CS3100/5100: Data Structures and Algorithms

Programming Assignment #4

Due Dec. 9th, 2017, 11:30pm

1 Project Description

For this assignment, each student will implement a chaining Hash Table. The key used for the operations specified as follows will be short strings, for example, the American last names in the attached file. Assume a string consists of k characters. $s[i]$ represents the i-th character from left to right. We will try two hash code functions for strings: "djb2" and "sdbm" (`http://www.cse.yorku.ca/%7eoz/hash.html`) for chaining hash table. The actual hash function is $H(s) = djb2(s) mod M$ or $H(s) = sdbm(s) mod M$. M is the size of the hash table. M is better chosen to be a prime number between n and 2n, where n is the total number of strings. Here, we chose M to be 88799 or 88801.

In the beginning of your main() function, you should create a Hash Table. Then you should open the input file, read the file that contain the strings of last names, and insert them into the Hash Table using the insertion method for Hash Table. You should start with an empty Hash Table while inserting all the strings into the Hash Table. After all the strings are inserted into the Hash Table, print the number of occupied buckets in the Hash Table, and the load factor to the screen.

You then should implement a user interface (MENU) that supports the following operations:

- Insert new Entry: prompt user for a last name, insert it into the Hash Table. Your implementation should detect the insertion of a duplicated last name and reject the insertion. Display information telling whether or not the insertion is successful; if successful, display the bucket number that the last name is inserted.
- Delete an Entry: Ask the user for a last name and delete it from the Hash Table. Display information telling whether or not the deletion is successful. If successful, display the last name and the corresponding bucket number. Display information telling the delete is not successful, i.e., last name: not found.
- Search: Search for a last name given via the keyboard. If successful, display the last name and the bucket number that the last name is found. If not found, display information telling the search is not successful, i.e., last name: not found.
- Logfile: Write a formatted display of the hash table to the log file. The display should list each bucket of the Hash Table, indicating that the bucket is empty, or showing the key value.
- Quit.

The graduate students should implement two extra operations for the Hash Table:

- Delete a batch size of 20000, 40000, 60000, 80000 strings from the Hash Table respectively.

  The different batch size of the inputs should be generated randomly, i.e, you should use a random shuffle function to shuffle the list of input strings, then choose the first 20000, 40000,

60000, 80000 strings to delete one by one. You can use random_shuffle() function in C++ STL to randomly shuffle the list of strings. Please check the online reference about the random_shuffle() function in C++ STL. For this delete operation, you should start with a Hash Table having all strings inserted. Write a record into a file for each delete. This record contains information telling whether or not the deletion is successful. If successful, display the bucket that the last name is deleted.

- Search a batch size of 20000, 40000, 60000, 80000 strings from the Hash Table respectively. The different batch size of the inputs should be generated randomly, i.e, you should use a random shuffle function to shuffle the list of input strings, then choose the first 20000, 40000, 60000, 80000 strings to search one by one. Again, you can use random_shuffle() function in C++ STL to randomly shuffle the list of string. For this search operation, you should start with a Hash Table having all strings inserted. Write a record into a file for each search. The record for each search contains information telling whether or not the search is successful. If successful, display the last name and the bucket that the last name is found.

Running your program should produce a menu similar to the one shown in the example below. When loading a file with strings from the disk, all current entries in the Hash Table should be deleted, and the Hash table should be rebuilt.
MENU
(I)nsert new Entry
(D)elete Entry
(S)earch by last name
(L)ogfile
(Q)uit

You should implement both hash code functions. For the two different hash code functions, different log files will be generated. Try to compare the log files generated by using the two different hash code functions, including the length of the longest linked list of all the hash table bucket in the hash table, the number of empty hash table buckets. In other words, among the hash table based on these two hash code function, which hash table will have better performance for the given input. Try to write a simple report, and you should draw figures for comparison purpose in your report.

