Project 4 - Comparing Hashing Techniques

Due: Please check due date on BlackBoard

Objectives

The purpose of this project is to give you significant exposure to Hash Tables. You will be conducting experiments with probing hash tables using linear, quadratic and double hashing. You will be asked to analyze the data you collect and compare the data with the expected theoretical results.

Introduction

In this project, you will be inserting randomly generated integers into three hash tables. For one hash table, you will use linear probing for collision resolution. For another you will use quadratic probing. For the third hash table, you will use double hashing. You will be collecting data about clustering and probing in hash tables.  
  
Your program will attempt to insert N unique random integers into the hash tables, reporting the collected data on a specified interval (after every INTERVAL insertion attempts). For example, with N = 1000 and INTERVAL = 200, you will attempt to insert 1000 integers, and report data after attempting to insert 200, 400, 600, 800 and 1000 integers. See the sample output for recommended organization of the data. You may assume that N is a multiple of INTERVAL.   
  
Throughout this project description, "N" will refer to the number of attempted insertions into the hash table; "K" will refer to the key being inserted; "I" will refer to the probe number; and "M" will refer to the hash table size. These definitions are the same as those used in class.   
  
Your program will accept the following command line arguments (in this order)

1. the name of a file which contains unique random integers separated by whitespace.
2. N -- the total number of random integers to attempt to insert (e.g. 1000)
3. INTERVAL -- the interval (number of attempted insertions) for reporting data (e.g. 200)
4. M - the size of the hash table
5. R - the largest prime less than M

After attempting to insert INTERVAL integers, your program will output the following to the file reults.txt, the following data:

1. the total number of attempted insertions (successful and unsuccessful) so far
2. the value of lambda (successful insertions / table size)
3. the total number of probes so far (for both successful and unsuccessful insertions)
4. the average number of probes needed to insert (successfully or unsuccessfully) an integer into each hash table
5. the maximum number of probes needed to insert (successfully or unsuccessfully) an integer into each hash table
6. the number of successful insertions
7. the number of unsuccessful insertion attempts
8. the number of clusters in the hash table. where a cluster is defined as one or more consecutive occupied slots in the hash table. When counting clusters, do not "wrap around" from the end of the table back to the front.
9. the maximum cluster size in each hash table
10. the average cluster size in each hash table

Requirements, Notes, and Hints

* Code for the author's hash table class can be found at [Mark Weiss's web site](http://users.cis.fiu.edu/~weiss/dsaa_c++3/code/) or the given code (zipped). That code might require some minor edits to run on your computer or GL.
* The author has provided code for chaining and quadratic probing. You will need to add code for linear probing. You may not need to use Weiss's separate chaining code, but we include it as another example.
* The author's code may be set to automatically rehash the hash table when the table is 50% full. This code must be ***disabled*** so that we can study the effects that filling the hash table.
* The author's code also initializes the hash table to a size equal to the next prime after the integer passed to the constructor as M. You'll need to ***disable*** this code as well, so that it makes a table of size M, where M is passed as a parameter. FYI, the value of M we use will be a prime.
* An insertion is "unsuccessful" if h(k, i) == h(k, j) for some j not equal to i. You should check this condition after each probe so that we all count unsuccessful insertions in the same way.
* Your code must handle any exceptions thrown by the author's code or by your own classes.
* You'll need to create a driver file that reads command line arguments, and initiates the simulations.
* All "average" values should be printed with 2 decimal places of precision.
* Probing Functions

1. For all probing, use the function   
   **h( K ) = K mod M**
2. For linear probing, use the function   
   **h( K, I ) = ( h( K ) + I ) mod M**
3. For quadratic probing, use the function   
   **h( K, I ) = ( h( K ) + I2) mod M**
4. For double hashing, use the function   
   **h ( K, I ) = ( h( K ) + I \* h2( K ) ) mod M**   
   where **h2 ( K ) = R - ( K mod R )**   
   and where R is the largest prime that is smaller than the table size.

Sample Output

We have prepared some sample output for this project. Remember, you will not be printing to the screen, but to results.txt. The relevant files include:

* proj4sampleoutput.txt is what the output from your program might look like
* proj4sampletables.txt is a print of the actual hash tables for your debugging purposes

If your output differs significantly from this and you are certain that you are right, please see your instructor or a TA in order to discuss any discrepancies.

The following sample output is for formatting and organizational purposes only. No real data is provided. In this example, N = 1000 (the total number of insertions to attempt) and INTERVAL = 100 (the reporting frequency). Your output will consist of three tables like the one shown below. **There will be separate tables for linear probing, quadratic probing and double hashing.**

In the tables, the columns are defined as follows   
  
General:

* "N" -- the total number of attempted insertions.
* "lambda" -- the load factor -- the number of successful insertions / table size

For the "Inserts" columns:

* "success" -- the running total of the number of successful insertions
* "failed" -- the running total of the number of insertions that failed because no slot could be found.   
  Note that success + failed = N

For the "Probes" columns:

* "total" -- the running total number of probes for all insertions (successful and unsuccessful)
* "avg" -- the average number of probes for each insertion (successful and unsuccessful)
* "max" -- the maximum number of probes necessary for a single insertion (successful or unsuccessful)

For the "Cluster" columns:

* "number" -- the number of clusters in the table
* "avg" -- the average size of each cluster
* "max" -- the maximum cluster size in the table

Linear Probing Analysis (Table size = xxxxxx)

--- Inserts --- ------- Probes ------- ----- Clusters -----

N lambda success failed total avg max number avg max

200 0.xx xxxxx xxxxxx xxxxxxx xxx.xx xxxxx xxxxxx xxx.xx xxxxx

400 0.xx xxxxx xxxxxx xxxxxxx xxx.xx xxxxx xxxxxx xxx.xx xxxxx

600 0.xx xxxxx xxxxxx xxxxxxx xxx.xx xxxxx xxxxxx xxx.xx xxxxx

800 0.xx xxxxx xxxxxx xxxxxxx xxx.xx xxxxx xxxxxx xxx.xx xxxxx

1000 0.xx xxxxx xxxxxx xxxxxxx xxx.xx xxxxx xxxxxx xxx.xx xxxxx

## What to Submit

Read the [course project submission procedures](http://www.csee.umbc.edu/courses/undergraduate/341/fall13/projects/submission.shtml). *Submission closes by script immediately after midnight.* Submit well before the 11:59pm deadline, because 12:00:01 might already be late (the script takes only a few seconds to run).

You should copy over all of your C++ source code and have your .cpp files in their own directories which are in turn under the src directory. You must also supply an MAKE build file.

Make sure that your code is in the ~/cs341proj/proj4/src directory and not in any other subdirectory of ~/cs341proj/proj4/. In particular, the following Unix commands should work.

cd ~/cs341proj/proj4/src

make

make run **FILE**=data.txt **N**=10 **INTERVAL**=1 **M**=9 **R**=7

make clean