Jasmine Zeng

Professor Terry Atkison

CS 300

8 February 2023

**Project One**

**Vector Pseudocode**

**File Input Function**

GET file

IF file cannot be opened

PRINT file not found error message

ELSE file can be opened

WHILE file does not reach the end

READ each line

IF line has one or less elements in line

PRINT course information error message

ELSE

READ remainder of line

IF line has three or more elements

IF these elements are the first element in another line

READ remainder of line

ELSE print nonexistent course error message

CLOSE file

**Course Object Function**

INITIALIZE course vector

WHILE file does not reach the end

FOR each file line

FOR first and second element in line

ADD values to course vector

IF third or later values exist

FOR each value until a new line

ADD value to vector

**Print Course Information Function**

INITIALIZE course input

GET course input from advisor

FOR each initial element per line in the course vector

IF course input matches initial element

PRINT course number

PRINT course name

IF course has prerequisites

FOR each prerequisite

PRINT prerequisite course number

**Hashtable Pseudocode**

**File Input Function**

GET file

IF file cannot be opened

PRINT file not found error message

ELSE file can be opened

WHILE file does not reach the end

READ each line

IF line has one or less elements in line

PRINT course information error message

ELSE

READ remainder of line

IF line has three or more elements

IF these elements are the first element in another line

READ remainder of line

ELSE print nonexistent course error message

CLOSE file

**Course Object Function**

INITIALIZE course vector

INITIALIZE hashtable class

INITIALIZE insert method for hashtable class

WHILE file does not reach the end

FOR each file line

FOR first and second element in line

INTIALIZE storage for these elements

IF third or later values exist

ADD each value to storage

CALL insert method to insert stored elements into hashtable

**Print Course Information Function**

INITIALIZE course input

GET course input from advisor

IF key is in hashtable

PRINT course number

PRINT course name

IF course has prerequisites

FOR each prerequisite

PRINT prerequisite course number

**Tree Pseudocode**

**File Input Function**

GET file

IF file cannot be opened

PRINT file not found error message

ELSE file can be opened

WHILE file does not reach the end

READ each line

IF line has one or less elements in line

PRINT course information error message

ELSE

READ remainder of line

IF line has three or more elements

IF these elements are the first element in another line

READ remainder of line

ELSE print nonexistent course error message

CLOSE file

**Course Object Function**

INITIALIZE course

WHILE file does not reach the end

FOR each file line

FOR first and second element in line

INSERT course code

INSERT course name

IF third or later values exist

ADD each value to storage until new line

**Print Course Information Function**

WHILE node does not point to null

IF searched course exists

PRINT course code and name

SET integer value for course prerequisite(s) amount

IF course has prerequisite(s)

