



Week 16: General trees & binary trees

4009 DATA STRUCTURES & ALGORITHM (DSA)

Learning Outcomes

- ❖ Create, manipulate and understand the data structure of General and Binary trees
- ❖ Define and understand the tree data structures
- ❖ Implement Java coding of either general tree or binary tree
- ❖ Understand the importance of the tree data structure in computing

Content Overview

- ❖ General Trees
 - ❖ Tree definitions and properties
 - ❖ Formal Tree Definition
 - ❖ Visual breakdown of a tree ADT
- ❖ General trees
- ❖ The Tree ADT
- ❖ A simple version of the List Interface
- ❖ Simplified tree implemented data structure
- ❖ Binary trees
 - ❖ A Recursive Binary Tree Definition
 - ❖ Visual breakdown of a binary tree ADT
- ❖ The Binary ADT
- ❖ Simple binary tree implementation

Data Structure Trade-offs

Data structures	Advantages	Disadvantages
Array	Quick insertion, very fast access to index known	Slow search, slow deletion, fixed size
Ordered Array	Quicker Search than unsorted array	Slow insertion and deletion, fixed size
Stack	Provide Last-in First-out access	Slow access to other items
Queue	Provide First-in First out access	Slow access to other items
Linked List	Quick Insertion, Quick deletion	Slow search
Binary Tree	Quick search, insertion, deletion (if tree balanced)	Deletion algorithm is complex
Hash Table	Very fast access if key known, fast insertion	Slow deletion, access slow if key not known, inefficient memory usage
Heap	Fast insertion, deletion, access to large item	Slow access to other items
Graphs	Model real world situations	Some algorithms are slow and complex

General Trees

- Trees are non-linear data structure in computing.
- Trees enable data organisation and allow the implementation of many algorithms much faster than linear data structures such as arrays or linked lists.
- Besides data organisation, trees can be used in graphical user interfaces, databases and websites.
- Relationship in a tree are hierarchical with some objects being “above” and some “below” others.
- Main terminologies for the tree data structure include parent, child, ancestor and descendant.

Application of Lists in the real world

- Used in file systems – operating systems use tree structure to manage, navigate, organise and manipulate directories and files.
- Website through the use of Document Object Model (DOM) where webpages in html format are organised hierarchy.
- Company hierarchies where organisations charts are model as trees representing company structure and report roles in the organisation.

Tree definitions and properties

- A tree is an abstract data type that stores elements hierarchically. With the exception of the top element, each element in a tree has a parent element and zero or more children elements.
- A tree is usually visualized by placing elements inside ovals or rectangles, and by drawing the connections between parents and children with straight lines. (See Figure 8.2.)
- The **top element is the root of the tree**, but it is drawn as the highest element.
- The other elements being connected below (just the opposite of a botanical tree).

