Outline

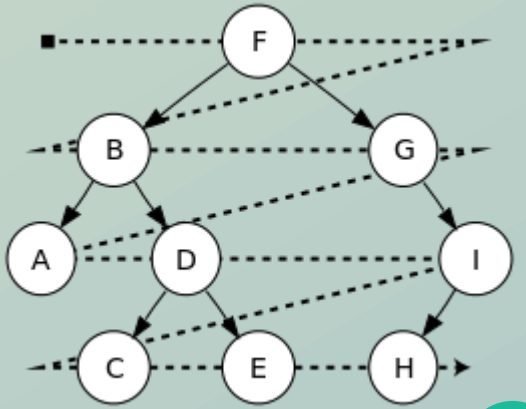
- application of stacks and queues: data structure traversal
- application of queues: simulation of customer queue
- random numbers
- trees
- expression trees
- binary tree traversal

Traversal of data structures

- in a linked list, each node has a link to at most one other node
- in a doubly-linked list, each node has a link to at most two other nodes
- there are more general data structures in which nodes can have links to multiple other nodes
- these data structures go by different names, including trees and graphs
- in some cases, we need to have a program start at one node and visit all the other nodes in the data structure: tree traversal or graph traversal

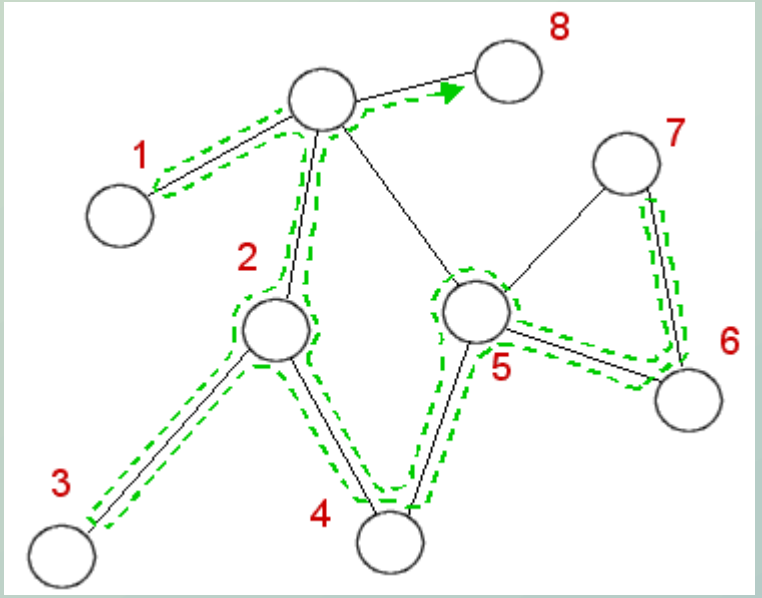
Breadth-first traversal

- **in general, I can start from a given node and put into a queue all the nodes it is connected to**
- **then, I repeatedly remove one node from the queue, visit it, and put into the queue all the nodes this new node is connected to**
- **if I keep doing this, staying away from nodes that have already been visited, I will eventually visit all connected nodes**
- **this is called breadth-first traversal:**
  - visually arranging the first node at the top
  - the nodes it is connected to right below it,
  - the nodes they are connected to right below them,
  - nodes are visited in left-to-right and top-to-bottom order



Depth-first traversal

- for a traversal, I could push the nodes on a stack instead of adding them to a queue
- then, the node I will visit next is the node I put on the stack most recently
- that means visiting every node connected to the most recently visited node, before visiting any node that was pushed onto the stack earlier
- this is called depth-first traversal because the traversal tends to go top-to-bottom, then climb back up and explore the next unvisited branch to the right



Double-ended queues

- sometimes, it is useful to be able to add and remove elements at either end of a queue
- this double-ended queue, or deque (pronounced either “D-Q” or “deck”), can do everything either a stack or a queue can do
- implementation, using either arrays or doubly-linked lists, is similar to the implementation of either a stack or a queue

Simulation of a customer queue

- **similar but not identical to the one in section 4.6 of the book**
  - the example in the book is menu driven, this is random
- **random arrivals: during each minute, one passenger arrives with a given probability, arrivalRate**
- **e.g. if arrival rate is 0.5, on average a person arrives every two minutes**
- **this is computed by testing**

```
if (Math.random() < arrivalRate) { // customer arrives
```

