Lab Exercise 4 - Profiling and Gprof

CS341L, Fall 2016

Week of October 10

In this lab, we will learn to use the `gprof` profiler to examine program performance. Submit this lab by copying your output to a text file and submitting the text file to demonstrate that you completed the lab.

1 Setup

First, we are going to download a simple program to run gprof on that can be run with a variety of different inputs and options, the n-gram dictionary program described in Section 5.14 of the book. Update your git repository (by running `git pull starter master` as before) and look in the lab4 directory to find it. Once updatd, compile various versions of the dictionary program by running `make dictionary`.

2 Compiling Programs for Profiling

Notice how `make` generated multiple versions of the dictionary, including a version called `dictionary-pg`. If you look carefully at the command that `make` used to generate this executable, it passed the compiler the `-pg` option. This adds extra instrumentation to the executable to gather performance data while the program runs. The resulting performance data will be placed in a file called `gmon.out`.

Run the compiled version of dictionary program as follows to collect some initial performance data:

```
> dictionary-pg -n 2 -file moby.txt
```

Note that with no options, dictionary is very slow, and figuring out why it's slow and how changing its behavior changes its performance is what we want to use the profiler for.

Examining the Generated Profiling Data

Now that we've made an initial run of the program and gathered performance data, we run the program `gprof` to examine the resulting output. Do so by running:

```
> gprof dictionary-pg > profile.txt
```

and then open the file `profile.txt`. The resulting profile has two sections:

1. A flat call profile, that shows which functions take the most time in the program
2. A call chain analysis, which shows, for each function, which functions call it and which functions it calls, along with statistics about how many times each function is called by each caller.

3 Using the flat profile

Examine the flat profile. What function does the default (very stupid) version of dictionary spend most of its time in? Why is this function so slow? Copy the first few entries of the flat profile into your output and your analysis of the result and the associated code.

4 Using the call chain analysis

Run the program with a different configuration and see how the profile changes:

```
> dictionary-pg -quicksort 1 -n 2 -file moby.txt
> gprof dictionary-pg > profile.txt
```

Look at the call profile now - a large chunk of time in the flat profile is spent in `find_ele_rec`, but a non-trivial amount of time is spent in `Strlen`, and this function is called almost 3 millions times! Use the call chain profile to determine what function is responsible for these calls, and what this function is doing that is causing these calls.

5 Getting more experience with gprof

Now remove the last obvious inefficiency and see how things change:

```
> dictionary-pg -quicksort 1 -lower 1 -n 2 -file moby.txt
> gprof dictionary-pg > profile.txt
```

At this point, effort has to go into optimizing data structure usage in the program. The book contains a description of the various optimizations that you can try. The options that control these are:

- `-size N` to change the size of the hash table
- `-find [0,1,2]` to change the version of the linked list function used
- `-hash [0,1,2]` to change the version of the hash funciton used
- `-n N` to change the size of the n-grams examined

Try different options, and see how this effects runtime and the resulting profile. Write a short summary of your findings and submit that as part of your lab writeup.