6-2: Project One

Dakota Keyes

CS300 DSA: Analysis & Design

VECTOR PSEUDOCODE pg. (2-5)

HASH TABLE PSEUDOCODE pg. (6-8)

BINARY SEARCH TREE PSEUDOCODE pg. (9-10)

MENU & PRINT COURSES PSEUDOCODE pg. (11-12)

TIME COMPLEXITY ANALYSIS pg. (13-16)

ADVANTAGES & RECOMMENDATIONS pg. (16)

VECTOR PSEUDOCODE

DECLARE variable “filename” and assign the path of file;

(CourseInformation.txt);

OPEN file using the variable “ilename”;

DECLARE variable “lineNumber” and initialize it to 1

FOR (each line in file) ;

++ “lineNumber”;

Split the line by comma into an array of strings called “tokens”;

IF(length of “tokens” > 2) {

PRINT “lineNumber” << “Error: not enough parameters on line” and

}

Continue to the next line;

IF (length of “tokens” > 3) {

PRINT “lineNumber” << “Error: too many parameters on line”;

}

Continue to the next line;

ASSIGN first token to a variable “courseNumber”;

ASSIGN second token to a variable “courseTitle”;

IF (length of “tokens” is 3) {

ASSIGN third token to a variable “prerequisite”;

}

CHECK file to see if course with the courseNumber “prerequisite”

(IF not in file) {

PRINT Line number << “Error: prerequisite course not found on line”

}

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

totalPrerequisites = prerequisites of course c

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

void printSampleSchedule(Vector<Course> courses) {

}

FOR (all courses) {

IF( course is same as courseNumber)

PRINT course info

FOR (each prerequisite of the course)

PRINT prerequisite information

}

Create new course object (course);

Send “course” to <courses> vector

}

Close the file

int numPrerequisiteCourses(Tree<Course> courses) {

}

FOR (each course in <courses>) {

ADD course to “courseTree”

courseNumber as the key;

}

FOR (each key in “courseTree”) {

PRINT “key”

}

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

}

int numPrerequisiteCourses(Tree<Course> courses) {

}

IF (course has prerequisite). {

PRINT “Prerequisite: “ << prerequisite courseNumber << endl

}

**Hash Table Pseudocode**

Reading File:

fstream access & open file

std:: fstream::open()

IF (return value = “-1”) {

PRINT file is found;

}

ELSE {

File is found & opened;

}

WHILE (! EOF) {

READ each line;

}

IF (There are > two values in line {

RETURN error;

}

ELSE {

READ parameters

IF (parameters >= 3) {

IF (third or > parameter is found elsewhere) {

CONTINUE

}

ELSE {

RETURN Error;

}

CLOSE file

Create Course Objects for a HashTable

Initialize Course Vector vector<Node> nodes;

Create HashTable Class;

Create Insert method to insert items to HashTable;

//Looping through the file

WHILE (! EOF) {

FOR (every line in file) {

FOR (1st and 2nd values) {

CREATE temp item to hold values

}

IF( = =3rd value exists) {

current value ++;

}

}

}

FOR (each value) {

CALL Inset method;

}

**Search and Print from HashTable:**

PROMPT user for Input;

ASSIGN input as key;

IF (Key is found) {

PRINTnumPrerequisiteCourses(Hashtable<Course> courses)

}

ELSE {

PRINT (Key not found)

}

FOR( each prerequisite of the course) {

PRINTCourseInformation(Hashtable<Course> courses, String courseNumber)

PRINTSampleSchedule(Hashtable<Course> courses)

**Binary Tree Search Pseudocode**

//Create root variable & set to null pointer

int numPrerequisiteCourses(Tree<Course> courses) {

IF (root == null ptr) {

CREATE left & right node;

CREATE inOrder function to pass the root node;

CREATE preOrder function;

CREATE. postOrder function;

CREATE the insert bid function (Node\* node, bid.bidId);

bidId = bid.bidId;

title = bid.title

fund = bid.fund;

amount = data.amount;

IF( root = = nullptr) {

SET(root = = new bid Node);

} ELSE {

ADD node root & bid;

Void BinarySearchTree:: Remove(string bidId) {

deleteNode(root,bidId);

}

Bid BinarySearchTree::Search(string BidId) {

Bid bid;

IF( root = = null) && root-> bidId == bidId) {

WHILE (cur =! Nullptr {

IF(cur bid exists) {

RETURN bid;

IF(bid < cur node) {

Traverse the left side of tree;

} ELSE {

Traverse the right side of the tree;

CREATE addNode function(Node\* node, Bid bid);

IF(node > add to left) {

IF(no left node exists) {

THIS-> node becomes the left node;

} ELSE {

IF(no right node exists) {

THIS->node becomes the right node;

} ELSE {

Recurse to the left side;

}

void printSampleSchedule(Tree<Course> courses)

