Jayden Cooley

Project One

12/2/2023

Vector Data Structure Pseudocode

Reading Data from File:

For i = 0, having i being less than the files row count, and incrementing i

courseNumber = file[i][1]

courseName = file [i][0]

If more information is available

preReqCourse = the course number in that spot

Creating Object Points:

Initialize Course Vector vector<int> courseInfo()

Loop through the file

CurrentItem = the first item

While not at last item

output course name and number for

prerequisites

print prerequisites until new line

output new line

currentItem = the next item in the list

Search Data Structure:

Ask for Input

Loop through vector

If input is the same as courseNumber

return course name, number, and prerequisites

Hash Table Data Structure Pseudocode

Course Structure

courseName;

courseNumber;

preReqCourse;

Const signed int DEFAULT\_SIZE = 8;

hashTable Class

Structure Node

Course\* course;

Node\*next;

Not assigned int key;

Node()///default

Node(Course course, unsigned int key) // both course and course key

Unsigned int size = DEFAULT\_SIZE;

hashTable();

vectorNode>nodes;

insert(Course course);

int numPreReqCourses(hashtableCourse> courses, string courseNumber)

Create key hashing course.Number

Retrieve node using key set to new node

WHILE node isn’t same as nullptr

IF node pointer course courseCourse equals numberNum

totalPreReq equals node.PreReq.size();

FOR each PreReq p in totalPreReq, add p’s PreReq to totalPreReq:

PRINT total number of PreReq

ELSE

Node equals node pointer next

void printCourseInfo(hashtableCourse> course, String

courseNum)

creates key by hashing course.Use key to get number from node and set to new node.

WHILE node isn’t equal to nullptr

IF node pointer course courseCourse equals numberNum

print out details about each course's prerequisites.

ELSE

node equals node pointer the next

unsigned int hash(int key)

key % return table size

VOID insert(course course)

hash(stoi(course.courseID) == unsigned key

check to see if node is empty.

If empty, add course to node.

IF run until an empty node is found

IF node isn’t empty

Course to be put at that place

Course parseLine(vector<string> &line)

IF (number of lines is two)

course newCourse

course courseName equal line 0

course courseNum matches line 1

set course requirements to empty vector

return a new one.

ELSE,

Vectorstring> will be used. tempPreReq

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

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

New courseCourse

course courseName equal line 0

course courseNum matches line 1

set course condition to tempBeforePreReq vector can return newCourse

Int main()

hashTable\* = hashTable()

Vector<string>temp

String: Ifstream infile("file name")

WHILE(getline(infile, line))

Stringstream ss(lie)

WHILE(ss.good())

String substr getline(ss, substr, ',')

temp.push\_back(substr)

table.insert(parseLine(temp))

temporary clear()

Tree Data Structure Pseudocode

Reading File:

Use fstream to open file

Make call to open file, if the return value is -1 then file isn’t found

Else file found

While it isn’t EOF

Read each line

IF there are less than two values in a line, return ERROR

ELSE read parameters

IF there is a 3rd or more parameter

IF 3rd or more parameter is in 1st parameter elsewhere continue

ELSE return Error

Close file

Create Course Objects Structure:

Initialize Course Structure struct Course

Loop through file, While not EOF

For each line in file

For 1st and 2nd value

Add course ID and Course Name

If a 3rd value exists

Add preReq until newline found

Create Tree and add Nodes:

Define Binary Tree Class

Create root that points to null

Create Insert method

If Root is Null, current Course is Root

Else if courseNum less than root, add left

If left is null, add courseNum

Else

if courseNum less than leaf add left

if courseNum greater than leaf add right

Else if courseNum greater than root, add right

If right is null, add courseNum

Else

if courseNum less than leaf add left

if courseNum greater than leaf add right

Search and Print from Tree:

Ask for Input

Create Print Method

If root is not null

Traverse left, output if found

Traverse Right, output if found

Search and Print from Tree:

Ask for Input

Create Print Method

If root isn’t null

Traverse left, output if found

Traverse Right, output if found

Menu Pseudocode

Create integer for switch statement, set to 0, name uInput

Create bid variable to access

WHILE input doesn’t equal 4;

PRINT 1: Load Data Structure

PRINT 2: Course List;

PRINT 3: Course;

PRINT 4: EXIT

SWITCH (uInput):

Case 1:

loadBids(bid);

break;

case 2:

Print: Course List

Break;

Case 3:

Print: “Thank You!”;

End program

Break;

Default:

Print: no input from user;

Break;

CS Course List Pseudocode

Create sorting string with parameter string s

Create char that sets length +1

Create string to character array

Sort array

Create 2 integers for alphabet and numbers

Create loop for alphabet integer if less than 97

Set alphabet int to +1

IF i is less than 97 set number int to +1

ELSE set alphabet int to +1

Return s

DRIVER

Create string for classes

PRINT courses in alpha order

With the advantages of each structure, creating a vector is one-dimensional which has handling data for this assignment to be a good thing, considering we’re only dealing with reading files and courses. There’s also less memory when the data structure is utilized, having more being able to be run in the project. A major disadvantage is that elements can’t be deleted, and main elements wouldn’t be able to have multiple data types.

Creating hash tables for this project has allowed much information to be organized and stored, having it being called through the project with a key. This allows creating, deleting, and calling within the project itself, allowing unique elements and synchronization among the project. This sadly comes at a cost as using hash tables effect speeds of synchronization among the entire project.

Creating a tree allows for way better organization, with an example being storing information on a left or right said due to certain requirements that can be expanded accordingly. With this, you’re able to run searches throughout the project easily but does end up slowing down the project that can make the project take longer to allow modifications.

Out of the three data structures, I think hash tables are the best to go with despite the disadvantage of running slower. There are many advantages to hash tables with it majorly being about organization. Being able to sort courses as well as being able to updating the project itself at a moment’s notice really helps with creating the project as a whole, getting it done much better than without it.