SOFT1002
Lecture 21: Trees and Composite Design Pattern

School of Information Technologies

Today’s Lecture

- Trees
- Recursive algorithms on trees
- Code for trees and their algorithms
- Design Patterns
- Composite Design Pattern
- Read Kingston Chapter 19

Trees

- We like to organize things in hierarchical manner.
- Trees are used to represent various hierarchical interrelationships.

Some Application of Trees

- Family Trees
- Organizational chart
- Sports tournament chart
- Linguistics (structure of language)
- Computer File Systems (directories/folders)

Trees

- A **rooted tree** consists of a root node and a finite set of subtrees, which are themselves rooted trees.
- Note: it is allowed to have no subtrees (an empty set)
- In that case, the tree is just a single node!

Example

- Different subtrees can have different shapes
- Each node can have a value; in some applications this is a string; in others it is another object; sometimes, it’s a number
Terminology

Children of node X: the root nodes of the sub-trees of the tree (or sub-tree, or sub-sub-tree etc) whose root is X

Descendants of X: X, or its children, or its children’s children, or ...

Parent of Y: the node X such that Y is among the children of X

Note: every node except the root has a parent

Leaf node: any node with no children

Examples

The node with 14 is the parent of the node with 23
The node with 16 is a descendant of the node with 14
The node with 18 is a leaf; the node with 23 is not a leaf

More terminology

Depth of node X: number of nodes including the root and X on the longest path from the root down to X

Height of node X: number of nodes (including X and the leaf) on the longest path from X down to a leaf

Height of a tree: height of the root node in the tree

Warning: these definitions are not standard; different books give different meanings to the words.

Examples

The node with 14 has depth 2
The node with 14 has height 3
Any leaf has height 1
The root has depth 1

Recursive algorithms on trees

Base case: tree with only one node (the root, with no subtrees)
Recursion step: how does the algorithm act on a tree, related to how it acts on the subtrees

Tree height

Height of tree with no subtrees: 1

Height of tree: $1 + \max(\text{Height of any subtree})$
Exercise: Count nodes

- countnodes = 11
- Count nodes of tree with no subtrees:
  - Answer: countnodes = 1
- Count nodes of tree
  - Answer: Count nodes of tree = 1 + sum(count nodes in each subtree)

Tree Traversal

- Visiting all the nodes in a tree:
  - And do something with each
  - Eg print them, or add their values, or find the largest...
- Key issue: what order to visit them?
  - Compare with traversing a list which has an obvious order

Tree traversal

- Pre-order:
  - Each node is visited before its descendents.
- Post-order:
  - The descendents are visited before the node.

Pre-order

- For binary trees (e.g., where there are distinct left and right children) there is in-order:
  - Traverse left subtree, then visit the node, then traverse right subtree.

Post-order

Varieties of Trees

- Binary trees
- Ordered trees
- Rooted trees
- Free trees

Trees are used in representing relational structures and implementing various data structures
### Binary trees

- A binary tree is a node with a left subtree (or null) and a right subtree (or null).

  ![Binary tree example](image)

  - V3 is left child of V2;
  - V2 is right child of V0.

  Note: V4 has null instead of a right subtree.

### Traversing ordered trees

- **Pre-order:** V0, V1, V2, V3, V4, V5
- **Post-order:** V1, V3, V5, V4, V2, V0
- **In-order:** V1, V0, V3, V2, V5, V4

### Coding trees

- Unfortunately, there is no simple general-purpose library class to use when representing tree structures.
- There are JTree, TreeModel and TreeNode in Swing.
- This leaves lots of alternative approaches to coding the information and the algorithms.

  - We look at 3 main styles:
    - Use a non-standard library class
    - Implement directly using node class containing references to other nodes
    - Implement directly with a mixture of classes (Composite design pattern)

### Use a Tree class

- Kingston text refers to a Tree class.
- This allows access to subtrees through an Iterator.
- Just as List allows Iterator-based access to elements.
- Because the library is already written, the recursive method you write is in the class that has a reference to a Tree.

