The Beauty Of Trees

Teacher Notes to support Tenderfoot Unit 2: Clever Stuff For Common Problems – Going beyond simple algorithms

An initial demonstration shows how compound data structures, such as lists and trees, can be viewed as subsets of the more general graph structure. A short class exercise demonstrates how a particular subset of a tree structure, a binary tree, can be constructed from a random set of numbers. A specific traversal returns the numbers in sorted order, without moving them within the structure.

Preparation required:
Binary Tree Traverse Exercise sheet for each pupil.
Plain paper.

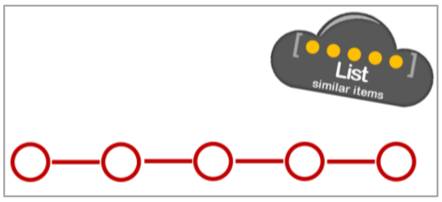
Variables, Lists, Trees and Graphs

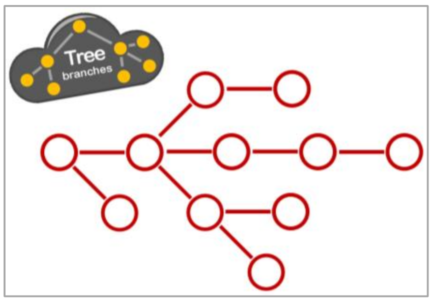
The famous computer scientist, Niklaus Wirth wrote a book in 1975 entitled Algorithms + Data Structures = Programs. A better way to think of it might be that computation acting on information leads to solutions to problems. Most students are familiar now with the term algorithm and the key constructs from which algorithms are built; sequence, selection and repetition. On the other side of the equation are data structures for holding information, such as lists, graphs and trees. The presentation shows how they 'fit together' as a family.

Variable
a single entity

The presentation does this by considering the development of a rail network. Initially we may have only one station. A single entity, such as a station can be stored as a named variable. Single variables can be thought of as the foundation of all other data structures. We can build more complex structures by aggregating variables – making a compound data structure.

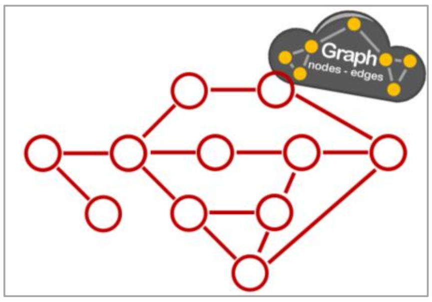
Once we connect two stations, a compound data structure makes sense. Because they are similar items a list would work well. And a list would continue to be appropriate whilst extra stations were added to this one line. A linear structure such as this could also cope with new stations inserted in the line, rather than appended to the end. But what happens when branch lines are added?





This new sort of arrangement is better thought of as a tree – with different paths or lines branching off. Trees can have many branches, but re distinguished by only having one path to a leaf node. They can also be rooted (or not) as we shall see with an exercise using a binary tree.

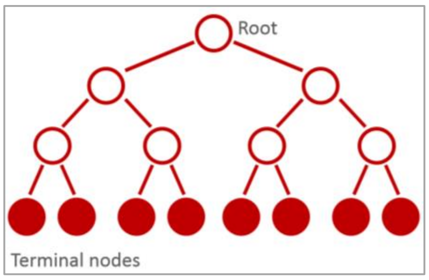
As the network continues to grow, branching paths may no longer be adequate for representing the railway. We may have lines that loop round, for example. To represent a rail network with circuits in it we need to be thinking in terms of graph structures, the most general representation. So Trees and Lists can be thought of as particular subsets of the more general graph structure.



We often introduce algorithmic constructs to children away from the computer. Developing familiarity with these makes the challenge of coding algorithms easier. Similarly, developing an appreciation of the role of data structures away from the computer helps develop an appreciation of their role in developing clever algorithms. The exercise overleaf introduces a novel way to solve a common problem.

