**Project One**

**Pseudocode and Analysis**

Brittany Elderts-Ajala

Southern New Hampshire University

CS-300 DSA: Analysis and Design

Professor Dewin Andujar

08/07/2022

**Vector Data Structure Pseudocode**

Create vector for course objects

Read File Function –

Open file for input using ifstream

Read data from file

For each line in the file

Initialize parameterCount to 0

For each ‘,’

++parameterCount

If parameterCount >= 1

Create new Course\* newCourse

Split course info by ‘,’

courseNumber = courseInfo[0]

courseTitle = courseInfo[1]

Else

Display error message – “Missing course number or title.”

If parameterCount > 2

coursePrerequisite1 = courseInfo[2]

If coursePrerequisite1 != any other lineData[0]

Output Error Message

If parameterCount > 3

coursePrerequisite2 = courseInfo[3]

If coursePrerequisite2 != any other lineData[0]

Output Error Message

Search vector for specific course Function –

User input userCourseNumber

Start search at head node of course vector

If userCourseNumber == currentNode.courseNumber

Print course information

Print prerequisites

Else

While userCourseNumber != currentNode.courseNumber

If userCourseNumber == nextNode.courseNumber

Print nextNode.course information

Print nextNode.prerequisites

Else

currentNode = next node

**Vector Data Structure Runtime Analysis**

An advantage to using a vector data structure is that it’s easy to add and remove data from the list. A disadvantage to using a vector data structure is that searching for a specific item in the structure requires looking at every item in the vector.

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **Times Executes** | **Total Cost** |
| **Compare user course number and current node course number** | **1** | **N** | **N** |
| **Compare user course number and current node course number** | **1** | **N** | **N** |
| **Compare user course number and next node course number** | **1** | **N** | **N** |
| **currentNode = nextNode** | **1** | **N** | **N** |
| **Compare user course number and current node course number** | **1** | **N** | **N** |
| **Compare user course number and next node course number** | **1** | **N** | **N** |
| **Output course information** | **1** | **N** | **N** |
| **Total Cost** | | | **7N** |
| **Runtime** | | | **O(N)** |

**Hash Table Data Structure Pseudocode**

Define Course parameters

Create hash table for course objects

Read File Function –

Open file for input using ifstream

Read data from file

For each line in the file

Initialize parameterCount to 0

For each ‘,’

++parameterCount

If parameterCount >= 1

Create new Course\* newCourse

Split course info by ‘,’

courseNumber = courseInfo[0]

courseTitle = courseInfo[1]

Else

Display error message – “Missing course number or title.”

If parameterCount > 2

coursePrerequisite1 = courseInfo[2]

If coursePrerequisite1 != any other lineData[0]

Output Error Message

If parameterCount > 3

coursePrerequisite2 = courseInfo[3]

If coursePrerequisite2 != any other lineData[0]

Output Error Message

Search hash table for specific course Function –

User input userCourseNumber

Start search at head node of hash table

If userCourseNumber == currentNode.courseNumber

Print course information

Print prerequisites

Else

While userCourseNumber != currentNode.courseNumber

If userCourseNumber == nextNode.courseNumber

Print nextNode.course information

Print nextNode.prerequisites

Else

currentNode = next node

**Hash Table Data Structure Runtime Analysis**

An advantage to using a hash table is there are different buckets which data can be stored in. These buckets provide a general idea of where a piece of data can be found, but data can be moved to different buckets that do not follow the standard pattern if the bucket is already occupied. This again means every item must be compared until a match is found.

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **Times Executes** | **Total Cost** |
| **Compare user course number and current node course number** | **1** | **N** | **N** |
| **Compare user course number and current node course number** | **1** | **N** | **N** |
| **Compare user course number and next node course number** | **1** | **N** | **N** |
| **currentNode = nextNode** | **1** | **N** | **N** |
| **Compare user course number and current node course number** | **1** | **N** | **N** |
| **Compare user course number and next node course number** | **1** | **N** | **N** |
| **Output course information** | **1** | **N** | **N** |
| **Total Cost** | | | **7N** |
| **Runtime** | | | **O(N)** |

**Binary Search Tree Data Structure Pseudocode**

Define Course parameters – courseNumber, courseTitle, coursePrerequisite1, coursePrerequisite2

Create Binary Search Tree for course objects

Read File Function –

Open file for input using ifstream

Read data from file

For each line in the file

Initialize parameterCount to 0

For each ‘,’

++parameterCount

If parameterCount >= 1

Create new Course\* newCourse

Split course info by ‘,’

courseNumber = courseInfo[0]

courseTitle = courseInfo[1]

Else

Display error message – “Missing course number or title.”

