Steven Klabunde

Southern New Hampshire University

Project One

Saba Jamaliannasrabadi

October 11, 2024

**Table Contents**

Vector Pseudocode & runtime…………………………………………………………………………………………page 3

Hash Table Pseudocode & runtime………………………………………………………………………………….page 6

Binary Search Table Pseudocode & runtime………………………………………………………………….page 10

Comparisons of Data Structures & Recommendations………………………………………………………….page 13

**Vector Data Structure**

**REQUIREMENT 1**

DEFINE variable for csv path and my file

INITIALIZE csv path

OPEN my file into csv path

CREATE a catch statement to verify file opened correctly

           CREATE an if statement to output file not found if not opened correctly

DEFINE and initialize variable for line

DEFINE a vector for courses

DEFINE variables for courseNumber, name, and prerequisite

CREATE a while statement to loop through all lines until an empty line is found

           PARSE each line into file

SPLIT line by comma into tokens

          CREATE an if statement to verify the line has at least two tokens

CONTINUE if two tokens found

OUTPUT error if less than two tokens

ASSIGN token[0] to courseNumber

ASSIGN token[1] to courseName

ASSIGN token[2] to prerequisite

CREATE if statement for each line

CHECK for prerequisite

CONTINUE if no prerequisites found

CREATE if statement if prerequisite is found

SEARCH existing course numbers and compare to prerequisite

CONTINUE if match found

OUTPUT error if match not found

**REQUIREMENT 2**

DEFINE structure named course

DEFINE courseNumber, courseTitle, prerequisite

INITIALIZE course vector

CREATE newCourse (courseNumber, courseTitle, prerequisites)

USE push back to add newCourse to the vector

CHECK for prerequisite

CONTINUE if no prerequisites found

CREATE if statement if prerequisite is found

SEARCH existing course numbers and compare to prerequisite

CONTINUE if match found

OUTPUT error if match not found

**REQUIREMENT 3**

CREATE a method to search for a course

READ input to find course being looked for

DEFINE a temp variable to hold input

CREATE a for loop to read each line

CREATE an if statement

OUTPUT course information if temp variable equals courseNumber

OUTPUT “Course Number: “ and courseNumber

OUTPUT “Course Title: “ and courseTitle

CREATE if statement

OUTPUT “Prerequisites: “ and prerequisites if prerequisites exist

CONTINUE if no prerequisites exist

OUTPUT error if no match found

The vector data structure has a linear time complexity which is O(n).  The cost per line is 1 for all courses.  As there are n courses, the runtime will be 1n or n.  To print the course information the runtime cost will be 1 as only 1 line will need to be read.  Each prerequisite of the course will also be 1n and printing the prerequisite course information will also be one n.  When this si add3ed together this gives 4n + 1 which in big O is O(n).

| **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) |

**Hash Table Data Structure**

IMPORT all needed libraries

DEFINE hash table structure to hold course data

CREATE and initialize variables for course title, course number, and prerequisites

CREATE class for hash table

           CREATE structure node

INITIALIZE course, key, and next pointer

CREATE method for hash

CREATE method to output list

CREATE method to use a list for hash table

CREATE main method

DEFINE variable for csv path and my file

INITIALIZE csv path

OPEN my file into csv path

CREATE a catch statement to verify file opened correctly

           CREATE an if statement

OUTPUT error of “file not opened” if unable to read file.

DEFINE and initialize variable for line

DEFINE variables for courseNumber, name, and prerequisite

CREATE a for statement to loop through all lines

            CREATE text parser method

OPEN the need csv file

CREATE a while statement to loop through all lines until an empty line is found

          PARSE each line into file

SPLIT line by comma into tokens

          CREATE an if statement to verify the line has at least two tokens

CONTINUE if two tokens found

OUTPUT error if less than two tokens

ASSIGN token[0] to courseNumber

ASSIGN token[1] to courseName

ASSIGN token[2] to prerequisite

CREATE if statement for each line

CHECK for prerequisite

CONTINUE if no prerequisites found

CREATE if statement if prerequisite is found

SEARCH existing course numbers and compare to prerequisite

CONTINUE if match found

OUTPUT error if match not found

RETURN list

DEFINE structure named course

DEFINE courseNumber, courseTitle, prerequisite

INITIALIZE hash table to store courses

CREATE method to insert newCourse

READ input for newCourse

ASSIGN input to tempCourse variable

CREATE if statement

ASSIGN tempCourse to courseTable[key] if empty

CREATE while loop if not empty

LOOP through line until empty [key] found

ASSIGN tempCourse to the empty courseTable[key]

CREATE search method

CREATE temp bucket

OBTAIN value being search for

ASSIGN that value to temp variable

CREATE while loop to read each course

CREATE if statement if course equals temp variable

SET temp variable to course

RETURN temp variable

CREATE print course method

CREATE temp bucket

SET temp bucket equal to hash

CREATE a while loop to read each bucket

CREATE output for course information found in bucket

CREATE validate list method

CREATE temp bucket

CREATE variable valid and assign as true

CREATE a while loop for to read each course

CREATE if statement if valid equals true

RETURN valid

CREATE else statement if valid equals false

CREATE break

END

Hash tables have a best case runtime of O(1) and a worst case runtime of O(n). As more buckets get filled, reading through each line can take longer every time. The cost per line is one and will execute n times. This gives a run time of 1n. This stays the same if the course is the same as courseNumber. Print out course and prerequisite information has a line cost of one and executes one time. The total run time is 2n+2 or O(n).

