Elliot Huh  
CS-300

Project 1

1. **Pseudocode Resubmissions**

***Vectors***

***Pseudocode to open file, reads data, parses each line, check file format errors***

//Declare Variables

String CourseNumber

String CourseName

String Prerequisite

Declare array for courses

//open file

Initialize ifstream

Open course file with ifstream

//Check if file exists

If file not found

Output file not found

Else

//if file is found

While file hasn’t ended

//read data

Get lines

//parse lines

For each line

If there is a comma

//Create parameters

Split line at the comma

Assign CourseNumber to first parameter

Assign CourseName to second parameter

If there is more than 2 parameters

Assign Prerequisite to every parameter after 2nd

Append line to course array

//Ensure there are at least two parameters

//Also takes addresses lines without commas

If course has less than two parameters

Remove line from array

Output: Course has error formatting

Continue

For all courses

If course has a prerequisite

For each prerequisite

//check if prerequisite can be found within the array

Loop through file

If prerequisite equals courseNumber

//prerequisite has been validated and no changes are made

Continue

//If prerequisite cannot be found within the array

Else

//Remove parameter from line

Remove Prerequisite from course

Output: Prerequisite cannot be found

END

***Pseudocode to create objects and store them***

//Structure of courses

Struct Course

String CourseNumber

String CourseName

String Prerequisite

Initialize vector

Open course file with ifstream

Get line

If there is a comma

Split line at comma

//Create object

Assign CourseNumber to first parameter

Assign CourseName to second parameter

//Check for prerequisites

If there is more than 2 parameters

For each parameter after the second

Assign prerequisite to every parameter after the 2nd

//create object and append to vector

Create new course object (CourseNumber, CourseName, Prerequisite)

Append to vector

//if there is no prerequisites

Else

//Create object and append to vector

Create new course Object (CourseNumber, CourseName)

Append to vector

END

***Pseudocode search the data structure then print out course info with prerequisites***

Input desired CourseNumber

//Loop through the vector

For all courses

//If inputted courseNumber exists within vector

If user input equal CourseNumber

//Print out course info

Output: CourseNumber with CourseName

If prerequisite exists for desired course

For each prerequisite of the course

//If prerequisite exists within vector

If prerequisite equals courseName

Output: CourseName with course information

//If prerequisite wasn’t found

Else

Output: CourseName Prerequisite doesn’t exist

Else Output: Course not found.

END

***Hash Table***

***Pseudocode to open file, read, parse and check for format errors.***

//initialize hashtable

Class hashtable

TableSize equals 100

Struct Course

CourseNumber

CourseName

Prerequisite

Course pointer next

//constructor

Hashtable::Hashtable

Resize TableSize

//open File

Initialize ifstream

Open course file

If value returns -1

Output: file not found

End

Else if file cannot be opened

Output: file cannot be opened

End

Else If file has been found and can open

For each line on file

Split line at comma

If there is less than 2 parameters

Output: invalid format

//do not add line to hashtable

Continue

//if format is valid

Else

Assign courseNumber to first parameter

Assign courseName to second parameter

If there is more than 2 parameters

Assign each parameter as prerequisite

Add course into hash table

//check if prerequisite exists

For each course in hash table

If prerequisite exists

For each prerequisite

//search for the CourseNumber that matches the prerequisite

Search for matching CourseNumber with the key

If CourseNumber exists within the Hashtable

//format has been verified continue

Continue

//if Prerequisite doesn’t exist

Else

Erase prerequisite from course

Continue

END

***Pseudocode to show how course object is created and stored.***

//Structure to store course info

Declare tableSize

Struct Course

CourseNumber

CourseInfo

Prerequisite

//Structure to hold courses

Struct Node

Bid bid

Key

Node pointer next

Hash

Return key % tableSize

Open course file

Try

For each line in file

Bid bid

Split each line by comma

courseNumber equals file parameter [0]

courseInfo equals file parameter [1]

Prerequisite equals all file parameter [2 and over]

Course equals new Course (courseNumber, courseName, Prerequisite)

