Brandon Mullins

CS-300

6-2 Project One

Southern New Hampshire University

**Pseduocode from Vector(Milestone #1):**

START

FUNCTION read\_file(filename):

OPEN fileStream with filename

IF unable to open fileStream:

PRINT "Unable to open file."

RETURN empty vector

END IF

lines = EMPTY vector

FOR each line in fileStream:

IF line format is incorrect:

PRINT "Error: Invalid line format."

CONTINUE

END IF

ADD line to lines

END FOR

CLOSE fileStream

RETURN lines

END FUNCTION

FUNCTION create\_course\_list(fileContents):

courses = EMPTY vector

FOR each line in fileContents:

CREATE new Course currentCourse

tokens = SPLIT line into tokens

count = 0

FOR each token in tokens:

IF count == 0:

currentCourse.courseNumber = token

ELSE IF count == 1:

currentCourse.courseTitle = token

ELSE:

IF courseNumberExistsInCourses(token, courses):

ADD token to currentCourse.prerequisites

ELSE:

PRINT "Error: Prerequisite " + token + " not found in course list."

END IF

count = count + 1

END FOR

IF count < 2:

PRINT "Error: Missing course number or title."

END IF

ADD currentCourse to courses

END FOR

RETURN courses

END FUNCTION

FUNCTION find\_course(courseList, targetCourseNumber):

FOR each course in courseList:

IF course.courseNumber == targetCourseNumber:

RETURN course

END IF

END FOR

RETURN empty course

END FUNCTION

FUNCTION course\_number\_exists\_in\_courses(targetCourseNumber, courseList):

FOR each course in courseList:

IF course.courseNumber == targetCourseNumber:

RETURN TRUE

END IF

END FOR

RETURN FALSE

END FUNCTION

FUNCTION print\_course\_info(courseList, targetCourseNumber):

targetCourse = CALL find\_course(courseList, targetCourseNumber)

IF targetCourse is empty:

PRINT "Course not found in schedule."

RETURN

END IF

PRINT "Course Number: " + targetCourse.courseNumber

PRINT "Course Title: " + targetCourse.courseTitle

IF targetCourse.prerequisites is empty:

PRINT "No prerequisites."

ELSE:

PRINT "Prerequisites:"

FOR each prerequisite in targetCourse.prerequisites:

PRINT prerequisite

END FOR

END IF

END FUNCTION

END

}**Pseduocode from Hash Table(Milestone #2):**

START

FUNCTION open\_and\_read\_file(file\_name):

OPEN file\_name for reading

IF file is not opened successfully:

PRINT "Error: File not found."

RETURN

END IF

FOR each line in file:

CALL parse\_and\_process\_line(line)

END FOR

CLOSE file

END FUNCTION

FUNCTION parse\_and\_process\_line(line):

SPLIT line into tokens

IF number of tokens < 2:

PRINT "Error: Invalid format, not enough parameters."

RETURN

END IF

course\_number = tokens[0]

course\_title = tokens[1]

prerequisites = EMPTY LIST

IF number of tokens > 2:

FOR each token from index 2 to end:

ADD token to prerequisites list

END FOR

CALL store\_course\_in\_hash\_table(course\_number, course\_title, prerequisites)

END FUNCTION

FUNCTION store\_course\_in\_hash\_table(course\_number, course\_title, prerequisites):

key = HASH(course\_number)

IF node at key is NULL:

// Create a new node

CREATE new\_course\_node as a CourseNode

new\_course\_node.course\_number = course\_number

new\_course\_node.course\_title = course\_title

new\_course\_node.prerequisites = prerequisites

new\_course\_node.next = NULL

INSERT new\_course\_node at hash key in hash table

ELSE:

current\_node = node at key

WHILE current\_node is not NULL:

IF current\_node.course\_number == course\_number:

UPDATE current\_node with new course information (optional)

RETURN

END IF

current\_node = current\_node.next

END WHILE

// insert as a new node

CREATE new\_course\_node as a CourseNode

new\_course\_node.course\_number = course\_number

new\_course\_node.course\_title = course\_title

new\_course\_node.prerequisites = prerequisites

new\_course\_node.next = node at key

INSERT new\_course\_node at hash key in hash table

END IF

END FUNCTION

FUNCTION print\_all\_courses():

// Loop the hash table

FOR index from 0 to table\_size - 1:

node = hash\_table[index]

IF node is not NULL:

PRINT "Course Number: ", node.course\_number

PRINT "Course Title: ", node.course\_title

PRINT "Prerequisites: ", node.prerequisites

// Traverse linked list

WHILE node.next is not NULL:

node = node.next

PRINT "Course Number: ", node.course\_number

PRINT "Course Title: ", node.course\_title

PRINT "Prerequisites: ", node.prerequisites

END IF

END FOR

END FUNCTION

FUNCTION HASH(course\_number):

RETURN HASH\_KEY = (course\_number % table\_size)

END FUNCTION

END

**Pseduocode from Binary Search Tree(Milestone #3):**

START

FUNCTION load\_and\_process\_file(file\_name):

OPEN file\_name for reading

IF file is not opened successfully:

PRINT "Error: File not found."

RETURN

END IF

FOR each line in file:

CALL parse\_and\_store\_course(line)

END FOR

CLOSE file

END FUNCTION

FUNCTION parse\_and\_store\_course(line):

SPLIT line into tokens

IF number of tokens < 2:

PRINT "Error: Invalid format, not enough parameters."

