Jacob Jamison

CS300

David Ostrowski

13 February 2022

**Project One Pseudocode**

**Vector**

#include <vector>

struct Course {

string courseNumber;

string courseName;

vector<string> prerequisites;

}

vector<Course> LoadDataStructure() {

data.open(“course.txt”)

While (get file data) {

Vector<Course> courses;

While{

Getline()

If(line = -1)

Break;

Course;

Vector<string>token

Course.courseNumber;

Course.courseName;

Course.prerequisites.pushback();

Course.pushback();

Close()

Return courses;

}

Void printCourse() {

string courseNumber= course.courseNumber;

string name=courseName;

vector<string> prerequisites=course.prerequisites;

Print “Course Number” endl;

Print “Course Name” endl;

Print “Prerequisites” endl;

For(int I = 0< prerequisites);

Print endl;

}

Void searchCourse() {

String courseNumber;

Int f=0;

Print “Enter course number” endl;

If courseNumber = courseNumber

Print Course;

If f =0;

Print “No course found” endl;

}

void LoadCourses(string fileName, vector<Course> &courses) {

Initialize fstream fileStream

Initialize string line

Initialize stringstream lineStream

Initialize string token

Open fileName with fileStream

Initialize int count

Fill lineStream with current line

Set count to 1

Create Course acourse for each line in file

Get token from lineStream up to ‘,’ until none left

if (count == 1) {

set acourse’s courseNumber to token

increment count

}

else if (count == 2) {

set acourse’s courseName to token

increment count

}

else {

if (token exists in courses as a course) add token to acourse’s

PreReqs

else output file format error

increment count

}

if (count < 2) {

output "Error in file format, each course must have course # and name."

}

push acourse to back of courses

Clear lineStream for next line

}

**Hash Table**

#include <algorithm>

#include <climits>

#include <string> // atoi

struct Course {

String courseNumber;

String courseName;

Vector<string> prerequisites;

}

const unsigned int DEFAULT\_SIZE = 8;

class Hashtable {

struct Node {

Course\* course;

Node\* next;

unsigned int key;

Node() {

key = UINT\_MAX;

next = nullptr;

}

Node(Course\* acourse): Node() {

course = acourse;

}

Node(Course\* acourse, unsigned int akey) : Node(acourse) {

key = akey;

}

}

unsigned int size = DEFAULT\_SIZE;

vector<Node> table;

}

return atoi(courseNum.substr(4).c\_str()) % size;

}

void Hashtable::Add(Course\* acourse) {

if (node == nullptr) {

Create new node with key

Insert contents newCourse into table position

}

else if (node’s key == UINT\_MAX) {

Update node key

Update node course

Update node next

}

else {

while node next != nullptr

Create new node with key

Set node next newCourse

}

void Hashtable::Print() {

For

each node in table

if (node key != UINT\_MAX) {

output node key

print node course

}

Create Node\* listNode set to next

while (listNode != nullptr) {

output listNode key

print listNode course

set listNode = next

}

Course\* Hashtable::Search(string courseNum) {

Create empty object

For

each node in table

if (node courseNumber = courseNum)

return node course;

Create Node\* listNode set node next

while (listNode != nullptr) {

if (listNode courseNumber = courseNum)

Return listNode;

listNode = listNode next

}

return empty object;

void LoadCourses(string fileName, Hashtable &Htable) {

Initialize fstream fileStream

Initialize string line

Initialize stringstream lineStream

Initialize string token

Open fileName with fileStream

Initialize int count

Fill lineStream with current line

Create new Course\* acourse

if (count = 1) {

set acourse courseNumber = token

increment count

}

else if (count = 2) {

set acourse courseName t= token

increment count

}

else {

if (token exists in Hashtable ) add token to acourse

Prerequisites;

else

Print “Error”

}

if (count < 2) {

Print "Error"

}

Add acourse to Hashtable

}**Binary Search Tree**

#include <iostream>

#include <vector>

#include <time.h>

#include <algorithm>

#include <climits>

#include <string> // atoi

struct Course {

String courseNumber;

String courseName;

vector<String> prerequisites;

}

Struct Node {

Course\* course;

Node\* left;

Node\* right;

If entry found

Return node->course;

If no entry found

Return course;

}

Class BinarySearchTree {

Private:

Node\* root;

Void addNode(Node\* node, Course course);

Void inOrder(Node\* node);

Node\* removeNode(Node\* node, Course course);

Public:

BinarySearchTree();

Void InOrder();

Void Insert(Course course);

Void Remove(Course course);

Course Search(string courseNumber);

}

BinarySearchTree::BinarySearchTree {

Root = nullptr;

