CS 346 – Lab #10 – File System Traversal

Today, let's have some fun with files. We will use the Java API in order to look up directory information about files and to traverse the file system. To begin, create a new directory called lab10 in your account. Please copy the file Demo.java from my folder here:

/home/chealy/www/cs346/lab10/Demo.java

Part 1 – directory information

Compile and run the Demo program. It will ask you to enter the name of a directory (folder). The output will be a detailed listing of files in that directory, along with several items of metadata about each file. Note that you can use either a relative or absolute path when telling your program which directory to examine. For example, you could specify ".", "..", "../lab09", and so on.

The Java API includes a class called File, which we will rely on in this lab. As you can see in Demo.java, we call many useful methods from the File class. Among these are:

- `isFile()`
- `length()`
- `getCanonicalPath()`

According to the online Java API documentation, what do these methods return?

Part 2 – traversing the file system

Now that you have seen some useful File class methods, let's put them to good use. In real life, people accumulate files on their computer, and become surprised when they see the total amount of disk space consumed. What is taking up all this space? Perhaps there are some huge files that aren't needed anymore. Or, maybe there are duplicate copies of big files that are unnecessary. We can write a program that could assist someone who is curious about their big files.

Write a program that asks the user to enter the name of a directory. The program will then traverse (i.e. visit) all of its files and subdirectories in order to determine:

- The 25 largest files, in descending order of size. If there are fewer than 25 files, then just list all the files.
- A listing of all pairs of files that have the same size (equal number of bytes). Let's limit this feature to only the "big" files that are at least 100 KB in size.

The key to this program is that it needs to traverse the hierarchical nature of the file system. This is because folders may contain other folders, which in turn may contain more folders, and so on. The program will need to be recursive. In fact, the general technique of traversing a tree in this fashion is sometimes called "recursive descent."

Implementation hints:

Overall, your program needs to maintain a list of information about files. For each file you encounter, it is useful to know its name, its "canonical path" long-form name, and its size (length) in bytes.

Because the traversal needs to be done recursively, it makes sense to write a separate recursive function to do the traversal. It would have the following structure:

```
Function traverse (with File object parameter "f"):
  Look up the name and canonical path of this file object
  If it is a file,
    Add it to your list of files
  Else if it is a directory,
    Create an array containing the list of files in this directory.
    For each file in this array, recursively call traverse on this file object.
```

```
Main program:
Ask the user to enter the directory from which to begin the search.
If the file object is not a directory, the program should gracefully exit.
Call the traverse function on the file object corresponding to this directory.
When the function returns, you should have a list of file entries.
Sort this list of files in descending order of size.
Print out the first 25 files in this list.
Write a second loop to traverse the file list.
  If file i and i+1 have the same size and this size is at least 100 KB,
    print out the size and the canonical paths of both files.
```

Progress Information

It's possible that the user will ask your program to traverse a rich collection of many thousands of files. And this traversal would take some time. In order to show reassuring progress information for the user, your program should include a statement that checks the number of files encountered in the traverse function. If this number is a multiple of 5000, then your program should print how many files it has seen so far. Otherwise, if your program churns away for several seconds without output, the user might wonder if something has gone wrong.

Output format:

When you print the top 25 files by size, print each file one per line. Print the file size (with enough room for a 10-digit number), followed by a space, followed by the canonical path name of the file.

When you print pairs of identical-sized files, the format should be similar: Print the file size, allocating space for up to a 10 digit number, followed by a space, followed by the canonical path name of one of the files in the pair. On the next line, print the second file directly under the first. In order to get the file names to line up, you will need to indent by 11 spaces.

Testing

Consider how you would test your program. You would use your own directories as input. For example, it may be a good idea for you to manufacture some large files that are the same size, so that your program can discover them.