Computer Science 62
Lab 5

Wednesday, February 24, 2010

Today's lab is an experiment in searching the web, the sort of thing that Google and Yahoo do every day. The lab is designed to contrast the breadth-first and depth-first search strategies.

We will use the `HTML` class written by Duane Bailey. It supports a constructor that takes a url in the form of a `String`.

```
HTML webpage = new HTML("http://www.cs.pomona.edu");
```

There is a `toString` method that returns the url as a `String`. The class `HTML` is iterable, and its iterator produces the links on the page—in the form of other `HTML` objects. These are the only features of the class that we will need today. From the point of view of our search strategies, all we need are positions which tell us (through an iterator, in this case) what the "next links" are. The `HTML` class has other features which you can discover by reading the code.

For today's exercise, to keep life simple we are *NOT* being a **good** crawler. A good crawler will follow the "Robots Exclusion Standard" (see `http://www.robotstxt.org/`), but Bailey's class is not aware of those boundaries. In addition, we are not making an effort to avoid fetching web pages from the same server too frequently.

Because of this:

- We will stop our search when our visited list has more than 20 or 30 members
- Try and run each of your experiments only once and then copy and paste the results somewhere to save for comparison.

- Confine the use of this code to within the lab today. If you want to learn how to crawl properly, I'd be happy to help you out and point you towards some more resources.

1 Getting Started

In Eclipse, open a new project and close the others. Copy over Bailey's HTML class and our various interfaces to the project by copying them from `/common/cs/cs062/labs/lab5/` to the `src` directory for your project. Then, in Eclipse select `File/Refresh`. Next, open a new Java class `WebCrawler` into which you will put your code for today.

2 Generic Search Method

In class, we saw general search code that can be changed to be either breadth-first search or depth-first search based on whether we use a stack or a queue. Start by flushing out the details of this search method to work for our situation (the basic code can be found in the class notes). The method header should be:

```
private static List<HTML> search(HTML startingPoint, Linear<HTML>
pending)
```

You'll need to make a few changes:

- support `HTML` objects (rather than generic E)
- choose any iterable list for `visited`
- when a page is visited (i.e. right before being added to the visited list) print out the url to the console (System.out.println(position))
- Your while loop (and therefore method) should terminate after visited reaches 20 or 30 elements.

Notice that you will be passing in the pending data structure, which will allow us to easily implement BFS and DFS.

3 BFS

We begin with a breadth-first search of a part of the Internet. Implement `bfs` as a (static) method in your class. It will take an `HTML` object and return a list of `HTML` objects. Choose one of the queue data structures discussed in class (you'll need to make a new class and copy the code from the notes on the course web page). use your `search` method above to implement `bfs`. Your `bfs` method should only be **one** line of code!

Type in the short `main` method below and use it to carry out `bfs` from a couple of sample url's.

```
public static void main(String[] args) {
    HTML root = new HTML("http://www.cs.pomona.edu/");

    bfs(root);
}
```

Copy and paste the results from each of your crawls somewhere for safekeeping.

4 DFS

Do the same thing for `dfs`, substituting a queue for a stack.

Run the depth-first search on the same url's as you tried above. Be sure that you can explain differences between the results of the two kinds of searches. Copy and paste these results and save them away as well.

5 Recursive DFS

Create another new method `recursiveDfs` that will carry out depth-first search recursively. Try it by yourself, but if you get stuck, I've included a generic version in the class notes, but you'll still need to modify it for our situation. There will be no loop and no `pending` stack. The `add` operations will become recursive calls, and the argument will replace the removed value. (Also, the `visited` list will have to be an additional argument. Why?)

Run the recursive depth-first search on the same url's. Are there any differences from the non-recursive searches?