}

void BinarySearchTree::InOrder() {

if (this->root = nullptr) {

Print

Return;

}

inOrder(this->root);

Return;

}

Void BinarySeachTree::Insert(Course course) {

if(root == nullptr)

Root = newNode(course);

Else

this->addNode(root, course);

}

Void BinarySearchTree::addNode(Node\* node, Course course) {

if(node->bidId > 0) {

if(node->left = nullptr)

node->left = new Node(course);

Else

this->addNode(node->left, course);

}

Else {

if(node->right = nullptr)

node->right = new Node(course);

Else

this->addNode(node->right, course);

}

}

void BinarySearchTree::Remove(string bidId) {

this->removeNode(root, bidId);

return;

}

Course BinarySearchTree::Search(string courseNumber) {

Node\* current = root;

while(current != nullptr) {

If course number matches;

Return current->course;

If current node compare to course number is < 0;

Current = current->left;

Else {

Current = current->right;

}

void LoadCourses(string fileName, BST &bst) {

Initialize fstream fileStream

Initialize string line

Initialize stringstream lineStream

Initialize string token

Open fileName with fileStream

Initialize int count

Fill lineStream

Set count to 1

Create Course acourse for each line in file

Get token from lineStream up to ‘,’ until none left

if (count == 1) {

set acourse’s courseNumber to token

increment count

}

else if (count == 2) {

set acourse’s courseName to token

increment count

}

else {

if (token exists in bst as a course) add token to acourse’s PreReqs

else output file format error

increment count

}

if (count < 2) {

output "Error in file format, each course must have course # and name."

}

Add acourse into bst

}

Course course;

Return course;

}

Int totalPrerequisites= 0;

int numPrerequisiteCourses(Tree<Course> courses, String courseNumber); {

Course course = courses.search(courseNumber);

while(course->prerequisites != 0);

For prerequisites;

courses.search(course->prerequisites->courseNumber);

totalprerequisites++

}

Void printCourseInformation(Tree<course> courses, String courseNumber) {

Course course = courses.search(courseNumber);

Print ;

while(course->Prerequisites != 0);

For prerequisites;

courses.search(course->prerequisites->courseNumber);

Print ;

}

Course parseLine(vector<string> &line) {

If lineSize = 2 {

Course newCourse;

course.courseName = line[0];

course.courseNumber = line[1];

Return newCourse;

}

Else {

Vector<string> tempPrerequisites;

for(int i = 2; i < line.size(); i++) {

tempPrerequisites.push\_back(line[i]);

}

Course newCourse;

course.courseName == line[0];

course.courseNumber == line[1];

Return newCourse;

}

Return 0;

}

**Menu**

Initialize string courseName

Initialize courseNumber

Initialize int choice to 0

Initialize int choice2 to 0

while (choice != 9) {

print "Menu:" endl;

print " 1. Load Schedule" endl;

print " 2. Display Course List" endl;

print " 3. Display Prerequisites" endl;

print " 9. Exit" endl;

print "Enter choice: ";

cin (choice);

switch (choice) {

case 1:

LoadCourses(courseName)

Break;

case 2:

while (choice2 == 0) {

print "1 ). Display Course List" endl;

print "2 ). Display Prerequisites" endl;

print "Enter choice: "

cin (choice);

switch (choice2) {

case 1:

print “ CSCI100, Introduction to Computer Science” endl;

print “ CSCI101, Introduction to Programming in C++” endl;

print “ CSCI200, Data Structures” endl;

print “ CSCI300, Introduction to Algorithms” endl;

print “ CSCI301, Advance Programming in C++” endl;

print “ CSCI350, Operating Systems” endl;

print “ CSCI400, Large Software Development” endl;

print “ MATH201, Discrete Mathematics”;

break;

case 2:

print "Enter course number: "

set prerequisites to.Search(courseName)

if (courseName is empty) print "That course does not exist” endl;

else print prerequisites;

break;

}

}

print "goodbye" endl;

return 0;

}

**Evaluation**

**Vector**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **Times Executed** | **Cost** |
| Initialize fstream fileStream | 1 | 1 | 1 |
| Initialize string line | 1 | 1 | 1 |
| Initialize stringstream lineStream | 1 | 1 | 1 |
| Initialize string token | 1 | 1 | 1 |
| Open fileName with fileStream | 1 | 1 | 1 |
| Initialize int count | 1 | 1 | 1 |
| Fill lineStream with current line | 1 | N | N |
| Create Course acourse for each line in file | 1 | N | N |
| Get token from lineStream up to ‘,’ until none left | 1 | N | N |
| if (count == 1) | 1 | 2n | 2n |
| set acourse’s courseNumber to token | 1 | N | N |
| increment count | 1 | N | N |
| else if (count == 2) | 1 | N | N |
| set acourse’s courseName to token | 1 | N | N |
| Increment count | 1 | N | N |
| else |  | N | N |
| if (token exists in course) | 1 | N | N |
| add token to acourse’s PreReqs | 1 | N | N |
| Else output file format error | 1 | 1 | 1 |
| increment count | 1 | N | N |
| if (count < 2) | 1 | 1 | 1 |
| output "Error in file format” | 1 | 1 | 1 |
| Clear lineStream for next line | 1 | N | N |
| **Total Cost** |  |  | **15n+9** |
| **Runtime** |  |  | **O(n)** |

