**Project One**

Justin Starr

Department of STEM

CS 300 – DSA: Analysis and Design

Professor Ricardo Scarello

June 16, 2024

**Project One**

**Pseudocode for Vector Data Structure**

// File handling

**void parseFile(string csvPath)** {

OPEN csvFile

IF csvFile found && csvFile size is not zero {

FOR LOOP all lines in the csvFile {

READ from input next line of csvFile and parse to file

IF line has less than two parameters {

DISPLAY message, the course cannot be added

} END IF

ELSE {

IF line is greater than or equal to two parameters and course pre-

requisite is already in file {

CREATE new course object

LOOP FOR parameters in the line {

ASSIGN Course attributes for each parameter in the

line

} END FOR LOOP

ADD the new line to the Course data structure

} END IF

} END ELSE

} END FOR LOOP

ELSE csvFile size is zero {

DISPLAY error message file could not be opened

} END ELSE

CLOSE csvfile

} END of parseFile function

// Load a CSV file containing courses into a container

// csvPath is the path to the CSV file to load

// function returns a container holding all of the courses read.

**vector<Course> loadCourses(string csvPath)** {

DEFINE a vector data structure to hold a collection of courses

INITIALIZE the CSV Parser using the given path

TRY {

FOR all rows in the csvFile {

IF number of parameters in row is greater than two {

CREATE a new course data structure.

ASSIGN the new course attributes from csv file

PUSHBACK the new course the Courses data structure

} END IF

ELSE number of parameters in row is less than two {

throw error message "course does not have enough parameters to

be added"

} END ELSE

} END FOR LOOP

} END TRY

CATCH (csv::Error& e){

// An error is caught when there are not at least two parameters on a line of

// the csvFile

// An error is caught when any prerequisite at the end of a line does not have

// another line in the csvFile that starts with that course number

DISPLAY any errors encountered reading the csv file

}END CATCH

RETURN the courses data structure

} END of loadCoarses function

// function for searching for a course by course number

**void searchCourse(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

} END FOR LOOP

RETURN course

} END IF

} END FOR LOOP

// This portion of the code is only reached if the course is not found

CREATE a new course object

RETURN empty course

} END of searchCourse function

**Pseudocode for Hash Table Data Structure**

// File handling

**void parseFile(string csvPath)** {

OPEN csvFile

IF csvFile found && csvFile size is not zero {

LOOP FOR all lines in the csvFile {

READ from input next line of csvFile and parse to file

IF line has less than two parameters {

DISPLAY message, the course cannot be added

} END IF

ELSE {

IF line is greater than or equal to two parameters and course pre-

requisite is already in file {

CREATE new course object

LOOP FOR parameters in the line {

ASSIGN Course attributes for each parameter in the

line

} END FOR LOOP

ADD the new line to the Course data structure

} END IF

} END ELSE

} END FOR LOOP

ELSE csvFile size is zero {

DISPLAY error message file could not be opened

} END ELSE

CLOSE csvfile

} END of parseFile function

// Load a CSV file containing courses into a container

// csvPath is the path to the CSV file to load

// function returns a container holding all of the courses read.

**void loadCourses(string csvPath, HashTable\* hashtable)** {

INITIALIZE the CSV Parser using the given path

TRY {

FOR all rows in the csvFile {

IF number of parameters in row is greater than two {

CREATE a new course data structure.

