Tamara Bonaci, Adrienne Slaughter

Northeastern University

November 29, 2018

Review: Proof Techniques

Some Graph and Tree Problems

Introduction to Trees

Section 1

Review: Proof Techniques

Proving Correctness

How to prove that an algorithm is correct?

Proof by:

Counterexample (*indirect proof*)

Induction (*direct proof*)

Loop Invariant

Other approaches: proof by cases/enumeration, proof by chain of i s, proof by contradiction, proof by contrapositive

Proof by Counterexample

Searching for counterexamples is the best way to disprove the correctness of some things.

Identify a case for which something is NOT true

If the proof seems hard or tricky, sometimes a counterexample works   Sometimes a counterexample is just easy to see, and can shortcut a proof

If a counterexample is hard to find, a proof might be easier

Proof by Induction

Failure to find a counterexample to a given algorithm does not mean "it is obvious" that the algorithm is correct.

Mathematical induction is a very useful method for proving the correctness of recursive algorithms.

Prove base case

Assume true for arbitrary value n

Prove true for case $n + 1$

Proof by Loop Invariant

Built o proof by induction.

Useful for algorithms that loop.

Formally: find loop invariant, then prove:

Define a Loop Invariant

Initialization

Maintenance

Termination

Informally:

Find *p*, a loop invariant

Show the base case for *p*

Use induction to show the rest.

Proof by Loop Invariant Is…

Invariant: something that is always true

A er finding a candidate loop invariant, we prove:

Initialization: How does the invariant get initialized?

Loop Maintenance: How does the invariant change at each pass through the loop?

Termination: Does the loop stop? When?

Steps to Loop Invariant Proof

A er finding your loop invariant:

- Initialization
  - Prior to the loop initiating, does the property hold?

Maintenance

A er each loop iteration, does the property still hold, given the initialization properties?

Termination

A er the loop terminates, does the property still hold? And for what data?

Things to remember

Algorithm termination is necessary for proving correctness of the entire algorithm.

Loop invariant termination is necessary for proving the behavior of the given loop.

Summary

Approaches to proving algorithms correct

Counterexamples

Helpful for greedy algorithms, heuristics

Induction

- Based on mathematical induction
- Once we prove a theorem, can use it to build an algorithm

Loop Invariant

- Based on induction
- Key is finding an invariant

Lots of examples

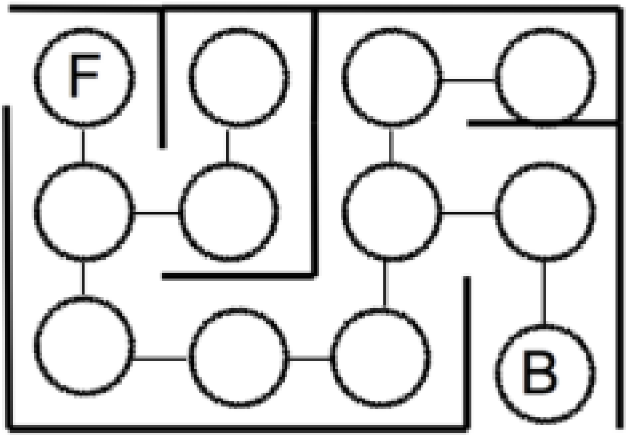
Section 2

Some Graph and Tree Problems

Outdoors Navigation

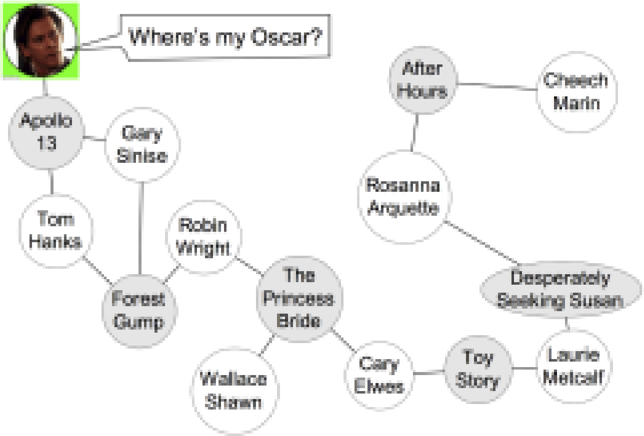
stern University

Indoors Navigation



Telecommunication Networks

Social Networks


Where's my Oscar?
Apollo 13
Gary Sinise
Tom Hanks
Robin Wright
Forest Gump
The Princess Bride
Wallace Shawn
Cary Elwes
Toy Story
Laurie Metcalf
Desperately Seeking Susan
Rosanna Arquette
After Hours
Cheech Marin

Section 3

Introduction to Trees

What is a Tree?



