Expert Systems

The game of animal is an old children's game. There are two participants in the game — the *player* and the *guesser*. The player is asked to think of an animal, which the guesser will try to guess. The guesser asks the player a series of yes-no questions, such as

Guesser: *Does it live on land?*

If the player answers 'yes', then the guesser can eliminate from consideration those animals that do not live on land, and use this information in formulating the next question, such as

Guesser: *Does it have four feet?*

Again, the answer to the question allows the user to eliminate from consideration either the animals with four feet, or those that do not have four feet. Carefully formulating each question allows the guesser to eliminate a large group of animals from consideration, based on the player's response. Eventually, the guesser knows of only a single animal with the given characteristics:

Guesser: *Is it an elephant?*

If the guesser is correct, then he or she wins the game. Otherwise, the player wins the game, and the guesser asks the player:

Guesser: *What is your animal?*

Player: *An aardvark.*

Guesser: *How does an elephant differ from an aardvark?*

Player: *An elephant has a trunk, but an aardvark does not.*

By remembering the new animal and the difference between his or her animal and the new animal, the guesser learns to distinguish between the two animals.

A computer program that plays the animal game provides a classic example of a situation in which a program can seemingly *learn* and thus display **artificial intelligence**. The user of the program assumes the role of the player and the program assumes the role of the guesser.

The program maintains a knowledge base of questions, each of which allows it to eliminate animals from consideration. When the program has narrowed its search to a single animal, it guesses that animal. If the guess is correct, the program wins. Otherwise the program asks the player to name the animal of which he or she was thinking, and then asks how to distinguish between this new animal and the animal it guessed. It then stores this question and the new animal in its knowledge base for the next time the game is played. The program thus exhibits some of the characteristics of learning each time it adds a new animal to its knowledge base.

As time passes and the program's knowledge base grows, it becomes more and more difficult for the player to think of animals that are not in the knowledge base — the program becomes an *expert* at guessing animals. Programs that exhibit expertise in some area through use of a knowledge base are known as **expert systems**, and the study of such systems is one of the branches of artificial intelligence. Examples of such systems range from welding experts that control welding robots on an automotive assembly line to legal experts that can help draw up standard legal documents.

Although most expert systems use fixed knowledge bases that the program is unable to modify, a program that plays the animal game is an example of a special *adaptive expert system*, because it adds new animals to its knowledge base as they are encountered. It is this ability to adapt its knowledge base that enables the animal program to simulate the process of learning.

The following program plays the animal game. For its knowledge base, it uses a special `DecisionTree` class built specially for this purpose.

```
/* animal.cpp plays the game of 'animal', in which the player
   thinks of an animal, and the program tries to guess it.
-------------------------------------------------------------------*/

#include <iostream>
using namespace std;
#include "DecisionTree.h"

bool playMore();
int main()
{
  cout << "\nWelcome to the game of Animal!\n";

  DecisionTree dTree("animal.data");  // load knowledge base

  do
  {
    cout << "\nThink of an animal, and I will try to guess
it...\n\n";
    int winner = dTree.descend();     // 0 == the person,
                                      // 1 == program

    if (winner)
      cout << "\nHa! Even a computer program can beat you...\n";
    else
       cout << "\nYou were just lucky...\n";
  }
  while ( playMore() );
```

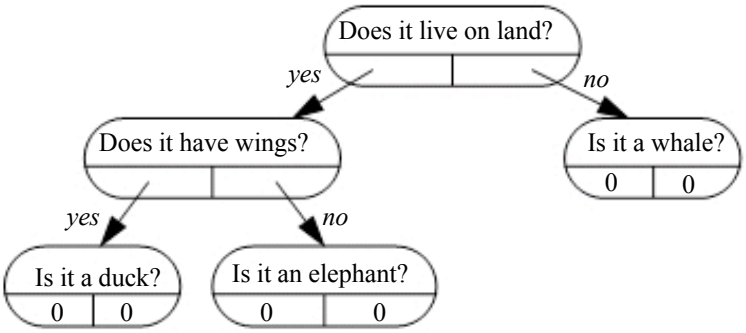
```
 } // knowledge base is auto-saved to file by DecisionTree destructor


bool playMore()
{
  char answer;
  cout << "\nDo you want to play again (y or n)? "; cin >> answer;
  return ((answer == 'y') || (answer == 'Y'));
}
```