ASSIGN the new course attributes from csv file

INSERT the new course to the Courses data structure (hashTable)

} END IF

ELSE number of parameters in row is less than two {

throw error message "course does not have enough parameters to

be added"

} END ELSE

} END FOR LOOP

} END TRY

CATCH (csv::Error& e){

// An error is caught when there are not at least two parameters on a line of

// the csvFile

// An error is caught when any prerequisite at the end of a line does not have

// another line in the csvFile that starts with that course number

DISPLAY any errors encountered reading the csv file

}END CATCH

} END of loadCoarses function

// function for searching for a course by course number

**void searchCourse(HashTable<Course> courses, String courseNumber)** {

CREATE the key for the given course

ASSIGN node by trying to retrieve the node using the key

// if a searched course is not found, return it

IF no entry exists for the key {

RETURN the course

} END IF

IF the entry is found for the key {

DISPLAY course information

FOR each prerequisite of the course {

CREATE tempKey for the prerequisite

ASSIGN tempNode by trying to retrieve the node using the tempKey

IF the tempNode is found for the tempKey {

DISPLAY course information

} END IF

} END FOR LOOP

} END IF

} END function

**Pseudocode for Binary Search Tree Data Structure**

// File handling

**void parseFile(string csvPath)** {

OPEN csvFile

LOOP FOR all lines in the csvFile {

IF csvFile found && csvFile size is not zero {

READ from input next line of csvFile and parse to file

IF line has less than two parameters {

DISPLAY message, the course cannot be added

} END IF

ELSE {

IF line is greater than or equal to two parameters and course pre-

requisite is already in file {

CREATE new course object

LOOP FOR parameters in the line {

ASSIGN Course attributes for each parameter in the

line

} END FOR LOOP

ADD the new line to the Course data structure

} END IF

} END ELSE

} END FOR LOOP

ELSE csvFile size is zero {

DISPLAY error message file could not be opened

} END ELSE

CLOSE csvfile

} END of parseFile function

// Load a CSV file containing courses into a container

// csvPath is the path to the CSV file to load

// function returns a container holding all of the courses read.

**void loadCourses(string csvPath, BinarySearchTree\* bst)** {

INITIALIZE the CSV Parser using the given path

TRY {

FOR all rows in the csvFile {

IF number of parameters in row is greater than two {

CREATE a new course data structure.

ASSIGN the new course attributes from csv file

INSERT the new course to the Courses data structure (Binary

Search Tree)

} END IF

ELSE number of parameters in row is less than two {

throw error message "course does not have enough parameters to

be added"

} END ELSE

} END FOR LOOP

} END TRY

CATCH (csv::Error& e){

// An error is caught when there are not at least two parameters on a line of

// the csvFile

// An error is caught when any prerequisite at the end of a line does not have

// another line in the csvFile that starts with that course number

DISPLAY any errors encountered reading the csv file

}END CATCH

} END of loadCourses function

// function for searching for a course by course number

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

ASSIGN current node equal to root node

WHILE LOOP current node is not nullptr {

IF current node’s course.courseNumber is the same as the passed-in

courseNumber {

DISPLAY course information

FOR each prerequisite of the course {

DISPLAY course number of the prerequisite

} END FOR LOOP

RETURN current’s course data

} END IF

IF the passed in courseNumber is smaller than the current node’s

course.courseNumber {

// Traverse the left side of the BST

ASSIGN current node to be current’s left node

} END IF

ELSE larger so traverse right side of BST {

ASSIGN current node to be current’s right node

} END ELSE

} END WHILE LOOP

// This portion of code is only reached if the passed in courseNumber is

// not found

// if the courseNumber is not found, create an empty course and return it

DEFINE course (type Course)

RETURN course

} END function

**Pseudocode for a Menu**

**int main(int argc, char\* argv[])** {

DEFINE csvPath (type string) and searchValue (type string)

DEFINE bst (type BinarySearchTree) and INITIALIZE the new binary search tree

DEFINE course (type Course)

DEFINE choice (type int) and INITIALIZE to zero

WHILE LOOP choice not equal to 9 {

DISPLAY “Welcome to the course planner.” and ENDLINE twice

DISPLAY “1. Load Data Structure” and ENDLINE

DISPLAY “2. Print Course List” and ENDLINE

DISPLAY “3. Print Course” and ENDLINE

DISPLAY “9. Exit” and ENDLINE

DISPLAY “ Enter Choice: “ and ENDLINE

INPUT choice

SWITCH (choice) {

case 1:

CALL loadCourses(csvPath, bst)

BREAK

END case 1

case 2:

DISPLAY “Here is a sample schedule:” and ENDLINE twice

CALL PrintCourses for bst

BREAK

END case 2

case 3:

DISPLAY “What course do you want to know about? “

STORE searchValue from user INPUT

STORE course = CALL searchCourse(searchValue) in bst

BREAK

END case 3

default:

DISPLAY choice “ is not a valid option”

BREAK

END default

} END SWITCH

} END WHILE LOOP

DISPLAY “Thank you for using the course planner!”

