Hiep Ha

CS-300

Professor Nathan Lebel

Oct. 13, 2024

6-2 Project One

**1. Vector Data Structure Pseudocode**

**Opening File and Parsing Data:**

FUNCTION loadCoursesFromFile(fileName)

OPEN file fileName FOR reading

IF file is not open THEN

PRINT "Error: Unable to open file"

RETURN

WHILE there are lines in the file DO

line = READ line from file

tokens = PARSE line into tokens using comma as delimiter

IF size of tokens < 2 THEN

PRINT "Error: Invalid course entry - less than two parameters"

CONTINUE

Course newCourse

newCourse.courseNumber = tokens[0]

newCourse.title = tokens[1]

newCourse.prerequisites = EMPTY LIST

FOR i FROM 2 TO size of tokens - 1 DO

ADD tokens[i] to newCourse.prerequisites

END FOR

ADD newCourse to courses

END WHILE

CLOSE file

END FUNCTION

**Print All Courses in Alphanumeric Order:**

FUNCTION printAllCourses()

SORT courses by courseNumber

FOR EACH course IN courses DO

PRINT course.courseNumber + ": " + course.title

END FOR

END FUNCTION

**2. Hash Table Data Structure Pseudocode**

**Opening File and Parsing Data:**

FUNCTION loadCoursesFromFile(fileName)

OPEN file fileName FOR reading

IF file is not open THEN

PRINT "Error: Unable to open file"

RETURN

WHILE there are lines in the file DO

line = READ line from file

tokens = PARSE line into tokens using comma as delimiter

IF size of tokens < 2 THEN

PRINT "Error: Invalid course entry"

CONTINUE

CALL createCourse(tokens[0], tokens[1], SUBLIST(tokens, 2))

END WHILE

CLOSE file

END FUNCTION

**Print All Courses:**

FUNCTION printAllCourses()

FOR EACH bucket in HashTable

FOR EACH course IN bucket DO

PRINT course.courseNumber + ": " + course.title

END FOR

END FUNCTION

**3. Tree Data Structure Pseudocode**

**Opening File and Parsing Data:**

FUNCTION loadCoursesFromFile(fileName)

OPEN file fileName FOR reading

IF file is not open THEN

PRINT "Error: Unable to open file"

RETURN

WHILE there are lines in the file DO

line = READ line from file

tokens = SPLIT line by commas

CALL insertIntoTree(tokens[0], tokens[1], SUBLIST(tokens, 2))

END WHILE

CLOSE file

END FUNCTION

**Print All Courses in Alphanumeric Order:**

FUNCTION printAllCourses(tree)

IF tree is empty THEN

RETURN

CALL printAllCourses(tree.left)

PRINT tree.courseNumber + ": " + tree.title

CALL printAllCourses(tree.right)

END FUNCTION

**4. Menu Pseudocode**

FUNCTION displayMenu()

PRINT "1. Load courses"

PRINT "2. Print all courses (alphanumeric order)"

PRINT "3. Print course information and prerequisites"

PRINT "9. Exit"

INPUT userChoice

IF userChoice == 1 THEN

CALL loadCoursesFromFile("courses.txt")

ELSE IF userChoice == 2 THEN

CALL printAllCourses()

ELSE IF userChoice == 3 THEN

INPUT courseNumber

CALL printCourseInfo(courseNumber)

ELSE IF userChoice == 9 THEN

RETURN

ELSE

PRINT "Invalid option, try again."

END FUNCTION

**5. Big O Runtime Analysis:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Data Structure** | **Insert** | **Search** | **Print All Courses** | **Advantages** | **Disadvantages** |
| **Vector** | O(n) | O(n) | O(n log n) (after sort) | Simple to implement, dynamic sizing | Insertion and search are slower as n grows |
| **Hash Table** | O(1) (avg case) | O(1) (avg case) | O(n) | Fast average insert/search, handles large datasets | Collisions can degrade performance, requires hashing |
| **Tree (BST)** | O(log n) (avg case) | O(log n) (avg case) | O(n) (in-order print) | Efficient sorted data access, balanced search | Needs balancing for efficiency, worst case O(n) |

**6. Evaluation:**

1. **Vector**:

* **Advantages**: Simple to implement, dynamic in size.
* **Disadvantages**: Slower insertion and search as the number of courses increases, and sorting is required for alphanumeric ordering.
* **Best Use**: Suitable when the dataset is small, and sorting is needed before searching.

1. **Hash Table**:

* **Advantages**: Fast average case insertion and search (constant time), efficient for large datasets.
* **Disadvantages**: Potential collisions that degrade performance; it does not maintain the order of elements.
* **Best Use**: Ideal for fast lookup and insertion, but less suitable when maintaining order is necessary.

1. **Binary Search Tree (BST)**:

* **Advantages**: Supports fast lookups (logarithmic time) and maintains elements in sorted order, making it suitable for tasks like printing courses alphanumerically.
* **Disadvantages**: Can become unbalanced, leading to slower operations (worst case O(n)).
* **Best Use**: Best when both fast lookups and sorted data are required.

**7. Recommendation:**

For this project, the **Binary Search Tree (BST)** is the best choice because it allows for efficient searching and maintains courses in alphanumeric order for easy printing. While a **Hash Table** offers fast lookup times, it does not maintain order, which is a key requirement for this project.