Exercises 13 (Tree ADTs)

13A The *class hierarchy* of a Java program represents the subclass relationship between classes. The class hierarchy can be represented by a tree.

(a) For any Java program, what is the root vertex of the class hierarchy tree?

(b) What property of the Java language makes the class hierarchy a tree?

(c) Draw the class hierarchy for the classes in a Java program you have written.

(d) If Java interfaces are included along with classes, why is the result no longer a tree?

13B Implement an ordered tree ADT as a Java class, `LinkedOrderedTree`.

13C Using only the methods of the `Tree` interface, write a Java method, `traversePreorder`, that visits, in pre-order, all of the vertices in a given tree.

13D Suppose that the `Tree<E>` interface is to be extended with the following method:

```
public Iterator<E> verticesPreorder ();
// Return an iterator that will visit all elements of this tree,
// in pre-order (i.e., each vertex is visited before its
// children).
```

Write a Java class, `TreePreorderIterator<E>`, and use it to implement this method.

13F Implement an unordered tree ADT where each vertex's children are stored in an array rather than an SLL. What are the time complexities of the `addChild` and `remove` operations?

13G Consider the linked implementation of an unordered tree. The explicit link to a vertex's parent can be removed, at the expense of making the `parent` operation slower.

Modify the unordered tree implementation accordingly. What is the time complexity of the `parent` operation now?

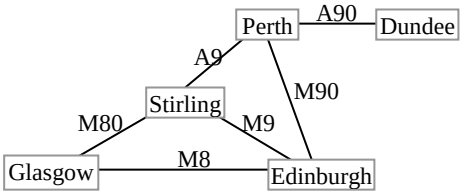
* **13H** Implement a specialized tree ADT for Java expressions. For simplicity, include only the arithmetic and relational operators, and literals of the primitive types **`boolean`** and **`int`**.

Equip your ADT with an `evaluate` method that returns the result of evaluating the Java expression. (*Hint*: Make the method return an object of class `Boolean` or `Integer`.)

** **13J** Extend your answer to Exercise 13H with a method, `typecheck`, that checks whether an operator is applied to operands of the correct type. For example, it should reject the trees corresponding to the expressions '`1 +` **`false`**', '**`false`** < **`tru`**e', and '`6 ==` **`true`**'. Note that the result of an arithmetic operation is of type **`double`** if any of its operands are of type **`double`**, otherwise its result is of type **`int`**.

Exercises 14 (Graph ADTs)

14A Consider the Scottish road network shown below. Find at least three distinct paths connecting Glasgow and Perth.



14B Consider the Scottish road network shown above. Draw diagrams showing how this undirected graph could be represented:

(a) using the edge-set representation

(b) using the adjacency-set representation

(c) using the adjacency-matrix representation (with $m = 8$, say).

14C In the edge-set representation of graphs, consider the following alternatives:

(a) Represent the vertex set by an *SLL*.

(b) Represent the edge set by a *hash-table*.

For each alternative, state what algorithms would be used to implement the graph operations, and determine their time complexities.

14D In the adjacency-set representation of graphs, consider the following alternatives:

(a) Represent the adjacency sets by *DLLs*.

(b) For a directed graph, provide each vertex with *two* adjacency sets, one for its in-edges and one for its out-edges.

For each alternative, state what algorithms would be used to implement the graph operations, and determine their time complexities.

14E Consider the illustration of the adjacency-matrix representation in the course notes.

(a) Show the effect of removing vertex V.

(b) Modify the implementation such that `removeVertex` keeps the matrix compact by shifting rows and columns. (For example, removing vertex V would reassign the numbers 3 and 4 to vertices W and X.) How would this change affect the time complexities of `removeVertex` and the other graph operations?

14F Modify the graph depth-first and breadth-first traversal algorithms to make them graph *searching* algorithms. Each should search the graph for a vertex whose element is equal to *target-elem*.

14G Implement the depth-first and breadth-first traversal algorithms in Java, using the `Graph` and `Digraph` interfaces.

14H Devise an algorithm to determine whether there is a path between two given vertices, *v* and *w*, in a given directed graph. (*Hint*: This is just a special case of a graph traversal algorithm.)

Implement your algorithm as a Java method.

14J Consider the Scottish road network shown below. Use the shortest-path algorithm to determine the shortest path from Edinburgh to Dundee.