# William Kendall

Project One

**Vector Pseudocode**

// Vector pseudocode

int readFile(Vector<Course> courses, string filepath) {

open filestream to filepath

if filestream is not open

print error message

return 1

else

print that filepath has been opened

initialize empty string vector strings

while filestream has not reached end-of-file

read each line into strings

while not at end of strings

char = first char of current string

location = 0 // tracks the field you are iterating through

new course object

while not at end of current string

if char is a comma

location = location + 1

continue

if location == 0

append char to courseNumber field

if location == 1

append char to courseName field

if location == 2

append char to prereq1 field

if location == 3

append char to prereq2 field

if location == 4

append char to prereq3 field

if location == 5

append char to prereq4 field

increment char index

if location < 1

print current object has less than 2 parameters

return 2

increment current string index

while not at the end of course objects

if current course prereqs do not match another courseNumber

print the prereq that is an error

return 3

}

Course() {

string courseNumber

string courseName

string prereq1

string prereq2

string prereq3

string prereq4

}

Course (num, name, prereq1 = “”, prereq2 = “”, prereq3 = “”, prereq4 = “”) {

courseNumber = num

courseName = name

prereq1 = prereq1

prereq2 = prereq2

prereq3 = prereq3

prereq4 = prereq4

}

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

totalPrerequisites = prerequisites of course c

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

return number of totalPrerequisites

}

void printSampleSchedule(Vector<Course> courses) {

for all courses

print course information

for each prerequisite of the course

print prerequisite course information

}

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 printCSAlphabetic(Vector<Course> courses) {

