Project One

Sergio Mateos

Southern New Hampshire University

**Vector Data Structure**

struct ClassCourse

   string courseNumber

   string nameCourse

   vector<string> prerequisites

vector<Course> LoadDataStructure(){

   ifstream fin("abcd.txt")

   vector<Course> courses.

   string line.

   while (1)

       getline(fin,line)

       If the end is reached break it

           break

       Course course

       vector<string> tokenizedInformation=tokenize(line,",")

       course.courseNumber=tokenizedInformation[0]

       course.name =tokenizedInformation[1]

       For-Loop to store if there are prerequisites

           course.prerequisites.push\_back(tokenizedInformation[i]

       courses.push\_back(course)

   fin.close();

   return the list of courses

void printCourse(Course course)

   string courseNumber= course.courseNumber

   string name=course.name

   vector<string> prerequisites=course.prerequisites

Print Course Name

   For-Loop to look for the prerequisites

       Print the prerequisites

void printCourseList(vector<Course> courses) {

   int n=courses.size()

   For-Loop to sort the list

       For-Loop to sort the list

           If courses[j].courseNumber > courses[j+1].courseNumber

               Swap the courses

   For loop for all the courses information {

       Print Course Information

void searchCourse(vector<Course> courses) {

   int n=courses.size()

   string courseNumber

   int f=0

   Print to get the Course Number

   Cin Course Number

   For loop to look for similar course number

       if course found then print it

if course with given course name not found then print error message

**Print on alphabetical order**

unsigned int alphOrder

for-loop for each element on the size of courses

alphOrder is equal to the number of the element

for-loop starting at j which is i+1 on the size of courses

if course at j is less than courses at i

alph = j

if alph is nor the i

swap i to alph

for-loop to print the courses in alphabetical order

print courses in alphabetical order

**Menu**

Print “1. Load Data”

Print “2. Couse List”

Print “3. Course”

Print “9. Exit”

Print “Make your selection”

while choice is not NULL

switch(choice)

case 1

Load data course data

break

case 2

Print Course List

break

case 3

Print Course

break

case 9

Print “Bye bye”

Program end

Break

default

print “Invalid Input”

break

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Open file** | 1 | 1 | 1 |
| **Load courses** | 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** | | | 3n + 1 |
| **Runtime** | | | O(n) |

**Hash Data Structure**

Course

String courseName

String CourseNum

Vector<string>prerequisites

CONST unsegment in SIZE = 8

Course HashTable

Struct Node

Course\* course

Node\* next

unsigned int key

Node()

Node(Course course, unisigned int key)

Unsigned int size = SIZE

Vector<Node>nodes

HashTable()

Add(Course course)

Void insertCourse(Course course)

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

if node empty

insert course at node

if node is not empty

while until node found

insert course at node

int numPrerequisiteCourses(Hashtable<Course> courses string courseNum)

key hashing the courseNum

retrieve node setting new node with the key

while node is not equal to null

if node pointer courseNum is equal to courseNum

totalPrerequisites = node.Prerequisites.size()

for each prerequisite p in totalPrerequisites

totalPrerequites = prerequisites + p

else

node equal node pointer next

void printCourseInformation(Hashtable<Course> courses, String courseNum)

key hashing the courseNum

retrieve node using key and set it to new node

while node is not equal to null

if node pointer courseNum = to courseNum

print prerequisite course info

else

node = node pointer next

unsigned int hash(int key)

return key percentage table size

void printSampleSchedule(Hashtable<Course> &courses)

if lineSize = two

course newCourse

course courseName = line[0]

course courseNumber = line[1]

set course prerequisites to empty tray

return newCourse

else

int i

Vector<string> sampleSchedule

for i = 2; i < line.size(); i++

sampleSchedule.push\_back(line[i])

course newCourse

course courseName = line[0]

course courseNumber = line[1]

set course prerequisites to sample Schedule

return newCourse

int main

HashTable\* table = HashTable()

Vector<strin> temp

String line

//Open file

ifstream infile(“name”)

while getline in name

string stream ss(line)

while ss.redable

string substr

getline(ss, substr, ‘, ‘)

table.insert(printSampleSchedule(temp))

temp clear

**Print on alphabetical order**

int inOrder(Hashtable<Courses> courses, string begin, string end)

low = begin

high = end

string mid

node = new node

while node is not NULL

for-loop for each course-on-course size

while low is less than

if low is less than high

low = low

else low is greater than high

low = mid

mid = high

high = low

for-loop to each cours on Course

print Courses on alphabetical order

**Menu**

Print “1. Load Data”

