**Project One Pseudocode and Evaluation**

Carter Wooton

SNHU CS 300

October 9, 2022

**Pseudocode for vector data structure**

Reading data from the file:

CREATE fstream object to open the file

CALL function to open file

IF file not open:

DISPLAY error message and return

WHILE it is not the end of file:

READ line

IF there are less than two values in the line, return an ERROR

IF there are more than two values in the line:

FOR each value until end of line:

IF value does not equal value of first parameter from previous lines:

RETURN an error message

ELSE save values in the line for further use

END WHILE

CLOSE file

Create course objects and store in a data structure:

CREATE Course structure including courseId, courseName, and a vector<string> of preReqs

CREATE empty Courses Vector<Course>

LOOP through file using previous function

FOR each line:  
 READ first value and assign to courseId

READ second value and assign to courseName

READ each following value and place in the preReqs vector

ADD newly constructed Course object to the Courses vector

END FOR

Search data structure for specific course and print information:

LOOP through vector:

IF attribute of Course object matches the search value (coursed or name):

OUTPUT the Course name and ID

LOOP through the object’s prerequisite vector:

OUTPUT the Course prerequisites

END LOOP

**Pseudocode for hash table data structure**

Reading data from the file:

CREATE fstream object to open the file

CALL function to open file

IF file not open:

DISPLAY error message and return

WHILE it is not the end of file:

READ line

IF there are less than two values in the line, return an ERROR

IF there are more than two values in the line:

FOR each value until end of line:

IF value does not equal value of first parameter from previous lines:

RETURN an error message

ELSE save values in the line for further use

END WHILE

CLOSE file

Create course objects and store in a data structure:

CREATE Course structure including courseId, courseName, and a vector<string> of preReqs

CREATE empty HashTable of appropriate length

LOOP through file using previous function

FOR each line:  
 READ first value and assign to courseId

READ second value and assign to courseName

READ each following value and place in the preReqs vector

HASH the Course object using the Hashing function

IF the hashed key is empty:

INSERT the Course object at that key

ELSE:

Find free space in the chain and insert object in that free space

END FOR

Search data structure for specific course and print information:

TAKE courseId as function parameter

HASH the Course.courseId parameter using the Hashing function to get the appropriate key to search

IF the chain at the key is empty:

Return

ELSE:

ITERATE through the chain at the key until the Course objects courseId == the search parameter

OUTPUT the Course name and ID

LOOP through the object’s prerequisite vector:

OUTPUT the Course prerequisites

END IF

**Pseudocode for binary search tree data structure**

Reading data from the file:

CREATE fstream object to open the file

CALL function to open file

IF file not open:

DISPLAY error message and return

WHILE it is not the end of file:

READ line

IF there are less than two values in the line, return an ERROR

IF there are more than two values in the line:

FOR each value until end of line:

IF value does not equal value of first parameter from previous lines:

RETURN an error message

ELSE save values in the line for further use

END WHILE

CLOSE file

Create course objects and store in a data structure:

CREATE Course structure including courseId, courseName, and a vector<string> of preReqs

CREATE empty BinarySearchTree

LOOP through file using previous function

FOR each line:  
 READ first value and assign to courseId

READ second value and assign to courseName

READ each following value and place in the preReqs vector

ADD the course object to the tree:

IF the tree is empty:

INSERT the course object at the root

ELSE:

Recursively go down tree, going left if the new courseId < the node’s courseId, and right if the new courseId > the node’s courseId

Once a null node is found, SET that node to the Course Object

END FOR

Search data structure for specific course and print information:

TAKE courseId as function parameter

SET currentNode to root of tree

WHILE currentNode is not a nullptr:

IF currentNode’s courseId matches the courseId:

RETURN the currentNode’s Course Object

ELSE IF currentNode’s courseId is greater than the courseId:

SET currentNode to currentNode.left

ELSE IF currentNode’s courseId is less than the courseId:

SET currentNode to currentNode.right

END IF

END WHILE

RETURN empty Course Object if Id is not found

**Pseudocode for menu for all data structures**

WHILE user choice is not 9:

OUTPUT menu text:

“Menu:

1: Load courses

2: Print course list

3: Print course”

INPUT userChoice

IF choice is 1:

RUN function to load file and place data into data structure

ELSE IF choice is 2:

RUN function to loop through the data structure and print an alphanumerically ordered list of all the courses

ELSE IF choice is 3:

RUN function to search for specific course in the data structure and output the title and prerequisites for the course

ELSE IF choice is 9:

EXIT program

**Pseudocode to print list of courses in alphanumeric order**

**Vector data structure**  
(Sort vector data structure using selection sort)

SET integers i, j, and indexSmallest to 0

FOR each item in the course vector:

SET indexSmallest to i

FOR each item j after i:

IF the courseId at j is less than the courseId at indexSmallest:

SET indexSmallest to j

END FOR

SWAP course at i with course at indexSmallest

END FOR

(Print items from the vector after sorting)

FOR each item in the course vector:

OUTPUT courseId, course name, and list of prerequisites on one line

**Hash table data structure**

(Hash table data structure is already sorted when items are inserted, so the table only needs to be printed)

ITERATE through each key in the table:

IF key is not empty:

ITERATE through each node in the key

OUTPUT courseId, course name, and list of prerequisites on one line

**Binary search tree data structure**

(Binary search tree data structure is already sorted as items are inserted, so the tree only needs to be traversed and printed

FUNCTION inOrder(Node pointer of root node):

IF node does not equal nullptr:

CALL inOrder(node->left) //recursively call to traverse down the tree to the left until the smallest ID is found

OUTPUT Course attributes on one line

CALL inOrder(node->right) // recursively call to traverse down the tree to the right until the end of the tree is found

END FUNCTION

**Evaluation**

**Runtime tables for checking file for errors and storing course objects in each data structure**









**Evaluation of each data structure**

Vectors: The vector data structure is a simple storage solution for the courses and is easy to code and maintain. As seen in the runtime chart, the vector data structure is the fastest for storing the objects from the file. However, when searching for a course, the vector must be traversed since it is unordered. The vector must also be sorted before all the courses can be printed in alphanumerical order. Since the program is most likely to be used to print the courses and to search for a specific course, the vector data structure may not be the best choice.

Hash table: The hash table data structure gives each course a unique key based on a hash function, and each object can be searched using that key for a runtime of O(1), if designed properly. It would be effective for the program to create the hash table with many buckets, so that each course can be stored without any collision, since there shouldn’t be any courses with the same courseId. Creating the course objects and storing them in the hash table may not be as quick as storing them in a vector, but that may only occur once or maybe twice if the table needs to be resized. Removing and searching the table would be O(1), and printing the classes in order should also be O(1) if the table has at least as many buckets as course objects in the file, since they are already ordered.

Binary search tree: The binary search tree stores nodes with one parent and up to two children. The runtime for the binary tree is like the hash table for the worst case, but a typical case would only have to visit a few levels of the tree. The binary tree also has a slower runtime for storing objects into the data structure, but it makes up by being effective at printing all of the classes, since it is already sorted, and for searching, which is typically fast. The only issue with the binary tree would be if the course file began with a courseId that began with A, or another letter near the beginning of the alphabet, since the tree would have most of its nodes on one side, instead of an even distribution. This would make the binary search tree less effective than the hash table.

**Recommendation**

My recommendation would be to use the hash table data structure. It would be effective for the typical use of the program, which is to search for a course and print its attributes, and to print all the courses. There would be no possible issue with loading the data from the file and into the data structure, unlike with the binary search tree, and the typical search for a course would have O(1) runtime.