Sample Run:

```
Welcome to the game of Animal!

Think of an animal, and I will try to guess it...

Does it live on land (y or n)? y

Does it have wings (y or n) ? n

Is it a(n) elephant (y or n)? y

Ha! Even a computer program can beat you...

Do you want to play again (y or n)? y

Think of an animal, and I will try to guess it...

Does it live on land (y or n)? n

Is it a(n) whale (y or n)? n

What animal are you thinking of? shark

Please enter a question, such that the answer is
 yes - for a(n) shark, and
 no - for a(n) whale
--> Is it cold-blooded

You were just lucky...

Think of an animal, and I will try to guess it...

Does it live on land (y or n)? n

Is it cold-blooded (y or n)? y

Is it a(n) shark (y or n)? n

What animal are you thinking of? electric eel

Please enter a question, such that the answer is
```

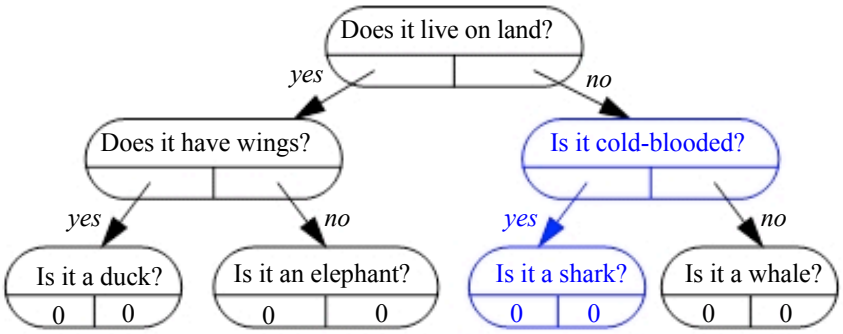
```
 yes - for a(n) shark, and

 no - for a(n) whale

--> Is meeting it a shocking experience

You were just lucky...

Do you want to play again (y or n)? n
```

The program plays the game by building a special tree to serve as its knowledge base, containing
the questions and the animals it "knows". For example, when the program was first written, it
"knew" only three animals: a duck, an elephant, and a whale. Its knowledge base was initially structured as follows:

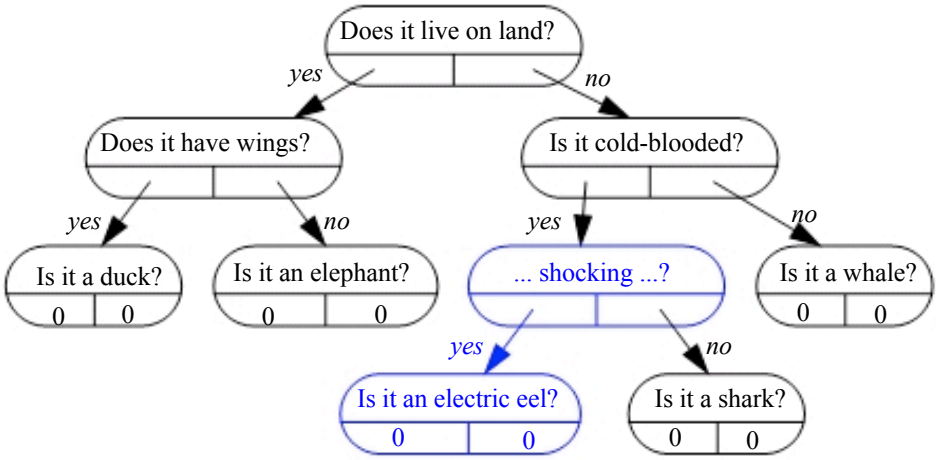


Such a structure is called a **decision tree** — each node in the tree contains the information needed to make a yes-no decision between its two subtrees. The guesser begins by asking the topmost question and, based on the player's response, follows the *yes* branch or the *no* branch to the next question. The guesser continues this process, descending through the tree until it reaches a leaf node (i.,e., the end of a line of questioning), in which case it asks its final question and guesses the animal stored in that node.