//hash the object

Create key for given bid

Retrieve node from key

//If the node from the key is empty append object to hashtable

If the node at the key is empty

Add object to the node

//Utilizing Linear probing for collision handling

//Find next open node if node at key is used

Begin at first node

While node is not empty

Move to next node

Add object to the next free node with the new key

Catch

Output: error

***Pseudocode to show how print course info and prerequisites.***

Hash table searchCourse

Input CourseNumber

Create key for given CourseNumber

//Search for the course

Retrieve node from the key

If node is full and matches CourseNumber

Return node

Output: CourseNumber and CourseName

//If node is empty or if node does not match the Inputted CourseNumber

Else

While hashtable has not ended

//If course

If node matches CourseNumber

Return node

Output: CourseNumber and CourseName

Node equals next

If no node was returned

Output: Course was not found

Hash table PrerequisiteSearch

If inputted CourseNumber has prerequisite courses

Output: Prerequisite courses:

For each prerequisite course

Call to searchCourse function using prerequisite as inputted CourseNumber

END

***Binary tree***

***Pseudocode to open, parse and check for format errors***

Struct course

courseName

courseNumber

preRequisite

ReadFile

Initialize ifstream

Open course file

If file not found

Output: error file not found

Else

//Temporary structure to process the file by chunks instead of looping through its entirety

Declare temp as temporary array

While file has not ended

Add up to 5 lines from the file to temp array

If line has already been processed

Skip line and move onto the new line

For each line in temp array

If there is a comma

Split line at comma

If line has less than 2 parameters

Remove line

Output: error formatting

Else

Assign courseName to 1st parameter

Assign courseNumber to 2nd parameter

If there is more than 2 parameters

For each parameter after 2nd

Assign prerequisite to every parameter after 2nd

For each prerequisite

Loop through file

If Prerequisite exists within file

Continue

Else

Remove prerequisite

Output this prerequiste did not exist

Else

Continue

Add course to the Binary tree end

Delete Temp array

END

***Design pseudocode to show how to create course object***

InOrderTraversal

Recur on left subtree

Output: course

Recur on right subtree

PostOrderTraversal

Recur on left subtree

Recur on right subtree

Output: course

PreOrderTraversal

Out: course

Recur left subtree

Recur right subtree

//structure to hold course info

Struct course

courseName

courseNumber

preRequisite

//Structure for tree nodes

Struct Node

Course course

Node pointer left

Node pointer right

createObject

Initialize ifstream

Open file

//read each line

For each line

Create object

Split line at comma

Assign courseName to 1st parameter

Assign courseNumber to 2nd parameter

If there is more than 2 parameter

Assign prerequisite For each parameter after 2nd

Course course create object (courseName, courseNumber, prerequisite)

AddNode

Initialize ifstream

Open file

//add course to the binary tree

For each course

If course is larger than add to the left

If left node is empty

Course becomes that node

Else

Add a new node to the left

Course becomes that node

Else

If right node is empty

Course becomes that node

Else

Add a new node to the right

Course becomes that node

***Pseudocode that will print out course info and prerequisite***

SearchNode

Bool exist equals false

Node pointer current equals node

While current is not null pointer

If current equals current course

Return current course

Exists equals true

Else if smaller than current node

Traverse left

Return course

Exists equals true

Else if larger than current node

Traverse right

Return course

Exists equals true

//if course is not found

Else

Output: course has not been found

userInput

Get user input for courseName

Call to searchNode function using user input

If exist equals true

Output: courseNumber, courseName

If course has prerequsite

For each prerequsite

//call to searchNode function using prerequisite as courseNumber

course equals prerequsite

Call to searchNode function

If exists equals true

Output: prerequsite, courseName

1. Menu Pseudocode

Struct Course {

String CourseNumber

String CourseName

String Prerequisite

}

**LoadVectorFile {**

Initialize ifstream

Open course file

Initialize vector array

If file is found and can open

For each line

Split line at comma

Assign CourseNumber to first parameter

Assign CourseName to Second parameter

If there is more than two parameters

Assign Prerequisite to every parameter after the second

Course create new course object (CourseNumber, CourseName, Prerequsite)

