Euntak Jang

CS 300

Dr. Webb

6-2: Project One

**Vector Data Structure Pseudocode**

// START

Structure Course:

String classString courseName (number)

Vector requirements

Vector<Course> allCourses

// MAIN

//PRINT "Welcome to ABCU Course Management System"

// WHEN TRUE:

//PRINT "Select an option:"

//PRINT "1. Load Data Structure "

//PRINT "2. "Print Course List"

// PRINT "3. Print the Course"

//PRINT "4. Exit"

Read user input and save it as a 'option' variable.

// IF "1" is selected: // Load the file data into the vector data structure.

// ELSE IF Call the loadDataIntoVector() method. // This function contains the previously written pseudocode.

//ELSE IF "2" is selected, call printCoursesInOrder().

// ELSE IF "3" is selected,

//PRINT "Enter course number:"

//Take user input and save it as 'courseNumber'.

//Call courseInformation.print(allCourses, courseNumber)

//ELSE IF you select "4",

// software will be terminated.

//Else:

//"Invalid selection." "Try once more."

//loadDataIntoVector() function:

// This would contain the pseudocode we had previously created for loading data from the file into the vector.

//printCoursesInOrder() function:

//AllCourses may be sorted by course.Courses are sorted alphabetically by number.

// FOR Every course in allCourses:

//PRINT the course.courseOf course, there is a number.courseName

//END

**HashTable Data Structure Pseudocode**

// Similar structure to Vector pseudocode

HashTable<String, Course> courseTable

//MAIN

//PRINT "Welcome to ABCU Course Management System"

// WHEN TRUE:

//PRINT "Select an option:"

//PRINT "1. Load Data Structure "

//PRINT "2. "Print Course List"

// PRINT "3. Print the Course"

//PRINT "4. Exit"

//loadDataIntoHashTable() function:

// This would contain the pseudocode we previously wrote for loading data from the file into the hash table.

//printCoursesInOrder() function:

//Vector<String> allCourseNumbers

// FOR courseTable.keys(), for each key:

Add the key to allCourseNumbers

// AllCourseNumbers should be sorted alphabetically.

//FOR every courseNumber of courses in allCourseNumbers:

course table = course.get(courseNumber)

course.course should be printed.course.courseName, number

// END

**Tree Data Structure Pseudocode**

// Similar structure to Vector pseudocode

TreeNode root = null

//MAIN

//PRINT "Welcome to ABCU Course Management System"

// WHEN TRUE:

//PRINT "Select an option:"

//PRINT "1. Load Data Structure "

//PRINT "2. "Print Course List"

// PRINT "3. Print the Course"

//PRINT "4. Exit"

//loadDataIntoTree() function:

// This would contain the pseudocode we had previously generated for loading data from the file into the tree.

//Call inOrderTraversal(root) in the function

printCoursesInOrder().

// inOrderTraversal(TreeNode node) function:

// IF node is not null, inOrderTraversal(node.leftChild) is called.

Node.course should be printed.node.course, data.courseNumberinOrderTraversal(node.rightChild) Data.courseName

// END

**Evaluation:**

1. **Vector**
   1. Advantages:
      1. Vectors are quite simple. They're similar to a list, and I can simply add courses to the end.
      2. Vectors are wonderful for seeing the sequence in which I added the courses since they maintain everything in the order I placed it in.
   2. Disadvantages:
      1. One disadvantage is that if I want to add or delete a course in the middle, it might take some time since I may need to move other courses around.
      2. Also, if I'm looking for a certain course, I can wind up checking every single course, which isn't very fast.
2. **HashTable**
   1. Advantages:
      1. When I enter a course number, and it instantly displays the course information without examining every course. It's very efficient to look things up
      2. When utilizing a hash table, I don't have to worry about the sequence of the courses because it doesn't care.
   2. Disadvantages:
      1. One problematic thing is that two course numbers may wish to go in the same location in the magic box at the same time. This is known as a "collision," and I need to figure out how to handle it.
      2. Another issue is that HashTables don't remember the sequence in which I added the courses, thus it's not the ideal option if I require them in a precise order.
3. **Tree**
   1. Advantages:
      1. The nicest feature is that they maintain courses in sorted order, so I can easily browse courses from lowest to highest course number
   2. Disadvantages:
      1. Making ensuring the tree is "balanced" is one thing I need to keep an eye on. If it is not, it may become overly tall and thin, and the hunt for a path may take longer.
      2. When adding or eliminating courses, there is also a bit more labor necessary to ensure the tree remains balanced.

After considering the project's requirements, I feel the HashTable is the best fit. The key reason is that it is really fast when I need to locate a certain course. If I additionally required the courses to be in a specified order, I'd probably consider using a tree. However, for our project, I believe HashTable has the advantage due to its quickness in locating courses.