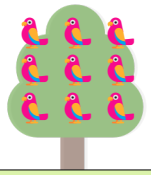
✨ No Laptop Monday Funday! ✨

Week09a

- directed graph preview
- trees



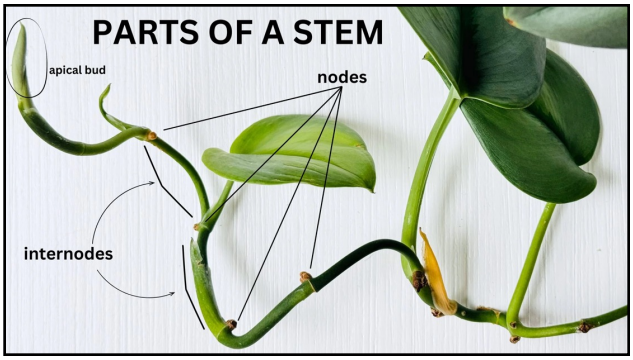
BOTANY WARMUP

what is a tree?
what is a branch?
what is a node?

ADDITIONAL WARMUP

what does binary mean?
what is a binary tree?
what does ternary mean?

[record lecture]



directed graphs

quick note: Math vs. CS

CS vs. Math

- a **directed graph** is a data structure
  - it has **nodes** that **refer** to any number of other nodes
- a **directed graph** G = (V, E) is also a mathematical object
  - it has **vertices** (**nodes**) V
  - it has **directed edges** E
    - **note:** E is a set of 2-tuples

these are really just two different ways of looking at the same thing

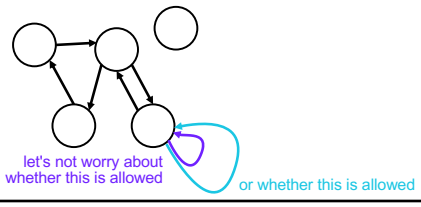
both are useful!
(Math to figure out the algorithm, CS to code it up)

in CSCI 136, i will focus on the CS view
(MATH 200, CSCI 256 will look at the Math side)

directed graph

the directed graph generalizes the linked list

- a node in a **directed graph** has references to any number of other nodes
  - nodes are drawn as circles
  - references are drawn as arrows



(whether we allow for self-edges, multiple edges, etc.
depends on context)

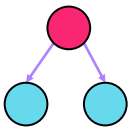
🙄 👍

tree

kinship terminology
(parent, child, etc.)

kinship terminology is useful for describing trees

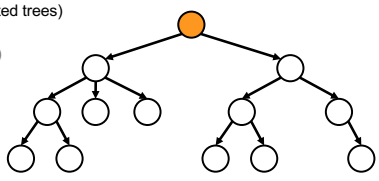
- a node has references to other nodes
  - it is their **parent**
  - they are its **children**



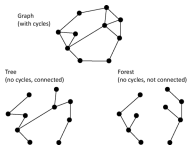
(directed) tree

a (directed) **tree** is a directed graph that...

- has exactly one node with zero parents (the **root**)
- has no nodes with more than one parent
- has no **cycles** (loops) of any kind
- is **connected** (just one tree, not multiple disconnected trees)
  - (otherwise, it would be a forest)



note: not joking



Monday Funday

time for everyone's favorite home game...

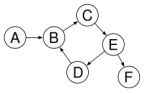
is it a tree?

tree rules

1. exactly one node with zero parents
2. no node with more than one parent
3. no **cycles**
4. is **connected**

no
violates 1 & 4

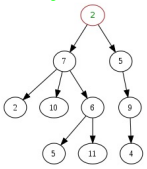
no
violates 2 & 3



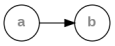
no
violates 2 & 3

no
violates 1 & 3

yes

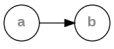


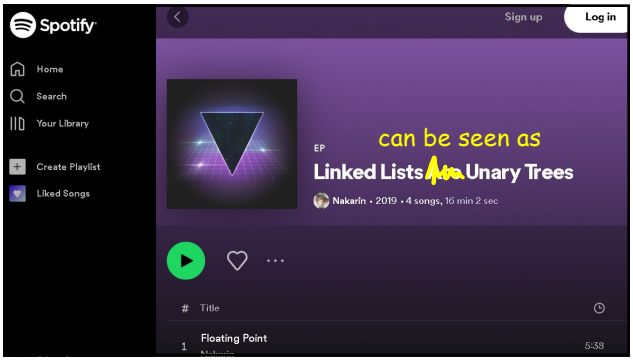
yes



is it a tree?

