CS-300 DSA: Project 1

Austin Palmer

**Runtime Analysis**

Each data structure has unique advantages and disadvantages that make them useful in certain situations. A vector is an easy-to-use data structure that is included in the vector library. This data structure has built in functions that allow easy manipulation and search of data and can be sorted through multiple techniques such as quick sort, selection sort, and merge sort. A disadvantage of an array is you must access it through an index marker which may not be practical when trying to store large amounts of data or if the data is sorted. A hash table is a great data structure to use when storing data that has a unique identifying key like storing passwords. The biggest advantage of a hash table is its speed, On average, the access speed of a hash table is O(1) since unique keys are mapped to a specific index in an array. A disadvantage of a hash table is collisions which, if not handled properly, can mess with the organization of data. A binary tree allows for easy access to data while maintaining a decent speed with an average case of O(logN). Additionally, trees are best used when needing to manipulate a large amount of data using their search, insert, and delete operations. A disadvantage of a binary tree is it only works on sorted or structured data such as a list of numbers or IDs. In my analysis, the binary tree and hash table have the same worst-case runtimes of O(N) and the vector has a worse runtime of O(N^2). When it comes to the requirements and functionality of the program a hash table would be the most practical. The program for the advising team needs to have fast search capabilities and the ability to store information based on a course ID. A hash table could take a course ID, hash it, and store all prerequisite and course information while maintaining a runtime complexity of O(1). Additionally, the class data does not necessarily have to be sorted as there is a clear structure to course progression only that ID belongs to that course. Below is the runtime analysis of the provided pseudocode for the schedule and all pseudocode associated with this project.

Vector numPrerequisiteCourses

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| totalPrerequisites = prerequisites of course c | 1 | 1 | 1 |
| for each prerequisite p in totalPrerequisites | 1 | n | n |
| add prerequisites of p to totalPrerequisites | 1 | n | n |
| print number of totalPrerequisites | 1 | 1 | 1 |
| **Total Cost** | | | 2n+2 |
| **Runtime** | | | O(n) |

Vector printSampleSchedule

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| for each course in courses | 1 | n | n |
| print course information | 1 | n | n |
| **Total Cost** | | | 2n |
| **Runtime** | | | O(n) |

Vector printCourseInformation

| **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 | n^2 | n^2 |
| print the prerequisite course information | 1 | n | 1 |
| **Total Cost** | | | n^2+2n+1 |
| **Runtime** | | | O(n^2) |

Hashtable numPrerequisiteCourses

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| for course in courses | 1 | n | n |
| if course has preReq1 | 1 | n | n |
| increment numPreReqs | 1 | n | n |
| if course has preReq2 | 1 | n | n |
| return numPreReqs | 1 | n | n |
| **Total Cost** | | | 5n |
| **Runtime** | | | O(n) |

Hashtable printSampleSchedule

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| for course in courses | 1 | n | n |
| calculate course % buckets | 1 | n | n |
| if key at course is not null | 1 | n | n |
| print course information | 1 | n | n |
| **Total Cost** | | | 4n |
| **Runtime** | | | O(n) |

Hashtable printCourseInformation

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| for all courses | 1 | n | n |
| if courseNumber is the same as courses at key | 1 | n | n |
| print course information | 1 | n | n |
| if courses at key has preReq1 or preReq2 | 1 | n | n |
| print preReq information | 1 | n | n |
| **Total Cost** | | | 5n |
| **Runtime** | | | O(n) |
|  | | |  |

BST numPrequisiteCourses

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| while current node is not nullptr | 1 | n | n |
| if node has preReq1 | 1 | n | n |
| increment numPreReqs | 1 | n | n |
| if node has preReq2 | 1 | n | n |
| return numPreReqs | 1 | n | n |
| **Total Cost** | | | 5n |
| **Runtime** | | | O(n) |

BST printSampleSchedule

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| while current course is not nullptr | 1 | n | n |
| if courseNumber is less than current courseNumber | 1 | n | n |
| recurse left | 1 | n | n |
| else if courseNumber is more than current courseNumber | 1 | n | n |
| recurse right | 1 | n | n |
| else | 1 | n | n |
| print course information | 1 | n | n |
| **Total Cost** | | | 7n |
| **Runtime** | | | O(n) |

