Francis Lapointe

CS 300 Project 1

8/6/2023

CS 300 Project One Pseudocode

**Reading Data from File:**

Use fstream to Open File;

Call to open file, if file is not found return -1;

Else { //file is found

While (not end of file) {

Read each line;

if course has less than 2 parameters on each line, return error;

if course has 2 or more parameters: continue;

}

Close file;

}

**Class Definitions:**

**Vector:**

Class Course {

Int courseNum;

String courseName;

Vector<string> prerequisite

}

**Hash Table:**

Class Course {

Int courseNum;

String courseName;

Vector<string> prerequisite

}

**Binary Tree:**

Class Course {

Int courseNum;

String courseName;

String prerequisite;

}

**Creating Course Objects:**

**Vector:**

Initialize Vector;

Loop Through file {

For first Value and second value,

Pushback and add value to vector;

If third Value exists:

Pushback and add value to vector;

}

**Hash Table:**

Create HashTable class

Create Insert method;

Loop through file {

For first value in file, convert courseNum to string, store in HashTable;

For second value, store in HashTable;

If third value exists,

Store in HashTable, else store 0 (no prerequisite);

}

**Binary Tree:**

Create Binary Search Tree Class

Create Insertion Method

Loop through file, storing prerequisites as parents of specific courses;

**Menu:**

Output Menu Options (switch statement)

1. Case 1: Load Courses //loads the data into courses using the insertion methods
2. Case 2: Print Course Lists // Prints List of Data in Alphanumerical order
3. Case 3: Print Course // Print a specified course and its information
4. Cast 9: Exit // closes program

**Print Courses:**

**Vector:**

For (int I = 0; I < size; I++){

Output CourseNum, output CourseName, if (there’s a prerequisite) output Prerequisite; end line;

}

**Hash Table:**

For (int I = 0; I < size of table; I++){

Output CourseNum, output CourseName, if (there’s a prerequisite) output Prerequisite; end line;

}

**Binary Tree:**

To print in Numerical order one would use preorder Traversal.

Preorder(){

preorder(root); //root = first course in list

}

preorder(Course\* course) {

If( course is not null null)

Output CourseNum, output CourseName;

if (there’s a prerequisite)

{output Prerequisite; end line;}

preorder(course’s left child); Preorder(course’s right child);

}

**Search data:**

**Vector:**

Temp\* course;

While (temp does not equal null) {

If(temp’s name equals name of course){

Return temp’s name (course if found)

}

Check the next course in the list of courses;

}

Else

{  
course is not found;

}

**Hash Table:**

Temp\* course;

While(tempt != null) {

If(temp’s name equals name of course)

{

Return course’s name;  
}

Check next course;

}

Else {

Course not found;

}

**Binary Tree:**

Search(string CourseNum){

set current Course equal to root;

while(current is not null){

if the courseNum of the current node is equal to the courseNum;

{ return course found;

}

if CourseNum of current is less than the CourseNum it’s searching for{

current is set equal to current’s left child //courseNum is smaller

}

Else {

Current is set equal to current’s right child //courseNum is larger

}

}

Course course;

Return course;

}

Run Time Analysis

**Vector**

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| For all Courses | 1 | N | N |
| If Course is Same CourseName | 1 | N | N |
| Print Course Information | 1 | 1 | 1 |
| For each Prerequisite | 1 | N | N |
| Print prerequisite Info | 1 | N | N |
|  |  | Total Cost | 5n + 1 |
|  |  | Runtime | O(N) |

Run Time Analysis

**Hash**

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| For all Courses | 1 | N | N |
| If Course is same as Course Num | 1 | N | N |
| Print Course Info | 1 | 1 | 1 |
| For each Prerequisite | 1 | N | N |
| Print prerequisite info | 1 | N | N |
|  |  | Total Cost | 5n + 1 |
|  |  | Runtime | O(N) |

Run Time Analysis

**Binary Search Tree**

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| For all Courses | 1 | N | N |
| If Course is same as Course Num | 1 | N | N |
| For each Prerequisite | 1 | N | N |
| Print Prerequisite Info | 4 | N | N |
| Print Course Information | 1 | N | 1 |
|  |  | Total Cost | 8n + 1 |
|  |  | Runtime | O(N) |

**Evaluation**

Each data structure provides its own unique advantages and disadvantages for the project’s requirements. Vectors have a disadvantage when searching for a specific course while having the advantage of being the quickest at insertions. Vectors also hold a disadvantage when it comes to deletion. Ultimately due to these disadvantages I cannot recommend it for this assignment.

Hash Tables carry the disadvantage of resolving collisions (two data sets holding the same ID) being very inefficient, which if this application were to be expanded would ultimately occur if there was any sort of change to a curriculum. They also come with the disadvantage of being unable to maintain the order of elements meaning that sorting and printing data becomes less efficient as the data would need to be extracted, sorted and then printed. Due to these disadvantage I cannot recommend it for this assignment.

Binary Search Trees have an advantage in sorting and searching but takes longer when it comes to modification. It also comes at a downside of a greater total cost with a cost of 8n + 1 while maintaining a runtime complexity of O(N). Given the data set and problem for this project I recommend a Binary Search Tree to be used for implementation, I view that its downsides do not outweigh the benefits they provide when printing, searching, and organizing of data.