- **one agent: processing a customer requires a time that is uniformly random between 0 and some defined maximum number of minutes**
- **every minute,**
  - check to see whether to add customers to each queue
  - check to see whether the agent is done taking care of the current customer
  - if so, select the next customer if any
  - update the time
- **once a customer is selected, that customer's statistics (waiting time) is updated**

random numbers

- tossing a fair coin is truly random -- there is no way to predict what the next toss will give
- computers do not find it easy to toss coins, unless they have specialized hardware
- instead, get a new number by applying to a starting number (the seed) a complicated function
- this new number is a pseudo-random value: it is computed, and therefore not random, but without knowing the exact function and the seed, there is no easy way to predict the new number from the old, and so it looks like a sequence of random numbers

random number seed

- a pseudo-random number is computed by applying a complicated function to a seed
- the initial seed can be a fixed value (e.g. 1), to give a repeatable sequence of random numbers
  - repeatability is good for debugging
- or the initial seed can be selected almost at random, e.g. the time of day when the program is run
  - this gives a different sequence each time, which looks more random
- each random number is the seed for the next random number

random numbers in the Java standard library

- Java's `Math.random()` returns a double uniformly distributed between 0 and 1
  - implemented by creating a new `Random` object
- the Java `Random` class by default is initialized to the current day and time, but the programmer can explicitly specify the seed

trees

- imagine having to store, in an organized fashion, all the descendants of a given person
- as in this family tree:

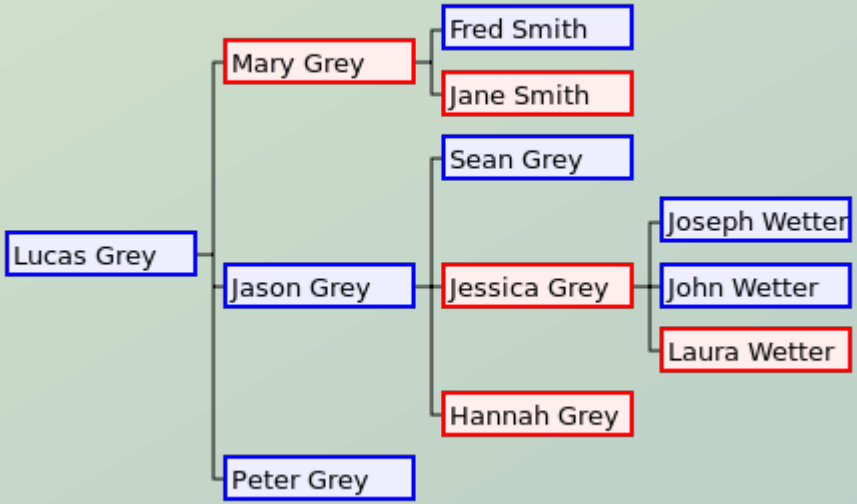


image by Derrick Coetzee, in entry “family tree” in Wikipedia

trees in computer science

- trees can hold any kind of data, even numbers:

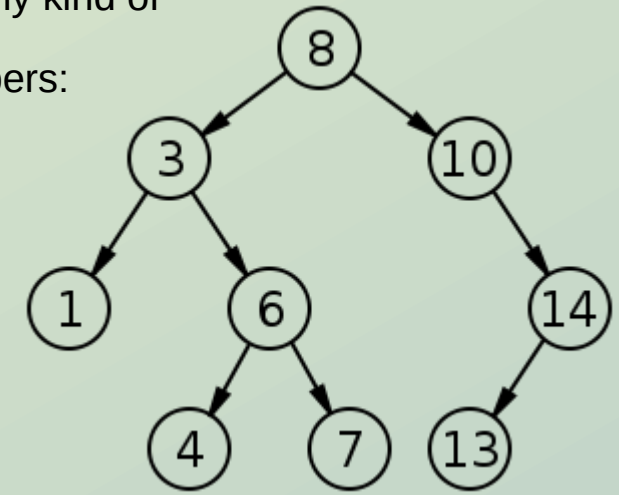


image by Derrick Coetzee, in entry “Search Tree” in Wikipedia

- a node in a tree is similar to a node in a linked list, except:
  - a linked list node has a reference to zero or one link nodes, whereas
  - a tree node has a reference to zero or more other tree nodes
- later we will consider how trees are implemented

