Ricky Suarez

CS-300

06/14/2025

Project 1: ABCU Advising Program Design

Pseudocode

```
Main() – Program Menu
If command line argument exists:
   Set csvPath to argument
Else:
   Set csvPath to default file location

While menuChoice != 9:
   Display menu:
      1. Load data
      2. Validate data
      3. Find course
      4. Print all courses
      9. Exit
   Get user input → menuChoice
   Get user input → dataChoice (vector, hash table, tree)

   If menuChoice == 1:
      If dataChoice == vector:
         Call loadBidsVector(csvPath)
      Else if dataChoice == hashTable:
         Call loadBidsHash(csvPath)
      Else if dataChoice == tree:
         Call loadBidsTree(csvPath)

   Else if menuChoice == 2:
      If vector: Call validateVector()
      If hashTable: Call validateHashTable()
      If tree: Call validateTree()

   Else if menuChoice == 3:
      Prompt for courseID → userSearch
      If vector: Call printCourseVector(userSearch)
      If hashTable: Call printCourseHash(userSearch)
```

```
    If tree: Call printCourseTree(userSearch)

  Else if menuChoice == 4:
    If vector: Call sortVector(); printVector()
    If hashTable: Call sortHash(); printHash()
    If tree: Call printTree()

  Else if menuChoice == 9:
    Output "Goodbye"
```

Course Structure

```
Struct Course:
  courseID
  courseName
  preCount
  preList (list of prerequisites)

Constructor:
  courseID = ""
  courseName = ""
  preCount = 0
  preList = []
```

File Parsing Logic (Used by All)

```
Function loadBidsX(csvPath):
  Open file at csvPath
  For each row:
    If formatting error, skip
    Parse courseID, courseName, and all prerequisites
    Create new Course object with parsed data
    Insert into data structure
```

Data Structure Implementations

```
Vector:
List<Course> courseList
Function printVector():
  For course in courseList:
    Output courseID, courseName
    For each prereq in course.preList:
```

```
        Output prereq


Hash Table:
List<Node*> hashTable
Function hash(courseID):
   Return int key based on hash logic
Function printHash():
   For each node in hashTable:
      If node is not null:
         Output courseID, courseName, prerequisites


Binary Search Tree:
Class Node:
   Course course
   Node* left
   Node* right
Function printTree(Node* node):
   If node is null: return
   printTree(node.left)
   Output node.course data
   printTree(node.right)
```

Runtime Analysis Chart

Operation	Vector	Hash Table	Binary Search Tree
Load Data	O(1)	O(1) – O(n)	O(log n) – O(n)
Search Course	O(n)	O(1) – O(n)	O(log n) – O(n)
Sort/Print	O(n log n)	O(n)*	O(n)

Data Structure Evaluation

- Vector: Fast to load and simple to implement. Poor search performance unless sorted. Sorting adds O(n log n) overhead.
- Hash Table: Offers excellent average-case search speed with $\Theta(1)$, but collisions can degrade performance to O(n). Depends heavily on a good hash function and enough buckets.
- Binary Tree: Performs well with balanced input (O(log n)), but unbalanced trees degrade to O(n). Good for ordered printing without extra sorting.

Recommendation

Based on the expectation that advisors will load data infrequently but search courses frequently, the hash table is the most efficient option. With an optimized hash function and sufficiently sized table, it allows for fast search and acceptable memory usage. The vector and tree structures, while useful, either take longer to search or are more sensitive to input order. Therefore, I recommend using the hash table for the final coded implementation.