Joshua Hale

August 10, 2024

**CS300**

6-2 Submit Project One

**Pseudocode for ABC University Course Information Program**

**Vector Data Structure Pseudocode**

1. **Opening the File and Reading Data**

BEGIN

OPEN file "course\_data.txt"

IF file is not found THEN

PRINT "Error: File not found"

RETURN

END IF

CREATE vector<Course> courses

WHILE (not end of file)

READ line from file

SPLIT line by comma into tokens

IF (number of tokens < 2) THEN

PRINT "Error: Invalid line format"

CONTINUE

END IF

SET courseNumber = tokens[0]

SET courseName = tokens[1]

CREATE vector<String> prerequisites

FOR i = 2 to (number of tokens - 1)

ADD tokens[i] to prerequisites

END FOR

CREATE Course object with courseNumber, courseName, prerequisites

ADD Course object to courses vector

END WHILE

CLOSE file

END

1. **Validating the File Format**

BEGIN

FUNCTION validateFile(Vector<Course> courses)

CREATE set<String> courseNumbers

FOR each course in courses

ADD course.courseNumber to courseNumbers

END FOR

FOR each course in courses

FOR each prerequisite in course.prerequisites

IF prerequisite NOT IN courseNumbers THEN

PRINT "Error: Prerequisite " + prerequisite + " does not exist"

RETURN FALSE

END IF

END FOR

END FOR

RETURN TRUE

END FUNCTION

END

1. **Creating Course Objects and Storing Them**

BEGIN

FUNCTION createCourse(String courseNumber, String courseName, Vector<String> prerequisites)

CREATE Course object

SET object.courseNumber = courseNumber

SET object.courseName = courseName

SET object.prerequisites = prerequisites

RETURN object

END FUNCTION

FOR each line in file

SPLIT line by comma into tokens

SET courseNumber = tokens[0]

SET courseName = tokens[1]

CREATE vector<String> prerequisites

FOR i = 2 to (number of tokens - 1)

ADD tokens[i] to prerequisites

END FOR

CREATE course object using createCourse function

ADD course object to courses vector

END FOR

END

1. **Searching for a Specific Course and Printing Information**

BEGIN

FUNCTION searchCourse(Vector<Course> courses, String courseNumber)

FOR each course in courses

IF course.courseNumber == courseNumber THEN

PRINT "Course Number: " + course.courseNumber

PRINT "Course Name: " + course.courseName

PRINT "Prerequisites: "

IF course.prerequisites.size() == 0 THEN

PRINT "None"

ELSE

FOR each prerequisite in course.prerequisites

PRINT prerequisite

END FOR

END IF

RETURN

END IF

END FOR

PRINT "Course not found"

END FUNCTION

END

1. **Pseudocode for the Menu**

BEGIN

WHILE true

PRINT "Menu:"

PRINT "1. Load Data"

PRINT "2. Print All Courses"

PRINT "3. Print Course Information"

PRINT "9. Exit"

INPUT choice

IF choice == 1 THEN

CALL loadData()

ELSE IF choice == 2 THEN

CALL printAllCourses()

ELSE IF choice == 3 THEN

INPUT courseNumber

CALL searchCourse(courses, courseNumber)

ELSE IF choice == 9 THEN

EXIT

ELSE

PRINT "Invalid choice. Try again."

END IF

END WHILE

END

1. **Printing All Courses in Alphanumeric Order**

BEGIN

FUNCTION printAllCourses(Vector<Course> courses)

SORT courses by courseNumber

FOR each course in courses

PRINT course.courseNumber + ": " + course.courseName

END FOR

END FUNCTION

END

**Hash Table Data Structure Pseudocode**

1. **Opening the File and Reading Data**

BEGIN

OPEN file "course\_data.txt"

IF file is not found THEN

PRINT "Error: File not found"

RETURN

END IF

CREATE HashTable<Course> courses

WHILE (not end of file)

READ line from file

SPLIT line by comma into tokens

IF (number of tokens < 2) THEN

PRINT "Error: Invalid line format"

CONTINUE

END IF

SET courseNumber = tokens[0]

SET courseName = tokens[1]

CREATE List<String> prerequisites

FOR i = 2 to (number of tokens - 1)

ADD tokens[i] to prerequisites

END FOR

CREATE Course object with courseNumber, courseName, prerequisites

INSERT courseNumber, Course object into courses hash table

END WHILE

CLOSE file

END

1. **Validating the File Format**

BEGIN

FUNCTION validateFile(HashTable<Course> courses)

CREATE set<String> courseNumbers

FOR each course in courses

ADD course.courseNumber to courseNumbers

END FOR

FOR each course in courses

FOR each prerequisite in course.prerequisites

IF prerequisite NOT IN courseNumbers THEN

PRINT "Error: Prerequisite " + prerequisite + " does not exist"

