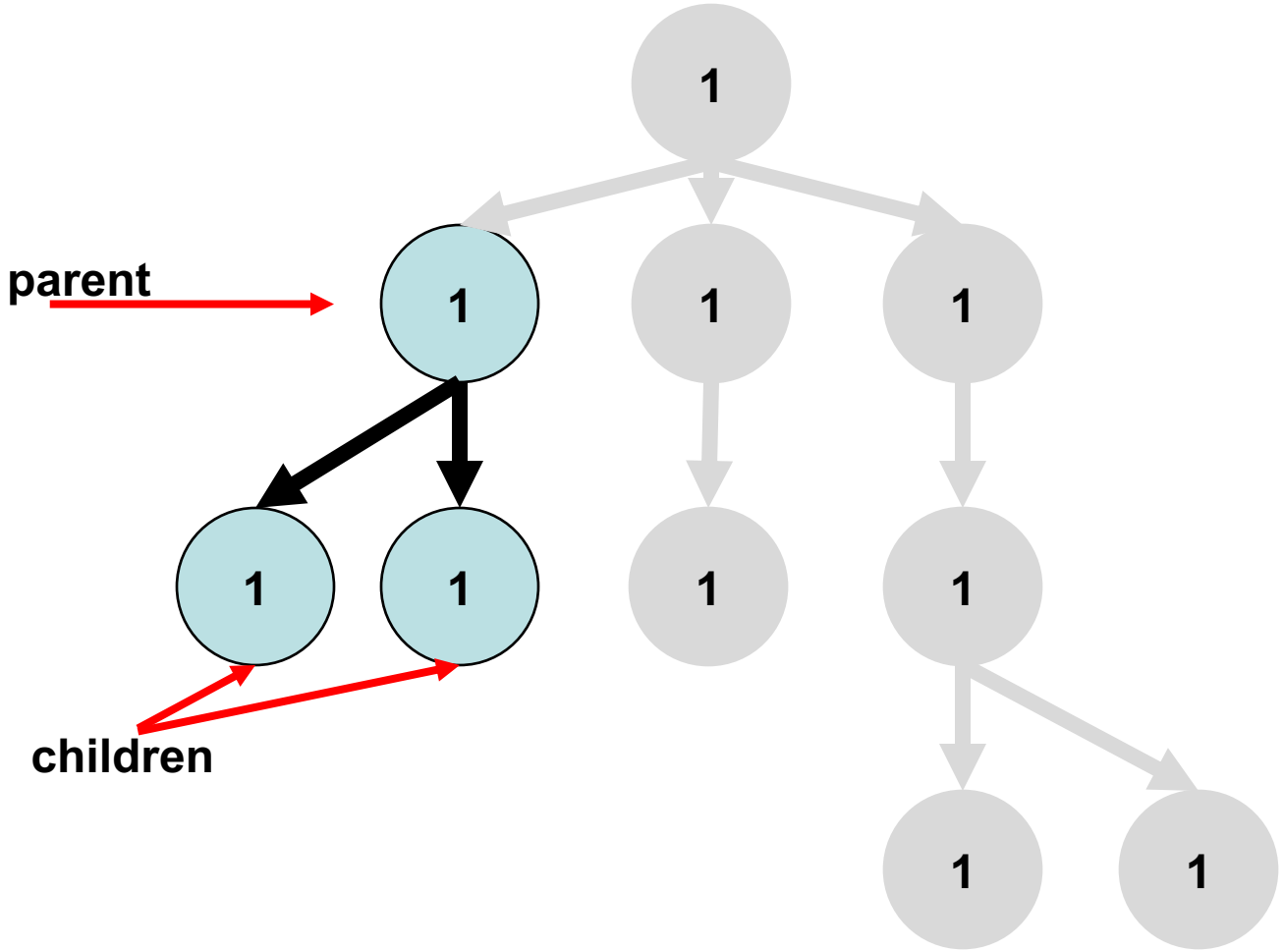


branches

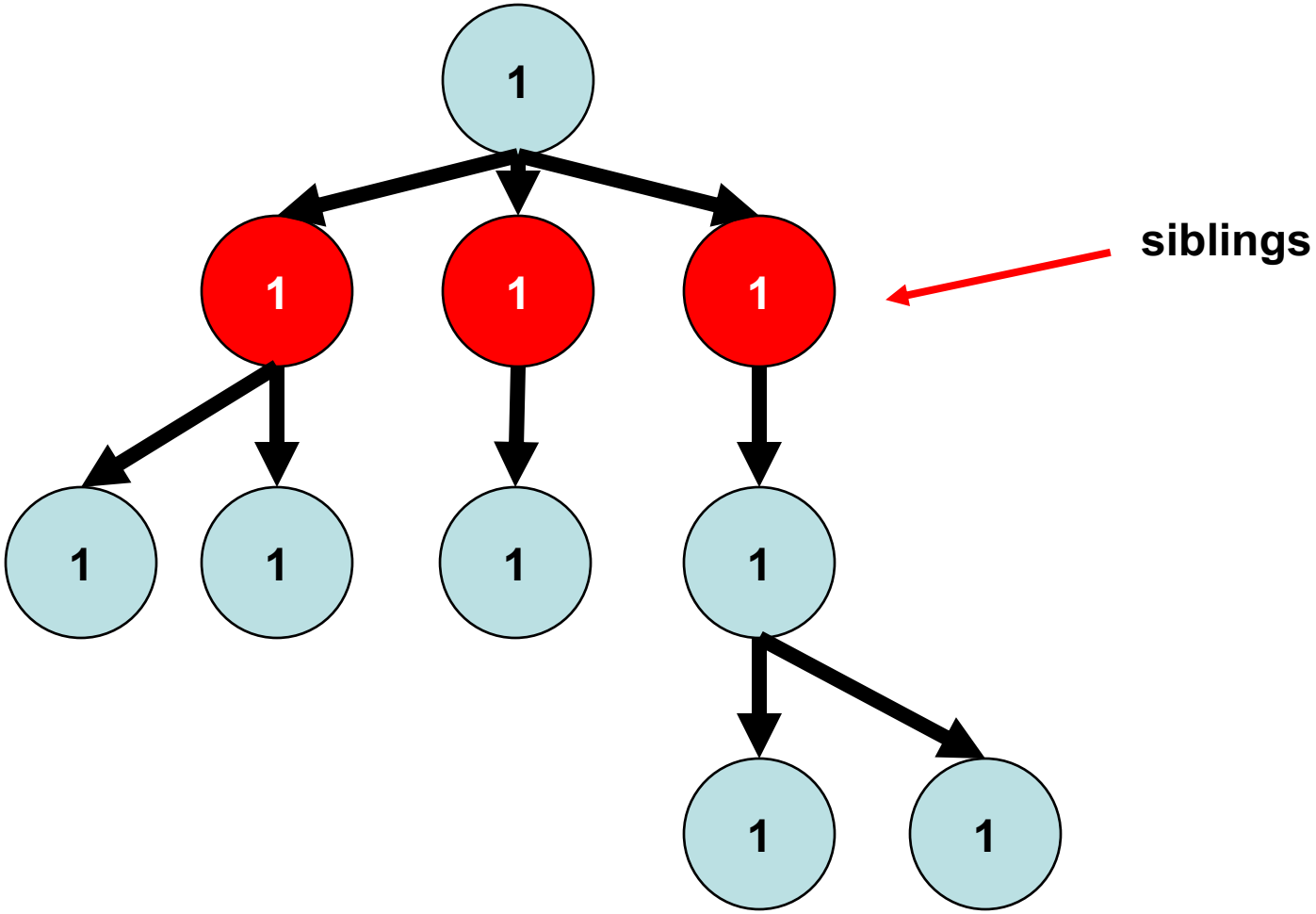
Trees



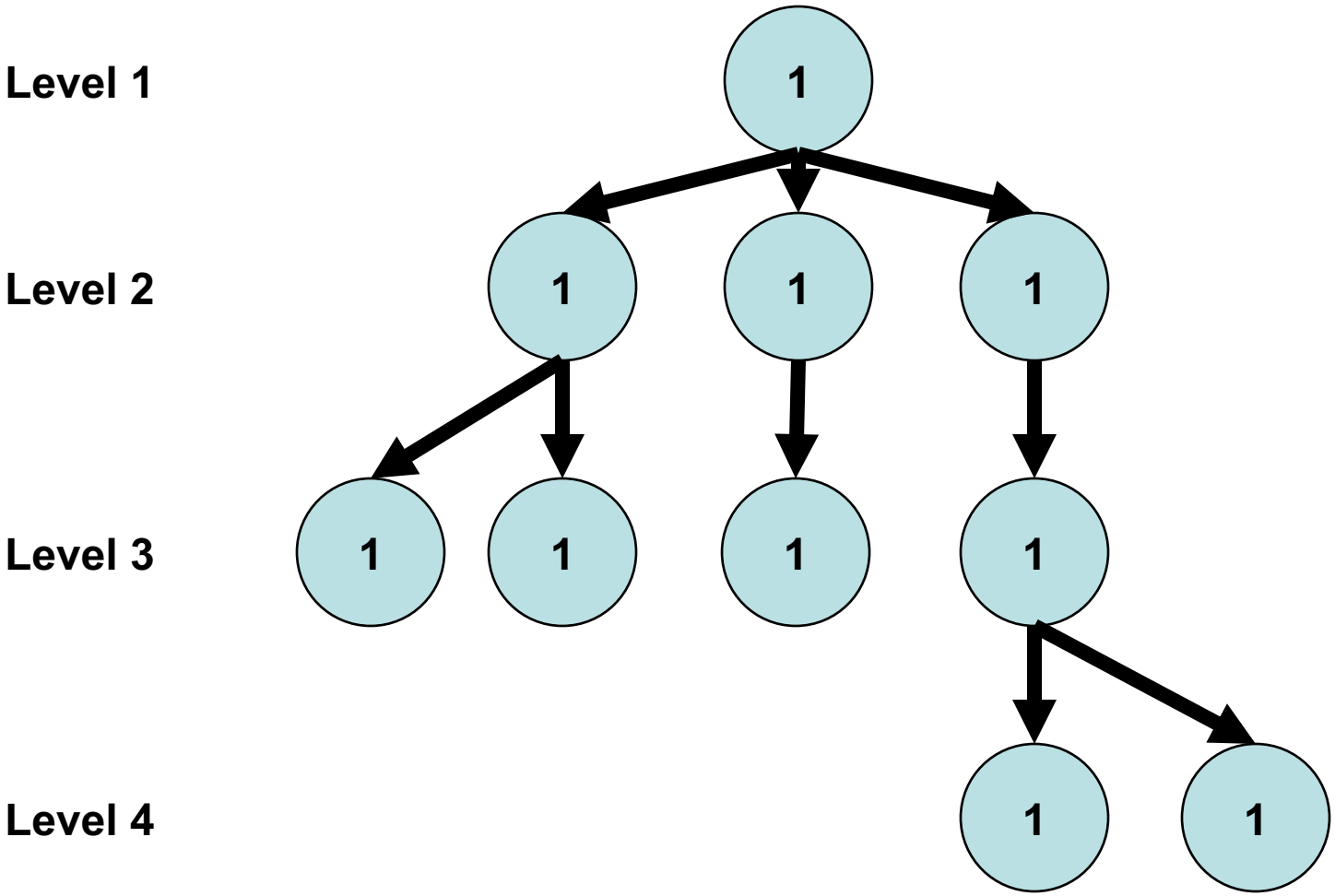
Trees



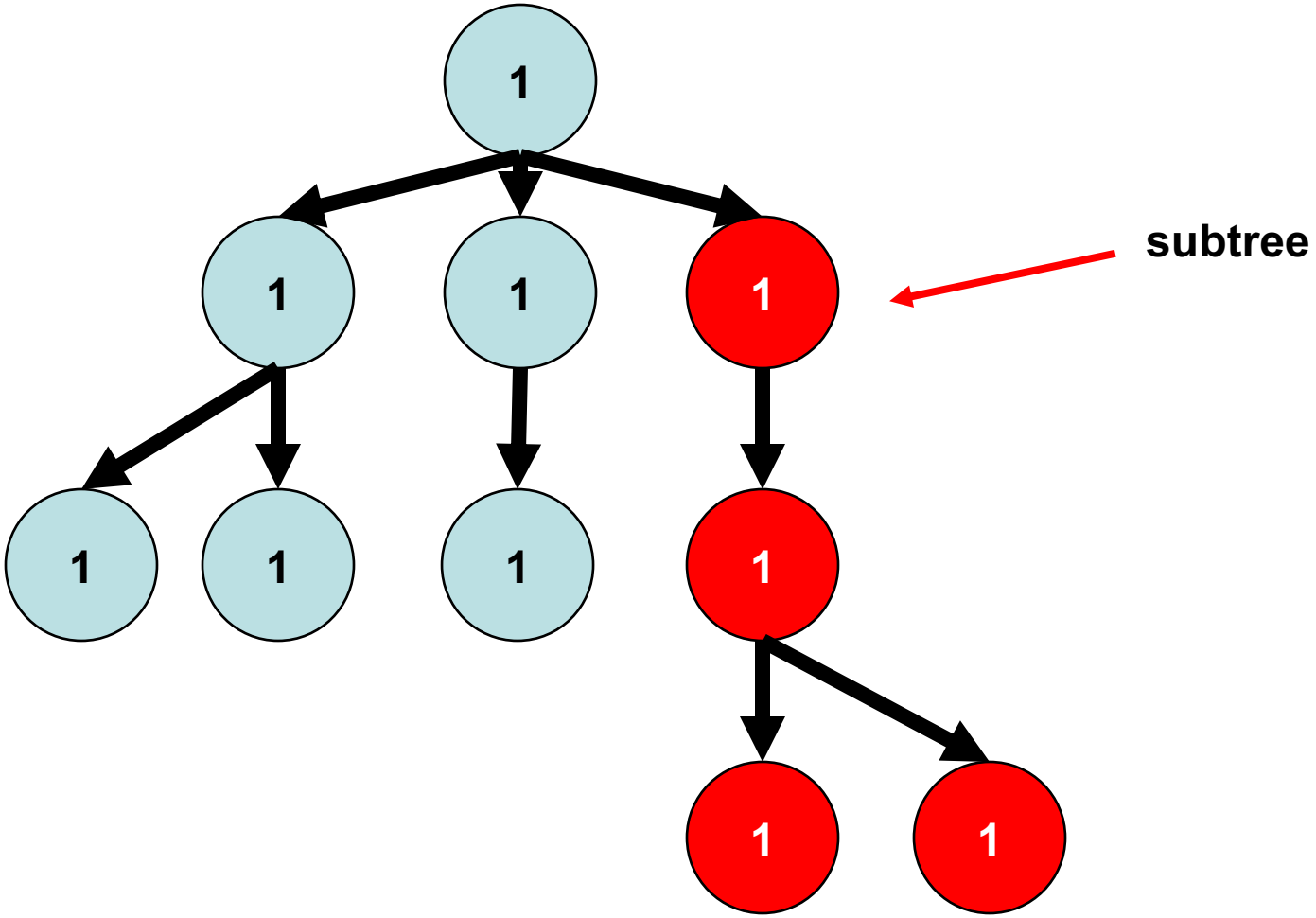
Trees



Trees

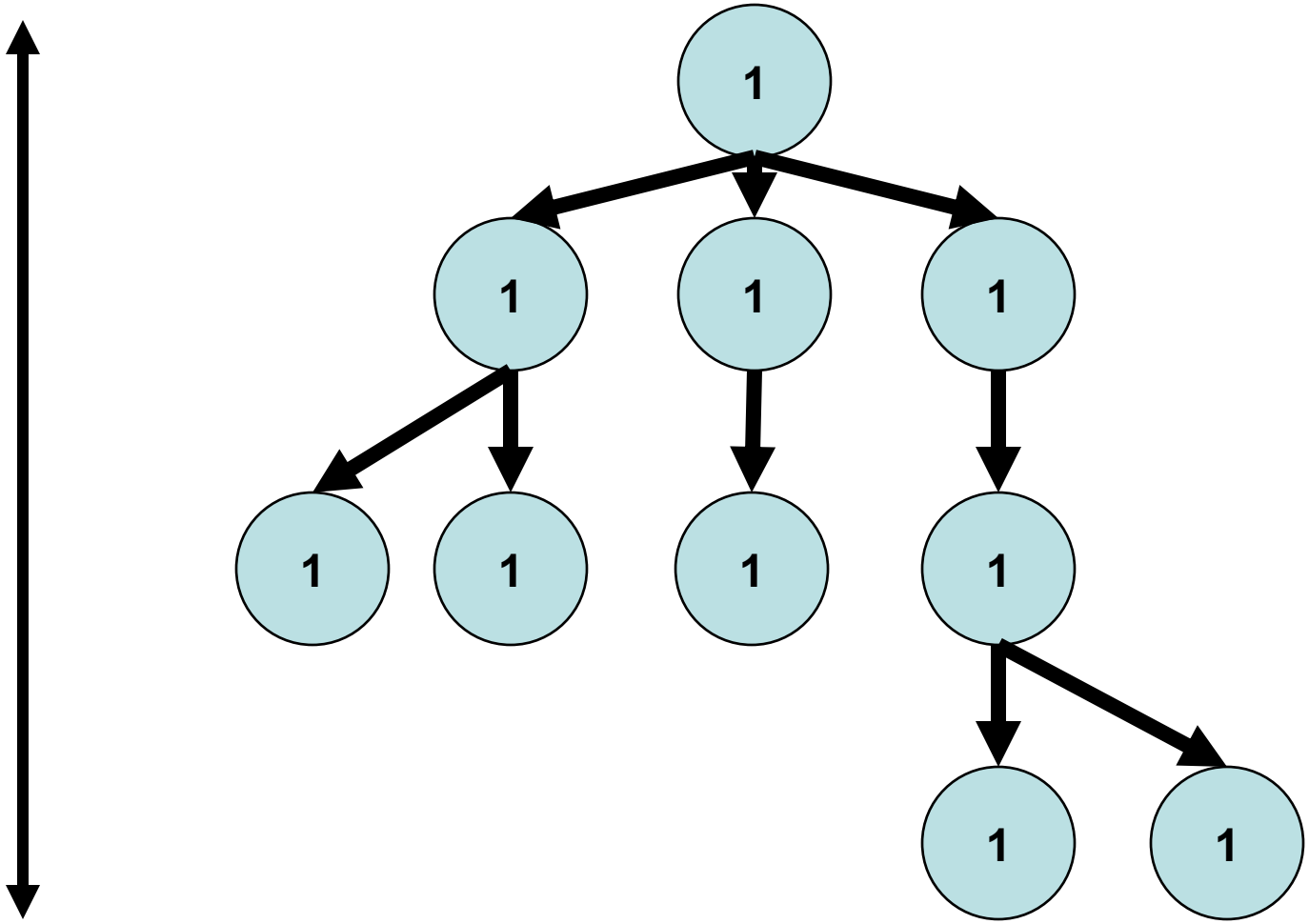


