Knowledge Management

Java tree implementation

Wednesday 12th of November 2014 12:08:10 PM

Tags

Java (/page/show/id/java-tag)

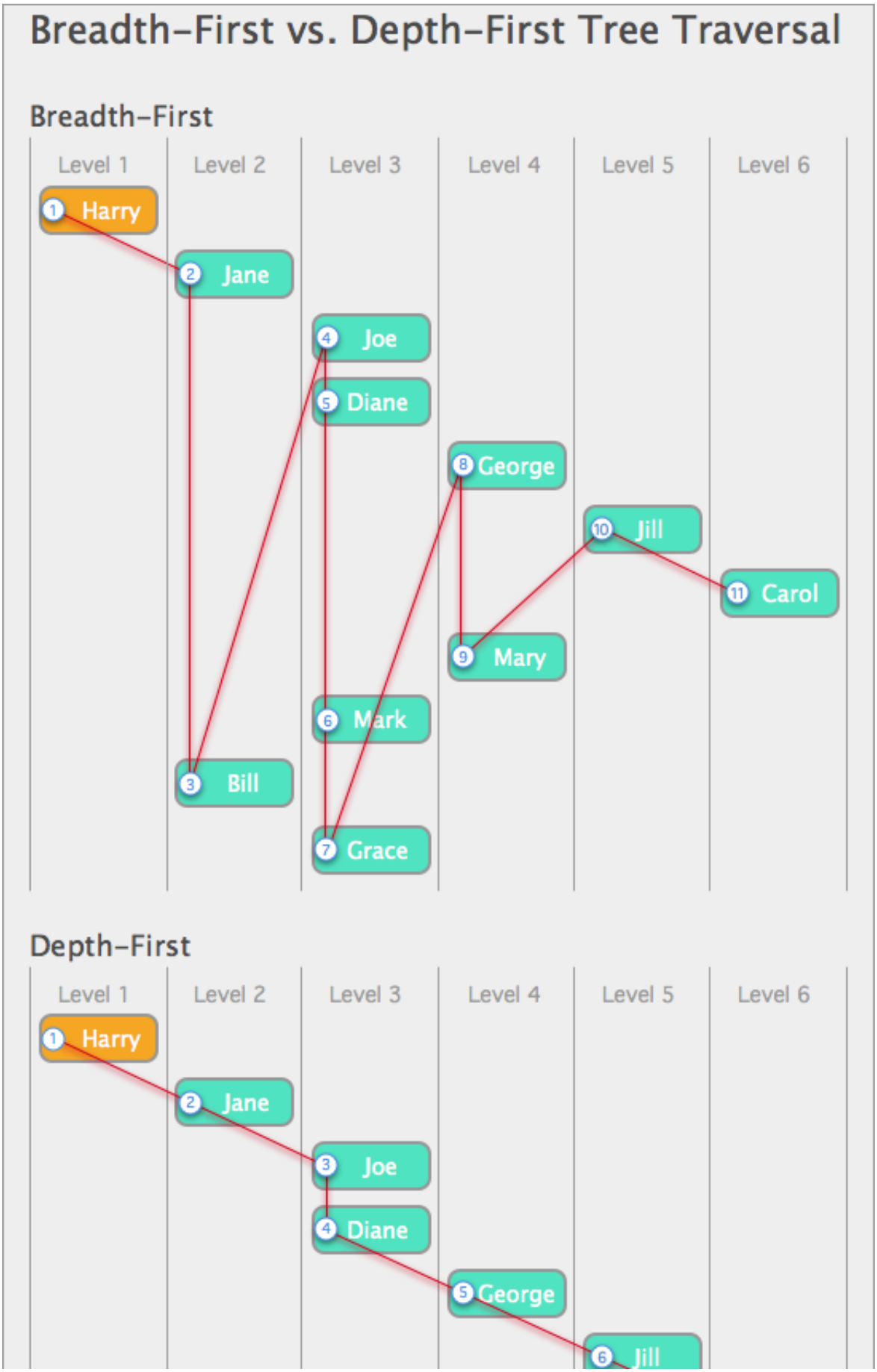
Breadth-first vs. depth-first tree traversal

