Module 6 Project 1

CS-300

SNHU

Drew Heard

Start

/////////////////////////////////////Vector

struct Course //course structure

string number

string title

preq = new List // list of course prerequisite

Course ()//default constructor

number = 0

title = empty

preq

preqCheck = False

Vector <course> CourseVect //create vector

LoadVector{

Ifstream info (filepath)

While (!info.eof()){

Course course // new course created

temp = getline

course number = temp.substr(0, temp.find(','))//add number to course structure

temp.erase(0, course number.size() + 2)

if (temp.size() > 0) {

course title = temp.substr(0, temp.find(','))//add name to course structure

temp.erase(0, course number.size() + 2)

int i = 0

while (temp.size() > 0)

string temppreq = temp.substr(0, temp.find(',')) //add preq to course structure

course preq.push\_back(temppreq)

temp.erase(0, temppreq.size() + 2)

i +=1

if ( course.number is 0 OR course.name is "empty"){ // check for valid entry

course.number = null

course.title = null // remove bad entries

else

CourseVect push back(course)

//preq check

for ( i=0, i< CourseVect.size), i++)

for (j=0, j < size of preq in CourseVect at i, j++) //

temp = searchVect for course(i).preq(j)

if (temp is null)

delete course(i)preq(j)

sort(CourseVect begin course number and end course number)

int size equals CourseVect.size() // vector size

} end LoadVector

course searchVect(CourseVect, number)

for (i=0, i < size, i++) // compare all numbers in CourseVect with number

if (CourseVect.at(i).number == number) //return if they match

return it

else

return null

void print(CourseVect)

course temp=searchVect(CourseVect, number)

print temp.number and temp.name

for(i=0, i<temp.preq.GetLength(preq), i++)

print temp.preq.at(i)

return

/////////////////////////////////////////////Hash Table

Create structures, variables, and class

struct Course{//course structure

string number

string title

preq = new List; // list of course prerequisite

Course ()

number = 0

title = empty

list preq (0)

Hashtable class

Structure node

Int Key

Course

Node \*next

Key =MAX INT

Couse course

Next = NULL

Vector <course> hashVector //create vector for Hash keys

Int tableSize =10

Hash()

Print()

Insert()

Hashtable (size){

tableSize = size

hashVector.resize(tableSize)

LoadHashtable{

Ifstream info (filepath)

While (!info.eof()){

Course course // new course created

new Node

temp = getline

course number = temp.substr(0, temp.find(','))//add number to course structure

temp.erase(0, course number.size() + 2)

if (temp.size() > 0) {

course title = temp.substr(0, temp.find(','))//add name to course structure

temp.erase(0, course number.size() + 2)

int i = 0

while (temp.size() > 0)

string temppreq = temp.substr(0, temp.find(',')) //add preq to course structure

course preq.push\_back(temppreq)

temp.erase(0, temppreq.size() + 2)

i +=1

key = hash number

if ( course.number is 0 OR course.name is "empty"){ // check for valid entry

course.number = null

course.title = null // remove bad entries

else

insert(course)

//preq check

for ( i=0, i< hashVector.size), i++)

Node node = (hashVector.at(i))

if (node->key != UINT\_MAX)

for (j=0, j < node course preq.size< j++)

temp = search(node course preq(j))

if (temp == null)

node course preq(j) = null

int size equals hashVector.size() // vector size

} end Load Hashtable

Hash(key)

Return calculated key (key%size)

Search(number)

key = hash(number)

Node\* node = (hashVector.at(key)

if ( (node != nullptr) AND (node->key != UINT\_MAX) AND (node's number == number))

return node->bid

if ((node == nullptr) || (node->key == UINT\_MAX))

return bid

Insert(course)

Key = hash course.number

Oldnode = hashVector(key)

If oldnode is null

Newnode = new node

Else if oldnode.key isn’t used

oldNode.key = key

oldNode.course = course

oldNode.next = nullptr

else

while oldnode isn’t null

oldnode = oldnode.next

oldnode.next = newnode

Remove(course)

key = hash(number))

hashVector.erase(hashVector.begin() + key)

}

Print() (print all)

