Assignment 3: Hash Tables
CS 758/858, Fall 2014
Due Monday, September 22 by **1:40pm**

Implementation

The skeleton code on the course web page is the start of a program that reads text from standard input and then generates random text in the same style on standard output. This is done by counting, for each adjacent pair of words in the input, the different words that come directly after it. For example, if the input is "w_1 w_2 w_2 w_2 w_1 w_2 w_2", then we get the following table of counts:

previous context	next word	count/total
w_1 w_2	w_2	2/2
w_2 w_2	w_2	1/2
	w_1	1/2
w_2 w_1	w_2	1/1

To generate text in this style, if we have already generated "$w_2 w_2$", then we have a probability of 0.5 of generating w_2 and a probability of 0.5 of generating w_1, but if we have generated "$w_1 w_2$", then we always generate w_2. (This sort of model is called 'a finite-state stationary Markov chain over tri-grams.') Rather than storing a large, mostly zero, three-dimensional array of size $|\text{vocabulary}|^3$, the program uses a dictionary data structure to store only those triples that are found in the input.

The skeleton code implements the `linkedlist` dictionary data structure. Complete the program by implementing an improved dictionary data structure that uses a hash table instead of a linked list in order to speed up lookup. Implement the following hash functions:

add3 adds the ASCII values of the first three characters of the word

kr the first hash function given in the lecture slides (the one that uses `mod`)

cw the second hash function given in the lecture slides (the one that uses a table)

Then measure their performance.

The skeleton program reads text from a file. We supply some text pulled from `www.gutenberg.org`, but feel free to experiment with other inputs. Command-line arguments control how many words of text are generated, which data structure is used to store the model, and, for the hash table, which hash function is used and an optional flag to dump hash bucket occupancy statistics. The output is the generated text, and optionally some information about the hash table.

Testing

On the course web page, we supply some utility programs and input files. Most of the programs we distribute in this class will tell you their command-line arguments if you run them with the `--help` option.

`babble` is your program. If you wanted to babble 50 words based on a file using a linked list as the dictionary:

```
./babble linkedlist 50 < file.txt
```

If you wanted to babble 50 words based on a file using a hashtable as the dictionary:

```
./babble hashtable <hashfun> 50 < file.txt
```

`babble-harness` runs your program and optionally displays a plot of the distribution of values across hash table buckets. The full usage is `babble-harness [-d] [-o <file prefix>] [--infile <filename>] [-a <hashfun>]+ [-v <verb>] <counts>`.

- **-b** The add binary name (default: ./babble)
- **-d** Display the preformance profiles to the screen.
- **-o** Output PDF files with the given prefix.

--infile Source text file.

-v Set the verbosity.

-a Add the hashfunction to the list of hashfunctions to test

-help Display this list of options

--help Display this list of options

To run the babbler using babbler.c as the training text, using the cw and kr hash functions, and generating a plot to blah.pdf: `./babble-harness --infile babbler.c -a cw -a kr -o blah.pdf`

Written Problems

1. Is there anything special that we should know when evaluating your implementation work?
2. What can you conclude about the performance of the various algorithm implementations from your experiments? Please include your plots in your submission (electronic and hardcopy).
3. Exercise 11.1–4 from CLRS.
4. (858 only) Exercise 11.2–6 from CLRS. Hint: if the chance of an event occurring at each trial is a/b, then the expected number of trials needed until the event occurs is b/a.
5. Exercise 11.3–1 from CLRS.
6. Exercise 12.1–2 from CLRS.
7. Exercise 12.2–4 from CLRS.
8. (858 only) Exercise 12.2–5 from CLRS.
9. Exercise 12.3-3 from CLRS.
10. What suggestions do you have for improving this assignment in the future?

Submission

Electronically submit your source code using the `~cs758/scripts/sub758` script on agate. Please make sure that your code (as submitted) compiles on agate with the makefile that you supply. Please make sure that your code runs with the harness because this will be used to grade your assignment.

Also, hand in a listing of your source code (2 pages per page, as with `a2ps -2`), along with your written work, to the TA in class.

Evaluation

In addition to correctness, your work will be evaluated on clarity and efficiency.

Tentative breakdown:

2 hashtable implementation

8 written problems