**Module 6-2: Project One**

**Christine Hershey**

**CS-300-T5601 DSA: Analysis and Design**

**June 9, 2022**

**//Vector Pseudocode**

**//Loading data into the data structure**

Ifstream myfile

Myfile.open(“name.csv”)

If the file fails to open

Print error

courseNames = second string of each of the lines

WHILE (myfile.good() not the end of file

String line;

Getline(myfile. Line)

Course(line)

**//creating course objects and storing into the vector**

Vector<course>courses

Course(line)

String name

Course \*courseName = new Course

FOR each string

//two parameters on each line//

courseName->number = the first parsed string

courseName->name = the second parsed string

IF there is no number or name

Error

Return

FOR each of the remaining string

**//this ensures any prerequisite exist as a course//**

IF prerequisites name is found in courseName

Prerequisites = first parsed string

courseName->prerequisites=prerequisites

prerequisites ++

Course.push\_back(courseName)

**//searching and printing specific course and printing information**

Cin<<search the item

FOR each course.at(i) with i less than course length

IF the course name = searched name

Print course information

Print prerequisites course information

**//print the course list**

selectionSort(courses)

unsigned int min

FOR i < course.size() ++ i

min equals i

FOR j= i + 1 j<courses.size() ++ j

IF courses.at j less than courses.at i

min equals j

IF min not equal i

swap courses at i and min

FOR i < courses.size()

Print all courses. at (i)

**//courses printed alphanumerically**

selectionSort(courses)

WHILE i<courses.size()

Print sourted courses.at(i)

**//Vector Run Time analysis for reading File**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **#Times Executes** | **Total Cost** |
| Ifstream  Myfile(“name.csv”) | 1 | 1 | 1 |
| IF file fails to open | 1 | n | n |
| Return error | 1 | 1 | 1 |
| courseNames = second string | 1 | 1 | 1 |
| while(myfile.good() | 1 | n | n |
| String substr; getlines(myfile, line) | 2 | n | n |
| Course(line) | 1 | n | n |
|  |  | **Total Cost** | 4n + 3 |
|  |  | **Run Time** | O(n) |

**//Vector Run Time analysis for Course Objects**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **#Times Executes** | **Total Cost** |
| For each string | 1 | 1 | 1 |
| courseName-> num = first string parsed | 1 | 1 | 1 |
| return error if no course name or num | 1 | 1 | 1 |
| For each string remaining | 1 | n | n |
| If prerequisite name found in courseNames | 1 | n | n |
| Prerequisites = remaining string parsed | 1 | n | n |
| courseName-> prerequisite = prerequisite | 1 | n | n |
| Prerequisites ++ Courses.push\_back(\*courseName ) | 1 | n | n |
|  |  | **Total Cost** | 6n + 3 |
|  |  | **Run Time** | O(n) |

**//Hash Table Pseudocode**

Struct Course

String courseName;

String courseNum;

Vector<string>prerequisite;

Const usigned int Default Size = 8;

Class HashTable

Struct node

Course \*course;

Node\*next;

Unsigned int key;

Node()

Node(Course course, unsigned key)

usigned int size = Default Size;

Vector<node>nodes;

HashTable();

Insert (Course course);

int numPrerequisitesCourses(Hashtable courses string courseNum)

create key hashing the courseNum

retrieve node using the key to create new node

WHILE node is not equal to nullptr

IF node pointer course courseNum is = to courseNum

Total prerequisites equal node.prerequisites.size();

FOR each prerequisite p in totalPrerequisites

Add prerequisites of p in total prerequisites

Print the total number of prerequisites

ELSE

Node = node pointer next

void printCourseInformation(Hashtable course, String courseNumber)

create a key by hashing the courseNum

retrieve node by using the key and set key to a new node

WHILE node is not equal to nullptr

IF node pointer course courseNum is equal to courseNum

print course information

FOR each prerequisite of the course

print prerequisite course information

ELSE

node = node pointer next

Int main()

HashTable\* table = new HashTable()

Vector<string>temp

String line

Ifstream infile(“file name”)

WHILE(getline(infile, line))

Stringstream ss(line)

WHILE(ss.good())

String substr

getline(ss, substr, ‘,’)

temp.push\_back(substr)

table.insert(parseLine(temp))

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **#Times Executed** | **Total Cost** |
| Create a key by hashing the courseNum | 1 | n | n |
| while node != nullptr | 1 | n | n |
| If node→course courseNum != courseNum | 1 | n | n |
| print course information | 1 | 1 | 1 |
| for each prerequisite of the course | 1 | n | n |
| print prerequisite information | 1 | n | n |
|  |  | **Total Cost** | **5n +1** |
|  |  | **Run Time** | **O(n)** |

temp clear()

//**Hash Table Run Time analysis**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **#Times Executed** | **Total Cost** |
| HashTable\* table = new HashTable() Vector<string> temp | 2 | 1 | 1 |
| Ifstream infile(“filestream”) | 1 | 1 | 1 |
| while(getline(infile, line)) | 1 | n | n |
| Stringstream ss(line) | 1 | n | n |
| while(ss.good()) | 1 | n | n |
| String substr; getline(ss, substr, ‘,’) temp.push\_back(substr) | 3 | n | n |
| table.insert(parseLine(temp)); | 1 | n | n |
|  |  | **Run Time** | **5n + 2** |
|  |  | **Total Cost** | **O(n)** |

**//Binary SearchTree Pseudocode**

Struct Course

String courseName

String courseNum

Vector<string>Prerequisite

Course()

Struct Node

Course course

Node\* right

Node\* left

Node()

