Zach DeLong

SNHU

13OCT2024

Professor Lusby

**Pseudocode**

**Load Courses into Vector**

FUNCTION loadCoursesVector(filePath)

DECLARE Vector<Course> courseList

OPEN file at filePath

IF file cannot be opened THEN

PRINT "Error: Cannot open file"

RETURN courseList

END IF

FOR each line in file

SPLIT line by commas into tokens

IF line has fewer than 2 elements THEN

PRINT "Error: Invalid line format"

CONTINUE

END IF

DECLARE Course course

course.courseNumber = tokens[0]

course.courseTitle = tokens[1]

FOR each token from tokens[2] to end

ADD token to course.prerequisites

END FOR

ADD course to courseList

END FOR

CLOSE file

RETURN courseList

END FUNCTION

**Search and Print Course in Vector**

FUNCTION searchCourseVector(courseList, courseNumber)

FOR each course IN courseList

IF course.courseNumber == courseNumber THEN

PRINT course details

RETURN

END IF

PRINT "Course not found"

END FUNCTION

**Load Courses into Hash Table**

FUNCTION loadCoursesHashTable(filePath)

DECLARE HashTable<Course> courseTable

OPEN file at filePath

IF file cannot be opened THEN

PRINT "Error: Cannot open file"

RETURN courseTable

END IF

FOR each line in file

SPLIT line by commas into tokens

IF line has fewer than 2 elements THEN

PRINT "Error: Invalid line format"

CONTINUE

END IF

DECLARE Course course

course.courseNumber = tokens[0]

course.courseTitle = tokens[1]

FOR each token from tokens[2] to end

ADD token to course.prerequisites

END FOR

INSERT course into courseTable using courseNumber as key

END FOR

CLOSE file

RETURN courseTable

END FUNCTION

**Search and Print Course in Hash Table**

FUNCTION searchCourseHashTable(courseTable, courseNumber)

course = FIND courseNumber IN courseTable

IF course is not NULL THEN

PRINT course details

ELSE

PRINT "Course not found"

END IF

END FUNCTION

**Load Courses into BST**

FUNCTION loadCoursesBST(filePath)

DECLARE BST<Course> courseTree

OPEN file at filePath

IF file cannot be opened THEN

PRINT "Error: Cannot open file"

RETURN courseTree

END IF

FOR each line in file

SPLIT line by commas into tokens

IF line has fewer than 2 elements THEN

PRINT "Error: Invalid line format"

CONTINUE

END IF

DECLARE Course course

course.courseNumber = tokens[0]

course.courseTitle = tokens[1]

FOR each token from tokens[2] to end

ADD token to course.prerequisites

END FOR

INSERT course INTO courseTree using courseNumber as key

END FOR

CLOSE file

RETURN courseTree

END FUNCTION

**Search and Print Course in BST**

FUNCTION searchCourseBST(courseTree, courseNumber)

course = SEARCH courseNumber IN courseTree

IF course is not NULL THEN

PRINT course details

ELSE

PRINT "Course not found"

END IF

END FUNCTION

**All three data structures will need the same menu pseudocode**

FUNCTION displayMenu()

PRINT "1. Load Courses"

PRINT "2. Print all Courses"

PRINT "3. Search for a Course"

PRINT "9. Exit"

READ user input

IF user input == 1 THEN

CALL loadCourses(filePath)

ELSE IF user input == 2 THEN

CALL printAllCourses(dataStructure)

ELSE IF user input == 3 THEN

CALL searchCourse(dataStructure, courseNumber)

ELSE IF user input == 9 THEN

EXIT program

END IF

END FUNCTION

Runtime Analysis:

* Vector:
  1. Load: O(n) to insert n courses, where n is the number of courses.
  2. Search: O(n) for searching a course in an unsorted vector.
  3. Insert: O(1) for appending, O(n) for insertion if maintaining an order.
* Hash Table:
  1. Load: O(n) to insert n courses, assuming minimal collisions.
  2. Search: O(1) on average but O(n) in the worst case with collisions.
  3. Insert: O(1) on average but O(n) in the worst case.
* BST:
  1. Load: O(n log n) for balanced insertions, O(n^2) if unbalanced.
  2. Search: O(log n) for a balanced tree, O(n) for an unbalanced tree.
  3. Insert: O(log n) for a balanced tree, O(n) for an unbalanced tree.

**Evaluation and Recommendation:**

Considering the runtime analysis, I recommend using the hash table due to its average-case O(1) search and insert performance. Although vectors are simpler to implement, their O(n) search time makes them less efficient. Binary search trees are a good alternative, especially if they remain balanced, but the hash table provides better overall performance for this scenario due to its efficiency in searching and inserting.