COMPSCI 220 S1 C   Assignment 3

Department of Computer Science

The University of Auckland

Due Sunday, 17 May 2015, 8.30 *pm*

This assignment is worth 18 marks; it represents 6% of your total course grade.

In this assignment you will develop program that models a distributed hash table (DHT) as an array of eight separate SimpleHT tables.

You are given two Java files, a3_q1.java and a3_q2.java, containing all the classes you will need; your task is to modify these files to answer questions 1 and 2 below.

You are also given an input file, dns-data-in.txt, to use as input to both parts 1 and 2. For each part, you are also given sample output files, a3_q1_out.txt and a3_q2_out.txt. Use these to check that your modified programs produce output which matches these files.

All these files are available in a single zip file on 220's Cecil Resources page.

Part 1: Distributed Hash Table Program [9 marks]

The two Java files provide a hash-table class, SimpleHT. SimpleHT is a simple-to-use class that implements a hash table using Java's Hashtable class. That class has rather different methods to those discussed in lectures. SimpleHT implements a hash table, using a string as its key, and a single String as that key's value. [As you'll see from the source code, if the key isn't already in the table, we insert its initial value; otherwise we set a new value for it.]

Your input data is a set of DNS records from the U Auckland network, showing requests from computers outside our network, and the (anonymised) source Internet address they came from. Use the first (domain name) field as the key for your DHT. The source Address for the domain name will be stored as the value corresponding to the domain name.

The input file has the following format:

```
Each record (line) has two blank-separated fields:
  domain_name source_address,  e.g.
student.auckland.ac.nz. 52.125.122.229
artsnet.auckland.ac.nz. 181.132.131.229
ec.auckland.ac.nz. 173.119.112.26
```

a3_q1.java reads the data file, calling update() in the DHT for each inout record.

1. a3_q1.java provides the SimpleHT class, and a preliminary version of the DHT class – our model of a distributed hash table.

CONTINUED

a3_q1 reads data from standard input line by line, looks up the record for each line's key, and sets its value to the source address. Your version of the DHT class must implement its find() and statistics() functions. For question 1, you must only store a value for each key in a *single* SimpleHT.

At end-of-file, print out

- the number of lines read from standard input
- the number of entries in each of the eight SimpleHTs (i.e. the SimpleHT sizes)
- the total number of entries in the DHT
- the maximum and minimum SimpleHT sizes
- the difference between max and min SimpleHT sizes

2. Test your program using the sample input data file;
Note: for marking, your program will be tested using a larger data file.

Part 2: Making the DHT resilient [9 marks]

In part 1 above you only stored the value for each key in a single hash table. For this part, you are to store it in enough of the hash tables to survive having *two* of them unavailable.

1. a3_q2.java uses an ArrayList to store all the keys (domain names), it uses that to check that it can read all the records. Copy your changes fo find() and statistics() over into a3_q2.java.
2. Change your code for DHT.place() to implement redundancy. If your current DHT code stores a record in table t, store it in tables $t + 1$ and $t - 1$ as well.
3. Change your code for DHT.find() to retrieve a record from any of the three tables you stored it in.
HINT: DHT.loose() makes a SimpleHT unavailable by setting its entry in the SimpleHT array to null.
4. Now disable tables 6 and 7. Make sure that a3_q2 now gives output that matches a3_q2_out.txt.

Questions involving programming

- You are expected to use Java. However, for this assignment you may also use python. If you do that, use Python dictionaries instead of SimpleHTs, and note that automarker expects a python3 source file to have a name ending in .py3.
- A sample input file, and output files for each question are given. The marker checks your output with a text comparison program, so it must be in EXACTLY the right format.
- You must submit your answer via the automated feedback and submission system ("automarker"). You may take account of the feedback given by the automarker, and resubmit before the deadline without penalty. There is a limit of 15 submissions for each subproblem.
- Your program(s) may be tested on much larger input files. Marks will be allocated for correctness of the programs. Simply "passing" the automarker may not guarantee maximum marks, but it will guarantee full marks for correctness.

CONTINUED

Input and output format

Pay attention to line breaks and beware of nonstandard software. For best results, use a Linux environment such as login.cs (the automarker does).

The format of the input and output files is specified above, and is used for the sample input and output files. Remember, the automarker expects your output file to match the sample output *EXACTLY!*

Dates and Marks

The due date is Sunday, 17 May 2013, 8:30 pm.