Christopher Wilson  
4/9/2025  
CS-300 project one  
   
 Pseudocode

Reading the file  
FUNCTION openFile:

Use fstream to open the file

Make call to open file

IF (return value is -1):

Print file was not found error  
ELSE:  
 WHILE (it is not at the end of file):

IF (there are less than two values in a line):

Print an error

ELSE IF (there is a third or more parameters):

IF (third or more parameter is in the first parameter):

Continue  
 ELSE:

Print an error

Vector  
  
FUNCTION Partition (courses, begin, end):

Set low as begin

Set high as end

Calculate middle as begin plus end divided by 2

Set pivot as the course number at middle index

Set finished as false

WHILE (finished is false):

WHILE (course number at low index is less than pivot):

increment low

WHILE (course number at high index is greater than pivot):

decrement high

IF (low is greater than or equal to high):

return high

Set finished as true

ELSE:

Swap courses at low and high indexes

Increment low

Decrement high

FUNCTION quickSort (courses, begin, end):

Set mid to 0

IF (begin is greater than or equal to end):

return

Call Partition to divide courses into low and high parts

Set mid to the partition index returned by Partition

Recursively call quickSort on low partition (from begin to mid)

Recursively call quickSort on high partition (from mid plus one to end)

FUNCTION selectionSort (courses):

Define min as the current minimum index

Set size to the number of courses in the vector

FOR each index from 0 to size minus one:

Set min as current index

FOR each subsequent index:

IF (course number at current index is less than course number at min):

Set min as current index

IF (min is not equal to current index):

Swap courses at current index and min

FUNCTION selectionSort (courses):

Define min as the current minimum index

Set size to the number of courses in the vector

FOR (each index from 0 to size minus one):

Set min as current index

FOR (each subsequent index):

IF (course number at current index is less than course number at min):

Set min as current index

IF (min is not equal to current index):

Swap courses at current index and min

FUNCTION printCoursePrerequisites (courses, targetCourseNumber):

Set found as false

FOR each course in courses:

IF (course number equals targetCourseNumber):

print course number and course title

IF (course has prerequisites):

Print prerequisites

ELSE:

Print this course has no prerequisites

Set found as true

break

IF (found is FALSE):

Print course not found

Hash tables

Class HashTable:

attributes:

tableSize as integer

nodes as vector of Node pointers

Constructor(size):

Set tableSize to size

Resize nodes vector to tableSize

Initialize all entries to nullptr

Destructor():

FOR each node in nodes vector:

Delete node and all chained nodes

Function hash(key):

Return key converted to integer Modulo tableSize

Function insertBid(bid):

Set key to hash(bid's ID converted to int)

Get node at index key

IF (node is empty):

Create new Node with bid and key

Set nodes[key] to new Node

ELSE:

IF (node is marked unused (key == UINT\_MAX)):

Set node's key to new key

Set node's bid to new bid

Set node's next to nullptr

ELSE:

Traverse to end of chain

Add new Node to end

FUNCTION printAllBids():

For (each index in table):

Get current node at index

WHILE (node exists):

IF (node is used (key != UINT\_MAX)):

Print node's key and bid info

Move to next node in chain

FUNCTION removeBid(bidID):

Set key to hash(bidID converted to int)

Get node at index key

IF (node exists and is used):

Mark node as unused (SET key to UINT\_MAX)

FUNCTION searchBid(bidID):

Initialize empty bid

Set key to hash(bidID converted to int)

Get node at index key

IF (node doesn't exist or is unused):

Return empty bid

IF (head node's bidID matches):

Return head node's bid

WHILE (next node exists):

IF (node's bidID matches):

Return node's bid

Move to next node

Return empty bid

FUNCTION printCoursesAlphabetically():

Create temp vector of all courses

FOR (each index in hash table):

Get current node

WHILE (node exists):

IF (node is used):

Add node's course to temp vector

Move to next node

Sort temp vector by courseNumber using quickSort

Sort temp vector by courseNumber using selectionSort

FOR (each course in sorted vector):

Print course.number, course.title

FUNCTION printCoursePrerequisites(courseNumber):

Search for course using searchBid(courseNumber)

IF (course found):

IF (course.prerequisites exists):

Print the prerequisites

ELSE:  
 Print no prerequisites

ELSE:

Print there was not prerequisite and course

Binary search tree  
  
Class BinarySearchTree:

ATTRIBUTES:

root as Node pointer initialized to nullptr

Destructor():

Call deleteTree(root)

