Group31 Project 3 Report

Rylan Abraham, Yen Pham, Alexis Phu, Thong Nguyen

Project Description

In this project we created three different hash tables to act as a database for geography, age, and disability information. For the first part of this project we took in information and sent it to the hash tables. Next we made three different hash tables, each one using different hashing and collision strategies to store the data. Each hash table has functions that are used to manipulate the date, those functions being: INSERT(string ,string), UPDATE(string ,string), SELECT(string ,string), DELETE(string ,string), DISPLAY(), and WRITE().

All of the functions should be in all of the hash tables and the functions do the following: INSERT() inserts a tuple of data into a hash table. UPDATE() edits the data at the key given. SELECT() searches the hash table for a query. DELETE() deletes all rows that match the query. DISPLAY() outputs the tables, and WRITE() outputs the tables into a file.

Reading in input and coordinating output (Yen Pham)

For this portion of the project, we read in the main file from the user, and passed the files containing the information for the tables to the respective tables, which handled reading in those files themselves. The majority of this part was implemented in main to make for simpler coordination between each part. The file handling is done by the individual tables in order to streamline the population of the table, since each table will have to insert its data with its own function. An instance of each class (age, disability, and geography) are declared to store the table information and allow access to the functions within each class. After checking for the age, geography, and disability input files and sending them to the corresponding file handling function for its table, the program next reads the commands.

Each command read in is identified by using a regular expression comparison. The commands are compared to the patterns recognized as corresponding to the function calls. After the function the command asked for was determined, the program reads in which table it should execute for by comparing the next line to “age”, “disability” and “geography.” It will then call the corresponding function within the classes. For the display and write functions, the functions for all tables are called, so that every table’s information can be output. For each command, once the command type and the table needed are identified, the programs calls the corresponding function within the indicated table’s class. If there is an error in opening the file, this is reported to the user before quitting the program. If not, the program will output a message to the terminal when the commands are completed.

Geography table (Alexis Phu)

For this portion of the project, we created a geography class to hold all of the data from the files about geography, and a geographys class to hold the table itself. The data was stored in a pointer to objects of the geography class. We also created a class to implement a linked list, which contained pointers to the elements of the table and was used to walk through the array and manipulate the information. The geographys class also kept track of the table’s count, capacity, and scheme. All of the functions to insert, delete, select, update, and output the data for the geography table were defined within the geographys class as well. The hashing method we used for this part was perfect hashing, and the collision strategy was quadratic probing. The array size begins at a size of 50, and the capacity is doubled any time there is not enough room to store the data.

Each table was given the file with the tuples of information from the main file and was responsible for reading its own data. The way that the tuples were handled was through the insert function, which reads through the data and inputs the information onto a temporary object. The function then compares the new object to any objects that are already stored in the table to ensure there are no repeated replan ID’s before hashing the key and inserting the tuple into the table. If there is a repeat, an error message is printed. The update function reads through the given information in the main file, which is passed into the function as a parameter, and compares it to the information in the table, replacing it with the new data. The tuple is then rehashed to ensure it is placed in the correct place. If the program cannot complete the process, a message is printed to the terminal. Similarly, the select function reads in the information from the command and compares it to those in the table, reporting to the user whether or not a match was found, and listing the matches if they exist. Delete again reads the data and compares it to the table entries, reporting to the user if there is a match, and deleting that entry by using the tombstoning method. For the display function, the information in the table is output to the terminal, while for the write function, it is output to the out_geography.csv file.

Age table (Rylan Abraham)

For this portion of the project we created a hash table for the age data. For age we created a class called age to hold the age data, and a class called ages as a hash table which has an array of age pointers. For this table’s hashing function and collision strategy we did modulo hashing and linear probing. The hash table has several functions to manipulate the data, all of which are mentioned in the project description.

For INSERT we first found the key value for the tuple inputted, then we checked if something was already in that slot of the hash table if so it is reported to the user and the data is not overwritten if not the data is inserted into the hash table, and reports a successful insertion. For UPDATE we first found the key value for the tuple inputted, then we check if the slot is occupied if so replace the tuple with the inputted tuple, and report a success, if not report that there was no update. For SELECT we search the hash table for all rows that match the query, and it outputs all rows that match, if none it will report that nothing matched the query. DELETE does the same thing as SELECT except that instead of reporting the rows that match, it deletes them. For DISPLAY it outputs the current state of the hash table in a tabular format. Lastly for WRITE it transfers the data from the hash table onto a separate file.

Disability table (Dan Nguyen)

For this portion of the project used string multiplicative hashing and double probing to store and manipulate the disability table. Turning a given string (string letter) into an index to be used for double probing. As required, the code contains the following functions: UPDATE, SELECT, DELETE, DISPLAY, WRITE. The UPDATE function takes the inserted data, break it down, and replace/update data by using string hashing to get the index from the geography name (the following functions follow a similar process of proof checking the given string with the stored information). The SELECT function will break down the given information and check whether the information correctly matches any stored data, if found, it would display information of that data, if not, it would say none were found. For the DELETE function it will check and compare the string given with the information stored, if it matches then it will reset data at that index, if the information does not match, it will output a failure to delete message. The DISPLAY function will print and output all information stored. The WRITE function will transfer all data to a separate file.