# 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).

// Read File pseudocode

void readFile(File f) {

**while (not end of file) {**

**Course C = new Course();**

**error = false;**

**line[] = split line on ‘,’;**

**if( line length < 2) {**

**error = true;**

**}**

**if( not error ) {**

**set code to line[0];**

**set title to line[1];**

**if( line length > 2 ) {**

**for(int i = 2; i<line length; i++){**

**append line[i] to course prereqs;**

**}**

**}**

**Insert C in datatype given;**

**}**

**}**

}

// Inserting in Hashmap pseudocode

void insertInHashMap(Course C) {

**hash\_id = generate a hash key using the course code;**

**if(hash\_id not in use){**

**create a node with C as the course and hash\_id as the key**

**insert node in the hashtable**

**} else {**

**If(node at that hash\_id is UINT\_MAX) {**

**Update node at hash\_id to the new one;**

**} else {**

**While(next node from hash\_id is not null){**

**Append the new node to the end;**

**}**

**}**

**}**

}

// 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**

**return totalPrerequisites length**

}

void printSampleSchedule(Vector<Course> courses) {

**for each course in courses**

**print course information**

**for each prerequisite of course**

**print prerequisite course information indented**

}

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 sortCourses(Vector<Course> courses){

**current node = course.head.next;**

**while (current node not null) {**

**next = current.next;**

**searchNode = current.prev**

**while(searchNode not null and searchNode.courseNumb > current.courseNumb){**

**searchNode = searchNode.prev;**

**}**

**remove currentNode;**

**if(searchNode is null){**

**current.prev = null;**

**coursesPrepend(currentNode);**

**} else {**

**Course insert after search node current node;**

**}**

**Current node = next node;**

**}**

}

void printCourse(Vector<Course> courses) {

**current node = courses.head;**

**while (current node not null) {**

**print current node information;**

**current node = current node’s next;**

**}**

}

// Hashtable pseudocode

int numPrerequisiteCourses(Hashtable<Course> courses, Course C) {

**totalPrereq = prereqs of course C;**

**num of totalPrereq = 0;**

**for each P in totalPrereq**

**add 1 to num of totalPrereq;**

**add length of prereqs of P to num of totalPrereq**

**print num of totalPrereq**

**return num of totalPrereq**

}

void printSampleSchedule(Hashtable<Course> courses) {

**for all key, value pair in courses**

**print key course name**

**if value has prerequisits**

**for each P in prerequisites**

**print P**

}

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

**generate a hash\_id from the string courseNumber given;**

**get the first item in the list of the key hash\_id**

**if( item course number == courseNumber given) {**

**print course\_id, name;**

**for each P in course prereqs {**

**print P**

**}**

**} else {**

**For each course O in the list of key = hash\_id**

**If( o id == courseNumber given ) {**

**print course\_id, name;**

**for each P in course prereqs {**

**print P;**

**break;**

**}**

**}**

**}**

}

// Tree pseudocode

int numPrerequisiteCourses(Tree<Course> courses, Course c) {

**totalPrereq = left and right of Course c;**

**for each course if totalPrereq**

**add the left and the right child if exist to the totalPrereq;**

**return totalPrereq size;**

}

void printSampleSchedule(Tree<Course> courses) {

**while course in courses != null {**

**print course information**

**if(course.left not null){**

**print course.left information**

**}**

**If(course.right not null){**

**Print course.right information**

**}**

**}**

}

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

**while course in courses is not null {**

**if(course.id == courseNumber){**

**print course information**

**if(course.left not null){**

**print course.left information**

**}**

**If course.right not null {**

**Print course.right information**

**}**

**}**

**}**

}

void printCourse(Course course) {

**if (course is null) {**

**return;**

**}**

**printCourse(course left)**

**print course information**

**printCourse(course right)**

}

int printMenu(){

**print ‘1 - Load Data Structure’;**

**print ‘2 - Course List’**

**print ‘3 - Course Information’**

**print ‘9 - Exit’**

**input choice;**

**switch choice :**

**case 1:**

**call loadData function**

**return 1;**

**break;**

**case 2:**

**call printSampleSchedule function**

**return 1;**

**break;**

**case 3:**

**call printCourseInformation function**

**return 1;**

**break;**

**case 9:**

**print ‘GoodBye’**

**return 0;**

**break;**

**default:**

**print ‘Invalid Choice’**

**print 1;**

**break;**

}

**Vector**

**Runtime Analysis for printCourseInformation**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 1 | n | n |
| **if the course is the same as courseNumber** | 1 | n | n |
| **print out the course information** | 1 | 1 | 1 |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 1 | n | n |
| **Total Cost** | | | 4n + 1 |
| **Runtime** | | | O(n) |

**Runtime Analysis for printSample Schedule**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for each course in courses** | 1 | n | n |
| **print course information** | 1 | n | n |
| **for each prerequisite of course** | 1 | n | n |
| **print prerequisite course information indented** | 1 | n | n |
| **Total Cost** | | | 4n |
| **Runtime** | | | O(n) |

**Runtime Analysis for numPrerequisiteCourses**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **totalPrerequisites = prerequisites of course c** | 1 | 1 | 1 |
| **for each prerequisite p in totalPrerequisites** | 1 | n | n |
| **add prerequisites of p to totalPrerequisites** | 1 | n | n |
| **print number of totalPrerequisites** | 1 | 1 | 1 |
| **Total Cost** | | | 2n + 2 |
| **Runtime** | | | O(n) |

