CS 211: Computer Architecture, Spring 2015
Programming Assignment 2: Count Unique Addresses

Due: March 1st, 5pm

1 Introduction

This assignment requires you to use hash table to count number of unique addresses in the trace of the memory addresses performed by a program. You need to implement a program called `count`. The input of `count` program is a text file with 64-bit addresses and the output is the number of unique addresses in the trace.

2 Problem specification: Use hash table to count number of unique addresses

2.1 Hash table with chaining

In this assignment, you will implement a hash table with chaining. An important part of a hash table is collision resolution. In this assignment, we want you to use chaining, which is different from Assignment 1. Figure 1 shows an example of using separate chaining in hash table. More information about chaining with linked list can be found at Wikipedia, `http://en.wikipedia.org/wiki/Hash_table#Separate_chaining_with_linked_lists`.

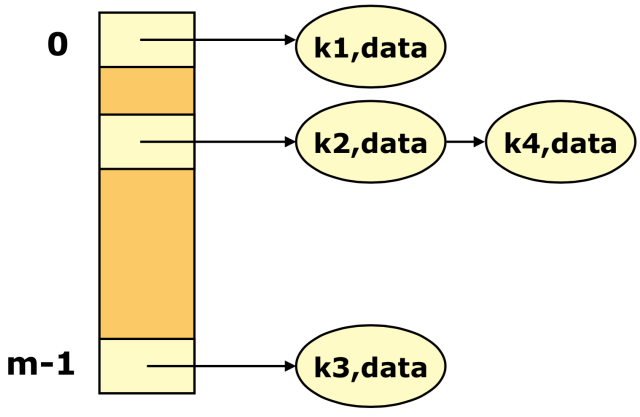

Figure 1: Example of hash table with separate chaining. Use a linked list to chain on collision.

2.2 Count number of unique addresses

Input format: This program takes a file name as argument from the command line. The file can have several lines to millions of lines. Each line contains an address in hexadecimal format, ie `0x7f1a91026b00`. You can use `pintool` (`http://pintool.org`) to generate the list of addresses accessed by a program. The address is represented as 64-bit hexadecimal.

Output format: Your program should print number of unique addresses in the file. There should be no leading or trailing white spaces in the output. Your program should print "error" (and nothing else) if the file does not exist.

Example Execution:

Lets assume we have 3 text files with the following contents. "file1.txt" is empty and,
file2.txt:
0x7f1a9804ae19
0x7f1a9804ae1c
0x7f1a9804ae1c
0x7f1a9804ae1c

file3.txt:
0x7f1a9804ae19
0x7f1a9804ae1c
0x7f1a9804ae1c
0x7f1a9804ae19
0x7f1a9804ae16
0x7f1a9814ae1c

./count file1.txt
0

./count file2.txt
2

./count file3.txt
4

./third file4.txt
error

2.3 Scalability

In this assignment, we will test your program with millions of lines of addresses. You should initialize the hash table with 1000 entries. The largest test case contains 5 million lines with approximately 15,000 unique addresses. In this case, the average number of nodes in each linked list is 15.

2.4 Coding and Testing

We have already provided you the tar with the appropriate directory structure. When you untar it, you should have a directory pa2 with a count subdirectory.

You have to add your implementation by editing the `.c` and `.h` files in the subdirectory count in the pa2 directory.

We have provided the autograder for you to test your final submission as a separate tar file. Please notice that the test cases in autograder are different from above examples.

The autograder is a python script that you can use to grade your assignment. It will compile and test with the test cases inside the directories. To test while you are working, use the command (assuming autograder.py and pa2 are in the same directory):

```
python autograder.py
```

You need to edit the `.c` and `.h` files in the subdirectories count for the assignment. You also need to edit Makefile.

Once you are done with the assignment, create a tar file (see below in Section 3). You need to upload this tar file to sakai. If you want to grade your tar file before submission, use the command

```
python autograder.py tar
```

To add a test case, you have to add 2 files. The file testn.txt where n is a number will contain the test input and resultn.txt will contain the correct output. Note that, 'n' can be any digit number. If you need further help, visit an office hour or post a question in sakai discussion forums.

3 Submission

You have to e-submit the assignment using Sakai. Your submission should be a tar file named `pa2.tar`. To create this file, put everything that you are submitting into a directory (folder) named `pa2`. Then, `cd` into the directory containing `pa2` (that is, `pa2`'s parent directory) and run the following command:

```
tar cvf pa2.tar pa2
```

To check that you have correctly created the tar file, you should copy it (`pa2.tar`) into an empty directory and run the following command:

```
tar xvf pa2.tar
```

This should create a directory named `pa2` in the (previously) empty directory.

The `pa2` directory must contain folder `count`. The `count` folder must contain c source files, header files and a Makefile. For example, after typing make, your program must generate an executable file called `count`.

- `Makefile`: there should be at least two rules in your Makefile:
  1. `count`: build your `count` executable.
  2. `clean`: prepare for rebuilding from scratch.
- source code: all source code files necessary for building `count`.

4 Grading Guidelines

This is a large class so that necessarily the most significant part of your grade will be based on programmatic checking of your program. That is, we will build a binary using the Makefile and source code that you submitted, and then test the binary for correct functionality against a set of inputs. Thus:

- You should make sure that we can build your program by just running `make`.
- You should test your code as thoroughly as you can. *In particular, your code should be adept at handling exceptional cases.* For example, programs should *not* crash if the argument is not a proper number or file does not exist.
- Your program should produce the output following the example format shown in previous sections. Any variation in the output format can result **up to 100% penalty**. There should be no additional information or newline. That means you will probably not get any grade is you forgot to comment out some debugging message.

Be careful to follow all instructions. If something doesn't seem right, ask.