Aston University
BIRMINGHAM UK

CS2DSA Data Structures and Algorithms in Java

Mini-project

This set of exercises is designed to give you further practice on skills relating to following learning outcomes of the module:

- Select and apply suitable Abstract Data Types (ADTs) within a principled software development process and ADT implementations, based on appropriate criteria.
- State Java language support for ADTs and use them for software development.

Verbal feedback on your work-in-progress / partial solution will be given to you at the dedicated CS2DSA practical session(s).

1. Decision trees

A decision tree is a tree diagram that summarizes all the different possible stages of a process that solves a problem by means of a sequence of decisions. It is a "flowchart-like structure" in which each internal (non-terminal) node represents a question, each tree branch indicates an answer to its question, and each leaf (terminal) node includes the solution to the problem (e.g. either 0 or 1). In binary decision trees, each non-terminal node has two edges only, left and right reference along with the data element. Figures 1 and 2 show examples of binary trees. The node at the top of the hierarchy of a tree is called the root node.

Decision Tree is deemed as powerful and popular tool for Artificial Intelligence (AI), particularly in classification and prediction tasks (Mitchell, 1997). It has many applications in Finance, Medicine and Business.

2. Example applications of Decision Trees

2.1. Expert system

Expert system is an important application of AI that aims to mimic the decision-making ability of a human to solve problems. In expert systems, decisions are often taken by collecting answers from humans to a series of questions using a binary decision tree model. Each parent (non-terminal node) in a decision tree is a question that has a finite number of responses. Answers to these questions can be either be true or false, yes or no, or multiple choice. Each possible answer to the question corresponds to a child of that node. Each child might be an additional question or a conclusion. Nodes that are conclusions would have no children, and so they would be leaves.

Figure 1 (extracted from (Carrano and Savitch, 2003)) represents an expert system example that uses binary decision tree to diagnose a problem with a television. Starting from the root, a question about the TV condition is evaluated. Depending on the user answer one of the two branches is selected. If the next branch leads to a non-terminal node another question will be displayed to user. This procedure is repeated until a final terminal node is reached, which normally represents the decision. A decision tree provides operations that move us along a path through the tree and access the current node.

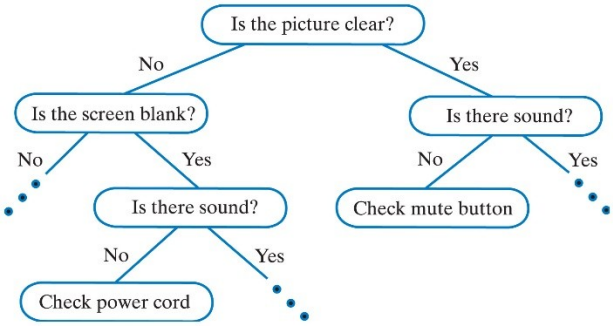

Figure 1: Expert system example that uses binary decision Tree to diagnose a problem with a television (Carrano and Savitch, 2003)

2.2. Diagnostic Systems in Medicine

Diagnostic Systems in Medicine is another important application of binary decision tree. In Figure 2 a binary decision tree is used for breast cancer diagnosis and treatment. The decision tree has been designed through consultation with experts, and it was particularly used to help doctors take decision about breast cancer treatment.

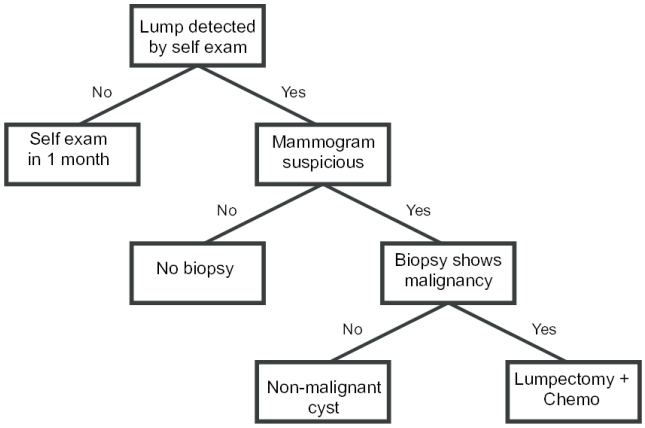

Figure 2. An example of a simple decision tree that might be used in breast cancer diagnosis and treatment, extracted from (Cruz and Wishart, 2006)