By implementing such a tree as a linked structure, new nodes can be easily inserted(or deleted). Thus, in the first game, the program "got lucky" because it happened that the animal the player was thinking of (an elephant) was one of the three it knew about. However in the second game, the user was thinking of a shark — an animal the program did not know about. Using the information supplied by the player, the program learned to distinguish between a whale and a shark by creating and inserting the nodes that will allow it to distinguish between the two animals:



Similarly, in the next game, the program learned to distinguish between a shark and an electric eel:



The program thus "learns" by expanding its decision tree each time it encounters an animal that is not already there.

To avoid "forgetting" what it has "learned", the `DecisionTree` destructor writes the tree's data to a file when the program terminates, and the `DecisionTree` constructor reads from this same file to initialize the `DecisionTree` when the program is run again. This simple mechanism allows the program to "remember" what it has "learned" in previous games. Over the course of time, such a program can become quite adept at identifying animals, based on the characteristics it is taught.

Attached is the source code for the `DecisionTree` class and the interested reader is invited to study it further.

DecisionTree.h

```
/* This file contains the declaration for class DecisionTree.
-----------------------------------------------------------------*/

#ifndef DECISION_TREE
#define DECISION_TREE

#include <iostream>     // istream, ostream, <<, >>, ...
#include <string>       // string
using namespace std;
```

```
class DecisionTree
{
public:
  DecisionTree(const string& fileName);
  ~DecisionTree();

  int descend();

 private:
  DecisionTree(const DecisionTree&);            // disable copying
  DecisionTree& operator=(const DecisionTree&); // disable assignment

  void read(const string& fileName);
  void write(const string& fileName);

  struct Node
  {
    Node(const string& question);
    Node(const string& question, Node* leftChild, Node* rightChild);
    ~Node();

    string myQuestion;    // the question I store
    Node*  myYesChild;    // where to go if the answer is 'y'
    Node*  myNoChild;     // where to go if the answer is 'n'

  };

  typedef Node* NodePtr;

  void buildTree(istream& InF, NodePtr& NPtr);
  int  descendTree(NodePtr& NPtr);
  void learnSomething(NodePtr& NPtr);
  void writeTree(ostream& OutF, NodePtr NPtr);

  NodePtr myRoot;
  string  myDataFile;
};
```

```
/* Explicit-value constructor
   Receive: fileName, a string.
   PRE:  fileName is the name of a text file containing DecisionTree
         data.
   POST: I am a DecisionTree containing the data found in fileName.
-----------------------------------------------------------------*/
inline DecisionTree::DecisionTree(const string& fileName)
{
  myRoot = 0;             // just in case fileName is empty...
  read(fileName);         // build myself using data from fileName
  myDataFile = fileName;  // save for use by destructor
}

/* Destructor
   PRE: my lifetime is over
   POST: my data (including anything I have learned)
         has been saved to myDataFile.
-----------------------------------------------------------------*/
inline DecisionTree::~DecisionTree()
{
  write(myDataFile);      // write myself back to data file
  delete myRoot;          // delete root node (recursively deleting
everything)
  myRoot = 0;             // unnecessary, but ...
}

/* Node constructor
   Receive: A question for the DecisionTree
   POST: I am a DecisionTree::Node containing question
         && myYesChild == 0 && myNoChild == 0.
-----------------------------------------------------------------*/
inline DecisionTree::Node::Node(const string& question)
{
   myQuestion = question;
   myYesChild = 0;
   myNoChild = 0;
}

inline DecisionTree::Node::Node(const string& question,
                                DecisionTree::NodePtr left,
                                DecisionTree::NodePtr right)
{
  myQuestion = question;
  myYesChild = left;
  myNoChild = right;
}

/* Node destructor
   PRE: my lifetime is over (someone has used 'delete' on my handle)
   POST: both of my subtrees have been deleted
-----------------------------------------------------------------*/
inline DecisionTree::Node::~Node()
{
    delete myYesChild;    // delete yes subtree (recursively)
    delete myNoChild;     // delete no subtree (recursively)
}
```

```
/* method to 'walk' the tree from its root to a leaf
    (really just a wrapper for the recursive method).
   return: 1 if the answer to the leaf-node's question was 'yes'
           0 if the answer at the leaf-node's question was 'no'
   Relies on the recursive method descendTree()
-----------------------------------------------------------------*/
inline int DecisionTree::descend()
{
  return descendTree(myRoot);
}

#endif
```

