CS 300 Pseudocode Document

Project One

Nii Amatey Tagoe

// Tree pseudocode **to define how the program opens the file reads the data from the file, parses each line, and checks for file format errors**

LoadAndValidateCourseData(File name){

Open File name for reading

If the file cannot be opened

Print "Error: Cannot open the file."

Initialize coursesTree

Initialize courses list as empty list

While not the end of the file

Read the line from the file

If the line is not empty

courseData = ParseLine(line)

If Validate Course Data(courseData, coursesList) is false

Print "Error: Invalid course data."

End If

Add courseData to coursesList

End If

End While

For each course data in courses list

Insert course data into courses Tree

End For

Close file

}

void ParseLine(line){

Split the line into tokens using commas to separate it

Return tokens

void Validate Course Data(courseData, coursesList)

If Length(courseData) < 2

Return false

End If

For each prerequisite in course data starting from index 2

If the prerequisite is not on the course list

Return false

End If

End For

Return true

}

//Tree pseudocode for creating course objects and storing

void InsertCourseDataIntoTree(coursesTree, courseData)

course number = courseData[0]

title = courseData[1]

prerequisites = Slice(course data, 2) // Get all elements from index 2 onwards

courseObject = new Course(courseNumber, title, prerequisites)

coursesTree.Insert(courseObject)

}

//Tree pseudocode for printing **out course information and prerequisites**

void printCourseInformation(coursesTree) {

If coursesTree is not empty

InOrderTraversal(coursesTree.Root)

Else

Print "There are no courses to be displayed."

End If

End Function

void InOrderTraversal(node)

If the node is not null or unoccupied

InOrderTraversal(node.Left)

PrintCourse(node.Value)

InOrderTraversal(node.Right)

End If

void PrintCourse(course)

Print "Course Number, " + course.courseNumber + ", Title, " + course.title

If course.prerequisites is not empty

Print "Prerequisites, " + Join(course.prerequisites, ", ")

Else

Print "Prerequisites: None"

End If

}

// Vector Pseudocode to define how the program opens the file reads the file, parses each line, and checks for file format errors

LoadCourseDataFromFile(filename){

Open file containing course with filename for reading

If file is successfully opened

Initialize vector

While not end of the file

Read a line from the file

If the line has at least two parameters

Parse the line into courseNumber, courseTitle, and prerequisites if there is any

if each prerequisite exists as a courseNumber in the vector

If validation is successful

Proceed to create and store course objects

Else

Print “error invalid prerequisite”

End If

Else

Print error about invalid line format

End If

End While

Else

Print “error unable to open file”

End If

}

// Vector Pseudocode to create course objects and store them in the appropriate data structure(vector)

void CreateAndStoreCourse(courseNumber, courseTitle, prerequisites) {//the parameter prerequisites is a vector

Create a new course object with courseNumber and courseTitle

For each prerequisite in prerequisites

Add prerequisite to the course object's prerequisites list

Add the course object to the vector

}

//Vector Pseudocode to search for a specific course and print out course information and prerequisites

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

**for all courses**

**if the course is the same as courseNumber**

**print out the course information**

**for each prerequisite of the course**

**print the prerequisite course information**

**else**

**Print “No prerequisite”**

**if the course is not found**

**Print “Course not found”**

**return**

}

//Hash Table Pseudocode to define how the program opens the file reads the file, parses each line, and checks for file format errors

LoadCoursesFromFile(filename) {

Initialize Hash Table

Open file with name filename for reading

If the file is not opened

Print "Error while opening file"

Endif

While not at the end of the file

Read the line from the file

Parse line into course number, course title, and prerequisites

If not IsValidFormat(course number, course title)

Print "This is an invalid file format"

Continue to the next iteration

Endif

//Creating a new Course object with the course number and course title

For each prerequisite in prerequisites

If not IsCourseExists(prerequisite, Hash Table)

Print "Prerequisite course was not found: ", prerequisite

Continue to the next iteration

Endif

Add prerequisite to the Course object

Next

Insert Course object into Hash Table with course number as key

End While

Close file

}

//Hash Table pseudocode code to validate and create course objects

Void IsValidFormat(course Number, course title){

If the course number is empty or the course title is empty

Return False

Endif

Return True

}

Void IsCourseExists(course number, Hash Table)

Return Hash Table containing course number

//Hash Table pseudocode to print out course information and prerequisites if there is any.

void printCourseInformation(Hash table<Course> courses, String course Number) {

For each key in hashTable keys

Set course = hashTable[key]

Print "Course Number, Course Title,

If course prerequisites are not empty

Print Prerequisites

For each prerequisite in course prerequisites

Print prerequisite

Else

Print "There are no prerequisites"

Endif

}

**Pseudocode for the menu**

Function LoadDataStructure(filename):

Open filename for reading

For each line in filename:

Parse the line to extract course information

Create a course object or dictionary with course details

Add the course object to data\_structure

Close filename

Print "Data loaded successfully."

