**Assignment 3: Hash Tables**
**CS 758/858, Fall 2024**
Due at **11:30pm on Wed Sep 11**

## 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. This is done by counting, for each adjacent pair of words in the input, the different words that come directly after it. (In other words, a 'Small Language Model'.) 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 "$w_2w_2$", 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 "$w_1w_2$", 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 `array` and `linkedlist` dictionary data structures. You can use these as examples showing what functions you are supposed to implement and how they are supposed to work. Complete the program by implementing an improved dictionary data structure that uses a hash table instead of an array or 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 (multiplicative hashing using $\times$)

**tab** the second hash function given in the lecture slides (tabulation hashing using a table)

Then measure their performance. Be sure that your load factor stays upper-bounded by a constant.

The skeleton program reads text from stdin. 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 generated text is sent to stdout and optionally some information about the hash table is dumped to stderr.

## Testing

In addition to the skeleton code, the course web page has a test harness and reference solution. Sample data is available on `agate` in `~cs758/data/asn3/`. 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 an array as the dictionary:

```
./babble array 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 generates a plot of the distribution of values across hash table buckets. This is currently being rewritten — an explanation of command line usage will be given in recitation (or try '`--help`').

**Written Problems**

1. Briefly summarize which parts of your program are working or not. Include transcripts or plots showing the successes or failures. Is there anything else 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. What suggestions do you have for improving this assignment in the future?

**Submission**

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.

Electronically submit your assignment as described in the instructions from the programming tips sheet on the course web page, using the assignment name `3-undergrad` or `3-grad`, depending on whether you are enrolled in 858. Your files should include your `written.pdf` as well (feel free to use LaTeX or to scan/photograph handwritten work) and plots showing the performance of your implementation.

**Evaluation**

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

Tentative breakdown:

**8** hashtable implementation

**2** written problems (**3** for 858)