**Hash Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **Times Executed** | **Cost** |
| Initialize fstream fileStream | 1 | 1 | 1 |
| Initialize string line | 1 | 1 | 1 |
| Initialize stringstream lineStream | 1 | 1 | 1 |
| Initialize string token | 1 | 1 | 1 |
| Open fileName with fileStream | 1 | 1 | 1 |
| Initialize int count | 1 | 1 | 1 |
| Fill lineStream with current line | 1 | 1 | 1 |
| Create new Course\* acourse | 1 | N | N |
| if (count = 1) { | 1 | N | N |
| set acourse courseNumber = token | 1 | 2n | 2n |
| increment count | 1 | N | N |
| else if (count = 2) { | 1 | N | N |
| set acourse courseName t= token | 1 | 2n | 2n |
| increment count | 1 | N | N |
| else { |  |  |  |
| if (token exists in Hashtable ) add token to acourse | 1 | n—1 | N |
| else |  |  |  |
| Print “Error” | 1 | 1 | 1 |
| if (count < 2) { | 1 | N | N |
| Print "Error" | 1 | 1 | 1 |
| Add acourse to Hashtable | 1 | N | N^2 |
| **Total Cost** |  |  | **N^2+11n+9** |
| **Runtime** |  |  | **O(n^2)** |

**Binary Search Tree (BST)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **Times Executed** | **Cost** |
| Initialize fstream fileStream | **1** | **1** | **1** |
| Initialize string line | **1** | **1** | **1** |
| Initialize stringstream lineStream | **1** | **1** | **1** |
| Initialize string token | **1** | **1** | **1** |
| Open fileName with fileStream | **1** | **1** | **1** |
| Initialize int count | **1** | **1** | **1** |
| Fill lineStream with current line | **1** | **N** | **N** |
| Set count to 1 | **1** | **N** | **N** |
| Create Course acourse for each line in file | **1** | **N** | **N** |
| Get token from lineStream up to ‘,’ until none left | **1** | **2n** | **2n** |
| if (count = 1) | **1** | **N** | **N** |
| set acourse’s courseNumber to token | **1** | **N** | **N** |
| increment count | **1** | **N** | **N** |
| else if (count = 2) | **1** | **N** | **N** |
| set acourse’s courseName to token | **1** | **N** | **N** |
| increment count | **1** | **N** | **N** |
| else |  |  |  |
| if (token exists in bst as a course) | **1** | **N** | **N** |
| add token to acourse’s PreReqs | **1** | **N** | **N** |
| Else output file format error | **1** | **1** | **1** |
| increment count | **1** | **N** | **N** |
| if (count < 2) | **1** | **1** | **1** |
| output "Error in file format” | **1** | **1** | **1** |
| add acourse into bst | **1** | **N** | **N^2** |
| **Total Cost** |  |  | **N^2+14n+9** |
| **Runtime** |  |  | **O(n^2)** |

**Vector advantages:**

1. Easily implemented

2. Traversing is efficient

3. Insertion at end is in constant time

**Vector disadvantages:**

1. The insertion or deletion in front takes more time

2. It doesn’t work efficiently if the data size is too large

3. Must be sorted to use search capabilities

**Hash Tables advantages:**

1.Insertion and deletion can be performed in constant time

2. Direct access to items

3. Quick search is possible.

**Hash Tables disadvantages:**

1. Implementation is difficult

2. Too much space is needed

3. Storing of random items can cause issues

**BST advantages:**

1. Less space is required

2. Easy to implement

3. Retrieves items in order

**BST disadvantages:**

1. Overkill for small numbers of elements

2. Only works on sorted lists

3. Tree needs to stay balanced

**Recommendation**

For this assignment I recommend the vector. After looking at all the data the BST would probably be faster, but the vector is easily implemented and can work for small item lists. The drawback of having to be sorted is not an issue in this case. The quick insertion and traversing make it a perfect candidate for this project. The BST is my second choice due to speed and the hash table is too large too and cumbersome with the data handling to consider in this application.