tree properties

- a tree has one root node, from which all other nodes can be reached by following links
- each node in a tree, except the root node, has exactly one parent
- each node in a tree can have zero or more children
- the other children of a node's parent are the node's siblings
- nodes are in a hierarchical relationship:
  - node X is an ancestor of another node Y, or
  - node X is a descendant of another node Y, or
  - node X is on a different branch than node Y
- nodes without children are leaf nodes
- nodes with children and a parent are interior nodes

tree properties exercise

- in-class exercise (everyone together): identify the

- root node
- interior nodes
- leaf nodes
- parent of node 6
- children of node 6
- ancestors of node 9
- descendants of node 7

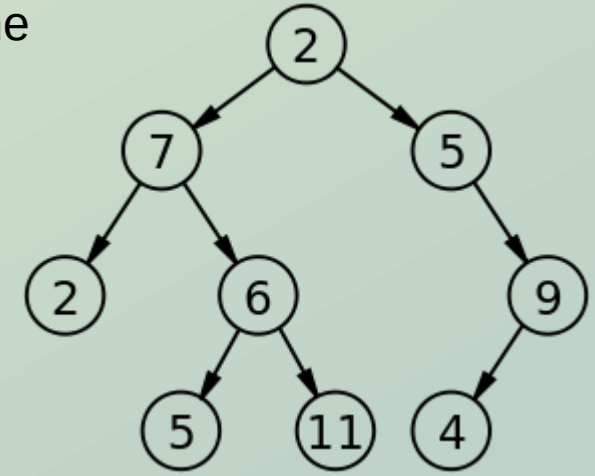


image by Derrick Coetzee, in entry “Tree (data structure)” in Wikipedia

more tree definitions

- a subtree is a node with all its descendants
- the level of a node is the number of its ancestors (including itself), so e.g. the root is always at level 1:
  - the children of the root are at level 2
  - their children are at level 3
  - the children of a node at level n are at level n+1
- sometimes the word depth is used instead of level
  - sometimes the level or depth does not include the node itself, so the root is at level 0
- the height of a tree is the maximum depth of any node in the tree
  - this may also be called the depth of the tree
- in a binary tree each node has at most two children
- likewise, in a ternary tree each node has at most three children

types of binary trees

- in a balanced binary tree the height of each subtree of every node differs by at most one

  (this is a recursive definition)

  there are also other definitions of balanced binary trees
- in an expression tree, each internal node contains an operator, and each leaf node is an operand (value)
  - subtrees are evaluated first, yielding a value. Once all its children have been evaluated, we can evaluate a node
- in a binary search tree each node has a value greater than every node in its left subtree, and less than every node in its right subtree

