Scott Bosdash

SNHU

Project 1

Pseudocode for vector:

// FUNCTION for prerequisites for a course

int prerequisiteCourses(Vector<Course> courses, Course c) {

int count = 0

FOR EACH prerequisite p in c.prerequisites

count += prerequisiteCourses(courses, courses[getIndex(courses, p)])

RETURN count + c.prerequisites.length }

// FUNCTION to print schedule of courses

void printSchedule(Vector<Course> courses) {

FOR EACH course c in courses PRINT(c.courseNumber + " - " + c.name) }

// FUNCTION to PRINT course info with corresponding prerequisites

void printCourseInformation(Vector<Course> courses, String courseNumber) {

FOR EACH course c in courses

IF c.courseNumber == courseNumber print(c.courseNumber + " - " + c.name) if c.prerequisites.length > 0 print("Prerequisites:")

FOR EACH prerequisite p in c.prerequisites print(p + " - " + courses[getIndex(courses, )].name) }

Pseudocode for hash table:

// FUNCTION for the number of prerequisites for a course

int prerequisiteCourses(Hashtable<Course> courses, Course c) {

int count = 0

FOR EACH prerequisite p in c.prerequisites count += prerequisiteCourses(courses, courses.get(p))

RETURN count + c.prerequisites.length }

// FUNCTION to PRINT a sample schedule of all courses

void printSchedule(Hashtable<Course> courses) {

FOR EACH course c in courses PRINT(c.courseNumber + " - " + c.name) }

// FUNCTION to PRINT course information and prerequisites void printCourseInformation(Hashtable<Course> courses, String courseNumber) {

Course c = courses.get(courseNumber)

IF c != null PRINT(c.courseNumber + " - " + c.name)

IF c.prerequisites.length > 0 print("Prerequisites:")

FOR EACH prerequisite p in c.prerequisites print(p + " - " + courses.get(p).name) }

Pseudocode for binary tree:

// FUNCTION to count number of prerequisites for courses

int prerequisiteCourses(Tree<Course> courses, Course c) {

int count = 0

FOR EACH prerequisite p in c.prerequisites count += rerequisiteCourses(courses,courses.get(p)) RETURN count + c.prerequisites.length }

// FUNCTION to PRINT schedule of all courses

void printSchedule(Tree<Course> courses) {

FOR EACH course c in courses.inorderTraversal() print(c.courseNumber + " - " + c.name) }

// FUNCTION to PRINT course info with prerequisites

void printCourseInformation(Tree<Course> courses, String courseNumber) {

Course c = courses.get(courseNumber) if c != null print(c.courseNumber + " - " + c.name)

if c.prerequisites.length > 0 PRINT("Prerequisites:")

FOR EACH prerequisite p in c.prerequisites PRINT(p + " - " + courses.get(p).name) }

**Vector:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Operations** | **Cost per line** | **Execution times** | **Big O Value** |
| **Opening and reading a file** | **1** | **O(n)** | **O(n)** |
| **Parsing each line while creating objects for course** | **1** | **O(n)** | **O(n)** |

**Hash Table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Operations** | **Cost per line** | **Execution times** | **Big O Value** |
| **Opening and reading a file** | **1** | **O(n)** | **O(n)** |
| **Parsing each line while creating objects for course** | **O(1)** | **O(n)** | **O(n)** |

**Binary Search Tree:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Operations** | **Cost per line** | **Execution times** | **Big O Value** |
| **Opening and reading a file** | **1** | **O(n)** | **O(n)** |
| **Parsing each line while creating objects for course** | **O(log n)** | **O(n)** | **O(n log n)** |

**Advantages:**

**Vector-**Vectors have high performance when accessing sequential data and have a large memory storage.

**Hash Table-**Hash tables have a fast lookup time and are best suited to retrieve data quickly.

**Binary tree (or other trees)-**Binary trees are effective and efficient when using insertion or deletion and are good for sorting.

**Disadvantages:**

**Vector-**When using deletion or insertions, they are costly in the middle (O(n)) and searching for data is slower than with a hash table or a tree.

**Hash Table-**Since it is very complex, it needs to have a large memory storage.

**Binary Trees(and others)-**Also, like the hash table, a higher memory usage due to its structuring.

**Recommendation:**

Given the data that is being worked on, I would say that a hash table would be best suited for the operations. The fast lookup time is the selling point of it, as it will give the user faster access to the course info that is being searched for. Each course having a unique ID will speed the process up even more.