3. Guessing game application in Java

Decision trees have been widely used in the gaming industry. They allow players to make choices and accordingly the game can predict the outcome from these choices. Decision trees allow the player to create their own story or outcome by choosing the options they think are best. Hence, it is important for such applications to use:

- a suitable data structure to represent the binary tree data, and
- an efficient algorithm that can traverse the tree in order to predict the outcome

3.1. Your Task

Design and implement a standalone Java application which uses a decision tree structure to model a guessing game. The application will be implemented using the Java language and the Java Collections Framework (JCF).

In this application, the user will be asked to think of an object, the program will then attempt to predict/guess this object by asking the user a series of yes/true or no/false questions that can best describe the object in mind. Based on

the answers provided, the application will attempt to make a guess. If the guess is correct, the application wins the game, otherwise it loses. This program uses a binary decision tree to make the guess.

The application uses an input data file which includes a series of yes/true or no/false questions, which describe the conditions shared by a group of objects, as well as their answers. A further description of the file format and contents is provided below (cf. Section 3.4). The application will read the input file and model it in a binary tree format, where each internal node, including the root, contains a question that has two possible answers (yes/true or no/false). Each answer points to a child node, which can be another question (internal node) or a conclusion (leaf node).

A sample run of the program is provided in Figure 3. The user will initially be asked the question which appears on the first line of the file "`Is it an animal?`". If the user answers yes, it will go to the next line of the file "`Can it swim?`", And so on. The game continues in this fashion until the game reaches an answer line that begins with "A:", such as, "A: `lion`". An answer line indicates that the computer has asked enough questions that it is ready to guess what object the user is thinking of. The application will display the predicted animal name "`Would your object happen to be lion`". The user will be asked to confirm if the predicted animal matches the one in mind by typing (yes/true or no/false). If the guess is correct, the application wins, otherwise it loses.

```
Welcome to the Guessing Game
Please enter the file name? Shortquestions.txt
Please think of an object for me to guess...

Is it an animal? YES
Can it swim? NO
Does it have pointy ears? NO
Would your object happen to be lion YES
Hooray, I got it right!

Would you like to play again?
```

Figure 3. A sample run of the program

3.2. Functional requirements

1. The user will provide a reference to an input file that stores a set of questions/answers to be used by the program during the game.
2. The user will be notified if the file is not found using an appropriate message.
3. The user will be able to respond (with **yes/true or no/false)** to a series of questions. Each question will be displayed in one line.
4. The user will be able to confirm whether the predicted animal matches the one in mind or not.
5. The user will be notified if the application won or lost the game using an appropriate message.
6. The user will be offered to end the game or replay again.

3.3. Non-functional requirements

1. A user will interact with the system through a Text-based User Interface (TUI) that works with command-line type of input, with all output displayed by the standard output stream.
2. The response time of the application should be short. Ideally all operations should executes in constant time, i.e. O(1).
3. The display of results for each query should be clear and easy to understand.
4. The application needs to be robust and display appropriate messages should any run time errors occur.

3.4. Input data files

You will be provided with two data files.

- **ShortQuestions.txt** includes a short set of questions and answers that is to be used for testing purposes.
- **LongQuestions.txt** contains approximately 10 thousand questions and answers about animal obtained from animal game that is to be used to run the game.

In both files, each line either represents a single question or a single answer. Lines that start with "Q:" represent yes/no or true/false questions, and lines that start with "A:" represent answers, that is to be guessed by the application. The files will be used by the program to construct a binary decision tree.

