CS 300: Project One

-BINARY TREES-

Reading from file

IMPORT fstream for file reading

CALL to open file

IF return equals -1

File is not found

RETURN ERROR

ELSE

WHILE not the EOF

READ each line

IF there are less than two values

RETURN ERROR

ELSE read

IF more than two parameters

IF third or more parameter is elsewhere

CONTINUE

ELSE

RETURN Error

CLOSE file

Create course objects:

INITIALIZE COURSE STRUC

WHILE NOT EOF

FOR each line

FOR first two values

CREATE course ID and NAME

IF third value present

ADD prereq

Tree and Adding nodes:

CREATE binary tree

CREATE root that equals null

CREATE insert method

IF the root equals nulls

Current course is root

ELSE IF course is less than root

ADD leaf to the left

ELSE

ADD leaf to the right

Printing and searching from tree:

GET user input

IF root does not equal null

IF node equals user input

PRINT

ELSE IF user input less than

TRAVEL left

Print

Else

TRAVEL right

PRINT

-HASH TABLE-

Reading file

IMPORT fstream to open and read file

CALL to open file

IF file is unable to open

RETURN -1

ELSE file is open

WHILE it is not EOF

READ each line

IF there are less than 2 items in a line

RETURN ERROR

ELSE read

IF three or more parameters exist

IF third or more parameter exists in first parameter

Continue

ELSE

RETURN error

CLOSE file.

Creating hashtable and course objects

INITIALIZE Course vector<NODE>

CREATE class for hashtables

CREATE method to insert items not CLASS hashtables

LOOP through file

WHILE not EOF

For each line in file

For first and second variables

CREATE temp holders

IF third value exists

Add to current variables

CALL insert method for each variable

Searching and printing

GET user input

ASSIGN input to key

IF key is in hashtable

PRINT course information

If course has prerequisites

FOR each prerequisite

PRINT prerequisite.

-VECTOR SORTING-

CALL to open the file

IF return is -1 file is not found

ELSE (file is found)

WHILE it is not the EOF

READ through each line.

IF only 1 value in line

Return error.

ELSE

CONTINUE

IF there is three or more parameters

IF third or more parameters are in first parameter

CONTINUE

ELSE

RETURN error

CLOSE file

CREATE Course object structure for storing

INITIALIZE vector<int>

WHILE loop through file

WHILE not EOF

FOR each line in file

Use pushback until new line occurs.

GET user input for course

LOOP through vector

IF input matches Vector

PRINT course information

IF prerequisites exist

FOR each prerequisite

PRINT prerequisite

-MENU LOOP-

INITIALIZE house keeping variables

CREATE bid variable

WHILE user input does not equal

PRINT

1: Load File Data

2: Print Course List

3: Print Course

4: Exit

GET user input

SWITCH CASE using user input

SWITCH (Input)

CASE 1:

Load bids(bid)

CASE 2:

PRINT courses alphanumerically

CASE 3:

PRINT course

PRINT prerequisites

CASE 4:

PRINT “Goodbye”

DEFAULT:

PRINT “Invalid input”

-ALPHANUMERIC ORDER-

CREATE sortVector(course)

CREATE partition method(course, begin, end)

SET low index to first element and high index to last element

SET mid point to low index plus (high index minus low index) divided by 2

SET pivot equal to midpoint

WHILE pivot is less than high index

DECREMENT high index

SWAP lower values to left of pivot and high values to the right

SET temp equal to low index

SET low equal to high

SET high equal to low

CREATE quick sort method using the courses high and low indexes

SET midpoint equal to 0, Low index to begin, highindex to end

IF begin is greater than or equal to end

RETURN

CALL quicksort recursively

CREATE display method

For each item in container

PRINT course

IF prereq exist

PRINT prereq

TREE

CREATE an inorder method

WHILE node doesn’t equal null

If smaller

Left node

Else

Right node

|  |  |  |
| --- | --- | --- |
| VECTOR | Times calles | TOTAL |
| Create course | 1 | 1 |
| While not EOF | n | n |
| For each line | n | n |
| Use pushback | n | n |
| TOTAL COST |  | 4n+1 |
|  |  | O(n) |

|  |  |  |
| --- | --- | --- |
| HASH TABLE | Times called | Total |
| Create course | 1 | 1 |
| Create hashtable class | 1 | 1 |
| While not EOF | n | n |
| FOR each line | n | N |
| For first and second | n | n |
| IF third value | n | n |
| IF key exist in hashtable | n | N |
| Print course information | 1 | 1 |
| If course has prereq | n | N |
| For each prere1 | n | N |
| Print prereq | 1 | 1 |
| TOTAL |  | 7n+4 |
|  |  | O(n) |

|  |  |  |
| --- | --- | --- |
| BINARY TREE |  |  |
| Create course | 1 | 1 |
| While not EOF | n | n |
| FOR each line | n | N |
| For first two values | n | n |
| Create id and name | n | N |
| IF third value exists | n | N |
| Add prereq | n | n |
| Create binary tree | 1 | 1 |
| Create root | 1 | 1 |
| Create insert method | 0 | 0 |
| If root is null | n | n |
| Else if course is less | n | N |
| else | n | n |
| Get user input | 1 | 1 |
| If root is not null | n | N |
| If node is user input | n | N |
| print | 1 | 1 |
| Else if user input less than | n | N |
| Travel left | n | N |
| else | n | n |
| Travel right | n | n |
|  |  | 16n+4 |
| TOTAL COST |  | O(n) |

Advantages and disadvantages:

A vector is a great tool but does fall behind in some categories. For example, one instance where a vector lack is the ability to search through to find what you are looking for. In a vector you will need to search every item until a match is found. One place a vector does well however, is being the fastest with the shortest runtime of 4n+1. A hash table is another equally great tool if utilized properly. Being the fastest method to search since they are able to use a user input as a key and match that key among the items in the table. The downside of the hash table is the slower sleep during its initial creation. It also does not have the ability to be sorted making it not an ideal choice for this assignment. A binary tree is faster to sort than a vector which is a great but searching inside of a tree is not as efficient as a hash table since you still have to make many comparisons. I believe it would be best to use a vector for this since its runtime is the lowest and has the capability to be sorted alphanumerically.