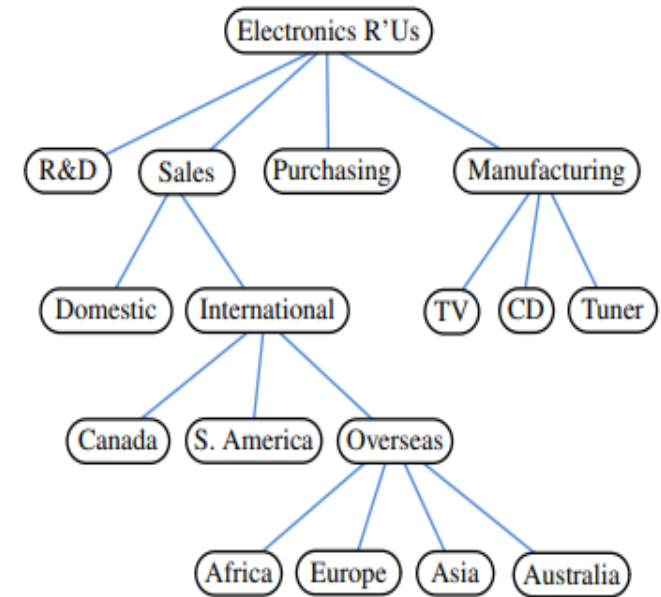


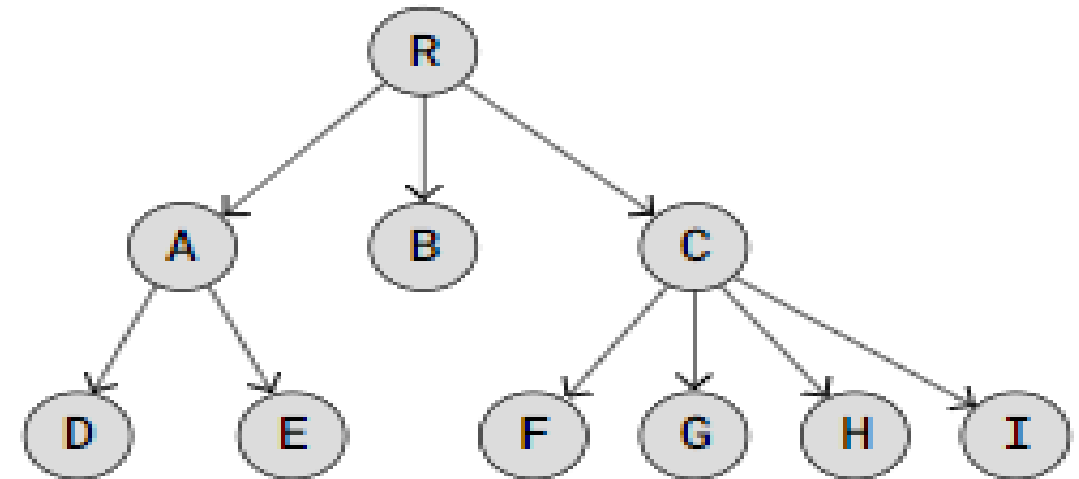
Figure 8.2: A tree with 17 nodes representing the organization of a fictitious corporation. The root stores *Electronics R'Us*. The children of the root store *R&D*, *Sales*, *Purchasing*, and *Manufacturing*. The internal nodes store *Sales*, *International*, *Overseas*, *Electronics R'Us*, and *Manufacturing*.

Formal Tree Definition

- Formally, we define a tree T as a set of nodes storing elements such that the nodes have a parent-child relationship that satisfies the following properties:
 - If T is nonempty, it has a special node, called the root of T , that has no parent.
 - Each node v of T different from the root has a unique parent node w ; every node with parent w is a child of w .
- *Note, a tree can be empty, meaning that it does not have any nodes.*
- **Two nodes that are children of the same parent are siblings.** A node v is external if v has no children. A node v is internal if it has one or more children. **External nodes are also known as leaves.**

Visual breakdown of a tree ADT

- The first node in a tree is called the root node.
- A link connecting one node to another is called an edge.
- A parent node has links to its child nodes. Another word for a parent node is internal node.
- A node can have zero, one, or many child nodes.
- A node can only have one parent node.
- Nodes without links to other child nodes are called leaves, or leaf nodes.



Root = **R**

Parent nodes = **R, A & C**

Child nodes = **A, B, C, D, E, F, G, H & I**

Leaf nodes = **B, D, E, F, G, H & I**

General trees

Ordered trees

A tree is ordered if there is a meaningful linear order among the children of each node; that is, we purposefully identify the children of a node as being the first, second, third, and so on. Such an order is usually visualized by arranging siblings left to right, according to their order.

The Tree abstract data type (ADT)

- The tree abstract data type (ADT) that supports the following methods:
 - **root()**: Returns the position of the root of the tree (or null if empty)
 - **parent(p)**: Returns the position of the parent of position p (or null if p is the root).
 - **children(p)**: Returns an iterable collection containing the children of position p (if any).
 - **numChildren(p)**: Returns the number of children of position p.
 - **isInternal(p)**: Returns true if position p has at least one child.
 - **isExternal(p)**: Returns true if position p does not have any children.
 - **isRoot(p)**: Returns true if position p is the root of the tree.
 - **size()**: Returns the number of positions (and hence elements) that are contained in the tree.
 - **isEmpty()**: Returns true if the tree does not contain any positions (and thus no elements).
 - **iterator()**: Returns an iterator for all elements in the tree (so that the tree itself is Iterable).
 - **positions()**: Returns an iterable collection of all positions of the tree

