Josh Hall

CS 300

Mod 6

Project 1

12/5/24

Vector Structure

Function LoadCoursesFromFile(filename)

Open file with name filename

If file cannot be opened

Print "Error: Cannot open file"

Return

End If

Initialize vector "courses"

Initialize set "courseNumbers"

While not end of file

Read line from file

Split line by commas into tokens

If tokens.size < 2

Print "Invalid line format"

Continue

Create course with tokens[0], tokens[1], and tokens[2:]

If valid course

Add course to courses

Add tokens[0] to courseNumbers

End While

Close file

Return courses

End Function

Struct Course

number

title

prerequisites (vector)

End Struct

Function CreateCourse(number, title, prerequisites, courseNumbers)

Initialize course

course.number = number

course.title = title

For each prerequisite in prerequisites

If prerequisite in courseNumbers

Add prerequisite to course.prerequisites

Else

Print "Prerequisite not found"

Return invalid course

End For

Return course

End Function

Function PrintCourse(courses, courseNumber)

For course in courses

If course.number == courseNumber

Print course.number, course.title, course.prerequisites

Return

Print "Course not found"

End Function

Function PrintAllCourses(courses)

Sort courses by course.number

For course in courses

Print course.number, course.title

End Function

Function ShowMenu()

While true

Print "1. Load Courses"

Print "2. Print All Courses"

Print "3. Print Course Information"

Print "9. Exit"

Input choice

Switch choice

Case 1:

courses = LoadCoursesFromFile("course\_data.txt")

Break

Case 2:

PrintAllCourses(courses)

Break

Case 3:

Print "Enter course number:"

Input courseNumber

PrintCourse(courses, courseNumber)

Break

Case 9:

Exit

Default:

Print "Invalid choice"

End Switch

End While

End Function

Hash table

Start

# Open and Read File

Open file "course\_data.txt" for reading

If file is not open

Output "Error: Unable to open file"

Exit program

End If

# Initialize a set to store all course numbers

Initialize set "courseNumbers"

# Parse Each Line and Validate Data

While not end of file

Read line from file into variable "line"

Split "line" by commas into list "tokens"

If number of tokens < 2

Output "Error: Invalid line format"

Continue to next line

End If

Set courseNumber to tokens[0]

Set courseTitle to tokens[1]

Set prerequisites to tokens[2:] (rest of tokens)

For each prerequisite in prerequisites

If prerequisite does not exist in courseNumbers

Output "Error: Prerequisite not found in course list"

End If

End For

Add courseNumber to courseNumbers

End While

Close file

# Create Course Objects and Store in Hash Table

Initialize hash table "courseTable" with default size

Open file "course\_data.txt" for reading

While not end of file

Read line from file into variable "line"

Split "line" by commas into list "tokens"

Set courseNumber to tokens[0]

Set courseTitle to tokens[1]

Set prerequisites to tokens[2:] (rest of tokens)

Create Course object "course"

Set course.number to courseNumber

Set course.title to courseTitle

Set course.prerequisites to prerequisites

Insert course into courseTable using courseNumber as key

End While

Close file

# Print Course Information and Prerequisites

Function PrintCourse(courseTable, courseNumber)

course = courseTable.Search(courseNumber)

If course is not found

Output "Course not found"

Return

End If

Output "Course Number: " + course.number

Output "Course Title: " + course.title

Output "Prerequisites: " + Join(course.prerequisites, ", ")

End Function

Function ShowMenu()

While true

Print "1. Load Courses"

Print "2. Print All Courses"

Print "3. Print Course Information"

Print "9. Exit"

Input choice

Switch choice

Case 1:

courses = LoadCoursesFromFile("course\_data.txt")

Break

Case 2:

PrintAllCourses(courses)

Break

Case 3:

Print "Enter course number:"

Input courseNumber

PrintCourse(courses, courseNumber)

Break

Case 9:

Exit

Default:

Print "Invalid choice. Please try again."

