Project One

Ashley Johnson

Southern New Hampshire University

CS 300

## Pseudocodes

**VECTOR DATA STRUCTURE**

* Opening/reading data

OPEN course information file

READ each line in the file

CHECK if the lines have at least two parameters

IF the lines have two or more parameters

PARSE parameters and store the parameters in variables

CHECK if the provided prerequisites in the line exist as a course

IF the prerequisite doesn’t exist

SKIP line and move to the next one

CREATE new course object with the parsed info

STORE course object in vector data structure

REPEAT all the lines until the file has been processed

END

* Creating course objects and storing in the appropriate vector data structure

DEFINE class named Course

DEFINE instance variables in Course class to store the course number, course title, and prereq

OPEN file that contains course info

READ each line in the file

PARSE parameters and store them in variables

CREATE new course object with the parsed info

STORE course object in vector data structure

REPEAT all the lines until the file has been processed

END

* Search the data structure and Print course info and prereq

INPUT user inputs the course number that prints out the course number and prereq

SEARCH vector data structure for course object that matches the input

IF object is found

PRINT the course number, course title, and prereq

ELSE course object not found

PRINT error message

END

**HASHTABLE**

INITIALIZE node using resize to change the size of table

SET table size to class size

RESIZE nodes to class

ERASE nodes from the beginning

RETURN key mode to the default size

CREATE a key for the given bid

SET unsigned key to the Hash function

RETRIEVE the node using a key

IF no entry is found for a key

ASSIGN node to key position

ELSE find next node that is open

ADD new node to the end

FOR iterates the beginning and the end of a node

IF key != UINT\_MAX

PRINT key, title, amount, funds, and bidId

SET node to next iterator

WHILE node != nullptr

PRINT key, title, amount, funds, and bidId

SET node = next node

SET unsigned key to HASH function

IF node key != UINT\_MAX

IF bidId matches

PRINT remove top node for bidId

IF next node = nullptr

SET node key = UINT\_MAX

RETURN bid

ELSE skip node for next node

RETURN bid

ELSE set pointer to current and next node

WHILE current node != nullptr

SKIP node

DELETE node

SET node to nullptr

RETURN

SET previous node to current

SET current node to next

CREATE key for given bid with HASH function

IF key entry is found

RETURN bid

IF key entry is not found

RETURN empty bid

WHILE node != nullptr

IF current node matches

RETURN bid

LOOP list until match is located

RETURN empty bid if nothing is matched

**BINARYSEARCHTREE**

* **Background codes**

CREATE class Binarysearchtree

DEFINE insert, remove, and search methods

CREATE root = null

* **Creates the insert method**

IF root = null

new item root

ELSE

IF item < root

INSERT left

IF left = null

INSERT item

ELSE

IF item > root

INSERT right

IF right = null

INSERT item

RETURN

* **Creates the remove method**

IF the root is empty

RETURN

ELSE

TRAVERSE left

IF node is found

DELETE node and SET to null

ELSE

TRAVERSE right

IF node is found

DELETE node and SET to null

* **Creates the search method**

IF root != null

TRAVERSE left and RETURN if item is found

ELSE

TRAVERSE right and RETURN if item is found

ELSE

RETURN if the item is not found

* **Displays the user’s menu**

Prompts the user for entry

IF the user enters one: load bids

READ file

DISPLAY file

PRINTS clock

IF the user enters two: all bids are displayed

ACCESS file

DISPLAYS all bids

IF the user enters three: find bids

IF the bid is not found

DISPLAY Bid is not found message

ELSE

DISPLAY the bid

IF the user enters four: remove bids

REMOVE bidKey

IF the user enters nine: Exit program

DISPLAY goodbye message

**MENU**

INTIALIZE user’s choice to equal 0

WHILE user input does not equal 9

PRINT Menu

PRINT “1. Load Data”

PRINT “2. Print list of courses”

PRINT “3. Print specific course”

PRINT “9. Exit menu”

PRINT “Make a selection”

INPUT user’s selection

IF selection != 1, 2, 3, or 9

PRINT “Invalid choice”

ELSE

IF selection = 1

LOAD course data

IF selection = 2

PRINT the list of courses

IF selection = 3

PRINT “Search for course number”

PRINT course information

IF selection = 9

PRINT “Goodbye”

Charts

**Vector**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 1 | 1 | 1 |
| **if the course is the same as courseNumber** | 1 | n | n |
| **print out the course information** | 1 | n | n |
| **for each prerequisite of the course** | 1 | 1 | 1 |
| **print the prerequisite course information** | 1 | 1 | 1 |
| **Total Cost** | | | 2n + 1 |
| **Runtime** | | | O(n) |

**HashTables**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 0 | n | n |
| **if the course is the same as courseNumber** | 0 | n | n |
| **print out the course information** | 0 | n | n |
| **for each prerequisite of the course** | 0 | n | n |
| **print the prerequisite course information** | 0 | n | n |
| **Total Cost** | | | 5n + 1 |
| **Runtime** | | | O(n) |

**Binary Tree Search**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 0 | n | n |
| **if the course is the same as courseNumber** | 0 | n | n |
| **print out the course information** | 0 | n | n |
| **for each prerequisite of the course** | 0 | n | n |
| **print the prerequisite course information** | 0 | n | n |
| **Total Cost** | | | 5n + 1 |
| **Runtime** | | | O(n) |

Advantages and Disadvantages

**Vector**

Vectors are arrays that are dynamic and provide a simple straightforward structure for data. Vectors has the ability to store and manage the course information. With my analysis, I see that Vectors are relatively efficient with loading the courses from the CSV file. The time complexity allows the vector to have quick data insertion thus making it appropriate for courses that are moderate in number. Vectors also has the ability to efficiency to search and print the course information based on the course number. But Vectors can pose an inefficiency with the sorting operation that is required for printing all the courses in alphabetical order.

**Hash Table**

Hash Tables are great for efficient data storage and data retrieval. While Hash Tables are not as fast as Vectors, it can still be used to load courses into a hash table. Hash Tables does allow for fast data inserting times. Hash tables can be great for pulling prerequisites for a course and printing the course information and prerequisites out if the prerequisites is the main focus but like Vectors the issue arises with placing the courses in alphabetical order because using a hash table does not guarantee the ability to sort thus causing the coder to user additional steps.

**Binary Tree Search**

Binary Tree Search is an organized hierarchical structure that is used for storing the course information. The runtime for the functions is reasonable for loading the courses into a binary tree search. The balanced structure of a Binary Tree provides rapid insertion times for the data. The advantages allows for the tree to be searched quick which allows for frequent data retrieval. The binary tree also has an efficient course for printing and sorting like Vectors.

**Recommendation**

Each of the data structures allows for several advantages and disadvantages when managing the course information. Vectors have a simple, straightforward structure while the time complexity makes it ideal for small operations and data sets. Hash tables are great for rapid data storage and data retrieval making them ideal for applications that need quick searches to happen but if there is chaining then hash tables are less efficient than vectors or binary tree searches. Binary Tree Search structures allow for quick searches and efficient printing but can need additional steps for sorting. I would recommend using the binary tree search structure to maintain the course information for the ABCU system because the binary tree has an efficient runtime for printing, and has advantages for sorting the courses in alphabetical order.