# CS 300 Pseudocode Document

**1.**

// Vector pseudocode

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

OpenRead file named CourseList (not exactly sure what the file is named at this time)

IF file did not open correctly

      PRINT error message stating it did not open properly

END IF

Create Vector, save as Courses

FOR each line in the file

Separate lines in file with a comma

IF there are GREATER THAN 2 parameters per line

Add to prerequisites

ELSE

PRINT an error message

END FOR

}

void printSampleSchedule(Vector<Course> courses) {

Create Hashtable, save table as courses

Initialize the table to being empty

FOR each line in courseInfo

Create object with course number as first parameter, save as courseNum

Create object with course title as second parameter, save as courseTitle

If any parameters exist after the first two, they’re prerequisites.

Using courseNum as a key identifier, save object into Hashtable

END FOR

}

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

FOR each course

If courseNum EQUALS course

PRINT courseNum, courseName,

FOR each prerequisites

PRINT prerequisite information

END FOR

END IF

END FOR

}

// Hashtable pseudocode

int numPrerequisiteCourses(Hashtable<Course> courses) {

OpenRead file named CourseList (still not exactly sure what the file is named at this time)

IF file did not open correctly

      PRINT error message stating it did not open properly

END IF

Create array, save as Courses

FOR each line in the CourseList file

Separate lines in file with a comma

IF there are at least 2 parameters per line

Add to courses

ELSE

PRINT an error message

END IF ELSE

END FOR

}

void printSampleSchedule(Hashtable<Course> courses) {

Create Hashtable, save table as courseInfo

Initialize the table to being empty

FOR each line in Courses

Create object with course number as first parameter, save as courseNum

Create object with course title as second parameter, save as courseTitle

IF there are 2 or more parameters

Create object with course prerequisites as the third parameter, save as coursePreqs

END IF

Using courseNum as a key identifier, save object into Hashtable

END FOR

}

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

FOR each course

If courseNum EQUALS course

PRINT courseNum, courseName,

FOR each prerequisites

PRINT each prerequisite for that CourseNum and courseName

END FOR

END IF

END FOR

}

// Tree pseudocode

int numPrerequisiteCourses(Tree<Course> courses) {

Open read file named courseInformation

IF file did not open correctly

PRINT error message stating file did not open

END IF

Create Vector, save as courses

FOR each line in courseInformation

Separate each line in file with comma

First separated value is the course number, save as courseNumber

Second is the course title, save as courseTitle

Third separated value(s) if any, are prerequisites, save as coursePreReq

IF parameters IS GREATER THAN OR EQUAL to 2

Add to courses

IF third parameter exists

Add to coursePreReq

END IF

ELSE

PRINT ERROR MESSAGE

END ELSE

END FOR

}

void printSampleSchedule(Tree<Course> courses) {

Define structure of Binary Search Tree

Create bid structure for tree node

Initialize the tree to be empty by Root EQUALS null

FOR each line in file

IF root EQUALS null

Root EQUALS New node

ELSE

Call addNode function, which will add node to where it belongs

(**addNode function**

IF current course number is smaller than Root

IF left node EQUALS null

course number becomes left child

END IF

ELSE

Recurse down left child node

END ELSE

END IF ELSE

END IF

ELSE

IF right child node EQUALS null

node becomes the right child

END IF

ELSE

Recurse down right node

END ELSE

END IF ELSE

END IF ELSE)

