Course Struct {string courseCode, string courseName, string array preReqs}

PRIVATE vector<HashNode> courseHashedList

CREATE vector<Course> courseList

STRUCT HashNode{int key, Course course, Node\* next}

PRIVATE TreeNode\* root

STRUCT TreeNode{Course course, TreeNode\* left, TreeNode\* right}

Vector Read File

OPEN file

CREATE string temp

READLINE and store in temp

WHILE file is not empty

CREATE Course tempCourse

IF temp does not contain “,”

OUTPUT “Error in file format”

END

ELSE

STORE index of “,” in tempIndex

STORE substring of start to tempIndex in tempCourse.courseCode

IF index of “,” after tempIndex is not -1

STORE index of “,” after tempIndex in nextComma

STORE substring of tempIndex + 1 to nextComma in

tempCourse.courseName

IF index of “,” after nextComma is not -1

WHILE index of “,” after nextComma is not -1

STORE substring of nextComma + 1 to index of “,” after

nextComma in tempStr

SEARCH courseList for tempStr

IF found

STORE tempStr in tempCourse.preReq.PushBack

STORE index of “,” after nextComma in

nextComma

ELSE

OUTPUT “Pre req not found!”

END

ELSE

STORE substring of nextComma to end in tempStr

SEARCH courseList of tempStr

IF found

STORE tempStr in tempCourse.preReq.PushBack

STORE index of “,” after nextComma in

nextComma

ELSE

OUTPUT “Pre req not found!”

END

ELSE

STORE substring of tempIndex + 1 to end in tempCourse.courseName

ADD tempCourse to courseList

READLINE and store in temp

Print Course Info Vector (string courseNumber)

FOR all courses

IF course is the same as courseNumuber

OUTPUT course information

FOR each preReq in course

OUTPUT preReq course information

Read File Hash Table

CREATE HashNode tempNode

OPEN file

CREATE string temp

READLINE and store in temp

WHILE file is not empty

CREATE Course tempCourse

IF temp does not contain “,”

OUTPUT “Error in file format”

END

ELSE

STORE index of “,” in tempIndex

STORE substring of start to tempIndex in tempCourse.courseCode

IF index of “,” after tempIndex is not -1

STORE index of “,” after tempIndex in nextComma

STORE substring of tempIndex + 1 to nextComma in

tempCourse.courseName

IF index of “,” after nextComma is not -1

WHILE index of “,” after nextComma is not -1

STORE substring of nextComma + 1 to index of “,” after

nextComma in tempStr

SEARCH courseHashedList for tempStr

IF found

STORE tempStr in tempCourse.preReq.PushBack

STORE index of “,” after nextComma in

nextComma

ELSE

OUTPUT “Pre req not found!”

END

ELSE

STORE substring of nextComma to end in tempStr

SEARCH courseHashedList of tempStr

IF found

STORE tempStr in tempCourse.preReq.PushBack

STORE index of “,” after nextComma in

nextComma

ELSE

OUTPUT “Pre req not found!”

END

ELSE

STORE substring of tempIndex + 1 to end in tempCourse.courseName

GENERATE hash key for courseID

INSERT tempCourse at key

READLINE and store in temp

HashINSERT (Key, course)

SET tempNode as node at Key in courseHashedList

IF tempNode is null

CREATE newNode with key and course

INSERT newNode at courseList.begin + key

ELSE

IF tempNode’s key is UINT MAX

SET tempNode’s key to key

SET tempNode’s course to course

SET tempNode’s next to null

ELSE

WHILE tempNode’s next is not null

SET tempNode to tempNode’s next

SET tempNode’s next to a new Node

PrintCourseHashTable (CourseID)

CREATE hash key from CourseID

GET node at key in courseHashedList

IF node is null or node’s key is UINT MAX

RETURN

IF node is not null and node’s key is not UINT MAX and node’s course’s courseID is equal to courseID

OUTPUT course information

FOR each preReq in course

OUTPUT preReq course information

WHILE node is not null

IF node’s key is not UINT MAX and node’s course’s courseID is equal to courseID

OUTPUT course information

FOR each preReq in course

OUTPUT preReq course information

SET node to node’s next

Read File Tree

CREATE TreeNode tempNode

OPEN file

CREATE string temp

READLINE and store in temp

WHILE file is not empty

CREATE Course tempCourse

IF temp does not contain “,”

OUTPUT “Error in file format”

END

ELSE

STORE index of “,” in tempIndex

STORE substring of start to tempIndex in tempCourse.courseCode

IF index of “,” after tempIndex is not -1

STORE index of “,” after tempIndex in nextComma

STORE substring of tempIndex + 1 to nextComma in

tempCourse.courseName

IF index of “,” after nextComma is not -1

WHILE index of “,” after nextComma is not -1

STORE substring of nextComma + 1 to index of “,” after

nextComma in tempStr

SEARCH courseList for tempStr

IF found

STORE tempStr in tempCourse.preReq.PushBack

STORE index of “,” after nextComma in

nextComma

ELSE

OUTPUT “Pre req not found!”