Right = left = nullptr

Node(Course aCourse) : Node()

this→course = Course

**//BST = BinarySearchTree**

class CourseBST

Add

Remove

Search

Inorder

void CourseBST∷PrintCourseInfo(string courseNum)

this→printCourseInfo(root, courseNum)

voidCourseBST∷addNode(Node\* node, Course course)

IF(node→course courseNum compares (course.courseNum) > 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(courses)

ELSE

this→addNode(node→right, course)

void CourseBST∷printSampleSchedule(Node\* node)

IF(node != nullptr)

printSampleSchedule(node→left)

print node→course.courseNum, node→course,courseName

printSampleSchedule(node→right)

return

void CourseBST∷printCourseInfo(Node\* current, string courseNum)

WHILE (current != nullptr)

IF (current→course.coureNum compares(courseNum) = 0

print current course.courseNum, current course.courseName

unsigned int size = NumPrerequisitesCourses(current→course)

print prerequisites

unsigned int i = 0

FOR (i = 0, I<size i ++)

print current→course.preRequisites.at(i)

IF( i != size – 1)

print “, “

IF (i = 0)

print no prerequisites needed

return

ELSE

IF (courseNum compare(current→course. courseNum) < 0 current equals

Current = current→left

ELSE

current equals current→right

int main()

Tree\* tree = new Tree()

Vector<string>temp

String line

Ifstream infile(“file name”)

WHILE(getline(infile,line))

Stringstream ss(line)

WHILE(ss.good())

string substr

getline(ss, substr, ‘,’)

temp.push\_back(substr)

tree.insert(parseLine(temp))

temp.clear

**//This Prints the course list**

inOrder (Node\* node)

IF node != nullptr

inOrder→node.left

print course information

inOrder→node.right

**//Prints the list alphanumerically**

Bst→inOrder()

print inOrder

**//BST Run Time and Analysis**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **#Times Executed** | **Total Cost** |
| If(node courseNum compares courseNum | 1 | 1 | 1 |
| If(node left = nullptr) | 1 | 1 | 1 |
| add the new node to the left subtree | 1 | 1 | 1 |
| ELSE(traverse nodes left subtree) | 1 | n | n |
| If(node right equals nullptr) | 1 | 1 | 1 |
| add the new node to the right subtree | 1 | 1 | 1 |
| ELSE(traverse node right subtree) | 1 | n | n |
|  |  | **Total Cost** | **2n+5** |
|  |  | **Run Time** | **O(n)** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **#Times Executed** | **Total Cost** |
| Tree\* tree = new Tree() | 2 | 1 | 1 |
| Ifstream infile(“filestream”) | 1 | 1 | 1 |
| while(getline(infile, line)) | 1 | n | n |
| Stringstream ss(line) | 1 | n | n |
| while(ss.good()) | 1 | n | n |
| String substr; getline(ss, substr, ‘,’) temp.push\_back(substr) | 3 | n | n |
| tree.insert(parseLine(temp)) | 1 | n | n |
|  |  | **Total Cost** | **4n + 2** |
|  |  | **Run Time** | **O(n)** |

**//Menu that is printed**

CourseBST\* coursesBST = nullptr

print Hello!! Welcome to your course Planner

String option = 0

int userOption = option[0] – ‘0’

WHILE (userOption != 9)

print 1. Load Data Structure

print 2. Print The Course List

print 3. Print Course print

print 9. Exit

Print Please select a option

Cin >> option

IF (userOption.length() = 1)

userOption = option[0] – ‘0’

ELSE

userOption equal 0

bool success = 1

switch(userOption)

case 1:

IF(courseBST = nullptr)

courseBST = new CourseBST()

IF (csv.Path.length() = 0)

print Please enter the file:

cin >> csvPath

correct equals loadCourses(csvPath, coursesBST)

IF(correct)

print Courses have been loaded

ELSE

print Error, courses not found

csvPath equal “ “

break

case 2:

IF(courseBST != nullptr and correct)

print Sample course schedule

coursesBST->PrintSampleSchedule()

ELSE

print loaded courses

break

case 3:

IF(coursesBST != nullptr and correct)

IF(courseId.length() = 0)

print Please select a course to get course information

cin>> coursed

FOR(auto& userOption : coursed) userOption equals userOption

coursesBST->PrintCourseInfo(courseId)

ELSE

print loaded course

coursed equals “ “

break

default:

IF(userOption != 9)

print option not valid

break

print Thank you, Goodbye!

**Evaluation and Comparison**

Vector has the advantage of being the fastest for reading the files and adding objects. Vector is also in my opinion, the easiest out of all three structures. Items are just added to the end of the vector, but with that comes a slower run time. Vector is also a bit slower because when searching a specific course number, it must search each item until it finds a match.

The Hash Table has the advantage of quickly searching the list. This is because hash Tables create keys, and the courses will be easily searched and located. Hash tables, however, can be a little slower when implementing and creating the lists because each item has to have a key created and find where to be inserted. In addition, a disadvantage is that the table cannot be sorted. Each value must be extracted, sorted, and printed to sort alphanumerically.

One of the most significant advantages of a Binary Tree is that it searches faster than a vector. This is because searching for a specific course is quickly done. Running down the tree until the value is found is much faster than the vector but maybe not as easy as the Hash Table. A disadvantage would be that if all nodes on the tree ended up on the left, it would have to search each element.

**Recommendation**

Each data structure would work for this project. It is just a matter of which one I would find most useful. I think that the vector data structure would be too slow for what the client is looking for. The Hash Table would not provide an organized list. This project requires the courses to be listed alphanumerically, and I would suggest using a Binary search tree. The list could be traversed with a tree much easier than a vector or hash table.