RETURN FALSE

END IF

END FOR

END FOR

RETURN TRUE

END FUNCTION

END

1. **Creating Course Objects and Storing Them**

BEGIN

FUNCTION createCourse(String courseNumber, String courseName, List<String> prerequisites)

CREATE Course object

SET object.courseNumber = courseNumber

SET object.courseName = courseName

SET object.prerequisites = prerequisites

RETURN object

END FUNCTION

FOR each line in file

SPLIT line by comma into tokens

SET courseNumber = tokens[0]

SET courseName = tokens[1]

CREATE List<String> prerequisites

FOR i = 2 to (number of tokens - 1)

ADD tokens[i] to prerequisites

END FOR

CREATE course object using createCourse function

INSERT courseNumber, course object into courses hash table

END FOR

END

1. **Searching for a Specific Course and Printing Information**

BEGIN

FUNCTION searchCourse(HashTable<Course> courses, String courseNumber)

Course course = GET courseNumber from courses hash table

IF course is not found THEN

PRINT "Course not found"

RETURN

END IF

PRINT "Course Number: " + course.courseNumber

PRINT "Course Name: " + course.courseName

PRINT "Prerequisites: "

IF course.prerequisites.size() == 0 THEN

PRINT "None"

ELSE

FOR each prerequisite in course.prerequisites

PRINT prerequisite

END FOR

END IF

END FUNCTION

END

1. **Pseudocode for the Menu**

BEGIN

WHILE true

PRINT "Menu:"

PRINT "1. Load Data"

PRINT "2. Print All Courses"

PRINT "3. Print Course Information"

PRINT "9. Exit"

INPUT choice

IF choice == 1 THEN

CALL loadData()

ELSE IF choice == 2 THEN

CALL printAllCourses()

ELSE IF choice == 3 THEN

INPUT courseNumber

CALL searchCourse(courses, courseNumber)

ELSE IF choice == 9 THEN

EXIT

ELSE

PRINT "Invalid choice. Try again."

END IF

END WHILE

END

1. **Printing All Courses in Alphanumeric Order**

BEGIN

FUNCTION printAllCourses(HashTable<Course> courses)

CREATE List<Course> sortedCourses

FOR each course in courses

ADD course to sortedCourses

END FOR

SORT sortedCourses by courseNumber

FOR each course in sortedCourses

PRINT course.courseNumber + ": " + course.courseName

END FOR

END FUNCTION

END

**Binary Search Tree Data Structure Pseudocode**

1. **Opening the File and Reading Data**

BEGIN

OPEN file "course\_data.txt"

IF file is not found THEN

PRINT "Error: File not found"

RETURN

END IF

CREATE BinarySearchTree<Course> courses

WHILE (not end of file)

READ line from file

SPLIT line by comma into tokens

IF (number of tokens < 2) THEN

PRINT "Error: Invalid line format"

CONTINUE

END IF

SET courseNumber = tokens[0]

SET courseName = tokens[1]

CREATE List<String> prerequisites

FOR i = 2 to (number of tokens - 1)

ADD tokens[i] to prerequisites

END FOR

CREATE Course object with courseNumber, courseName, prerequisites

INSERT course object into courses binary search tree

END WHILE

CLOSE file

END

1. **Validating the File Format**

BEGIN

FUNCTION validateFile(BinarySearchTree<Course> courses)

CREATE set<String> courseNumbers

FOR each course in courses

ADD course.courseNumber to courseNumbers

END FOR

FOR each course in courses

FOR each prerequisite in course.prerequisites

IF prerequisite NOT IN courseNumbers THEN

PRINT "Error: Prerequisite " + prerequisite + " does not exist"

RETURN FALSE

END IF

END FOR

END FOR

RETURN TRUE

END FUNCTION

END

1. **Creating Course Objects and Storing Them**

BEGIN

FUNCTION createCourse(String courseNumber, String courseName, List<String> prerequisites)

CREATE Course object

SET object.courseNumber = courseNumber

SET object.courseName = courseName

SET object.prerequisites = prerequisites

RETURN object

END FUNCTION

FOR each line in file

SPLIT line by comma into tokens

SET courseNumber = tokens[0]

SET courseName = tokens[1]

CREATE List<String> prerequisites

FOR i = 2 to (number of tokens - 1)

ADD tokens[i] to prerequisites

END FOR

CREATE course object using createCourse function

INSERT course object into courses binary search tree

END FOR

END

1. **Searching for a Specific Course and Printing Information**

BEGIN

FUNCTION searchCourse(BinarySearchTree<Course> courses, String courseNumber)

Course course = FIND courseNumber in courses binary search tree

IF course is not found THEN

PRINT "Course not found"

RETURN

END IF

PRINT "Course Number: " + course.courseNumber

PRINT "Course Name: " + course.courseName

PRINT "Prerequisites: "

