Kyle Lund

CS-300

# Project 1

## Pseudocode for milestones

### Reading File

Open File

For each line in the file:

Split each line with a comma to get course details

If length of details is less than 2

Print error

courseNumber = details[0]

prerequisites = details[2:]

for prerequisite in prerequisites:

if prerequisite not in courseNumbersSet:

print error

Close file

### Vector

//create course object and store in vector data

CourseObject = CreatecourseObject(courseNumber, courseTitle, prerequisite)

VectorAppend(courseVector, courseObject)

newCourse = new CourseObject()

newCourse.courseNumber = courseNumber

newCourse.courseTitle = courseTitle

newCourse.prerequisite = prerequisite

return newCourse

//Search and print information

Input a string “courseNumber”

For each course in courses

If course number = “CourseNumber”

Print course, courseNumber, courseTitle, and prerequisite

Break

If no course is found print error

### HashTable

//Create HashTable

Initializd course Nodes

Create hash table class

While looping through file

For each line in file

For first and second values

Hold values in created temp item

If third value exists

Add to the current value

Insert method for each value

Search and output for hash table

Assign an input to key

If key is found

Print out course information

For each prerequisite in course

Print prerequisite information

### Tree

Create Tree and add nodes:

Define Binary tree class

Create root that points to null

Create insert method

If root is null, current course is root

Else if course number is less than root add left

If left is null add course number

Else if right is null, this course becomes right

Else recurse course down left node

Printing course information

Traverse the binary search tree in order

For each course object

Print courseNumber and courseTitle

If course object has prerequisites, print each prerequisite

End

## Menu Pseudocode

Create a Bid variable to access

While input does not equial 4;

Print 1. Load data structure

Print 2. Course List

Print 3. Course

Print 4. Exit

Switch

Case1:

loadBids(bid);

break;

Case2:

Print course list;

Break;

Case3:

Print course;

Break;

Case4:

End program;

Break;

## Alphanumeric order Pseudocode

### Vector

bool compareCourses

return a.courseNumber < b.courseNumber;

void printCourses(Vector courses)

sort(courses.begin(), courses.end(), compareCourses);

Print list

for each Course c in courses

print course number and title

### HashTable

print courses

void printCourses(Hashtable courses)

convert the hashtable to vector

Vector sortedCourses;

for each entry in courses

add entry.value to sortedCourses;

Sort courses based on alphanumeric course number

sort(sortedCourses.begin(), sortedCourses.end(), compareCourses);

Print list

for each Course c in sortedCourses

print course number and title

### Tree

Function to insert a course into the tree

void insertCourse(Tree courses, Course c)

print courses function

void printCourses(Tree courses)

print courses in order

inOrderTraversal;

In-order traversal function

void inOrderTraversal(Tree courses)

if courses is not empty

inOrderTraversal(courses.leftChild);

print course number and title

inOrderTraversal(courses.rightChild);

## Evaluation

### Vectors

|  |  |  |  |
| --- | --- | --- | --- |
| Operations | Cost Per Line | Number of times executed | Big O value |
| Opening and reading a file | 1 | O(n) | O(n) |
| Parsing Each Line and Creating Course Objects | 1 | O(n) | O(n) |

**Advantages:**

Creating a vector is one-dimensional and performs better in sequential access of data. It also has large memory storage.

**Disadvantages:**

Inserting and deleting data elements in the middle take a lot of time O(n). Searching for data is slower than hashtables and trees.

### HashTable:

|  |  |  |  |
| --- | --- | --- | --- |
| Operations | Cost Per Line | Number of times executed | Big O value |
| Opening and reading a file | 1 | O(n) | O(n) |
| Parsing Each Line and Creating Course Objects | O(1) | O(n) | O(n) |

**Advantages:**

The has table has a very fast look up time and is fast for retrieval of data elements. A hash table allows for data to be organized and stored through the project with a key. You are able to call, create, and delete the project.

**Disadvantages:**

Hashtables are complex and take up memory as well as slowing speeds due to sychronization

### Tree

|  |  |  |  |
| --- | --- | --- | --- |
| Operations | Cost Per Line | Number of times executed | Big O value |
| Opening and reading a file | 1 | O(n) | O(n) |
| Parsing Each Line and Creating Course Objects | O(log n) | O(n) | O(n log n) |

**Advantages:**

Trees are efficient for operations like insertion and deletion. They are also suitable for sorting data elements. Allows for organization by storing data in the left or right branch and can run searches throughout the project.

**Disadvantages:**

Similar to modify as a hashtable but takes longer to modify. Has a higher memory usage due to tree structure.

## Recommendation

My recommendation out of the three data structures would be the Hash Tables. The most important thing for a user when searching for their classes and prerequisites would be the look up times. The Hash table provides the fastest look up times for retrieval of data. It will provide quick access to course information and each course can have a unique identifier making hash tables the most suitable data structure.