Project One

Thomas Fiske

10/4/2022

# Vector pseudocode

CLASS Course {

String courseNumber

String courseName

String preReq1

String preReq2

}

FUNCTION - printCourseInformation (Vector<Course> courses, string courseNumber) {

PRINT ( “Please input the name or course number of the course you are looking for”)

INPUT IN >> searchInput

IF ( find(courses.begin, courses.end, searchInput) != courses.end ){

PRINT course.courseNumber

PRINT course.courseName

PRINT course.preReq1

PRINT course.preReq2

}

ELSE {

PRINT(“Cannot find that course, please try again)

}

}

FUNCTION - readFileToVector(Vector<Course> courses, filename){

If stream fileName

STRING courseNumber

STRING courseName

STRING preReq1

STRING preReq2

Course tempCourse

fileName.open(“Course information file”)

WHILE (fileName.open)) {

If (argc > 2) {

PRINT “This file does not have 2 or more arguments”

}

}

ELSE IF (argc == 2){

fileName >> courseNumber

fileName >> courseName

tempCourse.courseNumber = courseNumber

tempCourse.courseName = courseName

courses.push\_back(tempCourse);

}

ELSE IF (argc == 3){

fileName >> courseNumber

fileName >> courseName

fileName >> preReq1

tempCourse.courseNumber = courseNumber

tempCourse.courseName = courseName

tempCourse.preReq1 = preReq1

courses.push\_back(tempCourse);

}

ELSE {

fileName >> courseNumber

fileName >> courseName

fileName >> preReq1

fileName >> preReq2

tempCourse.courseNumber = courseNumber

tempCourse.courseName = courseName

tempCourse.preReq1 = preReq1

tempCourse.preReq1 = preReq2

courses.push\_back(tempCourse);

}

}

**FUNCTION - MAIN**

STRING searchInput

readFileToVector(Vector<Course> courses, filename)

WHILE TRUE {

PRINT ( “Please input the name or course number of the course you are looking for”)

INPUT IN >> searchInput

IF ( find(courses.begin, courses.end, searchInput) != courses.end ){

PRINT course.courseNumber

PRINT course.courseName

PRINT course.preReq1

PRINT course.preReq2

Exit loop

}

ELSE

PRINT(“Cannot find that course, please try again)

Continue loop

}

# Hash Table Psuedocode

CLASS Course {

String courseNumber

String courseName

String preReq1

String preReq2

}

VECTOR <Node> nodes

FUNCTION numPrerequisiteCourses(Hashtable<Course> courses) {

prerequisites = hashtable(courses)

while (nodes != nullptr) {

if ( course number = key) {

print course info

}

For ( each prerequisites)

Print prerequisite information

}

}

FUNCTION printSampleSchedule(Hashtable<Course> courses) {

while (nodes != nullptr) {

print course info

if (course has a node on the left) {

PRINT prerequisite to the left

}

ELSE IF (course has a node on the right) {

PRINT prerequisite to the right

}

}

FUNCTION printCourseInformation(Hashtable<Course> courses, String courseNumber) {

while (nodes =! Nullptr) {

if (nodes->next == key) {

PRINT course info

}

}

}

FUNCTION - readFileToVector(Vector<Course> courses, filename){

If stream fileName

STRING courseNumber

STRING courseName

STRING preReq1

STRING preReq2

Course tempCourse

fileName.open(“Course information file”)

WHILE (fileName.open)) {

If (argc > 2) {

PRINT “This file does not have 2 or more arguments”

}

}

ELSE IF (argc == 2){

fileName >> courseNumber

fileName >> courseName

tempCourse.courseNumber = courseNumber

tempCourse.courseName = courseName

courses.push\_back(tempCourse);

}

ELSE IF (argc == 3){

fileName >> courseNumber

fileName >> courseName

fileName >> preReq1

tempCourse.courseNumber = courseNumber

tempCourse.courseName = courseName

tempCourse.preReq1 = preReq1

courses.push\_back(tempCourse);

}

ELSE {

fileName >> courseNumber

fileName >> courseName

fileName >> preReq1

fileName >> preReq2

tempCourse.courseNumber = courseNumber

tempCourse.courseName = courseName

tempCourse.preReq1 = preReq1

tempCourse.preReq1 = preReq2

courses.push\_back(tempCourse);