END

ELSE

STORE substring of nextComma to end in tempStr

SEARCH courseList of tempStr

IF found

STORE tempStr in tempCourse.preReq.PushBack

STORE index of “,” after nextComma in

nextComma

ELSE

OUTPUT “Pre req not found!”

END

ELSE

STORE substring of tempIndex + 1 to end in tempCourse.courseName

INSERT tempCourse

READLINE and store in temp

TreeINSERT (course)

IF root is null

SET root as new TreeNode(course)

ELSE

CALL addNode(root, course)

addNode(TreeNode\* node, Course course)

IF node’s course’s courseID is greater than course’s courseID

IF node’s leftNode is null

SET node’s leftNode to new Node course

ELSE

CALL addNode(node’s leftNode, course)

ELSE

IF node’s rightNode is null

SET node’s rightNode to new Node course

ELSE

CALL addNode(node’s rightNode, course)

PrintCourseTree (CourseID)

SET TreeNode\* curr to root

WHILE curr is not null

IF curr’s course’s courseID is equal to course ID

OUTPUT course information

FOR each preReq in course

OUTPUT preReq course information

IF curr’s course’s courseID is greater than courseID

SET curr to curr’s left

IF curr’s course’s courseID is less than courseID

SET curr to curr’s right

**MENU**

CREATE int choice

OUTPUT 1. Load Vector Structure, newline, 2. Load Hash Structure, newline, 3. Load Tree Structure, newline, 4. Print Course List, newline, 5. Print Course, newline, 9. Exit, newline

GET choice

WHILE choice is not equal to 9

IF choice is 1

CALL Vector Read File

IF choice is 2

CALL Read File Hash Table

IF choice is 3

CALL Read File Tree

IF choice is 4

CREATE int outputChoice

OUTPUT 1. Print all from vector, newline, 2. Print all from hash table, newline,

3. Print all from tree, newline

GET outputChoice

IF outputChoice is 1

CALL print all from vector

IF outputChoice is 2

CALL print all from hash

IF outputChoice is 3

CALL print all from tree

IF choice is 5

CREATE outputChoice

OUTPUT Which course ID are you looking for?

GET courseID and store in string

OUTPUT 1. Get course from vector, newline, 2. Get course from hashtable,

Newline, 3. Get course from tree, newline

GET outputChoice

IF outputChoice is 1

CALL Print Course info Vector(courseID)

IF outputChoice is 2

CALL PrintCourseHashTable(courseID)

IF outputChoice is 3

CALL PrintCourseTree(courseID)

OUTPUT 1. Load Vector Structure, newline, 2. Load Hash Structure, newline, 3. Load

Tree Structure, newline, 4. Print Course List, newline, 5. Print Course, newline, 9. Exit, newline

GET choice

**Print All From Vector**

CALL c++ sort method and store in sortedVector

FOR int I = 0 and I is less than sortedVector’s size while incrementing I by 1

CALL Print Course Info Vector (sortedVector.at(i)

**Print All From Hash Table //from doing some research is seems like hashTables are really not made for sorting**

CREATE HashNode least equal to -1

FOR int I equals 0 and I is less than courseHashedList’s size - 1 and incrementing I by 1

FOR int j equals I+1 and j is less than courseHashList’s size and increment j by 1

IF courseHashList at j is not null

IF least is equal to -1

SET least to j

IF courseHashList at j’s courseID is less than courseHastList at least’s courseID

SET least to j

IF least is not equal to i

SWAP courseHashList at I and courseHashList at least

FOR I = 0 and I less than courseHashList’s size and incrementing I by 1

IF courseHashList at I is null or node’s key is UINT MAX

RETURN

ELSE

OUTPUT course information

FOR each preReq in course

OUTPUT preReq course information

WHILE node is not null

IF node’s key is not UINT MAX and node’s course’s courseID is equal to courseID

OUTPUT course information

FOR each preReq in course

OUTPUT preReq course information

SET node to node’s next

**Print All From Tree**

CALL inOrder (root)

**inOrder (TreeNode\* node)**

if node is not null

CALL inOrder(node’s left)

OUTPUT course information

FOR each preReq in course

OUTPUT preReq course information

CALL inOrder(node’s right)

