Project One

Start while loop for Menu while input not equal 9:

1. Load courses
2. Print courses
3. Search for a course

9.) Exit program

Create cmd (char) and store user’s input

If cmd == ‘1’ then prompt the user to load file path

If user doesn’t enter anything then the default file path will be ran

If user enters an invalid file path they will be displayed with “file not found”

If cmd == ‘2’ then print all courses using the binary search tree’s inOrder method

If cmd == ‘3’ then prompt user to input the course id

Create courseKey to takes user’s input

Create courseKeyUpper

Loop 0…courseKey’s size and increment by 1

Concatenate the char at index of courseKey to add it to courseKeyUpper

Create a struct Course course and assign it the binary search tree’s Search method with courseKeyUpper as its parameter

Reset courseKeyUpper to “” to keep from adding the previous string

Check if course’s course id is not empty and print that course and display the prerequisites; if any.

Else display Course Id not found

If cmd == ‘9’ then display Good Bye to the user and exit while loop

Parser:

**csvParser:**

Create a container that uses a vector for loading csvCourses and pass csvPath as a parameter

Create a vector for Course called courses

Init csv parser as file and set to csv parser and csvPath as its parameter

Try

Start loop 0… file row count

Create Course course

Set courseId to file [current row] [column 1]

Set courseName to file [current row] [column 2]

Append course to courses

Catch csv error

Print error code

Return courses

**Course structure:**

struct Course {}

coursedId (String)

courseName (String)

preReqs (String)

Course (){}

courseId = “”

courseName = “”

preReqs = “”

Class HashTable{}

-struct Node{}

Course (struct)

key (int)

next (\*Node)

+hash()

+printAll()

+Vector<> hashTable

**Class BinaryTree{}**

-struct Node{}

Course(struct)

right pointer (null)

left pointer (null)

-root

+printCourses()

+BinaryTree()

**Print Courses:**

Loop 0… size of courses

Print course[current index].courseId, course[current index].courseName

End loop

Analysis:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Vector | Hash Table | Binary Search Tree |
| insert | O(1) | O(1) – O(n) | O(log n) |
| search | O(n) | O(1) – O(n) | O(log n) – O(n) |
| sort | O(n log n) – O(n^2) | O(n) | O(n) |

Using the csvParser we need to start by reading line by line (once) which is O(n) and then we read word by word in each line(once) O(n) which is O(n) \* O(n) which is still O(n)

After csv parsed:

A vector can insert in constant time but searching data is linear O(n) and sorting data comes at a cost being O(n^2) or O(log n) depending on the algorithm used

A hash table can insert in constant time but if there is a collision in the bucket then time will be O(n) and the same can be said for searching with a hash table and sorting data is O(n)

Binary tree loading is O(log n) however depending if the tree is balanced or not searching can be O(log n) or O(n) and sorting is O(n)

Recommendation:

I recommend using the Binary search tree since it would be more consistent in space and time complexity regardless of the its size