Elana Masboob

CS 300

6/10/2023

Project 1

**Vector Table:**

Create method void printCourseInformation(Vector<Course> courses, String courseId)

Get input for courseId

While vector is not empty

if the input is the same as courseId

output course.courseId << output course.name

while (prereq = true)

output course.prereq

**Hash Table Pseudocode:**

Structure Course

String courseName

String courseNumber

Vector<string>prereq;

Const unsigned int default\_size = 8;

Class hashTable

Structure node

Course\* course;

Node\* next;

Unsigned int key;

Node()

Node(course, course, unsigned int key)

Unsigned int size = default\_size;

Vector<node>nodes;

hashTable();

insert(course, course);

int preReqCourses(hashTable<course> courses string coursenum){

create key using courseNumber

retrieve node using key set to new node

while node is not equal to nullptr

if node pointer courseNumber is equal to courseNumber

totalPrereqs equal node.Prereqs.size();

for each preReq p in totalPrereqs

add preReqs of p to totalPrereqs

print number of totalPrereqs

else

node = node pointer next

void printCourseInfo(hashTable<course> course, String courseNum){

create key by hashing the courseNumber

retrieve node by using key and set to new node

while node is not equal to nullptr

if node pointer course courseNum is equal to courseNum

print course information

for each prereq course information

print prereq course information

else

node equals node pointer next

unsigned int hash(int key)

return key percentage table size

void insert(course,course)

unsigned key equals hasg(stoi(course.courseId)

check if node empty

IF node empty

insert course at node

IF node is not empty

WHILE loop until empty node found

Insert course at that node

Course parseLine(vector<string> &line)

IF (linesize = 2)

Course newCourse

Course courseName = line[0]

Course courseNumber = line[1]

Set course prereq to empty vector

Return newCourse

ELSE

Vector<string>temprereq

FOR(int I = 2; I < line.size(): i++)

Tempprereq.push\_back(line[i]);

Course newCourse

Course courseName = line[0]

Course courseNumber = line[1]

Set course prereq to temprereq vector

Return newcourse

Int main()

hashTable\* table equal new hashTable()

vector<string>temp

string line

ifstream infile(“file name”)

while(getline(infile, line))

stringstream ss(line)

WHILE(ss.good())

String substr

Getline(ss,substr, ‘,’)

Temp.push\_back(substr)

Table.insert(parseLine(temp))

Temp clear()

}

}

**Binary Tree Pseudocode:**

Create root variable and set equal to nullptr;

Create a left and right node;

Create function for inorder to pass through root variable;

Create function for postorder to pass through root variable;

Create function for preorder to pass through root variable;

Create insert bid function

If root == nullptr;

Set root = equal to new node for bid

Else add node root and bid

Create remove function to take parameter root and bidid

Create search function with parameter bidid

Set the current node = root

While current != nullptr

If current bid is found, return

If bid is smaller than current node

Transverse to left

Else

Transverse to left

Create addnode function with parameters node\*node, bid bid

If node is greater

Then add to left

If no left node

Then node becomes left

Else

If there is no right node

Then node becomes right

Else recurse to left

Create inorder function with parameters node\*node

If node != nullptr

Set inorder to not left

Print bidid, title, amt, and fund

Set inorder to right

Create preoreder function with parameters node\*node

If node != nullptr

Set preorder to not left

Preint bidid, title, amt, and fund

Set preorder to right

Create postorder function with parameters node\*node

If node != nullptr

Set postorder to not left

Print bidid, title, amt, and fund

Set postorder to right

**Reading File:**

Use fstream to open file

Create method void loadcourse(string csvPath, dataStructure)

Make call to open file, if return value is “-1”, file = not found

Else if file is found

While it is not eof (end of file)

Read each line

If there are less than two values in line, return error

Else read parameters

If there is a third or more parameter

If third or more parameter is in first parameter elsewhere, continue

Else return error

Close file

**Hold Course Information:**

Create struct Course{}

Create identifiers: courseID, courseName, preReq

//vector

Vector<Course>loadCourses(string, csvPath)

Create a data structure and add to the collection courses

Course course;

Course.courseID = file[i][1];

Course.name = file[i][0];

While not end of line

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

Courses.push\_back(course);

**Menu:**

Set choice to 0;

Create while loop for menu. While choice is not equal to 4

Output menu choices (1. Load Course File, 2. Print Course List 3. Print Individual Course 4.Exit)

Create switch(choice)

Case 1: loadCourses(courseFile, dataStructure) FIXME: use structure of data structure chosen

Case 2: printSorted(courses) call function to print sorted class list

Case 3: printCourseInformation(courseId)

Case 4: Terminate Program

**Print Sorted List:**

**//Vector**

Create sorted print method printSorted(courses)

Create partition method int partition(vector<Course>& courses, int begin, int end)

