Kenneth T Dandrow 8/9/24

Dr. Kevin Eaton CS-300

**6-2 Project One: Pseudocode and Runtime Analysis**

**DATA STRUCTURES:**

**Vector**

**Pseudocode for Selection Sort**

Function selectionSort(bids):

Input: A list of bids

Output: A sorted list of bids by title

**Pseudocode for Quicksort**

Function partition(bids, begin, end):

Input: A list of bids, a starting index (begin), and an ending index (end)

Output: The index of the partition

Function quickSort(bids, begin, end):

Input: A list of bids, a starting index (begin), and an ending index (end)

Output: A sorted list of bids by title

**Pseudocode for Main Program**

Function main():

Input: Command line arguments (argv)

Output: None

Summary of the Main Program's Flow:

Menu Option 1:

Load bids from a CSV file and measure the time taken.

Menu Option 2:

Display all loaded bids.

Menu Option 3:

Sort bids using selection sort and measure the time taken.

Menu Option 4:

Sort bids using quicksort and measure the time taken.

Menu Option 9:

Exit the program.

**Hash Table**

**Reading File:**

Use “<fstream>” to be able to open file

Make call to open file, if the return value is “-1” then file is not found

Else file is found

While it is not the EOF (End of File)

Read each line

IF There are less than two values in a line, return ERROR

ELSE read parameters

IF there is a third parameter or more

IF third or more parameter is in first parameter elsewhere continue

ELSE return ERROR

Close file;

**Create Course Objects HashTable:**

Initialize Course Vector vector<Node> nodes

Create HashTable class

Create Insert method to insert items to HashTable

Loop through file

While not EOF (End of File)

For each line in file

For first and second value

Create temp item to hold values

If a third value exists

Add to current value

Call insert method for each value;

**Search and Print from HashTable:**

Ask for input

Assign input to key in

If key is found

Print out the course information for each prerequisite of the course

Print prerequisite course information;

**Binary Search Tree (OLD)**

Root

Root = nullptr

Insert

Root = to new node

Else

Add new root

Search

Cur node = root

If

Traverse < cur root, left

Else

Right

Return

Add node

<node

Go left

Add node

InOrder

Place nodes in order

Id, title, amount, fund

PostOrder

Place nodes in order

Id, title, amount, fund

PreOrder

Place nodes in order

Id, title, amount, fund

Remove

Remove node by root/bidId

**Binary Search Tree (NEW)**

*I wanted to redo my Binary Search Tree pseudocode. I was not happy with the way the older pseudocode looked and translated. The following is the new submission…*

**Structure Definitions:**

Structure Bid:

bidId: String

title: String

fund: String

amount: Float

Structure Node:

bid: Bid

left: Node

right: Node

**Binary Search Tree Class:**

Class BinarySearchTree:

Function Constructor():

root = null

Function Insert(bid: Bid):

If root is null:

root = Create new Node with bid

Else:

Call addNode(root, bid)

Function Remove(bidId: String):

root = removeNode(root, bidId)

Function Search(bidId: String) -> Bid:

Return searchNode(root, bidId)

Function DisplayAllBids():

Call inOrder(root)

//// Private help functions

Function addNode(node: Node, bid: Bid):

If bid.bidId < node.bid.bidId:

If node.left is null:

node.left = Create new Node with bid

Else:

Call addNode(node.left, bid)

Else:

If node.right is null:

node.right = Create new Node with bid

Else:

Call addNode(node.right, bid)

Function removeNode(node: Node, bidId: String) -> Node:

If node is null:

Return null

If bidId < node.bid.bidId:

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

Else If bidId > node.bid.bidId:

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

Else:

If node.left is null:

Return node.right

Else If node.right is null:

Return node.left

minNode = findMin(node.right)

node.bid = minNode.bid

node.right = removeNode(node.right, minNode.bid.bidId)

Return node

Function searchNode(node: Node, bidId: String) -> Bid:

If node is null:

Return null

If bidId == node.bid.bidId:

Return node.bid

Else If bidId < node.bid.bidId:

Return searchNode(node.left, bidId)

Else:

Return searchNode(node.right, bidId)

Function inOrder(node: Node):

If node is not null:

Call inOrder(node.left)

Print node.bid

Call inOrder(node.right)

Function findMin(node: Node) -> Node:

While node.left is not null:

node = node.left

Return node

**Main Program:**

Function main():

bst = Create BinarySearchTree

choice = 0

While choice != 9:

Print menu options

Input choice

If choice == 1:

Input CSV filepath

Load bids from CSV

For each bid:

bst.Insert(bid)

Else If choice == 2:

bst.DisplayAllBids()

Else If choice == 3:

Input bidId

bid = bst.Search(bidId)

If bid is not null:

Print bid details

Else:

Print "Bid not found"

Else If choice == 4:

Input bidId

bst.Remove(bidId)

Print "Bid removed"

End While

**Pseudocode for a Menu**

PRINT "1. Load Data"

PRINT "2. Print Course List"

PRINT "3. Print Course Details"

PRINT "9. Exit"

INPUT choice

IF choice == 1 THEN

CALL LoadData\_Vector(fileName) OR LoadData\_HashTable(fileName) OR

LoadData\_Tree(fileName)

ELSE IF choice == 2 THEN

CALL PrintCourseList\_Vector() OR PrintCourseList\_HashTable() OR

PrintCourseList\_Tree()

ELSE IF choice == 3 THEN

INPUT "Enter course ID:" courseNumber

CALL PrintCourseDetails\_Vector(courseNumber) OR

PrintCourseDetails\_HashTable(courseNumber) OR PrintCourseDetails\_Tree(courseNumber)

ELSE IF choice == 9 THEN

PRINT "Exiting..."

