Albert Chapman

Project 1

struct Course(){

String courseNumber

String name

Vector<string> prerequisites

Course(){

courseNumber null

String null

Prerequisits null

}

Struct Node{

Course course

Node \*left

Node \*right

Node(){

left equals null

right equals null

}

Node(Course aCourse) :

Node(){

Course equals aCourse

}

}

Class HashTable{

Private:

Struct Node{

Course course

Unsigned int key

Node \*next

Node(){

Key = Max

Next = NULL

}

Node(Course aCourse) : Node(){

Course = aCourse

}

}

Vector <Course> courses

Unsigned int tableSize = default

Unsinged int hash(int key)

Public:

HashTable()

Void searchCourse()

}

Class Tree{

Private:

Node\* root;

Public:

Node searchCourse(Tree<Courses>, string courseNumber)

Void printCourse(node)

Void printAlphanumaric()

}

Void printCourse(Node node){

Cout node course’s courseId, name

for loop for prerequisites

}

Void printAlphanumaric(){

Node node equals root

If the current node is not null

Recurse to the left

printCourse(node)

recurse to the right

}

Node searchCourse(Tree<Course> courses, String courseNumber) {

Set a temp Node to the root of the Tree

While temp doesn’t equal a null pointer

If the temp node’s course number equals the searched course number

Return the temp nodes course information

If the temp nodes course number is greater than the searched course number

Shift the temp node to the left node

Else shift the temp node to the right

All else fails, return an empty bid

}

Void addCourse(Node\* node, Course course){

If the root is null

Root is equal to new node course

Elseif the courseID to the new node is larger than current node

If there is no left node

The new left node is this course

Else

Recurse down the left node

Else

If there is no right node

Set the right node to this course

Else

Recurse down the right side

}

Void readFile(Tree& tree){

String courseNumber()

Ifstream input

Open the file in input

If the file doesn’t open

Print error messege and return

String line, courseNumber,name,check

vector<string> prerequisites

while a line doesn’t fail, get the line and set it to line{

stringstream the line

int column index equal to zero

while there are variables in the string stream{

if column index is zero and the cin is valid

cin courseNumber

elseif column index is 1

cin name

elseif column index is greater than 1{

cin check

if check is found in any courses.courseName

prerequisits.push\_back() check

}

Increment column index

}

If column index is at least 2{

Course newCourse(courseName,name,prerequisists)

Tree.addNode(newCourse)

}

}

close the file

}

Void main{

Tree\* tree

tree new Tree

Switch statement

Case1:

readFile(tree)

case2:

Tree.printAlphanumaric()

Case3:

Tree.printNode(Tree.searchCourses(cin)

Case9:

return

}

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line cost | Times executed | Total cost |
| String CourseNumber | 1 | n | n |
| ifStrem input | 1 | n | n |
| Open file in input | 1 | n | n |
| If the file doesn’t open | 1 | n | n |
| String line courseNumber name check | 1 | n | n |
| Vector prerequisites | 1 | n | n |
| While a line doesn’t fail | 1 | n | n |
| Stringstream the line | 1 | n | n |
| Int column index to zero | 1 | n | n |
| While there are variables in the stream | 1 | 3 | 3n |
| Column index zero | 1 | n | n |
| Cin course number | 1 | n | n |
| If column index is 1 | 1 | n | n |
| Cin name | 1 | n | n |
| If the column index is greater than 1 | 1 | n | n |
| Input into the prerequist vecor | 1 | n | n |
| Increment column value | 1 | n | n |
| If column index was at least 2 | 1 | n | n |
| New course | 1 | n | n |
| Tree.addNode(newCourse) | 1 | n | n |
| If the root is null | 1 | n | n |
| If the course Id is greater | 1 | n | n |
| If there is no left node | 1 | n | n |
| Recurve down left node | 1 | N log n | n log n |
| If there is no left node | 1 | n | n |
| Insert in this course | 1 | n | n |
| Close the file | 1 | n | n |
|  | | Total Cost | 26n+3n+nlogn |
|  | | Runtime | O(nlogn) |

The Vector data structure holds each course object in an array as they are iterated through during the file reading in the order they are put in. Searching and inserting new elements into a vector is simple. Searching can possibly be time consuming if the course is at the end of the vector.The hash tree puts the courses in an array that are ordered by the value of the key value they produce based on the course id and the key on the hashmap. With a key, the right data can be accessed directly in a best case scenario. However, if there is multiple elements at the same key, called collisions, the same problems with a normal vector would arise. The binary search tree creates a node network based on the course id being larger or smaller than the first node that is put in. This makes searching for a specific course efficient, but increases the time to insert a new course.

Based on these things, I believe that a hashtree would be best as it would decrease search and insertion time for users. Even though there is the possibility of running into collisions, the dataset should be small enough that even if a chain is needed for a key, it should be rather short. The chain will at least be shorter than the worst case scenario for both the normal vector and binary search tree.