A simple version of the Tree Interface

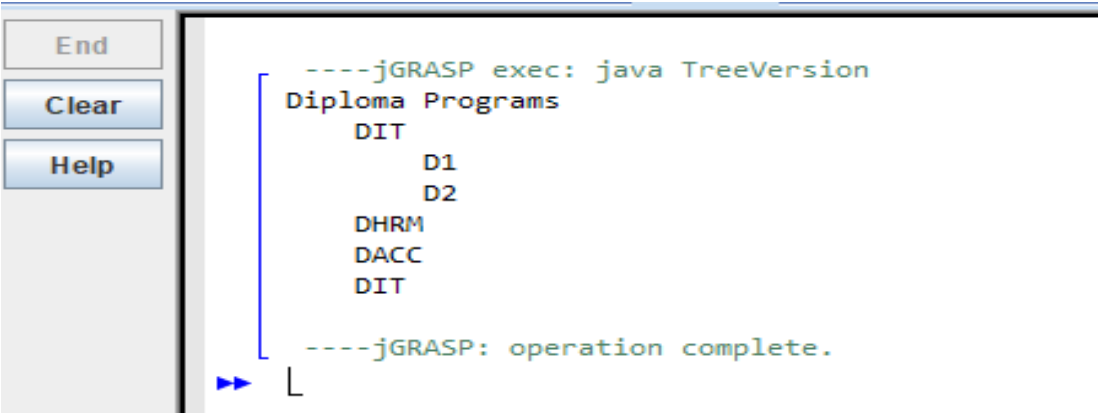
```
1  /** An interface for a tree where nodes can have an arbitrary number of children. */
2  public interface Tree<E> extends Iterable<E> {
3      Position<E> root();
4      Position<E> parent(Position<E> p) throws IllegalArgumentException;
5      Iterable<Position<E>> children(Position<E> p)
6                               throws IllegalArgumentException;
7      int numChildren(Position<E> p) throws IllegalArgumentException;
8      boolean isInternal(Position<E> p) throws IllegalArgumentException;
9      boolean isExternal(Position<E> p) throws IllegalArgumentException;
10     boolean isRoot(Position<E> p) throws IllegalArgumentException;
11     int size();
12     boolean isEmpty();
13     Iterator<E> iterator();
14     Iterable<Position<E>> positions();
15 }
```

Code Fragment 8.1: Definition of the Tree interface.

Simplified tree implemented data structure

```
1 import java.util.ArrayList;
2 import java.util.List;
3
4 class TreeNode {
5     String value;
6     List<TreeNode> children;
7
8     TreeNode(String value) {
9         this.value = value;
10        this.children = new ArrayList<>();
11    }
12
13    void addChild(TreeNode child) {
14        children.add(child);
15    }
16 }
17
18 public class TreeVersion {
19
20     public static void main(String[] args) {
21         // Create the root node
22         TreeNode root = new TreeNode("Diploma Programs");
23
24         // Add children in the same order as stack push
25         TreeNode node1 = new TreeNode("DIT");
26         root.addChild(node1);
27
28         TreeNode child1 = new TreeNode("D1");
29         node1.addChild(child1);
30
31         TreeNode child2 = new TreeNode("D2");
32         node1.addChild(child2);
33
34         TreeNode node2 = new TreeNode("DHRM");
35         root.addChild(node2);
36
37         TreeNode node3 = new TreeNode("DACC");
38         root.addChild(node3);
39
40         TreeNode node4 = new TreeNode("DIT");
41         root.addChild(node4);
42     }
```

```
43
44     // Print tree recursively
45     printTree(root, 0);
46 }
47
48 static void printTree(TreeNode node, int level) {
49     // Indent based on tree level
50     for (int i = 0; i < level; i++) {
51         System.out.print("    ");
52     }
53
54     System.out.println(node.value);
55
56     for (TreeNode child : node.children) {
57         printTree(child, level + 1);
58     }
59 }
60 }
```



End
Clear
Help

```
----jGRASP exec: java TreeVersion
Diploma Programs
  DIT
    D1
    D2
  DHRM
  DACC
  DIT
----jGRASP: operation complete.
```

Binary trees

- A binary tree is an ordered tree with the following properties:
 - ❑ Every node has at most two children.
 - ❑ Each child node is labeled as being either a left child or a right child.
 - ❑ A left child precedes a right child in the order of children of a node.
- The subtree rooted at a left or right child of an internal node v is called a left subtree or right subtree, respectively, of v . A binary tree is proper if each node has either zero or two children. Some people also refer to such trees as being full binary trees. Thus, in a proper binary tree, every internal node has exactly two children. A binary tree that is not proper is improper