RETURN 0

} END main function

The following pseudocode is for the printCourses function if the data structure used is a **Vector**:

**void PrintCourses(vector<Course> courses)** function {

FOR all courses in the vector data structure LOOP {

DISPLAY course information

} END FOR LOOP

} END function

The following pseudocode is for the printCourses function if the data structure used is a **hash table**:

**void HashTable::PrintCourses()** function {

// for loop to iterate over all nodes from beginning to end

FOR LOOP from beginning to ending nodes {

ASSIGN searchNode (type node) to the node at the current iteration

IF the searchNode’s key does not equal the max key value {

DISPLAY the searchNode’s course data

ASSIGN searchNode to searchNode’s next node

WHILE LOOP searchNode is not nullptr {

DISPLAY searchNode’s course data

ASSIGN searchNode to searchNode’s next node

} END WHILE LOOP

} END IF

} END FOR LOOP

} END function

The following pseudocode is for the printCourses function if the data structure used is a **binary tree:**

**void BinarySearchTree::PrintCourses()** function {

ASSIGN this node as the root node and CALL printCourses and PASS the root node

} END function

**void BinarySearchTree::printCourses(Node\* node)** function {

IF the current node is not equal to nullptr {

CALL printCourses and pass node’s left node

DISPLAY node’s course information

CALL printCourses and pass node’s right node

} END IF

} END function

**Runtime Analysis**

**Runtime analysis for Vector Data Structure**

**void parseFile(string csvPath) function:**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **OPEN csvFile** | 1 | 1 | 1 |
| **IF csvFile found && csvFile size is not zero** | 1 | 1 | 1 |
| **FOR LOOP all lines in the**  **csvFile** | 1 | n | n |
| **READ from input next**  **line of csvFile and**  **parse to file** | 1 | n | n |
| **IF line has less than**  **two parameters** | 1 | n | n |
| **DISPLAY message,**  **course cannot be**  **added** | 1 | n | n |
| **ELSE IF line is**  **greater than or equal**  **to two parameters and**  **course prerequisite is**  **already in file** | 1 | n | n |
| **CREATE new course**  **object** | 1 | n | n |
| **LOOP FOR**  **parameters in the**  **line** | 1 | n^2 | n^2 |
| **ASSIGN Course**  **attributes for**  **each parameter**  **in the line** | n | n | n |
| **ADD the new line**  **to the Course data**  **structure** | 1 | n | n |
| **ELSE csvFile size is zero** | 1 | 0 | 1 |
| **DISPLAY error message file**  **could not be opened** | 1 | 0 | 1 |
| **CLOSE csvFile** | 1 | 1 | 1 |
| **Total Cost** | | | n^2+8n+5 |
| **Runtime** | | | O(n^2) |

**vector<Course> loadCoarses(string csvpath) function**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| **DEFINE a vector data structure to hold a collection of courses** | 1 | 1 | 1 |
| **INITIALIZE the CSV Parser using the given path** | 1 | 1 | 1 |
| **TRY {** | 1 | 1 | 1 |
| **FOR all rows in the**  **csvFile** | 1 | n | n |
| **IF number of**  **parameters in row is**  **greater than two** | 1 | n | n |
| **CREATE a new**  **course data**  **structure** | 1 | n | n |
| **ASSIGN the new**  **course attributes**  **from the csv file** | 1 | n | n |
| **PUSHBACK the new**  **course to the**  **Courses data**  **structure** | 1 | n | n |
| **ELSE number of**  **parameters in row is**  **less than two** | 1 | n | n |
| **THROW error**  **message** | 1 | n | n |
| **CATCH (csv::Error& e)** | 1 | n | n |
| **DISPLAY any errors**  **encountered** | 1 | n | n |
| **RETURN the course data structure** | 1 | 1 | 1 |
| **Total Cost** | | | 9n + 4 |
| **Runtime** | | | O(n) |