Function PrintCourseList():

If data\_structure is empty:

Print "Must load the data structure first."

Return

Sort data\_structure by course number

For each course in data\_structure(Vector, Hash Table, or Tree):

Print course title

void PrintCourse(course\_number):

If data\_structure is empty:

Print "load the data structure first."

Return

For each course in data\_structure(Vector, Hash Table, or Tree):

If the course's ID matches course\_number:

Print "Title: ", course's title

Print "Prerequisites: ", course's prerequisites

Return

Print "Course not found."

void ExitTheProgram():

Exit the program

**Pseudocode that will print out the list of the courses in the Computer Science program in alphanumeric order.**

//Vector pseudocode

Void PrintCoursesFromVector(coursesVector)

Sort coursesVector by course Number that is alphanumerically

For each course in courses Vector

Print the course Number and course title

End function

//Hash Table pseudocode

Void PrintCoursesFromHashTable(coursesHashTable)

Initialize courses list as an empty list

For each key, value in courses Hash Table

Insert (key, value) into courses list

EndFor

Sort courses List by the key that is alphanumerically

For each (course Number, course title) in the courses list

Print course Number, Course Name

EndFor

//Tree pseudocode

Void PrintCoursesFromTree(root)

If the root is not null

PrintCoursesFromTree(root.left)

Print root.courseNumber, root.name

Print Courses From Tree(root.right)

EndIf

**Evaluating the run-time and memory of data structures**

**//Hash Table**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Initialize hash table** | 1 | 1 | 1 |
| Opening of file | 1 | 1 | 1 |
| **Checking if the file opened** | 1 | 1 | 1 |
| Read the line from the file | 1 | n | n |
| Parse line |  | n | n |
| Validating the format of the lines check | 1 | n | n |
| For each prerequisite | 1 | n \* p  where p is the number of prerequisites and n is the number of courses | np |
| If the course exists check | n | n \* p  where p is the number of prerequisites and n is the number of courses | n^2p |
| Adding a prerequisite | 1 | n \* p  where p is the number of prerequisites and n is the number of courses | np |
| Inserting course object | n | n | n^2 |
| Closing file | 1 | 1 | 1 |
| **Total Cost** | | | (n^2p) + (n^2) + (np) + n + 1 |
| **Runtime** | | | O(n^2p) |

**//Tree**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Opening of file** | 1 | 1 | 1 |
| Reading each line in the file | 1 | n | n |
| **Parsing each line** | 1 | n | n |
| Validating the course by checking the length | 1 | n | n |
| Validating the prerequisites by searching the list | n | n \* p  where p is the number of prerequisites and n is the number of courses | n^2p |
| Adding course data to the list | 1 | n | n |
| Inserting course data into the tree | Log n | n | n Log n |
| Closing the file | 1 | 1 | 1 |
| **Total Cost** | | | (n^2p) + (n Log n) + n + 2 |
| **Runtime** | | | O(n^2p) |

**//Vector**

| **Code** | **Line Cost** | **# Times Executed** | **Total Cost** |
| --- | --- | --- | --- |
| **Opening the file for reading** | 1 | 1 | 1 |
| Initialize vector | 1 | 1 | 1 |
| **Read each line from the file** | 1 | n | n |
| Parse the line | 1 | n | n |
| Check if the line has at least two parameters | 1 | n | n |
| Validate each prerequisite | n | n \* p  where p is the number of prerequisites and n is the number of courses | (n^2\*p) |
| Creating and storing the course objects | 1 | n | n |
| Adding prerequisites | 1 | n \* p  where p is the number of prerequisites and n is the number of courses | n\*p |
| Print “Error invalid prerequisite” | 1 | 1(worst case) | 1 |
| Print “error file cannot be opened | 1 | 1(worst case) | 1 |
| **Total Cost** | | | 4n + (n^2\*p) + n\*p |
| **Runtime** | | | O(n^2p) |

**The advantages and disadvantages of each structure in your evaluation.**

Based on the advisor's requirements each data structure possesses its advantages and disadvantages, primarily, the vector data structure generally has the advantage of direct and less complex access to the elements, and the vector data structure is effective and efficient in iterating over all elements. Some disadvantages include the slow performance of insertion as well as deletion.

Secondly, the tree data structure basically has a logarithmic performance and facilitates order-related queries. The downside is that the tree data structure involves tedious implementations compared to the other data structures.

Finally, the hash table data structure is more efficient with the searching, deletion, and insertion. In this case, courses can be looked up and created with much faster speed. The disadvantage of this scenario is that execution can degrade with poor hash function leading to collisions that is when an item being inserted into a hash table maps to the same bucket as an existing item in the hash table.

In conclusion, the data structure I will recommend or might use in my code in terms of the requirements and the Big O analysis above would be the hash table. Hash table I believe will provide much faster look-up times, also reading the file and creating objects.