Alexis Del Rosario

Project 1

CS-300 DSA: Analysis and Design

Dr. Webb

**Pseudocode for Vector**

1. Reading data from file:

Open the course information file

IF file cannot be opened, print error message and exit

Initialize empty vector data structure

Loop through lines in file

Check that there are 2 tokens, course number and title

IF not, print error message and continue to next line

1. Creating course objects and storing them in vector:

Create a class with instance variables for course number, title, and prerequisites

Initialize empty vector to store course objects

Loop through lines in file

Create course object

Store tokens in corresponding course object instance variable

Add prerequisite course to course object

Add course object to vector

Vector should have multiple course objects

1. Search vector for a specific course and print out course information and prerequisites:

Ask user for a course number

Loop through courses in course vector

IF course number matches input, return course object

IF no course number matches input, return null

Loop through prerequisites from list

Print prerequisite course number

IF none found, return null

**Pseudocode for Hash Table**

1. Design pseudocode to define how the program opens the file, reads the data from the file, parses each line, and checks for file format errors:

Open the CourseInformation.txt file

Generate hash table

FOR each line in file

Check IF line has less than two parameters

Display error message that line is invalid

Continue on to the next line

ELSE IF prerequisite course is not found, display error message

Continue on to the next line

ELSE

2. Design pseudocode to show how to create course objects and store them in the appropriate data structure:

Create “course” as an object

Using first parameter, set course number

Using second parameter, set course title

Using remaining parameters, set prerequisites

Using course number as key, add object to hash table

3. Design pseudocode that will print out course information and prerequisites:

Define function to print course info: function printCourseInfo (courseTable, courseNumber)

Get course object from hash table

Print course info (course number and title)

IF course has prerequisites

FOR each prerequisite

GET prerequisite from hash table

Print function printCourseInfo (courseTable, prerequisite)

Define function to print a sample schedule

Define courses to include in sample schedule

Courses = [“CS101”, “CS201”, CS301”, etc]

For each course in courses

printCourseInfo

Call functions to print course info and sample schedule

printCourseInfo(courseTable, “CS101”)

printSampleSchedule(courseTable)

**Pseudocode for Tree**

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

1. Open file inputFile(“CourseInformation.txt”);
2. Read lines from file while(getline(inFile, line)) {

Parse line to get course number, title and prerequisites }

Make sure there are no file format errors if(courseNumber.empty () || courseTitle.empty ()) { cout << error}

Check for 2 parameters on each line if(prerequisites.size() > 0) {

prerequisiteFound = true }

1. Create a new course object Course\* course = new Course

Insert course object into binary search tree

1. Close file inFile.close();

Design pseudocode to show how to create course objects and store them in the appropriate data structure:

1. Create a class named Course with instance variables for each parameter class Course {

string courseNumber

string title

vector<string> prerequisites;

1. Define function that uses a line of course information to create course object Course createCourseObject(string line) {

Parse line to get course number, title, and prerequisites

Create new course object Course course;

Set course number, title, and prerequisites of course object

}

1. Return course object return course;
2. Function to insert course object into binary search tree

void insertCourseObject(TreeNode &root, Course course) {

Traverse tree to find position based on course number

Insert course object into position

}

Design pseudocode that will print out course information and prerequisites:

1. Define function to print course information from binary search tree

Void printCourses(TreeNode\* root){

Traverse tree in order

For each course

Print course information

If they have prerequisites, print prerequisites

**Pseudocode for Menu**

Using a switch for menu

While user input != 4;

Output menu options

Option 1 – Load data structure

Option 2 – Print course list

Option 3 – Print Course

Option 4 – Exit

Get user input

Depending on user input, do the following:

If Case 1 is selected:

Call function for loading data structure;

Output message that data has been loaded;

Break;

If Case 2 is selected:

Call function for printing course list;

Output course list;

Break;

If Case 3 is selected:

Get user input for course number to print;

Call function to print course information and prerequisites;

Output course information and prerequisites;

Break;

If Case 4 is selected:

Exit program;

Output message that program is exiting;

Break;

Default:

Display message: no valid input was entered;

Break;

**Pseudocode to print list of courses in alphabetical order**

Get Computer Science courses list from data structure

Check to see if each course is a computer science course

If true, add course to a list

Using merge sort, sort courses in list in alphabetical order

Print sorted list with each course and information

**Runtime Analysis**

| **Vector** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create Vector** | 1 | 1 | 1 |
| **For each line** | 1 | n | n |
| **Create Vector course item** | 1 | n | n |
| **While prerequisite exists** | 1 | n | n |
| **Append Prerequisite** | 1 | n | n |
| **Pushback course item** | 1 | n | n |
| **Total Cost** | | | 5n + 1 |
| **Runtime** | | | O(n) |

| **HashTable** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create Hash Table** | 1 | 1 | 1 |
| **Insert Method** | 0 | 0 | 0 |
| **Generate key for course** | 1 | n | n |
| **If no entry found** | 1 | n | n |
| **Set node to key** | 1 | n | n |
| **Else** | 1 | n | n |
| **Set old node key to unitmax, set old node to course and old node next to null pointer** | 4 | n | 4n |
| **Else** | 1 | n | n |
| **Find next open node** | 1 | n | n |
| **Add new newNode to end** | 1 | n | n |
| **For each new line** | 1 | n | n |
| **Generate vector course item** | 1 | n | n |
| **While prerequisite exists** | 1 | n | n |
| **Append Prerequisite** | 1 | n | n |
| **Insert course item** | 1 | n | n |
| **Total Cost** | | | 16n + 1 |
| **Runtime** | | | O(n) |

| **Tree** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Add node method** | 0 | 0 | 0 |
| **If root is null, add root** | 1 | 1 | 1 |
| **If node is less than the root, add to left** | 1 | n | n |
| **If left node doesn’t exist** | 1 | n | n |
| **Node is set to left node** | 1 | n | n |
| **If node is greater than root, add to right node** | 1 | n | n |
| **If right node doesn’t exist** | 1 | n | n |
| **Node is set to right node** | 1 | n | n |
| **For each line** | 1 | n | n |
| **Create vector course item** | 1 | n | n |
| **While there are prerequisites** | 1 | n | n |
| **Append prerequisites** | 1 | n | n |
| **Insert course item** | 1 | n | n |
| **Total Cost** | | | 11n + 2 |
| **Runtime** | | | O(n) |

**Advantages and Disadvantages of Data Structures**

The advantage that the vector data structure has over the other data structures is its speed when reading a file and adding the course objects. Also, its runtime is the shortest out of all three data structures at 5n+1. A disadvantage to using a vector however is searching for an item as it has to search for each item individually. The advantage that the hash table has is being able to search for items relatively fast using a search key. A disadvantage of the hash table is that it is prone to collisions which can slow down look up times and its runtime is the highest at 16n+1. Trees have the advantage of being able to search very quickly, especially compared to a vector. Its disadvantage, however, is that it can be difficult to implement compared to the other structures.

**Recommendation**

Based on the advantages and disadvantages, I would recommend using the hash table for the program. Its ability to search for items rapidly compared to the other structures is one of the main reasons for my selection. It is also simple to add and remove elements and the fact that you can mitigate potential collisions with appropriate implementation is another reason for my recommendation.