Derek Clark Project One

1a.

Utilize fstream to open the file

If file opened successfully

Continue

Else

Return error “File did not open”

When file opens:

Initialize starting row of csv

While not at the end of the file iterate through the lines

Read line

If there are less than 2 parameters:

return error “Must contain at least 2 parameters”

if line contains 2 parameters

add course number and course title to list of courses

if line contains 3 parameters or more

for each additional parameter check if there is another line in the file that

contains the same course number

if a corresponding course number is found

continue

else

return error “course not found in prerequisite”

next line

close file

1b.

Initialize course structure Course Id, title, list of prerequisites

While not at end of file

For each line in csv

Set first value to course Id

Set second value to course title

If there are more values

Add following values to list of prerequisites

1c.

Display course function:

Loop through courses until null

If course in courses

Print course id, title loop through prerequistes until null

2.

**Create Menu:**

Initialize choice 0

While choice not equal to 9

Print out list of options:

Menu

1.Load courses

2.Print ordered course list

3.Print individual course

9.Exit

Switch pass in choice:

If choice is 1

Call Load courses function pass in csv path and data structure

If choice is 2

Call print sorted function pass in courses

If choice is 3

Call print course information function passing in course id

If choice is 9

Terminate while loop

2a.

**Load data into vector:**

Vector<Course> courses

For line in file

While not end of line

For line in file

Create Course

Push coursenumber into course in courses first position

Push courseName into course in courses second position

If more parameters exist and valid:

Loop until no more exists:

Push data into course prerequisite vector next position

Next line

**Load data into hash table:**

Initialize hash table:

Size

Methods:

Hash key create

Hashtable insert

Define course

CourseID = string

CourseName = String

Prerequisites=vector<Prerequisites> Prerequisites

Initialize Vector<Course> courses

Parsing in file to hash table

While not at the end of the file

for each line in file

Create a course in courses

Set first value in line to course ID

Set second value in line to Course title

If additional values add to prerequisite vector

Call hash function to hash value for course id

Call insert method to add course and key into hash table

**Load data into binary tree:**

Create Tree

Inialize tree with root equal to null

Create binary tree insert method

If root does not exist

Create new node pass it in the course

If root does already exists

Call add node method and pass it the root and course

Create binary tree add node method

If the root course id is larger than the course id being passed add it to the left side

If the left node does not exist already make this course the left node

If the left node does already exist recurse down left node

Else if the root course id is smaller than the course id being passed add it to the right side

If the right node does not exists make this course the right node

If the right node does already exists continue to the right

2c.

Search for specific course and print:

**Vector:**

Prompt user for course id

If course id does not exist in courses

Print course does not exists

If course id does exist in courses

Print course title with id and loop through prerequisites

**Hashtable:**

Prompt user for course id

Create key for given course using hash function

If entry found for the key

Print course title and loop through prerequisites found at key

If no entry found for the key

Return empty course

While node not equal to null

If the current node matches print course title and loop prerequisites

If it is not set node equal to next node

**Binary tree:**

Initialize current node as root

While current is not equal to null

If current courses course id is the searched for course id

Print current course title and loop prerequiests

If searched course id is smaller than current course id

Set current node to left node

Else if larger

Set current node to right node

2d.

if the user inputs a 9 the while loop ends

print goodbye

return 0;

3.

Print course list in alphanumeric order

**Vector:**

vector sort method passing in courses, first course and last course

initialize mid equal to 0

if there are one or zero courses in courses or if begin is greater than or equal to mid

return

pass in courses begin and end to the partition method and set it equal to mid

call sort method passing in courses, begin and mid

call sort method passing in courses, mid+1 and end

vector partition method passing in the courses vector, the first courses, the last course

set low to the first element

set high to the last element

set pivot point to begin+(end-begin)/2 or middle point

set complete to false

while not complete

while course title compared to pivot course title is less

increment low by 1

while high course title is greater than pivot course title

decrement high by 1

if low is less than or equal to high

completed

else

swap low courses and high courses

increment low

decrement high

return high

Print sorted:

Call sort method passing in courses, 0, and index of last course

Loop through courses after sort is called

Display courses from index 0 to last index

**Hashtable:**

if the hash table is ordered

for each key in hash table

print value associated with its id, title, prerequisites

if hash table is unordered

iterate through hash tables keys

copy value of each keys node into vector including id, title and prerequisites

sort new vector using vector sorting algorithm above

iterate through the vector starting at index 0 till vector end

print course id, title, prerequisites

**Binary tree:**

Create inOrder method

Traverse the binary tree

Call inOrder function and pass in the binary tree root

inOrder function:

if the node being passed exists

call inOrder function on left nodes

print course information from node

call inOrder function on right nodes

4.

|  |  |  |
| --- | --- | --- |
| Vector | Cost per line | Number of executions |
| Reading the file | 1 | n |
| Creating vector course object | 1 | n |
| Total | Memory: O(n) | Runtime: O(n) |

Reading the file takes as long as there are lines in the file. Creating a course object is as easy as inserting each line parsed though for number of lines time onto the vector. The memory needed will be at worst the number of courses needed to be stored, the worst run time in total is the number of line/courses being read in.

|  |  |  |
| --- | --- | --- |
| Hash Table | Cost per line | Number of executions |
| Reading the file | 1 | n |
| Creating Hash Table course object | 1 | n |
| Inserting course into hash table | 1 | n |
| Total: | Memory: O(n) | Runtime: O(n) |

Each line in the csv file is n lines and it will take O(n) run time to read the lines. Creating a hash table course object will take as many courses as there are because each course will need an associated key and value pair loading in the line from the csv. I believe that inserting it into a hash table will actually take constant time because it is not like a vector or binary tree in which we need to iterate through other values in order to insert but instead directly inserts it into its key.

|  |  |  |
| --- | --- | --- |
| Binary Tree | Cost per line | Number of executions |
| Reading lines from file | 1 | n |
| Creating course object | 1 | n |
| Inserting object into binary tree | 1 | Log n |
| Total | Memory: O(n) | Runtime: O(n log n) |
|  |  |  |

It will take n run time to read the lines from file and parse them through to the binary tree because it will have to go through n lines. There are as many course objects as there are lines in the csv file so it will have a runtime of O(n). I believe that in terms of memory it will take O(n) because as the number of courses increases so does the amount of nodes on the tree. Once the object is created it will take O(log n) to insert it into the tree because you can either go right or left halving the options for the node to be inserted.

5.

Different data structures have different advantages and disadvantages when it comes to finding a solution that best fits a particular problem. There are many factors that come in to play when deciding on the best data structure to use for a certain problem like run time, memory usage, whether you will need to frequently access the last items in it, insertion/deletion of items, sorting items, the amount of items etc… Vectors have the advantage of being flexible and dynamic automatically adjusting their size when elements are being added or removed from it. The disadvantage of using vectors is when the data set is not sorted and finding specific elements within the data structure will take as much time as there are elements before it which if the element is found near the beginning or if there are not many elements in the vector may take not too long but as the size of the vector grows so will the time it takes to iterate through to find the element. Of course vectors do have an advantage when the data set is sorted, or the index is known instantly allowing you to call on the element by its index retrieving it. The main advantage of using a hash table is that they offer key value pairs in which elements are stored by key and can easily be retrieved, inserted, and deleted typically in constant time. The main disadvantage of using a hash table is that may have to deal with collisions more so when utilizing a weak hash function which is when multiple inputs hash to the same output. One disadvantage I also identified when developing this project is that it requires additional unnecessary steps to sort a hash table because you will need to utilize an additional data structure. The advantage of using a binary tree is that it maintains the order of values within it and the height of the tree remains smaller than the number of elements inside it making it very memory efficient. Order is maintained with all the values smaller than the parent put on the left while any value bigger is stored on the right. With its advantages there are also disadvantages like significant memory usage in storing pointers to the right and left children of a parent node and not having constant time lookups like an index for vectors or hash key for hash tables.

6.

Out of the three data structures I have discussed in this paper the data structure that I would recommend and plan to use in my code is the vector data structure. I believe this data structure to be the most flexible, adaptable with many great sorting and searching algorithms, and the most straight forward to use. Reading through the lines of the csv file for vectors has a worst case run time of O(n) which means that it grows based on the number of lines that it needs to read in. Creating a vector course object will take O(n) time Aswell because it will be based on the number of courses that are being inserted into the vector. Overall vectors have good sorting algorithms and search algorithms which for our case works in our favor if we need to sort through the courses or find a particular course in the list of courses. If I were to need quick insertion, I would use a hash table for its constant runtime using keys.