**Module 6 Project One**

Zane Milo Deso

Southern New Hampshire University

**Project One**

## Introduction

As a Computer Science student at SNHU I was assigned the task of applying non-coding development methodologies for outlining an algorithmic design and evaluating complex data structures that solve a given problem using advanced algorithmic designs. In this project I have crafted the pseudocode for each program containing data structures such as vectors, hash tables, and binary search trees. Furthermore, I completed a runtime analysis on each variation of the program, compared the advantages and disadvantages of each, and recommended academic advisors for the Computer Science department at ABCU on which to use.

## Vector Data Structure

|  |  |  |
| --- | --- | --- |
| **Code** | **# Times Executes** | **Total Cost** |
| ParseFile() | n^2 | n^2 |
| PrintAll() | n^2 | n^2 |
| **Total Cost** | | 2\*O(n^2) |
| **Runtime** | | O(n^2) |

**Pseudocode**

**/**/Menu

void menu() {

WHILE Input is not 9:

PRINT menu options:

IF input is 1:

CALL parseFile(PATH\_TO\_FILE, ‘,’)

IF input is 2:

CALL printAll()

IF input is 3:

PRINT Please input courseId to look up

SET courseId to INPUT

CALL searchCourse(vector, courseId)

IF input is 9:

EXIT  
}

### // Parse File & Create Object

void parseFile(String path, Char separator) {

OPEN path to file

DECLARE temp list

DECLARE validCourse list

FOR each row in file

APPEND first string to validCourse

FOR each row in file

INIT CourseObject

FOR each string separated by separator

APPEND string to temp list

IF length of temp list < 2:

PRINT error, format for line is incorrect

BREAK // Line did not meet format, moving to next line

SET CourseObject->CourseID to first temp list string

SET CourseObject-> CourseName to second temp list string

SET j to 2

WHILE j <= length of temp list

IF string IN validCourse list

APPEND string to CourseObject->Preqs

j++

APPEND CourseObject to courses vector //Add object to Vector

}

### // Find and Print given 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

}

### // Print All Course Information

Void PrintAll() {

DECLARE results list

FOR each element in the vector

IF it is not empty

APPEND Course to the results list

QSORT results list // Sorts results from least to greatest

FOR each result in results list  
 PRINT current->CourseObject.ID, current->CourseObject.Title,

current->CourseObject.Preq

}

## Hash Table Data Structure

|  |  |  |
| --- | --- | --- |
| **Code** | **# Times Executes** | **Total Cost** |
| ParseFile() | n^2 | n^2 |
| PrintAll() | n^2 | n^2 |
| **Total Cost** | | 2\*O(n^2) |
| **Runtime** | | O(n^2) |

**Pseudocode**

### //Menu

void menu() {

WHILE Input is not 9:

PRINT menu options:

IF input is 1:

CALL parseFile(PATH\_TO\_FILE, ‘,’)

IF input is 2:

CALL PrintAll()

IF input is 3:

PRINT Please input courseId to look up

SET courseId to INPUT

CALL Search(courseId)

IF input is 9:

EXIT  
}

### // Parse File & Create Object

void parseFile(String path, Char separator) {

OPEN path to file

DECLARE temp list

DECLARE validCourse list

FOR each row in file

APPEND first string to validCourse

FOR each row in file

INIT CourseObject

FOR each string separated by separator

APPEND string to temp list

IF length of temp list < 2:

PRINT error, format for line is incorrect

BREAK // Line did not meet format, moving to next line

SET CourseObject->CourseID to first temp list string

SET CourseObject-> CourseName to second temp list string

SET j to 2

WHILE j <= length of temp list

IF string IN validCourse list

APPEND string to CourseObject->Preqs

j++

insert(CourseObject Course) // Call to add object to Hash Table

}

Void insert(CourseObject Course) {

SET key to CONVERTED last three chars of Course->CourseID string to integer

Hash(key)

SET current to node[key]

IF current is empty

SET current node to INIT new node(Course, key)