Print “2. Couse List”

Print “3. Course”

Print “9. Exit”

Print “Make your selection”

while choice is not NULL

switch(choice)

case 1

Load data course data

break

case 2

Print Course List

break

case 3

Print Course

break

case 9

Print “Bye bye”

Program end

Break

default

print “Invalid Input”

break

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Open file** | 1 | n | n |
| **Read each course line** | 1 | n | n |
| **Load Requirement** | 1 | 1 | 1 |
| **Increase counter** | 1 | n | n |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 1 | n | n |
| **Total Cost** | | | 5n + 1 |
| **Runtime** | | | O(n) |

**Binary Tree Structure**

Load text libraries and headers

Struct to hold course data

Struct Course

courseId

courseName

preCount

preList

Course() contruct

courseId = courseName = “”

preCount = 0

preList = “”

Class BinaryTree

Struct Node

Course

left pointer

right pointer

root

numPrerequisiteCourses

printSampleSchedule

printSampleSchedule

int numPrerequisiteCourses(Tree<Course> courses)

open file at the path string

while row until end of file

if first and second strings are present

add the first string at courseId

add the second string at courseName

while until file has no value

increase variable preCount for each prerequisite

concatenate localstring named preNames for each prerequisite

add preCount to struct at preCount

add preName to struct at preList

return tempList

void printSampleSchedule(Tree<Course> courses)

create tempCourse of type Node

set tempCourse to the bucket at the hash location to Sting

loop for search for each course

if string is the same as courseId

set tempCourse to Course

return temCOurse

void printCourseInformation(Tree<Course> courses, String courseNumber) create tempCourses of type bucket

tempCourse = to root

while loop tempCourses is NULL

if the Node at tempCourse contains a bidId equal to string

print coursesId in Course found in tempCourses

print courseName in Course found in tempCourses

Loop to count count from 0 to preCount

For each Course in preList

Print passing preList

if the Node at tempCourses contains courseId less to string

set tempCourses = to the left Node

if the Node at tempCourses contains courseId less to string

set tempCourses = to the right Node

int main

create new BinaryTree courseTree of the struct Course

get file path from user

if not pass default location

call numPrerequisiteCourses

call printSampleSchedule

get the userSearch for courseId

call printCourseInformation

**Printing on alphabetical order**

inOrder Node\* node

if node is not nullptr

call inOrder starting from left node in recursive order

call inOrder starting from right node in recursive order

sort courses from low to high from pointer

print inOrder list

**Menu**

Print “1. Load Data”

Print “2. Couse List”

Print “3. Course”

Print “9. Exit”

Print “Make your selection”

while choice is not NULL

switch(choice)

case 1

Load data course data

break

case 2

Print Course List

break

case 3

Print Course

break

case 9

Print “Bye bye”

Program end

Break

default

print “Invalid Input”

break

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Define Node Structure** | 1 | 1 | 1 |
| **Loop for prerequisites** | 1 | n | n |
| **courseId = string** | 1 | n | n |
| **corseName = string** | 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** | | | 6n + 2 |
| **Runtime** | | | O(n) |

**Evaluation**

Each Structure has different ways to store data and all of them will have their advantage and disadvantages.

Vector Data Structure is the simplest to develop on code and quick to understand, but it’s slower at searching for courses O(n). First, the needs to be sorted the array to print the list of courses in order at O(n log n). Another disadvantage, you can only add courses when you are loading the data file and if you need to fix the data it’s hard for to that happen.

Hash Data Structure is more confusing in terms of code and development compared to vector data, but the implementation is significantly more efficient compared to vector, and inserting would be faster in the best and worst cases. The speed of searching is fast for the best and worst cases (O(n) and O(1)). But whenever you try to print in order, Hash Data Structure would be slower than Vector Data Structure since Hash data require checking all the buckets and looking for the nodes in the buckets.

Binary Tree Data Structure overall this confusing on the design of the code, but it will be categorized in the middle since if we compare the searching speed it would be faster than Vector Data Structure in the best and worst case, but it will be relatively slower than Hash Data Structure, that would also be applied on the insertion. But sorting and printing a list in order is the fastest.

**Recommendation**

Each data structure is unique and effective based on the situation or requirements. In our scenario, Vector Data Structure would be the best option given the small amount of data that the program will be dealing with. The simplicity of the code and the easier of development would be the best option. But if the amount of data increases Binary Tree Data Structure would be the best option.