**Caleb Jones**

**CS 300**

**6-2 Project One**

# Vector Data Structure Pseudocode:

The following pseudocode define:

* Design pseudocode to define how the program opens the file, reads the data from the file, parses each line, and checks for file format
* Design pseudocode to show how to create course objects and store them in the appropriate data structure
* Design pseudocode that will search the data structure for a specific course and print out course information and prerequisites

Design pseudocode to define how the program opens the file, reads the data from the file, parses each line, and checks for file format:

OPEN file

WHILE reading each line until end of file

ADD line to fileVector

FOR all lines in fileVector // to remove any incomplete data

SPLIT into a temporary splitVector BY commas

IF splitVector size < 2 THEN

REMOVE line from fileVector

CONTINUE

ELSE

ADD course from splitVector at 0 to a courseList

vector

IF splitVector size > 2 THEN

FOR each course after splitVector at 2

IF course is NOT in courseList

REMOVE line from fileVector

CONTINUE

Design pseudocode to show how to create course objects and store them in the appropriate data structure:

DEFINE Course Class as Object

CourseNumber as string

CourseName as string

CoursePrerequisites as vector of strings

FOR each line in fileVector

CREATE new CourseObject

SPLIT into a temporary splitVector BY commas

IF splitVector size is 2 THEN // has no prerequisite info

SET CourseObject.CourseNumber as splitVector at 0

SET CourseObject.CourseName as splitVector at 1

SET CourseObject.Prerequisites as NULL

ELSE

SET CourseObject.CourseNumber as splitVector at 0

SET CourseObject.CourseName as splitVector at 1

FOR remaining length of splitVector

ADD item of split vector into

CourseObject.Prerequisites vector

Design pseudocode that will search the data structure for a specific course and print out course information and prerequisites:

INPUT courseToSearch

SET ITERATOR to FIND course in vector // use C++ built in FIND

IF ITERATOR does not reach end of vector // found item

CALL void printCourseInformation(Vector<Course> courses,

String courseNumber) {

for all courses

if the course is the same as courseNumber

print out the course information

IF prerequisite is NOT NULL THEN

for each prerequisite of the course

print the prerequisite course

information

}

ELSE

OUTPUT course not found

# Hash Table Data Structure Pseudocode:

The following pseudocode define:

* Design pseudocode to define how the program opens the file, reads the data from the file, parses each line, and checks for file format
* Design pseudocode to show how to create course objects and store them in the appropriate data structure
* Design pseudocode that will print out course information and prerequisites

The course structure is to be applied to a hash table. This will store and print the data.

\*\* To initially parse the data the pseudocode will be reused from 3-3 Vector Data Milestone

Design pseudocode to define how the program opens the file, reads the data from the file, parses each line, and checks for file format:

OPEN file

WHILE reading each line until end of file

ADD line to fileVector

FOR all lines in fileVector // to remove any incomplete data

SPLIT into a temporary splitVector BY commas

IF splitVector size < 2 THEN

REMOVE line from fileVector

CONTINUE

ELSE

ADD course from splitVector at 0 to a courseList

vector

IF splitVector size > 2 THEN

FOR each course after splitVector at 2

IF course is NOT in courseList

REMOVE line from fileVector

CONTINUE

Design pseudocode to show how to create course objects and store them in the appropriate data structure:

DEFINE HashTable data structure

DEFINE Course Class as Object

CourseNumber as string

CourseName as string

CoursePrerequisites as vector of strings

FOR each line in fileVector

CREATE new CourseObject

SPLIT into a temporary splitVector BY commas

IF splitVector size is 2 THEN // has no prerequisite info

SET CourseObject.CourseNumber as splitVector at 0

SET CourseObject.CourseName as splitVector at 1

SET CourseObject.Prerequisites as NULL

PUSH CourseObject to the end of HashTable structure

ELSE

SET CourseObject.CourseNumber as splitVector at 0

SET CourseObject.CourseName as splitVector at 1

FOR remaining length of splitVector

ADD item of split vector into

CourseObject.Prerequisites vector

PUSH CourseObject to the end of HashTable structure

Design pseudocode that will print out course information and prerequisites:

ITERATE from beginning of nodes of the hashTable to end, FOR

SET key to hash value at given node CourseNumber

IF key is 0

CONTINUE to skip empty values

ELSE IF key is in use

PRINT Hashtable about Course info for given key

# Tree Data Structure Pseudocode:

The following pseudocode define:

* Design pseudocode to define how the program opens the file, reads the data from the file, parses each line, and checks for file format errors
* Design pseudocode to show how to create course objects and store them in the appropriate data structure
* Design pseudocode that will print out course information and prerequisites

The course structure is to be applied to a tree. This will store and print the data.