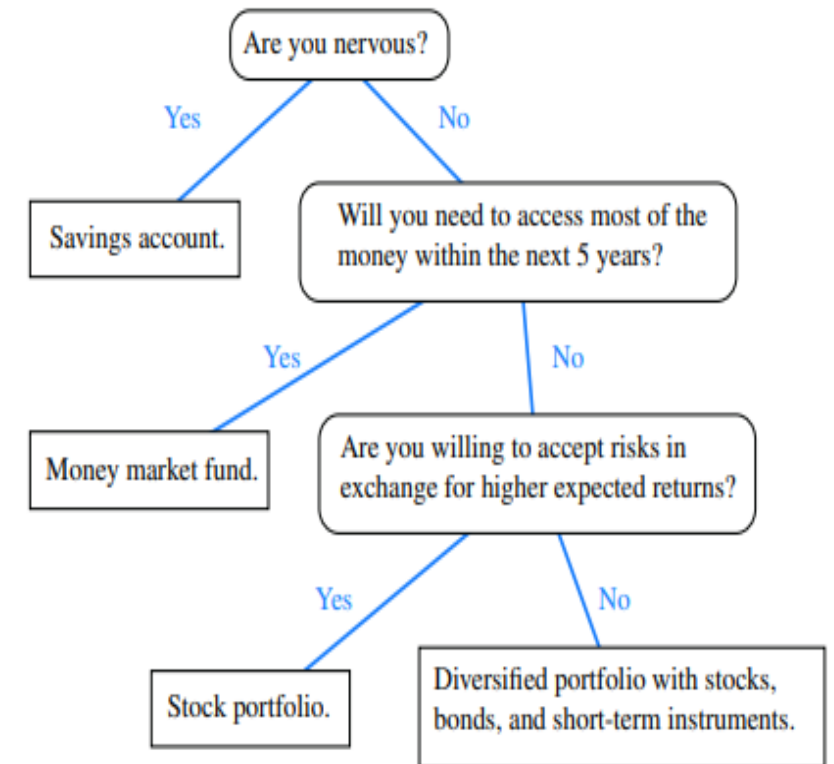


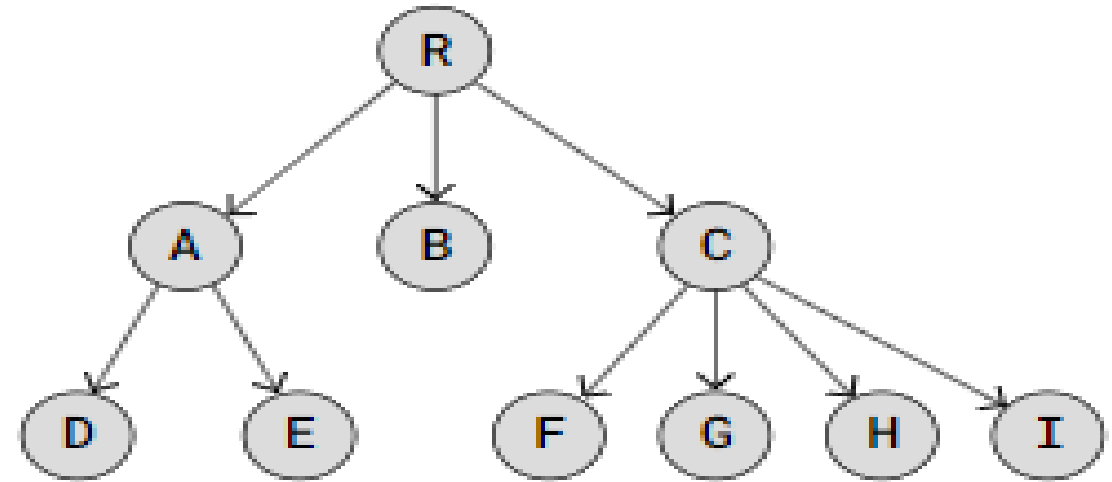
Figure 8.5: A decision tree providing investment advice.

A Recursive Binary Tree Definition

- Incidentally, we can also define a binary tree in a recursive way.
- In that case, a binary tree is either:
 - An empty tree.
 - A nonempty tree having a root node r , which stores an element, and two binary trees that are respectively the left and right subtrees of r . We note that one or both of those subtrees can be empty by this definition.

Visual breakdown of a binary tree ADT

- A **parent** node, or **internal** node, in a Binary Tree is a node with one or two **child** nodes.
- The **left child node** is the child node to the left.
- The **right child node** is the child node to the right.
- The **tree height** is the maximum number of edges from the root node to a leaf node.