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **While there are courses** | 1 | n | n |
| **if the course is the same as courseNumber** | 1 | n | n |
| **Print out the course information** | 1 | 1 | 1 |
| **Print the prerequisite of the course information** | 1 | 1 | 1 |
| **Total Cost** | | | 2n + 2 |
| **Runtime** | | | O(n) |

**Tree Data Structure Pseudocode**

IMPORT all needed libraries

DEFINE the binary tree structure to hold course data

CREATE and initialize variables for course title, course number, and prerequisites

DEFINE tree node class

CREATE structure binary tree

INITIALIZE course, key, and next pointer

CREATE method for tree

CREATE method to output list

CREATE a constructor to initialize the tree node

CREATE a class for Binary Search

CREATE a constructor to initialize binary search tree

Assign null to this.root

CREATE a method to insert a course

CREATE if statement for if root is null, the new node is the root

CREATE else statement to insert node if root is not null

CREATE a method for inserting course via recursive method

CREATE an if else statement comparing higher/lower nodes

CREATE an if statement looking for matches

OUTPUT course already in system

INSERT course into the correct location if no match found

CREATE a method to search tree for node

DEFINE a current note

CREATE a while loop to loop through all data

CREATE an if statement to determine if match found

CREATE an output stating course already in file if match found

CREATE an else statement to continue search if match not found

CREATE an output if no match found

CREATE main method

DEFINE variable for csv path and my file

INITIALIZE csv path

OPEN my file into csv path

CREATE a catch statement to verify file opened correctly

           CREATE an if statement to output file not found if not opened correctly

DEFINE and initialize variable for line

DEFINE variables for courseNumber, name, and prerequisite

CREATE a for statement to loop through all lines

            CREATE text parser method to parse each line and check for errors

OPEN the needed csv file

CREATE a while loop to go through each line

CREATE if statement verifying 1st and 2nd string are there

OUTPUT error statement if course number or name not included

VALIDATE prerequisites

OUTPUT error if prerequisite not already in file

CREATE a new course

INPUT all needed course information

CALL search method

CREATE if statement, if not found

CALL add node method

CREATE else statement, if found

OUTPUT error

CREATE print course method

CREATE a while loop to read each node

CREATE if statement to find matching course in system

OUTPUT course not found if no match

CREATE if statement if match found

OUTPUT for two prerequisites – cousreNumber – courseName – prerequisitesOne – prerequisitesTwo.

OUTPUT for one prerequisite – cousreNumber – courseName – prerequisitesOne .

OUTPUT for zero prerequisites – cousreNumber – courseName

END

Binary search trees have a best case run time of O(Logn) and a worst case run time of O(n). This will depend on whether the tree is perfect and/or complete. There are different ways to travel the tree which are in order, preorder, and post order. This determines if the user starts from the left side, the right side, the top or root, or the bottom nodes.

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **While node is not null** | 1 | n | n |
| **Travers left subtree** | 1 | Logn | Logn |
| **Travers right subtree** | 1 | Logn | Logn |
| **Print the prerequisite course information** | 1 | 1 | 1 |
| **Total Cost** | | | n + 2Logn |
| **Runtime** | | | O(Logn) |

Comparisons of Data Structures

Using a vector data structure’s main advantage is the simplicity of the code. It is easier to write and also easier to read and understand than the other data structures. Vectors are resizable so they can be adjusted as the data grows larger. One of its main disadvantages is that it can be slower, especially when performing a search function. When inserting new data, reordering the information can take multiple steps resulting in a worsening runtime.

Hash tables have a faster loop up due to using the hash function. No shifting is needed to insert new data. Collisions can be handled efficiently with proper code and will be sorted into the correct buckets. It is better on memory as it holds the pairs of keys and values and doesn’t need any other information. One big disadvantage of hash tables is the limited space as eventually all buckets will fill. The code is more complex than a vector. Data is not stored in any specific order as it is controlled by the keys.

Binary search trees allow for fast and efficient searches. This is because the information is stored in a very specific order. Data is relatively easy to both insert and delete. The code is fairly to write and understand. A larger disadvantage to binary search trees is that it can take a lot of memory due to linking the parent and child. Searching can be very quick if the tree is perfect and complete, but an unbalanced tree can lead to inefficient searches.

Recommendations

I would not recommend using the vector data structure. Out of the three structures this is the clunkiest and most inefficient method. The Binary search tree and the hash table both have advantage a big consideration will be the size of the end file. Because hash tables are not a easy to resize a file with growing amount of data would be better suited for the binary search table because the university may add more courses. I would recommend the binary search tree. This allows for searching of courses so advisor can verify the student has the correct prerequisites.