RETURN

END IF

course\_number = tokens[0]

course\_title = tokens[1]

prerequisites = EMPTY LIST

IF number of tokens > 2:

FOR each token from index 2 to end:

ADD token to prerequisites list

END FOR

CALL validate\_prerequisites(prerequisites, file)

CALL store\_course\_in\_tree(course\_number, course\_title, prerequisites)

END FUNCTION

FUNCTION validate\_prerequisites(prerequisites, file):

FOR each prerequisite in prerequisites:

IF prerequisite does not exist in file:

PRINT "Error: Prerequisite not found: " + prerequisite

RETURN

END IF

END FOR

END FUNCTION

FUNCTION store\_course\_in\_tree(course\_number, course\_title, prerequisites):

course = NEW Course(course\_number, course\_title, prerequisites)

IF tree.root IS null:

tree.root = NEW TreeNode(course)

ELSE:

CALL insert\_course\_into\_tree(tree.root, course)

END FUNCTION

FUNCTION insert\_course\_into\_tree(node, course):

IF course.course\_number < node.course.course\_number:

IF node.left IS null:

node.left = NEW TreeNode(course)

ELSE:

CALL insert\_course\_into\_tree(node.left, course)

ELSE:

IF node.right IS null:

node.right = NEW TreeNode(course)

ELSE:

CALL insert\_course\_into\_tree(node.right, course)

END FUNCTION

FUNCTION print\_course\_information():

CALL traverse\_tree\_in\_order(tree.root)

FUNCTION traverse\_tree\_in\_order(node):

IF node IS NOT null:

CALL traverse\_tree\_in\_order(node.left)

PRINT "Course Number: " + node.course.course\_number

PRINT "Course Title: " + node.course.course\_title

IF node.course.prerequisites IS NOT empty:

PRINT "Prerequisites: " + JOIN(node.course.prerequisites, ', ')

ELSE:

PRINT "Prerequisites: None"

CALL traverse\_tree\_in\_order(node.right)

END FUNCTION

FUNCTION main():

filename = "courses\_data.txt"

TRY:

CALL load\_and\_process\_file(filename)

CALL print\_course\_information()

CATCH FileNotFoundException:

PRINT "Error: File not found."

CATCH IOError:

PRINT "Error: Unable to read the file."

CATCH Exception AS e:

PRINT "Error:", e

END FUNCTION

END

**This is the table given for Vector**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 1 | n | n |
| **if the course is the same as courseNumber** | 1 | n | n |
| **for each prerequisite of the course** | 1 | 1 | 1 |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 1 | n | n |
| **Total Cost** | | | 4n + 1 |
| **Runtime** | | | O(n) |

**HashTable**

void printCourseInformation(Hashtable courses, String courseNumber) {

for all courses

if the course is the same as courseNumber

print out the course information

for each prerequisite of the Hashtable[course]

print the prerequisite course information

}

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 1 | n | n |
| **if the course is the same as courseNumber** | 1 | n | n |
| **for each prerequisite of the course** | 1 | 1 | 1 |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 1 | n | n |
| **Total Cost** | | | 4n + 1 |
| **Runtime** | | | O(n) |

**Binary Search Tree**

void printCourseInformation(Tree courses, String courseNumber) {

for all Nodes

if the course is the same as courseNumber

print out the node's information

if course has left node

print left node as prerequisite course information

if course has right node

print right node as prerequisite course information

end Function

else

if course has left node

goto left node

if course has right node

goto right node

}

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 1 | n | n |
| **if the course is the same as courseNumber** | 1 | n | n |
| **for each prerequisite of the course** | 1 | 1 | 1 |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 1 | n | n |
| **Total Cost** | | | 7n + 1 |
| **Runtime** | | | O(n) |

**Pseudocode:**

START

PRINT "Welcome to the ABCU Course Planner!

WHILE choice != 9:

PRINT "Pick an option:"

PRINT "1. Load Data"

PRINT "2. Print Course List"

PRINT "3. Print Course Info"

PRINT 9. Exit”

GET userChoice

IF userChoice == 1:

PRINT "Enter the file name to load the course data:"

GET filename

IF FILE\_EXISTS(filename):

CALL load\_courses\_from\_file(filename)

PRINT "Courses loaded successfully."

ELSE:

PRINT "This file was not found. Please try again."

ELSE IF userChoice == 2:

IF data\_loaded: // Check if data is loaded first

PRINT "Here are all the courses sorted by course number:"

CALL print\_all\_courses\_alphabetically()

ELSE:

PRINT "No courses loaded. Please load the data first."

ELSE IF userChoice == 3:

IF data\_loaded:

PRINT "Enter the course number to get info:"

GET courseNumber

CALL print\_course\_info(courses, courseNumber)

ELSE:

PRINT "No courses loaded. Please load the data first."

ELSE IF userChoice == 9:

PRINT "Thanks you for using my Course Planner. Goodbye!"

BREAK

ELSE:

PRINT "This is an invalid. Please try again."

END WHILE

END

**Evaluation:**

Based on all the previous pseudocode and strategies involved, I think I would recommend using a hash table for this project. I feel like the Hash table provides average time complexity, which I think is ideal for accessing course information by the course number. I don’t believe there are any disadvantages for using any of them(vector, hash table, and BST).