Trees



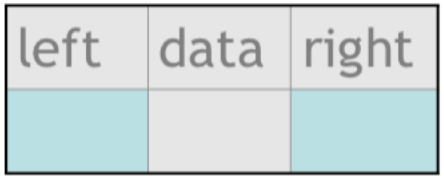
Trees

Height = 4 -> longest path from the root to any node, in # nodes



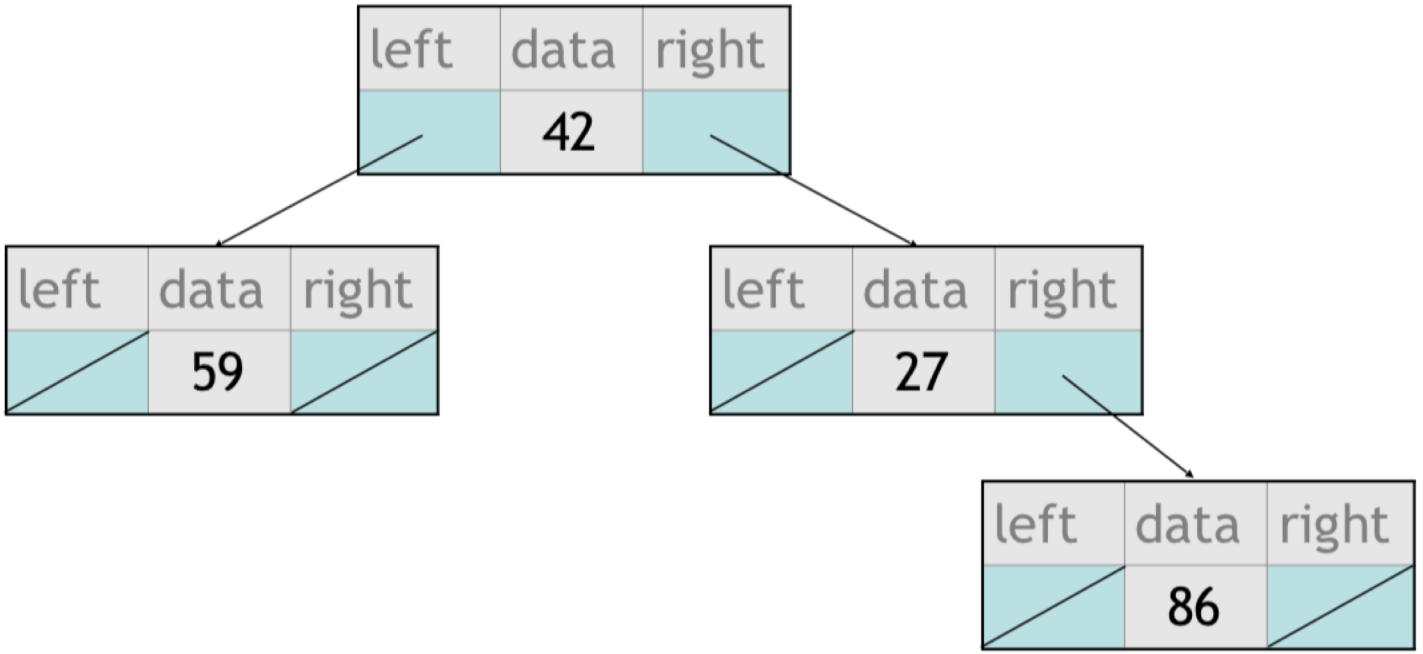
TreeNode

```
// A TreeNode is one node in a binary tree of integers.
struct TreeNode {
