CSCI 262 Data Structures

Spring 2019

Project 5: Animal (Twenty Questions)

(Back to Assignments)

Purpose

- To practice implementing and working with binary trees

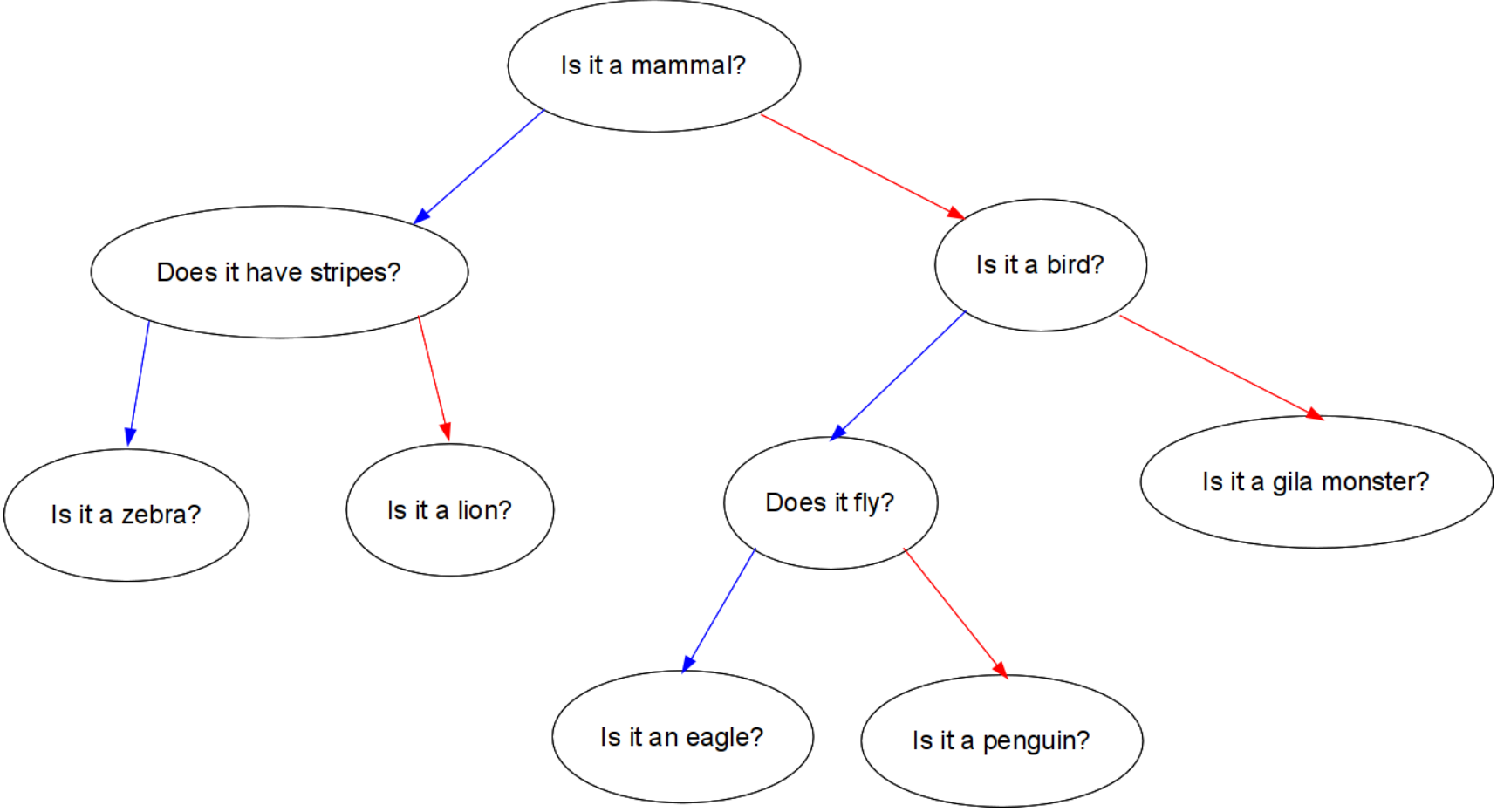
Before You Begin

- Read the sections in their entirety before you start programming!