If there is less than two parameters

Remove object

If course has prerequisite

Loop through vector

If perquisite course isn’t found

Remove prerequisite from object

**LoadHashFile {**

Initialize ifstream

Open course file

Initialize vector array

If file is found and can open

For each line

Split line at comma

Assign CourseNumber to first parameter

Assign CourseName to Second parameter

If there is more than two parameters

If there is more than two parameters

Assign Prerequisite to every parameter after the second

Course create new course object (CourseNumber, CourseName, Prerequsite)

If there is less than two parameters

Remove object

Create key for given course

Retrieve node from key

//If the node from the key is empty append object to hashtable

If the node at the key is empty

Add object to the node

//Utilizing Linear probing for collision handling

//Find next open node if node at key is used

Begin at first node

While node is not empty

Move to next node

Add object to the next free node with the new key

//check if prerequisite exists

Get key from prerequisite

Get node from the key

If the courseName doesn’t match with prerequisite

Remove object

**LoadTreeFile {**

**PrintVectorCouseList {**

Size equals len of course vector array

//Use sort function from #include algorithm

Sort course vector array, Size

Output: List of all courses

For each course

Output: CourseNumber : CourseName : Prerequisite

**PrintHashTableCourseList {**

//Convert Hashtable into vector array

Initialize course vector array

While node is not nullpointer

Append course into course vector array

Node equals next node

//After all nodes are converted use same vector sort strategy

Size equals len of course vector array

//Use sort function from #include algorithm

Sort course vector array, Size

Output: List of all courses

For each course

Output: CourseNumber : CourseName : Prerequisite

**PrintTreeCourseList {**

//In Order Traversal

If node is not nullpointer

Recur on left subtree

Output: CourseNumber : CourseName : Prerequisite

Recur on right subtree

**SearchVectorCourse {**

Input desired CourseNumber

//Loop through the vector

For all courses

//If inputted courseNumber exists within vector

If user input equal CourseNumber

//Print out course info

Output: CourseNumber with CourseName

If prerequisite exists for desired course

For each prerequisite of the course

//If prerequisite exists within vector

If prerequisite equals courseName

Output: CourseName with course information

//If prerequisite wasn’t found

Else

Output: CourseName Prerequisite doesn’t exist

Else Output: Course not found.

**SearchHashCourse {**

Hash table searchCourse

Input CourseNumber

Create key for given CourseNumber

//Search for the course

Retrieve node from the key

If node is full and matches CourseNumber

Return node

Output: CourseNumber and CourseName

//If node is empty or if node does not match the Inputted CourseNumber

Else

While hashtable has not ended

//If course

If node matches CourseNumber

Return node

Output: CourseNumber and CourseName

Node equals next

If no node was returned

Output: Course was not found

Hash table PrerequisiteSearch

If inputted CourseNumber has prerequisite courses

Output: Prerequisite courses:

For each prerequisite course

Call to searchCourse function using prerequisite as inputted CourseNumber

**SearchTreeCourse {**

**Menu {**

While user does not choose 4

Output:

Load file: 1

Print all courses: 2

Search a course: 3

Exit: 4

Input: Users choice

**//**The function being called to is dependent on the data structure chosen

If user chooses 1:

Output: Load file

Call to loadVectorfile or loadHashfile or loadTreefile

If user chooses 2:

Output: print all courses in numerical order

Call to Printvectorcourselist or printhashcourselist or printTreecourselist

If user chooses 3:

Output: Search a course

Call to Searchvectorcourse or searchhashCourse or SearchTreecourse

//Function to exit

If user chooses 4:

