# CS 300 Project 1

Amauri Hopewell

Below is the pseudocode for a data structure to hold course information, and 3 different approaches to reading in course information from a file and storing the properly formatted data: using a vector, using a hash table, and using a binary search tree. All approaches will use the same Struct for storing each Course (ie, each valid line from the file), and the binary search tree will also use a Node Struct. All approaches use a 2-pass technique for reading in the file. During the first pass, I will create a list of all courses in the file, using a vector. During the 2nd pass, I will add the ones whose prerequisites are in that list to the vector, hash table, or binary search tree, depending on the approach chosen, and discard the rest.

# Code used in all approaches:

**//The following Struct is used for all approaches**

Struct Course {

**String number**

**String title**

**Vector<String> prereqNumbers**

**bool comp()(Course a, Course b)//to allow sorting later**

**{**

**return results of standard string compare on .number fields**

**}**

}

**//The following function is used for all approaches**

//It uses a vector regardless because it is used for

//the 2-pass approach for loading the file

bool courseExists(Vector<String> courseNumbers, String testNumber) {

**for all courseNumbers**

**if the testNumber is the same as courseNumber**

**return True**

**return false //only activates if a match is not found**

}

//Important note for later runtime analysis: This is O(N)

//Because in the worst case, it would have to sort through

//N-1 courses to find the prerequisite for the Nth course

**// The following function is used for all approaches**

// It uses a vector because it is used for

// the 2-pass approach for loading the file

bool validLineFormat (vector<String> courseNumbers, String line) {

**int count = line.split[“,”].size**

**if count == 2 //2 parameters from 1 comma**

**return true**

**else if count > 2 //if there are 3 parameters**

**Vector<String> params = line.split[“,”] //split by commas**

**for i in 2:params.size //loop through courses in 3rd param**

**if (courseExists(courseNumbers, params[i]))**

**do nothing**

**else**

**return false**

**return true // if no invalid courses detected in for loop**

**else if count < 2 //for wrong parameter numbers**

**return false**

}

//Important note for later runtime analysis: This is O(N)

//Because it calls courseExists, which is O(N), and the rest are

// constant lines

# Vector approach:

/\*\*

**\*Vector approach**

\*\*/

void searchCourse(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**

}

Vector<Course> readFile(File file){

**Vector<String> courseNumbers //for formatting check**

**Vector<Course> courses //for final storage**

**ifstream.open(file)**

**for line in file**

**courseNumbers.append(line.parameter[0])//first pass**

**for line in file //2nd pass to check for validity of prereqs**

**if (validLineFormat(courseNumbers, line))**

**Course newCourse=Course(line.param[0], line.param[1])**

**if line has 3 params**

**for number in line.param[2]**

**newCourse.prereqNumbers.append(number)**

**courses.append(newCourse)**

**else**

**discard line**

**return courses**

}

**//Runtime complexity analysis estimate:**

**/\* first 3 lines execute once, first for loop executes N times with up \* to 1 line, for N+3. Then second for loop executes N times, with up \*to 3+K+X lines for K course prerequisites and X lines in \*validLineFormat(courseNumbers, line). But validLineFormat is O(N), as \*noted earlier, so it can be simplified and treated as taking N lines \*The total is N+3 + N(3+K+N) = N^2+(3+K)N+3, which is O(N^2)**

**\*This is a worst case estimate, and it won’t normally take that long**

**\*/**

**//Memory analysis estimate:**

**/\* The main object here will be a Vector<Course> objects, which will \*take N\*(A+B+K\*A) bytes, where A is the maximum length of a course \*number, B is the maximum length of a course title, and K is the \*maximum number of prerequisites. Another main object will be the \*Vector<String> courses, which will be maximum memory complexity N\*A.**

**\*This is O(N) memory complexity.**

**\*/**

void printCourses(Vector<Course> courses){

**if (courses != nullptr)**

**//use earlier comparison operator from struct**

**//to allow use of stableSort for easy sorting**

**stableSort(courses.end(), courses.begin(),Course.compare())**

**for i in 0:courses.length()**

**print course[i].number + “: “**

**print course[i].title**

**print newline**

**if courses[i].prereqs != nullptr**

**print “prerequisites: “**

**for j in 0:courses[i].prereqNumbers.length()**

**print courses[i].prereqNumbers[j]**

**else**

**print “no perquisites”**

**print newline**

}

/\*\*

**\*end Vector approach**

\*\*/

# Run Time and Memory Analysis Summary and Recommendation:

|  |  |
| --- | --- |
|  | **Vector** |
| **Loading Courses From File (Time)** | O(N^2) due to validating while loading |
| **Loading Courses From File (Memory)** | O(N) |