Kelly Illescas

CS 300

Project One Pseudocode and Runtime Analysis

Open file: courseFile.open (“courseFile.csv”);

Access courseFile using loop:

if (!courseFile)

cout “file not found.”

else

cout “file found.”

Read file using while loop:

while (readLine)

if line = 2 parameters: cout courseNumber + “, “ + name;

else if line = 3 parameters: cout courseNumber + “, “ + name + “, “ + prerequisite1;

else if line = 4 parameters: cout courseNumber + “, “ + name + “, “ + prerequisite1 + “, “ + prerequisite2;

else: cout “Error. Please try again.”

Ensure prerequisite exists:

if prerequisite1 != courseNumber

cout “Error. Course not valid.”;

else if prerequisite2 != courseNumber

cout “Error. Course not valid.”;

else

cout “Course found.”;

Create course objects:

int numPrerequisiteCourses(Vector<Course> courses, Course a) {

totalPrerequisites = prerequisites of course a

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

int numPrerequisiteCourses(Vector<Course> courses, Course b) {

totalPrerequisites = prerequisites of course b

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

int numPrerequisiteCourses(Vector<Course> courses, Course c) {

totalPrerequisites = prerequisites of course c

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

int numPrerequisiteCourses(Vector<Course> courses, Course d) {

totalPrerequisites = prerequisites of course d

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

int numPrerequisiteCourses(Vector<Course> courses, Course e) {

totalPrerequisites = prerequisites of course e

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

int numPrerequisiteCourses(Vector<Course> courses, Course f) {

totalPrerequisites = prerequisites of course f

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

int numPrerequisiteCourses(Vector<Course> courses, Course g) {

totalPrerequisites = prerequisites of course g

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

int numPrerequisiteCourses(Vector<Course> courses, Course h) {

totalPrerequisites = prerequisites of course h

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

Print sample schedule:

void printSampleSchedule(Vector<Course> courses) {

}

Create course objects for hash table:

int numPrerequisiteCourses(Hashtable<Course> courses, Course a) {

totalPrerequisites = prerequisites of course a

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

int numPrerequisiteCourses(Hashtable <Course> courses, Course b) {

totalPrerequisites = prerequisites of course b

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

int numPrerequisiteCourses(Hashtable <Course> courses, Course c) {

totalPrerequisites = prerequisites of course c

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

int numPrerequisiteCourses(Hashtable <Course> courses, Course d) {

totalPrerequisites = prerequisites of course d

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

int numPrerequisiteCourses(Hashtable <Course> courses, Course e) {

totalPrerequisites = prerequisites of course e

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

int numPrerequisiteCourses(Hashtable <Course> courses, Course f) {

totalPrerequisites = prerequisites of course f

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

int numPrerequisiteCourses(Hashtable <Course> courses, Course g) {

totalPrerequisites = prerequisites of course g

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

int numPrerequisiteCourses(Hashtable <Course> courses, Course h) {

totalPrerequisites = prerequisites of course h

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

Print sample schedule:

void printSampleSchedule(Hashtable <Course> courses) {

}

Print all courses and prerequisites:

void printCourseInformation(Vector<Course> courses, String courseNumber) {

for all courses:

if course is the same as courseNumber

cout courseInformation

for each prerequisite

cout prerequisiteInformation

Create course objects for hash table:

int numPrerequisiteCourses(Hashtable<Course> courses, Course a) {

totalPrerequisites = prerequisites of course a

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

int numPrerequisiteCourses(Hashtable <Course> courses, Course b) {

totalPrerequisites = prerequisites of course b

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

int numPrerequisiteCourses(Hashtable <Course> courses, Course c) {

totalPrerequisites = prerequisites of course c

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

int numPrerequisiteCourses(Hashtable <Course> courses, Course d) {

totalPrerequisites = prerequisites of course d

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

int numPrerequisiteCourses(Hashtable <Course> courses, Course e) {

totalPrerequisites = prerequisites of course e

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

int numPrerequisiteCourses(Hashtable <Course> courses, Course f) {

totalPrerequisites = prerequisites of course f

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

int numPrerequisiteCourses(Hashtable <Course> courses, Course g) {

totalPrerequisites = prerequisites of course g

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

int numPrerequisiteCourses(Hashtable <Course> courses, Course h) {

totalPrerequisites = prerequisites of course h

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

Print sample schedule:

void printSampleSchedule(Hashtable <Course> courses) {

}

Print all courses and prerequisites:

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

for all courses:

if course is the same as courseNumber

cout courseInformation

else for each prerequisite

cout prerequisiteInformation

Create course objects for hash table:

int numPrerequisiteCourses(Tree<Course> courses, Course a) {

totalPrerequisites = prerequisites of course a

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

int numPrerequisiteCourses(Tree <Course> courses, Course b) {

totalPrerequisites = prerequisites of course b

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

int numPrerequisiteCourses(Tree <Course> courses, Course c) {

totalPrerequisites = prerequisites of course c

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

int numPrerequisiteCourses(Tree <Course> courses, Course d) {

totalPrerequisites = prerequisites of course d

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

int numPrerequisiteCourses(Tree <Course> courses, Course e) {

totalPrerequisites = prerequisites of course e

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

int numPrerequisiteCourses(Tree <Course> courses, Course f) {

totalPrerequisites = prerequisites of course f

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

int numPrerequisiteCourses(Tree <Course> courses, Course g) {

totalPrerequisites = prerequisites of course g

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

int numPrerequisiteCourses(Tree <Course> courses, Course h) {

totalPrerequisites = prerequisites of course h

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

Print sample schedule:

void printSampleSchedule(Tree <Course> courses) {

}

Print all courses and prerequisites:

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

for all courses:

if course is the same as courseNumber

cout courseInformation

else for each prerequisite

cout prerequisiteInformation

Insertion sort for courses:

InsertionSort (courseNumber, numbersSize) {

i = 0;

j = 0;

temp = 0; //temp value for swap

for (i = 1; i < numbersSize; ++i) {

j = i;

//insert numbers[i] into sorted part, stopping once numbers [i] in correct position

while (j > 0 && numbers[j] < numbers[ j – 1]) {

//swap numbers [j] and numbers[ j – 1]

temp = numbers[j]

numbers[j] = numbers[ j – 1]

numbers[ j – 1] = temp

--j

}

}

}

Main Menu:

int main() {

courseNumber = { CSCI100, CSCI101, CSCI200, MATH201, CSCI300, CSCI301, CSCI350, CSCI400 }

NUMBERS\_SIZE = 8

i = 0

cout << “UNSORTED: “ << endl;

for(i = 0; i < NUMBERS\_SIZE; ++i) {

cout << courseNumber[i] << “ “ << endl;

}

InsertionSort(courseNumber, NUMBERS\_SIZE)

cout << “SORTED: “ << endl;

for(i = 0; i < NUMBERS\_SIZE; ++i) {

cout << courseNumber[i] << “ “ << endl;

}

void displayMenu () {

int userChoice;

cout “Please make a selection. Select 1 to load the data structure, 2 to print the course list, 3 to print a course and its prerequisites, or 4 to exit.”;

cin userChoice;

}

}

**Runtime Analysis**

Vector: O(n)

A vector is a useful tool for lists, but because adding an item causes it to immediately become unsorted, it is not ideal for runtime efficiency. That said, it is the second-best tool for this project.

Hash Table: O(n)

A perfect hash table maps items to buckets with no collisions. Because this is most likely not possible with *n* courses, we must focus on the worst case runtime, which maps items to buckets and then must use chaining due to collisions. The worst case, then, is O(n). The other disadvantage to the hast table is that it is unordered, so searching through it requires a key. Depending on the size of the hash table, this could become incredibly time-consuming.

Binary Search Tree: O(n)

I would definitely plan to use a binary search tree in my code, mainly because the runtime for a BST is much more efficient. It is also simpler to search for items in a BST; therefore, the BST is the ideal method for a course search. In the worst case, a BST can still have a runtime of O(n), but if the tree is organized well, its efficiency can easily be O(1).