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

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| **FOR all courses** | 1 | n | n |
| **IF the course is the same**  **as courseNumber** | 1 | n | n |
| **PRINT out the course**  **information** | 1 | n | n |
| **FOR each**  **prerequisite of**  **the course** | 1 | n^2 | n^2 |
| **PRINT the**  **prerequisite**  **course information** | 1 | n | n |
| **RETURN course** | 1 | n | n |
| **CREATE a new course object** | 1 | 0 | 1 |
| **RETURN empty course** | 1 | 0 | 1 |
| **Total Cost** | | | n^2+5n+2 |
| **Runtime** | | | O(n^2) |

**Runtime analysis for Hash Table Data Structure**

**void parseFile(string csvPath)**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **OPEN csvFile** | 1 | 1 | 1 |
| **IF csvFile found && csvFile size is not zero** | 1 | 1 | 1 |
| **FOR LOOP all lines in the**  **csvFile** | 1 | n | n |
| **READ from input next**  **line of csvFile and**  **parse to file** | 1 | n | n |
| **IF line has less than**  **two parameters** | 1 | n | n |
| **DISPLAY message,**  **course cannot be**  **added** | 1 | n | n |
| **ELSE IF line is**  **greater than or equal**  **to two parameters and**  **course prerequisite is**  **already in file** | 1 | n | n |
| **CREATE new course**  **object** | 1 | n | n |
| **LOOP FOR**  **parameters in the**  **line** | 1 | n^2 | n^2 |
| **ASSIGN Course**  **attributes for**  **each parameter**  **in the line** | n | n | n |
| **ADD the new line**  **to the Course data**  **structure** | 1 | n | n |
| **ELSE csvFile size is zero** | 1 | 0 | 1 |
| **DISPLAY error message file**  **could not be opened** | 1 | 0 | 1 |
| **CLOSE csvFile** | 1 | 1 | 1 |
| **Total Cost** | | | n^2+8n+5 |
| **Runtime** | | | O(n^2) |

**void loadCoarses(string csvPath, HashTable\* hashTable**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| **INITIALIZE the CSV Parser using the given path** | 1 | 1 | 1 |
| **TRY {** | 1 | 1 | 1 |
| **FOR all rows in the**  **csvFile** | 1 | n | n |
| **IF number of**  **parameters in row is**  **greater than two** | 1 | n | n |
| **CREATE a new**  **course data**  **structure** | 1 | n | n |
| **ASSIGN the new**  **course attributes**  **from the csv file** | 1 | n | n |
| **INSERT the new**  **course to the**  **Courses data**  **Structure**  **(hashTable)** | 1 | n | n |
| **ELSE number of**  **parameters in row is**  **less than two** | 1 | n | n |
| **THROW error**  **message** | 1 | n | n |
| **CATCH (csv::Error& e)** | 1 | n | n |
| **DISPLAY any errors**  **encountered** | 1 | n | n |
| **Total Cost** | | | 9n + 2 |
| **Runtime** | | | O(n) |