FOR each prerequisite

PRINT prerequisite information

IF there is more than one prerequisite

PRINT comma and space

ENDIF

ENDFOR

ENDIF

ELSE course does not have prerequisites

PRINT no prerequisites notification

ENDELSE

ENDIF

IF course number input is smaller than current course number in binary tree

DECREMENT to the left

ENDIF

ELSE IF course number input is bigger than current course number in binary tree

INCREMENT to the right

ENDELSEIF

ELSE

PRINT course not found notification

ENDELSE

ENDWHILE

**General Pseudocode**

**Menu Function**

INITIALIZE load file variable

INITIALIZE course number variable

SET load file variable to name of file containing course data

SET course number variable to any course number in the file for testing

INITIALIZE menu choice variable

WHILE menu choice variable is not exit command

SWITCH choice

CASE 1

IMPORT course data with load file variable

ENDCASE

CASE 2

PRINT alphanumerical list of all courses in file

ENDCASE

CASE 3

PRINT course information from file with course number variable

ENDCASE

ENDSWITCH

ENDWHILE

RETURN zero to exit program

**Print Alphanumerically Ordered Course Information Function**

INITIALIZE iterator variable, course list length variable, and compare variable

FOR each line in the data file

FOR each course

SET compare variable to comparison between current course and next course

IF current course has a higher value than the next course

SWAP lines

ENDIF

ENDFOR

ENDFOR

**Evaluation**

**Data Structure Features and Run Time Calculations**

| **Vector Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **initialize course input** | 1 | 1 | 1 |
| **get course input from advisor** | 1 | 1 | 1 |
| **for all courses** | 1 | n | n |
| **if the course is the same as courseNumber** | 1 | n | n |
| **print out the course information** | 2 | 2 | 2 |
| **if the course has prerequisite(s)** | 1 | 1 | 1 |
| **for each prerequisite** | 1 | n | n |
| **print the course number** | 1 | n | n |
| **Total Cost** | | | 4n + 5 |
| **Runtime** | | | O(n) |

The vector data structure provides an intermediate run time of all the presented data structures. Vectors offer strong searching features which will allow advisors to quickly find courses and their respective information, like its course code and prerequisite(s). Additionally, vector sizes can be tailored and managed easily. This limits the amount of memory it will take up in the school’s system. However, vectors may not be ideal for a system with large and complex datasets – for example, the system may contain course data for other majors, concentrations, and disciplines; lower-level courses in this data set may count as general education courses as well.

| **Hashtable Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **initialize course input** | 1 | 1 | 1 |
| **get course input from advisor** | 1 | 1 | 1 |
| **if the course is the same as courseNumber** | 1 | n | n |
| **print out the course information** | 2 | 2 | 2 |
| **if the course has prerequisite(s)** | 1 | 1 | 1 |
| **for each prerequisite** | 1 | n | n |
| **print the course number** | 1 | n | n |
| **Total Cost** | | | 3n + 5 |
| **Runtime** | | | O(n) |

The hash table data structure provides a good run time out of the presented data structure options. This feature is helpful to advisors when handling multiple students’ requests and schedules, especially before the fall and spring semesters. However, hash tables naturally store data in random fashion. Since it would be more efficient to sort this data by each course’s level of difficulty and prerequisite requirements, random ordering can lead to volatile run times. Sorting would require an additional variable which would produce longer run times.

| **Binary Tree Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **while node does not point to null** | 1 | n | n |
| **if the course is the same as courseNumber** | 1 | n | n |
| **print out the course information** | 2 | 2 | 2 |
| **set integer value for course prerequisite amount** | 1 | 1 | 1 |
| **if the course has prerequisite(s)** | 1 | 1 | 1 |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course number** | 1 | n | n |
| **if there is more than one prerequisite** | 1 | 1 | 1 |
| **print comma and space** | 1 | n | n |
| **else the course does not have prerequisites** | 1 | 1 | 1 |
| **print no prerequisites notification** | 1 | 1 | 1 |
| **if the course input is less than the current course node** | 1 | 1 | 1 |
| **decrement left** | 1 | 1 | 1 |
| **if the course input is greater than the current course node** | 1 | 1 | 1 |
| **increment right** | 1 | 1 | 1 |
| **else print course not found notification** | 1 | 1 | 1 |
| **Total Cost** | | | 5n + 12 |
| **Runtime** | | | O(n) |

The binary tree data structure has a more meticulously written program, producing the longest run time. However, they naturally sort data sets and behave in a responsive manner; iterating left to right depending on the value of the searched item and the value of the current node it is traversing. The greatest weakness of this data structure is that it does not bode well with deletion. This can be an issue for universities constantly updating their course catalog. Binary trees also produce null pointers, which may lead to random errors when advisors are trying to use the system.

**Recommendation**

Based on these findings, I recommend ABC University (ABCU) to use vectors for their course information software. Vectors provide an adequate run time out of all the presented data structures, given how flexible they are compared to the others. They offer natural sorting powers that hashtables do not. Additionally, they can edit or delete elements, which binary trees are not efficient at. As a standalone, they are great at searching and retrieving elements. All of these features are necessary for advisors to complete their responsibilities in an efficient manner. This is under the assumption that ABCU advisors are tasked with typical requirements like handling multiple students’ degree progress and scheduling under time constraints; distributing course pathways; editing course pathways; and finding details of particular courses.