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

}

**PSEUDOCODE FOR MENU**

//Input and Out

Include <iostream>

Print” 1.Load Data Structure.”; endln;  
Print “2.Print Course List.”; endln;  
Print “3.Print Course.; endln;  
Print “0. Exit.; endln;  
Print “Enter your selection”; endln;

//Get user input w/ cin

User\_input = cin;

// If/else loop to execute options from menu

If(user\_input == 0) {

Break; {

} else if (user\_input == 1) {

CSV getline() for data structure file

} else if ( user\_input == 2) {

// Display Course List

Print “Here is a sample schedule:”; endln;  
Print “CSCI100, Introduction to Computer Science”; endln;  
Print “CSCI101, Introduction to Programming in C++”; endln;  
Print “CSCI200, Data Structures”; endln;  
Print “CSCI301, Advance Programming in C++”; endln;  
Print “CSCI300, Introduction to Algorithms”; endln;  
Print “CSCI350, Operating Systems”; endln’  
Print “CSCI400, Large Software Development”; endln;  
Print “MATH201, Discrete Mathematics”; endln;

} else if(user\_input == 3) {

// Print Course Code & Prerequisites, ascending order, courses to the

//farthest left have no prerequisites, courses on the right have courses on left as //prerequisites

Print “Enter Course Code For information and Prerequisites”; endln;

Cin >>

Print ”CSCI100 -> CSCI101 -> CSCI200 -> CSCI300 -> CSCI350 -> CSCI400” endln;

Print “ CSCI101 -> CSCI301 ->CSCI400” endln;

Print “ MATH201 -> CSCI300 ->CSCI350 -> CSCI400” endln;

// need to finish the loop to display error message if user\_input is incorrect selection

// ie anything besides 1, 2, 3, or 0, prompt the user to reenter a correct number & display error

} else {

Print “Error. Enter a valid selection.. or else” endln;

//Continue to Display Menu on console

**Time Complexity Analysis**

## Example Runtime Analysis

When you are ready to begin analyzing the runtime for the data structures that you have created pseudocode for, use the chart below to support your work. This example is for printing course information when using the vector data structure. As a reminder, this is the same pairing that was bolded in the pseudocode from the first part of this document.

**Vector Data Structure Time Complexity Analysis**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 1 | n | n |
| **if the course is the same as courseNumber** | 1 | n | n |
| **print out the course information** | 1 | 1 | 1 |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 1 | n | n |
| **Total Cost** | | | 4n + 1 |
| **Runtime** | | | O(n) |

**Binary Tree Search Time Complexity Analysis**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **METHOD for adding a node** | 0 | 0 | 0 |
| **IF (root = nullptr) + node** | 1 | 1 | 1 |
| **IF(node < root) + to left node** | 1 | n | n |
| **IF(left node does not exist)** | 1 | n | n |
| **THIS node = = left node** | 1 | n | n |
| **ELSE IF(node > root) + to right** | 1 | n | n |
| **IF(right node does not exist)** | 1 | n | n |
| **THIS node = =right node** | 1 | n | n |
| **FOR(every line in the file)** | 1 | n | n |
| **CREATE vector <course\_item>** | 1 | n | n |
| **WHILE(prerequisite exists)** | 1 | n | n |
| **APPEND the prerequisite** | 1 | n | n |
| **INSERT function for course item** | 1 | n | n |
|  |  |  |  |
| **Total Cost** | | | 11n + 2 |
| **Runtime** | | | O(n) |

**Hash Table Time Complexity Analysis**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **CREATING Hash table** | 1 | 1 | 1 |
| **INSERT method** | 0 | 0 | 0 |
| **CREATING Key for Courses** | 1 | n | N |
| **IF (not found)** | 1 | n | N |
| **ASSIGN Node (key)** | 1 | n | N |
| **ELSE** |  | n | N |
| **ASSIGN key = UINT\_MAX, set old node to course, old node next ->nullptr** | 3 | n | 3n |
| **ELSE** | 1 | N | N |
| **LOCATE next new node** | 1 | N | N |
| **ADDING new node to end** | 1 | N | N |
| **FOR (each line in the file)** | 1 | N | N |
| **CREATE the vector w/ <course item>** | 1 | N | N |
| **IF (prerequisite exists)** | 1 | N | N |
| **APPEND** | 1 | N | N |
| **INSERT course item** | 1 | n | N |
| **Total Cost** | | | 15n + 1 |
| **Runtime** | | | O(n) |

Advantages/Disadvantages

Recommendations

One of the advantages of BST is that you can get all the keys of ordered list by doing an Inorder traversal. Hash tables do not have a simple method like BST for sorting a list. Hash Tables are not as well suited for smaller data sets as BST’s either. Hash Tables will use more memory as they require space to store collusions and hash values. Hash Tables would be a better option if this list was much larger, but since the list is relatively small, I think a Binary Search Tree would be the best data structure to utilize. For simple operations, like searching, inserting or removing Hash Tables do have a quicker time complexity than BST’s. If the underlying bucket could be allocated only once and not have to be resized, the hash table might be a better option. However, with the insert and remove functions, this may be difficult to resize the hash table.