EECS 560 Lab – Experimental Profiling on Hash Tables

Prof.: Dr. Shontz, GTAs: Chiranjeevi Pippalla, Prashanthi Mallojula

100 Points

Due date:
11:59pm, Monday, 03/02/2020 for Tuesday labs
11:59pm, Wednesday, 03/04/2020 for Thursday labs

Purpose:

The purpose of this lab is to compare the performance of different closed hashing techniques in C++.

General Requirements:

In this lab, you are required to implement an experimental profiling model to compare the performance of two different closed hashing techniques: Quadratic probing and Double Hashing using an array of structures (not arrays in STL) and a timer class. The performance is to be compared for the build and find operations of each hash table. The experiment is repeated multiple times for different input data sizes, and then CPU times for these two operations are recorded and compared. Below are the requirements and specifications.

Data specifications:

Input data: There is no input data file required for this lab. You are required to generate input data using a random number generator. There will 5 different sets of inputs generated with the below specifications:

Random number generator: Sample code is provided below

……

```
#include <stdlib.h>     // srand, rand
….
int random_number;

 srand (time(NULL));   //Initialize random seed: This means every time you run the program it will
                          change the numbers generated

random_number = rand() % 10 + 1;  // Generate random number between 1 and 10:

//Execute the required functionality
```

……

Table size for all hash tables m = 1,000,003 (must be a prime number)
Double hashing value R = 773
Maximum k value k = 25

EECS 560 Lab – Experimental Profiling on Hash Tables

Prof.: Dr. Shontz, GTAs: Chiranjeevi Pippalla, Prashanthi Mallojula

Number of integers to input for the 5 sets: 0.1m, 0.2m, 0.3m, 0.4m, and 0.5m (where m is the table size). These sizes are used so that the load factor will not exceed 0.5. You can use floor(input size) to make the input size an integer.
For example: input size 0.1m where m= 1,000,003
Input size = 0.1 * 1,000,003 = 100,000.3
Input size = **floor(100000.3) = 100,000**

Random number range: Generate a sequence of random integers in the range 1 to 5,000,000 for the given input sizes.

Find random number sequence: The input size for generation of a random sequence of integers to find is 0.01m. (You can use floor(input size) to make input size as integer). The same range should be used for the random number generation.

Hash function = (x) % m, where x=random number generated, m=table size
= (5000) % 1,000,003 = 5000

Hashing details:

If there is no collision, the index should be calculated as per the hash function given above. Otherwise, try resolving the collision using quadratic probing and double hashing as explained below:

Hashing with Quadratic Probing: Use the m and k values that were provided above.

$f_i = i^2$, $\forall i$, $0 \le i \le k-1$, where x = summation of ASCII values and numbers

$h_0(x) = (h(x) + 0^2) \bmod m = h(x) \bmod m$,

$h_1(x) = (h(x) + f_1) \bmod m = (h(x) + 1^2) \bmod m$,
$h_2(x) = (h(x) + f_2) \bmod m = (h(x) + 2^2) \bmod m$,
...
$h_{k-1}(x) = (h(x) + f_{k-1}) \bmod m = (h(x) + (k-1)^2) \bmod m$.

Double Hashing: h^+ function: Use the R value provided above.

$h^+(x) = R - (x \bmod R)$, where $R < m$ is a prime number.

Now, define $f_i = ih^+$.
Observe that,
$h_0(x) = (h(x) + 0h^+(x)) \bmod m = h(x) \bmod m$,
$h_1(x) = (h(x) + f_1) \bmod m = (h(x) + 1h^+(x)) \bmod m$,
$h_2(x) = (h(x) + f_2) \bmod m = (h(x) + 2h^+(x)) \bmod m$,

...

$h_{k-1}(x) = (h(x) + f_{k-1}) \bmod m = (h(x) + (k-1)h^{+}(x)) \bmod m.$

Rehash:
Rehashing is not required since the load factor will not exceed 0.5 for the sizes of data sets being considered here.

Execution Procedure: As you execute the code, the result should be displayed as shown in the **sample output** given below. There are no specific options to be implemented for user interaction. But internally you are required implement the following capabilities:

- Build a hash table and insert elements into the table: Build the hash table by Inserting elements from the generated datasets into the hash table. The CPU time should be noted before and after the build for each input data size. Compute the difference between these two values to obtain and record the build time. This is to be implemented for both hashing techniques simultaneously.
- Find an element in the hash table: A different random data sequence is generated for the Find option for each input set. Sequentially find each element and count the numbers of successful and unsuccessful finds. To be implemented for both hashing techniques.
- The CPU build time needs to be calculated for all the sets of input data and for both hashing techniques.
- The results should be displayed. The code should also exit.

Implementation Procedure: Each step of this lab is explained below. Avoid implementing a single class design. Divide the work into different classes.

1-Organization of experimental profiling:
1. Implement the required hashing algorithms.
2. Generate a suitable set of input data and apply hashing with quadratic probing and double hashing on the hash tables with the generated random data as input. Calculate the build time for each method.
3. Now, generate another sequence of random numbers of size 0.01m **using the same seed** and try to find each element of this generated sequence in each hash table of the two hashing methods. Record the number of integers found and the number of integers not found for each hashing method.
4. Note the build time, number of integers found, and the number of integers not found for each hashing method.
5. Repeat the above 4 steps with different input sizes 0.1m, 0.2m, 0.3m, 0.4m, and 0.5m and with different **random seeds**. **Initializing random seeds will change the set of data generated. Make sure a seed is implemented for each input set.**
6. Display all the values of the build time, the number found, and the number not found as shown in the sample output for each input size.