**Runtime Analysis for Loading Structures**

**Vector**

| Code | Line Cost | # Times Executes | Total Cost |
| --- | --- | --- | --- |
| OPEN file | 1 | 1 | 1 |
| CREATE string temp | 1 | 1 | 1 |
| READLINE and store in temp | 1 | 1 | 1 |
| WHILE file is not empty | 1 | N | N |
| CREATE Course tempCourse | 1 | N | N |
| IF temp does not contain “,” | 1 | N | N |
| OUTPUT “Error in file format” | 1 | N | N |
| END | 1 | N | N |
| ELSE | 1 | N | N |
| STORE index of “,” in tempIndex | 1 | N | N |
| STORE substring of start to tempIndex in tempCourse.courseCode | 1 | N | N |
| IF index of “,” after tempIndex is not -1 | 1 | N | N |
| STORE index of “,” after tempIndex in nextComma | 1 | N | N |
| STORE substring of tempIndex + 1 to nextComma in tempCourse.courseName | 1 | N | N |
| IF index of “,” after nextComma is not -1 | 1 | N | N |
| WHILE index of “,” after nextComma is not -1 | 1 | N | N |
| STORE substring of nextComma + 1 to index of “,” after nextComma in tempStr | 1 | N | N |
| SEARCH courseList for tempStr | 3 | N2 | 3n2 |
| IF found | 1 | N | N |
| STORE tempStr in tempCourse.preReq.PushBack | 1 | N | N |
| STORE index of “,” after nextComma in  nextComma | 1 | N | N |
| ELSE | 1 | N | N |
| OUTPUT “Pre req not found!” | 1 | N | N |
| END | 1 | N | N |
| ELSE | 1 | N | N |
| STORE substring of nextComma to end in tempStr | 1 | N | N |
| SEARCH courseList of tempStr | 3 | N2 | 3n2 |
| IF found | 1 | N | N |
| STORE tempStr in tempCourse.preReq.PushBack | 1 | N | N |
| STORE index of “,” after nextComma in  nextComma | 1 | N | N |
| ELSE | 1 | N | N |
| OUTPUT “Pre req not found!” | 1 | N | N |
| END | 1 | N | N |
| ELSE | 1 | N | N |
| STORE substring of tempIndex + 1 to end in tempCourse.courseName | 1 | N | N |
| ADD tempCourse to courseList | 1 | N | N |
| READLINE and store in temp | 1 | N | N |
| Total Cost | | | 6n2 + 32n + 3 |
| Runtime | | | O(n2) |

**HashTable**

| Code | Line Cost | # Times Executes | Total Cost |
| --- | --- | --- | --- |
| CREATE HashNode tempNode | 1 | 1 | 1 |
| OPEN file | 1 | 1 | 1 |
| CREATE string temp | 1 | 1 | 1 |
| READLINE and store in temp | 1 | 1 | 1 |
| WHILE file is not empty | 1 | N | N |
| CREATE Course tempCourse | 1 | N | N |
| IF temp does not contain “,” | 1 | N | N |
| OUTPUT “Error in file format” | 1 | N | N |
| END | 1 | N | N |
| ELSE | 1 | N | N |
| STORE index of “,” in tempIndex | 1 | N | N |
| STORE substring of start to tempIndex in tempCourse.courseCode | 1 | N | N |
| IF index of “,” after tempIndex is not -1 | 1 | N | N |
| STORE index of “,” after tempIndex in nextComma | 1 | N | N |
| STORE substring of tempIndex + 1 to nextComma in tempCourse.courseName | 1 | N | N |
| IF index of “,” after nextComma is not -1 | 1 | N | N |
| WHILE index of “,” after nextComma is not -1 | 1 | N | N |
| STORE substring of nextComma + 1 to index of “,” after nextComma in tempStr | 1 | N | N |
| SEARCH courseList for tempStr | 2 | N | 2n |
| IF found | 1 | N | N |
| STORE tempStr in tempCourse.preReq.PushBack | 1 | N | N |
| STORE index of “,” after nextComma in  nextComma | 1 | N | N |
| ELSE | 1 | N | N |
| OUTPUT “Pre req not found!” | 1 | N | N |
| END | 1 | N | N |
| ELSE | 1 | N | N |
| STORE substring of nextComma to end in tempStr | 1 | N | N |
| SEARCH courseList of tempStr | 2 | N | 2n |
| IF found | 1 | N | N |
| STORE tempStr in tempCourse.preReq.PushBack | 1 | N | N |
| STORE index of “,” after nextComma in  nextComma | 1 | N | N |
| ELSE | 1 | N | N |
| OUTPUT “Pre req not found!” | 1 | N | N |
| END | 1 | N | N |
| ELSE | 1 | N | N |
| STORE substring of tempIndex + 1 to end in tempCourse.courseName | 1 | N | N |
| GENERATE hash key for courseID | 1 | N | N |
| INSERT tempCourse at key | 13 | n | 13n |
| READLINE and store in temp | 1 | N | N |
| Total Cost | | | 49n + 4 |
| Runtime | | | O(n) |