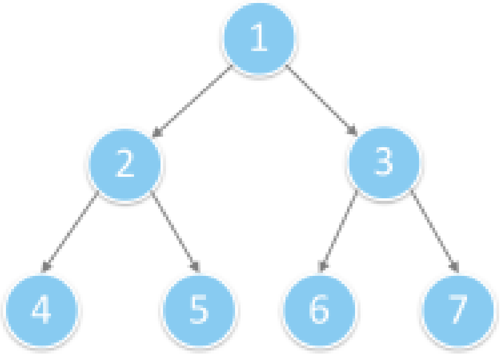
Tree - a directed, acyclic structure of linked nodes

- Directed - one-way links between nodes (no cycles)
- Acyclic - no path wraps back around to the same node twice (typically represents hierarchical data)

Tree Terminology: Nodes

Tree - a directed, acyclic structure of linked nodes

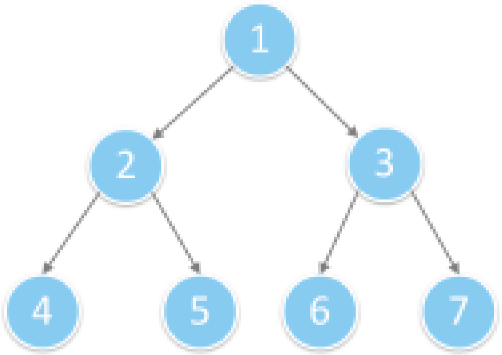
Node - an object containing a data value and links to other nodes   All the blue circles



Tree Terminology: Edges

Tree - a directed, acyclic structure of linked nodes

Edge - directed link, representing relationships between nodes   All the grey lines



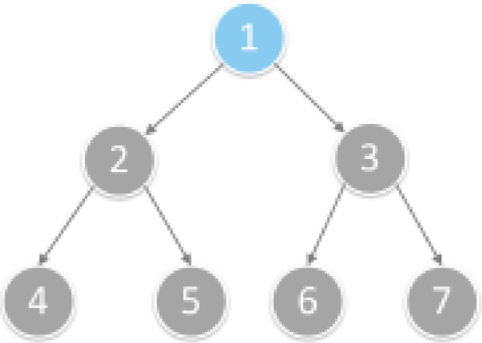
Root Node

Tree - a directed, acyclic structure of linked nodes

Root - the start of the tree tree)

