|  |  |
| --- | --- |
| **DATA STRUCTURE** | **PURPOSE** |
| 2D Array | Storing all tuples in a segmented manner. |
| Hash Table | Storing all arrays hashed and probed quadratically in linked nodes. |

**Data Structures and Algorithms Semester Project**

Group Members: Huzaifa Liaqat 390696, Ali Awais Safdar 369475



**Source code:**

**Hash.h:**

#pragma once

#include "TemplateLinklist.h"

class HashTableLinkedList {

public:

const int size = 8000;

int use = 0;

Node\_Hash\* table = new Node\_Hash[size]; // table points towards a node which points towards a data entry in the hash table

HashTableLinkedList(string\*\* array);

void insertIntoTable(int index, string showid, string\*\* content);

int getKey(string showID);

void search(string showid);

};

HashTableLinkedList::HashTableLinkedList(string\*\* array) { //constructs a null hash table of 8000 indexes and inserts data acoording to the dataset

for (int i = 0; i < 8000; i++) {

table[i] = nullptr;

}

for (int i = 0; i < 5650; i++) {

insertIntoTable(i, array[i][0], array);

}

}

void HashTableLinkedList::insertIntoTable(int index, string showid, string\*\* content) { //takes showid from the 2d array and generates a key for a tuple

int key = getKey(showid);

if (table[key] != nullptr) {

int i = 1;

while (table[key] != nullptr) {

key = (getKey(showid) + i \* i) % size;

i++;

if (use == size) {

break;

}

}

}

Node\_Hash\* nn = new Node\_Hash(); // stores the data associated with a show id in a linked list

nn->showid = showid;

nn->l1 = new Template\_LinkList(index, content);

table[key] = nn;

use++;

}

int HashTableLinkedList::getKey(string showid) {

// Add up the ASCII values of all the characters in the show ID

int sum = 0;

for (char c : showid) {

sum += (int)c;

}

// Return the sum modulo the table size to get the key

return sum % size;

}

void HashTableLinkedList::search(string showid) {

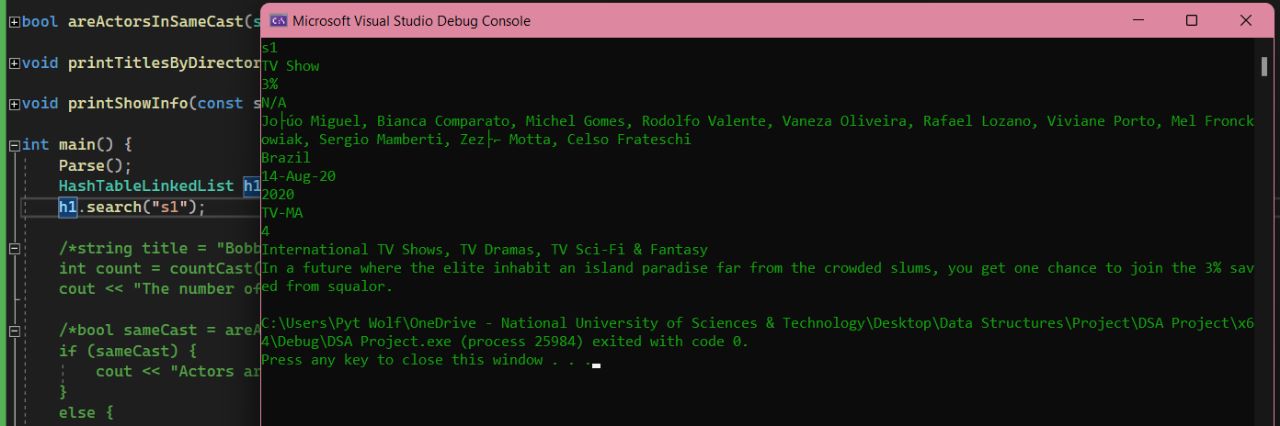
int idx = 0;

idx = getKey(showid);

cout << table[idx]->showid << endl; // displays the showid at a particular key, then traverses the list to which the node is pointing and prints its data.

table[idx]->l1->print\_data();

