Joshua Williamson

August 7th, 2025

CS-300-12333-M01

**Vector Pseudocode**

**main()**

main Function()

Initialize vector<Course> courseVector

Initialize int userChoice = 0

Initialize string userInput = ""

**Call displayMenu()**

Assign userChoice with input from user

Loop while userChoice is not 9

If userChoice == 1

Assign filePath = getCSVPath()

Assign courseVector = parseCourseFile(filePath)

If userChoice == 2

Sort courseVector alphanumerically by courseNumber

Call printAllCourses(courseVector)

If userChoice == 3

Prompt user for course number

Assign input to userInput

Call searchCourse(courseVector, userInput)

Call displayMenu()

Assign userChoice with input from user

End

**getCSVPath()**

Prompt user for course file path

If input is empty

Return default path "ABCU\_CourseData.txt"

Else

Return user input

End

**parseCourseFile(filePath)**

Open file at filePath

If file fails to open

Output "File not found or unreadable."

Return empty vector

**Initialize vector<Course> courseVector**

WHILE file has lines

Read line

Split line by commas into tokens

If tokens.length < 2

Output "Invalid line format, skipping: " + line

Continue

Create new Course course

Set course.courseID = tokens[0]

Set course.courseName = tokens[1]

FOR i = 2 to tokens.length - 1

Add tokens[i] to course.preList

Set course.preCount = course.preList.size()

Add course to courseVector

Return courseVector

End

**searchCourse(courseVector, id)**

For each course in courseVector

If course.courseID == id

Call displayCourse(course)

Return

Output "Course not found."

End

**printAllCourses(courseVector)**

Sort courseVector by courseID

For each course in courseVector

Call displayCourse(course)

End

**displayCourse(course)**

Output "Course Number: " + course.courseID

Output "Course Title: " + course.courseName

If course.preCount == 0

Output "Prerequisites: None"

Else

Output "Prerequisites:"

For each prereq in course.preList

Output " - " + prereq

End

**Hash Table Pseudocode**

main()

**main Function()**

Initialize vector<LinkedList> courseLibrary of size tableSize

Initialize int userChoice = 0

Initialize string userInput = ""

**Call displayMenu()**

Assign userChoice with input from user

Loop while userChoice is not 9

If userChoice == 1

Assign filePath = getCSVPath()

Assign parsedRows = parseCourseFile(filePath)

For each row in parsedRows

Call addCourse(courseLibrary, row)

If userChoice == 2

Call printAllCourses(courseLibrary)

If userChoice == 3

Prompt user for course number

Assign userInput with input from user

Call displayCourse(searchCourse(courseLibrary, userInput))

Call displayMenu()

Assign userChoice with input from user

End

parseCourseFile(filePath)

Open file

Initialize vector<vector<string>> rows

WHILE file has lines

Split line by commas into row

If row.length < 2

Output "Invalid format. Skipping line."

Continue

Add row to rows

Return rows

End

addCourse(courseLibrary, row)

Create Course newCourse

Set newCourse.courseID = row[0]

Set newCourse.courseName = row[1]

FOR i = 2 to row.size() - 1

Add row[i] to newCourse.preList

Set newCourse.preCount = newCourse.preList.size()

key = hashFunction(newCourse.courseID) % tableSize

Add newCourse to courseLibrary[key]

End

**searchCourse(courseLibrary, id)**

For each LinkedList in courseLibrary

For each node in list

If node.courseID == id

Return node

Return empty Course

End

**printAllCourses(courseLibrary)**

For each LinkedList in courseLibrary

For each node in list

Call displayCourse(node)

End

**displayCourse(course)**

Output "Course Number: " + course.courseID

Output "Course Title: " + course.courseName

If course.preCount == 0

Output "Prerequisites: None"

Else

Output "Prerequisites:"

For each prereq in course.preList

Output " - " + prereq

End

**BinarySearchTree Pseudocode**

**main()**

main Function()

Initialize BinarySearchTree<Course> courseTree

Initialize vector<Course> parsedCourses

Call getCSVPath(), assign result to csvPath

Call parseCourseFile(csvPath), assign result to parsedCourses

If parsedCourses is empty

Output "No valid courses found. Exiting."