Output: Goodbye

Return 0

END

1. Alphanumeric Pseudocode

SortVectorList {

Size equals len of course vector array

//Use sort function from #include algorithm

Sort course vector array, Size

Output: List of all courses

For each course

Output: CourseNumber : CourseName : Prerequisite

SortHashList {

//Convert Hashtable into vector array

Initialize course vector array

While node is not nullpointer

Append course into course vector array

Node equals next node

//After all nodes are converted use same vector sort strategy

Size equals len of course vector array

//Use sort function from #include algorithm

Sort course vector array, Size

Output: List of all courses

For each course

Output: CourseNumber : CourseName : Prerequisite

SortTreeList {

//In Order Traversal

If node is not nullpointer

Recur on left subtree

Output: CourseNumber : CourseName : Prerequisite

Recur on right subtree

Evaluations

1. Evaluate run-time and memory of data structures.

Vector

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Declare array for courses** | 1 | 1 | 1 |
| **Initialize instream** | 1 | 1 | 1 |
| **Open file** | 1 | 1 | 1 |
| **If file not found** | 1 | n | n |
| **Else file is found** | 1 | n | n |
| **While file hasn’t ended** | 1 | n | n |
| **Get lines** | 1 | n | n |
| **For each line** | 1 | n | n |
| **If there is a comma** | 1 | n | n |
| **Split line at comma** | 1 | n | n |
| **Assign CourseNumber to first parameter** | 1 | n | n |
| **Assign CourseName to SecondParameter** | 1 | n | n |
| **If there is more than 2 parameters** | 1 | n | n |
| **Assign prerequisite to every parameter after 2nd** | 1 | n | n |
| **Append line to course Array** | 1 | n | n |
| **If course has less than two parameters** | 1 | n | n |
| **Remove Line from array** | 1 | n | n |
| **Output: Course has error formatting** | 1 | n | n |
| **For all courses** | 1 | n | n |
| **If course has prerequisite** | 1 | n | n |
| **For each prerequisite** | 1 | n | n |
| **Loop through file** | 1 | n | n |
| **If prerequisite equals CourseNumber** | 1 | n | n |
| **Else** | 1 | n | n |
| **Remove prerequisite from course** | 1 | n | n |
| **Output: Prerequisite cannot be found** | 1 | n | n |
| **Struct Course** | 1 | 1 | 1 |
| **Initialize vector** | 1 | 1 | 1 |
| **Create new object (CourseNumber, CourseName, Prerequisite)** | 1 | n | n |
| **Append to vector** | 1 | n | n |
| **Input Desired CourseNumber** | 1 | n | n |
| **For all Courses** | 1 | n | n |
| **If user input equals CourseNumber** | 1 | n | n |
| **Output: CourseNumber with CourseName** | 1 | n | n |
| **If prerequisite exists for desired course** | 1 | n | n |
| **For each Prerequisite of the course** | 1 | n | n |
| **Output: CourseName and CourseNumber** | 1 | n | n |
| **Else if prerequisite wasn’t found** | 1 | n | n |
| **Output: Prerequisite not found** | 1 | n | n |
| **Else if course not found** | 1 | n | n |
| **Output: Course not found** | 1 | n | n |
| **Total Cost** | | | 41n + 5 |
| **Runtime** | | | O(n) |