}



**Template\_Linklist.h:**

#pragma once

//#include <iostream>

//#include <string>

//#include "Parser.h"

#include <iostream>

using namespace std;

class Template\_Node {

public:

string data;

Template\_Node\* next;

Template\_Node\* prev;

};

class Template\_LinkList {

public:

Template\_Node\* loc;

Template\_Node\* ploc;

Template\_Node\* first;

Template\_Node\* last;

Template\_LinkList(int index, string\*\* content) { //function takes in a 2d array and stores the tuple at a given index into a linked list.

for (int i = 1; i < 12; i++) {

insert\_data(content[index][i]);

}

}

~Template\_LinkList() {

}

bool is\_empty() {

if (first == nullptr and last == nullptr) {

return true;

}

else {

return false;

}

}

void insert\_data(string a) {

Template\_Node\* nn = new Template\_Node();

nn->data = a;

if (is\_empty()) {

nn->next = nullptr;

nn->prev = nullptr;

first = nn;

last = first;

}

else {

last->next = nn;

nn->prev = last;

last = nn;

loc = last;

}

}

string get\_data(int index) {

Template\_Node\* current = first;

// Iterate to the node at the given index

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

current = current->next;

if (current == nullptr) {

// Return an empty string if the index is out of range

return "";

}

}

// Return the data at the node

return current->data;

}

void print\_data() {

loc = first;

while (!is\_empty() and loc != nullptr) {

cout << loc->data << endl;

loc = loc->next;

}

if (is\_empty()) {

cout << "err";

}

}

};

class Node\_Hash {

public:

string showid;

Template\_LinkList\* l1;

};

**Parser1.h:**

#ifndef PARSER\_H

#define PARSER\_H

#include <iostream>

#include <string>

#include <vector>

#include <fstream>

using namespace std;

vector<vector<string>> \_content;

string\*\* content = new string \* [5650];

enum class CSVState {

UnquotedField,

QuotedField,

QuotedQuote

};

vector<string> readCSVRow(const string& row) {

CSVState state = CSVState::UnquotedField;

vector<string> fields{ "" };

size\_t i = 0;

for (char c : row) {

switch (state) {

case CSVState::UnquotedField:

switch (c) {

case ',':

fields.push\_back(""); i++;

break;

case '"': state = CSVState::QuotedField;

break;

default: fields[i].push\_back(c);

break;

}

break;

case CSVState::QuotedField:

switch (c) {

case '"': state = CSVState::QuotedQuote;

break;

default: fields[i].push\_back(c);

break;

}

break;

case CSVState::QuotedQuote:

switch (c) {

case ',':

fields.push\_back(""); i++;

state = CSVState::UnquotedField;

break;

case '"':

fields[i].push\_back('"');

state = CSVState::QuotedField;

break;

default:

state = CSVState::UnquotedField;

break;

}

break;

}

}

return fields;

}

vector<vector<string>> readCSV(istream& in) {

vector<vector<string>> table;

string row;

int counter = 0;

/\*while (!in.eof() && counter <= 5650) {

getline(in, row);

if (in.bad() || in.fail()) {

break;

}

auto fields = readCSVRow(row);

table.push\_back(fields);

counter++;

}\*/

while (getline(in, row)) {

if (in.bad() || in.fail()) {

break;

}

auto fields = readCSVRow(row);

if (!fields.empty()) {

table.push\_back(fields);

counter++;

}

}

return table;

}

void Parse() {

ifstream infile("Netflix.csv");

\_content = readCSV(infile);

for (int i = 0; i < 5650; i++) {

content[i] = new string[12];

for (int j = 0; j < 12; j++) {

content[i][j] = \_content[i][j];

}

}

\_content.clear();

}

#endif

**Source.cpp:**

#pragma once

#include <iostream>

#include <string>

#include <cstring>

#include "TemplateLinklist.h"

#include "Parser.h"

using namespace std;

bool areActorsInSameCast(string actor1, string actor2, HashTableLinkedList& table) {

for (int i = 0; i < 8000; i++) {

if (table.table[i] != nullptr) {

if (table.table[i]->l1->get\_data(3).find(actor1) != string::npos && table.table[i]->l1->get\_data(3).find(actor2) != string::npos) {