    int data;           // data stored at this node
    TreeNode* left;     // pointer to left subtree
    TreeNode* right;    // pointer to right subtree

    // Constructs a node with the given data and links.
    TreeNode(int data, TreeNode* left, TreeNode* right) {
        this->data = data;
        this->left = left;
        this->right = right;
    }

    bool isLeaf() const {
        return left == nullptr && right == nullptr;
    }
};
```



TreeNode

- A basic **tree node object** stores data and pointers to left/ right
  - Multiple nodes can be linked together into a larger tree



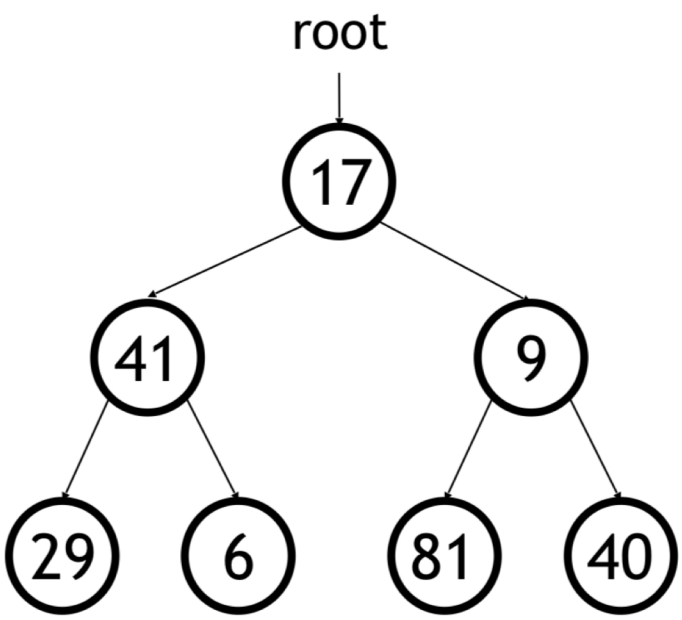
Common Tree Operations

- print
- height
- size
- contains
- deleteTree

print

- Write a function named **`print`** that accepts a tree node pointer as its parameter and prints the elements of that tree, one per line.
  - A node's left subtree should be printed before it, and its right subtree should be printed after it.
  - Example: `print(root);`

```
29
41
6
17
81
9
40
```