IF course.prerequisites.size() == 0 THEN

PRINT "None"

ELSE

FOR each prerequisite in course.prerequisites

PRINT prerequisite

END FOR

END IF

END FUNCTION

END

1. **Pseudocode for the Menu**

BEGIN

WHILE true

PRINT "Menu:"

PRINT "1. Load Data"

PRINT "2. Print All Courses"

PRINT "3. Print Course Information"

PRINT "9. Exit"

INPUT choice

IF choice == 1 THEN

CALL loadData()

ELSE IF choice == 2 THEN

CALL printAllCourses()

ELSE IF choice == 3 THEN

INPUT courseNumber

CALL searchCourse(courses, courseNumber)

ELSE IF choice == 9 THEN

EXIT

ELSE

PRINT "Invalid choice. Try again."

END IF

END WHILE

END

1. **Printing All Courses in Alphanumeric Order**

BEGIN

FUNCTION printAllCourses(BinarySearchTree<Course> courses)

CALL inOrderTraversal(courses.root)

FUNCTION inOrderTraversal(TreeNode node)

IF node is not null THEN

CALL inOrderTraversal(node.left)

PRINT node.course.courseNumber + ": " + node.course.courseName

CALL inOrderTraversal(node.right)

END IF

END FUNCTION

END

**Runtime Analysis**

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Vector | Hash Table | Binary Search Tree |
| File Reading | O(n) | O(n) | O(n) |
| Insertion | O(1) | O(1) | O(log n) |
| Search | O(n) | O(1) | O(log n) |
| Printing Alphanumeric | O(n log n) | O(n log n) | O(n) |
| Memory Usage | High | Medium | Medium |
| Initialization | Fast | Fast | Slow |

**Advantages and Disadvantages**

**Vector:**

* **Advantages:** Simple to implement, fast access and insertion for small datasets.
* **Disadvantages:** Slow search for large datasets, high memory usage for large datasets.

**Hash Table:**

* **Advantages:** Fast search, insertion, and deletion. Efficient for large datasets.
* **Disadvantages:** Complexity in handling collisions, memory overhead.

**Binary Search Tree:**

* **Advantages:** Efficient search, insertion, and deletion for balanced trees. Memory efficient.
* **Disadvantages:** Slower initialization can degrade to O(n) for unbalanced trees.

**Recommendation**

For this project, I recommend using the **Hash Table** data structure. The hash table provides the best performance for the search operation, which is crucial for retrieving course information quickly. Additionally, it handles large datasets efficiently with constant time complexity for insertion, deletion, and search operations. The memory overhead and complexity in handling collisions are acceptable trade-offs for the performance benefits.

**Improvement: Error Handling**  
Details: Add more comprehensive error handling. Handle cases where the file might be corrupt or lines are formatted incorrectly more gracefully.

**Improvement: Optimization**  
Details: Simplify and optimize the logic where possible, using built-in functions or more efficient loops to improve clarity and performance.

**Improvement: Commenting**  
Details: Add detailed comments to explain each step in the pseudocode, which will help demonstrate understanding and attention to detail.

**Improvement: Input Validation**  
Details: Include input validation for menu choices to handle unexpected or incorrect inputs more robustly.

**Improvement: Modularization**  
Details: Break down menu handling into separate functions for each action to improve readability and maintainability.

**Improvement: Sorting Optimization**  
Details: Optimize the sort function to demonstrate an understanding of more advanced sorting algorithms, if applicable.

**Improvement: Error Handling**  
Details: Ensure the function handles edge cases, such as an empty course list, gracefully.

**Improvement: Detailed Analysis**  
Details: Expand the runtime analysis to include edge cases, such as worst-case scenarios and large datasets. Explain the implications of these cases on the runtime.

**Improvement: Comparative Analysis**  
Details: Provide a more in-depth comparison between the data structures under different conditions (e.g., when the data set is small vs. large, or when operations are predominantly search-based vs. insertion-based).

**Improvement: Scenario-based Discussion**  
Details: Provide more examples and scenarios where each advantage or disadvantage would be particularly impactful.

**Improvement: Graphical Representation**  
Details: Consider adding a simple graph or table that visualizes the trade-offs between the data structures based on different criteria (e.g., speed vs. memory usage).

**Improvement: Risk Mitigation**  
Details: Discuss potential challenges with the Hash Table, such as collision handling, and how you plan to mitigate these risks.

**Improvement: Future Considerations**  
Details: Mention any future scalability concerns and how the chosen data structure could be adapted or optimized in the future.

**Improvement: Proofreading**  
Details: Perform a thorough proofreading to eliminate any grammatical errors or awkward phrasing.

**Improvement: Clarity and Precision**  
Details: Ensure that all sentences are clear and precise, using concise language to convey complex ideas and avoid any ambiguity.