CS 323\_33 Programming Language: C++

Project #3 HashTable

Andres Quintero

Due Date:

Soft copy: 2/18/2020

Hard copy: 2/19/2020

Submission:

Soft copy: 2/18/2020

Hard copy: 2/18/2020

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Main():

Step 1: inFile 🡨open input file using argv[1]

outFile1, outFile2 🡨 open outfiles using argv[2] and argv[3]

Step 2: createHashAry (hashTableAry, bucketSize) // on your own

Step 3: storeDataIntoHashTable (inFile, outFile2)

Step 4: printHashTable (outFile1)

Step 5: Close all files

**Source code: main.cpp**

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

class listNode {

public:

string data;

int count = 0;

listNode\* next = NULL;

listNode(string parameterData,int parameterCount, listNode\* parameterNext){

data = parameterData;

next = parameterNext;

count = parameterCount;

};

};

class hashTable{

public:

int bucketSize; //hardcode to 29?

listNode\*\* hashTableAry;

};

// Prototypes

void createHashAry(hashTable\* hashTable, int bucketSize);

void storeDataIntoHashTable(fstream& inFile, fstream& outFile, hashTable\* hashTable, int bucketSize);

int Doit(string data, int bucketSize);

void listInsert(listNode\* listHead, listNode\* newNode);

listNode\* findSpot(listNode\* listHead, listNode\* newNode);

void printList(int index, fstream& outFile, hashTable\* hashTable);

void printNode(listNode\* node, fstream& outFile);

void printHashTable(int bucketSize, fstream& outFile, hashTable\* hashTable);

int main(int argc, char\* argv[]) {

int bucketSize = 29; // hardcode to 29 as assignment says

// int bucketSize = stoi(argv[4]); // pass as final cli argument

// Opening files

fstream inFile(argv[1]);

fstream outFile1(argv[2], fstream::out);

fstream outFile2(argv[3], fstream::out);

hashTable\* table = new hashTable();

createHashAry(table, bucketSize);

storeDataIntoHashTable(inFile, outFile2, table, bucketSize);

printHashTable(bucketSize, outFile1, table);

// Closing Files

inFile.close();

outFile1.close();

outFile2.close();

}

// Functions

void printHashTable(int bucketSize, fstream& outFile, hashTable\* hashTable) {

int index = 0;

while(index < bucketSize) {

printList(index, outFile, hashTable);

index++;

}

}

void printNode(listNode\* node, fstream& outFile) {

if(node->next == NULL){

outFile << "(" << node->data << ", " << node->count << ", " << "NULL" << ") -> NULL";// because NULL->data will cause a segment falt 11

} else {

outFile << "(" << node->data << ", " << node->count << ", " << node->next->data << ") ->";

}

}

void printList(int index, fstream& outFile, hashTable\* hashTable) {

outFile << "hashTable[" << index << "]-> ";

listNode\* printSpot = hashTable->hashTableAry[index];

while(printSpot != NULL){

printNode(printSpot, outFile);

printSpot = printSpot->next;

}

outFile << endl;

}

listNode\* findSpot(listNode\* listHead, listNode\* newNode){

listNode\* Spot = listHead;

while(Spot->next != NULL && Spot->next->data < newNode->data){

Spot = Spot->next;

}

if(Spot->next != NULL && Spot->next->data == newNode->data){

Spot->next->count++;

Spot = NULL;

}

return Spot;

}

void listInsert(listNode\* listHead, listNode\* newNode){

listNode\* Spot = findSpot(listHead, newNode);

if(Spot != NULL){

newNode->next = Spot->next;

Spot->next = newNode;

}

}

int Doit(string data, int bucketSize) {

unsigned int val = 1;

int stringLength = data.length();

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

val = val \* 32 + int(data[i]);

}

return val % bucketSize;

}

void storeDataIntoHashTable(fstream& inFile, fstream& outFile, hashTable\* hashTable, int bucketSize) {

string data;

while(inFile >> data){

listNode\* newNode = new listNode(data, 1, NULL);

int index = Doit(data, bucketSize);

listNode\* listHead = hashTable->hashTableAry[index];

listInsert(listHead, newNode);

printList(index, outFile, hashTable);

}

}

void createHashAry(hashTable\* hashTable, int bucketSize) {

hashTable->bucketSize = bucketSize;

hashTable->hashTableAry = new listNode\*[bucketSize];

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

hashTable->hashTableAry[i] = new listNode("dummy", 0, NULL);

}

}

**Output: Hashtable**

**A screenshot of a social media post

Description automatically generated**