ELSE

PRINT "Invalid option."

END IF

WHILE choice != 9

END

**Pseudocode: Sorting and Printing Courses In Alphanumeric Order**

**Vector**

FUNCTION PrintCourseList\_Vector()

SORT coursesVector BY courseNumber

PRINT "Course List (Vector):"

FOR EACH course IN coursesVector DO

PRINT course.courseNumber + " - " + course.title

END FOR

END FUNCTION

**Hash Table**

FUNCTION PrintCourseList\_HashTable()

CREATE sortedCoursesList

FOR EACH key IN hashTable DO

ADD hashTable[key] TO sortedCoursesList

END FOR

SORT sortedCoursesList BY courseNumber

PRINT "Course List (Hash Table):"

FOR EACH course IN sortedCoursesList DO

PRINT course.courseNumber + " - " + course.title

END FOR

END FUNCTION

**Binary Tree**

FUNCTION PrintCourseList\_Tree()

PRINT "Course List (Tree):"

IN\_ORDER\_TRAVERSAL(rootNode)

END FUNCTION

FUNCTION IN\_ORDER\_TRAVERSAL(node)

IF node IS NOT NULL THEN

IN\_ORDER\_TRAVERSAL(node.left)

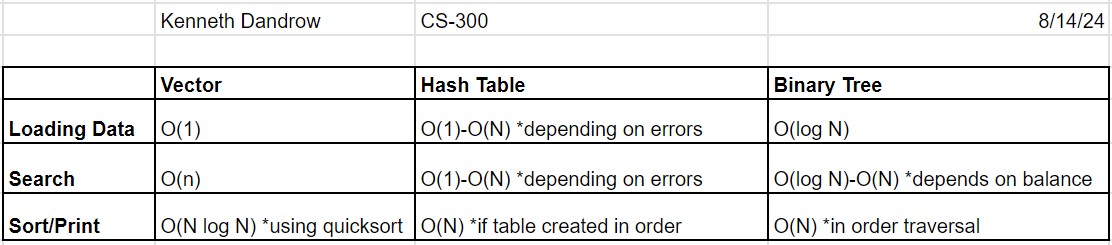
PRINT node.courseNumber + " - " + node.title

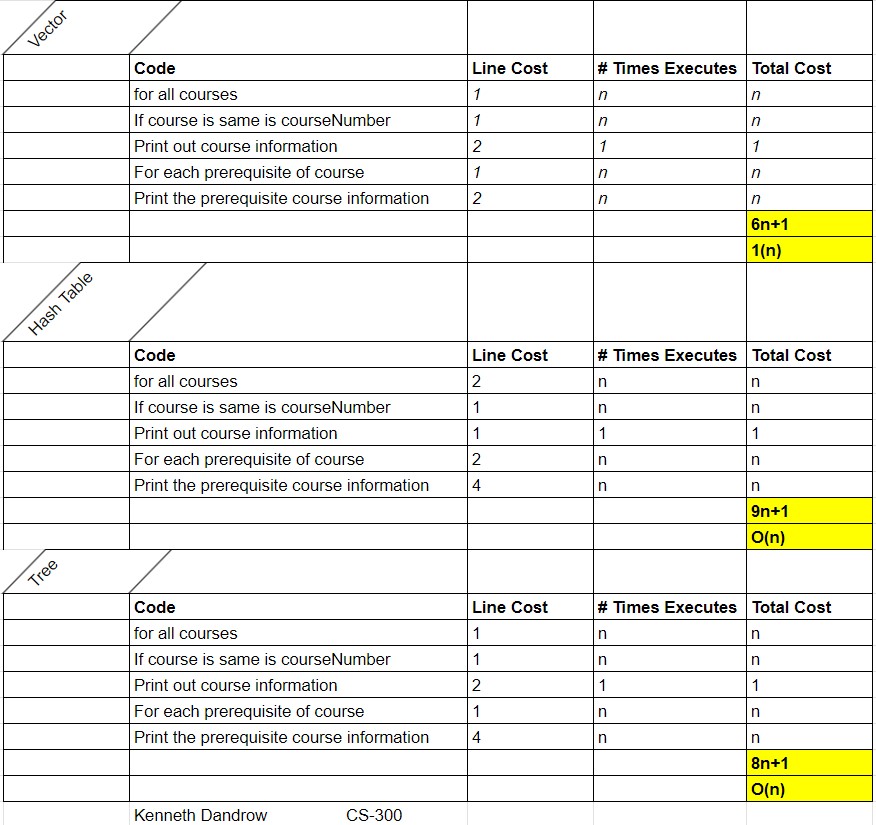
IN\_ORDER\_TRAVERSAL(node.right)

END IF

END FUNCTION

**Runtime Analysis:**





When you're considering which data structure is the fastest, it depends on what you need to do. If you're just loading data, a Hash Table is usually the quickest because it can store each item in a specific spot quickly. Searching for a course is also very fast with a Hash Table because you can usually find what you're looking for right away. However, if you want to print a list of courses in order, a Binary Search Tree is better. This is because a tree naturally keeps the data sorted, so you can easily print everything in the right order.

**Recommendation:**

I think the Hash Table is the best choice in this particular instance. The Hash Table can run slower but has many more pros like scalability, organization and efficiency.