3.5. How to produce the binary decision tree from a data file?

The tree nodes should appear in pre-order (i.e., in the order produced by a pre-order traversal of the tree). For example, consider the below sequence of text (questions and answers) that is provided in the data file ShortQuestions.txt.

```
Q: Is it an animal?
Q: Can it swim?
A: fish
Q: Does it have pointy ears?
A: vampire bats
A: lion
Q: Does it have wheels?
A: car
Q: Does it have a nice smell?
A: flower
A: socks
```

Figure 4. Linear form of the questions and answers

The binary decision tree of the above text should be represented as follows:

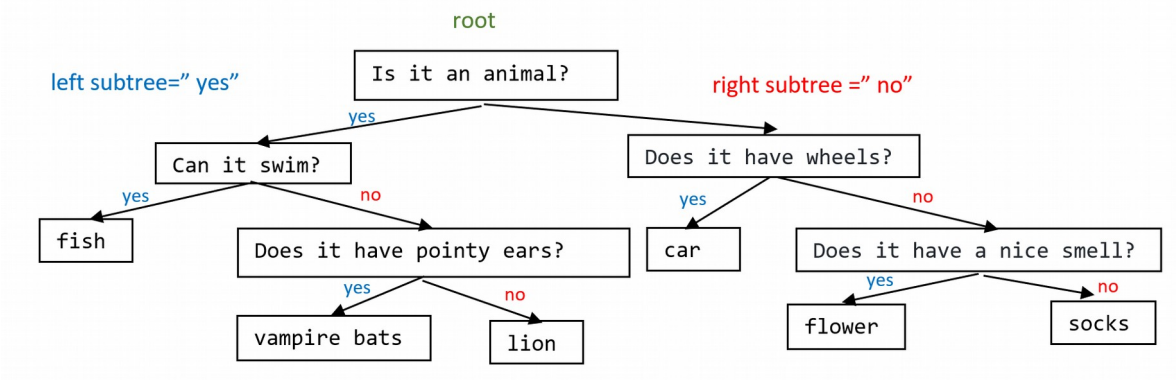

Figure 5. Binary tree form of the questions and answers

You will implement a **recursive** read function that works as follows:

```
read (input file)
  read one line, which starts with either "Q:" or "A:"
  if the line starts with "A:" (an "answer")
       then
              return the "answer"
```

```
    else
            read ("yes" left sub-tree)
            read ("no" right sub-tree)
            return a new "question" with the sub-trees inside
```

3.6. Given Java Code

- BinaryTreeADT.java (interface to a binary tree data structure)
- BinaryTreeNode.java (class representing a node in a binary tree with a left and right)

4. Hints about Implementation

1. You are expected to implement a Binary Tree class with linked structure, which implements the BinaryTreeADT interface (provided to you).
2. Elements of the developed Linked Binary Tree will be of type BinaryTreeNode (provided to you), where each node includes a Question or an Answer.
3. You are expected to implement a GuessingGame class which traverses nodes represented by the Linked Binary Tree.
4. The GuessingGame class will read the input file and model a binary decision tree to guess the object.
5. The GuessingGame will include a static method (main) that will run the game.
6. You might like to consider using some of the following facilities provided by J2SDK:
7. IO facilities:
   - java.util.Scanner
   - java.io.BufferedReader
   - java.io.File
   - java.io.InputStreamReader
   - java.io.FileInputStream
8. To facilitate your system testing, you might like to implement tester classes/methods using JUnit. This will enable tester objects to be created for simulating the behaviours of various test cases, hence reducing the time required to repeatedly enter the same set of data into the system after each modification made to the system.

5. References

Mitchell, T., 1997. Machine learning.

Carrano, F.M. and Savitch, W.J., 2003. *Data structures and abstractions with Java*. Upper Saddle River, NJ, USA: Prentice Hall.

Cruz, J.A. and Wishart, D.S., 2006. Applications of machine learning in cancer prediction and prognosis. *Cancer informatics*, *2*, p.117693510600200030.