Advanced Data Structures Homework 2

Name: Ali Çağatay Yüksel
ID: 040050220
Instructor: Sanem Kabadayı

Question 1)

Mapping function used in this project takes every letter in Turkish alphabet as digits of a number in base 29 and then converts them to decimal numbers. For example '**a**' means decimal 1 and '**z**' means decimal 29 according to this approach. A string with length two, let's say "**az**", will map into

$$1 \times 29^1 + 29 \times 29^0 = 58$$

This function can map any sequence of characters into integers, limited by the max value of data type unsigned long long int.
This function is a member method of HashTable class included in hashSearch.cpp file.

Question 2)

Structure used to solve the problem is a *linked list of linked lists* which is every node of the outer list includes the file name and an inner list of the words that occur in that file. One can lookup the words by scanning the inner nodes and if the queried word is there, file whose name is stored in the outer node is returned. Both of the lists are implemented as C++ classes with *add, print* and *find* methods provided publicly. List operations goes like this:

1. Open file with the given file name and create a node for it.

2. For every word in that file look for the words list first, if it does not exists, create a new node.
3. If a string is searched, look up every word in every file node and return the ones which exactly match the given string and the ones which include it as a prefix.

It took **137410 ms** to search all the words in the *words.txt* fle using link list search.

Question 3)

HashTable class included in *hashSearch.cpp* is a hash table implementation using double hashing to avoid collisions. This class' most important member is *table[]* array which is the hash table itself. *table* is an array of *struct BucketNode* with size **10000**. Because there are about 5000 entries in the *words.txt* file provided, and it is a good idea to keep the load factor of a hash table below **0.7**, such a size seemed reasonable.

Two functions are provided to do hashing. Both function *hash1* and *hash2* take the key value evaluated using the *str2int()* function described in Question 1, and return its remainder over 10000 for hash1 and 7 for hash2. *hash* function does the double hashing using these two functions. When adding an entry to the table, hash(key, 0) is computed first, if the corresponding slot is full which means there is a collision, hash(key, 1) is tried and so forth. The number of collisions occurred filling the hash table with the words given in words.txt file is **2469**, which is expectable according to the **Birthday Paradox**. It took only **1500 ms** to search all the entries in the file words.txt which means hash table search is a lot faster than linked list search.