HashTable

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Initialize hashtable** | 1 | 1 | 1 |
| **TableSize equals 100** | 1 | n | n |
| **Struct Course** | 1 | 1 | 1 |
| **Resize TableSize** | 1 | n | n |
| **Initialize instream** | 1 | 1 | 1 |
| **Open file** | 1 | 1 | 1 |
| **If file not found** | 1 | n | n |
| **Else if file cannot be opened** | 1 | n | n |
| **Else If file has been found and can be opened** | 1 | n | n |
| **For each line on file** | 1 | n | n |
| **Split line at comma** | 1 | n | n |
| **If there is less than 2 parameters** | 1 | n | n |
| **Output: invalid format** | 1 | n | n |
| **Else if there’s 2 or more parameters** | 1 | n | n |
| **Assign CourseNumber to first parameter** | 1 | n | n |
| **Assign courseName to second parameter** | 1 | n | n |
| **If there is more than 2 parameters** | 1 | n | n |
| **Assign each parameter as prerequsite** | 1 | n | n |
| **Add course into hashtable** | 1 | n | n |
| **For each course in hash table** | 1 | n | n |
| **If prerequisite exists** | 1 | n | n |
| **For each prerequisite** | 1 | n | n |
| **Search for matching courseNumber with key** | 1 | n | n |
| **If CourseNumber exists within hashtable** | 1 | n | n |
| **Else if prerequisite doesn’t exist** | 1 | n | n |
| **Erase prerequisite from course** | 1 | n | n |
| **Struct Node** | 1 | 1 | 1 |
| **Return key % tableSize** | 1 | n | n |
| **Try** | 1 | n | n |
| **For each line in file** | 1 | n | n |
| **Course equals new course (courseNumber, courseName, Prerequisite)** | 1 | n | n |
| **Create key for given bid** | 1 | n | n |
| **Retrieve node from key** | 1 | n | n |
| **If node at the key is empty** | 1 | n | n |
| **Add object to the node** | 1 | n | n |
| **Begin at first node** | 1 | n | n |
| **While node is not empty** | 1 | n | n |
| **Add object to next free node with the new key** | 1 | n | n |
| **Catch** | 1 | n | n |
| **Output: error** | 1 | n | n |
| **Input CourseNumber** | 1 | n | n |
| **Create new key for given course** | 1 | n | n |
| **Retrieve node from the key** | 1 | n | n |
| **If node is full and matches courseNumber** | 1 | n | n |
| **Return node** | 1 | n | n |
| **Output: CourseNumber and CourseName** | 1 | n | n |
| **Else** | 1 | n | n |
| **While hashtable has not ended** | 1 | n | n |
| **If node matches CourseNumber** | 1 | n | n |
| **Return node** | 1 | n | n |
| **Output: CourseNumber and CourseName** | 1 | n | n |
| **Node equals next** | 1 | n | n |
| **If no node was returned** | 1 | n | n |
| **Output: course was not found** | 1 | n | n |
| **If inputted CourseNumber has prerequisite courses** | 1 | n | n |
| **Output: prerequisite courses:** | 1 | n | n |
| **For each prerequisite course** | 1 | n | n |
| **Call to searchCourse function using prerequisite as inputted CourseNumber** | 1 | n | n |
| **Total Cost** | | | 58n + 5 |
| **Runtime** | | | O(n) |

Binary Tree

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Struct Course** | 1 | 1 | 1 |
| **Initialize ifstream** | 1 | 1 | 1 |
| **Open file** | 1 | 1 | 1 |
| **If file not found** | 1 | n | n |
| **Else file is found** | 1 | n | n |
| **Declare temp as temporary array** | 1 | 1 | 1 |
| **While file has not ended** | 1 | n | n |
| **Add up to 5 lines from the file to temp array** | 1 | n | n |
| **If line has already been processed** | 1 | n | n |
| **Skip line and move onto new line** | 1 | n | n |
| **For each line in temp array** | 1 | n | n |
| **If there is a comma** | 1 | n | n |
| **Split line at comma** | 1 | n | n |
| **If line has less than 2 parameters** | 1 | n | n |
| **Remove line** | 1 | n | n |
| **Else if there is 2 or more parameters** | 1 | n | n |
| **Assign CourseName to 1st parameter** | 1 | n | n |
| **Assign CourseNumber to 2nd Parameter** | 1 | n | n |
| **If there is more than 2 parameters** | 1 | n | n |
| **For each parameter after 2nd** | 1 | n | n |
| **Assign prerequisite to every parameter after 2nd** | 1 | n | n |
| **For each prerequisite** | 1 | n | n |
| **Loop through file** | 1 | n | n |
| **If prerequisite exists within file** | 1 | n | n |
| **Else if prerequisite does not exist witin file** | 1 | n | n |
| **Remove prerequisite** | 1 | n | n |
| **Output: this prerequisite did not exist** | 1 | n | n |
| **Else if there’s no prerequisites, continue** | 1 | n | n |
| **Add course to Binary tree end** | 1 | n | n |
| **Delete Temp Array** | 1 | n | n |
| **InOrderTraversal** | 1 | 1 | 1 |
| **PreOrderTraversal** | 1 | 1 | 1 |
| **PostOrderTraversal** | 1 | 1 | 1 |
| **Struct Node** | 1 | 1 | 1 |
| **Course course create object (courseName, courseNumber, prerequisite)** | 1 | n | n |
| **For each course** | 1 | n | n |
| **If course is larger then add to the left** | 1 | n | n |
| **If left node is empty** | 1 | n | n |
| **Course becomes that node** | 1 | n | n |
| **Else if left node is not empty** | 1 | n | n |
| **Add new node** | 1 | n | n |
| **Course becomes that new node** | 1 | n | n |
| **Else if course is smaller then add to the right** | 1 | n | n |
| **If right node is empty** | 1 | n | n |
| **Course becomes that node** | 1 | n | n |
| **Else if right node is not empty** | 1 | n | n |
| **Add a new node to the right** | 1 | n | n |
| **Course becomes that new node** | 1 | n | n |
| **Bool exists equals false** | 1 | n | n |
| **Node pointer current equals node** | 1 | n | n |
| **Whilst current is not null pointer** | 1 | n | n |
| **If current equals current course** | 1 | n | n |
| **Return current course** | 1 | n | n |
| **Exists equals true** | 1 | n | n |
| **Else if smaller than current node** | 1 | n | n |
| **Traverse left** | 1 | n | n |
| **Return course** | 1 | n | n |
| **Exists equals true** | 1 | n | n |
| **Traverse right** | 1 | n | n |
| **Return course** | 1 | n | n |
| **Exists equals true** | 1 | n | n |
| **Else** | 1 | n | n |
| **Output: course has not been found** | 1 | n | n |
| **Get user input for courseNumber** | 1 | n | n |
| **Call to searchnode function using user input** | 1 | n | n |
| **If exists equals true** | 1 | n | n |
| **Output: courseNumber, courseName** | 1 | n | n |
| **For each prerequisite** | 1 | n | n |
| **Call to searchNode function** | 1 | n | n |
| **If exists equals true** | 1 | n | n |
| **Output: prerequistie, courseName** | 1 | n | n |
| **Total Cost** | | | 71n + 8 |
| **Runtime** | | | O(n) |

