COMPSCI 250: Introduction to Computation

Lecture #23: Recursion on Trees

David Mix Barrington and Ghazaleh Parvini

27 October 2023

Recursion on Trees

- Trees to Represent Expressions
- A Recursive Definition of Expression Trees
- Types of Expressions
- Prefix, Infix, and Postfix Strings for Expressions
- Parsing and Evaluating Expressions
- The Definition of Truth
- Call Trees for Algorithms

A Recursive Tree Definition

- (Note: Five slides repeated from Lecture #22)
- A single node, with no edges, is a rooted tree and the node is its root.
- We can make a rooted tree out of one or more existing rooted trees plus a new node x. The root of the new tree is x, and we add edges from x to the roots of each of the existing trees.
- The only possible rooted trees are those made by the two rules above.

Induction on Rooted Trees

- This is a recursive definition of rooted trees.
- As with our other recursively defined types, we now have a new Law of Mathematical Induction for rooted trees.
- If we prove P(T) whenever T has only one node, and that P(T) is true when T is made from subtrees U_1, U_2,..., U_k and P(U_i) is true for all i, then we may conclude that P(T) is true for any rooted tree T.

A Theorem About Rooted Trees

- Let's use this induction rule to prove a theorem.

- **Theorem**: If T is any rooted tree with n nodes and e edges, then e = n - 1.

- Base Case: If T is a one-node tree, then e = 0 and n = 1 so e = n - 1 is true.

- Now we have to set up the inductive step.

A Theorem About Rooted Trees

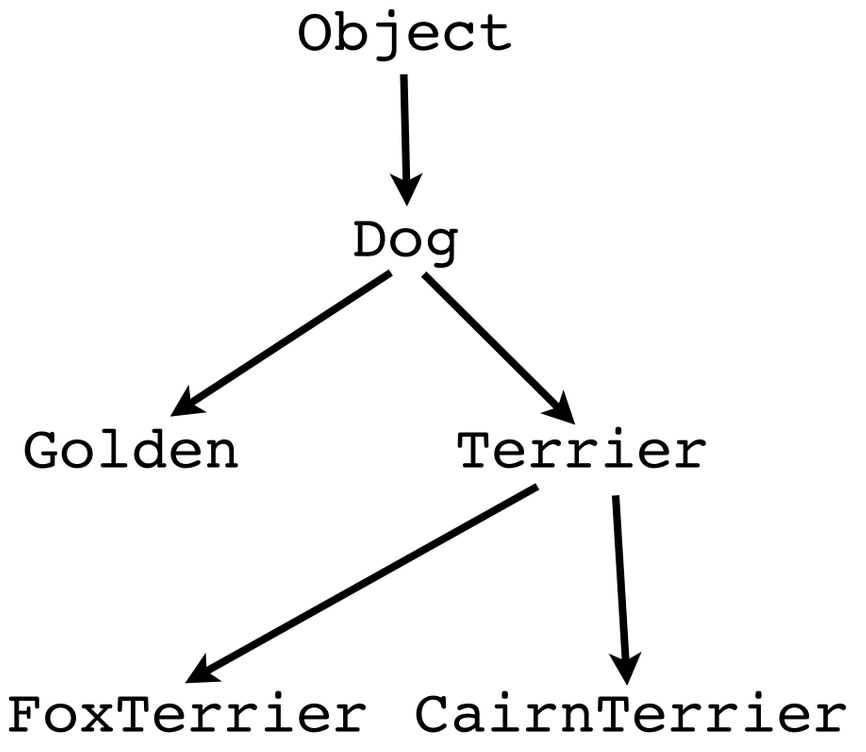
- Inductive Step: Let T be made by the second rule from U_1, U_2,..., U_k and say that each of the U_i's has n_i nodes and e_i edges, so that $e_i = n_i - 1$ by the IH.
- T has all the nodes and edges from all the subtrees, plus one new node (its root) and k new edges (one from its root to each of the existing roots).

A Theorem About Rooted Trees

- So n, the number of nodes in T, is the sum of the n_i's plus 1.
- And e, the number of edges in T, is the sum of the e_i's plus k.
- The sum S of the e_i's is the sum of the n_i's minus k, so $e = S + k$ and $n = (S + k) + 1$, and therefore $= n - 1$.
- We've completed the inductive step and thus proved our P(T) for all rooted trees T.