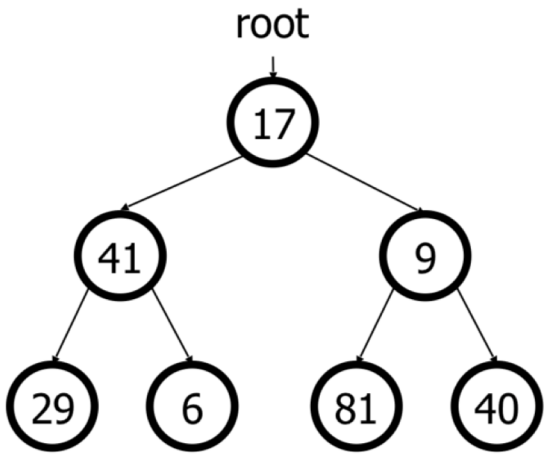
print

```
void print(TreeNode* node) {
    // (base case is implicitly to do nothing on NULL)
    if (node != nullptr) {
        // recursive case: print left, center, right
        print(node->left);
        cout << node->data << endl;
        print(node->right);
    }
}
```

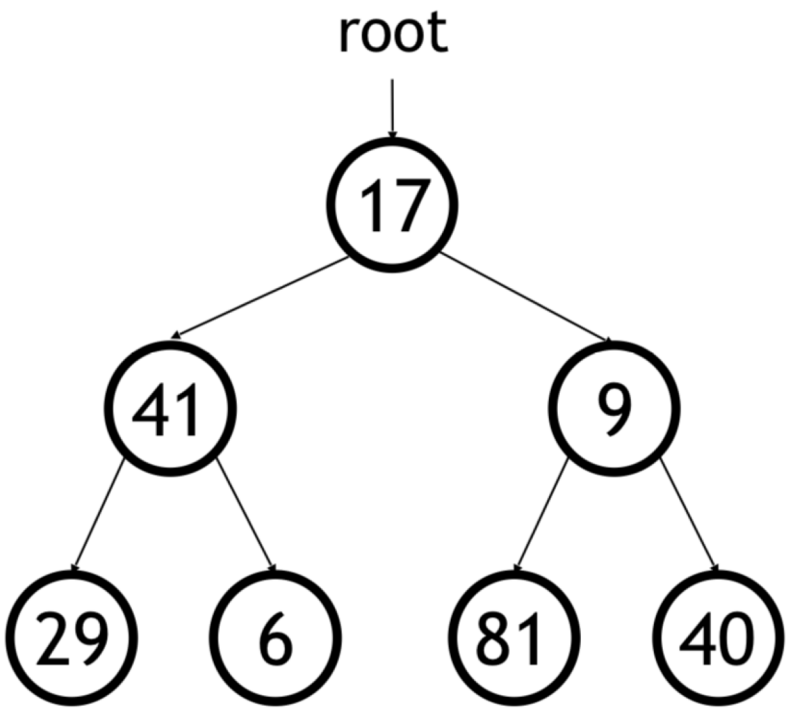
Traversal

- **traversal**: An examination of the elements of a tree.
  - A pattern used in many tree algorithms and methods

- Common orderings for traversals:
  - **pre-order**: process root node, then its left/right subtrees
  - **in-order**: process left subtree, then root node, then right
  - **post-order**: process left/right subtrees, then root node



Traversal



- pre-order: 17 41 29 6 9 81 40
- in-order: 29 41 6 17 81 9 40
- post-order: 29 6 41 81 40 9 17

Recap

- Arrays
- Announcements
- Trees

Next time: more trees!