**PSEUDOCODE**

Struct Course

string courseNumber

string courseTitle

vector prerequisite

END

VOID readFile(string file)

DECLARE vector<string> temp

DECLARE vector<string> courseNumbers

OPEN file

IF file does not open:

PRINT error

FOR each line in file:

temp = parse line by comma

ADD course number to courseNumbers

IF temp is less than 2:

PRINT error message

ELSE IF temp is more than 2:

IF the prerequisite number is not in courseNumbers

PRINT error message

CLOSE file

END

VOID loadVector(Course course)

ADD course to vector courses

END

VOID PrintVector()

FOR vector courses begin to end iterate:

PRINT courseNumber, courseName, each prerequisite

END

VOID loadHashTable(Course course)

CREATE key for the given course

node = node at the key position

IF node is null

CREATE a new node with course and key

ADD new node to HashTable courses

ELSE

WHILE node is not null

node = next node

CREATE a new node with course and key

END

VOID PrintHashTable()

FOR node being to end iterate

IF key is in use

PRINT courseNumber, courseName, each prerequisite

node = next node

WHILE node is not null

PRINT courseNumber, courseName, each prerequisite

node = next node

END

VOID loadTree(Course course)

IF root is null

CREATE a new node with course

ELSE

CALL addToTree with root and course

END

VOID addToTree(Node node, Course course)

IF node courseNumber is larger than course courseNumber

IF node left is null

CREATE a new node with course

ELSE

RECURSE addNode with left node and course

IF node courseNumber is smaller than course courseNumber

IF node right is null

CREATE a new node with course

ELSE

RECURSE addNode with right node and course

END

VOID PrintTree()

CALL printAll(root)

END

VOID printTree(Node node)

IF node is not null

printAll(left node)

PRINT courseNumber, courseName, each prerequisite

printAll(right node)

END

VOID courseMenu(string file)

PRINT menu options

SWITCH option

case 1: CALL loadStructures(file)

case 2: CALL printCourseList()

case 3: GET input course title

CALL printCourse(courseTitle)

case 4: EXIT

END

VOID loadStructures(string file)

DECLARE Course course

FOR each line in file:

temp = parse line by comma

SET course to temp values

loadVector(course)

loadHashTable(course)

loadTree(course)

Vector Merge(Vector coursesNumbers, left, j, right)

mergedSize = right – left + 1

mergePos = 0

rightPos = 0

mergedNumbers = new int[mergedSize]

leftPos = left

rightPos = j + 1

WHILE (leftPos <= j AND rightPos <= right):

IF (coursesNumbers[leftPos] <= coursesNumbers[rightPos]):

mergedNumbers[mergePos] = coursesNumbers[leftPos]

increment leftPos

ELSE

mergedNumbers[mergePos] = coursesNumbers[rightPos]

increment rightPos

increment mergePos

WHILE (leftPos <= j):

mergedNumbers[mergePos] = coursesNumbers[leftPos]

increment leftPos

increment mergePos

WHILE (rightPos <= right):

mergedNumbers[mergePos] = coursesNumbers[rightPos]

increment rightPos

increment mergePos

FOR mergePos = 0 to mergeSize less than 1:

courseNumbers[left + mergePos] = mergedNumbers[mergePos]

END

Vector MergeSort(Vector coursesNumbers, left, right)

j = 0

IF left is greater than right:

j = (left + right) / 2

MergeSort(courseNumbers, left, j)

MergeSort(courseNumbers, j + 1, right)

Merge(courseNumbers, left, j, right)

END

**EVALUATION**

readFile

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| DECLARE vector<string> temp | 1 | 1 | 1 |
| **DECLARE vector<string> courseNumbers** | 1 | 1 | 1 |
| **OPEN file** | 1 | n | n |
| **IF file does not open:** | 1 | n | n |
| **PRINT error** | 1 | 1 | 1 |
| **FOR each line in file** | 1 | n | n |
| **temp = parse line by comma** | 1 | n | n |
| **ADD course number to courseNumbers** | 1 | n | n |
| **IF temp is less than 2** | 1 | n | n |
| **PRINT error message** | 1 | 1 | 1 |
| **ELSE IF temp is more than 2** | 1 | n | n |
| **IF the prerequisite number is not in courseNumbers** | 1 | n | n |
| **PRINT error message** | 1 | 1 | 1 |
| **CLOSE file** | 1 | n | n |
| **Total Cost** | | | 9n + 5 |
| **Runtime** | | | O(n) |

loadVector(Course course)

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| ADD course to vector courses | 1 | n | n |
| **Total Cost** | | | n |
| **Runtime** | | | O(n) |

loadHashTable

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| CREATE key for the given course | 1 | n | n |
| node = node at the key position | 1 | n | n |
| IF node is null | 1 | 1 | 1 |
| CREATE a new node with course and key | 1 | n | n |
| **ADD new node to HashTable courses** | 1 | n | n |
| **ELSE** | 1 | 1 | 1 |
| **WHILE node is not null** | 1 | n | n |
| **node = next node** | 1 | n | n |
| **CREATE a new node with course and key** | 1 | n | n |
| **Total Cost** | | | 5n + 1 |
| **Runtime** | | | O(n) |

BST

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| IF root is null | 1 | n | n |
| CREATE a new node with course | 1 | n | n |
| ELSE | 1 | 1 | 1 |
| CALL addToTree with root and course | 1 | 0(n) | 0(n) |
| **Total Cost** | | | 0(n) + 2n + 1 |
| **Runtime** | | | O(n) |

Based on the advisor’s requirements I think the best data structure to be used here is the Binary Search Tree. When we load courses into our BST we are loading them in order. This makes it easier to print courses alphanumerically rather than continuously sorting our data structure. Another advantage to the BST, is if we hit the end of the tree and don’t find a param course we know it does not exist. The disadvantage with BST comes when we have larger datasets. But in our case, BST is perfect. Another disadvantage is their worst-case time complexity. I believe the advantages outweigh the disadvantages. Therefore, I recommend BST for our data structure. We can see from the Big O analysis that BST has a best-case scenario of O(n).