**Algorithm Analysis and Data Structures**

**CS 5343.502(Spring 2020)**

**Assignment 6**

**Divya Birla**

**2021514344**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**QUESTION:**

Implement Hash table.

Pick 20 random words.  Each word must be of different lengths, maximum length 8 and minimum length 3.

The words will be of letters a-zA-Z and the space character.

Insert them into a hash table.

You can use a library for only the hash function.

The collision resolution scheme should be open addressing - quadratic.

Initially the table size is 31.  The program should increase the table size and rehash at load factor of .5

So after you inserted about 15 or 16 words, your program automatically doubles the table size and re-inserts (automatically) the old words and then continue the insert of additional words.

You do not have to insert the words manually (one by one) but you can add the words in a file and let your program read from the file

At the end print the total number of collisions you get.

Submit your code and print screen of your execution

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**SOURCE CODE:**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Course: CS 5343.502 – Spring 2020

\* Assignment <6>

\* Name: Divya Birla

\* E-mail: dxb190021@utdallas.edu

\* Submitted: <04/28/20>

This program performs Hashing by reading words in words.txt, which is a file having 20 animal names.

The program reads these names into a string vector and attempts to hash it into a hashtable of size 31.

For any collisons that occur it uses quadratic probing for collison resolution where if slot hash(x) % TableSize is full, then we try (hash(x) + i^2) % TableSize

Here, i is the number of collisions for the same input.

Once loadfactor reaches>=0.5, the hash table is resized to the next prime number 67.

The program also shows the total number of collisons that occur to execute the entire hashing.

Note: loadfactor = (number of elements hashed / hashtable capacity).

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

#include <functional>

using namespace std;

//function to insert elements into hashtable

void insert(int hashkey, vector<string> &wordlist, float &loadfactor, int &collisions, int tablesize, vector<string> &hashtable, int count) {

//counter to keep track of collisions that occur for each word

int i=0;

//To insert into hastable given hashtable is empty at the key generated

if (hashtable[hashkey] == "") {

hashtable[hashkey] = wordlist[count];

cout << "\n Word " << wordlist[count] <<" inserted at position "<< hashkey<< " of hashtable"<< endl;

}

else

{

//To find an empty space in the hashtable for inserting

while (!(hashtable[hashkey] == "")) {

cout << "\n Collision occured while inserting word "<<wordlist[count]<< " for frequency= "<<i+1<<endl;

//Flag to keep track of collisions

collisions++;

//Formula for quadratic probing

hashkey = (hashkey + (i \* i)) % tablesize;

i++;

}

//Insertion at empty space found using quadratic probing

hashtable[hashkey] = wordlist[count];

cout << "\n Word " << wordlist[count] << " now inserted at position " << hashkey << " of hashtable" << endl;

}

}

//Function to create hashtable and generate hash key

bool makehashtable(vector<string> wordlist, vector<string>hashtable, int &tablesize, int&collisions, int &count, bool tablehalffull) {

//Flag to keep track if table is half full

tablehalffull = false;

//To convert string to int values

hash<string> hasher;

//To store hash key value(it is made unsigned to let it accomodate case-sensitive situation of string)

unsigned int hashkey;

float loadfactor = 0.0;

//Loop to generate hash key for each word and then insert

for (int i = 0; i < count; i++) {

//Hashkey generated

hashkey = hasher(wordlist[i]) % tablesize;

//Insert at hashkey

insert(hashkey, wordlist, loadfactor, collisions, tablesize, hashtable, i);

loadfactor = ((float)(i + 1) / (float)(tablesize));

if (loadfactor >= 0.5) {

tablehalffull = true;

cout << "\n Resizing of hash table required ";

break;

}

}

cout << "\n Loadfactor now is " << loadfactor;

return tablehalffull;

}

int main() {

cout << "\n--PROGRAM TO PERFORM QUADRATIC PROBING AND REHASHING--\n";

bool tablehalffull = false;

int collisions = 0;

int count = 0;

int tablesize = 31;

//To store words in an array

vector<string> wordlist;

//To store values in hashtable

vector<string> hashtable;

string word;

ifstream infile;

//To read file of words

infile.open("words.txt");

if (infile.is\_open()) {

//To read words from file

while (getline(infile, word)) {

wordlist.push\_back(word);

count++;

}

infile.close();

}

else {

cout << "\n\nCould not open file";

cout << "\n\n";

system("pause");

return 0;

}

hashtable.resize(31);

//To make hashtable of size 31 considering loadfactor

tablehalffull = makehashtable(wordlist, hashtable, tablesize, collisions, count, tablehalffull);

cout << "\n\n Total collisions occured in first loop of hashing =" << collisions << "\n";

//Loadfactor reached threshold value

if (tablehalffull == true) {

//To store reshashed elements

hashtable.resize(67,"");

tablesize = 67;

cout << "\n------------HASH TABLE RESIZED-----------\n";

tablehalffull = makehashtable(wordlist, hashtable, tablesize, collisions, count, tablehalffull);

}

cout << "\n\n All total collisions that occured during hashing =" << collisions << "\n";

cout<< "\n\n";

system("pause");

return 0;

}

**OUTPUT:**





