# 6-2 Project One

# Andrew Piccirilli

# SNHU CS-300

Open file

Call to open file

If file not found return Error message and return value -1

Else File found

While not file end

Read every line

If line has less than 2 values return Error

Else read parameters

Parse parameters and store

Check prerequisite inline exists as course in file

loadCourses(csvPath, bst: BinarySearchTree):

file = readCSVFile(csvPath)

// Check file exists

If file doesn’t exist:

Display "File not found."

return

// Loop through each row in file

for i = 0 to file.rows() - 1:

// Create course object with ID and name from file

course = CreateCourse(file[i].getCourseId(), file[i].getCourseName())

// While not at end of the line in file

while not endOfLine(file[i]):

// Assign prerequisites to course object

course.prerequisites = file[i].getPrerequisites()

// Insert course into binary search tree

bst.Insert(course)

printCourseInformation(courses: Vector<Course>, courseId: String):

courseIdInput = GET\_INPUT("Enter Course ID: ")

// Loop until the vector is empty

While courses.isNotEmpty():

// Check courseIdInput matches current course's ID

If courseIdInput equals courses[0].courseId:

// Output course ID and name

Output courses[0].courseId, courses[0].courseName

// Remove first element from vector

courses.removeAtIndex(0)

// Loop until prereq is false

While prereq is true:

// Output prerequisites of course

Output courses[0].prerequisites

// Remove first element from vector

courses.removeAtIndex(0)

printCourseInformation(courses: HashTable<String, Course>, courseId: String):

courseIdInput = GET\_INPUT("Enter Course ID: ")

key = courseIdInput

// Check if key exists in hash table

If key exists in courses:

// Return course associated with key

return courses[key]

Else:

// Return null if key does not exist

Return null

printCourseInformation(courses: Tree<Course>, courseId: String):

courseIdInput = GET\_INPUT("Enter Course ID: ")

currentNode = courses.root

// Loop until currentNode is null

While currentNode not null:

// Check if courseIdInput matches current node's course ID

If courseIdInput equals currentNode.course.getCourseId():

// Output course ID and name

Output currentNode.course.getCourseId(), currentNode.course.getCourseName()

// Set currentNode to null to exit loop

currentNode = null

else if courseIdInput less than currentNode.course.getCourseId():

// Move to left child node

currentNode = currentNode.left

Else:

// Move to right child node

currentNode = currentNode.right

// Loop until prereq is false

While prereq is true:

// Output the prerequisites of course

Output currentNode.course.getPrerequisites()

// Set currentNode to null to exit loop

currentNode = null

Menu():

choice = 0

// Loop until choice is 4

While choice not equal to 4:

Display MenuOptions()

// Get user input for choice

choice = GET\_INPUT()

// Switch based on user choice

SWITCH choice:

CASE 1:

// Get file path and structure type

filePath = inputFilePath()

structureType = selectStructureType()

// Create selected data structure

selectedStructure = createStructure(structureType)

// Load courses into selected structure

loadCourses(filePath, selectedStructure)

CASE 2:

If selectedStructure is Vector:

If courses.isNotEmpty():

// Print sorted courses

printSorted(courses)

Else:

Display "Please load data first."

CASE 3:

If selectedStructure is not null:

courseId = GET\_INPUT("Enter Course ID: ")

// Print information for given course ID

printCourseInformation(selectedStructure, courseId)

else:

Display "Please load data first."

CASE 4:

Display "Exiting program."

|  |  |  |  |
| --- | --- | --- | --- |
| Vector | Line Cost | Times Executed | Total Cost |
| Read file | 1 | n | n |
| Create course object | 1 | n | n |
| Assign prerequisite | 1 | n | n |
| Insert into vector | 1 | n | n |
| Remove elements from vector | 1 | n | n |
| Create vector | 1 | 1 | 1 |
|  |  | Total | 5n + 1 |
|  |  | Runtime | O(n) |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Tree | Line Cost | Times Executed | Total Cost |
| Create Tree | 1 | 1 | 1 |
| Read file | 1 | n | n |
| Create course object | 1 | n | n |
| Assign prerequisites | 1 | n | n |
| Insert into tree | 1 | n | n |
|  |  | Total | 4n + 1 |
|  |  | runtime | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| Hash Table | Line cost | Times executed | Total cost |
| Create hash table | 1 | 1 | 1 |
| Read file | 1 | n | n |
| Create course object | 1 | n | n |
| Assign prerequisite | 1 | n | n |
| Insert into hashtable | 1 | n | n |
|  |  | Total | 4n + 1 |
|  |  | Runtime | O(n) |

Evaluation

Every data structure has advantages and disadvantages vectors are simple to implement, easy to use, uses iteration to read through files quickly, but they can be costly during resizing, insertions and deletions. Hash Tables can manage large databases easily, can search entire list relatively fast using a key, and are fast with insertions and deletions, but can be a bit more difficult to implement and use and take up more memory. If order traversal is required then hash tables aren’t the best fit due to being unordered. Trees are faster at searching than vectors, they have ordered data storage, and if balanced can improve their performance. The issues of using trees is that they can be complex to implement, take up a decent amount of memory, and have slower insertion/deletion.

If I had to recommend a data structure, as much as I am not a fan of implementing them, I would have to say trees. They can be fast to search since their ordered and can go through branches and leaves quickly, and while I already mentioned it being ordered for something such as required in this program would be a big help. If hash tables were capable of being ordered then I would say they would work great since searching using a key can be fast and insertions and deletions are relatively easy, and vectors are so easy to implement but with the fact they have to iterate and having difficulty with insertions and deletions kind of makes them less useful than trees, in my opinion, in this situation.