If parameterCount > 2

coursePrerequisite1 = courseInfo[2]

If coursePrerequisite1 != any other lineData[0]

Output Error Message

If parameterCount > 3

coursePrerequisite2 = courseInfo[3]

If coursePrerequisite2 != any other lineData[0]

Output Error Message

Search Binary Search Tree for specific course Function –

User input userCourseNumber

Start search at head node of Binary Search Tree

If userCourseNumber == currentNode.courseNumber

Print course information

Print prerequisites

Else

While currNode != null

Traverse down the left of Binary Search Tree until a match is found

Print Node.course information

Print node.prerequisites

If no match is found, traverse down the right of Binary Search Tree until a match is found.

Print Node.course information

Print node.prerequisites

Else

Output “No matching courses”

**Binary Search Tree Data Structure Runtime Analysis**

An advantage to using a binary search tree is that when using an organized binary search tree, only half of the tree must be compared based on the way data is sorted upon input. A disadvantage to using a binary search tree is that it is more complicated to code, but I believe it is the most efficient data structure in this circumstance.

**\*I would recommend using a Binary Search Tree for ABCU project.\***

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **Times Executes** | **Total Cost** |
| **userCourseNumber = user input** | **1** | **1** | **1** |
| **currNode = head** | **1** | **1** | **1** |
| **Compare userCourseNumber and currNode.courseNumber** | **1** | **N** | **N** |
| **Check currNode != null** | **1** | **N** | **N** |
| **Traverse down left side of tree comparing courseNumber for a match** | **a** | **N** | **a \* N** |
| **Traverse down right side of tree comparing courseNumber for a match** | **b** | **N** | **b \* n** |
| **Output “No matching courses”** | **1** | **1** | **1** |
| **Total Cost** | | | **1+1+N+N+**  **a\*N+b\*N+1** |
| **Runtime** | | | **O(N)** |

**Menu Pseudocode**

Output menu options – “Load Data Structure”, “Print Course List”, “Print Course”, and “Exit”

If user selects “Load Data Structure”

Define Course parameters – courseNumber, courseTitle, coursePrerequisite1, coursePrerequisite2

Create data structure for course objects

Open file for input using ifstream

Read data from file

For each line in the file

Initialize parameterCount to 0

For each ‘,’

++parameterCount

If parameterCount >= 1

Create new Course\* newCourse

Split course info by ‘,’

courseNumber = courseInfo[0]

courseTitle = courseInfo[1]

Else

Display error message – “Missing course number or title.”

If parameterCount > 2

coursePrerequisite1 = courseInfo[2]

If coursePrerequisite1 != any other lineData[0]

Output Error Message

If parameterCount > 3

coursePrerequisite2 = courseInfo[3]

If coursePrerequisite2 != any other lineData[0]

Output Error Message

If user selects “Print Course List”

Create data structure to store courses in alphanumeric order

Traverse data structure created during “Load Data Structure”

While currNodeLoaded != null

Remove letters from currNodeLoaded.courseNumber

If currNodeLoaded.courseNumber == 100

currNodeLoaded = head node of organized data structure

Else If prevNodeOrganized.courseNumber > currNodeLoaded.courseNumber

prevNodeOrganized = nextNodeOrganized

currNodeLoaded = prevNodeOrganized

Else

currNodeLoaded = nextNodeOrganized

currNodeLoaded = nextNodeLoaded

Traverse organized structure to print course information

While currNodeOrganized != null

Output currNodeOrganized.courseNumber, currNodeOrganized.courseTitle, and currNodeOrganized.coursePrerequisites if applicable

currNodeOrganized = nextNodeOrganized

If user selects “Print Course”

Prompt user to input course number

Traverse data structure until matching course number is found

If userCourseNumber == currentNode.courseNumber

Print course information

Print prerequisites

Else

While userCourseNumber != currentNode.courseNumber

If userCourseNumber == nextNode.courseNumber

Print nextNode.course information

Print nextNode.prerequisites

Else

currentNode = next node

If user selects “Exit”

Exit the progrom