1. Advantages and disadvantages of each structure

**Vector**

The vector data structures offers some advantages over the other structures. Its simplistic implementation allows for faster run times. This structure handles insertions and deletions well. Vectors have advantages when it comes to data modifications. Although the vectors simplistic nature will cause some disadvantages in other aspects. When searching for a specific item, vectors will need to loop through the entire list to find the desired item which can slow runtimes down immensely.

Whilst there are no clear advantages or special capabilities with sorting it is moderately capable of sorting. In terms of the course project, vectors offer a great simplistic implementation of all the necessary functions. This is good for keeping the total cost low which can improve runtimes. Vectors will have some issues when it comes to searching for courses and prerequisites. But it will perform well with creating objects and loading them in. Sorting may not be powerful, but the vector can handle it.

**Hashtable**

The Hashtable holds advantages in searching capabilities. Items are assigned to keys that make finding specific items easy. Hash tables also have strong insertion and deletion capabilities thanks to the keys. Hash tables do have disadvantages with sorting and organizing. It can be harder to change the key or choose exactly which key you want an item to be signed to.

For the Course project the hashtable has an incredible advantage when dealing with prerequisites and searching for courses. Some user functions and runtimes will greatly benefit from the direct access capabilities. Although this project does have sorting and organizing requirements, which can increase runtimes. This may be slower than vectors. There will be no issues with loading capabilities.

**Binary Tree**

Binary trees holds advantages with sorting and organization capabilities. Items are easily sorted which can make the entire data list easier to work with. The tree can be traversed in different ways which can improve search times for specific items. Binary trees have a more complex structure and implementation. This can make data modifications harder and hold a higher total cost. Runtimes can be slower from this.

For the Course project sorting is present and binary trees will hold a big advantage with this function. Searching may be faster than vectors, But slower than has tables. Due to the complex nature loading may be slower when compared to the others. It will also require a heavier cost overall.

1. Data structure that will be used.

I personally recommend vectors. Out of all the data structures vectors had the lowest cost and is evident that vectors may have the lowest runtime. The simplistic nature of vectors can make it a jake of all trades. The project requires the data to complete a diverse set of functions and tasks. Vectors will be able to handle all required functions. There is a disadvantage with its searching and retrieval speeds, but its low cost and simpler implementation can outweigh those disadvantages. Binary trees held a great advantage with sorting, but the complicated implementation didn’t justify the advantage of one aspect of the project. Hash tables does great with retrieval speeds which is repeatedly needed throughout the course project. But the weaknesses to sorting capabilities and higher cost swayed me more towards vectors. There is also an added versatility bonus when working with vectors. This can provide a creative advantage over other structures.