End Switch

End While

End Function

Binary Search Tree

Function LoadCoursesFromFile(filePath)

Open file at filePath for reading

If file is not open

Output "Error: Unable to open file"

Return

End If

Initialize set "courseNumbers"

While not end of file

Read line from file into variable "line"

Split "line" by commas into list "tokens"

If number of tokens < 2

Output "Error: Invalid line format"

Continue to next line

End If

Set courseNumber to tokens[0]

Set courseTitle to tokens[1]

Set prerequisites to tokens[2:] (rest of tokens)

For each prerequisite in prerequisites

If prerequisite does not exist in courseNumbers

Output "Error: Prerequisite not found in course list"

End If

End For

Add courseNumber to courseNumbers

End While

Close file

Initialize BinarySearchTree "courseTree"

Open file at filePath for reading

While not end of file

Read line from file into variable "line"

Split "line" by commas into list "tokens"

Set courseNumber to tokens[0]

Set courseTitle to tokens[1]

Set prerequisites to tokens[2:] (rest of tokens)

Create Course object "course"

Set course.number to courseNumber

Set course.title to courseTitle

Set course.prerequisites to prerequisites

Call courseTree.Insert(course)

End While

Close file

Return courseTree

End Function

Function PrintCourse(courseTree, courseNumber)

course = courseTree.Search(courseNumber)

If course is not found

Output "Course not found"

Return

End If

Output "Course Number: " + course.number

Output "Course Title: " + course.title

If course.prerequisites is not empty

Output "Prerequisites: " + Join(course.prerequisites, ", ")

Else

Output "Prerequisites: None"

End Function

Function PrintAllCourses(courseTree)

Call courseTree.InOrder() (This method will traverse and print all courses)

End Function

Function InOrderTraversal(node)

If node is not NULL

InOrderTraversal(node.left)

Output node.course.number, node.course.title, node.course.prerequisites

InOrderTraversal(node.right)

End Function

Function Insert(courseTree, course)

If courseTree is empty

Set courseTree.root to new Node(course)

Else

Set currentNode to courseTree.root

While true

If course.number < currentNode.course.number

If currentNode.left is null

Set currentNode.left to new Node(course)

Break

Else

Set currentNode to currentNode.left

Else

If currentNode.right is null

Set currentNode.right to new Node(course)

Break

Else

Set currentNode to currentNode.right

End While

End Function

Function ShowMenu()

While true

Output "1. Load Courses"

Output "2. Print All Courses"

Output "3. Print Course Information"

Output "9. Exit"

Input choice

Switch choice

Case 1:

courseTree = LoadCoursesFromFile("course\_data.txt")

Break

Case 2:

PrintAllCourses(courseTree)

Break

Case 3:

Output "Enter course number:"

Input courseNumber

PrintCourse(courseTree, courseNumber)

Break

Case 9:

Exit

Default:

Output "Invalid choice"

End Switch

End While

End Function

|  |  |  |  |
| --- | --- | --- | --- |
| Runtime Complexity  (Worst case) | | | |
| Operation | Vector | Hash Table | Binary Search Tree |
| Initialize data structure | O(1) | O(1) | O(1) |
| Open File | O(1) | O(1) | O(1) |
| Read line from file | O(n ) | O(n) | O(n) |
| Split line by commas | O(n) | O(n) | O(n) |
| Validate format | O(n) | O(n) | O(n) |
| Create course object | O(n) | O(n) | O(n) |
| Insert data into data structure | O(n) | O(n) | O(n) |
| Total runtime complexity | O(n) | O(n) | O(n) |
| Memory usage | O(n) | O(n) | O(n) |

Evaluation

The three data structures in consideration (Vector, Hash Table, and Binary search tree) all present unique benefits and challenges. All three algorithms have a worst-case scenario of O(n) but this does not tell the whole story. Vectors are simple and efficient for sequential access and adding new elements to the end of the vector. These are great properties, but it also means that it will not be ordered by default and will cause some performance loss when printing all courses alphanumerically. Vectors also struggle with the insertion of elements in the middle of the block. Hash tables are great for lookups and insertions with a best case of O(1) making it great for large data sets and repeated access jobs. Hash tables are very fast but can struggle with collisions that result in a worst-case of O(n), also it cannot be stored in a sorted form which means that any sorting would need to be done when the user asks for a sorted output or a second data structure would be needed to hold the sorted data. This would add to the already slightly complicated nature hash table. Binary search keep data sorted by default making printing in a certain order quite efficient also insertions and lookups can be as fast as O(log n). The big problem for that binary search tree is that It may become unbalanced as the data set grows. My recommendation is based on the idea that sorting and printing all courses will be infrequent, and that the most common occurrence will be look-ups given this basis I would recommend a hash table as it boasts the fastest possible performance and is the most scalable data structure being considered.