### Tree class API

- In `Tree`
  ```java
  /* return the value stored in the root */
  Object rootVal();
  
  /* return an Iterator */
  Iterator treeIterator();
  ```

### Code an algorithm

- One parameter is a `Tree t`.
- Inside method, obtain iterator and examine the subtrees of `t`.
  - Call the method recursively with each appropriate subtree as argument.
  - Combine the results from recursive calls to get return value for the method on `t`.

- Recall that Iterator provides
  ```java
  Boolean hasNext();
  Object next();
  ```

  - In this case, the result from `next()` should be cast to type `Tree`.
Tree height

- Where is the base case?
- Answer: It is hidden; for a single node tree, the while loop body is not executed at all!

```
public int height(Tree tree) {
  int maxHeight = 0;
  Iterator it = tree.treeIterator();
  while(it.hasNext()) {
    Tree subtree = (Tree) it.next();
    int thisHeight = height(subtree);
    if (thisHeight > maxHeight)
      maxHeight = thisHeight;
  }
  return (1+maxHeight);
}
```

Coding traversals

```
public void preorderTraversal(Tree tree) {
  visit(tree.rootVal());
  Iterator it = tree.treeIterator();
  while(it.hasNext()) {
    Tree subtree = (Tree) it.next();
    preorderTraversal(subtree);
  }
}
```

```
public void postorderTraversal(Tree tree) {
  visit(tree.rootVal());
  Iterator it = tree.treeIterator();
  while(it.hasNext()) {
    Tree subtree = (Tree) it.next();
    postorderTraversal(subtree);
  }
}
```

Binary tree nodes

- You can also define your own trees, using a Node class with references to its children
- This is especially common when representing binary trees
- Similar code is found inside some of the Java collection library classes!
- Often each node also has two pieces of information: a value and a label

```
public class Node {
  Node left;
  Node right;
  int value;
  String label;
  //methods
}
```

Methods are:
- Constructor
  public Node(Node l, Node r, int value, String label);
- Getters
  public Node getLeft();
  public Node getRight();
  public int getValue();
  public String getLabel();
- Recursive method(s)

Code an algorithm

- Write a method in the Node class
  - The target node is implicit parameter on which we recurse
  - Inside the method, call recursively on left and right
  - Always check first to be sure target != null
- Combine results of recursive calls to get return value from method

```
public int height() {
  if ((left == null) && (right == null)) {
    return 1;
  }
  if ((left != null) && (right == null)) {
    return 1 + left.height();
  }
  if ((left == null) && (right != null)) {
    return 1 + right.height();
  }
  if ((left != null) && (right != null)) {
    int leftheight = left.height();
    int rightheight = right.height();
    if (leftheight > rightheight) {
      return 1+leftheight;
    } else {
      return 1+rightheight;
    }
  }
  // never reached, needed for Java compiler return 1;
}
```

Tree height

Base case
Rewritten code

- By careful thought about null cases and default values for those, one can often rewrite the code to have less repetition

```java
public int height() {
    int leftheight = 0;
    int rightheight = 0;
    if (left != null) {
        leftheight = left.height();
    }
    if (right != null) {
        rightheight = right.height();
    }
    if (leftheight > rightheight) {
        return 1+leftheight;
    } else {
        return 1+rightheight;
    }
}
```

Composite Design Pattern

- In many situations, we want tree structures where the different nodes have very different characteristics and structures, but also there is some common behaviour
  - For example, in a GUI, there are buttons, textfields, etc, arranged in nested panels
  - The key is to use a Composite design pattern

Design Patterns

- Description of communication objects and classes that are customized to solve a general design problem in a particular context.
- Accumulation of experience by clever designers
  - Useful terminology for communication among designers and coders
  - Design pattern is to design as code cliché is to code