**Hash Table**

**Runtime Analysis for printCourseInformation**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **totalPrereq = prereqs of course C;** | 1 | 1 | 1 |
| **num of totalPrereq = 0;** | 1 | 1 | 1 |
| **for each P in totalPrereq** | 1 | n | n |
| **add 1 to num of totalPrereq;** | 1 | n | n |
| **add length of prereqs of P to num of totalPrereq** | 1 | n | n |
| **print num of totalPrereq** | 1 | 1 | 1 |
| **Total Cost** | | | 3n + 3 |
| **Runtime** | | | O(n) |

**Runtime Analysis for printSample Schedule**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all key, value pair in courses** | 1 | n | n |
| **print key course name** | 1 | n | n |
| **if value has prerequisits** | 1 | n | n |
| **for each P in prerequisites** | 1 |  |  |
| **print P** | 1 |  |  |
| **Total Cost** | | |  |
| **Runtime** | | | O() |

**Runtime Analysis for numPrerequisiteCourses**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **generate a hash\_id from the string courseNumber given** | 1 | 1 | 1 |
| **get the first item in the list of the key hash\_id** | 1 | 1 | n |
| **if( item course number == courseNumber given) {** | 1 | 1 | n |
| **print course\_id, name;** | 1 | 1 | 1 |
| **for each P in course prereqs {** | 1 | N | N |
| **print P** | 1 | 1 | 1 |
| **else {** | 1 | 1 | 1 |
| **For each course O in the list of key = hash\_id** | 1 | n | N |
| **If( o id == courseNumber given** | 1 | n | N |
| **print course\_id, name;** | 1 | N | N |
| **for each P in course prereqs {** | 1 |  |  |
| **print P;** | 1 |  |  |
| **Total Cost** | | |  |
| **Runtime** | | | O() |

**Binary Tree**

**Runtime Analysis for printCourseInformation**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **totalPrereq = left and right of Course c;** | 1 | 1 | 1 |
| **for each course if totalPrereq** | 1 | n | n |
| **add the left and the right child if exist to the totalPrereq;** | 1 | n | n |
| **Total Cost** | | | 2n + 1 |
| **Runtime** | | | O(n) |

**Runtime Analysis for printSample Schedule**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **while course in courses != null** | 1 | n | n |
| **print course information** | 1 | n | n |
| **if(course.left not null){** | 1 | n | n |
| **print course.left information** | 1 | n | n |
| **If(course.right not null)** | 1 | n | n |
| **Print course.right information1** | 1 | n | n |
| **Total Cost** | | | 6n |
| **Runtime** | | | O( |

**Runtime Analysis for numPrerequisiteCourses**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **while course in courses != null** | 1 | n | n |
| **if(course.id == courseNumber)** | 1 | n | n |
| **print course information** | 1 | n | n |
| **if(course.left not null){** | 1 | n | n |
| **print course.left information** | 1 | n | n |
| **If(course.right not null)** | 1 | n | n |
| **Print course.right information1** | 1 | n | n |
| **Total Cost** | | | 7n |
| **Runtime** | | | O( |

**Runtime Analysis for printMenu**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **print ‘1 - Load Data Structure’;** | 1 | 1 | 1 |
| **print ‘2 - Course List’** | 1 | 1 | 1 |
| **print ‘3 – Course information’** | 1 | 1 | 1 |
| **print ‘9 - Exit’** | 1 | 1 | 1 |
| **Input choice** | 1 | 1 | 1 |
| **switch choice :** | 1 | 1 | 1 |
| **case 1:**  **call loadData function**  **return 1;**  **break;** | 4 | 1 | 1 |
| **case 2:**  **call printSampleSchedule function**  **return 1;**  **break;** | 4 | 1 | 1 |
| **case 3:**  **call printCourseInformation function**  **return 1;**  **break;** | 4 | 1 | 1 |
| **case 9:**  **print ‘GoodBye’**  **return 0;**  **break;** | 4 | 1 | 1 |
| **default:**  **print ‘Invalid Choice’**  **print 1;**  **break;** | 4 | 1 | 1 |
| **Total Cost** | | | 11 |
| **Runtime** | | | O |

**Evaluation**

**The program will be fed a csv file, a file reader will read the file line by line and extract the information needed to be input. If we chose a specific data structure, this data structure will have a function that inputs all the courses coming from the file into the data structure. We can go further and make only one function that inputs data into any data structure by giving the data structure as an argument and branching cases for each one.**

**Once the file is read, each line will be translated to an object, this object will have fields like title, number, prerequisite that hold all the information needed for the course. The object is then fed to a node which hold information about the node in the data structure picked. For example the list can have next, previous and data. The tree will have left, right and data etc.**

**Here is the side to side comparison of each data structure we evaluated above.**

|  |  |  |
| --- | --- | --- |
| **Data structure** | **Advantages** | **Disadvantages** |
| **Vectors** | * **Dynamic sizing** * **Holds multiple objects** * **Ability to remove elements from it** | * **Memory consumption** |
| **Hash Tables** | * **Better synchronization** * **Faster searching** * **Constant time for CRUD operations** | * **Collisions** * **No null values** * **Large data will have collisions** |
| **Binary Trees** | * **Very fast search** * **Simple and easy to understand** * **Easy to store data** * **Efficient traversing from parent to child** * **Hierarchical structure** | * **Many null pointers** * **Access operation is slower** * **Dependency on the height of the tree** * **Deletion is complicated** |

**Figure 2.1 – Table of comparison**

**Comparing the three data structures and taking into consideration the problem we must solve; I would recommend using the Binary Tree data structure.**

* **The binary tree will have the lowest total cost between them all**
* **It also provides a programmatical map of the schedule which paves a way to take these courses**
* **It will hold each course in a sorted manner and its’ prerequisites below it**