For loop through Hash Vector

Print each node’ course data

If node has chained pointers

Print chained nodes course data

Print (course name) (search and print one course)

Create hashed key and new node for course name

If new node isn’t null AND key is used AND new node’s name == course name

Print node’s course information

If node is null OR key isn’t used

Print “Not found”

////////////////////////////////////////////////////////Binary Tree

Create structures, variables, and class

struct Course //course structure

string number

string title

preq = new List // list of course prerequisite

Course ()

number = 0

title = empty

list preq (0)

BinaryTree class

Structure node

Course

Node \*next

Couse course

Left is NULL

Right is NULL

Node root ptr

Print()

InsertNode()

RemoveNode()

Search()

Open and read file from given file path

LoadTree{

Ifstream info (filepath)

While (!info.eof()){

Course course // new course created

new Node

temp = getline

course number = temp.substr(0, temp.find(','))//add number to course structure

temp.erase(0, course number.size() + 2)

if (temp.size() > 0) {

course title = temp.substr(0, temp.find(','))//add name to course structure

temp.erase(0, course number.size() + 2)

int i = 0

while (temp.size() > 0)

string temppreq = temp.substr(0, temp.find(',')) //add preq to course structure

course preq.push\_back(temppreq)

temp.erase(0, temppreq.size() + 2)

i +=1

key = hash number

if ( course.number is 0 OR course.name is "empty"){ // check for valid entry

course.number = null

course.title = null // remove bad entries

else

InsertNode(node, number)

//preq check

Start at root

If node isn’t null

Recurse left child back into print

temp = search(node course preq(j))

if (temp == null)

node course preq(j) = null

Recurse right child back into print

} end LoadTree

InsertNode(Node, course number)

If node’s course number is greater than bid’s course number

If left child is null

Add node as left child

Else recurse left child back into insertNode

Else

If right child is null

Add node as right child

Else recurse right child back into insertNode

RemoveNode(root, coursenumber)

If node is null return it

Else

if coursenumber < current coursenumber recurse down the left subtree

if coursenumber > current coursenumber recurse down the right subtree

else

if there are no kids, delete the node

if there is one child, replace the node with the child

else

reallocate kids to right child’s left sub tree

Search(coursenumber)

Current node equals root

While current isn’t null

If current course number and target course number equal, return current node

if course number is smaller than current node’s, then traverse left

if course number is greater than current node’s, then traverse right

return course

Print() (print all in order)

Start at root

If node isn’t null

Recurse left child back into print

Print node information

Recurse right child back into print

////////////////// menu

print request for file path

filepath = user input

print request for course number

targetNumber = user input

uin = 0

while uin isn't 9

Print menu

print "1 for load file into structure"

print "2 for print list"

print "3 for print course"

Print "9 for exit"

uin = user input

switch (uin)

case 1

load data structure(filepath)

break

case 2

print()

break

case 3

search(targetNumber)

print course info

break

case 9

break

End

|  |  |  |  |
| --- | --- | --- | --- |
| **Vector Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **Loading file** | 26 | n | n |
| **Preq Check** | 5 | n^2 | n^2 |
| **search** | 3 | n | n |
| **print** | 4 | n | n |
| **Total Cost** | | | 3n + n^2 |
| **Runtime** | | | O(n^2) |
| **Table Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **Loading file** | 29 | n | n |
| **Preq Check** | 7 | n^2 | n^2 |
| **search** | 3 | n | n |
| **Hash** | 1 | 1 | 1 |
| **Insert** | 12 | n | n |
| **Remove** | 2 | 1 | 1 |
| **Print all** | 4 | n | n |
| **Print course** | 5 | 1 | 1 |
| **Total Cost** | | | 5n + n^2 + 3 |
| **Runtime** | | | O(n^2) |
| **Tree Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **Loading file** | 26 | n | n |
| **Preq Check** | 7 | 1 | 1 |
| **search** | 6 | n | n |
| **print** | 5 | n | n |
| **Insert** | 8 | 1 | 1 |
| **Remove** | 9 | 1 | 1 |
|  |  |  |  |
| **Total Cost** | | | 3n + 3 |
| **Runtime** | | | O(n) |