Pseudocode

File Intake:

Vector <String> intakeToVector(string file){

Declare Vector<String> newVec

Declare string currLine

Open file with fstream

If (file does not open)

Throw error message

Return

Iterate through file delimited by newline

Append line to newVec

Close file

return newVec

}

Create Courses:

For Some Course{

String courseId

String courseTitle

Vector<String> prereqs

}

Vector

Vector<Course> createCourseList(Vector<string> courseStrVec){

Declare Vector<Course> newCourseList

Declare stringstream ss of courseString

Declare int count

For Each courseString in courseStrVec

Count = 0

Declare course newCourse

While ss is good:

Getline of ss, store in tempStr, delimit by comma(,)

If count is 0: set tempStr to courseId

ElIf count is 1: set tempStr to courseTitle

Else: append tempStr to prereqs

Append newCourse to newCourseList

Return newCourseList

}

Void printList(Vector <Course> courseList){

For currCourse in courseList:

Print courseId and courseTitle

If prereqs is not empty:

For currPrereq in prereqs:

Print currPrereq

}

Vector<String> sortList (Vector<string> courseList){

Vector sort courseList from beginning to end:

If currCourse courseId is less than nextCourse courseId

Swap currCourse and nextCourse

}

Hash table

For some Hashtable{

Node node{

Course course

unsigned int key

Node next

}

Vector<Node> nodes

unsigned int tableSize = DEFAULT\_SIZE

unsigned int hash(int key)

}

Void printTable(){

For all node in nodes:

If currNode’ key is not UNIT\_MAX

Print node course’s info

Set nextNode to node’s next

while next node is not null:

print nextNode’s course’s info

set nextNode to nextNode’s next

}

Course createCourse(string courseLine){

Declare Count = 0

Declare course newCourse

While ss is good:

Getline of ss, store in tempStr, delimit by comma(,)

If count is 0: set tempStr to courseId

ElIf count is 1: set tempStr to courseTitle

Else: append tempStr to prereqs

Increment count by 1

Return newCourse

}

Void fillTable(Vector<string> courseStrVec){

Declare stringstream ss of courseString

For Each courseString in courseStrVec

Declare Course newCourse as createCourse(courseString)

Declare unsigned int key based on newCourse’s coursed

If nodes does not contain key:

Create newNode with newCourse and key

Insert into nodes

Else:

if node is not used:

Passing old node key to UNIT\_MAX, set to key, set old node to Course and old node next to null pointer

Else:

find the next open node

add new newNode to end

create the key for the given Course

}

Tree

For some Node node{

Course course

Node left

Node right

}

For some Tree{

Node root

}

Void printTreePreorder(Node node){

if node is not equal to null ptr

output node’s course’s info

printTreePreorder node’s left

printTreePreorder node’s right

}

Course createCourse(string courseLine){

Declare Count = 0

Declare course newCourse

While ss is good:

Getline of ss, store in tempStr, delimit by comma(,)

If count is 0: set tempStr to courseId

ElIf count is 1: set tempStr to courseTitle

Else: append tempStr to prereqs

Increment count by 1

Return newCourse

}

Void Insert(Course course){

If root is null: root becomes new Node with newCourse

Else:

Call addNode with root and course

}

Void addNode(Node node, Course course){

If node courseId is larger than course’s courseId:

If no left node

this node becomes left

Else:

call addNode with node’s left and course

Else:

If no right node:

this node becomes right

Else:

Call addNode with node’s right and course

}

Void fillTree(Vector<string> courseStrVec){

Declare stringstream ss of courseString

For Each courseString in courseStrVec

Declare Course newCourse as createCourse(courseString)

Call insert with newCourse

Menu:

Switch:

Case 1:

Vector:

Call createCourseList with file

Hash table:

Call fillTable with file

Tree:

Call fillTree

Case 2:

Vector:

Sort by courseId

Call printList

Hash table:

Sort by courseId

Call printTable

Tree:

Call printByPreorder

Case 3:

Intake courseId

Call Search with courseId

Else Return

Time Complexity Tables

Vector

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Declare stringstream ss of courseString | 1 | 1 | 1 |
| For Each courseString in courseStrVec | 1 | n | n |
| Count equals 0 | 1 | 1 | 1 |
| Create var new Course newCourse | 1 | 1 | 1 |
| Getline of ss, store in tempStr, delimit by comma(,) | 1 | n | n |
| If count is 0: set tempStr to courseId | 1 | 1 | 1 |
| ElIf count is 1: set tempStr to courseTitle | 1 | 1 | 1 |
| Else: append tempStr to prereqs | 1 | 1 | 1 |
| Append newCourse to newCourseList | 1 | n | n |
| **Total Cost** | | | 3n +6 |
| **Runtime** | | | O(n) |

Hash

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Declare stringstream ss of courseString | 1 | 1 | 1 |
| For Each courseString in courseStrVec | 1 | n | n |
| Create var new Course newCourse as createCourse(courseString) | 1 | n | n |
| Create unsigned int key based on newCourse’s courseId | 1 | 1 | 1 |
| If nodes does not contain key: | 1 | n | n |
| Create newNode with newCourse and key | 2 | 1 | 1 |
| Insert into nodes | 1 | 1 | 1 |
| Else | 1 | 1 | 1 |
| if node is not used: | 1 | n | n |
| Passing old node key to UNIT\_MAX, set to key, set old node to Course and old node next to null pointer | 2 | 1 | 1 |
| Else: | 1 | n | n |
| find the next open node | 1 | n | n |
| add new newNode to end | 1 | 1 | 1 |
| create the key for the given Course | 1 | 1 | 1 |
| Declare Count = 0 | 1 | 1 | 1 |
| Declare course newCourse | 1 | 1 | 1 |
| While ss is good: | 1 | n | n |
| Getline of ss, store in tempStr, delimit by comma(,) | 1 | 1 | 1 |
| If count is 0: set tempStr to courseId | 2 | 1 | 1 |
| ElIf count is 1: set tempStr to courseTitle | 2 | 1 | 1 |
| Else: append tempStr to prereqs | 2 | 1 | 1 |
| Increment count by 1 | 1 | 1 | 1 |
| Return newCourse | 1 | 1 | 1 |
| **Total Cost** | | | 7n+16 |
| **Runtime** | | | O(n) |

Tree

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Declare stringstream ss of courseString | 1 | 1 | 1 |
| For Each courseString in courseStrVec | 1 | n | n |
| Create var new Course newCourse as createCourse(courseString) | 1 | n | n |
| Call insert with newCourse | 1 | n | n |
| If root is null: root becomes new Node with newCourse | 2 | 1 | 1 |
| Else: | 1 | 1 | 1 |
| Call addNode with root and course | 1 | n | n |
| If node courseId is larger than course’s courseId: | 1 | 1 | 1 |
| If no left node | 1 | 1 | 1 |
| this node becomes left | 1 | 1 | 1 |
| Else: | 1 | 1 | 1 |
| call addNode with node’s left and course | 1 | n | n |
| Else: | 1 | 1 | 1 |
| If no right node: | 1 | 1 | 1 |
| this node becomes right | 1 | 1 | 1 |
| Else: | 1 | 1 | 1 |
| Call addNode with node’s right and course | 1 | n | n |
| Declare Count = 0 | 1 | 1 | 1 |
| Declare course newCourse | 1 | 1 | 1 |
| While ss is good: | 1 | n | n |
| Getline of ss, store in tempStr, delimit by comma(,) | 1 | 1 | 1 |
| If count is 0: set tempStr to courseId | 2 | 1 | 1 |
| ElIf count is 1: set tempStr to courseTitle | 2 | 1 | 1 |
| Else: append tempStr to prereqs | 2 | 1 | 1 |
| Increment count by 1 | 1 | 1 | 1 |
| Return newCourse | 1 | 1 | 1 |
| **Total Cost** | | | 7n+19 |
| **Runtime** | | | O(n) |

Analysis

The three forms of data storage offer unique capabilities for storing data. A vector allows for the simple creation of the structure with a low total cost, with a trade-off of having a potentially poor search method of O(n). The hash table allows for easy search but at the cost of not being able to sort. The binary tree is generated in order but would also have the worst search speed of O(n). With all three having a runtime complexity of O(n), any of the three would be a valid choice. I recommend using the vector structure since all three structures show similar time complexity, and the vector’s sort can be easily handled without having to use the structuring time of a tree.