yes

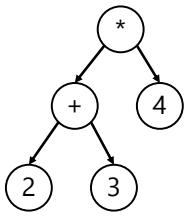




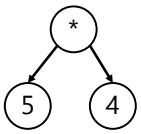
uses of trees

simplifying expressions

(2 + 3) * 4



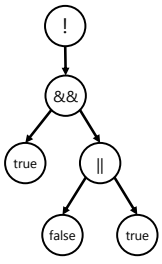
5 * 4



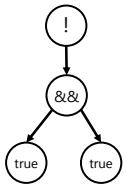
20

20

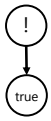
!(true && (false || true)



!(true && true)

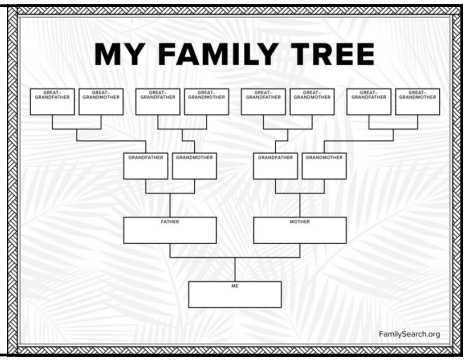


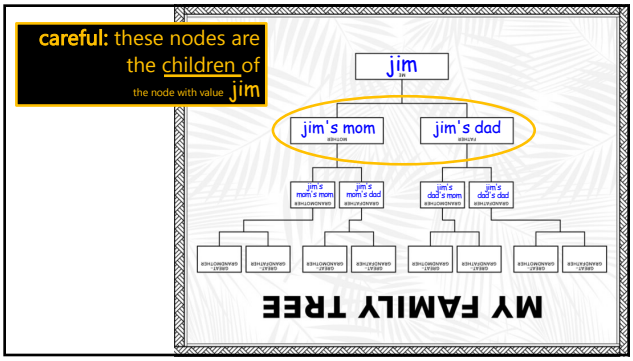
!true



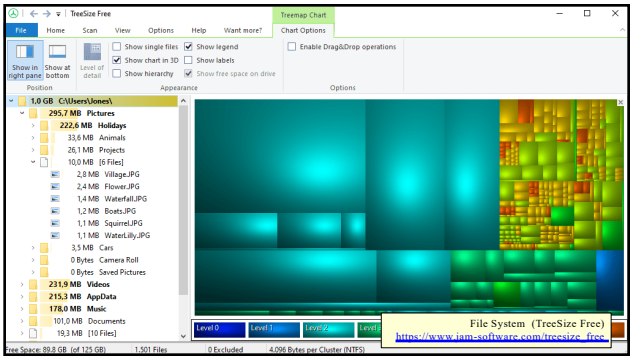
false

and so much more!

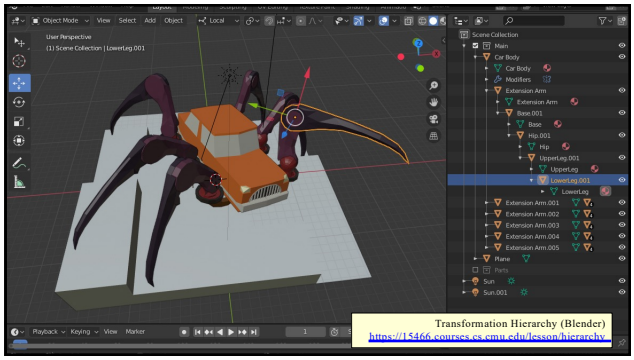


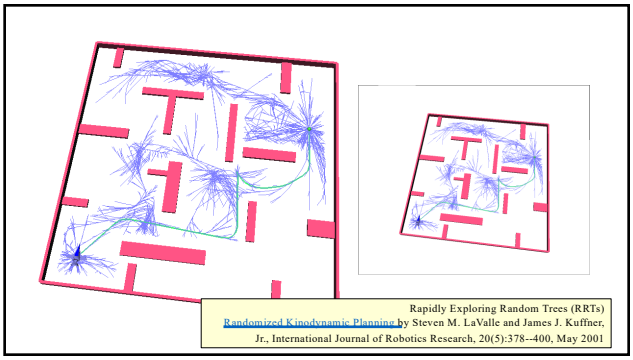


note:
family trees considered deeply confusing;
let us never talk of them again



File System (TreeSize Free)
https://www.jam-software.com/treesize_free


Transformation Hierarchy (Blender)
https://15466.courses.cs.cmu.edu/lesson/hierarchy


