**CS300 Project One**

**Pseudocode:**

fileRead(file)

# Open file

OPEN file

SET courseList = []

SET prereqList = []

# For vector

SET VECTOR courses = []

# For hash table

INITIALIZE HASH TABLE courses

# For binary tree

INITIALIZE BINARY TREE courses

# Loop through lines

WHILE NOT endFile {

# Read line and split it into list

SET line = readLine(file)

SET list = split(line, ‘,’)

# Check for format error

IF list.length < 2 {

THROW “Error, invalid format”

}

# Set info to the variables

SET courseNumber = list[0]

SET name = list[1]

# Make a seperate list for the N amount of prereqs

SET prerequisite = []

IF length of list > 2 {

FOR i from 2 to length of list – 1{

APPEND list[i] into prereq

}

}

# Make lists for course number and prereqs

APPEND courseNumber into courseList

FOR prereq IN prerequisite {

APPEND prereq into prereqList

}

# Create object, add data to it, then populate the structure

SET course OBJECT

SET course.courseNumber = courseNumber

SET course.name = name

SET course.prerequisite = prerequisite

# For vector

INSERT course INTO courses

# For hash table and binary tree

INSERT course INTO courses WITH KEY courseNumber

}

# Check to make sure prereq ins in courses

FOR prereq IN prereqList {

CHECK prereq IN courseList

IF NOT in courseList {

THROW “Prerequisite class not in course list”

}

}

}

# Looks for course with number, if found in object, return the objects information as print.

DEFINE findCourse(courseNumber) {

FOR course IN courses {

IF course.courseNumber = courseNumber {

PRINT course.courseNumber

PRINT course.name

FOR each prereq IN course.prerequisite {

PRINT prereq

}

}

}

#Sorts alphanumerically and prints

sortPrintList(list) {

# For vector

SORT courses BY course.courseNumber

FOR each course IN courses {

PRINT course.courseNumber

}

# For hash table

SET keys = SORT key(courses)

FOR key in keys {

SET course = courses[key]

PRINT course.courseNumber

}

# For binary tree

IF node IN course IS NOT null {

SortPrintList(node->left)

PRINT course.courseNumber

SortPrintList(node->right)

}

}

# Main function to put everything together

Main() {

PRINT “””

Option 1: Load data into structure

Option 2: Sort and print courses

Option 3: Find a course with a course number

Option 9: Close program

“””

SET input = 0

GET file

WHILE input IS NOT 9 {

GET NUMBER input

IF input = 1 {

fileRead(file)

}

ELSE IF input = 2 {

SortPrintList(courses)

}

ELSE IF input = 3 {

GET course

findCourse(course)

}

}

EXIT()

}

**Run time:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **Vector** | **Hash Table** | **Binary Tree** |
| Opening file | O(1) | O(1) | O(1) |
| Setting variables | O(1) | O(1) | O(1) |
| Initializing structure | O(1) | O(1) | O(1) |
| Reading/processing lines | O(n) | O(n) | O(n) |
| Inserting object into structure | O(n) | O(1) | O(log(n)) |

**Advantages/Disadvantages:**

**Vectors:**

With a vector, initializing is very simple and there’s no concern about making sure nodes are established. On top of that, doing operations such as inserting, deleting, finding, and sorting are simple because many of these operations are built in already. Furthermore, a vector can be used for many different purposes.

Some disadvantages would be that when inserting, the operation is always a linear time complexity, and there’s no way to speed that up. Furthermore, searching in a vector is a linear time complexity, which can be costly if you’re looking for data. Overall, although a vector is simple to use, it’s still inefficient.

**Hash Table:**

A hash table is one of the most efficient data structures for operations such as inserting, finding, and deleting. For these operations, it has a constant time complexity, however with its worst case, it could be linear time which is still not detrimental.

Some disadvantages would be that if collisions occur, the time complexity can worsen, and the program can take up extra memory (Geeks). Furthermore, a hash table does not maintain order.

**Binary Tree:**

A binary tree is flexible, which means that it can hold data as needed without costing too much (Geeks). In addition to that, binary trees remain organized and ordered.

Some disadvantages include needing additional memory for the pointers and making sure the tree is balanced. Furthermore, when doing operations, binary trees have a logarithmic time complexity and for multiple operations, this will become inefficient.

**Recommendation:**

For my code, I plan to use a hash table for many different reasons. A hash table is more efficient than a vector or binary tree because most of its operations are of constant time. Additionally, the order of my data is not a key factor, so I do not need a data structure that is easily sorted. Finally, the worst-case scenario in terms of complexity for a hash table is linear time, which still matches the complexity of a vector. Overall, a hash table would be the best choice for this scenario.

**References**

GeeksforGeeks. (2023a, March 28). *Applications, advantages and disadvantages of hash data structure*. <https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-hash-data-structure/>

GeeksforGeeks. (2023b, March 29). *Applications, advantages and disadvantages of Binary Tree*. <https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-binary-tree/>