**Justin Brown**

**CS 300**

**10/7/22**

**Project One**

**Vector Data Structure**

**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 using ofstream

While(!eof)

Read file using fstream

Variables: courseNum, courseTitle, coursePreReqs

CHECK that there are 2 parameters on each line

IF not 2 parameters

PRINT error

ELSE

continue

CHECK that end of line has another line that begins with courseNum

IF not

Print error

ELSE

Continue

**Pseudocode to show how to create course objects and store them in the appropriate data structure**

Create a course structure that holds:

Course number

Course title

Course prereqs

While(!eof)

CHECK that there are 2 parameters on each line

IF not 2 parameters

PRINT error

ELSE

CREATE course object for each line from FILE

ADD course object to <vector> classInfo

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

void printCourseInformation(Vector<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

}

int numPrerequisiteCourses(Vector<Course> courses, Course c) {

totalPrerequisites = prerequisites of course c

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

**Create pseudocode for a menu**

LOAD data into Vector (choice 1)

(choice 2) printCourseList(Vector<course> courses){

}

(choice 3) printCourse(){

}

(choice 4) exit

break;

**Design pseudocode that will print out the list of the courses in the Computer Science Program in alphanumeric order**

FOR courses in vector<course>

sort(course.begin(); course.end());

FOR i = course.begin; i<course.end; i++

PRINT i

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line-Cost** | **#Times Executed** | **Total Cost** |
| While(!eof) | 1 | n | n |
| Check 2 params each line | 1 | n | n |
| IF not 2 params | 1 | n | n |
| PRINT error | 1 | 1 | 1 |
| Check next line | 1 | n | n |
| IF ! begin w course Num | 1 | n | n |
| Create course obj FOR EACH line in file | 1 | n | n |

Total Cost = 6n + 1

RunTime = O(n)

A vector is a data structure that offers both advantages and disadvantages. An advantage to a vector is that it can change in size. So if a certain computer science course got added it would be able to be added to a vector. A disadvantage in a vector is that it can be costly on memory. A worst-case runtime of a vector is O(n) this isn’t bad. So it largely depends on the size of the data in the vector.

**Hash Table Data Structure**

OPEN file using ofstream

While(!eof)

Read file using fstream

Variables: courseNum, courseTitle, coursePreReqs

CHECK that there are 2 parameters on each line

IF not 2 parameters

PRINT error

ELSE

continue

CHECK that end of line has another line that begins with courseNum

IF not

Print error

ELSE

Continue

**int numPrerequisiteCourses(Hashtable<Course> courses) {**

return numOfPrereqs

**}**

**void printSampleSchedule(Hashtable<Course> courses) {**

FOR Loop

WHILE I < hashTableSize, i++

If i! = NULL

// the sampleSchedule

Cout << courses

**}**

**void printCourseInformation(****Hashtable<Course> courses, String courseNumber) {**

FOR Loop

WHILE I < hashTableSize, i++

If i! = NULL

FOR loop for courseNum

Cout << course num

Cout << title

Cout << prereqs

**}**

While(!eof)

CHECK that there are 2 parameters on each line

IF not 2 parameters

PRINT error

ELSE

CREATE course object for each line from FILE

ADD course object to Hashtable<Course> courses

**Create pseudocode for a menu**

LOAD data into HashTable (choice 1)

(choice 2)

void printCourseInformation(Hashtable<Course> courses, String courseNumber) {

FOR Loop

WHILE I < hashTableSize, i++

If i! = NULL

FOR loop for courseNum

Cout << course num

Cout << title

Cout << prereqs

}

(choice 3) printCourse(){

For course in hashTable

Print course

}

(choice 4) exit

**Pseudocode that will print out the list of the courses in the Computer Science program in alphanumeric order**

Check each key then the value it gives

Sort the values

PRINT the values in ascending order to the screen

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line-Cost** | **#Times Executed** | **Total Cost** |
| While(!eof) | 1 | n | n |
| Check 2 params each line | 1 | n | n |
| IF not 2 params | 1 | n | n |
| PRINT error | 1 | 1 | 1 |
| Check next line | 1 | n | n |
| IF ! begin w course Num | 1 | n | n |
| Create course obj FOR EACH line in file and add to hash table | 1 | n | 1 |

Worst case runtime will be O(n), however the average case is O(1)

A hashtable is another data structure that provides both advantages and disadvantages. Many of the great benefits of a hashtable is how quickly it can perform its different operations. This offers an average runtime of constant time. Making it quicker than a vector. The disadvantages of the hashtable are seen when there is a lot of data, and it isn’t able to be optimally implemented.

Tree Pseudocode

OPEN file using ofstream

While(!eof)

Read file using fstream

Variables: courseNum, courseTitle, coursePreReqs

CHECK that there are 2 parameters on each line

IF not 2 parameters

PRINT error

ELSE

continue

CHECK that end of line has another line that begins with courseNum

IF not

Print error

ELSE

Continue

Create a course structure that holds:

Course number

Course title

Course prereqs

CHECK that there are 2 parameters on each line

IF not 2 parameters

PRINT error

ELSE

CREATE course object for each line from FILE

ADD course object to tree data structure

int numPrerequisiteCourses(Tree<Course> courses) {

RETURN the number of prereq courses

}

void printSampleSchedule(Tree<Course> courses) {

PRINT to screen a sample schedule FROM tree data structure

}

void printCourseInformation(Tree<Course> courses, String courseNumber) {

PRINT course number

PRINT course title

PRINT course prereqs

}

**2)**. **Create Pseudocode for a menu**

LOAD data into Tree (choice 1)

(choice 2) print Course List

void printCourseInformation(Tree<Course> courses, String courseNumber) {

PRINT course number

PRINT course title

PRINT course prereqs

}

(Choice 3) print Course

(Choice 4) Exit

Return 0

**3). Design Pseudocode that will print out the list of the courses in the Computer Science program in alphanumeric order**

Sort the courses in the cs program

Build a tree with the sorted courses

Traverse the tree in order

Print the courses

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line-Cost** | **#Times Executed** | **Total Cost** |
| While(!eof) | 1 | n | n |
| Check 2 params each line | 1 | n | n |
| IF not 2 params | 1 | n | n |
| PRINT error | 1 | 1 | 1 |
| Check next line | 1 | n | n |
| IF ! begin w course Num | 1 | n | n |
| Create course obj FOR EACH line in file and add to tree | 1 | n | n |

Runtime O(n)

The tree data structure has both advantages and disadvantages. A main advantage of the tree data structure is seen with big data sets. It can provide logarithmic runtime for its operations. A disadvantage of the tree data type is these good runtimes depend on if the tree we have is balanced or not.

After completing the analysis of all three data structures, I will be coding my second project using a hash table. The constant time that it provides, given the data set, will make the code run the quickest. This is important when writing a program to fill the requirements that the advising provided.