END ELSE

}

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

FOR each node in the tree

If courseNumber EQUALS course number

PRINT CourseNumber, courseName

FOR each preReq

PRINT CoursePreReq

END FOR

END IF

END FOR

}

**2.**

PRINT "1 : load data" to console

PRINT "2 : Print list of courses" to console

PRINT "3 : Print a specific course" to console  
PRINT  "4 : Exit" to console

Prompt for user input, save as input

**3.**

WHILE Input DOES NOT EQUAL 4

IF input EQUALS 1

      IF current courseNumber IS LESS THAN courseNumber

            current course number gets inserted before the compared courseNumber

      END IF

      ELSE-IF current courseNumber IS GREATER THAN compared courseNumber

            current courseNumber is to be inserted after courseNumber

      END ELSE-IF

PRINT sorted courseNumbers

END WHILE

**4.**

## Example Runtime Analysis

| **Code for a vector structure** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| OpenRead file named CourseList (not exactly sure what the file is named at this time) | 1 | 1 | 1 |
| IF file did not open correctly | 1 | N | N |
| PRINT error message stating it did not open properly | 1 | 1 | 1 |
| END IF | 1 | 1 | 1 |
| Create Vector, save as Courses | 1 | 1 | 1 |
| FOR each line in the file | 1 | N | N |
| Separate lines in file with a comma | 1 | N | N |
| IF there are at greater than 2 parameters per line | 1 | N | N |
| Add to courses | 1 | N | N |
| PRINT an error message | 1 | 1 | 1 |
| Create Hashtable, save table as courses | 1 | 1 | 1 |
| Initialize the table to being empty | 1 | N | N |
| FOR each line in courseInfo | 1 | N | N |
| Create object with course number as first parameter, save as courseNum | 1 | 1 | 1 |
| Create object with course title as second parameter, save as courseTitle | 1 | 1 | 1 |
| IF any parameters exist after the first two, they’re prerequisites. | 1 | N | N |
| Using courseNum as a key identifier, save object into Hashtable | 1 | N | N |
| FOR each course | 1 | N | N |
| IF courseNum EQUALS course | 1 | N | N |
| PRINT courseNum, courseName | 1 | 1 | 1 |
| FOR each prerequisites | 1 | N | N |
| PRINT prerequisite information | 1 | 1 | 1 |
| O(1)+O(n)+O(1)+O(1)+O(1)+O(n)+O(n)+O(n)+O(n)+O(1)+O(1)+O(n)+O(n)+O(1)+O(1)+O(n)+O(n)+O(n)+O(n)+O(1)+O(n)+O(1) | | |  |
| **Total Cost** | | | 12n + 11 |
| **Runtime** | | | O(n) |

| **Code for a Hash table structure** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| OpenRead file named CourseList (not exactly sure what the file is named at this time) | 1 | 1 | 1 |
| IF file did not open correctly | 1 | N | N |
| PRINT error message stating it did not open properly | 1 | 1 | 1 |
| END IF | 1 | 1 | 1 |
| Create array, save as Courses | 1 | 1 | 1 |
| FOR each line in the file | 1 | N | N |
| Separate lines in file with a comma | 1 | N | N |
| IF there are at greater than 2 parameters per line | 1 | N | N |
| Add to courses | 1 | N | N |
| PRINT an error message | 1 | 1 | 1 |
| Create Hashtable, save table as courseInfo | 1 | 1 | 1 |
| Initialize the table to being empty | 1 | N | N |
| FOR each line in courses | 1 | N | N |
| Create object with course number as first parameter, save as courseNum | 1 | 1 | 1 |
| Create object with course title as second parameter, save as courseTitle | 1 | 1 | 1 |
| IF any parameters exist after the first two, they’re prerequisites. | 1 | N | N |
| Using courseNum as a key identifier, save object into Hashtable | 1 | N | N |
| FOR each course | 1 | N | N |
| IF courseNum EQUALS course | 1 | N | N |
| PRINT courseNum, courseName | 1 | 1 | 1 |
| FOR each prerequisites | 1 | N | N |
| PRINT prerequisite information | 1 | 1 | 1 |
| END FOR | 1 | 1 | 1 |
| O(1)+O(n)+O(1)+O(1)+O(1)+O(n)+O(n)+O(n)+O(n)+O(1)+O(1)+O(n)+O(n)+O(1)+O(1)+O(n)+O(n)+O(n)+O(n)+O(1)+O(n)+O(1) +O(1) | | |  |
| **Total Cost** | | | 12n + 11 |
| **Runtime** | | | O(n) |

