**Runtime Analysis**

**Function: lineParse / loadCourses**

|  |  |  |  |
| --- | --- | --- | --- |
| **Line of Code** | **Cost per Execution** | **Number of Executions** | **Total Cost** |
| Loop through file lines | 1 | n | O(n) |
| Tokenize line (split by commas) | 1 | n | O(n) |
| Create Course object | 1 | n | O(n) |
| Assign values to course fields | 1 | n | O(n) |
| Add course to data structure | f(n) | n | O(n·f(n)) |

Where f(n) is:

* Vector: O(1) average
* Hash Table: O(1) average
* BST: O(log n) average

**Total Runtime: O(n·f(n))**

**Function: listValidate / validateFile**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Data Structure | Outer Loop (n) | Inner Loop (avg m) | Check Existence | Total Cost |
| Vector | O(n) | O(m) | O(n) | O(n·m·n) = O(n^2) |
| Hash Table | O(n) | O(m) | O(1) | O(n·m) |
| BST | O(n) | O(m) | O(log n) | O(n·m·log n) |

**Evaluation of Data Structures**

**Vector**

* Insertion: O(1)
* Search: O(n)
* Memory: Contiguous, cache-friendly
* Pros: Simple to implement
* Cons: Inefficient for validation and search

**Hash Table**

* Insertion: O(1) average
* Search: O(1) average
* Memory: Higher due to hashing overhead
* Pros: Fastest validation/search
* Cons: No ordering, risk of collisions

**BST (Binary Search Tree)**

* Insertion: O(log n) average
* Search: O(log n) average
* Memory: Moderate (node-based)
* Pros: Maintains order
* Cons: Poor performance when unbalanced

**Recommendation**

**Hash Table Structure**

* Fastest for loading and validation
* Constant time search and insert operations
* Handles large datasets efficiently

While a BST offers ordered traversal, it is not needed for validation. A vector's O(n^2) behavior in validation makes it unsuitable.

|  |  |  |  |
| --- | --- | --- | --- |
| **Summary Table** |  |  |  |
| Operation | Vector | Hash Table | BST |
| Insert Course | O(1) | O(1) | O(log n) |
| Search Course | O(n) | O(1) | O(log n) |
| Validate Prerequisites | O(n^2) | O(n·m) | O(n·m·log n) |
| Memory Efficiency | High | Medium | Medium |
| Order Support | No | No | Yes |
| Overall Suitability | Poor | Best | Good |

**Vector Pseudocode**

// Define a Course Object

Class Course

String courseNum

String courseName

Vector prereqList

// Define Constructor

Course(number, name, prereqs)

// Define File Operations

FUNCTION lineParse(path)

CREATE vector named tempList

OPEN file found in path

LOOP through file

CREATE Course object named newCourse

READ line and SPLIT into tokens by comma

IF tokens contain at least 2 elements

SET newCourse.courseNum = first token

SET newCourse.courseName = second token

SET newCourse.prereqList = remaining tokens (if any)

ELSE

PRINT "Error. Invalid Format"

ADD newCourse to tempList

CLOSE file

RETURN tempList

FUNCTION listValidate(courseList)

CREATE Bool named valid = True

FOR course in courseList

IF course.prereqList ISNOT NULL

FOR prereq in course.prereqList

SET found = False

FOR courseCheck in courseList

IF courseCheck.courseNum == prereq

found = True

BREAK

IF found == False

PRINT "Error: Prerequisite", prereq, "not found."

valid = False

RETURN valid

// Search for specific course

FUNCTION searchCourse(input, courseList)

FOR course in courseList

IF course.courseNum == input

RETURN course

RETURN NULL // If not found

// Print Course

FUNCTION printCourse(courseList)

CREATE string input

PRINT "Enter course number: "

GET user input and store in input

SET course = searchCourse(input, courseList)

IF course IS NULL

PRINT "Error: Course not found."

