# CS 300 Pseudocode Document

## Function Signatures

Below are the function signatures that you can fill in to address each of the three program requirements using each of the data structures. The pseudocode for printing course information, if a vector is the data structure, is also given to you below (depicted in bold).

// 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) {

vector<Course> printedCourses

while length of printedCourses is less then length of courses

boolean prerequisitePrinted is true

Boolean printed is false

for each item in courses vector

set prerequisitePrinted to true

if item has no prerequisites

print item

printed = true

add item to printedCourses

else

for each prerequisite

if prerequisite not in printedCourses

Set prerequisitePrinted to false

if prerequisitePrinted is true

Print item

Printed = true

Add item to printedCourses

}

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**

}

Void loadFileIntoVector(path, LinkedList \*list) {

Display “Loading file” with new line

Create tree

Initialize CSV parser to path argument

String errormessage is a blank string

For each row in header

Display data

Try

Create Bid data structure to hold the bid collection

For i=0 to the number of rows in file

If line I at position 0 is null

Throw error “Missing course ID”

If line I at position 1 is null

Throw error “Missing course Name”

Read line i at position 0 and send it to courseId

Read line i at position 1 and send it to courseName

Append courseId, courseName,coursePrerequisits, numberOfPrerequisits to the end of list

Catch error e

Display error message

}

Void loadFileIntoHashTable(HashTable, tree) {

Display “Loading file” with new line

Initialize parser to path variable

Declare header vector with file header

String errormessage is a blank string

For each row in header

Display data

Try

Create Bid data structure to hold the bid collection

For i=0 to the number of rows in file

If line I at position 0 is null

Throw error “Missing course ID”

If line I at position 1 is null

Throw error “Missing course Name”

Read line i at position 0 and send it to courseId

Read line i at position 1 and send it to courseName

Insert Bid into hashtable

Catch error e

Display error message

}

Void loadFileIntoTree(path, tree) {

Display “Loading file” with new line

Create tree

Initialize parser to path variable

Declare header vector with file header

String errormessage is a blank string

For each row in header

Display data

Try

Create Bid data structure to hold the bid collection

For i=0 to the number of rows in file

If line I at position 0 is null

Throw error “Missing course ID”

If line I at position 1 is null

Throw error “Missing course Name”

Read line i at position 0 and send it to courseId

Read line i at position 1 and send it to courseName

Define Boolean counting as True

Set j as 2

While (counting)

If j is not null

If search(line I at position j) is null

Throw error “No Such Course” & (line I

at position j

Read line i at position j and push\_back the

coursePrerequisites

Increment numberOfPrerequisits by 1

Increment j

Else

Set counting to False

Get key for courseId with hash function

Push key, courseId, courseName,coursePrerequisits, numberOfPrerequisits to end of hashtable

Catch error e

Display error message

}

// Menu Pseudocode

Int main(int argc, char argv[]) {

Define arguments - Define Path as path to file

String course

Define BinarySearchTree\* bst

Bst is new BinarySearchTree()

Bid bid

Choice = 0

While choice is not 9

Display “Menu:” end line

Display “1. Load File” end line

Display “2. Print Alphabetized Course List” end line

Display “3. Print Course info” end line

Display “9. Exit” end line

Switch choice

Case 1:

loadFile(Path,bst)

Case 2:

InOrder()

Case 3:

printCourseInfo(course)

}

//Sort Vector

Void hashTableInOrder() {

Sort(vector beginning,vector end)

For each node

printCourseInfo

}

//Sort HashTable

Void hashTableInOrder() {

For each node

printCourseInfo

}

//Sort binarySearchTree

Void binarySearchTreeInOrder() {

treeInOrder(root)

}

void treeInOrder() {

if node is nullptr

treeInOrder(node left)

display node

treeInOrder(node right)

