**Project 1**

**COURSE STRUCTURE**

Vector<course> preReqs;

int courseNumber;

string courseName;

**READING FILE TO ALL DATA STRUCTURES**

**loadCourses()**

INITIALIZE file and input stream

INITIALIZE temp as character array

INITIALIZE vector tmpCourse

INITIALIZE vector courses

OPEN file

IF file is not open

PUT Could not open file to ouput

FOR line in file

SET line to line input

FOR character in line

IF character is not comma

PUSH back character in temp

ELSE

PUSH back temp in tmpCourse

IF tmpCourse size is less than 2

CLOSE file

PUT error message

RETURN

DECLARE course object

SET course number to tmpCourse at index 0

SET course name to tmpCourse at index 1

SET course prerequisites to remaining indexes in tmpCourse

**(USE INSERT FUNCTION FOR WHICHEVER DATA STRUCTURE HERE**

**WITH COURSE AS THE PARAMETER)**

FOR course in courses

IF current course’s prerequisites are not in courses

REMOVE prerequisite

CLOSE file

**PRINTING COURSE FOR EACH DATA STRUCTURE**

**VECTOR**

**PrintCourse(Vector<Course> courses)**

PUT “Enter course number: “ to output

SET cnumber to input

FOR course in courses

IF course is same as cnumber

PRINT course information

FOR prerequisites of course

PRINT prerequisites course information

**HASHTABLE**

**PrintCourse(courseNumber)**

SET key to courseNumber hashed

INITIALIZE node with courseTable at key

IF node is null or node key is UINT\_MAX

PUT error to output

ELSE IF nodes courseNumber is same as passed courseNumber

PUT nodes course info to output

ELSE

WHILE node is not null

IF nodes courseNumber is same as courseNumber

PUT course info to output

SET node to next node

**BINARY SEARCH TREE**

**PrintCourse(courseNumber)**

INITIALIZE node as root

WHILE node is not null

IF nodes couseNumber is same as courseNumber

PUT course to output

FOR course in courses prereqs

PUT course info to ouput

ELSE

IF nodes courseNumber is greater than courseNumber

SET node to left node

ELSE

SET node to right node

**Menu**

INITIALIZE choice with 0

INITIALIZE BST as CourseList

WHILE choice is not 4

PUT “MENU”

“1: Load Courses

“2: Print Course List

“3: Print Course

“4: Exit

GET input for choice

SWITCH with choice

CASE 1

LoadCourses(file, CourseList)

CASE 2

PUT CourseList to output with Print function

BREAK

CASE 3

PUT “Enter course number”

GET input for courseNumber

PUT CourseList.Search(courseNumber) to output

BREAK

CASE 4

PUT “Goodbye” to output

EXIT()

DEFAULT

PUT “Invalid choice. Try again”

**VECTOR RUNTIME**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| FOR course in courses | 1 | n | n |
| IF course is same as cnumber | 1 | n | n |
| PRINT course information | 1 | 1 | 1 |
| FOR prerequisites of course | 1 | 2 | 3 |
| PRINT prerequisites course information | 1 | 1 | 1 |
| **Total Cost** |  |  | 5 + 2n |
| **Runtime** |  |  | O(n) |

**HASHTABLE RUNTIME**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| SET key to courseNumber hashed | 1 | 1 | 1 |
| INITIALIZE node with courseTable at key | 1 | 1 | 1 |
| IF node is null or node key is UINT\_MAX | 1 | 1 | 1 |
| PUT error to output | 1 | 1 | 1 |
| ELSE IF nodes courseNumber is same as passed courseNumber | 1 | 1 | 1 |
| PUT nodes course info to output | 1 | 1 | 1 |
| ELSE | 1 | 1 | 1 |
| WHILE node is not null | 1 | n | n |
| IF nodes courseNumber is same as courseNumber | 1 | 1 | 1 |
| PUT course info to output | 1 | 1 | 1 |
| SET node to next node | 1 | 1 | 1 |
| **Total Cost** |  |  | 10 + n |
| **Runtime** |  |  | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| INITIALIZE node as root | 1 | 1 | 1 |
| WHILE node is not null | 1 | Log(n) | Log(n) |
| IF nodes couseNumber is same as courseNumber | 1 | 1 | 1 |
| PUT course to output | 1 | 1 | 1 |
| FOR course in courses prereqs | 1 | 1 | 1 |
| PUT course info to ouput | 1 | 1 | 1 |
| ELSE | 1 | 1 | 1 |
| IF nodes courseNumber is greater than courseNumber | 1 | 1 | 1 |
| SET node to left node | 1 | 1 | 1 |
| ELSE | 1 | 1 | 1 |
| SET node to right node | 1 | 1 | 1 |
| **Total Cost** |  |  | 10 + log(n) |
| **Runtime** |  |  | Log(n) |

**BINARY SEARCH TREE RUNTIME**

**ALL STRUCTURES PRINTING RUNTIMES ARE EQUIVALENT TO LOADING THE BIDS WITH EACH STRUCTURE.**

**VECTOR**

Advantages:

Predefined functions

Indexing

Resizable

Disadvantages:

Manual Sorting

Shifting required for removing

**HASHTABLE**

Advantages:

Fast runtime

Key indexing

Disadvantages:

Complexity

Memory

**BINARY SEARCH TREE**

Advantages:

Fast runtime

Pre-sorted

Disadvantages:

First insert determines balance

Recursion leaves room for error and overflow

**CHOICE OF STRUCTURE**

**&**

**JUSTIFICATION**

Each structure has its advantages and disadvantages. My choice of structure to hold each course would be a BST. The binary search tree can be pre-sorted upon loading the list allowing for ease of implementation of printing an alphanumeric ordered list. The runtime of this structure is also reliably quick and allows for quick searching for individual courses. For any type of schedule format with prioritized events or “items”, my choice of structure would be a tree as it is the simplest to utilize given these parameters.

SIDENOTE: “Big O Notation” and “Runtime Complexity” is still a bit of a challenge for me. If you have any tips I will gladly accept them. I fully understand each structure and how each can be faster than the other in specific cases as well as how they can be utilized in different practices but analyzing runtime efficiency of an algorithm still eludes me somehow. Thanks if there is anything to correct please let me know.