//Open the file

Void openFile(courseInformation) {

Try {

OF = OPEN(courseInformation)

}

Catch {

Print “Unable to open file”

}

}

//Read and store info

While not eof

Read line

If line has less than 2 values

Error

Else

read the parameters

If more than 2 parameters

If parameter in the first parameter

Keep reading lines

Else

error

**Pseudocode for each data structure**

// Vector pseudocode

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

}

void printSampleSchedule(Vector<Course> courses) {

for (i = 0, to length of courses – 1) {

if courseNumber at i is equal to the course {

return i

}

}

}

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**

}

// Hashtable pseudocode

int numPrerequisiteCourses(Hashtable<Course> courses) {

for each prerequisite in the total amount of prerequisites

add prerequisites to int totalPrerequisites

print totalPrerequisites

}

void printSampleSchedule(Hashtable<Course> courses) {

if the course has prerequisites

print that course’s prerequisites

}

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

for an unsigned integer equals to 0, is less than the table size, and increases by one

if the node key is not equal to the unit max

print the integer node course id and prerequisites

while the next node is not null

node equals the next node

print the node key course id and prerequisites

return

}

// Tree pseudocode

int numPrerequisiteCourses(Tree<Course> courses) {

root equals null

if root null

current course is root

else if course number is less than root

add to left node

if left is null

add course number

else

add to right node

if right is null

add course number

}

void printSampleSchedule(Tree<Course> courses) {

traverse inorder the courses root

print information

}

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

if root is not null

traverse left

if found

output information

traverse right

if found

output information

}

**Pseudocode for menu**

// Vector pseudocode

Void Menu

If input is 1

Load file data

If input is 2

printCourseInformation(Vector<Course> courses, String courseNumber)

if input is 3

Request information for course ID

If vector not null

If input and course id are equal

Print course.courseID and course prerequisites

If input is 4

end

// Hash table pseudocode

Void menu

If input is 1

Load file data

If input is 2

printCourseInformation(Hashtable<Course> courses, String courseNumber)

if input is 3

input course id

key equals course id

if current node equals key

output course

else

while not null

check node against key

if node is null

return null

if key equals course id

output course information

next node

if input is 4

end

// Tree pseudocode

If input is 1

Load file data

If input is 2

printCourseInformation(Tree<Course> courses, String courseNumber)

if input is 3

input course id

root equals current node

while current node not null

if current equals course id

print course id and name

if prerequisites

print prerequisites

if root is less than course id

set current to left

else set current to right

if input equals 4

end

| **Vector** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create vector** | 1 | 1 | 1 |
| **For each line in the file** | 1 | n | n |
| **Create vector course item for course id, course name, and prerequisite** | 3 | n | n |
| **Total Cost** | | | 4n + 1 |
| **Runtime** | | | O(n) |

| **Hash Table** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create hash table** | 1 | 1 | 1 |
| **Create key** | 1 | 1 | 1 |
| **For each line in the file** | 1 | N | N |
| **If no data in the node** | 1 | N | N |
| **Add data to the key** | n | N | N |
| **Else** | 1 | N | N |
| **Set old key to max** | 1 | N | N |
| **Set old course and prerequisite to nullptr** | 2 | N | N |
| **Find open node** | n | N | N |
| **Fill node** | 1 | N | N |
| **Add node to end** | 1 | N | N |
| **Total Cost** | | | 9n + 2 |
| **Runtime** | | | O(n) |

| **Tree** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create Tree** | 1 | 1 | 1 |
| **If root is null** | 1 | 1 | 1 |
| **Add root** | 1 | 1 | 1 |
| **If node is less than root** | 1 | N | N |
| **Left node** | 1 | N | N |
| **If node is greater than root** | 1 | N | N |
| **Right node** | 1 | N | N |
| **For each line in the file** | 1 | N | N |
| **Create vecter with course id, course name, and prerequisite** | 3 | N | N |
| **Total Cost** | | | 7n + 3 |
| **Runtime** | | | O(n) |

Advantages of using vectors is that they read and add files much faster than the other two methods. They are much simpler to manipulate with adding and removing data. However, vectors have a disadvantage when searching for data as it will have to read through each line of data in order to find the desired data. Hash tables have a better advantage when it comes to searching for data, since each item has a key for it. With these keys, however, hash tables are much slower for adding data and initialing setting up the table. Binary trees are best for sorting the information. With the different routes and branches, it can be easier than vectors to find data. However, this only works if the tree is balanced. If not, the search time will be much slower.

I would recommend using a hash table for this project. While setting up the official code and list will take more time, the final result will be a program that is very quick to sort and search for data.