DecisionTree.cpp

```
/* This file contains the definition for class DecisionTree.
-----------------------------------------------------------------*/

#include "DecisionTree.h"
#include <fstream>            // ifstream, ofstream, ...
#include <cstdlib>            // exit()
using namespace std;

/* read tree-data from fileName
   receive: fileName, a string.
   input(fileName): a sequence of Node values.
   POST: I am a DecisionTree containing the data from fileName.
   Relies on: buildTree()
   Called by: DecisionTree constructor
-----------------------------------------------------------------*/
void DecisionTree::read(const string& fileName)
{
  ifstream fin( fileName.data() );

  if (!fin.is_open())
  {
    cerr << "\nDecisionTree::load(fileName) unable to open fileName: '"
         << fileName << "'; exiting...\n";
    exit (-1);
  }

  buildTree(fin, myRoot);

  fin.close();
}
```

```
/* utility to recursively build the tree by reading from an istream
   Receive: fin, an istream'
            nPtr, the 'root' of a DecisionTree.
   POST: the tree has been built from nPtr 'down' using values
         from fin.
   Called by: read()
-----------------------------------------------------------------*/

void DecisionTree::buildTree(istream& fin, DecisionTree::NodePtr& nPtr)
{
                                // trivial case: fin is empty, do nothing
  if (!fin.eof() && fin.good()) // nontrivial case:
  {
    string question;               // a. read 1 node's data
    getline(fin, question);
    int leftSubtreeExists,
        rightSubtreeExists;
    fin >> leftSubtreeExists >> rightSubtreeExists;
    char separator;
    fin.get(separator);
                                   // b. build a new node for that data
    nPtr = new DecisionTree::Node(question);

    if (leftSubtreeExists)        // c. if necessary
      buildTree(fin, nPtr->myYesChild);  //   build 'yes' node
recursively

    if (rightSubtreeExists)       // d. if necessary
      buildTree(fin, nPtr->myNoChild);   //   build 'no' node
recursively
  }
}


/* recursively 'walk' a tree from its root to a leaf
   return: 1 if the answer to the leaf-node's question was 'yes'
           0 if the answer at the leaf-node's question was 'no'
   Called by: descend()
-----------------------------------------------------------------*/

int DecisionTree::descendTree(NodePtr& nPtr)
{
  char response;
  int result;

  if (nPtr != 0)                          // validate parameter
  {
      do                                  // get a y or n response
      {                                   //  to this node's question
        cout << "\n" + nPtr->myQuestion;
        cin >> response;
      }
      while ((response != 'y') && (response != 'n'));

    // if this is a leaf node
    if ((nPtr->myYesChild == 0) && (nPtr->myNoChild == 0))
      if (response == 'y')           //  and we guessed correctly
```

```
        result = 1;                  //   we won!
      else                           //  otherwise, we lost, so
      {
        learnSomething(nPtr);        // learn about the new animal
        result = 0;
      }
      else                           // otherwise: it's a non-leaf node
      if (response == 'y')           //  so follow the appropriate link
        result = descendTree(nPtr->myYesChild);
      else
        result = descendTree(nPtr->myNoChild);
  }

  return result;
}

/* utility to expand my 'knowledge base';
    invoked when I reach a leaf node and the answer is 'n'.
   Receive: nPtr, the handle of the leaf Node.
   POST: nPtr now points to a new interior node
         && nPtr->myNoChild points to the old leaf (animal)
         && nPtr->myYesChild points to a new leaf (animal)
-----------------------------------------------------------------*/