Trees to Represent Expressions

- Trees are useful for representing collections of objects in a hierarchical structure, where every object except one has a unique "parent" object.
- We've mentioned people in an organization, classes in an inheritance hierarchy, and files in a directory/folder system.



Trees to Represent Expressions

- **Expressions** are collections of **atomic values** connected by **operators**.
- We've seen **boolean expressions** in the first part of the course, and they are dealt with in more detail in Excursion 9.2, which will be the subject of Discussion #7 next week.
- There are also **arithmetic expressions** as in Java.
- Even whole programs can be thought of as expressions.

Trees to Represent Expressions

- Operators can be **unary**, meaning that they take one argument (like $\neg$ or -) or **binary**, meaning that they take two (like $\wedge$, $\vee$, $+$, or $\times$). In general we could also have ternary, 4-ary, or k-ary operators for any natural k.
- Our expressions are trees because each proper subexpression has exactly one parent. (The entire expression, the **root** of the tree, has no parent.)

A Definition of Expression Trees

- We can give a recursive definition of expression trees that is very similar to our other recursive definitions:
- (1) A single atomic value is an expression tree.
- (2) A k-ary operator, acting on a sequence of k expression trees, gives an expression tree.
- (3) The only expression trees are those given by rules (1) and (2).

Induction on Expression Trees

- Rule (3) gives us a Law of Induction for expression trees.
- If we prove that P(a) is true for any atomic value a, and prove that P(E) is true whenever E is any k-ary operator acting on any k expression trees E_1,..., E_k such that P(E_i) is true for all i, then we have proved that P(E) is true for *any* expression tree E.

Types of Expressions

- In **boolean expressions** the atomic values are 0 and 1 (false and true), or variables ranging over those values, and the operators are $\neg$, $\wedge$, $\vee$, $\oplus$, $\rightarrow$, and $\leftrightarrow$.
- In Excursion 9.2 we use just $\wedge$,$\vee$, and $\neg$.
- The $\neg$ operator is unary and all the other operators are binary.

Types of Expressions

- In Java **arithmetic expressions**, the atomic values come from one of the number types, and the operators are +, ×, -, /, % (for integer types), and so forth.
- The - operator can be either unary or binary, while all the others are binary.
- We'll consider our own arithmetic expressions to use just +, ×, -, and /.

Clicker Question #1

- Remember that the **depth** of a tree node is the length of the path from the root to the node. If I have an arithmetic expression such that (i) all its operators are binary, (ii) the maximum depth is 3, and (iii) there are 6 total leaves, how many leaves does the tree have at depth 3?
-

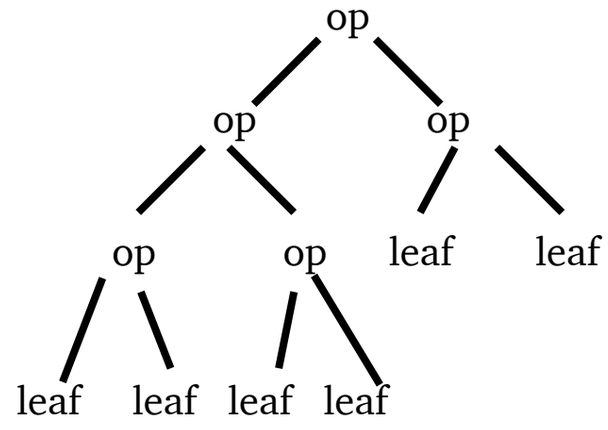
(a) 2 (b) 5
(c) 4 (d) 6

Not the Answer

Clicker Answer #1

- Remember that the **depth** of a tree node is the length of the path from the root to the node. If I have an arithmetic expression such that (i) all its operators are binary, (ii) the maximum depth is 3, and (iii) there are 6 total leaves, how many leaves does the tree have at depth 3?