| **Code for a Tree structure** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| OpenRead file named CourseInformation | 1 | 1 | 1 |
| IF file did not open correctly | 1 | N | N |
| PRINT error message stating it did not open properly | 1 | 1 | 1 |
| END IF | 1 | 1 | 1 |
| Create array, save as Courses | 1 | 1 | 1 |
| FOR each line in courseInformation | 1 | N | N |
| Separate lines in file with a comma | 1 | 1 | 1 |
| First separated value is the course number, save as courseNumber | 1 | 1 | 1 |
| Second is the course title, save as courseTitle | 1 | 1 | 1 |
| Third separated value(s) if any, are prerequisites, save as coursePreReq | 1 | 1 | 1 |
| IF parameters IS GREATER THAN OR EQUAL to 2 | 1 | N | N |
| Add to courses | 1 | 1 | 1 |
| IF second parameter exists | 1 | N | N |
| Add to coursePreReq | 1 | 1 | 1 |
| ELSE | 1 | 1 | 1 |
| PRINT ERROR MESSAGE | 1 | 1 | 1 |
| END FOR | 1 | 1 | 1 |
| Define structure of Binary Search Tree | 1 | 1 | 1 |
| Create bid structure for tree node | 1 | 1 | 1 |
| Initialize the tree to be empty by Root EQUALS null | 1 | 1 | 1 |
| FOR each line in file | 1 | N | N |
| IF root EQUALS null | 1 | N | N |
| Root EQUALS New node | 1 | 1 | 1 |
| ELSE | 1 | 1 | 1 |
| Call addNode function, which will add node to where it belongs | 1 | NLogN | NlogN |
| END ELSE | 1 | 1 | 1 |
| FOR each node in the tree | 1 | N | N |
| If courseNumber EQUALS course number | 1 | N | N |
| PRINT CourseNumber, courseName | 1 | 1 | 1 |
| FOR each preReq | 1 | N | N |
| PRINT CoursePreReq | 1 | 1 | 1 |
| O(1)+O(n)+O(1)+O(1)+O(1)+O(n)+O(1)+O(1)+O(1)+O(1)+O(n)+O(1)+O(n)+O(1)+O(1)+O(1)+O(1)+O(1)+O(1)+O(1)+O(n)+O(n)+O(1)+O(1)+O(nLogn)+O(n)+O(n)+O(1)+O(n)+O(1)+O(1)+O(n) | | |  |
| **Total Cost** | | | 12n + 21 |
| **Runtime** | | | O(n)+ (nLogn) |

**5.**

**Evaluation**

      An advantage of a vector data structure is that it is compact. This enables fast processing when the program is run. Another is that the graphics are more accurate which makes the application more appealing to the user. It also enables to manage data easily. The last advantage I want to discuss is that it does not take up too much disk space. Therefore, it does not require much storage space to be implemented. A disadvantage of a vector data structure is that the technology is expensive when heading towards more integrated systems. Another drawback is that it uses memory excessively. Therefore, it will affect the memory usage and potentially result in longer run times.

      An advantage of a hash table data structure is that it uses cache mapping in order to look up the data fast. Therefore, it is more efficient when comparing it to a binary tree. It relies on constant time for searches, and results in a high-speed manipulation of the program as well as data retrieval or fast lookups. Another advantage is that updating hash tables are insertion and deletion are easier. Meaning that instead of updating pointers, you only need to update one index for each operation.  A  disadvantage is that they're inefficient when there are multiple collisions undergoing in the table. Also, hash tables do not allow null values, so in the case of class prerequisites, some classes may not have prerequisites. Therefore, the space in the hash table is null. Another disadvantage is that they do not maintain an order of elements that are stored in the table. Thus, making it hard to casll the print function to print class info in order.

      An advantage to a binary search tree is that space is used efficiently. Meaning that most nodes only require 2 pointers to each sub tree. Another advantage is that since binary search trees are organized in a particular way, it allows for easy traversing of the tree to be easier to implement while programming. This also allows for faster searching when looking for something in the tree. Since the tree is already in order, it eliminates half of the nodes at each level. Also, data retrieval is fast since the tree is already sorted in order whether it be numerically or alphabetical. A disadvantage of a binary search tree is that when the tree is not balanced, it can cause the run time to be affected such as the search for a node is slower and resulting in being less efficient.  Another disadvantage is when it comes to duplicated data. Since binary search trees do not support duplicated data values, it makes it very challenging to be sorted. Lastly, When working with insertion and deletion of nodes, it can be cumbersome if the tree is imbalanced causing the runtime to be slower.

**6.**

**Recommendation**

      The data structure I plan on using in my code would be the hash table. I am going to implement this into my code because the course numbers will all lead to an unbalanced tree. When mapping out the course information, the tree only has a right subtree that leads to another right sub tree and so on. When looking at the Big O notation chart and seeing the run time analysis. It appears to me that the hash table with the data provided is by far the best worst-case scenario run time analysis at O(n). As comparing it to the other structures, I believe that the hash table will be the best case because when they need to reevaluate the courses for prerequisites in the future, it will be easier to maintain and update along the way.