EECS 560 Lab – Experimental Profiling on Hash Tables

Prof.: Dr. Shontz, GTAs: Chiranjeevi Pippalla, Prashanthi Mallojula

2-Input data generating using random number generator:

Generate random sequences of data for use in building the hash tables. You should set the table size, i.e., m to be 1,000,003. The range of the integers in the random sequence should always be 1 to 5,000,000. For building the hash table for each hashing method, follow the below set of sequences of data:

1 Set: Initialize random seed. The range of data should be from 1 to 5,000,000 and the input size should be 0.1*m.
2 Set: Initialize random seed. The range of data should be from 1 to 5,000,000 and the input size should be 0.2*m.
3 Set: Initialize random seed. The range of data should be from 1 to 5,000,000 and the input size should be 0.3*m.
4 Set: Initialize random seed. The range of data should be from 1 to 5,000,000 and the input size should be 0.4*m.
5 Set: Initialize random seed. The range of data should be from 1 to 5,000,000 and the input size should be 0.5*m.

For finding the data in each of the hash tables**:**

1 Set: Hash Input 0.1*m: The range should be 1 to 5,000,000 and the find input size should be 0.01*m.
2 Set: Hash Input 0.2*m: The range should be 1 to 5,000,000 and the find input size should be 0.02*m.
3 Set: Hash Input 0.3*m: The range should be 1 to 5,000,000 and the find input size should be 0.03*m.
4 Set: Hash Input 0.4*m: The range should be 1 to 5,000,000 and the find input size should be 0.04*m.
5 Set: Hash Input 0.5*m: The range should be 1 to 5,000,000 and the find input size should be 0.05*m.

3-CPU time recording:

Include <time.h>

The CPU time is used to compare the performance of the two hashing methods using the build time. The build time is calculated as below.

1. Generate a random sequence of numbers as explained above.
2. Record the CPU time (Store it) (Start time)
3. Apply and insert each entry from the generated random set of input data into the quadratic probing hash table.
4. Record the current time (end time).
5. Calculate the time to build the hash table, i.e., build time = end time - start time (store it).
6. Repeat this for the hash table for double hashing.
7. Repeat all the above steps for different random sets of data of different input sizes (i.e., 0.1m, 0.2m, 0.3m, 0.4m, and 0.5m).
8. Store and display the build times as shown in the table of sample output.

EECS 560 Lab – Experimental Profiling on Hash Tables

Prof.: Dr. Shontz, GTAs: Chiranjeevi Pippalla, Prashanthi Mallojula

4-Data recording and analysis:

Data recording is done for three details for each hashing method and for each input set of data.

1. For build time
2. For successful find count
3. For unsuccessful find count.

Input data is generated and inserted into the hash tables one random number at a time.

1. For each input size, the build time is calculated and recorded.
2. With the other sequence of randomly generated data for the find operation, each value is checked to see if it exists in the hash table. The counts for successful find and unsuccessful find are recorded.
3. Store the counts for each hashing technique.
4. Repeat the above three steps for all the input sizes. Now we have recorded the build time, number of successful finds, and the number of unsuccessful finds for all the input sizes for each hashing method.
5. Display your output.

Expected Results:

The following outputs are just for illustrative purposes only.

Sample output (should be recorded in a table).

Performance (Quadratic Probing)

Input size	100,000	200,000	300,000	400,000	500,000
Build (s)	0.012568	0.0214294	0.0340122	0.0477508	0.0624752
Number of items successfully found	199	393	600	755	923
Number of items not found	9802	9608	9401	9245	9078

Performance (Double Hashing)

Input size	100,000	200,000	300,000	400,000	500,000
Build (s)	0.017538	0.0214490	0.0333129	0.0437500	0.0564789
Number of items successfully found	1000	1393	1800	1345	923

EECS 560 Lab – Experimental Profiling on Hash Tables

Prof.: Dr. Shontz, GTAs: Chiranjeevi Pippalla, Prashanthi Mallojula

Number of items not found	8802	7608	7001	7510	9078

Report: Submit a report by answering the questions shown below along with your Lab work. Your answers should not be more than 5 to 6 sentences for each question.

1. Observe your output for quadratic probing and double hashing and compare the performance of both techniques for the build and find operations.
2. Which hashing technique performs better when searching and why?
3. Justify the worst case complexity of each of your experimental profiling results and compare them to the worst case complexity of the theoretical results for the build times with each of the hashing techniques.

Grading rubric:

- Full grade: The program should execute without any issues with all the options executed and with no memory leaks.
- Points will be taken off for execution errors, such as memory leaks, segmentation/program abort issues and missing handling of invalid cases.
- Programs that are compiled but do not execute will earn in the range of 0 to 50% of the possible points. Your grade will be determined based on the program design and the options implemented in the code.

Submission instructions:

- All files, i.e., the source files and Makefile, should be zipped in a folder.
- Include a ReadMe.txt if your code requires any special instructions to run.
- The naming convention of the folder should be LastName_Lab5.zip (or .tar or .rar or .gz).
- Email it to your respective lab instructor: chiranjeevi.pippalla@ku.edu (Chiru) or prashanthi.mallojula@ku.edu (Prashanthi) with subject line EECS 560 Lab5.
- Your program should compile and run on the **Linux machines** in **Eaton 1005D using g++.**