- (a) 2
  (b) 5
  (c) 4
  (d) 6

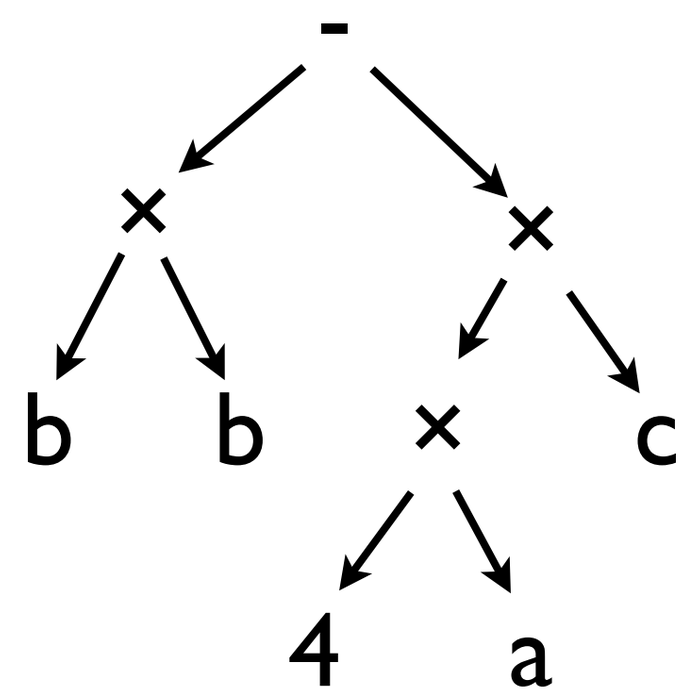


Types of Expressions

- Later in Chapter 5 and 14 we'll work with **regular expressions**, where the atomic values are letters and $\emptyset$, and there is one unary operator * and two binary operators $+$ (for union) and $\cdot$(for concatenation of languages).
- An induction over all regular expressions will have *two* base cases and *three* inductive cases.

Prefix, Infix, and Postfix Strings

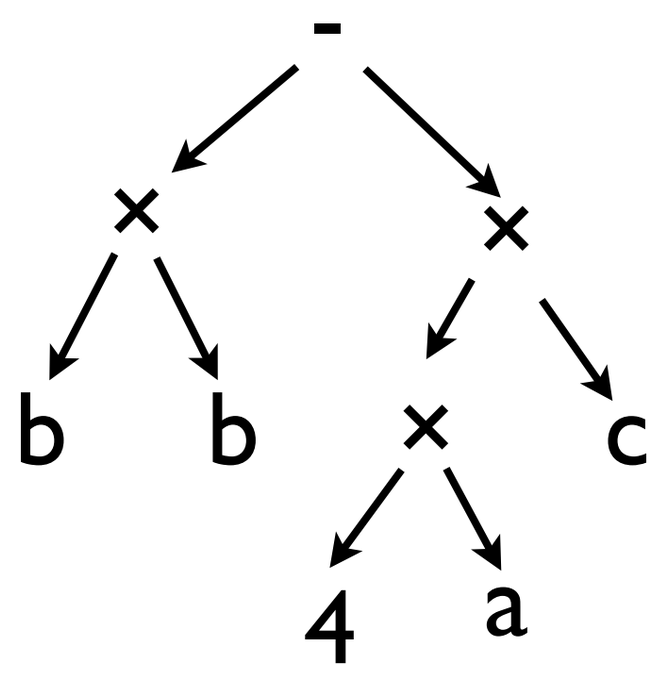
- There are three ways to represent a boolean or arithmetic expression by a string.



- For an example, take the arithmetic expression “b×b - 4×a×c” that occurs in the quadratic formula. The expression tree for this formula has nine nodes -- the root is a - operator, its children are × operators, and the leaves are atomic values a, b, c, and 4.

Prefix, Infix, and Postfix Strings

- "b×b - 4×a×c" is the **infix** string for this expression. The **prefix** string for it is "-×bb××4ac" and the **postfix** string is "bb×4a×c×-".



- Note that each string contains the same atomic and operator symbols, just in a different order. (Some infix strings also contain parentheses, making them longer than the other two.)