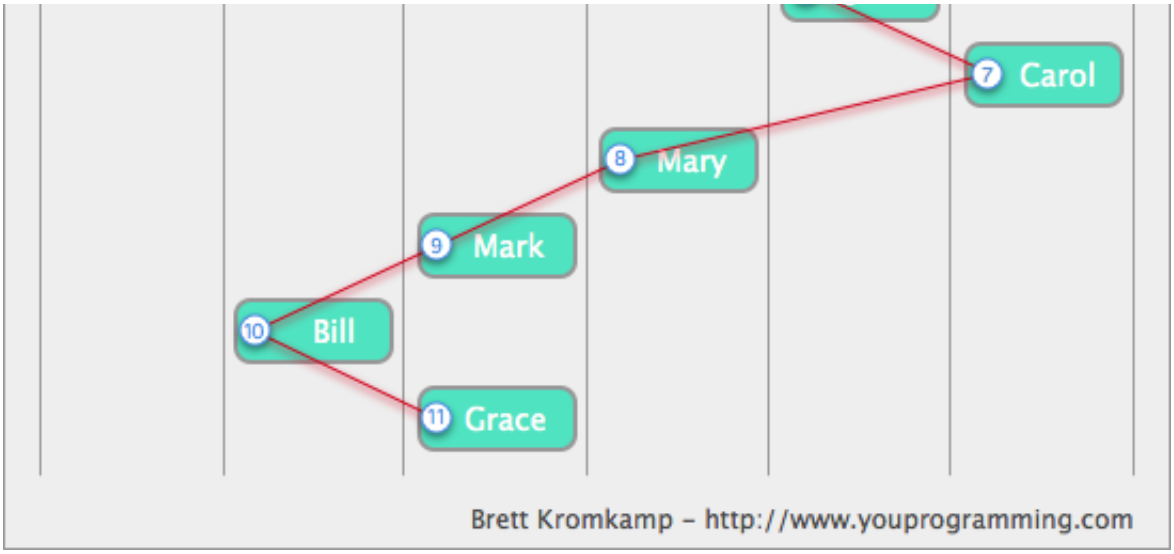
One of PerfectLearn (http://www.perfectlearn.com)'s features is the ability to visually display (and navigate) the topic map graph. In order to do so, I had to write a *getRelatedTopics* method that builds a tree structure of a topic's related topics to a given degree of separation. The Java source code below is my implementation of a tree structure and related classes (specifically, both a depth-first (http://en.wikipedia.org/wiki/Depth-first_search) and breadth-first (http://en.wikipedia.org/wiki/Breadth-first_search) iterator).

For the sake of comparison, check out the equivalent Python tree implementation (python-3-tree-implementation).

Source code

- App.java
- TraversalStrategy.java
- Node.java
- BreadthFirstTreeIterator.java
- DepthFirstTreeIterator.java
- Tree.java





Recursive functions and iterators

Although I am normally quite good with recursive functions (http://en.wikipedia.org/wiki/Recursion), for some reason when writing the *breadth-first* iterator I struggled. If you take a look at the code you will see that, although I managed to write the iterator, it is probably an inefficient implementation. Suggestions on how to improve the code would be more than welcome :-) Please keep in mind that the code is not production-ready, yet.

Update, August 23, 2012:

One talented programmer that I know has already pointed out a potential flaw with the implementation of the two iterators; that is to say, both iterators do their processing "*up front*" within the call to their respective constructors. A better implementation would be to determine the next node to visit "*on the fly*" (as the current node is being traversed). I hope to post an updated version of the iterators using this technique within the next couple of days (how I miss Python (python-3-tree-implementation)-like generators (http://docs.python.org/py3k/tutorial/classes.html#generators) in Java ;-)).

Update, April 14, 2014:

I've added *App.java* (and the accompanying execution output) to clarify how to use the *Tree* class. Furthermore, I have refactored the *Tree* class to use a *HashMap* to hold the nodes instead of an *ArrayList* which dramatically increases the class's performance. In addition, I also cleaned up the code a bit with regards to variable and method naming. Finally, within the next day or two I will extend the code to support arbitrarily typed '*payload*' objects for the individual tree nodes.

Output (from running *App.java*)

```
/Library/Java/JavaVirtualMachines/jdk1.8.0.jdk/Contents/Home/bin/java ...

Harry
    Jane
        Joe
        Diane
            George
                Jill
                    Carol
            Mary
        Mark
    Bill
        Grace

***** DEPTH-FIRST ITERATION *****
Harry
Jane
Joe
Diane
George
Jill
Carol
Mary
Mark
Bill
Grace

***** BREADTH-FIRST ITERATION *****
Harry
Jane
Bill
Joe
Diane
Mark
Grace
George
Mary
Jill
Carol

Process finished with exit code 0
```

