**CS 300 Project One Pseudocode**

**Brandon Cook**

**August 11, 2024**

# Vector Pseudocode

*Create Course class*

Declare private variables:

String courseID

String courseName

Vector of type String coursePrereqs

Declare public methods:

Constructor Course()

setCourseID(courseID)

setCourseName(courseName)

setPrerequisite(prereqs)

getCourseID()

getCourseName()

getPrerequisite()

*End Course class*

*Create main() class*

Initialize empty vector list of type Courses named courseList

Open data file to be read

*If file cannot be read*

Print error message

*Else*

*While loop to iterate over each line of file*

Read line from file

Split line at each comma using delimiter

*If number of segments on each line is greater than or equal to 2*

Initialize new Course object

Extract courseID from first segment

Extract courseName from second segment

*If there are still segments remaining*

Extract prerequisites, adding them to prerequisite vector list

*End if*

Add Course object to courseList

*End if*

*End while*

Close file

*End if*

*End main()*

*Create method printCourseInfo(courseID)*

Initialize Boolean variable courseFound, set to false

*For loop to iterate over each item in courseList*

*If item.getCourseID() is equal to courseID*

Print courseID

Print courseName

*If item.getPrerequisites() is not null*

*For loop to iterate over number of items in prerequisites*

Print prerequisite

*End for*

*End if*

Set courseFound to true

Break/exit loop

*End if*

*End for*

*If courseFound is false*

Print “Course does not exist.”

*End if*

*End printCourseInfo()*

# Hash Table Pseudocode

*Create Course class*

Declare private variables:

String courseID

String courseName

Vector of type String coursePrereqs

Declare public methods:

Constructor Course()

setCourseID(courseID)

setCourseName(courseName)

setPrerequisite(prereqs)

getCourseID()

getCourseName()

getPrerequisite()

*End Course class*

*Create main() class*

Initialize empty hash table named courseTable with key type ‘String,’ data type ‘Course’

Open data file to be read

*If file cannot be read*

Print error message

*Else*

*While loop to iterate over each line of file*

Read line from file

Split line at each comma using delimiter

*If number of segments on each line is greater than or equal to 2*

Initialize new Course object

Extract courseID from first segment

Extract courseName from second segment

*If there are still segments remaining*

Extract prerequisites, adding them to prerequisite vector list

*End if*

Set Course object’s courseID, courseName, and prerequisites

Add Course object to courseTable with key courseID

*End if*

*End while*

Close file

*End if*

*End main()*

*Create method printCourseInfo()*

*For each key-value pair courseID, Course in courseTable*

Print courseID

Print courseName

*If item.getPrerequisites() is not null*

*For loop to iterate over number of items in prerequisites*

Print prerequisite

*End for*

*End if*

*End for*

*End printCourseInfo()*

# Binary Search Tree Pseudocode

*Create struct Course()*

courseID

courseName

prerequisites

Constructor Course()

*End Course*

*Create struct Node()*

*Course*

*Right pointer*

*Left pointer*

*Create class BST()*

*Declare public methods:*

*BST()*

*printCourseInfo()*

*End class BST()*

*Create main() class*

Initialize empty BST named courseTree

Open data file to be read

*If file cannot be read*

Print error message

*Else*

*While loop to iterate over each line of file*

Read line from file

Split line at each comma using delimiter

*If number of segments on each line is greater than or equal to 2*

Initialize new Course object

Extract courseID from first segment

Extract courseName from second segment

*If there are still segments remaining*

Extract prerequisites, adding them to prerequisite vector list

*End if*

Create node and set Course object’s courseID, courseName, and prerequisites

Add node to tree

*End if*

*End while*

Close file

*End if*

*End main()*

*Create method printCourseInfo()*

*For each key-value pair courseID, Course in courseTree*

Print courseID

Print courseName

*If item.getPrerequisites() is not null*

*For loop to iterate over number of items in prerequisites*

Print prerequisite

*End for*

*End if*

*End for*

*End printCourseInfo()*

# Menu Pseudocode

Create variable choice of type integer, set to 0

While choice is not 9

Display menu options to user

Get input choice from user

If 1

Call method to load file into data structure

If 2

Call method to print alphanumeric ordered list of courses

If 3

Take input courseTitle from user

Call method to search and print courseTitle along with prerequisites

Else If not 1, 2, 3, or 9

Display invalid choice option to user

End If

End While, exit program

# Vector Alphanumeric Pseudocode

Create method sortVector()

Iterate from i=0 over length of vector minus 1

Iterate from j=0 over length of vector minus i minus 2

If vector at index j is greater than index j+1

Swap index j and index j+1

End If

End loop

End loop

End method sortVector()

Create method printVector()

Iterate over vector

Print course number, title, and prerequisites

End loop

End method printVector()

# Hash Table Alphanumeric Pseudocode

Create method sortHash(courseList)

Iterate over i=0 to length of list minus 1

Iterate over j=0 to length of list minus i minus 2

If list at index j is greater than index j+1

Swap index j and index j+1

End if

End loop

End loop

End method sortHash

Create method printHash()

Create list courseList

For each key, value pair in hashTable

Create course with courseNumber key, courseTitle and prerequisite values

Add course to courseList

End for

Call sortHash(courseList)

Iterate over length of list

Print sorted list course number, title, and prerequisites

End loop

End method printHash()

# Binary Search Tree Alphanumeric Pseudocode

Create method inOrderTraversal()

If current node is not null

Call inOrderTraversal(node.left)

Print course information, course number, course title, prerequisites

Call inOrderTraversal(node.right)

End if

End method inOrderTraversal

# Evaluation

|  |  |  |  |
| --- | --- | --- | --- |
|  | Insertion | Search | Memory |
| Vector | O(n) | O(n) | O(n) |
| Hash Table | O(n) | O(n) | O(n) |
| Binary Search Tree | O(log n) | O(log n) | O(n) |

All three data structures are consistent when it comes to memory usage. When sorting a hash table, it proves difficult due to the way they are formed using key-value pairs. When ordering these, the way I did it was transferring each pair into a vector and then sorting the vector, which adds to the runtime. Binary search trees use fewer operations when sorting due to how it maintains a sorted order initially. An in-order traversal can be used to quickly print an alphanumeric list. With this in mind, a binary search tree would be the best solution for this problem and is what I will be using in my code.