Rapidly Exploring Random Trees (RRTs)
Randomized Kinodynamic Planning by Steven M. LaValle and James J. Kuffner, Jr., International Journal of Robotics Research, 20(5):378--400, May 2001

Hierplane (Mark)

[homework preview]

different ways to implement trees

```
ArrayList<Node> children;

Node leftChild;
Node rightChild;

Node[] children;
```

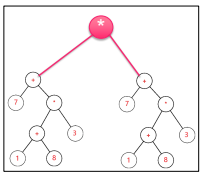
✨ Laptop Wednesday! ✨

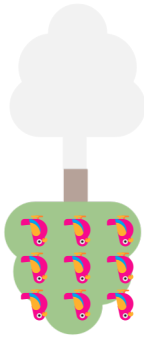
Week09b

- for-each loop
- tree terminology
- terrific tree traversal tutorial

WARMUP
what does **depth** mean?
what does **breadth** mean?
what does **first** mean?

BONUS WARMUP
evaluate this expression tree





[address the election?]

[record lecture]

 for-each loop

☠️ for-each loop

Python's for loop is a **for-each loop**

```python
lst = [ True, 5.0, "Hello" ]
for element in lst: # for each item in the list...
    print(element)
```

a **for-each loop** can be a convenient alternative to a for loop

```java
for (int turretIndex = 0; turretIndex < turrets.length; ++turretIndex) {
    Turret turret = turrets[turretIndex];
    ...
}
```

```java
for (Turret turret : turrets) { // for each turret in turrets...
    ...
}
```

a **for-each loop** can be a convenient alternative to a for loop

```java
for (int strokeIndex = 0; strokeIndex < currentFrame.size(); ++strokeIndex) {
    ArrayList<Point> stroke = currentFrame.get(strokeIndex);
    for (int pointIndex = 0; pointIndex < stroke.size(); ++pointIndex) {
        Point point = stroke.get(pointIndex);
        ...
    }
}
```

```java
for (ArrayList<Point> stroke : currentFrame) {
    for (Point point : stroke) {
        ...
    }
}
```

forgive me 😳
i swear all that indexing built character

for-each can be convenient but...

you never *need* a for-each loop
(some languages don't even have one)

for-each loops can get you into trouble if you don't fully understand how references and Objects work in Java

for-each loop dangers

- a **for-each loop provides a "convenient reference"**
  - **LESS powerful than a regular for loop**

```
for (int i = 0; i < things.length; ++i) {
    Thing thing = things[i]; // thing is a "convenient reference"
    ...
}
```

```
for (Thing thing : things) { // thing is a "convenient reference"
    ...
}
```

for-each loop dangers

```
for (int i = 0; i < things.length; ++i) {
    things[i] = new Thing(); // does write to array 😀👍
}
```

```
for (int i = 0; i < things.length; ++i) {
    Thing thing = things[i]; // thing is a "convenient reference"
    thing = new Thing(); // does NOT write to array
}
```

```
for (Thing thing : things) { // thing is a "convenient reference"
    thing = new Thing(); // does NOT write to array
}
```

for-each loop for iterating over a `String`

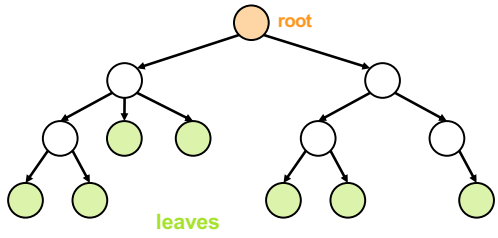
- in Java, a for-each loop can iterate through `String`'s with `toCharArray()`
  - **`toCharArray()` is O(n), but fine for CSCI 136**

```
for (char c : string.toCharArray()) {
    ...
}
```

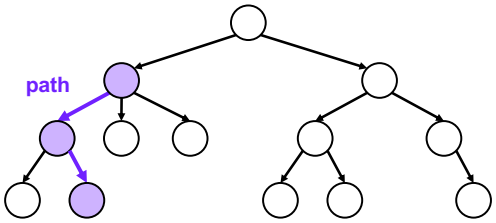
more tree terminology

more tree terminology

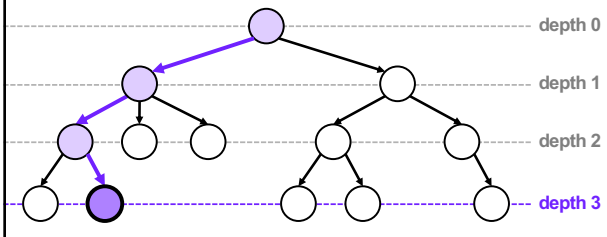
the **root** has no parents
a **leaf** has no children



a **path** is a chain of distinct edges



a node's **depth** is the number of edges
in *the path from the root to that node*

