**CS 300 Project One: Vector Data Structure Pseudocode**

// 1a. File validation pseudocode

bool ValidateFile(string filePath) {

Open file using filePath

Create a list of courseNumbers

For each Row in file:

If Row has no element or less than 2 elements

Display error

Return false

For each Row in file:

Add Row-Element-1 to courseNumbers

If Row has more than 2 Elements:

For each Row-Element from Row-Element-3 to last Row-Element:

If Row-Element is included in CourseNumbers list

continue

Else return false

}

//1b. Vector load pseudocode

vector<Course> LoadCourses(string filePath) {

Display fileLoadingMessage

**ValidateFile** using filePath

If **ValidateFile** is successful or returns true:

Create a vector of Courses

For each Row in File:

Create a new Course

Store Row-Element-1 as courseNumber

Store Row-Element-2 as courseName

If Row has more than 2 Elements:

For each Row-Element from Row-Element-3 to last Row-Element:

If Row-Element is included in CourseNumbers list

Store Row-Element as coursePrerequisite

Else Store coursePrerequisite as none

Return Courses

}

//1b. Hash Table load pseudocode

HashTable<Course> LoadCourses(string filePath) {

Display fileLoadingMessage

**ValidateFile** using filePath

If **ValidateFile** is successful or returns true:

Create a new hash Table with a predetermined number of nodes or table-size

For each Row in File:

Create a new Course

Store Row-Element-1 as courseNumber

Store Row-Element-2 as courseName

If Row has more than 2 Elements:

For each Row-Element from Row-Element-3 to last Row-Element:

If Row-Element is included in CourseNumbers list:

Store Row-Element as coursePrerequisite

Else Store coursePrerequisite as none

Pick a position on the hash-table using a hash function

If there is no node or an unused node at the picked position:

Create a new node at the position

Store the created Course inside the new node

Else if there is one or more used nodes at picked position:

Create a new node at the position

Store the created Course inside the new node

Add the created node after the last node at the position

Return HashTable

}

//1b. Tree Load Pseudocode

Tree<Course> LoadCourses(string filePath) {

Display fileLoadingMessage

**ValidateFile** using filePath

If **ValidateFile** is successful or returns true:

Create a new Tree

For each Row in File:

Create a new Course

Store Row-Element-1 as courseNumber

Store Row-Element-2 as courseName

If Row has more than 2 Elements:

For each Row-Element from Row-Element-3 to last Row-Element:

If Row-Element is included in CourseNumbers list:

Store Row-Element as coursePrerequisite

Else Store coursePrerequisite as none

Create new tree node containing the created Course and add it to tree

Return Tree

}

//1c. Print course information and prerequisites

void PrintCourseInformation(<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

}

//2. Menu pseudocode

Display menu options (1- load data, 2- print course list, 3- print course, q- quit)

Case option 1:

**LoadCourses**(string filePath)

Case option 2:

**LoadCourses**(string filePath)

**PrintCourseInformation**(<Course> courses, String courseNumber)

Case option 3:

**LoadCourses**(string filePath)

**PrintCourseList**(<Course> courses)

//3. Print course list pseudocode

void PrintCourseList((<Course> courses) {

Sort courses by courseName

For each Course in the sorted courses

Print the courseNumber and courseName for the Course

}

|  |  |  |  |
| --- | --- | --- | --- |
|  | Vector | Hash Table | Tree |
| File Validate | 9n + 5 | 9n + 5 | 9n + 5 |
| Data load Cost | 8n + 5 | 17n + 4 | 10n + 5 |
| Print Course Cost | 4n + 1 | 4n + 1 | 4n + 1 |
| Total Line Cost | 21n + 11 | 30n + 10 | 23n + 11 |
| Run time worst case | On | On | On |

Recommendation:

The line cost analysis indicates that the vector has the least run time cost but cased on the run time worst cases which are the same, all data structures may yield the highest possible algorithmic efficiency depending on processing speed.