Lab04

100 Points

Due date:
11:59pm, Monday 02/25/2019 for Tuesday labs.
11:59pm, Wednesday 02/27/2019 for Thursday labs.

Purpose:
The purpose of this lab is to implement experimental profiling on different hash tables and CPU timing.

General Requirements:
For this lab, you are required to implement an experimental profiling on three different hash tables from lab2 and lab3. **First of all, you need to make sure your implementations of the three hash tables are correct**. All hash tables **DO NOT ALLOW** duplicate elements. Hence you need to correct your code first before you can compare their performance. After you verify the correctness of all your hash tables with the provided input file test.txt, you can move on to the performance comparison. In this lab, we will compare the performance for the build and find operations for each hash table. Then we will repeat the experiment multiple times and take the average for each hash table. You are also required to submit a written report to discuss the following:

1- Organization of experimental profiling.
2- Input data generated using the random number generator.
3- CPU time recording in C++.
4- Data recording and analysis.
5- Performance comparison, observations, and summary. (Experimentally determine the complexity of functions for each hash table, and compare them to their theoretical complexities. If they are different, you need to explain why.)
6- Conclusions.

IMPORTANT: Please submit your report in a separate file (e.g., report.pdf).

Random Number Generator:
```
#include <stdlib.h>
#include <time.h>

int num;
/*initialize random seed*/
srand(time(NULL));
/*generate a random number between 1 and 10 inclusively*/
num = rand() % 10 + 1;
```

CPU Timing:

```
#include <time.h>
clock_t t;
/*start the clock*/
t = clock();
/*insert code to be timed here*/
t = clock() – t;
```

==Experimental Profiling==:
table size (a prime number) m = 1,000,003
R = 100,003
k = 20

The number of random numbers being generated are 0.1m, 0.2m, 0.3m, 0.4m, 0.5m. Use the floor function to get a whole number.

Steps:

1. Build all three hash tables with the same table size m and insert 0.1m random numbers into the tables (the range should be from 1 to 5m). Use CPU timing to record the build time for each table.
2. Then generate 0.01m random numbers (the range should be from 1 to 5m), find those numbers in the hash tables, and then record the CPU time for the successful finds and unsuccessful finds for each hash table. The time for successful finds and unsuccessful finds should both be cumulative not the average time for one single find operation.
3. Repeat steps 1 and 2 five times using a different seed each time; then take the average.
4. Repeat steps 1-3 with different number of random numbers being generated 0.2m, 0.3m, 0.4m, and 0.5m.

Expected Results:
File for testing your hash table correctness:
m = 7
R = 5
k = 10
test.txt elements: 16 17 29 11 88 14 25

```
-----------------------------------------------------------
Please choose one of the following commands:
1-   Test HashTables
2-   Performance Comparison
3-   Exit
>1

Open Hashing:
0:  -> 14
1:  -> 29
```

```
2:  -> 16
3:  -> 17
4:  -> 25 -> 88 -> 11
5:
6:

Hash Table with Quadratic Probing:
0: 14
1: 29
2: 16
3: 17
4: 11
5: 88
6: 25

Hash Table with Double Hashing:
0: 14
1: 29
2: 16
3: 17
4: 11
5: 25
6: 88

------------------------------------------------------------
Please choose one of the following commands:
1-   Test HashTables
2-   Performance Comparison
3-   Exit
>2
```

Performance (Open Hashing):

	100,000	200,000	300,000	400,000	500,000
Build	…ms				
Found					
Not Found					

Performance (Quadratic Probing):

	100,000	200,000	300,000	400,000	500,000
Build	…ms				
Found					
Not Found					

Performance (Double Hashing):

	100,000	200,000	300,000	400,000	500,000
Build	…ms				
Found					
Not Found					

```
-----------------------------------------------------------
Please choose one of the following commands:
1-   Test HashTables
2-   Performance Comparison
3-   Exit
>3
Byebye!
```

Submission:
Follow the conventions below to facilitate grading:

Report
- **Please include your report (report.pdf) in your folder before compressing it.**

Source Code
- Place all your source files (*.cpp, *.hpp) and input files in a single folder with no subfolders.
- Name your folder using the convention Lastname_LabX (e.g., Smith_Lab04).
- Include a functioning Makefile inside the folder. (The makefile should also include the clean command.)
- **Verify that your code runs on lab Linux machines before submission.**

Compressed File
- Compress using .zip, .rar, or .tar.gz.
- Name your file using the convention Lastname_LabX (e.g., Smith_Lab04.zip).

Email
- Use the following subject for your email: Lastname_LabX (e.g., Smith_Lab04).
- Send you code to l290w868@ku.edu if you are in one of Lei's sections or to dhwanipandya1401@ku.edu if you are in one of Dhwani's sections.
- Anytime you have a question about the lab, put the word question in the subject of the email.