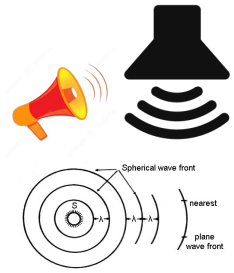


level means either *depth* or *(depth + 1)*
depending on who you ask
🤷

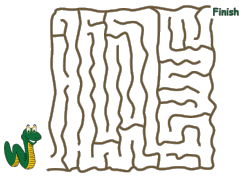
tree traversal

breadth-first vs. depth-first

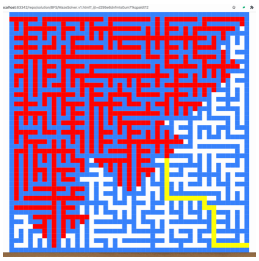
breadth-first vs. (pre-order) **depth-first**

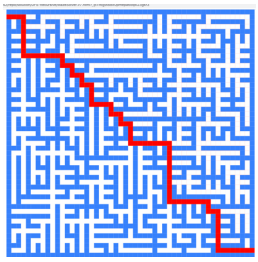


"a persistent, optimistic, very thorough worm" –Mark Hopkins

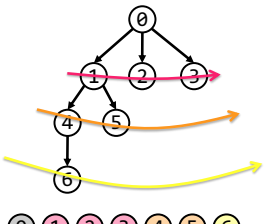


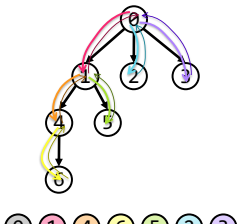
breadth-first vs. (pre-order) **depth-first**





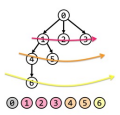
breadth-first vs. (pre-order) **depth-first**



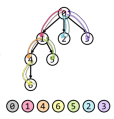


tree traversal (walking the tree)

- **tree traversal** means iterating through all of the tree's nodes
  - **breadth-first** first explores as broadly (widely) as possible
    - one **level** (**depth**) at a time
    - uses a **queue**



  - **depth-first** first explores as deeply as possible
    - one **root-to-leaf path** at a time
    - uses a **stack** (or **recursion**, which uses the callstack)
    - ☠ **there are actually many different variants of depth-first traversal (i will explain them when we get to binary trees)**



why might we want to traverse a tree?

why did we want to traverse a linked list?

do we ever *traverse* an array?

gathering all words in a **trie** (HW09)

searching for something (HW10?)

evaluating an **expression tree**

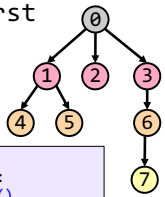
gathering all the nodes into a list (Tut09)

Terrific Tree Traversal Tutorial

explanation

traverseBreadthFirst

- 0 added to queue
- 0 removed from queue
- 1 added to queue
- 2 added to queue
- 3 added to queue
- 1 removed from queue
- 4 added to queue
- 5 added to queue
- 2 removed from queue
- 3 removed from queue
- 6 added to queue
- 4 removed from queue
- 5 removed from queue
- 6 removed from queue
- 7 added to queue
- 7 removed from queue

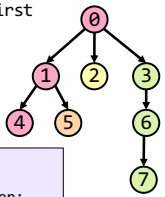



```
queue.add(root)
while queue not empty:
  curr = queue.remove()
  ... # process curr
  for child in curr.children:
    queue.add(child)
```

0 1 2 3 4 5 6 7

traversePreOrderDepthFirst

- start recurse(0)
- start recurse(1)
- start recurse(4)
- finish recurse(4)
- start recurse(5)
- finish recurse(5)
- finish recurse(1)
- start recurse(2)
- finish recurse(2)
- start recurse(3)
- start recurse(6)
- start recurse(7)
- finish recurse(7)
- finish recurse(6)
- finish recurse(3)
- finish recurse(0)




```
def recurse(curr):
  ... # process curr
  for child in curr.children:
    recurse(child)
recurse(root)
```

0 1 4 5 2 3 6 7