Composite Design Pattern

- Represent part-whole hierarchies as objects.
- The composite pattern defines a common interface for a tree-like hierarchy of objects. Some of these objects (composite objects) may be composed of other objects from the same hierarchy.
  - i.e., ignore the difference between composite objects and individual objects

```
public class AbstractComposite {
    public void operator() {}
    public void addComponent(CompositeComponent c) {}
    public void removeComponent(CompositeComponent c) {}
    public Component getChild(int i) {}
}
```
Example: Expression Tree

- An expression tree shows the structure of an algebraic expression.
- The above tree structure gives the structure of \( \sqrt{b^2 - 4ac} \).

How can we represent the Expression Tree?

- Each node is an object, but it is not helpful if the tree contains different kind of nodes.
- What are the common methods in these nodes?
  - public double eval();
- How will they differ?
  - Some nodes contain literals → eval() will simply return the value
  - Some nodes contain expressions (addition/subtraction) → eval() will need prior computation of its constituent expressions.

Use Inheritance to Model Node Types (Expression)

- The Expression interface enforces the design
  - All classes which implement it must provide an eval() method.
- The above is an example of the composite design pattern
  - LiteralExpression is a simple type.
  - AddExpression and SubExpression are composite types.

Use Inheritance to Model Node Types (TurtleGraphics)

- The Command interface enforces the design
  - All classes which implement it must provide an execute() method.
- The above is an example of the composite design pattern
  - Forward, TurnRight, TurnLeft and Toggle are simple types.
  - Repeat is a composite type.

Command Language in Turtle Graphics

- Program to control a “turtle” moving around a grid.
- Move forward, turn, toggle (changes drawing colour), and these can be put in repeating groups.

Sample Instructions

```java
public interface Command {
    public void execute(TurtleFrame win);
}
```

Command.java
Forward.java

```java
public class Forward implements Command, TurtleTokens {
    private int numSteps;
    public Forward(int numSteps) {
        this.numSteps = numSteps;
    }
    public void execute(TurtleFrame win) {
        win.forward(numSteps);
    }
}
```

Repeat.java

```java
public class Repeat implements Command, TurtleTokens {
    private Command seq;
    private int repetitions;
    public Repeat(Command seq, int repetitions) {
        this.seq = seq;
        this.repetitions = repetitions;
    }
    public void execute(TurtleFrame win) {
        for (int i = 0; i < repetitions; i++)
            seq.execute(win);
    }
}
```

Object Diagram

```
... Diagram showing various turtle commands and their interactions...
```

DSP1

- Due 5pm on Friday 25th May
- Submit code as .zip on WebCT

DSP2

- Data Structures Problem 2 is from Big Java 2nd ed pg 770:
  - 20.13: Implement a queue as a circular array.
    - Use two index variables, head and tail that contain the index of the next object to be removed, and the next object to be added, respectively.
    - Write a main method that shows your code in action.
    - DEMO AND GRADING IN LAB TIME
- Due Friday 1st June

Summary: Recursion and Composites

- For structures built from Composite design pattern, each class will provide the method, e.g., evaluate():
  - atomic classes can execute evaluate() directly (base cases);
  - composed classes will have code that includes recursive calls of evaluate() on the contained objects.
Inductive reasoning

- To prove correctness of recursive code, use proof by general induction.
- Claim: it works on inputs of size $n$.
- Prove this, using assumption that it works on all inputs whose size is smaller than $n$.
  - For which we always need a base case, for inputs of size 0 or 1.

What have we done?

- Trees
- Recursive algorithms on Trees
- Alternative coding styles
- Composite Design Pattern
- Recursive algorithms on composites

Reminder: week 10

- Next week is a vacation.
- Then, in week 10, there is no lecture on Monday.
  - Public holiday.
- Quiz 3 in class on Thursday.
  - Covers grammars, recursion, trees, composites.