ELSE

WHILE next node is not empty

SET current to next

SET next to INIT new node(Course, key) // Add obj to HashTable

}

Unsigned int hash(int key) {

RETURN key % size of hash table

}

### // Find and Print given course

Course Search(string CourseId){

INIT empty course to return if not found

CONVERT CouseId to int

HASH CourseId

SET current pointer to the calculated key of CourseId hash value

IF current.CourseId == CourseId  
 PRINT current.Course Info and prerequisites

WHILE current.next is not nullptr

IF current.next == CourseId

PRINT current.next.Course info and prerequisites

ELSE

SET current to current.next

RETURN course

}

### // Print All Course Information

Void PrintAll() {

DECLARE results list

FOR each bucket in the table

SET current to current bucket node

IF it is not empty

APPEND Course to the results list

WHILE next node is not empty

SET current to next

APPEND Course to the results list

QSORT results list // Sorts results from least to greatest

FOR each result in results list  
 PRINT current->CourseObject.ID, current->CourseObject.Title,

current->CourseObject.Preq

}

## Binary Search Tree Data Structure

|  |  |  |
| --- | --- | --- |
| **Code** | **# Times Executes** | **Total Cost** |
| ParseFile() | n+m | O(n+m) |
| PrintAll() | Logn | O(Logn) |
| **Total Cost** | | O(Logn^2+m) |
| **Runtime** | | O(Logn+m) |

**Pseudocode**

### //Menu

void menu() {

WHILE Input is not 9:

PRINT menu options:

IF input is 1:

CALL parseFile(PATH\_TO\_FILE, ‘,’)

IF input is 2:

CALL printInOrder(root)

IF input is 3:

PRINT Please input courseId to look up

SET courseId to INPUT

CALL Search(courseId)

IF input is 9:

EXIT  
}

Struct Course{

DECLARE CourseID

DECLARE CourseName

DECLARE Prequisites List

CONSTRUCTOR Course{}

}

Struct Node{

DECLARE Course

DECLARE right pointer

DECLARE left pointer

CONSTRUCTOR Node {

SET left pointer to nullptr

SET right pointer to nullptr

}

CONSTRUCTOR Node with course

}

### // Parse File & Create Object

void parseFile(String path, Char separator) {

OPEN path to file

DECLARE validCourses list

FOR rows in file

DECLARE temp list

WHILE each string is separated by separator

SPLIT the string

APPEND the string to temp list

APPEND first string up to separator to validCourses list

IF length of temp < 2

PRINT Course lacking at least 2 paramaters

BREAK

ELSE

DECLARE i = 2

INIT new course object from struct

SET courseID to first index in temp list

SET title to second index in temp list

WHILE i < length of temp list // appends all prerequisite to object list

IF temp list at index i is NOT IN validCourses

PRINT Course Prerequisite not found in valid courses

ELSE

APPEND string to Course.Prerequisties list

CALL insertNode Function with newly instantiated Course Object as args

}

void insertNode(Course course) {

IF root is nullptr

SET root to INIT Node with param course object

ELSE

CALL addNode(root, course)

}

void addNode(Node\* node, Course course) {

IF last 3 chars of course.courseID < node.courseID

IF node.left is nullptr

SET node.left to INIT Node with param course object

ELSE

RECURS addNode(node.left, course)

ELSE

IF node.right is nullptr

SET node.right to INIT Node with param course object

ELSE

RECURS addNode(node.right, course)

}

### // Find and Print given course info and prerequisites

Course Search(string CourseId) {

DECLARE & INIT empty course //If courseId not found returns empty course

SET current node to root

WHILE current node is not nullptr

IF current.CourseId == CourseId  
 PRINT current->CourseObject.ID, current->CourseObject.Title,

current->CourseObject.Preq

RETURN current.course

RETURN course

}

### // Print All Course Information

void printInOrder(Node\* node) {

IF node is nullptr

RETURN

CALL printInOrder(node.left)

PRINT node.course.courseID, node.course.title, node.course.prerequisites

CALL printInOrder(node.right)

}

## Advantages & Disadvantages

|  |  |  |
| --- | --- | --- |
| Data Structure | Advantage(s) | Disadvantage(s) |
| Vector | Fast insertion  Space Complexity is small | Slow look up  As input grows, becomes slower |
| Hash Table | Key required to access  Fast look up | Requires Space for pointers  Slower insertion  Requires more knowledge to code  As input grows, becomes slower |
| Binary Search Tree | Fast look up  Tree sorted as it is made  Fast Insertion  Alternate ways to traverse | Requires Space for pointers  Requires more knowledge to code |

## Recommendation & Conclusion

In conclusion, I would strongly encourage the academic advisors of the Computer Science department to use the Binary Search Tree data structure as it has the advantage of being fast at insertions and look ups as well as not needing to use a separate function to sort the values as the tree balances as it is adding nodes. A Binary Search Tree over the other available implemented data structures would strongly meet the use case of the academic advisors to look up courses and their prerequisites with the added benefit as the database grows the search function should not slow down as much as a vector or hash map. Using this data structure will increase the program's efficiency, leading to a largely better user experience and reliability as the course list grows.