more types of binary trees

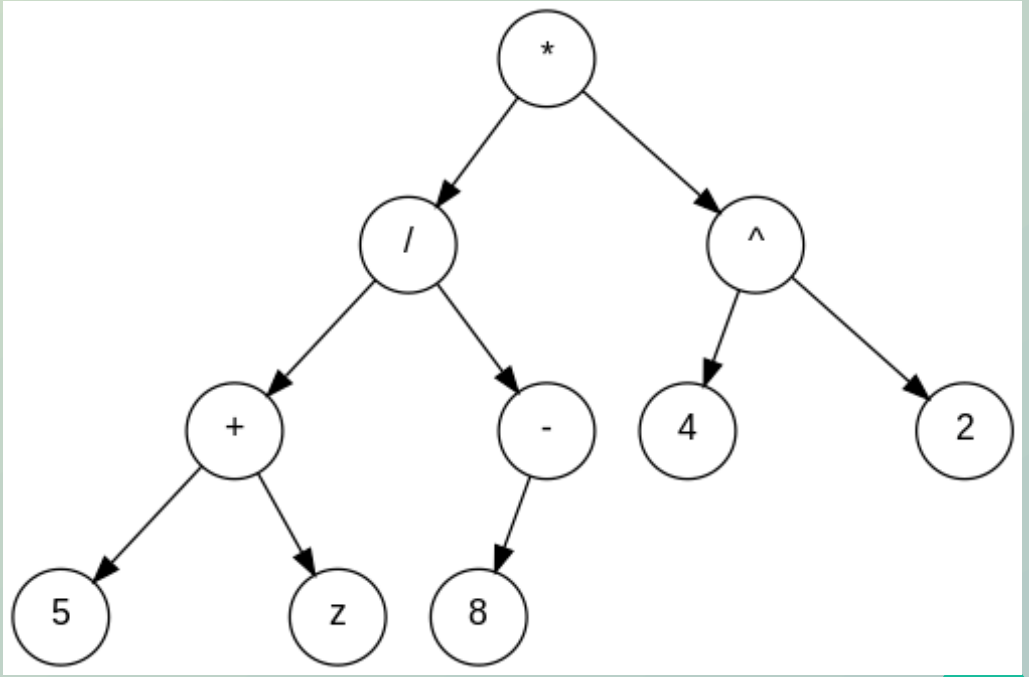
- a perfect binary tree of height n has $2^n - 1$ nodes, of which 2^{n-1} are leaves
- a complete binary tree is a perfect binary tree up to the last level. All the leaves at the lowest level are as far to the left as possible
- in a full binary tree, each node has either 0 or two children (non-leaf nodes are full)
- in a Huffman tree, each internal node has two children. Each leaf node represents something to encode, such as a letter. The binary string to encode the letter is found by going down the tree from the root to the leaf. Each time we go left contributes a 0 bit to the code, and each time we go right contributes a 1 bit to the code.

- perfect binary trees and complete binary trees are always balanced
- a Huffman tree is always a full binary tree

expression trees

- an expression tree is a tree where:
  - every node with children is an operator
  - every leaf node is an operand
- each operator operates on the values of its children
- an expression tree corresponding

  to ((5 + z) / -8) * 4 ^ 2):



binary tree traversals

- if we want to visit each node in a binary tree in depth-first order, we have a choice of how to do it:
  - preorder traversal: visit the root node, then recursively visit the left subtree, then the right subtree
  - inorder traversal: recursively visit the left subtree, then visit the root node, then recursively visit the right subtree
  - postorder traversal: recursively visit the left subtree, then the right subtree, then the root node
- what does “visit a node” mean?
  - could mean printing the value
  - could mean saving the value in a data structure

binary tree traversal exercises

- do pre-order, inorder, and post-order traversals of these trees

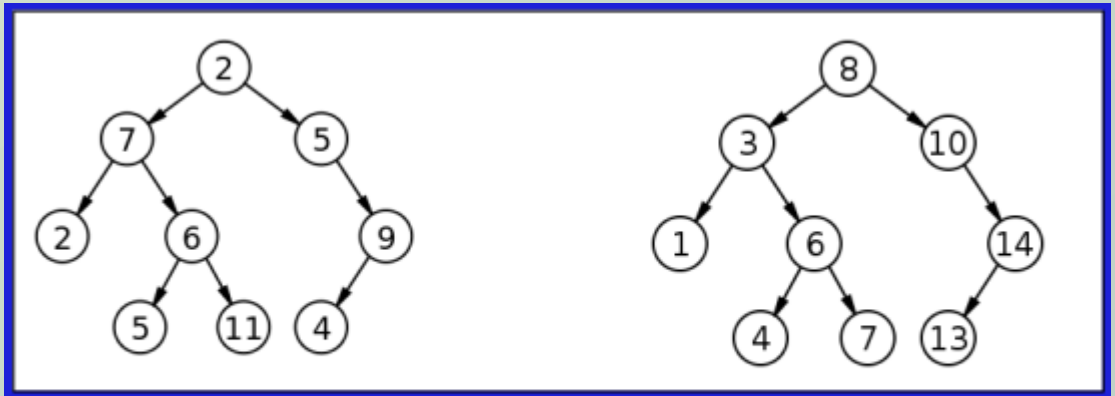


image by Derrick Coetzee, in entry "Tree (data structure)" in Wikipedia

image by Derrick Coetzee and Booyabazooka, in entry "Binary search tree" in Wikipedia

expression tree traversal exercise

- do a pre-order, in-order, and post-order traversal of this tree