Mac Grier

CS 300

10/12/2025

**Project 1**

**Pseudocode**:

Main:

GET csvFilePath

IF NOT csvFilePath

csvFilePath = default

WHILE choice IS NOT exit

OUTPUT menu

GET INPUT choice

WHILE choice IS NOT valid

THROW ERROR

GET INPUT choice

GET INPUT dataStructure

WHILE dataStructure IS NOT valid

THROW ERROR

GET INPUT dataStructure

IF choice IS “Load Data”

IF dataStructure IS “Binary Search Tree”

NEW BinarySearchTree courseTree

LoadCourses()

Store in courseTree

ELSE IF dataStructure IS “Hash Table”

NEW HashTable courseTable

LoadCourses()

Store in courseTable

ELSE IF dataStructure IS “Vector”

NEW Vector<Course>

LoadCourses()

Store in Vector

ELSE IF choice IS “Validate”

IF dataStructure IS “Binary Search Tree”

ValidateTree(courseTree)

ELSE IF dataStructure IS “Hash Table”

ValidateTree(courseTable)

ELSE IF dataStructure IS “Vector”

Validate(courseVector)

ELSE IF choice IS “Print One”

GET INPUT searchParam

IF dataStructure IS “Binary Search Tree”

PRINT courseTree.search(searchParam)

ELSE IF dataStructure IS “Hash Table”

PRINT courseTable.search(searchParam)

ELSE IF dataStructure IS “Vector”

PRINT searchVector(searchParam)

ELSE IF choice IS “Print All”

IF dataStructure IS “Binary Search Tree”

PrintTree()

ELSE IF dataStructure IS “Hash Table”

PrintTable()

ELSE IF dataStructure IS “Vector”

printVector()

ELSE IF choice IS exit

OUTPUT “Goodbye”

PrintVector():

Sort(courseVector.begin(), courseVector.end(), [](course\* a, course\* b)

Return a.courseID < b.courseID

FOR course in courseVector

PRINT CourseID, CourseName, Prerequisites

SearchVector(searchParam):

Search = find(courseVector.begin(), courseVector.end(), searchParam)

IF search IS NOT courseVector.end()

RETURN string search.courseID, search.courseName, search.prerequisites

ELSE

RETURN not found

PrintTable():

FOR keyValuePair IN courseTable

ADD key TO keyList

Sort(keyList)

FOR key IN keyList

tempCourse = courseTable[key]

PRINT courseID, courseName, prerequisites

PrintTree():

IF courseTree.root IS NULL

Return

Curr = courseTree.root

Stack<node\*> nodeStack

WHILE curr IS NOT NULL OR nodeStack IS EMPTY

WHILE curr IS NOT NULL

nodeStack.push(curr)

curr = curr left

curr = nodeStack.top()

nodeStack.pop()

PRINT curr Node

Curr = curr right

Struct Course:

String courseID

String courseName

Vector<string> prerequisites

CLASS HashTable:

Struct Node:

String key

Course value

Node\* next

Insert(key, value)

Remove(key)

Search(key)

CLASS BinaryTree:

Struct Node:

Course key

Node\* left

Node\* right

Insert(node, key)

Search(node, key)

Remove(node, key)

AddCourses (String csvFilePath)

OPEN fileStream ( csvFilePath )

Vector<String> rawCourses

String inputLine

Course newCourse

WHILE fileStream NOT EMPTY

inputLine = getNextLine

split inputLine into different elements

IF inputLine size < 2 OR

PRINT invalid input

ELSE

newCourse.courseID = inputLine.at(0)

courseNumbers.append(newCourse.courseID)

newCourse.courseName = inputLine.at(1)

If inputLine.at(2) exists

newCourse.prerequisites = inputLine.at(2)

rawCourses.push\_back(inputLine)

courseVector.push\_back(newCourse)

FOR course in courseVector

IF course.prerequisites IS NOT EMPTY

Separate course.prerequisites at comma

IF ALL IN courseNumbers

PASS

ELSE

courseVector.remove(course)

CLOSE fileStream

**Evaluation:**

Reading the File Run Time Calculations

|  |  |  |  |
| --- | --- | --- | --- |
| Line | operation | Cost per execution | Times executed |
| 1 | Open file | 1 | 1 |
| 2 | Create temp vector | 1 | 1 |
| 3 | Create input string | 1 | 1 |
| 4 | Create course obj | 1 | 1 |
| 5 | While loop | 1 | n+1 |
| 6 | Get next line | 1 | n |
| 7 | Split line | 1 | n |
| 8 | Invalid input catch | 1 | <=n |
| 9 | Set temp courseID | 1 | <=n |
| 10 | Set temp courseName | 1 | <=n |
| 11 | Check for prereqs | 1 | <=n |
| 12 | Add to raw courses | 1 | <=n |
| 13 | Add to courseVector | 1 | <=n |
| 14 | Init for loop | 1 | 1 |
| 15 | Condition check | 1 | n\*k + n |
| 16 | Check if course has prerequisites | 1 | N\*k |
| 17 | Close filestream | 1 | 1 |

Assuming each course has at most k prerequisites. Most of the operations happen withing the while loop which executes once per line in the file. The total runtime of this operation would be 0(n).

**Advantages and Disadvantages:**

A vector data structure has the advantage of being simple and easy to use. That said, it is very slow at searching unless it is sorted which must be done manually. If the list of courses was small and did not need to be inserted into or deleted often, I would say a vector is a fine data structure choice. However, because of the needs stated by ABCU’s computer science department, I would not recommend a vector for this project.

The binary search tree brings with it some significant advantages over the vector. In this data structure, data is kept sorted in an alphanumeric way. It is significantly more complex and so has a larger memory impact with its pointers and nodes. If the tree becomes unbalanced, the binary search tree can also become just a glorified linked list.

Using a hash table to hold the course data brings some important advantages to note. Firstly, it is possible to achieve close to 0(1) or constant time with a search in a hash table assuming it has been crafted to avoid collisions. To properly handle collisions, the application should have a probing method that find the next available “bucket.” While data is stored in an efficient way, it is also not inherently ordered which means they cannot be easily printed in an alphanumeric way. Also similar to a binary search tree, they take up more memory with their buckets and points.

**My Recommendation:**

For ABCU’s purposes, I would recommend the hash table to store their course data. They should not need to print out the courses in an alphanumeric way, and they will be inserting and looking up often. While a binary tree could reliably operate at its average of 0(log n), the hash table with correct design work could achieve a 0(1) run time complexity. This means that the hash table’s searches would remain at a constant speed regardless of how many courses were contained within it.