The graduate students should have two extra operations implemented. The graduate student should also measure the running time of deleting and searching for different batch sizes, and report the running time using a table or a figure. For each batch size, you should repeat running the program for each batch size for 10 times. After you get the running time for each run, calculate the average running time for each batch size for delete and search respectively (In other words, after you get the running time (c_i) for each run, calculate the mean μ, $\mu = 1/n \sum_{i=1}^{n} c_i$.) If you choose to use figures to report the running time, then x-axis should be the batch size, and y-axis should be the running time. You should provide two figures, one for delete, and one for search. You can use any tool to generate the figure, such as excel, matlab, and gnuplot, etc.

You can use the following code segment to measure the execution time of a piece of code. There are other functions that can be used to measure execution time, and you are free to use other functions.
include < *ctime* >
.....
clock_t start = clock();

the piece of code that you want to measure the time

clock_t end = clock();
double duration = $(end - start)$/CLOCKS_PER_SEC;

"duration" is the execution time in seconds spent on the piece of code. If the number is too small, use $duration = end - start$ instead. But you should use the same setting cross different experiments to make the running time comparable.

2 Requirements

1. In order to use the c++ compiler environment installed under the school's unix server, unixapps1.wright.edu, you need to connect to this unix server remotely using a secure shell client, putty. You can remotely connect to this unix server, unixapps1.wright.edu, on campus from a Wright State computer or use your own laptop connecting to the WSU wifi network named WSU-Secure. Note that you cannot remotely connect to this computer using a secure shell client using computers outside Wright State University without installing VPN or use the campus WSU_EZ_CONNECT wifi network.

2. You must submit an ELECTRONIC COPY of your source program through Pilot before the due date. If for some reason Pilot is unavailable, submit your source code to the instructor Meilin Liu.

3. Your main program should create a user interface similar to the example above. The file name for the main program should be lab4.cpp.

4. Submit all your source codes (LinkedNode.h, SList.h, SList.cpp, HashTable.h, HashTable.cpp, and lab4.cpp), makefile, possibly a README file, and any other required files. You are recommended to explain your programs clearly in the README file.

5. All the submitted project files should have: Course Number / Course Title, Your Name, Prof.s Name, Date, and the Project Name. If you did not include these required contents in your submitted files, then 5 points will be deducted. You also need to submit a makefile or a compiling command to compile your source codes. If not, another 5 points will be deducted.

6. The instructor will test your programs under WSU's UNIX environment, e.g.,unixapps1.wright.edu. It is YOUR responsibility to make your programs workable and runnable by others under school's UNIX environment.

7. The programming assignment is individual. You must do the project by yourself. If you allow others to copy your programs or answers, you will get the same punishment as those who copy yours.

3 Hash code functions

The two hash code functions are attached here too. Again, you are encouraged to check the original website (`http://www.cse.yorku.ca/%7eoz/hash.html`) for these two hash code functions.

djb2: This algorithm (k=33) was first reported by dan bernstein many years ago in comp.lang.c. Another version of this algorithm (now favored by bernstein) uses xor: $hash(i) = hash(i-1)*33^{str[i]}$; the magic of number 33 (why it works better than many other constants, prime or not) has never been adequately explained.

- Psuedo code:
  unsigned long
  hash(unsigned char *str)
  {
  unsigned long hash = 5381;
  int c;
  while (c = *str++)
  hash = ((hash << 5) + hash) + c; /* hash * 33 + c */
  return hash;
  }

sdbm: This algorithm was created for sdbm (a public-domain re-implementation of ndbm) database library. It was found to do well in scrambling bits, causing better distribution of the keys and fewer splits. It also happens to be a good general hashing function with good distribution. The actual function is $hash(i) = hash(i-1) * 65599 + str[i]$; what is included below is the faster version used in gawk. [there is even a faster, duff-device version] the magic constant 65599 was picked out of thin air while experimenting with different constants, and turns out to be a prime. this is one of the algorithms used in berkeley db (see sleepycat) and elsewhere.

- Pseudo code:
  static unsigned long
  sdbm(str)
  unsigned char *str;
  {
  unsigned long hash = 0;

  int c;
  while (c = *str++)
  hash = c + (hash << 6) + (hash << 16) - hash;
  return hash;
  }