**Vector Pseudocode**

**void ReadFile(Vector<Course> courses):**

INPUT file name

SET fileName to input value

OPEN fileName

IF fileName is NOT open:

OUTPUT “Error opening file”

RETURN

WHILE not end of fileName:

READ fileName until next line

SET readLine to the line read

IF readLine does not have at least 1 comma on line:

RETURN

ELIF readLine has 2 or more commas:

FOR commas in readLine:

SET preReq to prerequisite course after comma

WHILE not end of file:

IF preReq is not in fileName more than once:

OUTPUT “Prerequisite course does not exist”

RETURN

SET vector at index to data

Increment index

CLOSE fileName

**Class Course:**

// Constructors

Course(courseID, courseTitle):

SET this courseID to courseID

SET this courseTitle to courseTitle

Course(courseID, courseTitle, preReq1):

SET this courseID to courseID

SET this courseTitle to courseTitle

SET this preReq1 to preReq1

Course(courseID, courseTitle, preReq1, preReq2):

SET this courseID to courseID

SET this courseTitle to courseTitle

SET this preReq1 to preReq1

SET this preReq2 to preReq2

// Getters

getCourseID():

RETURN courseID

getCourseTitle():

RETURN courseTitle

getPreReq1():

RETURN preReq1

getPreReq2():

RETURN preReq2

// Setters

setCourseID(courseID):

SET this courseID to courseID

setCourseTitle(courseTitle):

SET this courseTitle to courseTitle

setPreReq1(preReq1):

SET this preReq1 to preReq1

setPreReq2(preReq2):

SET this preReq2 to preReq2

**void courseSearch(Vector<Course> courses, Course c):**

FOR each course in courses:

IF c is course:

printCourseInformation(courses, course.getCourseID)

printPrerequisiteCourses(courses, c)

**int numPrerequisiteCourses(Vector<Course> courses, Course c):**

SET totalPrerequisites to prerequisites of course c

FOR each prerequisite p in totalPrerequisites:

APPEND prerequisites of p to totalPrerequisites

PRINT number of totalPrerequisites

**void printCourseInformation(Vector<Course> courses, String courseNumber):**

FOR all courses:

IF the course is the same as courseNumber:

PRINT out the course information

FOR each prerequisite of the course:

PRINT the prerequisite course information

**Hash Pseudocode**

**void ReadFile(Hash<String> courses):**

INPUT file name

SET fileName to input value

OPEN fileName

IF fileName is NOT open:

OUTPUT “Error opening file”

RETURN

WHILE not end of fileName:

READ fileName until next line

SET readLine to the line read

IF readLine does not have at least 1 comma on line:

RETURN

ELIF readLine has 2 or more commas:

FOR commas in readLine:

SET preReq to prerequisite course after comma

WHILE not end of file:

IF preReq is not in fileName more than once:

OUTPUT “Prerequisite course does not exist”

RETURN

CALCULATE bucket index

STORE data in hash at bucket index

CLOSE filename

**Class Course:**

// Constructors

Course(courseID, courseTitle):

SET this courseID to courseID

SET this courseTitle to courseTitle

Course(courseID, courseTitle, preReq1):

SET this courseID to courseID

SET this courseTitle to courseTitle

SET this preReq1 to preReq1

Course(courseID, courseTitle, preReq1, preReq2):

SET this courseID to courseID

SET this courseTitle to courseTitle

SET this preReq1 to preReq1

SET this preReq2 to preReq2

// Getters

getCourseID():

RETURN courseID

getCourseTitle():

RETURN courseTitle

getPreReq1():

RETURN preReq1

getPreReq2():

RETURN preReq2

// Setters

setCourseID(courseID):

SET this courseID to courseID

setCourseTitle(courseTitle):

SET this courseTitle to courseTitle

setPreReq1(preReq1):

SET this preReq1 to preReq1

setPreReq2(preReq2):

SET this preReq2 to preReq2

**int numPrerequisiteCourses(Hashtable<Course> courses):**

FOR course in courses:

IF course has preReq1:

INCREMENT numPreReqs

IF course has preReq2:

INCREMENT numPreReqs

RETURN numPreReqs

**void printSampleSchedule(Hashtable<Course> courses):**

FOR course in courses:

CALCULATE course % buckets

IF key at course is not null:

PRINT course information

**void printCourseInformation(Hashtable<Course> courses, String courseNumber)**

FOR all courses:

IF courseNumber is the same as courses at key:

PRINT course information

IF courses at key has preReq1 or preReq2:

PRINT preReq information

**BST Pseudocode**

**void ReadFile(Tree<String> courses):**

INPUT file name

SET fileName to input value

OPEN fileName

IF fileName is NOT open:

OUTPUT “Error opening file”

RETURN

WHILE not end of fileName:

READ fileName until next line

SET readLine to the line read

IF readLine does not have at least 1 comma on line:

RETURN

ELIF readLine has 2 or more commas:

FOR commas in readLine:

SET preReq to prerequisite course after comma

WHILE not end of file:

IF preReq is not in fileName more than once:

OUTPUT “Prerequisite course does not exist”

RETURN

// Implemented through an addNode method

RECURSE courses by courseId

CLOSE filename

**void Insert(Tree<Course> courses, String courseNumber):**

IF node courseNumber is greater than current courseNumber:

IF node left is nullptr:

INSERT node left

ELSE:

RECURSE left

ELSE:

IF node right is nullptr:

INSERT node right

ELSE:

RECURSE left

**Class Course:**

// Constructors

Course(courseID, courseTitle):

SET this courseID to courseID

SET this courseTitle to courseTitle

Course(courseID, courseTitle, preReq1):

SET this courseID to courseID

SET this courseTitle to courseTitle

SET this preReq1 to preReq1

Course(courseID, courseTitle, preReq1, preReq2):

SET this courseID to courseID

SET this courseTitle to courseTitle

SET this preReq1 to preReq1

SET this preReq2 to preReq2

// Getters

getCourseID():

RETURN courseID

getCourseTitle():

RETURN courseTitle

getPreReq1():

RETURN preReq1

getPreReq2():

RETURN preReq2

// Setters

setCourseID(courseID):

SET this courseID to courseID

setCourseTitle(courseTitle):

SET this courseTitle to courseTitle

setPreReq1(preReq1):

SET this preReq1 to preReq1

setPreReq2(preReq2):

SET this preReq2 to preReq2

**int numPrerequisiteCourses(Tree<Course> courses) {**

WHILE current node is not nullptr:

IF node has preReq1:

INCREMENT numPreReqs

IF node has preReq2:

INCREMENT numPreReqs

RETURN numPreReqs

**void printSampleSchedule(Tree<Course> courses):**

WHILE current course is not nullptr:

PRINT course

RCURSE

**void printCourseInformation(Tree<Course> courses, String courseNumber)**

WHILE current course is not nullptr:

IF courseNumber is less than current courseNumber:

RECURSE left

ELSE IF courseNumber is more than current courseNumber:

RECURSE right

ELSE

PRINT course information

**Menu Pseudocode**

**int main()**

**//** Data structure depends on the chosen structure of the project

INITIALIZE data structure courses

SWITCH:

// Load data structure and input data

CASE 1:

// Call readFile from above pseudocode

readFile(courses)

BREAK

// Print all courses

CASE 2:

// Call printSampleSchedule from above pseudocode

printSampleSchedule(courses)

BREAK

// Print specific course

CASE 3:

INPUT course ID

SET courseId to input

// Call printCourseInformation from above pseudocode

printCourseInformation(courses, courseId)

BREAK

// Exit

CASE 4:

PRINT “Exiting…”

RETURN -1

**Alpha Numeric Sorting/Printing Pseudocode**

**int Sort(vector<Course> courses):**

SET size to size of courses

FOR every course while less than size:

SET current to current course index

FOR every course to the right of current course:

IF current course name is greater than right course name:

SET current to right course index

IF current index is not minimum index:

SWAP bids at current index and bids at minimum index

RETURN courses

**void Print(vector<Course> courses):**

FOR every course in courses:

PRINT course