Exit program

For each course in parsedCourses

Insert course into courseTree

If NOT validatePrerequisites(parsedCourses)

Output "Invalid prerequisite detected. Exiting."

Exit program

Prompt user for course number to search

Assign input to userSearch

Call searchAndPrintCourse(courseTree.root, userSearch)

End

**getCSVPath()**

Prompt user for course file path

If input is empty

Return default path "ABCU\_CourseData.txt"

Else

Return user input

End

**parseCourseFile(filePath)**

Open file at filePath

If file fails to open

Output "File not found or unreadable."

Return empty vector

Initialize vector<Course> courseList

WHILE file has more lines

Read line

Split line by commas into tokens

If tokens.length < 2

Output "Invalid line format, skipping: " + line

Continue

Initialize Course newCourse

Set newCourse.courseID = tokens[0]

Set newCourse.courseName = tokens[1]

FOR i from 2 to tokens.length - 1

Add tokens[i] to newCourse.preList

Set newCourse.preCount = newCourse.preList.length

Add newCourse to courseList

Close file

Return courseList

End

**validatePrerequisites(courseList)**

Initialize set<string> validCourseIDs

For each course in courseList

Add course.courseID to validCourseIDs

For each course in courseList

For each prereq in course.preList

If prereq NOT in validCourseIDs

Output "Missing prerequisite: " + prereq + " for course " + course.courseID

Return false

Return true

End

**searchAndPrintCourse(node, targetID)**

If node is null

Output "Course not found."

Return

If targetID == node.course.courseID

Output "Course Number: " + node.course.courseID

Output "Course Title: " + node.course.courseName

If node.course.preCount == 0

Output "Prerequisites: None"

Else

Output "Prerequisites:"

For each prereq in node.course.preList

Output " - " + prereq

Return

If targetID < node.course.courseID

Call searchAndPrintCourse(node.left, targetID)

Else

Call searchAndPrintCourse(node.right, targetID)

End

**printAllCoursesInOrder(node)**

If node is not null

Call printAllCoursesInOrder(node.left)

Call displayCourse(node.course)

Call printAllCoursesInOrder(node.right)

End

**Run Time Analysis**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Vector** | **Hash Table** | **Binary Tree** |
| **Loading Data** | O(1) | O(1) – O(N)  *\*depends on if there are collisions* | O(log N) |
| **Search** | O(n) | O(1) – O(N)  *\*depends on if there are collisions* | O(log N) – O(N)  *\*depends on balance of the tree* |
| **Sort/Print** | O(N log N) *\*using quick sort* | O(N)  *\*assumes the table is created in order* | O(N)  *\*in order traversal* |

**Advantage Analysis**

Each data structure has trade-offs that would make it better suited in different use cases. Which structure depends on how often the data is loaded, searched, or printed. For example, if data is loaded once and searched frequently, a hash table may be better. If sorted output is required regularly, a Binary Search Tree is better suited. Vectors are useful for simple loading and occasional access where order is not critical.

* **Vector** can offer fast data loading with simple append operations. However, searching is linear time (O(N)), and sorting is costly at O(N log N). It is more ideal when fast loading is more important than fast lookup.
* **Hash Table** can provide the best performance for search and insert operations at O(1). However, collisions can degrade performance to O(N), and hash tables do not maintain order, making sorted output less efficient.
* **Binary Search Tree** maintains a sorted structure, which makes in-order traversals better for printing data. Search and insert operations are O(log N), but with inserting sorted data, it can become O(N).

**Recommendation**

**Binary Search Tree**

The Binary Search Tree offers the best overall performance for ABCU code. Hash table can provide a faster lookup, however, it cannot support sorted traversal without extra steps. Vectors, can suffer from slow searches and require additional sorting.

A Binary Search Tree can ensure consistent and efficient access patterns and meet all the software program's functional requirements.