The top-most node in the tree

Node without parents

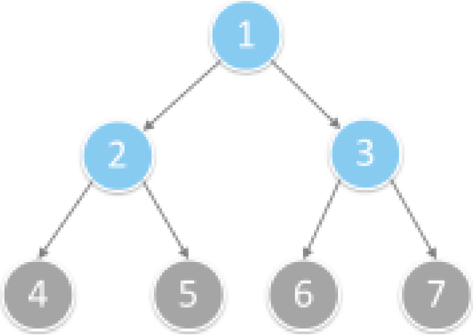


Parent Nodes

Tree - a directed, acyclic structure of linked nodes

Parent (ancestor) - any node with at least one child

The blue nodes

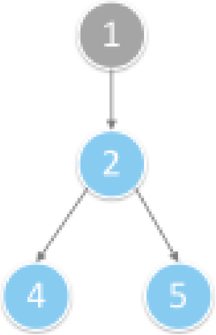


Children Nodes

Tree - a directed, acyclic structure of linked nodes

Child (descendant) - any node with a parent

The blue nodes

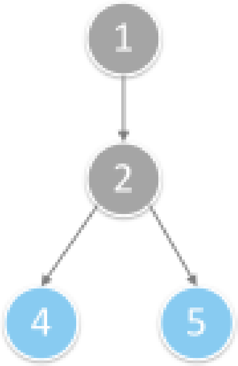


Sibling Nodes

Tree - a directed, acyclic structure of linked nodes

Siblings - all nodes on the same level

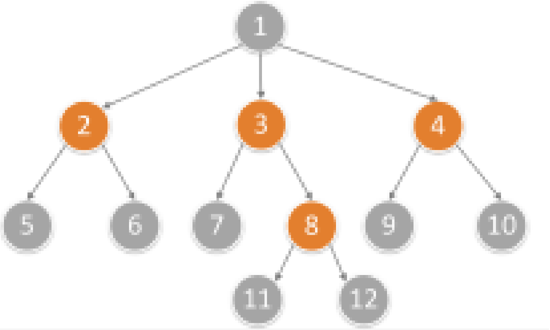
The blue nodes



Internal Nodes

Tree - a directed, acyclic structure of linked nodes

Internal node - a node with at least one children (except root)   All the orange nodes

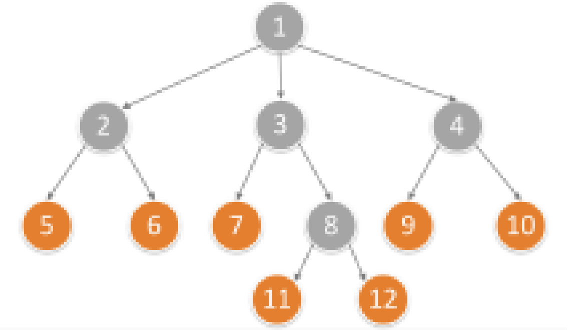


Leaf (External) Nodes

Tree - a directed, acyclic structure of linked nodes

External node - a node without children

All the orange nodes

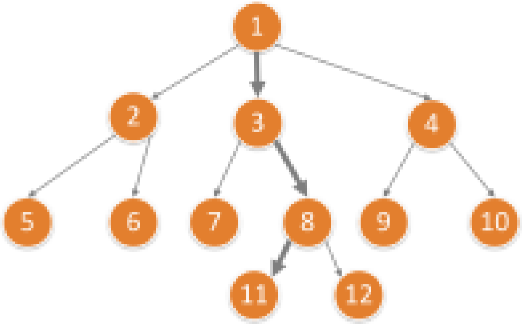


Tree Path

Tree - a directed, acyclic structure of linked nodes

Path - a sequence of edges that connects two nodes

All the orange nodes

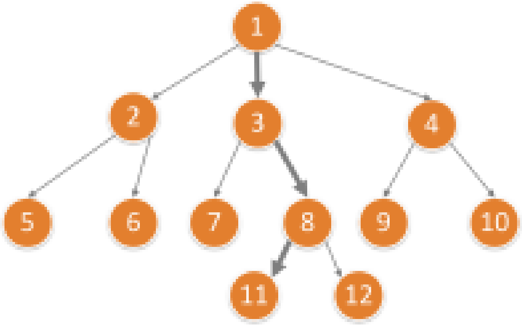


Node Level

Node level - 1 + [the number of connections between the node and the root]

The level of node 1 is 1

The level of node 11 is 4



Node Height

Node height - the length of the longest path from the node to some leaf

- The height of any leaf node is 0
- The height of node 8 is 1
- The height of node 1 is 3
- The height of node 11 is 0

Tree Height

Tree height

The height of the root of the tree, or

The number of levels of a tree -1.

The height of the given tree is 3.

What is Not a Tree?

Problems:

Cycles: the only node has a cycle

No root: the only node has a parent (itself, because of the cycle), so there is no root

What is Not a Tree?

Problems:

- Cycles: there is a cycle in the tree
  - Multiple parents: node 3 has multiple parents on di erent levels

What is Not a Tree?

Problems:

- Cycles: there is an undirected cycle in the tree
  - Multiple parents: node 5 has multiple parents on di erent levels

What is Not a Tree?