}

}

**FUNCTION - MAIN**

STRING key

FUNCTION CALL - readFileToVector(Vector<Course> courses, filename)

WHILE TRUE {

PRINT ( “Please input the name or course number of the course you are looking for”)

INPUT IN >> key;

FUNCTION CALL - numPrerequisiteCourses(Hashtable<Course> courses)

FUNCTION CALL - printSampleSchedule(Hashtable<Course> courses)

FUNCTION CALL - printCourseInformation(Hashtable<Course> courses, key)

}

ELSE

PRINT(“Cannot find that course, please try again)

# Binary Search Tree Psuedocode

STRUCT Course {

STRING courseNumber

STRING title

STRING preReq

}

STRUCT Node\* {

INT data

Node\* left

Node\* right

}

CONSTRUCTOR - Node() {

left = nullptr

right = nullptr

}

INITIALIZE - Node(Course aCourse) :

Node() {

course = aCourse

}

}

CLASS - BinarySearchTree {

PRIVATE:

Node\* root

void addNode(Node\* node, Course course)

Node\* removeNode(Node\* node, STRING courseNumber)

public:

BinarySearchTree()

virtual ~BinarySearchTree()

void Insert(Course course)

void Remove(STRING courseNumber)

Course Search(STRING courseNumber)

}

CONSTRUCTOR - BinarySearchTree::BinarySearchTree()

root = nullptr

}

DESTRUCTOR - BinarySearchTree::~BinarySearchTree() {

}

FUNCTION - BinarySearchTree::Insert(Course course) {

IF (root == nullptr){

root = new Node(course);

}

ELSE {

this->addNode(root, course);

}

}

void BinarySearchTree::Remove(STRING courseNumber) {

this->removeNode(root, courseNumber)

}

FUNCTION - Course BinarySearchTree::Search(STRING courseNumber) {

Node\* current = root

while (current != nullptr) {

IF (current->course.courseNumber.compare(courseNumber) == 0)

return current->course

}

IF (current->course.courseNumber.compare(courseNumber) < 0)

current = current->left

}

ELSE {

current = current->right

}

}

Course course

return course

}

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

IF (node->course.courseNumber.compare(course.courseNumber) > 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->left, course);

}

}

}

FUNCTION - Node\* BinarySearchTree::removeNode(Node\* node, STRING courseNumber) {

IF (node == nullptr) {

}

IF (courseNumber.compare(node->course.courseNumber) < 0) {

node->left = removeNode(node->left, courseNumber);

}

ELSE IF (courseNumber.compare(node->course.courseNumber) > 0 ){

node->right = removeNode(node->right, courseNumber);

}

ELSE {

IF (node->left == nullptr && node->right == nullptr) {

delete node;

node == nullptr;

}

ELSE IF (node->left != nullptr && node->right == nullptr) {

Node\* temp = node;

node = node->left;

delete temp;

}

ELSE IF (node->right != nullptr && node->left == nullptr) {

Node\* temp = node;

node = node->right;

delete temp;

}

ELSE {

Node\* temp = node->right;

while (temp->left != nullptr) {

temp = temp->left;

}

node->course = temp->course;

node->right = removeNode(node->right, temp->course.courseNumber);

}

}

return node;

}

FUNCTION - loadCourses(STRING csvPath, BinarySearchTree\* bst) {

csv::Parser file = csv::Parser(csvPath)

try {

for (unsigned int i = 0; i < file.rowCount(); i++) {

Course course

course.CourseNumber = file[i][1]

course.title = file[i][0]

course.preReq = file[i][8]

bst->Insert(course)

}

} catch (csv::Error &e) {

std::cerr << e.what() << std::endl

}

}

FUNCTION – displayMenu() {

PRINT – “Welcome, please insert the course number of the course you are looking for”

}

MAIN FUNCTION – INT MAIN () {

DECLARATIONS -

BinarySearchTree\* bst

bst = new BinarySearchTree()

Course course

STRING filePath = (File location)

STRING courseInput

IF STREAM fileName;

fileName.open(“Course information file”)

WHILE (fileName.open)) {

If (argc > 2) {

PRINT - “This file does not have 2 or more arguments”

}

Else {

PRINT - “FILE OPEN”

}

loadCourses(filePath, bst)