Clicker Question #2

- What are the postfix and prefix strings for the arithmetic expression whose infix string is given by “((a + b) × a) × (b + a))”?
  - (a) postfix “ababa+×+×”, prefix “×+ab×a+ba”
  - (b) postfix “ab+a×ba+×”, prefix “××++ababa”
  - (c) postfix “ab+aba+××”, prefix “×+ab×a+ba”
  - (d) postfix “ab+a×ba+×”, prefix “××+aba+ba”

Not the Answer

Clicker Answer #2

- What are the postfix and prefix strings for the arithmetic expression whose infix string is given by “((a + b) × a) × (b + a))”?
  - (a) postfix “ababa+×+×”, prefix “×+ab×a+ba”
  - (b) postfix “ab+a×ba+×”, prefix “××++ababa”
  - (c) postfix “ab+aba+××”, prefix “×+ab×a+ba”
  - (d) postfix “ab+a×ba+×”, prefix “××+aba+ba”

Prefix, Postfix, and Infix Strings

- We can recursively define each of the three strings from the expression.
- For example, “the postfix string of an atomic value is itself, and the postfix string of an operator applied to k subexpressions is the concatenation of the postfix strings for the subexpressions, followed by the symbol for the operator”.
- The other two definitions are similar.

Parsing, Evaluating Expressions

- A major problem in computer science is to take a string and **parse** it, which means to determine the expression tree that it represents.

- A compiler must take a string in a computer language and determine (1) whether it is a valid program, (2) how the string is broken down into language parts, and (3) what the meaning of the resulting program is. You'll see more about parsing in courses like COMPSCI 501 and 410.

Parsing, Evaluating Expressions

- The most common thing to do with an expression is to **evaluate** it, which means to determine its value by applying the operators to the atomic values.
- The basic evaluation algorithm for an expression is “If the expression is an atomic value, return the value. If it is an operation applied to subexpressions, evaluate each subexpression, apply the operator to the results, and return the new result”.

Parsing, Evaluating Expressions

- Parsing a program is just evaluating an expression over a complex set of values and operators.
- Here the “value” is the meaning of each subprogram. We might, for example, consider the meaning of each subprogram to be the machine-language program that implements it.

Clicker Question #3

- Consider a Java arithmetic expression where the atomic elements are naturals, containing all of the operators $+, \times$, and %. Which of the following is *true*?
- (a) an expression's value could evaluate to something other than a natural
- (b) if the expression contains no 0, it always produces a value
- (c) if there are no parentheses, the last operator evaluated is a +
- (d) if the atomic elements are all odd, the expression's value, if any, must be odd

Not the Answer

Clicker Question #3

- Consider a Java arithmetic expression where the atomic elements are naturals, containing all of the operators +, ×, and %. Which of the following is *true*?

- (a) an expression's value could evaluate to something other than a natural none of these operations on naturals produce a non-natural, though some don't give any value

- (b) if the expression contains no 0, it always produces a value 2 % (2 % 2)

- (c) if there are no parentheses, the last operator evaluated is a + hierarchy of operations

- (d) if the atomic elements are all odd, the expression's value, if any, must be odd 1+1=2

The Definition of Truth

- In this course we have been informal about what it means for a logical statement to be true or false in a given situation.
- But to do **metamathematics** (mathematics *about* mathematics), we would need a formal definition of what a **model** for a statement is, and whether a given statement is true in a given model.

The Definition of Truth

- The Polish logician Alfred Tarski gave a formal **definition of truth** in 1933, using induction on the definition of logical statements.
- A statement is built up from atomic statements using boolean operators and quantifiers. The truth value of atomic statements is assumed to be given in any particular model.

The Definition of Truth

- The truth of more complex statements can be defined by inductive rules, such as "∃x:P(x) is true if and only if there is an object z such that P(z) is true", in terms of the truth of simpler statements.
- Tarski's definition gives a justification for our four quantifier proof rules.

Call Trees for Algorithms

- If we have a method that makes recursive calls upon itself, but eventually terminates, we can make a diagram called a **call tree** that represents all the recursive calls.

- A node in the call tree represents a call to the method, and node x has node y as a child if the call y is made by the version of the method called by call x.

Call Trees for Algorithms

- The call tree is finite if every version eventually terminates, and the leaves of the call tree represent calls that cause no recursion.
- If we prove that every leaf call terminates with the right answer, and that every non-leaf call terminates with the right answer if all of its child calls do so, then by the Law of Induction for trees, we have proved that every recursive call (every call tree) corresponds to a version of the method that terminates with the right answer. This is how we prove that the method is correct.