Root = **R**

Parent nodes = **R, A, B & F**

Left child (A) – **C**

Right child (B) – **D**

B subtree – **B, E, F & G**

Child nodes = **A, B, C, D, E, F & G**

Leaf nodes = **B, D, E, F, G, H & I**

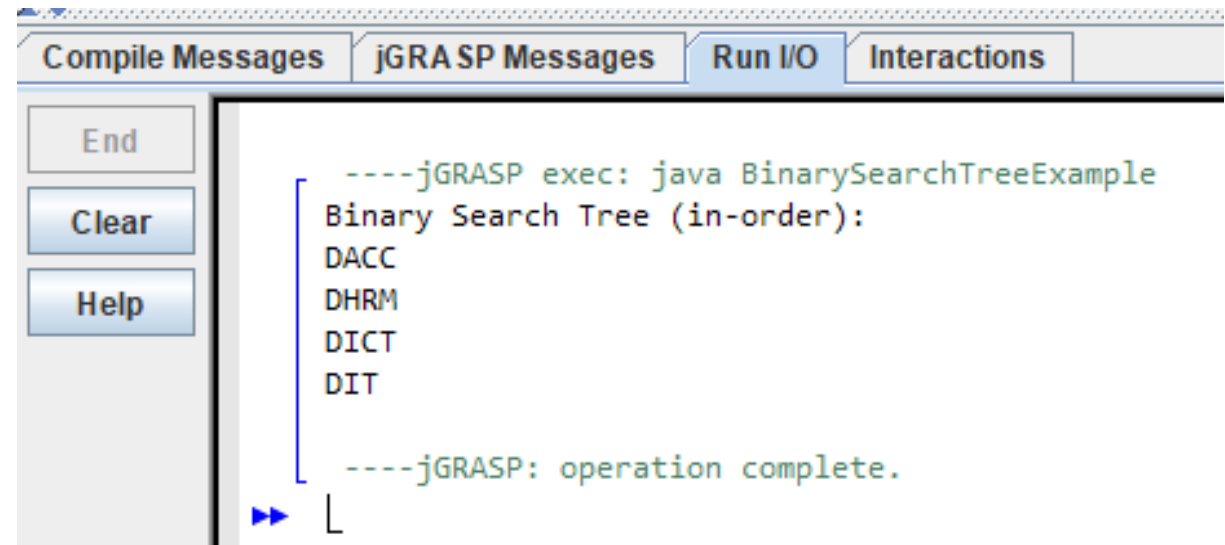
The Binary ADT

- As an abstract data type, a binary tree is a specialization of a tree that supports three additional accessor methods besides the other eleven methods discussed earlier:
 - ❑ **left(p)**: Returns the position of the left child of p (or null if p has no left child).
 - ❑ **right(p)**: Returns the position of the right child of p (or null if p has no right child).
 - ❑ **sibling(p)**: Returns the position of the sibling of p (or null if p has no sibling).

Simple binary tree implementation

```
1 class BinaryTreeNode {
2     String value;
3     BinaryTreeNode left;
4     BinaryTreeNode right;
5
6     BinaryTreeNode(String value) {
7         this.value = value;
8     }
9
10    void insert(String newValue) {
11        if (newValue.compareTo(value) < 0) {
12            if (left == null) left = new BinaryTreeNode(newValue);
13            else left.insert(newValue);
14        } else {
15            if (right == null) right = new BinaryTreeNode(newValue);
16            else right.insert(newValue);
17        }
18    }
19 }
20
21 public class BinarySearchTreeExample {
22
23     public static void main(String[] args) {
24         // Stack values (push order)
25         String[] itiCourses = {"DIT", "DHRM", "DACC", "DICT"};
26
27         BinaryTreeNode root = new BinaryTreeNode(itiCourses[0]);
28
29         for (int i = 1; i < itiCourses.length; i++) {
30             root.insert(itiCourses[i]);
31         }
32
33         System.out.println("Binary Search Tree (in-order):");
34         printInOrder(root);
35     }
36 }
```

```
36
37     static void printInOrder(BinaryTreeNode node) {
38         if (node == null) return;
39
40         printInOrder(node.left);
41         System.out.println(node.value);
42         printInOrder(node.right);
43     }
44 }
```



The screenshot shows a Java IDE window with four tabs: "Compile Messages", "jGRASP Messages", "Run I/O", and "Interactions". The "jGRASP Messages" tab is active, displaying the output of the program. On the left side of the IDE, there are three buttons: "End", "Clear", and "Help". The output text in the "jGRASP Messages" tab is as follows:

```
----jGRASP exec: java BinarySearchTreeExample
Binary Search Tree (in-order):
DACC
DHRM
DICT
DIT
----jGRASP: operation complete.
```

Below the output text, there is a blue double arrow icon pointing to the right, followed by a vertical line.

Topic next week

Week 17: Maps

DSA Week 16 activities

Refer to print out materials.

- This week, you are required to complete the questions and two labs.
 - Refer to the print out, answer all week 16 questions.
 - Refer to the print out, complete the two labs activities using the lab computers.

Note: You can complete the activities in any order, however, make afford to complete and understand everything which prepares you for well for the Final Exam.