RETURN

PRINT "Course Number:", course.courseNum

PRINT "Course Name:", course.courseName

IF course.prereqList.size() > 0

PRINT "Prerequisites:", JOIN(course.prereqList, ", ")

ELSE

PRINT "Prerequisites: None"

// Main Menu

FUNCTION Main()

CREATE vector named courseList

WHILE TRUE

PRINT "1. Load and Validate File"

PRINT "2. Search and Print Course"

PRINT "3. Exit"

PRINT "Enter choice:"

GET User input and store in choice

SWITCH choice

CASE 1:

PRINT "Enter file path: "

GET user input and store in path

SET courseList = lineParse(path)

IF listValidate(courseList) == False

PRINT "Error: Invalid file format."

CASE 2:

printCourse(courseList)

CASE 3:

PRINT "Exiting program."

RETURN

DEFAULT:

PRINT "Invalid option. Try again."

**Hash Table Pseudocode**

// Define a Course object

CLASS Course

STRING courseNum

STRING courseName

LIST<String> prereqList

CONSTRUCTOR Course(number, name, prereqs)

courseNum = number

courseName = name

prereqList = prereqs

// Define a hash table to store courses

DECLARE courseTable AS HashTable<String, Course>

// Load courses from file

FUNCTION loadCourses(path)

OPEN file at path

IF file is not open THEN

PRINT "Error: Cannot open file"

RETURN

DECLARE courseLines AS empty list

FOR EACH line IN file

REMOVE whitespace from line

IF line is empty THEN

CONTINUE

ADD line TO courseLines

CLOSE file

CALL validateFile(courseLines)

CALL storeCourses(courseLines)

// Validate prerequisites exist as courses

FUNCTION validateFile(courseLines)

DECLARE courseNumbers AS empty set

// First pass: collect all course numbers

FOR EACH line IN courseLines

SPLIT line by ',' INTO tokens

IF size of tokens < 2 THEN

PRINT "Error: Each line must contain at least a course number and a title."

EXIT program

ADD tokens[0] TO courseNumbers

// Second pass: validate prerequisites

FOR EACH line IN courseLines

SPLIT line by ',' INTO tokens

FOR i FROM 2 TO size of tokens - 1

SET prerequisite = tokens[i]

IF prerequisite NOT IN courseNumbers THEN

PRINT "Error: Prerequisite " + prerequisite + " not found as a course number."

EXIT program

// Store courses in hash table

FUNCTION storeCourses(courseLines)

FOR EACH line IN courseLines

SPLIT line by ',' INTO tokens

SET courseNumber = tokens[0]

SET courseTitle = tokens[1]

DECLARE prerequisites AS empty list

FOR i FROM 2 TO size of tokens - 1

ADD tokens[i] TO prerequisites

CREATE newCourse AS Course(courseNumber, courseTitle, prerequisites)

INSERT newCourse INTO courseTable WITH key = courseNumber

PRINT "Courses loaded and stored successfully"

RETURN courseTable

// Print a specific course

FUNCTION printCourse(courseTable, courseNumber)

SET course = courseTable.search(courseNumber)

IF course IS NULL THEN

PRINT "Course not found."

RETURN

PRINT course.courseNum + ": " + course.courseName

IF course.prereqList IS empty THEN

PRINT "Prerequisites: None"

ELSE

PRINT "Prerequisites:"

FOR EACH prereq IN course.prereqList

PRINT prereq

// Print all courses

FUNCTION printAllCourses(courseTable)

SET allCourses = courseTable.getAll()

FOR EACH course IN allCourses

PRINT course.courseNum + ": " + course.courseName

// Main menu

FUNCTION Main()

DECLARE courseTable AS new HashTable<String, Course>

WHILE TRUE

PRINT "1. Load and Validate File"

PRINT "2. Search and Print Course"

PRINT "3. Print All Courses"

PRINT "4. Exit"

PRINT "Enter choice:"

GET user input INTO choice

SWITCH choice

CASE 1:

PRINT "Enter file path: "

GET user input INTO path

CALL loadCourses(path)

CASE 2:

PRINT "Please enter course number:"

GET user input INTO courseNumber

CALL printCourse(courseTable, courseNumber)

CASE 3:

CALL printAllCourses(courseTable)

CASE 4:

PRINT "Exiting program."