Problems:

Disconnected components: there are two unconnected groups of nodes

Summary: What is a Tree?

A tree is a set of nodes, and that set can be empty

If the tree is not empty, there exists a special node called a root

The root can have multiple children, each of which can be the root of a subtree

Section 4

Special Trees

Special Trees

Binary Tree

Binary Search Tree

Balanced Tree

Binary Heap/Priority ƒeue

Red-Black Tree

Binary Trees

Binary tree - a tree where every node has at most two children

Binary Search Trees

Binary search tree (BST) - a tree where nodes are organized in a sorted order to make it easier to search

At every node, you are guaranteed:

All nodes rooted at the le child are smaller than the current node value

All nodes rooted at the right child are smaller than the current node value

Example: Binary Search Trees?

Binary search tree (BST) - a tree where nodes are organized in a sorted order to make it easier to search

Le tree is a BST

Right tree is not a BST - node 7 is on the le hand-side of the root node, and yet it is greater than it

Example: Using BSTs

Suppose we want to find who has the score of 15…

Example: Using BSTs

Suppose we want to find who has the score of 15:

- Start at the root

Example: Using BSTs

Suppose we want to find who has the score of 15:

- Start at the root
- If the score is > 15, go to the le

Example: Using BSTs

Suppose we want to find who has the score of 15:

- Start at the root
- If the score is > 15, go to the le
- If the score is < 15, go to the right

Example: Using BSTs

Suppose we want to find who has the score of 15:

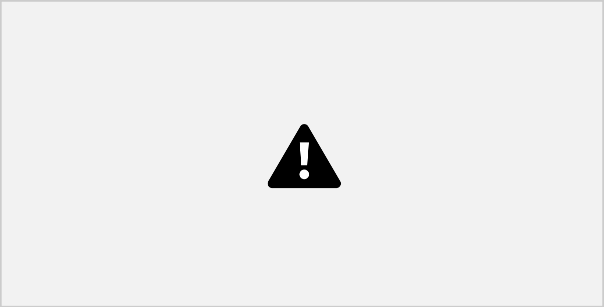
- Start at the root
- If the score is > 15, go to the le
- If the score is < 15, go to the right
- If the score == 15, stop

Balanced Trees

Consider the following two trees. Which tree would it make it easier for us to search for an element?

Balanced Trees

Consider the following two trees. Which tree would it make it easier for us to search for an element?



Observation: height is o en key for how fast functions on our trees are. So, if we can, we want to choose a balanced tree.

Tree Balance and Height

How do we define balance?

- If the heights of the le   and right trees are balanced, the tree is balanced, so:

Tree Balance and Height

How do we define balance?

- If the heights of the le   and right trees are balanced, the tree is balanced, so:
  - |(height(le) − height(right))|

Tree Balance and Height

How do we define balance?

- If the heights of the le and right trees are balanced, the tree is balanced, so:
    - |(height(le) − height(right))|
- Anything wrong with this approach?

Tree Balance and Height

How do we define balance?

- If the heights of the le and right trees are balanced, the tree is balanced, so:
  - |(height(le) − height(right))|
- Anything wrong with this approach?
- Are these trees balanced?

Tree Balance and Height

Observation: it is not enough to balance only root, all nodes should be balanced.

The balancing condition: the heights of all le and right subtrees di er by at most 1

Example: Balanced Trees?

- The le tree is balanced.
- The right tree is not balanced. The height di erence between nodes 2 and 8 is two.

Section 5

Tree Traversals

Tree Traversals

Challenge: write a recursive function that starts at the root, and prints out the data in each node of the tree below

Tree Traversals

Tree Traversals

Tree Traversals

Tree Traversals

Tree Traversals

Tree Traversals

Tree Traversals

Tree Traversals

Tree Traversals

Tree Traversals

Tree Traversals

Summary:

Tree Traversals

Tree Traversals

Challenge: write a non-recursive function that starts at the root, and prints out the data in each node of the tree below

Tree Traversals

Tree Traversals

Tree Traversals

Tree Traversals

Tree Traversals

Tree Traversals

Tree Traversals

Tree Traversals

Tree Traversals

Tree Traversals

Tree Traversals