Hatcher Blair

CS-300 Analysis and Design

Project 1

4/9/2023

Pseudo Code:

Int tablesize = 101

Const int HASH\_TABLE = 0

Const int VECTOR = 1

Const int TREE = 2

Struct Class {

String courseNum;

String courseName;

Vector<String> prerequisites;

}

Vector<Class> readCoursesVector (string pathToFile) {

Vector<class> classes

File = parser(pathToFile)

Try {

For ( i = 0; i < file.rowCount < i++ {

Class class;

Try {

Class.courseNum = File[i][0]

Class.courseName = File[i][1]

} Catch {

Improperly formatted file

}

If (File[i][2] exists) {

If (getCourseInfo(File[i][2] != null) {

Class.prerequisites.push\_back(File[i][2]

}

}

If (File[i][3] exists) {

If (getCourseInfo(File[i][3] != null) {

Class.prerequisites.push\_back(File[i][3]

}

}

If (File[i][4] exists) {

If (getCourseInfo(File[i][4] != null) {

Class.prerequisites.push\_back(File[i][4]

}

}

Classes.push\_back(class)

}

} Catch {

Error reading file

}

Return classes

}

Class getCourseInfoVector(string courseNum, class classes) {

Vector<class>::iterator it

It = find(classes.begin, classes.end, courseNum)

If (it == classes.end && classes.end.courseNum != courseNum) {

Return null

} else {

Return classes[it – classes.begin]

}

}

Vector<Class> readCoursesHash (string pathToFile) {

Vector<class> classes

File = parser(pathToFile)

Try {

For ( i = 0; i < file.rowCount < i++ {

Class class;

Try {

Class.courseNum = File[i][0]

Class.courseName = File[i][1]

} Catch {

Improperly formatted file

}

If (File[i][2] exists) {

If (getCourseInfo(File[i][2] != null) {

Class.prerequisites.push\_back(File[i][2]

}

}

If (File[i][3] exists) {

If (getCourseInfo(File[i][3] != null) {

Class.prerequisites.push\_back(File[i][3]

}

}

If (File[i][4] exists) {

If (getCourseInfo(File[i][4] != null) {

Class.prerequisites.push\_back(File[i][4]

}

}

hashTable->insert(class)

}

} Catch {

Error reading file

}

Return classes

}

Void hashTable insertHash (class) {

Key = hash(class.courseNum)

node = classes(key)

if(node.class == null) {

classes(key) = class

}

Else {

While (node.next != null) {

Node = node.next

}

Node.next = class

}

Int hashTable hash(key) {

Return key % tablesize

}

Vector<Class> readCoursesTree (string pathToFile) {

Vector<class> classes

File = parser(pathToFile)

Try {

For ( i = 0; i < file.rowCount < i++ {

Class class;

Try {

Class.courseNum = File[i][0]

Class.courseName = File[i][1]

} Catch {

Improperly formatted file

}

If (File[i][2] exists) {

If (getCourseInfo(File[i][2] != null) {

Class.prerequisites.push\_back(File[i][2]

}

}

If (File[i][3] exists) {

If (getCourseInfo(File[i][3] != null) {

Class.prerequisites.push\_back(File[i][3]

}

}

If (File[i][4] exists) {

If (getCourseInfo(File[i][4] != null) {

Class.prerequisites.push\_back(File[i][4]

}

}

classes.InsertTree(class)

}

} Catch {

Error reading file

}

Return classes

}

Void printClasses(Vector<class> classes) {

For (class in classes) {

Cout << class.courseNum << class.courseName

For (prereq in classes.prerequisites) {

Cout << prereq

}

Cout << endl

}

}

**Runtime analysis:**

**Printing: f(N) = N(4) + N2(2) + 1 = O(N)**

**Vector Insert: f(N) = 2 + N(13) = O(N)**

**Hash Table Insert: f(N) = 2 + N(15) = O(N)**

**Tree Insert: f(N) = 2 + N(15) = O(N)**

**Recommendation: All of the three different methods of storing and printing have a linear scaling over time, and so they will all sort comparably. However, the fastest structure for retrieving data is the tree. With a tree you are able to find a value with minimal operations and to print out values in the right order without needing to check the order. Because of this I am going to recommend that we implement this project using a tree data structure.**