**Juan J Narvarte**   
**February 19, 2023**   
**CS 300**

**Project One: Pseudocode and Runtime Analysis**

**Define course structure:**

Create struct Course {}

Create identifiers:

Course Num

Course Name

Vector<string> Prerequisites;

**VECTOR**

Create Course Objects Vector:

CLASS course has a constructor

CONSTRUCTOR Course() input and assign the name, title, and prerequisite.

IF LENGTH of each SPLIT() line is checked is less than 2 then the file is not in proper format

END CLASS

FUNCTION createObject() takes the vector of Course objects and the file as parameters.

IF readFile() returns the value TRUE the file is in the right format

For each line in file

For 1st and second value

Use pushback to add value to vector

If a 3rd value exists

Use pushback to add value until new line

Search and Print from Vector:

Ask for Input

Loop through vector

if the input is the same as courseNum

print out the course information

for each prerequisite of the course

print the prerequisite course information

**HASHTABLE**

Construct Hashtable

Create struct Node

Course course

Unsigned int key

Use vector string for Nodes

Define the table size (default\_size = 8)

Assign key value to Unsigned int key

Create insert method void HashTable::Insert(Course course)

create key for courses and search for node with the key value

if no entry found for the key

assign this node to the key position

else if node is used

assign old node key to UINT\_MAX

else find the next open node

add new Node

Assign node next to nullptr

End

void loadCourses(string csvPath, HashTable\* hashTable)

loop to read rows of a CSV file

for (unsigned int i = 0; i < file.rowCount(); i++) {

Create a data structure and add to the collection of courses

Course course;

course.courseId = file[i][1]; course.name = file[i][0];

while not end of line

course.prereq. = file[i][8];

hashTable->Insert(course);

**BINARY SEARCH TREE**

Reading File:

START fstream function to open file

Call to open file

IF the return value is less than 1 file is not found

return ERROR file is not found

Else file is found

While it is not the EOF (End of File)

Read each line

IF there are less than two values

RETURN ERROR

ELSE read parameters

IF there is a third or more parameter in first parameter

Read parameters

ELSE return Error

END (close file)

Define a Binary search tree for courses

CourseBST\* coursebst;

Coursebst = new coursebst();

Course course;

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

if (node->course.courseNum.compare(course.courseNum) > 0) {

if (node->left == nullptr) node->left = new Node(course); else this->addNode(node->left, course);

Search and print from binary tree:

Ask for input

Root points to null pointer

Create an insert method

IF the root is null the current course is the root

ELSE the courseNumber is less than the root

Add course to the left side

IF left equals NULL

Add the courseNumber

ELSE the courseNumber is less than

Add to the left

ELSE the course number is greater than

Add to the right

ELSE IF the courseNumber is greater than the root

Add course to the left side of the tree

IF the right is equal to NULL

Add the courseNumber

ELSE IF the course number is less than

Add to the left

ELSE IF the courseNumber is greater than

Add to the left

PRINT COURSE INFORMATION AND PREREQUISITES:

Set current node to equal to the root

ASK for user INPUT

WHILE current course number does not equal null pointer

IF the current courseNumber matches

Print course information

ELSE the current courseNumber is > 0

Traverse left

ELSE the current courseNumber is < 0

Traverse right

Loop until match is found

ELSE

PRINT “Error course not found”

END

**MENU**

Initialize int main(int argc, char\* argv[])

Use string csvPath to load csv file

Create case 2 to start argument for course information

Create case 3 for courseId

Create default path to csv file

Print “Welcome to the course planner!

Create a string for user choice

Int userChoice

While in menu

Cout << “1. Load Data Structure” << endl

Cout << “2. Print Course List” << endl

Cout << “3. Print Course” << endl

Cout << “9. Exit” << endl

Cout << “\nWhat would you like to do?”

Cin >> choice

Exit from menu

Cout << “\nThank you for using the course planner!” << endl

End

**EVALUATION: BIG O ANALYSIS**

|  |  |  |  |
| --- | --- | --- | --- |
| Vector | Line Cost | # Times  Executes | Total Cost |
| Create vector | 1 | 1 | 1 |
| Create Course Objects | 1 | n | n |
| Read File for course information | 1 | n | n |
| Pushback course item | 1 | n | n |
| While in Loop | 1 | n | n |
| Print the Prerequisites | 1 | 1 | 1 |
| Total cost | | | 4n + 2 |
| Runtime | | | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| **HashTable** | **Line Cost** | **#Times Executes** | **Total cost** |
| Create Hash Table | 1 | 1 | 1 |
| Create Struct Node | 1 | n | n |
| Initialize <vector> Node | 1 | n | n |
| Create Insert Method | 1 | 1 | 1 |
| Create key for course | 1 | n | n |
| Assign node to key | 1 | n | n |
| Assign old node key to UNIT\_MAX | 4 | n | 4n |
| Load Courses | 1 | n | n |
| Total Cost | | | 9n + 2 |
| Runtime | | | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| Binary Search Tree | Line Cost | #Times Executes | Total Cost |
| Create BST | 1 | 1 | 1 |
| Void courseBST::addNode | 1 | n | n |
| Root points to nullptr | 1 | n | n |
| Create insert method | 1 | 1 | 1 |
| If root is null, add root | 1 | n | n |
| If node is less than root, then add to left | 1 | n | n |
| If no left node, node becomes left | 1 | n | n |
| Total Cost | | | 5n + 2 |
| Runtime | | | O(n) |

While working on this project I learned the pros and cons of utilizing vectors, hashtables, and Binary search tree. Vectors are easy to implement and can save you time if used with binary search tree. For vectors to work properly, it must be sorted out correctly because removing items can take a huge amount of your time because of the shifting process. Also depending on the compiler currently being used for reallocation, vectors can take up a lot of space than vectors need to. The utilization of hastables can retrieve items faster than vector because of giving direct access to them. You can insert and delete it at any time regardless of the size of the table. However just like vectors, it can also take up more space than what is needed if formatted incorrectly. Retrieving items from the hashtable can be tedious because the item elements are not in order (this can also cause extensive delays). And lastly binary search tree retrieves items in order, and you can also insert/delete just like hastables but faster. How ever the shap of the BST depends on the first item inserted and BST can cause stack overflow when using recursion.

I recommend using Binary Search Tree (BST) for this project because it will make sorting out course objects easier. The course information must be displayed in alphabetical order and the BST is the best choice for that as I mentioned above. The course items are going to be traversed and that will save you time on sorting because that action is not required. Although it will take some time constructing the BST which is why I will still use vectors to fasten the process.