CSC 210 Program #4

www.hashtable.com
Feb 13, 2008

Logistics

Program #4 www.hashtable.com is:

- **Worth** 5 points, or 5% of your grade
- **Due** Friday Feb 22, 2008 (end of the day)
- **Covers** hash tables

Description

In Program #4, you'll implement your own hash table. Web site URL's and IP addresses will provide a nice testbed for some hash table experimentation.

So, you're surfing the net with your favorite browser. You enter a web site name, called a Universal Resource Locator (URL), and load in the web page. That's easy enough.

Well, under the hood, each URL must be converted to an Internet Protocol (IP) address so that the web site can be located on the net. For example: www.wikipedia.com has an IP address of `207.142.131.248`.

The trick is we need this lookup to happen whether there are 100 URL's, 100 thousand or 100 million or 100 billion. A hash table is just the ticket!

In Program #4, we will simulate this process and use our very own hash table to do it. Here are the steps your program should take:

1. Load a file of "fake" URLs and IP addresses that I have created for you.
2. Query the user for a URL.
3. Find the corresponding IP address for the URL in your hash table.
4. If the IP address was found, then display it
5. Else ask the user if he/she would like to add the URL to your table. If yes, then ask the user for a corresponding IP address for the URL and add it.
6. Goto step 2.

Query the user in a tiny GUI… your choice here. Don't spend your money on a fancy GUI though… spend it on getting your hash table to be nifty.

Hints

In Program #4:

- Implement your own generic hash table class with get, put, and remove methods. The book has a nice partial implementation.
- I simplified things (old school) by combining my key and value. It's easier and less expensive memory-wise. So, my generic hash table class is: `public class MyHashTable<E> { … }`
- I used chaining to resolve collisions. I think I used a `LinkedList`. You can do linear probing if you want.
- Your program should support 2 hash function mechanisms, similar to `Comparable`'s `compareTo()` method and the `Comparator` class for ordering objects. Your two ways of hashing should be:
    a. Override `hashCode()` with your own hash function
    b. Create a `Hasher<E>` interface that specifies a hashing function
- When you implement your hash function, please use Horner's Rule. I'll discuss this speed-up in class. Here are two links too (tutu?):
    - http://en.wikipedia.org/wiki/Horner_scheme
    - http://mathworld.wolfram.com/HornersRule.html
- Your hash table should allow the user to specify a load threshold, a value between 0 and 1. Resize and rehash your table whenever its occupancy exceeds the load threshold.
- Write a `main()` to test your hash table on small examples before moving to the big web sites file.
- Once your hash table is working, add support for reading a file of web sites and querying it. The web sites file has 20,000 URL-IP pairs. The file is on the k: drive and here: websites.txt. I also have a snippet of code to read the file in README.txt on the k: drive… just a little time-saver.
- And of course, add a tiny GUI to play with this web site/hash table stuff.

You'll likely have a generic array in your program somewhere. If you get a lint warning regarding this, you can ignore it. Ask me about this in class.

For your creative flair on Program #4, you can:

- Keep track of (and report) fun statistics like #collisions, #rehashes, performance overall, etc.
- You can compare your hash table to another data structure, like a BST or something.
- You can learn how to use Junit in NetBeans… see me if you want help on this.

- You can also learn the NetBeans UI builder and show it off to the class.
- Or the best, of course, is something you feel compelled to try.

Dang.

Start early, and good luck!

hash this… yow, bill

Grading

“Same as it ever was”

- David Byrne

1. **Your README file** – describing the state of your program... what works, what doesn't, what your “flair” is, etc.
2. **Your Net Beans folder** – including your Java source code, class files, etc.
3. **Your Javadoc** – generate using the “Build/Generate Javadoc” menu
4. **Your applet** – create a web page at w:/index.htm with an Applet of one of your favorite trees. Please include a link to your Javadoc.
5. **Your printout** – Please print one source file… your most important one, so that I have someplace where I can scratch my comments.

I expect your code to be beautiful. “No crappy code”™

Additional Notes

I’ll post additional notes directly on the blog.