\*\* To initially parse the data the pseudocode will be reused from 3-3 Vector Data Milestone

Design pseudocode to define how the program opens the file, reads the data from the file, parses each line, and checks for file format errors:

OPEN file

WHILE reading each line until end of file

ADD line to fileVector

FOR all lines in fileVector // to remove any incomplete data

SPLIT into a temporary splitVector BY commas

IF splitVector size < 2 THEN

REMOVE line from fileVector

CONTINUE

ELSE

ADD course from splitVector at 0 to a courseList

vector

IF splitVector size > 2 THEN

FOR each course after splitVector at 2

IF course is NOT in courseList

REMOVE line from fileVector

CONTINUE

Design pseudocode to show how to create course objects and store them in the appropriate data structure:

DEFINE Tree data structure

DEFINE Course Class as Object

CourseNumber as string

CourseName as string

CoursePrerequisites as vector of strings

FOR each line in fileVector

CREATE new CourseObject

SPLIT into a temporary splitVector BY commas

IF splitVector size is 2 THEN // has no prerequisite info

SET CourseObject.CourseNumber as splitVector at 0

SET CourseObject.CourseName as splitVector at 1

SET CourseObject.Prerequisites as NULL

PUSH CourseObject to the end of Tree structure BY

IF there is no root THEN

SET CourseObject as root

ELSE TRAVERSE tree until there is no left or

right leaf THEN

INSERT CourseObject to leaf

ELSE

SET CourseObject.CourseNumber as splitVector at 0

SET CourseObject.CourseName as splitVector at 1

FOR remaining length of splitVector

ADD item of split vector into

CourseObject.Prerequisites vector

PUSH CourseObject to the end of Tree structure BY

IF there is no root THEN

SET CourseObject as root

ELSE TRAVERSE tree until there is no left or

right leaf THEN

INSERT CourseObject to leaf

Design pseudocode that will print out course information and prerequisites:

ITERATE through tree with PREORDER TRAVERSAL

VISIT root

PRINT Course information and prerequisites if any

TRAVERSE left subtree

FOR EACH node

PRINT Course information and prerequisites if any

TRAVERSE right subtree

FOR EACH node

PRINT Course information and prerequisites if any

# Menu Pseudocode:

The following pseudocode define:

* Load Data Structure: Load the file data into the data structure
* Print Course List: This will print an alphanumerically ordered list of all the courses in the Computer Science department.
* Print Course: This will print the course title and the prerequisites for any individual course
* Exit: This will exit you out of the program

Load Data Structure: Load the file data into the data structure:

INPUT Menu selection

// ... A single data structure will be determined later ...

READ data from file into Vector data structure

CALL Vector data structure file read method

READ data from file into Hash Table data structure

CALL Hash Table data structure file read method

READ data from file into Tree data structure

CALL Tree data structure file read method

Print Course List: This will print an alphanumerically ordered list of all the courses in the Computer Science department.:

INPUT Menu selection

// ... A single data structure will be determined later ...

// The following methods are already written and will print alphanumerically

CALL VectorDataStructure print course list method

CALL HashTableDataStructure print course list method

CALL TreeDataStructure print course list method

Print Course: This will print the course title and the prerequisites for any individual course:

INPUT Menu selection

INPUT Course information

// ... A single data structure will be determined later ...

SEARCH Course information according to the data structure

// The following methods are already written and will print the individual course information

CALL VectorDataStructure print course method

CALL HashTableDataStructure print course method

CALL TreeDataStructure print course method

Exit: This will exit you out of the program:

INPUT Menu selection

BREAK loop

END program

# Print Course List Pseudocode:

The following pseudocode define:

* Sorting algorithm for a Vector data structure and print course list in a display from a Vector data structure
* Sorting algorithm for a Hash Table data structure and print course list in a display from a Hash Table data structure
* Sorting algorithm for a Tree data structure and print course list in a display from a Tree data structure

Sorting algorithm for a Vector data structure and print course list in a display from a Vector data structure:

SORT Vector with std::sort()

FOR EACH course in Vector

PRINT Course information

Sorting algorithm for a Hash Table data structure and print course list in a display from a Hash Table data structure:

// To sort a HashTable the values must be read into a different structure

CREATE Course Vector

FOR EACH Course in HashTable

ADD Course to CourseVector

SORT CourseVector with std::sort()

FOR EACH Course in CourseVector

PRINT Course information

Sorting algorithm for a Tree data structure and print course list in a display from a Tree data structure:

SORT Tree with InOrder Traversal

TRAVERSE left subtree

PRINT Course Information per node

RECURSIVE CALL InOrder(left->subtree)

REACH root of Tree

TRAVERSE right subtree

PRINT Course Information per node

RECURSIVE CALL InOrder(right->subtree)

