CSCE 2100 Project 2

Jeremiah Spears

Project Description

This program performs the task of reading in input files containing pokemon, pokemon trainers, and pokemon gyms. These files are to be read into the program and stored in hash table structures. Each table must use its own hash algorithm and hash function. Once the data is read in and stored the program will then take commands given by the user in the input file and perform them on the given table. These commands are: SELECT, DISPLAY, INSERT, DELETE, UPDATE, and WRITE. Each command must be implemented using a function. If the program is unable to complete the given command, it must give a useful feedback message letting the user know what happened. It must also give a feedback message on command success.

Data Structures

My implementation of the project utilizes three different hash table functions and three different hash table algorithms. Each table is represented using a separate class. For the pokemon table, the actual data for each individual pokemon is stored in a single instance of the pokemon struct, and the hash table is filled with the structs. I decided to store the individual elements in structs because it made it easier to deal with storing multiple values if needed and structs are much more lightweight in their implementation compared to classes.

The second hash table stores the trainer data. It stores the individual trainer data in the trainer struct and stores the structs in a vector of trainers. The final table is implemented in the gymTable class. It stores data in a vector of gym structs. In each struct within the three table classes the individual data is stored in a data string. This string holds the table values as given by the input files. In the main function the commands that have been read in are stored in the command vector.

System Functionality

Once run, the program will first ask the user to input an input file. This file will contain the additional input files that hold the pokemon, trainers, and gyms tables. The file will also include any commands that are to be run to modify or view the data from the tables. To run the commands the command vector is iterated through and each command is run in the order that it was given in the input file. The commands are implemented separately in each of the table class files.

Each of the three input files is read in using a separate function. This makes it easier to set up individual variables and read in the data to each class. To decide on the number of buckets for each hash table I experimented with different values. I decided to multiply the number of items in each table by 1.75. I found that this gave the best balance for each table. It allowed the entries to be distributed evenly while also not having too many empty buckets in the tables. I also counted the number of INSERT commands before reading in the file to make sure that no matter how many insert commands are run, the table will be able to have enough buckets from the point it is first constructed.

For the pokemon table I used a chaining hash table. To implement the chaining, I used a 2d vector of struct pokemon. If a hash collision occurs, a new node is created and the pokemon struct is pushed back onto the inner vector. This table uses a middle-square hash function to get the hash key using ID as the value. This function works by first squaring the ID and converting that to a string. It then extracts the middle digits of the string and converts back to an int and performs a mod calculation on the result giving the table key. In the trainer table I utilized a string hash function and the linear probing strategy. The hash function is implemented by adding up the ASCI values in the trainer firstname and lastname and then performing the modulo operation on them. In the final table the gyms, I decided to use the double hashing technique along with a simple modulus operation to get the hash key. This made it more accessible to perform two hashing functions. To get the second hashing function I found the nearest prime number to the number of buckets in the table and used that to generate a second hash key.