Introduction

If you've never played twenty questions before, you can read the wikipedia article for background. In essence, one person selects some specific thing which the other person will try to guess by asking yes/no questions. In our version of the game, we will assume the thing to guess is always some type of animal (e.g., "Zebra"), but in fact the game we will build can be easily extended to cover other kinds of things.

In this assignment you will be building a program which tries to guess the user's selected secret using a **binary decision tree**. In this kind of tree, every node represents a question, with the user's yes or no answer determining which child node to transition to. A game tree for a simple game of "animal" twenty questions would look like:



Note that a "yes" answer always goes to the left child (blue arrows), while a "no" answer always goes to the right child (red arrows). The leaves of the tree are terminal guesses, therefore they should represent very specific guesses of a type of animal. The internal nodes are more general, and are designed to separate larger categories of animals in order to narrow down. Note that all internal nodes must have two children, one for yes and one for no!

In this assignment, you will implement a learning version of twenty questions: one that not only plays the game, but learns new objects when it loses. You'll be able to save your learned tree to disk, so that you can exchange it with a friend, or start up again where you left off.

Twenty Questions is all about working with trees: building them, manipulating them, writing them to disk, reading them from disk, and everything else.

Outline

There are four principal parts to this assignment, detailed below.

Part 1: Reading the game tree

The first thing your program must be able to do is to read in an existing game decision tree from file. The decision trees are stored in file using a pre-order traversal. This means that the root is stored first, then the left subtree, then the right subtree. Note the recursive structure! You can read the file in using a pre-order

recursive traversal. The only trick is that we must distinguish between internal nodes (which have two children and require a recursive effort to read them in) and leaves (which form the base case of the recursion).

In our implementation each node is stored on a separate line, and each line starts with an indicator of which type of node it is. Internal nodes ("questions") begin with "#Q", while leaves ("answers") start with "#A". See the provided game tree file, animal.txt, for an example. (This file corresponds to the game tree illustration above - see if you can trace how to read it using a pre-order traversal.)

While technically you could leave the tree as a list in pre-order format, extending the game tree (part 3) will be substantially easier if you work with a binary tree implementation such as we discussed in class.

Part 2: Playing the Game

You will create a user interface and game logic to play the game of twenty questions using the game tree read in in part 1. After the user indicates that they wish to start the game, the game begins by displaying the first question, the one stored in the root of the tree, and prompting the user for a yes/no input. If the user answers yes, your program should continue by asking the question stored in the left child node, otherwise by asking the question in the right child node. You will need to maintain a pointer to the current node, moving it to the appropriate child node with every user answer. When the last question is a leaf, a yes answer will result in displaying a self-congratulatory "I win" message. Otherwise, the game should indicate that it is out of questions and has thus lost the game. At this point, the game should also prompt for an extension of the game tree (see part 3 below).

Here's a transcript of a successful game played using the provided game tree:

```
Welcome to 20 questions!
  1) Play the game
  2) Save the game file
  3) Quit
Please make your selection: 1
Is it a mammal? (y/n): n
Is it a bird? (y/n): y
Does it fly? (y/n): n
Is it a penguin? (y/n): y
YAY! I guessed your animal!
```

Part 3: Extending the Game Tree

When the game fails to guess the user's animal, the game should prompt the user to enter two new questions. The first question entered will be the question which would have caused the game to win twenty questions. E.g., if the user's secret animal is a platypus, the user should enter the question "Is it a platypus?". The second question distinguishes the new animal from the old animal. For instance, the second question might be "Does it lay eggs?".