App.java

```
/*
 * Copyright (C) 2007-2014 by Brett Alistair Kromkamp <brett@polishedcode.com>.
 */

package com.quesucede.tree;
```

```
import java.util.Iterator;

public class App {
    public static void main(String[] args) {

        Tree tree = new Tree();

        /*
         * The second parameter for the addNode method is the identifier
         * for the node's parent. In the case of the root node, either
         * null is provided or no second parameter is provided.
         */
        tree.addNode("Harry");
        tree.addNode("Jane", "Harry");
        tree.addNode("Bill", "Harry");
        tree.addNode("Joe", "Jane");
        tree.addNode("Diane", "Jane");
        tree.addNode("George", "Diane");
        tree.addNode("Mary", "Diane");
        tree.addNode("Jill", "George");
        tree.addNode("Carol", "Jill");
        tree.addNode("Grace", "Bill");
        tree.addNode("Mark", "Jane");

        tree.display("Harry");

        System.out.println("\n***** DEPTH-FIRST ITERATION *****");

        // Default traversal strategy is 'depth-first'
        Iterator<Node> depthIterator = tree.iterator("Harry");

        while (depthIterator.hasNext()) {
            Node node = depthIterator.next();
            System.out.println(node.getIdentifier());
        }

        System.out.println("\n***** BREADTH-FIRST ITERATION *****");

        Iterator<Node> breadthIterator = tree.iterator("Harry", TraversalStrategy.BREADT
H_FIRST);

        while (breadthIterator.hasNext()) {
            Node node = breadthIterator.next();
            System.out.println(node.getIdentifier());
        }
    }
}
```

TraversalStrategy.java

```
/*
 * Copyright (C) 2007-2014 by Brett Alistair Kromkamp <brett@polishedcode.com>.
 */

package com.quesucede.tree;

public enum TraversalStrategy {
    DEPTH_FIRST,
    BREADTH_FIRST
}
```

Node.java

```
/*
 * Copyright (C) 2007-2014 by Brett Alistair Kromkamp <brett@polishedcode.com>.
 */

package com.quesucede.tree;

import java.util.ArrayList;

public class Node {

    private String identifier;
    private ArrayList<String> children;

    // Constructor
    public Node(String identifier) {
        this.identifier = identifier;
        children = new ArrayList<String>();
    }

    // Properties
    public String getIdentifier() {
        return identifier;
    }

    public ArrayList<String> getChildren() {
        return children;
    }

    // Public interface
    public void addChild(String identifier) {
        children.add(identifier);
    }
}
```

BreadthFirstTreeIterator.java

```
/*
 * Copyright (C) 2007-2014 by Brett Alistair Kromkamp <brett@polishedcode.com>.
 */

package com.quesucede.tree;

import java.util.*;

/*
 * See URL: http://en.wikipedia.org/wiki/Breadth-first_search
 */

public class BreadthFirstTreeIterator implements Iterator<Node> {

    private static final int ROOT = 0;

    private LinkedList<Node> list;
    private HashMap<Integer, ArrayList<String>> levels;

    public BreadthFirstTreeIterator(HashMap<String, Node> tree, String identifier) {
        list = new LinkedList<Node>();
        levels = new HashMap<Integer, ArrayList<String>>();

        if (tree.containsKey(identifier)) {
            this.buildList(tree, identifier, ROOT);

            for (Map.Entry<Integer, ArrayList<String>> entry : levels.entrySet()) {
                for (String child : entry.getValue()) {
                    list.add(tree.get(child));
                }
            }
        }
    }

    private void buildList(HashMap<String, Node> tree, String identifier, int level) {
        if (level == ROOT) {
            list.add(tree.get(identifier));
        }

        ArrayList<String> children = tree.get(identifier).getChildren();

        if (!levels.containsKey(level)) {
            levels.put(level, new ArrayList<String>());
        }
        for (String child : children) {
            levels.get(level).add(child);
```

```
            // Recursive call
            this.buildList(tree, child, level + 1);
        }
    }

    @Override
    public boolean hasNext() {
        return !list.isEmpty();
    }

    @Override
    public Node next() {
        return list.poll();
    }

    @Override
    public void remove() {
        throw new UnsupportedOperationException();
    }
}
```

DepthFirstTreeIterator.java

```
/*
 * Copyright (C) 2007-2014 by Brett Alistair Kromkamp <brett@polishedcode.com>.
 */

package com.quesucede.tree;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedList;

/*
 * See URL: http://en.wikipedia.org/wiki/Depth-first_search
 */

public class DepthFirstTreeIterator implements Iterator<Node> {
    private LinkedList<Node> list;

    public DepthFirstTreeIterator(HashMap<String, Node> tree, String identifier) {
        list = new LinkedList<Node>();

        if (tree.containsKey(identifier)) {
            this.buildList(tree, identifier);
        }
    }
```

```
    private void buildList(HashMap<String, Node> tree, String identifier) {
        list.add(tree.get(identifier));
        ArrayList<String> children = tree.get(identifier).getChildren();
        for (String child : children) {

            // Recursive call
            this.buildList(tree, child);
        }
    }

    @Override
    public boolean hasNext() {
        return !list.isEmpty();
    }

    @Override
    public Node next() {
        return list.poll();
    }

    @Override
    public void remove() {
        throw new UnsupportedOperationException();
    }
}
```