Set lowIndex to first element, set highIndex to last element

Set midpoint to lowIndex + (highIndex - lowIndex) / 2

Set pivot to midpoint

Decrement highIndex while pivot is less than highIndex

Swap lower values to left of pivot, higher values to right of pivot

Set temp value to low index

Set low index to high index

Set high index to temp

Create quicksort method void quickSort(vector<Course>& courses, int begin, int end)

Set mid to 0, lowIndex to being, highIndex to end

If begin >= end, return

Set lowEndIndex to partition(courses, lowIndex, highIndex)

Make recursive call to quicksort

quickSort(courses, lowIndex, lowEndIndex);

quickSort(courses, lowEndIndex + 1, highIndex)

Create display course method void displayCourse(Course course) {

cout << course.courseId << ": " << course.name << " | " << course.prereq << endl;

Loop through vector to display courses

for (int i = 0; i < courses.size(); ++i)

displayCourse(courses[i])//Tree

Create inOrder method void BinarySearchTree::inOrder(Node\* node)

If (node != Nul)

Check most left side first

inOrder(node->left)

cout << course.courseId << ": " << course.name << " | " << course.prereq << endl;

check next right leaf

inOrder(node->right)

cout << course.courseId << ": " << course.name << " | " << course.prereq << endl;

**Runtime Analysis for Reading the File and Creating Course Objects:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Vector** | **Line Cost** | **Number of Times Executed** | **Total Cost** |
| **Create Vector** | **1** | **1** | **1** |
| **For each line in file** | **1** | **n** | **n** |
| **Create vector course item** | **1** | **n** | **N** |
| **While prereq exists** | **1** | **n** | **n** |
| **Append prereq** | **1** | **n** | **n** |
| **Pushback course item** | **1** | **N** | **N** |

**Total Cost: 5n + 1**

**Runtime: O(n)**

|  |  |  |  |
| --- | --- | --- | --- |
| **HashTable** | **Line Cost** | **Number of Times Executed** | **Total Cost** |
| **Create hash table** | **1** | **1** | **1** |
| **Insert method** | **0** | **0** | **0** |
| **Create key for course** | **1** | **n** | **n** |
| **If no entry found for key** | **1** | **n** | **n** |
| **Assign node to key** | **1** | **n** | **n** |
| **Else** | **1** | **n** | **n** |
| **Assisgn old node key to UNIT\_MAX, set to key, set old node to course and old node next to null pointer** | **4** | **n** | **4n** |
| **else** | **1** | **n** | **n** |
| **Find next open node** | **1** | **n** | **n** |
| **Add new newNode to end** | **1** | **n** | **n** |
| **For each line in file** | **1** | **n** | **n** |
| **Create vector course item** | **1** | **n** | **n** |
| **While prereq exists** | **1** | **n** | **n** |
| **Append prereq** | **1** | **n** | **n** |
| **Insert course item** | **1** | **n** | **n** |

**Total cost: 16n + 1**

**Runtime: O(n)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tree** | **Line cost** | **Number of Times Executed** | **Total Cost** |
| **Add node method** | **0** | **0** | **0** |
| **If root is null, add root** | **1** | **1** | **1** |
| **If node is less than root add to left** | **1** | **n** | **n** |
| **If no left node** | **1** | **n** | **n** |
| **This node becomes left** | **1** | **n** | **n** |
| **If node is greater than root add to right** | **1** | **n** | **n** |
| **This node becomes right** | **1** | **n** | **n** |
| **For each line in file** | **1** | **n** | **n** |
| **Create vector course item** | **1** | **n** | **n** |
| **While prereq exists** | **1** | **n** | **n** |
| **Append prereq** | **1** | **n** | **n** |
| **Insert course item** | **1** | **n** | **n** |

**Total Cost: 11n + 2**

**Runtime: O(n)**

Because each data structure is different, it has a set of advantages and disadvantages that depend on what the project is. When using vector sorting, it can prove difficult because the search must be based on a specific course, making the list check until said course is found. Vector sorting methods excel when reading files and adding course objects. The vector sorting method is straightforward and had the fastest performance.

Hash tables differ from vector sorting because of their speediness. Hash tables create a key that makes it easy to locate and print specific locations. The process of creating the key can become quite tedious and longsome, making that one of the disadvantages. Furthermore, hash tables cannot sort tables out itself, to print a sorted table out, the user must extract, sort, and print all the data in the table themselves.

In comparison, binary trees outperform vectors due to their quick sorting abilities. Although not as straightforward as hash tables, binary trees provide a faster alternative to vectors. The search time in a binary tree is represented as O(h), where h denotes the height of the tree.

Considering the project requirements, I recommend using a vector and implementing a sorting algorithm. The ability to rapidly sort and print the entire catalogue outweighs the slightly slower search times. In my opinion, the vector data structure proves to be the best option.