**void searchCourse(HashTable<Course>courses, String courseNumber)**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| **CREATE the key for the given course** | 1 | 1 | 1 |
| **ASSIGN node by trying to retrieve the node using the key** | 1 | 1 | 1 |
| **IF no entry exists for the key** | 1 | 1 | 1 |
| **RETURN the course** | 1 | 0 | 1 |
| **IF the entry is found for the key** | 1 | 1 | 1 |
| **DISPLAY course information** | 1 | 1 | 1 |
| **FOR each prerequisite of**  **the course** | 1 | n | n |
| **CREATE tempKey for the**  **prerequisite** | 1 | n | n |
| **ASSIGN tempNode by**  **trying to retrieve the**  **node using the tempKey** | 1 | n | n |
| **IF the tempNode is**  **found for the tempKey** | 1 | n | n |
| **DISPLAY course**  **information** | 1 | n | n |
| **Total Cost** | | | 5n+6 |
| **Runtime** | | | O(n) |

**Runtime Analysis for Binary Tree Data Structure**

**void parseFile(string csvpath)**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **OPEN csvFile** | 1 | 1 | 1 |
| **IF csvFile found && csvFile size is not zero** | 1 | 1 | 1 |
| **FOR LOOP all lines in the**  **csvFile** | 1 | n | n |
| **READ from input next**  **line of csvFile and**  **parse to file** | 1 | n | n |
| **IF line has less than**  **two parameters** | 1 | n | n |
| **DISPLAY message,**  **course cannot be**  **added** | 1 | n | n |
| **ELSE IF line is**  **greater than or equal**  **to two parameters and**  **course prerequisite is**  **already in file** | 1 | n | n |
| **CREATE new course**  **object** | 1 | n | n |
| **LOOP FOR**  **parameters in the**  **line** | 1 | n^2 | n^2 |
| **ASSIGN Course**  **attributes for**  **each parameter**  **in the line** | n | n | n |
| **ADD the new line**  **to the Course data**  **structure** | 1 | n | n |
| **ELSE csvFile size is zero** | 1 | 0 | 1 |
| **DISPLAY error message file**  **could not be opened** | 1 | 0 | 1 |
| **CLOSE csvFile** | 1 | 1 | 1 |
| **Total Cost** | | | n^2+8n+5 |
| **Runtime** | | | O(n^2) |

**void loadCourses(string csvpath, BinarySearchTree\* bst)**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| **INITIALIZE the CSV Parser using the given path** | 1 | 1 | 1 |
| **TRY {** | 1 | 1 | 1 |
| **FOR all rows in the**  **csvFile** | 1 | n | n |
| **IF number of**  **parameters in row is**  **greater than two** | 1 | n | n |
| **CREATE a new**  **course data**  **structure** | 1 | n | n |
| **ASSIGN the new**  **course attributes**  **from the csv file** | 1 | n | n |
| **INSERT the new**  **course to the**  **Courses data**  **Structure**  **(Binary Search**  **tree)** | 1 | n | n |
| **ELSE number of**  **parameters in row is**  **less than two** | 1 | n | n |
| **THROW error**  **message** | 1 | n | n |
| **CATCH (csv::Error& e)** | 1 | n | n |
| **DISPLAY any errors**  **encountered** | 1 | n | n |
| **Total Cost** | | | 9n + 2 |
| **Runtime** | | | O(n) |

**void searchCourse(Tree<Course>course, String courseNumber)**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| **ASSIGN current node equal to root node** | 1 | 1 | 1 |
| **WHILE current node is not nullptr** | 1 |  |  |
| **IF current node’s**  **course.courseNumber is the**  **same as the passed-in**  **courseNumber** | 1 | n/2 | n/2 |
| **DISPLAY course**  **information** | 1 | n | n |
| **FOR each prerequisite**  **of the course** | 1 | n | n |
| **DISPLAY course**  **number of the**  **prerequisite** | 1 | n | n |
| **RETURN current’s**  **course data** | 1 | n | n |
| **IF the passed in**  **courseNumber is smaller**  **than the current node’s**  **course.courseNumber** | 1 | n/2 | n/2 |
| **ASSIGN current node to**  **be current’s left node** | 1 | n | n |
| **ELSE larger so traverse**  **right side of BST** | 1 | n | n |
| **ASSIGN current node to**  **be current’s right**  **node** | 1 | n | n |
| **DEFINE course (type course)** | 1 | 0 | 1 |
| **RETURN course** | 1 | 0 | 1 |
| **Total Cost** | | |  |
| **Runtime** | | |  |