void DecisionTree::learnSomething(NodePtr& nPtr)
{
                                   // get new animal name
  cout << "\nWhat animal are you thinking of? ";
  char separator;
  cin.get(separator);
  string newAnimal;
  getline(cin, newAnimal);
                                   // extract old animal name
  int end = nPtr->myQuestion.find(" (y or n)?", 0);
  if (end == string::npos)
  {
    cerr << "DecisionTree::learnSomething(): ill-formatted question: '"
         << nPtr->myQuestion << "'; exiting ...\n" << endl;
    exit(1);
  }
  string oldAnimal = nPtr->myQuestion.substr(11, end-10);
                                   // get question to distinguish them
  cout << "\nPlease enter a question, such that the answer is\n"
       << "\tyes - for a(n) " << newAnimal
       << ", and\n\tno - for a(n) " << oldAnimal
       << "\n--> ";
  string newQuestion;
  getline(cin, newQuestion);

  NodePtr tempPtr = nPtr;          // save node containing oldAnimal
  nPtr = new Node(newQuestion + " (y or n)? ");
                                   // make node for new question
                                   // make node for newAnimal
  NodePtr newAnimalPtr = new Node("\nIs it a(n) " + newAnimal
                        + " (y or n)? ");
  nPtr->myYesChild = newAnimalPtr;  // arrange the 3 nodes correctly
  nPtr->myNoChild = tempPtr;
}
```

```
/* utility to write a DecisionTree's data to a data file.
    (mostly just a wrapper for the recursive writeTree() method).
   Receive: fileName, a string.
   POST: All of my data has been written to fileName.
   Called by: DecisionTree destructor.
-----------------------------------------------------------------*/
void DecisionTree::write(const string& fileName)
{
  ofstream fout( fileName.data() );

  if ( !fout.is_open() )
  {
    cerr << "\nDecisonTree::write(fileName): unable to open fileName: '"
         << fileName << "'; exiting...\n";
    exit (-1);
  }

  writeTree(fout, myRoot);

  fout.close();
}

/* Utility to recursively write a node and its subtrees to a file via
   an ostream
   Receive: fout, the stream to which data should be written;
            nPtr, the root of a DecisionTree.
   POST: fout contains the data of nPtr and its subtrees.
   Called by: write()
-----------------------------------------------------------------------*/
void DecisionTree::writeTree(ostream& fout, DecisionTree::NodePtr nPtr)
{
  if (nPtr != 0)
  {
    fout << nPtr->myQuestion << endl
         << (nPtr->myYesChild != 0) << ' '
         << (nPtr->myNoChild != 0) << endl;

    writeTree(fout, nPtr->myYesChild);
    writeTree(fout, nPtr->myNoChild);
  }
}
```

animal.cpp

```
/* animal.cpp plays the game of 'animal', in which the player
   thinks of an animal, and the program tries to guess it.

   If the program is unable to guess the player's animal,
   it "learns" to distinguish the player's animal from its animal
   (with the player's assistance).
-----------------------------------------------------------------------*/

#include <iostream>
#include "DecisionTree.h"
using namespace std;
```

```
bool playMore();

int main()
{
   cout << "\nWelcome to the game of Animal!\n";

   DecisionTree dTree("animal.data"); // load tree-data from file

   do
   {
      cout << "\nThink of an animal, and I will try to guess it...\n\n";

      int winner = dTree.descend();  // 0 == the person, 1 == the program

      if (winner)
         cout << "\nHa! Even a computer program can beat you...\n";
      else
         cout << "\nYou were just lucky...\n";
   }
   while (playMore());
}  // tree-data is auto-saved to file by DecisionTree destructor


bool playMore()
{
   char answer;
   cout << "\nDo you want to play again (y or n)? ";
   cin >> answer;
   return ((answer == 'y') || (answer == 'Y'));
}
```