# Evaluation:

* Vectors
  + The reading of the file operation will be O(1)
  + The initial validation of the file data will be O(N^2) due to N number of lines in the file and N number of possible prerequisites associated with each course on the given line.
  + Parsing the file into the vector data structure will be O(N^2) for the given number of lines and courses. This is because the worst case scenario is that the operation will have to loop adding n number of courses and then loop n number of prerequisite courses for each course (N \* N)
  + Sorting a vector with the native std::sort() yields a documented worst time complexity of O(N logN)
  + Advantages and Disadvantages:
    - It is beneficial that the vector is native in C++ and does not require any extra configuration outside of adding values into a new vector
    - It is beneficial that the building of the primary course object vector could be done in the parsing file step reducing the overall size of the code
    - It is beneficial that vectors already have an effective sort mechanism
    - A downside is that it is not possible to effectively associate courses with their prerequisites sense there is no clear native association
    - A downside is the slower search algorithms if the vector is unsorted
  + Runtime Analysis:

| Code | Line Cost | Times Executed | Total Cost |
| --- | --- | --- | --- |
| Reading File | 1 | 1 | 1 |
| Validating File | N | N | N^2 |
| Adding to Vector Data Structure | N | N | N^2 |
| Sorting | logN | N | N log N |
| Print Course information | 1 | 1 | 1 |
| Print Prerequisite Course Information | 1 | N | N |
|  |  | **Total Cost** | **(N logN) + 2N^2 + N + 2** |
|  |  | **Runtime** | **O(N^2)** |

* HashTable
  + The reading of the file operation will be O(1)
  + The initial validation of the file data will be O(N^2) due to N number of lines in the file and N number of possible prerequisites associated with each course on the given line.
  + Parsing the file into the HashTable data structure will be O(N^2) for the given number of lines and courses. This is because the worst case scenario is that the operation will have to loop adding n number of courses and then loop n number of prerequisite courses for each course (N \* N)
  + Sorting a HashTable is not possible in its given form therefore it will have to be transformed into a vector first which will be O(N) for the given number of keys. It can then utilize the native std::sort() for the new vector which will be O(N logN). Therefore the worst possible complexity to sort a hashtable will be O(N^2 logN)
  + Advantages and Disadvantages:
    - A benefit is that a key will be generated for each course and that can be effectively used to associate courses with their prerequisites and vice versa
    - A downside is that a hash table is not a hash map. Therefore the key does not have an innate meaning. Therefore sorting can not effectively be done and it has to be outsourced to another data structure essentially nullifying any benefit to having the hashtable to begin with.
    - A benefit is the effective search ability through key values
  + Runtime Analysis:

| Code | Line Cost | Times Executed | Total Cost |
| --- | --- | --- | --- |
| Reading File | 1 | 1 | 1 |
| Validating File | N | N | N^2 |
| Adding to Hash Table Data Structure | N | N | N^2 |
| Building Hash Table Data Structure | 1 | 1 | 1 |
| Sorting | logN | N^2 | N^2 log N |
| Print Course information | 1 | 1 | 1 |
| Print Prerequisite Course Information | 1 | N | N |
|  |  | **Total Cost** | **(N^2 logN) + 2N^2 + N + 3** |
|  |  | **Runtime** | **O(N^2)** |

* Tree
  + The reading of the file operation will be O(1)
  + The initial validation of the file data will be O(N^2) due to N number of lines in the file and N number of possible prerequisites associated with each course on the given line.
  + In order to build the Tree data structure it will be O(N^2) due to N number of courses and N number of prerequisites. Traversing the tree evenly on each side is O(1) therefore it does not factor into the complexity.
  + The sorting of the Tree is done through InOrder Traversal. Therefore the complexity is O(N) where N is the number of nodes that are of the tree that need to be traversed
  + Advantages and Disadvantages:
    - An advantage is that the tree structure will resemble the general path for courses and prerequisites more accurately
    - There are a number of features of a tree structure that lean to intuitive sorting by traversals and effective searching
    - A downside is that the general sorting of this is not as efficient as the other structures
    - Another downside is that the initial creation of the tree structure is time consuming and less effective for memory
  + Runtime Analysis:

| Code | Line Cost | Times Executed | Total Cost |
| --- | --- | --- | --- |
| Reading File | 1 | 1 | 1 |
| Validating File | N | N | N^2 |
| Adding to Tree Data Structure | N | N | N^2 |
| Building Tree Data Structure | 1 | 1 | 1 |
| Sorting | 1 | N | N |
| Print Course information | 1 | 1 | 1 |
| Print Prerequisite Course Information | 1 | N | N |
|  |  | **Total Cost** | **2N^2 + 2N + 3** |
|  |  | **Runtime** | **O(N^2)** |

# Recommendation:

After considering all the factors I firmly believe that a vector data structure would be the most effective way to organize the course information. The sorting algorithm Big O is the best by far ( O(N log N) ) and reading the file, parsing and validating the line and adding courses to the data structure could all be paired effectively with a time complexity of O(N^2) which is less than a number of the other structures. Vectors are already well established in the C++ framework and manage memory effectively limiting leaks. The searching complexities are also dramatically improved if the vector is sorted first. The printing of the courses will only be O(N) for N number of courses as the program iterates through the vector.