FUNCTION deleteTree(node):

IF (node is not null):

Call deleteTree(node.left)

Call deleteTree(node.right)

Delete node

FUNCTION insertBid(bid):

IF (root is null):

root = new Node(bid)

ELSE:

Call insertNode(root, bid)

FUNCTION insertNode(node, bid):

IF (bid.id is less than node.bid.id):

IF (node.left is null):

node.left = new Node(bid)

ELSE:

Call insertNode(node.left, bid)

ELSE:

IF node.right is null:

node.right = new Node(bid)

ELSE:

Call insertNode(node.right, bid)

FUNCTION searchBid(bidId):

Set current to root

WHILE (current is not null):

IF (current.bid.id equals bidId):

Return current.bid

IF (bidId is less than current.bid.id):

current = current.left

ELSE:

current = current.right

Return emptyBid

FUNCTION removeBid(bidId):

root = CALL removeNode(root, bidId)

FUNCTION removeNode(node, bidId):

IF (node is null):

RETURN node

IF (bidId is less than node.bid.id):

node.left = CALL removeNode(node.left, bidId)

ELSE (IF bidId is greater than node.bid.id):

node.right = CALL removeNode(node.right, bidId)

ELSE:

IF (node.left is null and node.right is null):

Delete node

node = null

ELSE IF node.left is null:

temp = node

node = node.right

DELETE temp

ELSE IF (node.right is null):

temp = node

node = node.left

Delete temp

ELSE:

temp = node.right

WHILE (temp.left is not null):

temp = temp.left

node.bid = temp.bid

node.right = CALL removeNode(node.right, temp.bid.id)

RETURN node

FUNCTION inOrder(node):

IF (node is not null):

Call inOrder(node.left)

Print node.bid.id, node.bid.title, node.bid.amount, node.bid.fund

Call inOrder(node.right)

FUNCTION postOrder(node):

IF (node is not null):

Call postOrder(node.left)

Call postOrder(node.right)

Print node.bid.id, node.bid.title, node.bid.amount, node.bid.fund

FUNCTION preOrder(node):

IF (node is not null):

Print node.bid.id, node.bid.title, node.bid.amount, node.bid.fund

Call preOrder(node.left)

Call preOrder(node.right)

FUNCTION printCoursesAlphabetically():

Call inOrderTraversal(root)

FUNCTION inOrderTraversal(node):

IF (node is not null):

Call inOrderTraversal(node.left)

Print node.course.number, "-", node.course.title

Call inOrderTraversal(node.right)

FUNCTION printCoursePrerequisites(courseNumber):

Set current to root

WHILE (current is not null):

IF (current.course.number equals courseNumber):

Print current.course.number and current.course.title

IF (current.course.prerequisites is not empty):

Print current.course.prerequisites

ELSE:

PRINT no prerequisites

Return

IF (courseNumber is less than current.course.number):

current = current.left

ELSE:

current = current.right

Print course and course number not found

Menu

userChoice = GET\_USER\_INPUT("Enter your choice: ")

SWITCH (userChoice)

CASE "1":

Call funcation openFile()

CASE "2":

Call function printCourcesAlphabetically()

CASE "3":

Call function printCoursePrerequisites(courseNumber)

CASE "9":

PRINT "Exiting program. Goodbye!"

RETURN

DEFAULT:

PRINT "Invalid choice. Please enter 1, 2, 3, or 9."

Run Time Analysis

For my analysis, I was required to use a big O analysis, which required me to calculate the total cost per line of code and executions per line. For my vector, the total cost is 5n with a run time of O(n). The total cost for a hash table is 9n+1 with a run time of O(n). The binary search tree has a total cost of 11n+1 with a run time of O(n). There are some advantages and disadvantages of vectors, hashes, and binary search trees. For vectors, the one advantage is that it is fast at reading files and adding to the course object. The disadvantage is that you have to go through the whole program to find a specific course. Another advantage is that it has the shortest runtime of the three methods. An advantage of hashes is that they are able to search through a list very quickly. The location can be searched and printed quickly. The one problem it has is that it takes a while to sort the list, which makes it a problem when trying to print it in an alphanumeric system. The advantages of binary trees are that they have a faster searching ability compared to vectors. It can also sort the order of the data quickly. But the disadvantage is that they are more complex to implement and are slower to insert and delete data compared to the vectors and hashes. After weighing the advantages and disadvantages I think that binary tree is the best fit for this project. This is because it requires the project to print in alphabetical order and print the prerequisites in which the binary tree is good at.