Michael Abell

CS-300

Project 1

## Vector

**File Access**

Open file from designated path

Read/Get each line

If the line is blank or does have the correct number of values

Error out

Output helpful error message “Check values”

Else

Continue to next line until end of file

**Object Definition**

List with the following entries

String courseNumber, mandatory

String name, mandatory

String prerequisite,

can be any real number >=0

will be the courseNumber only

Read each course from the data file

Write each line to Course object into the vector

**Print Course**

void searchCourse(Vector<Course> courses, String courseNumber) {

**for all courses**

**if the course is the same as courseNumber**

**print out the course information**

**for each prerequisite of the course**

**print the prerequisite course information**

**Print Alpha Numeric**

Sort the bids

Iterate through all the bids based on the size of the table

If current course is alphanumerically greater than the next bid then swap places

Keep iterating util the end

Print the newly sorted list of bids

## Hash

**File Access**

Open/load file from designated path

Read/Get each line

If the line is blank or does have the correct number of values(<2)

Error out

Output helpful error message “Check values”

Else

Continue to next line until end of file

**Object Definition**

Create hashTable class

Create a node

Course<Node>nodes

Declare coursed var

Declare courseTitle var

Declare preReq1 var

Declare preReq2 var

Declare preReqCount var

Set preReqCount to 0

Declare temp item to hold values

Declare current item that holds values with current pointer to next item

WHILE not end of file iterate through file

Declare unsigned int key

IF (node at key is not found)

INSERT new node at hash key %

ELSE IF (Node pointer to key equals UINT\_MAX)

Node pointer to key equals key

Node pointer to next equals nullptr

Node pointer to course = course

ELSE

WHILE (node -> next not equal to nullptr)

node equals node pointer to next. new node (course, key

Print Course Info

FOR loop to go through whole index of table

Print courseID, PreReq1, PreReq2

While the node isn’t null

Set the node to the next node in line

Print courseID, PreReq1, PreReq2

Return

End

**Print Alpha Numeric**

Sort the bids

Iterate through all the courses based on the size of the table

If current course is alphanumerically greater than the next bid then swap places

Keep iterating util the end

Print the newly sorted list of bids

## BST

**File Access**

Open/load file from designated path

Read/Get each line

If the line is blank or does have the correct number of values(<2)

Error out

Output helpful error message “Check values”

Else

Continue to next line until end of file

**Object Definition**

Create Binary Search Tree class

Create a root

Current course is the root

Create Branches

Left branches are for values(course values) less than the root

Right branches are for values(course values) less than the root

Print Course Info

FOR loop to go through whole index of the tree

Print courseID, PreReq1, PreReq2

While the root isn’t null

Set the pointer to the next node in line

Print courseID, PreReq1, PreReq2

Return

End

Print Alpha Numeric

Sort the courses by starting from the left branches and going up to the right

Print all

**Menu**

Display Menu options with a cout

Request input from user

Validate correct input

WHILE loop for the menu

While User input is not 9

Choice 1 will load the bid through the file access

Choice 2 will Print all the Courses through the print function

Choice 3 will search for, find, and print the single course that was input

Choice 9 will exit the program with a goodbye message and closing any open file.

|  |  |  |  |
| --- | --- | --- | --- |
| Code (Vector) | Line Cost | # Times Executed | Total Cost |
| Open File | 1 | 1 | 1 |
| Read Lines | 1 | N | N |
| Write Course to Vector | 1 | N | N |
| Write PreReq1 to Vector | 1 | N | N |
| Write PreReq2 to Vector | 1 | N | N |
|  |  |  |  |
| Total | Worst Case: 0(n) |  | 4n+1 |

|  |  |  |  |
| --- | --- | --- | --- |
| Code (Hash) | Line Cost | # Times Executed | Total Cost |
| Open File | 1 | 1 | 1 |
| Read Lines | 1 | N | N |
| Write Course to node | 1 | N | N |
| Write PreReq1 to node | 1 | N | N |
| Write PreReq2 to node | 1 | N | N |
| Create Next Node | 1 | N | N |
|  |  |  |  |
| Total | Worst Case: 0(n) |  | 5n + 1 |

|  |  |  |  |
| --- | --- | --- | --- |
| Code (BST) | Line Cost | # Times Executed | Total Cost |
| Open File | 1 | 1 | 1 |
| Read Lines | 1 | N | N |
| Create Root | 1 | N | N |
| Write to root | 1 | N | N |
| Write PreReq1 to root | 1 | N | N |
| Write PreReq2 to root | 1 | N | N |
| Compare next course | 1 | N | N |
| Make left or right branch at end of branch | 1 | N | N |
|  |  |  |  |
| Total | Worst Case: 0(n) |  | 8n+1 |

Analysis:

Each method attacks the data in it’s own way. The vector can just append the data at intake and then sort later when it’s time to display/print. SO it’s a quick intake, but the sorting can eat up cycles. The hash table keeps the data where it is an accesses it in a certain order, which once again can be a quick intake, but going back to access later it has quite a few iterations needed. Finally a binary search tree assigns the sorting based on the first data point at intake. This can lead to a highly skewed structure to the right or left to have to iterate over. If the data was known to be a midpoint datapoint then it would be optimal.

Recommendation:

My recommendation for this would be a vector structure to house the data. It’s suited to layers of data and can be iterated over and quickly sorted.