Steven Davis

Southern New Hampshire University

CS- 300 Project

Saba Jamaliannasrabadi

Struct course

String course number

String title

Initialize prereqquisites as empty optional list

Initialize courseList as empty list

Function readCourseData(file):

Open file with provide file name

If file does not exist

Print “file could not be found”

Return

Create vector course as empty vector

while each line in file

Split into tokens as courseData

If length of course Data is less than 2

Print “Missing course number or title”

Continue to next line in vector

Create newCourse as Course object

Set new course, courseNumber as courseData[0]

Set new course, Title as courseData[1]

If length of courseData less than 2

Add new course to courseList

For each course in course

For each prerequisites in course prerequisites

Set check Prerequisite as false

For each other course in course

If prerequisite equals other course number

Set validPrerequisite to true

Break

If check prerequisite is false

Print error in prerequisite collection

Return course

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

**for all courses**

**if the course is the same as courseNumber**

**print out the course information**

**for each prerequisite of the course**

**print the prerequisite course information**

**else**

**print no prerequisites**

**Main ()**

**Set file name as courses.txt**

**Set course as a result of readCourseData**

**Call void search course method**

//Hash Table:

Struct course

String course number

String title

Initialize prereqquisites as empty optional list

Initialize courseList as empty list

HashTable:: HashTable

Set Hashtable resize

Function readCourseData(file):

Open file with provide file name

If file does not exist

Print “file could not be found”

Return

Create hashTable course as empty hash Table<Course>

void searchCourse(HashTable<Course> courses, String courseNumber) {

for each line in file

Split into tokens as course number, title, prerequisites

If length of tokens is less than 2

Print “Missing course details”

Continue to next line

Create newCourse as Course(courseNumber, courseTitle, prerequisites)

Set course\_number, courseData[0]

Set course\_title, courseData[1]

Set prerequisites equals empty list

Insert course into hashTable using courseNumber as key

If the number of courseData greater than 2

For each token starting at third token

Add token to prerequisites list

Return course

}

**Main ()**

**Set file name as courses.txt**

**call readCourseData**

**Call void search course method**

Function print course info (hashTable string, course courseTable)

For each course in hashTable

Print “course number: << courseNumber

Print “Course Title: << courseTitle

If prerequisites is not empty

Print “prerequisites: “ << prerequisite

Else

Print “ No prerequisites”

//Binary Search Tree

Struct course

String course number

String title

Initialize prereqquisites as empty optional list

Initialize courseList as empty list

Function readCourseData(file):

Open file with provide file name

If file does not exist

Print “file could not be found”

Return

Create class BinarySearchTree

Create constructors for courseNumber equal to courseNumber

Create constructor for title set equal to title

Create constructor for prerequisites set equal to prerequisites

courseBST as new BinarySearchTree()

Function to create both list

while each line in file

Split into tokens as courseData

If length of course Data is less than 2

Print “Missing course number or title”

Continue to next line in vector

Create newCourse as Course object

Set new course, courseNumber as courseData[0]

Set new course, Title as courseData[1]

If length of courseData less than 2

Add new course to courseList

For each course in course

For each prerequisites in course prerequisites

Set check Prerequisite as false

For each other course in course

If prerequisite equals other course number

Set validPrerequisite to true

Break

If check prerequisite is false

Print error in prerequisite collection

Return course

Insert list int Binary SearchTree

Function BinarySearchTree:: insert(course)

If BST is empty

Set courseRoot = course

Return

Set current node = to courseRoot

While current node is not null

If course.courseNumber less than current node courseNumber

If current node left is null

Set current node left equals course

Return

Else

Set current node equal to node transverse left

Else if course.courseNumber greater than current node.courseNumber

If current node point to right is null

Set current node point to right to course

Return

Else

Set current node equal to current node point to right

Else

Return course

Print

Print courseNumber, courseTitle, prerequisites

BinarySearchTree InOrder traversal

If node is not empty

Set in order node to the left

Print course.nodecourse

Set in order node to the right

Funciont BinarySearchTree:: print all course

Call to inorderTraversal

Call to Print function

Menu:

Set choice to 0

While choice is not 4 loop

Print menu choice “1. Load Course File into course list”

Print menu choice “2. Print Course List in order”

Print menu choice "3. Print Individual Course title and prerequisites.”