RETURN

DEFAULT:

PRINT "Invalid option. Try again."

**Binary Search Tree Pseudocode**

// Define a Course object

CLASS Course

STRING courseNum

STRING courseName

LIST<String> prereqList

CONSTRUCTOR Course(number, name, prereqs)

courseNum = number

courseName = name

prereqList = prereqs

// Define a binary search tree to store courses

DECLARE courseTree AS BinarySearchTree<String, Course>

// Load courses from file

FUNCTION loadCourses(path)

OPEN file at path

IF file is not open THEN

PRINT "Error: Cannot open file"

RETURN

DECLARE courseLines AS empty list

FOR EACH line IN file

REMOVE whitespace from line

IF line is empty THEN

CONTINUE

ADD line TO courseLines

CLOSE file

CALL validateFile(courseLines)

CALL storeCourses(courseLines)

// Validate prerequisites exist as courses

FUNCTION validateFile(courseLines)

DECLARE courseNumbers AS empty set

// First pass: collect all course numbers

FOR EACH line IN courseLines

SPLIT line by ',' INTO tokens

IF size of tokens < 2 THEN

PRINT "Error: Each line must contain at least a course number and a title."

EXIT program

ADD tokens[0] TO courseNumbers

// Second pass: validate prerequisites

FOR EACH line IN courseLines

SPLIT line by ',' INTO tokens

FOR i FROM 2 TO size of tokens - 1

SET prerequisite = tokens[i]

IF prerequisite NOT IN courseNumbers THEN

PRINT "Error: Prerequisite " + prerequisite + " not found as a course number."

EXIT program

// Store courses in BST

FUNCTION storeCourses(courseLines)

FOR EACH line IN courseLines

SPLIT line by ',' INTO tokens

SET courseNumber = tokens[0]

SET courseTitle = tokens[1]

DECLARE prerequisites AS empty list

FOR i FROM 2 TO size of tokens - 1

ADD tokens[i] TO prerequisites

CREATE newCourse AS Course(courseNumber, courseTitle, prerequisites)

INSERT newCourse INTO courseTree WITH key = courseNumber

PRINT "Courses loaded and stored successfully"

RETURN courseTree

// Print a specific course

FUNCTION printCourse(courseTree , courseNumber)

SET course = courseTree .search(courseNumber)

IF course IS NULL THEN

PRINT "Course not found."

RETURN

PRINT course.courseNum + ": " + course.courseName

IF course.prereqList IS empty THEN

PRINT "Prerequisites: None"

ELSE

PRINT "Prerequisites:"

FOR EACH prereq IN course.prereqList

PRINT prereq

// Print all courses

FUNCTION printAllCourses(courseTree )

CALL inOrderPrint(courseTree.root)

FOR EACH course IN allCourses

PRINT course.courseNum + ": " + course.courseName

//Recursively print tree in order

FUNCTION inOrderPrint(node.left)

IF node IS NULL

RETURN

CALL inOrderPrint(node)

SET course = node.value

PRINT course.courseNum + “: “ + course.courseName

CALL inOrderPrint(node.right)

// Main menu

FUNCTION Main()

DECLARE courseTree AS new BinarySearchTree<String, Course>

WHILE TRUE

PRINT "1. Load and Validate File"

PRINT "2. Search and Print Course"

PRINT "3. Print All Courses"

PRINT "4. Exit"

PRINT "Enter choice:"

GET user input INTO choice

SWITCH choice

CASE 1:

PRINT "Enter file path: "

GET user input INTO path

SET courseTree = loadCourses(path)

CASE 2:

PRINT "Please enter course number:"

GET user input INTO courseNumber

CALL printCourse(courseTree , courseNumber)

CASE 3:

CALL printAllCourses(courseTree )

CASE 4:

PRINT "Exiting program."

RETURN

DEFAULT:

PRINT "Invalid option. Try again."