Think Of A Number

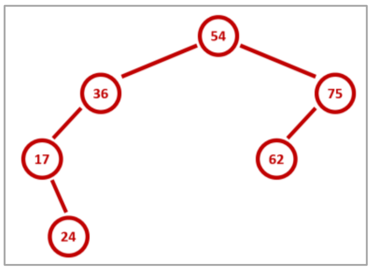
A binary tree is a particular subset of the more general tree structure. Firstly it is a rooted tree, which means a node is identified from which the tree springs. Somewhat confusingly, in computing, trees are usually drawn upside down. The root is at the top.



In a binary tree the root node has two and only two child nodes, hence the name. Each child node is referred to as the left and right. Each subsequent node also has two child nodes though it is possible that every node may not be used. Formally though, in a binary tree every node has two child nodes, apart from terminal nodes, which have no children.

You can play do this as a class activity with a child at the board drawing the binary tree as it grows or you can have each child creating their own tree on paper. The activity involves children picking numbers at random and can go on as long as you want. For simplicity, it is probably worth having limits, say between 1 and 100, and avoiding duplicate values.

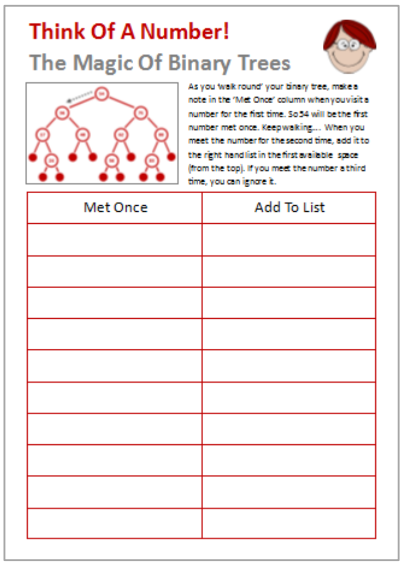
To demonstrate, we use ten numbers, selected at random, though, for the purpose of the animation, they had to be selected in advance. The presentation builds the binary tree by following simple rules. The first numbers are 54, 36, 75, 17, 24, 62 …



The first number, 54 becomes the root node. Because 36 is less than 54, it become the left child node and as 75 is greater than 54, it become the right child node. So where does 17 go? Starting at the root, we move left, because 17 is less than 54. It is also less than 36 so becomes the left child of 36. How about 24? Less than 54, and 36, but more than 17, so the right child of 17. 62 goes right of 54, but left of 75. Obviously we could go on and on, the slides completing the binary tree for the list of numbers.

In a binary tree, each node has two child nodes apart from terminal nodes. As every node containing a value should have two children, we can add empty terminal nodes to complete the tree.

Once built, we 'walk around' the structure (traverse the tree). Starting at the root, going anti-clockwise initially, pupils simply trace a line around the tree, making a note each time they meet a number, then moving it to a second column when they meet it a second time. Give each student a sheet to record the results.



Think Of A Number!

The Magic Of Binary Trees

As you 'walk round' your binary tree, make a note in the 'Met Once' column when you visit a number for the first time. So 54 will be the first number met once. Keep walking… When you meet the number for the second time, add it to the right hand list in the first available space (from the top). If you meet the number a third time, you can ignore it.

Met Once	Add To List

If they are struggling, there is a hidden slide which builds up the answer. It can be used for a whole class as a child traces the route on a whiteboard. The subsequent 3 slides walk through the process step by step, visiting each node in turn. The first slide takes the animation up to the return to the root node. Then up to the second visit of Node 75, the third slide completing the traversal. Use (or hide) whichever is most suitable.

Once completed, can the students see what has happened? By following this algorithm we can retrieve an ordered list without having to sort the values first.

Another Tenderfoot Unit: Doing Stuff And Doing Stuff Well considers ways to evaluate the efficiency of algorithms. This activity can be used when reflecting on the efficiency of different sorting algorithms. In this case, no values are moved or swapped (as they must be in a list). The sort is accomplished by the way the tree is built and the order in which nodes are visited in a traversal. Beautiful!