WHILE ( courseInput != “Q”) {

displayMenu()

INPUT - courseInput

PRINT - Course BinarySearchTree::Search(courseInput)

IF courseInput == “Q”

BREAK

}

}

# Menu and Course Print Psuedocode

FUNCTION – displayMenu(){

Int userChoice

PRINT – “Please choose an option, to choose an option, input the number next to the option. “

CIN >> userChoice

PRINT – “1: Load Data”

PRINT – “2: Print Course list”

PRINT – “3: Print Course”

PRINT – “4: Exit Application”

}

FUNCTION – printCourseList(Vector<Course> courses, filename){

Sort(courses.begin(), courses.end())

FOR ( i = 0; i < course.size(); ++i){

FOR( j = 1; j < course.size() +1; ++j){

PRINT – “Course” j

PRINT – courses.at(i).courseNumber

PRINT – courses.at(i).courseName

PRINT – courses.at(i).preReq1

PRINT – courses.at(i).preReq2

}

}

}

FUNCTION – main(){

WHILE TRUE {

displayMenu()

IF ( userChoice == 1){

readFileToVector(Vector<Course> courses, filename){

}

ELSE IF (userChoice == 2){

printCourseList(Vector<Course> courses, filename)

}

ELSE IF (userChoice == 3){

}

ELSE IF (userChoice == 4){

}

ELSE {

PRINT – “You chose”, userChoice, “That is not a correct option”

PRINT – “Please input a correct option”

Int userChoice()

}

# Evaluation

## Vector Run-time Analysis

|  |  |
| --- | --- |
| Vector Function | Run time |
| PRINT COURSE INFORMATION  READ FILE TO VECTOR | O(N)  O(N^2) |

## Hash Table Run-time Analysis

|  |  |
| --- | --- |
| Hash Table Function | Run Time |
| Num Prerequisite | O(n^2) |
| Print Sample Schedule | 0(n) |
| Print Course Information | O(n) |
| Read File to Vector | 0(n^2) |

## Binary Search Tree

|  |  |
| --- | --- |
| Binary Search Tree Function | Run Time |
| Search | 0(n) |
| Add Node | O(n^2) |
| Remove Node | O(n^2) |
| Load Courses | O(n) |

## Advantages/Disadvantages

The advantages of using a vector for a data structure is it is a very stable way to store data. Every piece of data that is put into a vector is given a position and that position can be called upon to retrieve that information. For example, if I needed to find the fourth thing in a vector, I can call upon that vector to give me the information about what is in that position in the vector. If in position three was the course CS-300 data structures & algorithms, and I called the vector like so: vector.at(3), it would return the course CS-300 Data Structures & Algorithms. Vectors can also be iterated through in order to print them out, so it makes it easy to print a list of values that are stored in a vector. The disadvantage of a vector is that it may need to be iterated through if the position of the data is not known in order to run it and that can increase run time.

The advantages of using a hash table are that a hash table can look up a value within a vector and find it easily. The hash table find attributes of the values within the vector, much like how objects have attributes, the keys in a vector can have them to and allows the code to find the value within the vector without iterating through it. This makes it so that the user can look through a vector and pull out the information that they need. The disadvantage of using a hash table is that the program needs to create a vector in order to use the hash table as well as all of the functions needed for the hash table. This can increase run time and code required for the program to work properly.

The binary search tree offers a great way to store data linearly. It is a great way to store files because of the hierarchal nature of the way files are stored on computers. There is a base location for a file, a drive, then the file can have files inside of it, and so on and so forth. This is much like how a binary search tree is set up and why it makes sense for data like this. In this application, we have a hierarchal nature of courses because there is a starter course, and then course that follow afterwards. This is why a binary search tree is a great way to store the data of the courses. The disadvantage of a binary search in order to find something in the tree, the program may have to run through several branches of the tree before finding it, this can increase run time.

## Recommendation

The data structure that I plan to use in my code will be a vector. The reason for this is because it has the least number of functions that are needed to run the code properly. It also is a secure way to store the data, as well as a good way to store the data about each course. The data for each course is a number, a name, and up to two perquisites. By creating an object for the courses, I can store the attributes within that, and then those objects within the vector. If I need to find a course within the vector, I can iterate through it with my given value and find it. The Vector also allows for me to easily print out a course list by iterating through it and printing each value.