# CS 300 Pseudocode Document

## Function Signatures

NOTE: I DID NOT SEE ANY INSTRUCTOR FEEDBACK THAT REQUESTED CHANGES SO THIS IS THE ORIGINAL PSEUDOCODE, UNCHANGED.   
  
THE TIME COMPLEXITY TABLE HAS BEEN UPDATED. WHILE BINARY SEARCH TREES GENERALLY HAVE A TIME COMPLEXITY OF O(LOG(N)) FOR BALANCED TREES, TIME COMPLEXITY IN THE PROGRAM BELOW IS DETERMINED BY THE MOST COSTLY OPERATIONS [TRAVERSING THE ENTIRE TREE AND PRINTING PREREQUISITE COURSE INFORMATION].

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).

#include <iostream>

#include <fstream>

#include <sstream>

#include <vector>

#include <unordered\_map>

USE standard namespace;

struct Course {

DEFINE string courseNumber;

DEFINE string courseTitle;

DEFINE vector<string> prerequisites;

};

struct TreeNode {

DEFINE Course course;

DEFINE TreeNode\* left;

DEFINE TreeNode\* right;

};

class Tree {

private:

DEFINE TreeNode\* root;

public:

Tree() {

SET root = nullptr;

}

void insertNode(Course course) {

TreeNode\* newNode = new TreeNode;

SET newNode->course = course;

SET newNode->left = nullptr;

SET newNode->right = nullptr;

if (root == nullptr) {

SET root = newNode;

} else {

SET TreeNode\* currentNode = root;

SET TreeNode\* parentNode = nullptr;

while (true) {

SET parentNode = currentNode;

if (course.courseNumber < currentNode->course.courseNumber) {

SET currentNode = currentNode->left;

if (currentNode == nullptr) {

SET parentNode->left = newNode;

break;

}

} else {

SET currentNode = currentNode->right;

if (currentNode == nullptr) {

SET parentNode->right = newNode;

break;

}

}

}

}

}

void loadCoursesFromFile(string filePath) {

ifstream file(filePath);

DEFINE string line;

if (file.is\_open()) {

while (getline(file, line)) {

SET Course course = parseCourseData(line);

CALL insertNode(course);

}

CALL file.close();

OUTPUT "Courses loaded successfully!";

} else {

OUTPUT "File not found. Please provide a valid file path.";

}

}

Course parseCourseData(string line) {

CALL istringstream iss(line);

DEFINE vector<string> tokens;

DEFINEstring token;

while (getline(iss, token, ',')) {

CALL tokens.push\_back(token);

}

if (tokens.size() < 2) {

THROW invalid\_argument("Invalid line format: At least two parameters are required.");

}

SET string courseNumber = tokens[0];

SET string courseTitle = tokens[1];

DEFINE vector<string> prerequisites;

for (size\_t i = 2; i < tokens.size(); i++) {

SET string prerequisiteCourseNumber = tokens[i];

if (!courseExists(prerequisiteCourseNumber)) {

THROW invalid\_argument("Invalid prerequisite: Prerequisite course not found.");

}

CALL prerequisites.push\_back(prerequisiteCourseNumber);

}

RETURN {courseNumber, courseTitle, prerequisites};

}

bool courseExists(string courseNumber) {

RETURN CALL findCourse(root, courseNumber);

}

bool findCourse(TreeNode\* currentNode, string courseNumber) {

if (currentNode == nullptr) {

RETURN false;

}

if (currentNode->course.courseNumber == courseNumber) {

RETURN true;

}

SET bool foundInLeft = findCourse(currentNode->left, courseNumber);

SET bool foundInRight = findCourse(currentNode->right, courseNumber);

RETURN foundInLeft or foundInRight;

}

void printCourseInformation(TreeNode\* currentNode, string courseNumber) {

if (currentNode == nullptr) {

RETURN VOID;

}

if (currentNode->course.courseNumber == courseNumber) {

OUTPUT "Course Number: " << currentNode->course.courseNumberl;

OUTPUT << "Course Title: " << currentNode->course.courseTitlel;

if (!currentNode->course.prerequisites.empty()) {

OUTPUT "Prerequisites: "l;

for (const string& prerequisite : currentNode->course.prerequisites) {

OUTPUT << "- " << VAR prerequisite;

}

}

RETURN VOID;

}

CALL printCourseInformation(currentNode->left, courseNumber);

CALL printCourseInformation(currentNode->right, courseNumber);

}

void printSampleSchedule(TreeNode\* currentNode) {

if (currentNode == nullptr) {

RETURN VOID;

}

OUTPUT "Course Name: " << currentNode->course.courseTitle;

if (currentNode->left != nullptr) {

OUTPUT << "Prerequisite: " << currentNode->left->course.courseNumber;

}

if (currentNode->right != nullptr) {

OUTPUT << "Prerequisite: " << currentNode->right->course.courseNumber;

}

CALL printSampleSchedule(currentNode->left);

CALL printSampleSchedule(currentNode->right);

}

int numPrerequisiteCourses(TreeNode\* currentNode) {

if (currentNode == nullptr) {

RETURN 0;

}

SET int totalPrerequisites = 0;

if (currentNode->left != nullptr) {

SET totalPrerequisites++;

SET totalPrerequisites += numPrerequisiteCourses(currentNode->left);

}

if (currentNode->right != nullptr) {

SET totalPrerequisites++;

SET totalPrerequisites += numPrerequisiteCourses(currentNode->right);

}

RETURN totalPrerequisites;

}

void printCourseInfoForAllCourses() {

CALL printSampleSchedule(root);

}

void printCourseInformation(string courseNumber) {

CALL printCourseInformation(root, courseNumber);

}

int getTotalPrerequisites(string courseNumber) {

SET TreeNode\* currentNode = root;

while (currentNode != nullptr) {

if (currentNode->course.courseNumber == courseNumber) {

RETURN numPrerequisiteCourses(currentNode);

} else if (courseNumber < currentNode->course.courseNumber) {

SET currentNode = currentNode->left;

} else {

SET currentNode = currentNode->right;

}

}

RETURN 0;

}

};

class ABCUniversity {

private:

SET Tree courseTree;

public:

ABCUniversity() {

ITITIALIZE ABCU constructor

}

void loadCourseData(string filePath) {

CALL courseTree.loadCoursesFromFile(filePath);

}

void printCourseInformation(string courseNumber) {

CALL courseTree.printCourseInformation(courseNumber);

}

void printTotalPrerequisites(string courseNumber) {

SET int totalPrerequisites = courseTree.getTotalPrerequisites(courseNumber);

OUTPUT << "Total prerequisites for " << courseNumber << ": " << totalPrerequisites;

}

void printSampleSchedule() {

CALL courseTree.printCourseInfoForAllCourses();

}

};

// MAIN METHOD

int main() {

INITIALIZE ABCUniversity university;

SET string filePath = "course\_information.txt";

CALL university.loadCourseData(filePath);

// Example usage:

CALL university.printCourseInformation("CS101");

CALL university.printTotalPrerequisites("CS101");

CALLuniversity.printSampleSchedule();

RETURN 0;

}

## 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** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 1 | n | n |
| **if the course is the same as courseNumber** | 1 | 1 | 1 |
| **print out the course information** | 1 | 1 | 1 |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 1 | n | O(n) |
| **Total Cost** | | | 3n + 2 |
| **Runtime** | | | O(n) |