Tree.java

```
/*
 * Copyright (C) 2007-2014 by Brett Alistair Kromkamp <brett@polishedcode.com>.
 */

package com.quesucede.tree;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;

public class Tree {

    private final static int ROOT = 0;

    private HashMap<String, Node> nodes;
    private TraversalStrategy traversalStrategy;

    // Constructors
    public Tree() {
        this(TraversalStrategy.DEPTH_FIRST);
    }
```

```
    public Tree(TraversalStrategy traversalStrategy) {
        this.nodes = new HashMap<String, Node>();
        this.traversalStrategy = traversalStrategy;
    }

    // Properties
    public HashMap<String, Node> getNodes() {
        return nodes;
    }

    public TraversalStrategy getTraversalStrategy() {
        return traversalStrategy;
    }

    public void setTraversalStrategy(TraversalStrategy traversalStrategy) {
        this.traversalStrategy = traversalStrategy;
    }

    // Public interface
    public Node addNode(String identifier) {
        return this.addNode(identifier, null);
    }

    public Node addNode(String identifier, String parent) {
        Node node = new Node(identifier);
        nodes.put(identifier, node);

        if (parent != null) {
            nodes.get(parent).addChild(identifier);
        }

        return node;
    }

    public void display(String identifier) {
        this.display(identifier, ROOT);
    }

    public void display(String identifier, int depth) {
        ArrayList<String> children = nodes.get(identifier).getChildren();

        if (depth == ROOT) {
            System.out.println(nodes.get(identifier).getIdentifier());
        } else {
            String tabs = String.format("%0" + depth + "d", 0).replace("0", "    "); // 4
spaces
            System.out.println(tabs + nodes.get(identifier).getIdentifier());
        }
        depth++;
```

```
        for (String child : children) {

            // Recursive call
            this.display(child, depth);
        }
    }

    public Iterator<Node> iterator(String identifier) {
        return this.iterator(identifier, traversalStrategy);
    }

    public Iterator<Node> iterator(String identifier, TraversalStrategy traversalStrategy)
{
        return traversalStrategy == TraversalStrategy.BREADTH_FIRST ?
                new BreadthFirstTreeIterator(nodes, identifier) :
                new DepthFirstTreeIterator(nodes, identifier);
    }
}
```

PerfectLearn Knowledge Management World War 2 (History) Case Study

YouProgramming Programming Tutorials YouTube Channel

(https://www.youtube.com/channel/UClhwPz1s8ZCxWjt4RZZxV-Q)

- YouProgramming (http://www.youprogramming.com), a website and accompanying YouTube channel (https://m.youtube.com/channel/UClhwPz1s8ZCxWjt4RZZxV-Q) with video tutorials and courses related to the Java and Python programming languages. With these tutorials I hope to teach you the art of software development in general and Java, Android, and Python development in particular.

Documentation
- Related
  - PerfectLearn Knowledge Management (/page/show/id/perfectlearn)
  - Python tree implementation (/page/show/id/python-3-tree-implementation)
  - Knowledge Management Using Topic Maps (/page/show/id/frontpage)
- Programming
  - Python tree implementation (/page/show/id/python-3-tree-implementation)

Association
- Related
  - Programming (/page/show/id/technology-programming)

Categorization
- Category
  - Java (/page/show/id/java-tag)

Go!

View Tags (/tag/show/id/java-tree-implementation)

Tweets

Follow

QueSucede.com @quesucede 1 Apr

Thinking about software business models: perfectlearn.com/2015/04/thinki... #softwaredevelopment #businessmodel #perfectlearn

Show Summary

QueSucede.com @quesucede 15 Feb

PerfectLearn (version 1.0) private beta testing has started! #perfectlearn #km #learning

Expand

Cédric Champeau @CedricChampeau 11 Feb

So it's official, I'm now a JavaOne Rock Star :) RT @steveonjava: Congratulations to the 2014 JavaOne Rock Stars! bit.ly/1DFFnxq

Retweeted by QueSucede.com

Show Summary

Gearóid Gman Maguire 10 Feb

@gearoidmaguire

@quesucede your Android tutorial part 2 really helped me understand Grids, keep it up.

Retweeted by QueSucede.com

Expand

Tweet to @quesucede

Google (https://www.google.com/+BrettKromkamp?rel=author)