**Binary Tree**

| Code | Line Cost | # Times Executes | Total Cost |
| --- | --- | --- | --- |
| CREATE TreeNode tempNode | 1 | 1 | 1 |
| OPEN file | 1 | 1 | 1 |
| CREATE string temp | 1 | 1 | 1 |
| READLINE and store in temp | 1 | 1 | 1 |
| WHILE file is not empty | 1 | N | N |
| CREATE Course tempCourse | 1 | N | N |
| IF temp does not contain “,” | 1 | N | N |
| OUTPUT “Error in file format” | 1 | N | N |
| END | 1 | N | N |
| ELSE | 1 | N | N |
| STORE index of “,” in tempIndex | 1 | N | N |
| STORE substring of start to tempIndex in tempCourse.courseCode | 1 | N | N |
| IF index of “,” after tempIndex is not -1 | 1 | N | N |
| STORE index of “,” after tempIndex in nextComma | 1 | N | N |
| STORE substring of tempIndex + 1 to nextComma in tempCourse.courseName | 1 | N | N |
| IF index of “,” after nextComma is not -1 | 1 | N | N |
| WHILE index of “,” after nextComma is not -1 | 1 | N | N |
| STORE substring of nextComma + 1 to index of “,” after nextComma in tempStr | 1 | N | N |
| SEARCH courseList for tempStr | 10 | N | 10n |
| IF found | 1 | N | N |
| STORE tempStr in tempCourse.preReq.PushBack | 1 | N | N |
| STORE index of “,” after nextComma in  nextComma | 1 | N | N |
| ELSE | 1 | N | N |
| OUTPUT “Pre req not found!” | 1 | N | N |
| END | 1 | N | N |
| ELSE | 1 | N | N |
| STORE substring of nextComma to end in tempStr | 1 | N | N |
| SEARCH courseList of tempStr | 10 | N | 10n |
| IF found | 1 | N | N |
| STORE tempStr in tempCourse.preReq.PushBack | 1 | N | N |
| STORE index of “,” after nextComma in  nextComma | 1 | N | N |
| ELSE | 1 | N | N |
| OUTPUT “Pre req not found!” | 1 | N | N |
| END | 1 | N | N |
| ELSE | 1 | N | N |
| STORE substring of tempIndex + 1 to end in tempCourse.courseName | 1 | N | N |
| INSERT tempCourse | 13 | n | 13n |
| READLINE and store in temp | 1 | N | N |
| Total Cost | | | 69n + 4 |
| Runtime | | | O(n) |

**Overall Analysis**

Using a vector to as the main data structure is the easiest to implement. This uses C++’s built in vector class which allows us to add, remove, and get the courses from an index. The C++ vector also has a built-in sort method which we can use to sort the list before printing. Some downsides to this structure are that when we are searching for a specific course we need to iterate through the entire list, which can run for the entire size of the list in worst case scenarios. When we add to this list we also are just adding in the order that the input file is in, there is no format or reasoning behind how we insert into the list. We also have to sort before printing which uses memory and time.

Using a hash table is more memory consuming, as we need to have a vector object that’s size is greater than the amount of data we have. This leads to a more complicated implementation as we need to use pointers and node linking. This least is by far the easiest to use for searching for a specific course or removing one. When we are searching it is just a call to create the hash value of the course ID, then a return of the hash value, looking to see if the courseID matches and if not then we search for linked nodes to that hash value. This is the fastest searching method. A major downside to the hash table is that it is not really made for sorting. In order to sort this structure, we had to implement a selection sort which runs at O(n2) complexity. This is can get very slow as the course list grows.

If we were to use a binary search tree for our data structure, there are many advantages with this structure. The biggest advantage to this structure is that it is sorted while it is created. This leads to our printing method being very easy to implement. Another advantage of this list is that, as long as it is balanced, searching for a course is just O(log n) complexity, which is quite fast. If this tree is not balanced, this will lead to O(n) time in the worst case. This structure also leads the use of recursion for its methods, which is easy to read and does not require writing as much code.

Based on the analysis of these three structures, we can see that all three have the same runtime when it comes to adding items to the list. All run at O(n) when adding to the structures, which provides no clear advantage. If we are searching for items specifically, we can see that the hash table will provide the fastest results for this case. This is then followed by the binary search tree and then the vector. Removing items from the vector is the easiest and fastest, followed by the hash table and then the binary tree. For sorting the binary search tree is the true winner. This sorts while items are being added to the list, leading to no extra process when trying to print the list. After that the vector would be next, followed by the hash table. In conclusion I believe the binary search tree would be the best structure to use. This structure is already sorted, allowing or print all method to run without any added overhead to sorting. When adding to the list, there is more code involved, but the overall complexity is not any slower or faster than the other two. In the average case of a balanced tree, the search for a single course is O(log n) which is not the fastest but it is still very fast. This data structure also does not use a vector that might have many unused indices, like a hash table would use, and this can be expanded easily if more courses are added.