Here's a transcript of a game in which a new animal and distinguishing question is added to the game tree:

```
Welcome to 20 questions!
  1) Play the game
  2) Save the game file
  3) Quit
Please make your selection: 1
Is it a mammal? (y/n): y
Does it have stripes? (y/n): n
Is it a lion? (y/n): n
BOO! I don't know!

Would you like to expand the game tree (y/n)? y
I asked the following:
Is it a mammal? YES
Does it have stripes? NO
Is it a lion? NO
Enter a new animal in the form of a question,
e.g., 'Is it a whale?':
Is it a platypus?
Now enter a question for which the answer is 'yes' for your new
animal, and which does not contradict your previous answers:
Does it lay eggs?
```

When the new questions are entered, the game needs to modify the game tree. Note that the last question answered is at a leaf. To expand the game tree, the question at the leaf ("Is it a lion?" in the example above) needs to be replaced with the new yes/no question ("Does it lay eggs?" in the example above). Then the old animal question ("Is it a lion?") and the new animal question ("Is it a platypus?") have to be added as new nodes to the game tree.

Once the new nodes are in the game tree, the user should be given the option of playing the game again (with the new questions) and/or saving the game tree to file.

Part 4: Saving the Game Tree

Saving the game tree is simply the reverse of reading in the game tree. You should provide a menu option for this purpose. To write the game tree out, simply do a pre-order traversal of your binary tree. For each node, write out either "#Q" or "#A" as appropriate, a space, and the question stored in the node. Be sure to close the stream after writing!

Implementation Hints

You can implement twenty questions however you like, but this section gives some how-to hints which you may find useful. Also, here is a rudimentary animal.cpp file to get you started (you don't have to use it if you don't want to).

Storing the Game Tree

- Keep a pointer to your root node in main, then pass the pointer to all of your other functions.
- For every node in the tree you need two pieces of information: the question that it represents, and whether or not it is a leaf. You could create a class or struct for these two pieces of information, or you could just store the question as a string, and use the fact that any leaf node will have two NULL children (and any non-leaf node will have two non-NULL children).
- When you are done with the game, you should probably clean up your tree (call delete on all of the nodes) using a post-order traversal. This is good practice. However, since you are only quitting the program at this point, technically it isn't necessary.

Reading the Game Tree

Create a **recursive** function

```
void read_preorder(node<string>* tree, ifstream &fin);
```

Inside your read_game_tree() function, you should create an empty node for your root and open an ifstream on "animal.txt". Then pass the pointer to the empty root node and the open ifstream object to your read_preorder() function.

Inside read_preorder(), you'll first want to read a single string from the input stream, which will contain either "#Q" or "#A", depending on which kind of node you are creating (internal or leaf). Read in a single char (or use fin.ignore()) to get rid of the space, then use getline() to get the rest of the line, which will be the question. Store the question in tree->data.

Now, if the first string was "#A", you know you have a leaf node, and this is the base case of your recursion. If the first string was instead "#Q", then you need to create an empty node for each of tree's two children. Then recursively call read_preorder on the left and right children.

Be sure to close your input stream after reading the game tree!

Writing the Game Tree

Just like reading the game tree, create a recursive method which takes a file stream (an ofstream in this case) and a node pointer. Write the file out using a pre-order traversal. Close the stream after writing!

Playing the Game

- Use a node pointer to keep track of the current question. When not on a leaf node, you'll set this pointer

to the left child when the user answers yes, and to the right child when the user answers no.
- Use a queue to record a history of the questions asked and the user's answers. You'll need this when prompting the user for a question to expand the game tree with. (You can just record the history of the user's answers, then reconstruct the path from the root pointer based on those answers.)

Downloads

- animal.txt
- animal.cpp

Grading:

Code compiles and runs:	5 points
README	5 points
Parts 1 and 2 (Reading the game tree and playing the game)	20 points
Part 3 (Expanding the game tree)	20 points
Part 4 (Saving the game tree)	10 points
Total:	**60 points**
Extra credit: innovations of your own design	Up to 3 points

Documentation:

README:

1. Include YOUR NAME and the names of all people who helped/collaborated as per the syllabus
2. Describe the challenges you encountered and how you surmounted them
3. What did you like/dislike about the assignment?
4. How long did you spend on this assignment?
5. A description of any novel features you added for extra credit

Submit a zip file on Canvas containing:

- README
- All source code (.cpp and .h files)