Hazelyn Cates
11/5/22
EN.605.620.81.FA22

README

This program was written in Python 3.8 in the IDE PyCharm, version 2021.2. This program writes the output to a file and can also be run on the command line if the input file(s) are in the same file as the program.

This program hashes values in a given input text file to their proper location in a hash table containing 120 slots. This program implements four main hashing schemes: division modulo 120, division modulo 113, division modulo 41, and multiplication using value 120. Schemes 1, 2, and 4 hash values to a hash table of size 120 with a bucket size of 1 per slot and use linear probing, quadratic probing, and chaining to handle collisions. Scheme 2 by comparison hash values to a hash table of size of 120, but with a bucket size of 3 (meaning there are 40 accessible slots).

The input files for this program contain numbers ranging in value from 1 to 99999. In the text input file, they are listed in no particular order in a single column, as such:

12083

00193

57739

99828

…etc.

This program consists of 14 functions, the first two of which implement the division and multiplication hash functions (respectively), while functions 3 through 8 insert the keys in the input file into their proper slot in the table (where bucket size = 1), being identical in methodology between the division and multiplication schemes. Functions 9 and 10 implement a chaining-like procedure for hashing the values when bucket size = 3, functions 11 through 13 implement the three aforementioned collision handling techniques, and function 14 prints the hash table, load factor, collision statistics, and the number of values unable to be hashed in each hashing scheme.

division_hashing and **multiplication_hashing** calculate hash values using division and multiplication hash functions, respectively.

division_insert_linear1 and **multiplication_insert1** insert the values from the input file into their corresponding slots in the hash table as calculated by a call to **division_hashing** and **multiplication_hashing**, respectively, when bucket size is equal to 1. If a collision occurs during hashing, both of these functions call the **linear_probing** function to calculate new hash values to hash the remaining values. Lastly, a call to **print_hashTable** within these functions prints the resulting hash table, load factor, collision statistics, and the number of values that could not be hashed.

division_insert_quadratic1 and **multiplication_insert_quadratic1** insert the values from the input file into their corresponding slot in the hash table as calculated by a call to **division_hashing** and **multiplication_hashing**, respectively, when the bucket size is equal to 1. If a collision occurs during hashing, both of these functions call the **quadratic_probing** function to calculate new hash values to hash the remaining values. Lastly, a call to **print_hashTable** within these functions prints the resulting hash table, load factor, collision statistics, and the number of values that could not be hashed.

division_insert_chaining1 and **multiplication_insert_chaining1** insert the values from the input file into their corresponding slot in the hash table as calculated by a call to **division_hashing** and **multiplication_hashing**, respectively, when the bucket size is equal to 1. If a collision occurs, both of these functions call the **chaining** function to create a list at the corresponding slot in the hash table, where multiple values can be hashed to the same slot. Lastly, each of these functions call the **print_hashTable** function to print the resulting hash table, load factor, collision statistics, and the number of values that could not be hashed.

division_insert_linear3 and **division_insert_quadratic3** insert values from the input file into their corresponding slot in the hash table as calculated by a call to **division_hashing** when the bucket size of each slot in the hash table is 3 (i.e. three values max can hash to the same slot). If a hash value is calculated that exceeds the size of the table or a collision occurs, these functions call the **linear_probing** and **quadratic_probing** functions respectively to calculate new hash values until all the values in the input file have been hashed correctly. Lastly, a call to **print_hashTable** within these functions prints the resulting hash table, load factor, collision statistics, and the number of values that could not be hashed.

linear_probing and **quadratic_probing** calculate and return new hash values as called in their respective insert functions (see above).

The **chaining** function appends the number from in the input file into its correct slot in the hash table when called in their respective insert functions (see above).

print_hashTable prints out the resulting hash table from the insert functions (see above), collision statistics, the calculated load factor and the number of values from the input file that could not be hashed. All of the results from each hashing scheme are appended to the end of the same file (“output.txt”) and each run is separated by asterisks and the title “Next Run”. In order to reset output.txt, delete it from its location in your file directory and re-run the program.