// Both actors were found in the cast of the same show

return true;

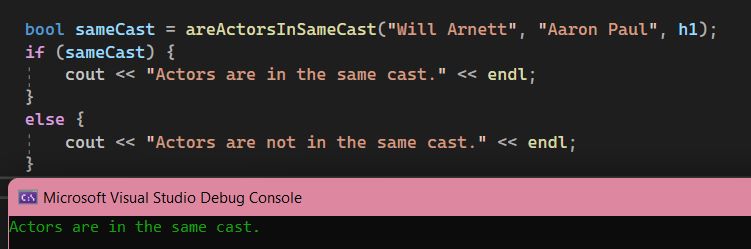
}

}

}

// Actors were not found in the same cast

return false;

}

int countCast(string title, HashTableLinkedList ht) {

int key = ht.getKey(title);

Template\_LinkList\* l1 = ht.table[key]->l1;

string cast = l1->get\_data(3);

int count = 0;

bool in\_name = false;

for (int i = 0; i < cast.size(); i++) {

if (cast[i] == ',' || i == cast.size() - 1) {

count++;

in\_name = false;

}

else if (!in\_name && cast[i] != ' ') {

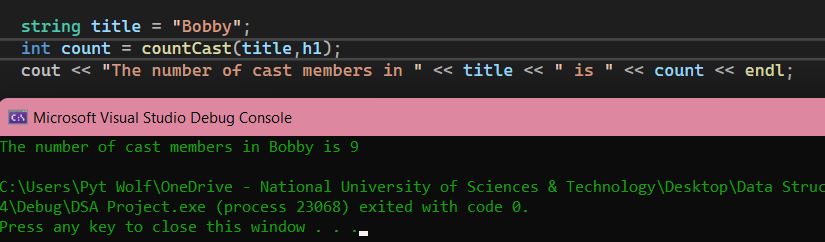
in\_name = true;

}

}

return count;

}



void printTitlesByDirector(const std::string& director, HashTableLinkedList& table) {

// Check if the hash table is empty

if (table.use == 0) {

cout << "The hash table is empty." << endl;

return;

}

bool found = false;

// Iterate through all the entries in the hash table

for (int i = 0; i < table.size; i++) {

if (table.table[i] != nullptr) {

// Check if the director name in the current entry matches the given director name

if (table.table[i]->l1->get\_data(2) == director) {

// Print the title of the show

cout << table.table[i]->l1->get\_data(1) << endl;

found = true;

}

}

}

if (!found) {

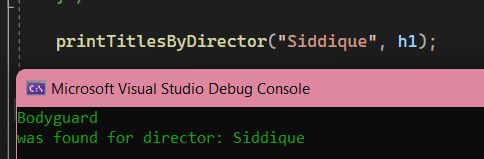
cout << "No shows found for director: " << director << endl;

}

else {

cout << "was found for director: " << director << endl;

}

}

void printShowInfo(const std::string& title, HashTableLinkedList& table) {

// Check if the hash table is empty

if (table.use == 0) {

cout << "The hash table is empty." << endl;

return;

}

bool found = false;

// Iterate through all the entries in the hash table

for (int i = 0; i < table.size; i++) {

if (table.table[i] != nullptr) {

// Check if the title in the current entry matches the given title

if (table.table[i]->l1->get\_data(1) == title) {

// Print the show information

std::cout << "Title: " << table.table[i]->l1->get\_data(1) <<std::endl;

std::cout << "Director: " << table.table[i]->l1->get\_data(2) << std::endl;

std::cout << "Country: " << table.table[i]->l1->get\_data(4) << std::endl;

std::cout << "Date Added: " << table.table[i]->l1->get\_data(5) << std::endl;

std::cout << "Release Year: " << table.table[i]->l1->get\_data(6) << std::endl;

std::cout << "Rating: " << table.table[i]->l1->get\_data(7) << std::endl;

std::cout << "Duration: " << table.table[i]->l1->get\_data(8) << std::endl;

}

}

}

}