Print menu choice “4. Exit”

Case 1 call Load Course function(CourseList)

Case 2 call print sorted list function

Case 3 call print course information

Case 9 exit program

break

| **Vector** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create vector** | 1 | 1 | 1 |
| **For each line in the file** | 1 | n | n |
| **Split into tokens** | 1 | n | n |
| **Create course list vector** | 1 | 1 | 1 |
| **Add to course list** | 1 | n | n |
| **While prerequisites in list** | 1 | n | n |
| **Print for each line in list** | 1 | n | n |
| **Total Cost** | | | 7n + 2 |
| **Runtime** | | | O(n) |

| **Hashtable code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Create hashTable | 1 | 1 | 1 |
| For each line in course list | 1 | n | n |
| **Split into tolkens 0 and 1** | 1 | n | n |
| If length of course data less than 2 | 1 | n | n |
| **Create course list new course** | 1 | 1 | 1 |
| **Set course number to data 0** | 1 | n | n |
| **Set course title to data 1** | 1 | n | n |
| **Using course number as key insert into hashtable** | 3 | n | 3n |
| **If course data greater than 2** | 1 | n | n |
| **Add data to the prereq list** | 1 | n | n |
| **Print course list** | 1 | n | n |
| **For each course in course list print** | 1 | n | n |
| **Print course title and number** | 1 | n | n |
| **Total Cost** | | | 15n + 5n |
| **Runtime** | | | O(n) |

| **Binary Search Tree** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create BST** | 1 | 1 | 1 |
| If BST is empty | 1 | n | n |
| Set courseRoot = course | 1 | 1 | 1 |
| Set current node = to courseRoot | 1 | 1 | 1 |
| While current node is not null | 1 | n | n |
| If course.courseNumber less than current node courseNumber | 1 | n | n |
| If current node left is null | 1 | n | n |
| Set current node equal to node transverse left | 1 | n | n |
| Else if course.courseNumber greater than current node.courseNumber | 1 | n | n |
| If current node point to right is null | 1 | n | n |
| Set current node point to right to course | 1 | n | n |
| Else Set current node equal | 1 | n | n |
| **Total Cost** | | | 12n + 2 |
| **Runtime** | | | O(log n) |

Each of these data structures offers advantages and disadvantages depending on the files you want to store and be able to search through. The main advantage of vectors is that this data structure has fast access, meaning you can access elements in the list efficiently. They are also very easy to implement compared to hash tables and Binary search trees. The main disadvantage is inserting or deleting the elements in the middle of the vector can shift large portions of the array, which could lead to performance issues. Going on to hash tables one of the main advantages is a fast search this data structure can search for specific elements in near constant time. Since this data structure uses keys to store the data you can also store diverse data types. One disadvantage of hash tables is that they can have collision handling problems multiple keys could get assigned to the index which would lead to performance issues. The other problem is that hash tables have a higher chance of more memory overhead which would use more memory than the other data structures. Using a Binary search tree the advantage this data structure has is its efficiency in sorting and retrieving the data within the search tree. Compared to the hash table it is easier to resize having a more dynamic resizing of the tree with less memory reallocation. The main disadvantage is that the binary search tree is more complex to implement compared to vectors and hash tables. Binary also shares a disadvantage with hash tables requiring more memory reallocation.

Finally, for this course list data I would like to recommend using a vector based on my runtime research the program would be faster. Since in college new courses are not added frequently losing the speed of insertion would not be a problem. The program would be able to print the whole list the fastest and specific searches would take longer the faster access makes up for the time lost in the search.