}

// Hashtable pseudocode

int numPrerequisiteCourses(Hashtable<Course> courses) {

declare totalPrequisites as a hashtable

for each prerequisite p in totalPrerequisites

add prerequisites of the hastable

print number of totalPrerequisites

}

void printSampleSchedule(Hashtable<Course> courses) {

for each key in courses hashtable

print key

if value has prerequisites

loop through prerequisites

print each prequisite

}

Void printCourseListTree(Tree<Course> courses) {

printTreeInOrder(root)

}

Void printTreeInOrder(root){

If node is not nullptr

Traverse left

Output course information for node

Traverse right

}

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

for each course in hashtable

if course is the same coursenumber

print out course info

for each perquisite in the hashtable at the course

print the prerequisite info

}

// Tree pseudocode

int numPrerequisiteCourses(Tree<Course> courses) {

   set totalPrerequisites to the left and right child  
   for each prerequisite in totalPrerequisites  
        the left and right Nodes need to be added to totalPrerequisites  
   print out the number of totalPrerequisites

}

void printSampleSchedule(Tree<Course> courses) {

while length of printedCourses is less then length of courses

Node = root

While (leftnode is not null or leftnode is not in printedCourses

And rightnode is not null or rightnode not in printedCourses)

If leftnode is not in printed courses and leftnode is not null

Traverse left

Else If rightnode is not in printed courses and rightnode is not null

Traverse right

Else If (leftnode is in printedCourses and rightnode is in printedCourses) or (leftnode is null and rightnode is in printedCourses) or (rightnode is null and leftnode is in printedCourses)

print Node

add Node to printedCourses

if root is not null and ((leftnode is in printedCourses and rightnode is in printedCourses) or (leftnode is null and rightnode is in printedCourses) or (rightnode is null and leftnode is in printedCourses)

print root

}

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

for each course node

If course is the same as courseNumber

Print course name and information

If course has a left node

Print left prerequisite information

If course has a right node

Print right prerequisite information

Else if course is greater

Traverse left

Else if course is less

Traverse right

}

## Example Runtime Analysis

When you are ready to begin analyzing the runtime for the data structures that you have created pseudocode for, use the chart below to support your work. This example is for printing course information when using the vector data structure. As a reminder, this is the same pairing that was bolded in the pseudocode from the first part of this document.

| **Code**  Void loadFileIntoVector(path, LinkedList \*list) { | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Display “Loading file” with new line | 1 | 1 | 1 |
| Create vector to store bids | 1 | 1 | 1 |
| Initialize CSV parser to path argument | 1 | 1 | 1 |
| String errormessage is a blank string | 1 | 1 | 1 |
| For each row in header | 1 | n | n |
| Display data | 1 | n | n |
| Try | 1 | 1 | 1 |
| Create Bid data structure to hold the bid collection | 1 | 1 | 1 |
| For i=0 to the number of rows in file | 1 | n | n |
| If line I at position 0 is null | 1 | n | N |
| Throw error “Missing course ID” | 1 | 1 | 1 |
| If line I at position 1 is null | 1 | n | N |
| Throw error “Missing course Name” | 1 | 1 | 1 |
| Read line i at position 0 and send it to courseId | 1 | n | N |
| Read line i at position 1 and send it to courseName | 1 | n | N |
| Append courseId, courseName,coursePrerequisits, numberOfPrerequisits to the end of list | 1 | n | N |
| Catch error e | 1 | 1 | 1 |
| Display error message} | 1 | 1 | 1 |
| **Total Cost** |  |  | 14n+17 |
| **Runtime** | | | O(n) |
|  | | |  |

Hashtable:

| **Code**  Void loadFileIntoHashTable(HashTable, tree){ | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Display “Loading file” with new line | 1 | 1 | 1 |
| Initialize parser to path variable | 1 | 1 | 1 |
| Declare header vector with file header | 1 | 1 | 1 |
| String errormessage is a blank string | 1 | 1 | 1 |
| For each row in header | 1 | n | n |
| Display data | 1 | n | n |
| Try | 1 | 1 | 1 |
| Create Bid data structure to hold the bid collection | 1 | 1 | 1 |
| Declare key variable | 1 | 1 | 1 |
| for i=0 to the number of rows in file | 1 | n | n |
| If line I at position 0 is null | 1 | n | n |
| Throw error “Missing course ID” | 1 | 1 | 1 |
| If line I at position 1 is null | 1 | n | n |
| Throw error “Missing course Name” | 1 | 1 | 1 |
| Calculate key | 1 | N | N |
| Based on key select the record in the list where it will be stored with the course data | 1 | N | n |
| Read line i at position 0 and send it to courseId | 1 | n | n |
| Read line i at position 1 and send it to courseName | 1 | n | n |
| Catch error e | 1 | 1 | 1 |
| Display Error Message} | 1 | 1 | 1 |
| **Total Cost** |  |  | 9n+11 |
| **Runtime** |  |  | O(n) |
|  |  |  |  |
|  |  |  |  |
|  | | |  |
|  | | |  |

Binary Tree:

| **Code**  Void loadFileIntoTree(path, tree) { | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Display “Loading file” with new line | 1 | 1 | 1 |
| Create tree | 1 | 1 | 1 |
| Initialize parser to path variable | 1 | 1 | 1 |
| Declare header vector with file header | 1 | 1 | 1 |
| String errormessage is a blank string | 1 | 1 | 1 |
| For each row in header | 1 | 1 | 1 |
| Display data | 1 | n | n |
| Try | 1 | 1 | 1 |
| Create Bid data structure to hold the bid collection | 1 | 1 | 1 |
| For i=0 to the number of rows in file | 1 | n | n |
| If line I at position 0 is null | 1 | n | n |
| Throw error “Missing course ID” | 1 | 1 | 1 |
| If line I at position 1 is null | 1 | n | n |
| Throw error “Missing course Name” | 1 | 1 | 1 |
| Read line i at position 0 and send it to courseId | 1 | n | n |
| Read line i at position 1 and send it to courseName | 1 | n | n |
| Define Boolean counting as True | 1 | n | n |
| Set j as 2 | 1 | N | N |
| While (counting) | 1 | n | N |
| If j is not null | 1 | N | N |
| If search(line I at position j) is null | 1 | N | N |
| Throw error “No Such Course” & (line I at position j | 1 | 1 | 1 |
| Read line i at position j and push\_back the coursePrerequisites | 1 | N | N |
|  |  |  |  |
| Increment numberOfPrerequisits by 1 | 1 | n | 1 |
| Increment j | 1 | n | N |
| Else | 1 | n | N |
| Set counting to False | 1 | 1 | 1 |
| Get key for courseId with hash function | 1 | 1 | 1 |
| Push key, courseId, courseName,coursePrerequisits, numberOfPrerequisits to end of hashtable | 1 | 1 | 1 |
| Catch error e | 1 | 1 | 1 |
| Display error message | 1 | 1 | 1 |
| **Total Cost** |  |  | 14n+17 |
| **Runtime** |  |  | O(n) |
|  | | |  |
|  | | |  |

When completing a Big-O runtime analysis of the algorithm for loading a file and parsing the data into a data structure, I analyzed the Vector, Hashtable, and Tree algorithms. Each data structure has their own advantages and disadvantages. For example, the vector algorithm is simple and easy to implement. That is beneficial for a project with little time to complete. The Tree data structure has the advantage of easily being able to store data relating to parent and child relationships which is exactly what is needed for course requirements. Hashtables also have advantages for easily sorting data, however it is sometimes not any better than a vector depending on how many collisions there are. The vector data structure was 14n+7 because most of the statements were performed during a loop. When completing the analysis of the HashTable algorithm, it was 9n+11. Finally, the Tree algorithm was 14n+17 because there were a lot of statements performed during a loop but then there were also a lot of lines which were not calling for or while loops. The Tree algorithm included a for loop and a while loop in the same program, while the Vector algorithm was just a for loop although it got the same Big-O score as the Tree algorithm and the HashTable algorithm was also just a for loop. My recommendation is the Tree algorithm because it fits the use case of the data structure the best out of the three although it is tied with vector for being the biggest Big-O score. Data with a parent/child relationship is best stored in a data structure that can support the relationships.