Kody Greenberg

CS 300

4/6/2023

Project One

# Data Structures:

**Vector Data Structure:**

// initialize variables

**INITIALIZE** class Course {

**string** courseID

**string** courseName

**vector<string>** coursePreReqs

};

**INITIALIZE** vector<Course> courses;

**INITIALIZE** string line variable for file navigation;

// load file and place data into courses vector

**OPEN** file into file variable using fstream;

**WHILE** End of File (EOF) has not been reached {

vector<string> courseLine

Course newCourse

**PARSE** each line seperated by "," into courseLine vector

**IF** courseLine.size < 2

**return** error "invalid data file"

**FOR** each element in courseLine {

**IF** element index equals 0

**ADD** to newCourse.courseID

**IF** element index equals 1

**ADD** to newCourse.courseName

**IF** courseLine.size > 2

**IF** element at index courseLine.size -1 does not equal courses.Course.courseID

**return** "prerequisites not found"

**IF** element index > 2, pushback to newCourse.course

}

}

**PUSHBACK** newCourse to courses

};

**CLOSE** file;

**Hash table Data Structure:**

**START**

// initialize variables

**INITIALIZE** Course Vector vector<Node> nodes

**CREATE** the Hash table Class

Make an **INSERT** method to store items to **Hash table**

// load file and place data into courses vector

**OPEN** file using fstream

**IF** return value is -1 file not found

**DISPLAY** "ERROR file not found"

**ELSE** file is found

**WHILE** not at the end of file

**FOR** each line in the file

**FOR** the first and second value

**CREATE** temporary item to hold item values.

**IF** a third value exist

**ADD** that value to current values

**CALL** this **INSERT** method for each value

**IF** there are less than 2 values

**DISPLAY** "ERROR"

**ELSE**

**READ** parameters

**IF** there are greater than 2 parameters

**IF** third parameter is first elsewhere

**CONTINUE**

**ELSE** "ERROR"

**DISPLAY** “Which course would you like to print information for?”

**USER** **INPUT**

**ASSIGN** the user input to a key

**IF** the key is found

**DISPLAY** the course information

**FOR** each prerequisite

**DISPLAY** the course information.

**ELSE**

**DISPLAY** "ERROR course not found"

**CLOSE** the file

**END**

**Binary Search Tree (BST) Data Structure:**

**START**

**OPEN** file using fstream

**IF** return value is -1 file not found

**DISPLAY** "ERROR file not found"

**ELSE** file is found

**WHILE** not at the end of file

**READ** file by parsing each line

**FOR** each line in the file

**CREATE** BinarySearchTree(BST) courseInfo to hold course info

**CREATE** string courseData

**GETLINE** from file into courseData

**WHILE** courseData length is greater than 0

**IF** courseData includes at least two parameters AND prerequisite(s) exists

**ADD** courseData into BST courseInfo

**ELSE**

**DISPLAY** "Error: insufficient course data"

**EXIT**

END

# Pseudocode Menu:

**START**

**DISPLAY** Menu for User

**Prompt** user for entry

**IF** user enters 1: Load Courses

**READ** file according to data structure

**IF** user enters 2: Print alphanumerically ordered list of courses

**FOR** each line in loadCourses()

**InOrder()**

**DISPLAY** all courses

**IF** user enters 3: Print Course

**DISPLAY** “Which course would you like to print?”

**USER INPUT**

**SEARCH** loadCourses()

**IF**  input matches a course

**DISPLAY** course information

**ELSE**

**DISPLAY** “Sorry course not found, please try another”

**USER INPUT**

**IF** user enters 9: Exit

**DISPLAY** message "goodbye”

**EXIT**

END

# Pseudocode to print in alphanumeric order:

**SET** mid to low + (high - low) / 2

**SET** pivot to courseName(mid)

**WHILE** courseName(low) is less than pivot

**SET** low to low + 1

**WHILE** pivot is less than courseName(high)

**SET** high = high - 1

**IF** low is greater than or equal to high

**RETURN**

**ELSE**

**CREATE** tmp to courseName(low)

**SET** courseName(low) to courseName(high)

**SET** courseName(high) to tmp

**SET** low to low + 1

**SET** high to high - 1

**RETURN** high

**FUNCTION** InOrder()

**CALL** quicksort(courseName, 0, SIZE - 1)

**DISPLAY** "Sorted in alphabetical order: "

**FOR** each course

**DISPLAY** course info

END

# Runtime Analysis:

**Vector:**

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Executions | **Total** |
| Open Fstream | 1 | 1 | 1 |
| While not EOF, read each line | 1 | n | n |
| If < 2 values return error | 1 | n | n |
| Else continue | 1 | n | n |
| Initialize course <<vector>> | 1 | 1 | 1 |
| For each line | 1 | n | n |
| Add value to vector | 1 | n | n |
| Pushback course | 1 | n | n |
| Total Cost | | | 7n+2 |
| Runtime | | | O(n) |

**Hash table:**

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Executions | **Total** |
| Open Fstream | 1 | 1 | 1 |
| While not EOF, read each line | 1 | n | n |
| If < 2 values return error | 1 | n | n |
| Else continue | 1 | n | n |
| Create Hash table | 1 | 1 | 1 |
| Insert Bid | 1 | n | n |
| Loop file | 1 | n | n |
| For each line | 1 | n | n |
| Create Tmp to hold values | 1 | n | n |
| Total Cost | | | 8n+2 |
| Runtime | | | O(n) |

**BST:**

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Executions | **Total** |
| Open Fstream | 1 | 1 | 1 |
| While not EOF, read each line | 1 | n | n |
| If < 2 values return error | 1 | n | n |
| Else continue | 1 | n | n |
| Create BST | 1 | 1 | 1 |
| Insert Bid | 1 | n | n |
| While NOT EOF | 1 | n | n |
| Loop file | 1 | n | n |
| For each line | 1 | n | n |
| For 1st and 2nd value, add courseID and courseName | 1 | n | n |
| If 3rd value exist add until newline found | 1 | n | n |
| Total Cost | | | 10n+2 |
| Runtime | | | O(n) |

# Advantages and Disadvantages of Each Data Structure:

**Vector:**

With Vector a major advantage is the speed at which it reads files and adds course objects. The disadvantage is that it is slow when searching for specific items in the data structure. For each input it has to search though each course in order until it is found or returns -1.

**Hash table:**

The hash table has the advantage of very fast search results since it maps keys to items in a list. However, they are not sorted and results in more time to sort the table in alphanumeric order.

**BST:**

The binary search tree has a great ability to organize and sort items easily. This allows the items to be searched for and found the fastest of the options considered. It does take longer to process the initial organization of the items than a hash table but the search in itself makes up the speed oof the organization. A disadvantage is that it takes one of the longest times to make modifications to the stored data.

# Recommendation:

Due to the nature of this project and the fact there will not be thousands upon thousands of items stored, I would recommend Vectors. This is because the biggest part of the requirement is to be able to create and add items. Vectors handle this aspect the most efficient of the data structures considered. It also is very fast with the least runtime of the three structures broken down. While it does take longer to search and print course information, with the data being stored that will be absolutely minimal.