**Advantages and Disadvantages of Vector, Hash Table, and Binary Tree Data Structures**

Advantages of vector data structures include dynamic sizing (vectors have the ability to shrink or grow dynamically), vectors support random access which helps with fast retrieval or manipulation of data, they are memory efficient, and they are supported by standard libraries (Usman, 2024). Disadvantages of vector data structures include dynamic resizing overhead (insertion and deletion operations beyond the memory-allocated capacity could incur overhead because of copying elements and memory reallocation), inefficient insertion and deletion operations unless occurring at the end of the data structure and using vectors can lead to memory fragmentation (Usman, 2024).

The advantages of using a hash table data structure are that operations such as inserting, deleting, and searching are very fast and can be done in O(1) time, which is constant time, and hash tables can store large amounts of data (Baeldung, 2023). Disadvantages include inefficiency due to collisions, which are often unavoidable when there is a large set of possible keys, hash data structures do not allow null values, they have limited capacity and can fill up, they are often complex in their implementation, and retrieving elements in a specific order can be difficult because hash tables do not maintain the order of elements (GeeksforGeeks, 2023).

Advantages of binary tree data structures include efficient searching for elements because search operations can be performed in O(Log n) time complexity, binary trees can be traversed in specific orders, such as in-order, pre-order, and post-order, binary trees are memory efficient, have fast insertion and deletion operations, and are easy to implement (GeeksforGeeks, 2023). The disadvantages of binary tree data structures are that they have a limited structure only allowing two child nodes per node, which in certain applications can limit their usefulness, if the tree is unbalanced it could lead to inefficient search operations, particularly when items are added to the tree in a random order, they can be space inefficient compared to other data structures, and the algorithms used to keep the structure balanced are complex (GeeksforGeeks, 2023).

**Recommendation**

When considering what data structure should be implemented for the academic advisors in the Computer Science department at ABCU, worst-case time complexity should be taken into consideration. The best algorithm is one that has a constant time complexity. However, the algorithms we are considering do not have a constant time complexity. We see that in the vector data structure, our time complexity when searching for courses is O(n^2) because the algorithm for this data structure has a nested loop. O(n^2) is called a quadratic time complexity because the running time of the algorithm grows as a square function of input size, which when considering large inputs, or sets of data, this can cause the algorithm to run for longer periods of time (GeeksforGeeks, 2024). When we look at the hash table data structure, we see we have a runtime complexity of O(n), which is better than the runtime complexity for the vector data structure, however, the time complexity is still dependent on how large the data or input size is and we should consider further our last data structure. When looking at the binary tree data structure, we see we have a time complexity of O. This is because each time the search is conducted half of the nodes it needs to search through are cut in half until the node is found. In the worst-case scenario, the binary tree data structure has the best time complexity and is, therefore, the data structure that would be best suited for the academic advisors in the Computer Science department at ABCU.

**References**

Baeldung. (2023, March 11). *Binary Trees vs Linked Lists vs Hash*

*Tables*. <https://www.baeldung.com/cs/binary-trees-vs-linked-lists-vs-hash-tables>

GeeksforGeeks. (2023, March 28). *Applications, Advantages and Disadvantages of Hash Data*

*Structure*. <https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-hash-data-structure/>

GeeksforGeeks. (2023, March 29). *Applications, Advantages and Disadvantages of Binary*

*Tree*. <https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-binary-tree/>

GeeksforGeeks. (2024, April 2). *What Does Big O(N^2) Complexity*

*Mean?* <https://www.geeksforgeeks.org/what-does-big-on2-complexity-mean/>

Usman, M. (2024, February 21). *Vector Data*

*Structure*. CodeLikeChamp. <https://codelikechamp.com/vector-data-structure/>