# CS 300 Pseudocode Document

John Barden

8/6/23

CS-300: Project one

Project one pseudocode.

// Course struct pseudocode

Create structure for course information

Store courseNum and courseName

Create vector to hold prerequisites

// Node struct

Create node with course data structure

Create left and right pointer

Recursively call Node method with default values

Set left and right pointers to nullptr

Call node constructor with course information

Create default node

Set course information

// Search Linked-list class

Create class for binary search tree

Create root node and needed storage for nodes

Constructor

Deconstructor

Insert into linked-list

Search for course

// Search tree class

Create class for search tree

Create root node and needed storage for nodes

Constructor

Deconstructor

Insert into tree

InOrder method for alphabetical order

Search for course

// Search hashTable class

Create class for hashTable

Create root node and needed storage for nodes

Constructor

Deconstructor

Insert into table with hashCode

Search for course

// Vector pseudocode

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

}

void printSampleSchedule(Vector<Course> courses) {

**Use sort operation through the courses vector**

**For every element within the course vector**

**Print current node information**

}

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**

}

// Hashtable pseudocode

int numPrerequisiteCourses(Hashtable<Course> courses) {

Call this object’s numPrerequisiteCourses vector

Return size of vector

}

void printSampleSchedule(Hashtable<Course> courses) {

**Create pointer within hashtable**

**While pointer returns data**

**Print hashcode from lowest to highest order of courseNumber**

**for each prerequisite of the course**

**print the prerequisite course information**

**Step pointer**

}

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

**Create pointer within hashcode table**

**Take in courseNumber and convert into hashcode**

**If first node is equal to hashcode**

**Print node information**

**Call prereq vector**

**For every prerequisite**

**Print index**

**Continue to step pointer**

**If current node and searched node are equal**

**Print node information**

**Call prereq vector**

**For every prerequisite**

Print index

}

// Tree pseudocode

int numPrerequisiteCourses(Tree<Course> courses) {

Call this object’s numPrerequisiteCourses vector

Return size of vector

}

void printSampleSchedule(Tree<Course> courses) {

**Create pointer within tree at root**

**Recursively call inOrder method moving pointer left**

**Print current pointer node information**

**Recursivley call inOrder method moving pointer right**

}

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

**Create pointer within tree at root**

**If the difference between current and searched course number is 0**

**Call current node prereq vector**

**For every element in prereq vector**

**Increment int**

**Return int**

**Else If the difference between current and searched course number less than 0:**

**Move pointer left until they are equal**

**Else**

**Move pointer right until they are equal**

}

Inorder()

Recursively call Inorder method with root node

LoadCourses with csvPath and each data structures

Initialize parcer and create a parser with csvPath argument

Create vector for header information set to file header info

For every element in vector

Print header column with desired spacer

Surround with try catch block, if parser returns a error

For every row in file

Create a course object

Set objects members with file information for corresponding index value.

Course number index 0

Course name index 1

If file index 2 returns something

Add to objects prerequisite vector

Duplicate object 2 times

Add one object into tree

Add one object into hashTable

Add original object into vector

Verify no more objects

Print out error information from parser

DisplayMenu()

Display menu to user

1. Load courses
2. Print all
3. Print search course
4. Exit

Take in users choice

Use switch case for choice

Call methods for desired choice

.

## Example Runtime Analysis

When you are ready to begin analyzing the runtime for the data structures that you have created pseudocode for, use the chart below to support your work. This example is for printing course information when using the vector data structure. As a reminder, this is the same pairing that was bolded in the pseudocode from the first part of this document.

| **Vector Print search courses** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 1 | n | n |
| **if the course is the same as courseNumber** | 1 | n | n |
| **print out the course information** | 1 | 1 | 1 |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 1 | n | n |
| **Total Cost** | | | 4n + 1 |
| **Runtime** | | | O(n) |

| **Vector Print all courses** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **For every element within the course vector** | n | n | n |
| **Print current node information** | 1 | 1 | 1 |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 1 | n | n |
| **Total Cost** | | | 3N+1 |
| **Runtime** | | | O(n) |

| **Hash code Print all courses** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **While pointer returns data** | 1 | n | n |
| **Print hashcode from lowest to highest order of courseNumber** | i | n | n |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 1 | n | n |
| **Total Cost** | | | 4n + 1 |
| **Runtime** | | | O(n) |

| **Hash code print searched course** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Take in courseNumber and convert into hashcode** | **1** | **n** | **n** |
| **If first node is equal to hashcode** | **1** | **N** | **N** |
| **Print node information** | **1** | **1** | **1** |
| **For every prerequisite** | **1** | **n** | **n** |
| **print the prerequisite course information** | **1** | **1** | **1** |
| **If current node and searched node are equal** | **1** | **n** | **n** |
| **Print node information** | **1** | **1** | **1** |
| **Total Cost** | | | **4N+3** |
| **Runtime** | | | O(N) |

| **Tree code Print courses** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create pointer within tree at root** | 1 | 1 | 1 |
| **Recursively call inOrder method moving pointer left** | 1 | O(log N) | O(log N) |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 1 | 1 | 1 |
| **Total Cost** | | | log(N)+2 |
| **Runtime** | | | O(log(N)) |

Vector structure:

Vector data structure has linear search speed a decent search speed. The larger the data base the slower a search will become at a constant rate. The print all in alphabetical order has a print all time complexity of N\*log(N) a rather unfavorable time. With increasing courses the search time will greatly increase the time for search. Vector data structure has built in sort functions that simplify the code needed to write, but there is extra code for checks that will result in a larger memory load.

Hash table:

Hash table have both linear time complexity for both search and print all courses. This is a decent time complexity and rather consistent. Hash tables require weight balancing to reduce the amount of collisions that occur, Every collision contributes to the time for insertion. Hash tables are comparable with the vector data structure.

Search tree:

The binary search tree has a logarithmic curve, much better then linear. This is due to the nature of a search tree dividing the tree in half and recursively search the half closest to the target. This requires balancing if one side collects too many objects. This will adversely effect the time complexity of the search tree. With the requirement of prerequisites help mitigate that issue due to higher level classes requiring at least one prerequisites or there is a chain with alternatives. The tree data structure is the most favorable data structure to use to control all courses.