for all courses

min = outer loop index

for all courses from outer loop index + 1 to last course

if course < course at min

min = inner loop index

tmp = outer loop course

outer loop course = course at min

course at min = tmp

for all courses

print course information

for each prerequisite of the course

print the prerequisite course information

}

int displayMenu() {

print load structure as option 1

print print course list as option 2

print print course as option 3

print exit as option 4

int userInput = userInput

if userInput > 4 or < 1

print error message

return -1

else

return userInput

}

**Hashtable Pseudocode**

// Hashtable pseudocode

int readFile(Hashtable<Course> courses, string filepath) {

open filestream to filepath

if filestream is not open

print error message

return 1

else

print that filepath has been opened

initialize empty string vector strings

while filestream has not reached end-of-file

read each line into strings

while not at end of strings

char = first char of current string

location = 0 // tracks the field you are iterating through

new course object

while not at end of current string

if char is a comma

location = location + 1

continue

if location == 0

append char to courseNumber field

if location == 1

append char to courseName field

if location == 2

append char to prereq1 field

if location == 3

append char to prereq2 field

if location == 4

append char to prereq3 field

if location == 5

append char to prereq4 field

increment char index

if location < 1

print current object has less than 2 parameters

return 2

increment current string index

while not at the end of course objects

if current course prereqs do not match another courseNumber

print the prereq that is an error

return 3

chars = last three chars of courseNumber

key = last three chars cast to int % some int

copy course to hashtable using key as index

delete course from vector of courses

}

Course() {

string courseNumber

string courseName

string prereq1

string prereq2

string prereq3

string prereq4

}

Course (num, name, prereq1 = “”, prereq2 = “”, prereq3 = “”, prereq4 = “”) {

courseNumber = num

courseName = name

prereq1 = prereq1

prereq2 = prereq2

prereq3 = prereq3

prereq4 = prereq4

}

int numPrerequisiteCourses(Hashtable<Course> courses) {

totalPrerequisites = prerequisites of course c

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

return number of totalPrerequisites

}

void printSampleSchedule(Hashtable<Course> courses) {

for all courses

print course information

}

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

hash = last three chars of courseNumber cast to int % some int

for all courses

if last three chars of current courseNumber hashed == hash

print out current course information

for each prerequisite of the course

print prerequisite course information

}

void printCSAlphabetic(Hashtable<Course> courses) {

for all courses

min = outer loop index

for all courses from outer loop index + 1 to last course

if course < course at min

min = inner loop index

tmp = outer loop course

outer loop course = course at min

course at min = tmp

for all courses

print course information

for each prerequisite of the course

print the prerequisite course information

}

int displayMenu() {

print load structure as option 1

print print course list as option 2

print print course as option 3

print exit as option 4

int userInput = userInput

if userInput > 4 or < 1

print error message

return -1

else

return userInput

}

**Tree Pseudocode**

// Tree pseudocode

int readFile(Tree<Course> courses, string filepath) {

open filestream to filepath

if filestream is not open

print error message

return 1

else

print that filepath has been opened

initialize empty string vector strings

while filestream has not reached end-of-file

read each line into strings

while not at end of strings

char = first char of current string

location = 0 // tracks the field you are iterating through

new course object

while not at end of current string

if char is a comma

location = location + 1

continue

if location == 0

append char to courseNumber field

if location == 1

append char to courseName field

if location == 2

append char to prereq1 field

if location == 3

append char to prereq2 field

if location == 4

append char to prereq3 field

if location == 5

append char to prereq4 field

increment char index

if location < 1

print current object has less than 2 parameters

return 2

increment current string index

while not at the end of course objects

if current course prereqs do not match another courseNumber

print the prereq that is an error

return 3

chars = last three chars of courseNumber

num = chars cast to int

if root node is null

copy course to root node

else if num < last three chars of current node

call addNode with left child as new argument

else if num > last three chars of current node

call addNode with right child as new argument

else

copy course to current node

delete course from vector of courses

}

Course() {

string courseNumber

string courseName

string prereq1

string prereq2

string prereq3

string prereq4

}

Course (num, name, prereq1 = “”, prereq2 = “”, prereq3 = “”, prereq4 = “”) {

courseNumber = num

courseName = name

prereq1 = prereq1

prereq2 = prereq2

prereq3 = prereq3

prereq4 = prereq4

}

int numPrerequisiteCourses(Tree<Course> courses) {

totalPrerequisites = prerequisites of course c

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

return number of totalPrerequisites

}

void printSampleSchedule(Tree<Course> courses) {

for all courses

print course information

}

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

hash = last three chars of courseNumber cast to int % some int

for all courses

if last three chars of current courseNumber hashed == hash

print out current course information

for each prerequisite of the course

print prerequisite course information

}

void printCSAlphabetic(Tree<Course> courses) {

for all courses // employ in-order traversal

print course information

for each prerequisite of the course

print prerequisite course information

}

int displayMenu() {

print load structure as option 1

print print course list as option 2

print print course as option 3

print exit as option 4

int userInput = userInput

if userInput > 4 or < 1

print error message

return -1

else

return userInput

}

## Runtime Analyses

**Read File and Create Course Objects (Vector)**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| open filestream to filepath | 1 | 1 | 1 |
| if filestream is not open | 1 | 1 | 1 |
| print error message | 1 | 1 | 1 |
| return 1 | 1 | 1 | 1 |
| else | 1 | 1 | 1 |
| print that filepath has been opened | 1 | 1 | 1 |
| initialize empty string vector strings | 1 | n | n |
| while filestream has not reached end-of-file | 1 | n | n |
| read each line into strings | 1 | n | n |
| while not at end of strings | 1 | n | n |
| char = first char of current string | 1 | n | n |
| location = 0 | 1 | n | n |
| new course object | 1 | n | n |
| while not at end of current string | 1 | n | n |
| if char is a comma | 1 | n | n |
| location = location + 1 | 1 | n | n |
| continue | 1 | n | n |
| if location == 0 | 1 | n | n |
| append char to courseNumber field | 1 | n | n |
| if location == 1 | 1 | n | n |
| append char to courseName field | 1 | n | n |
| if location == 2 | 1 | n | n |
| append char to prereq1 field | 1 | n | n |
| if location == 3 | 1 | n | n |
| append char to prereq2 field |  | n | n |
| if location == 4 | 1 | n | n |
| append char to prereq3 field | 1 | n | n |
| if location == 5 | 1 | n | n |
| append char to prereq4 field | 1 | n | n |
| increment char index | 1 | n | n |
| if location < 1 | 1 | n | n |
| print current object has less than 2 parameters | 1 | n | n |
| return 2 | 1 | 1 | 1 |
| increment current string index | 1 | n | n |
| while not at the end of course objects | 1 | n | n |
| if current course prereqs do not match another courseNumber | 1 | n | 1 |
| print the prereq that is an error | 1 | n | n |
| return 3 | 1 | 1 | 1 |
| **Total Cost** | | | 29n + 9 |
| **Runtime** | | | O(n) |

**Read File and Create Course Objects (Hashtable)**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| open filestream to filepath | 1 | 1 | 1 |
| if filestream is not open | 1 | 1 | 1 |
| print error message | 1 | 1 | 1 |
| return 1 | 1 | 1 | 1 |
| else | 1 | 1 | 1 |
| print that filepath has been opened | 1 | 1 | 1 |
| initialize empty string vector strings | 1 | n | n |
| while filestream has not reached end-of-file | 1 | n | n |
| read each line into strings | 1 | n | n |
| while not at end of strings | 1 | n | n |
| char = first char of current string | 1 | n | n |
| location = 0 | 1 | n | n |
| new course object | 1 | n | n |
| while not at end of current string | 1 | n | n |
| if char is a comma | 1 | n | n |
| location = location + 1 | 1 | n | n |
| continue | 1 | n | n |
| if location == 0 | 1 | n | n |
| append char to courseNumber field | 1 | n | n |
| if location == 1 | 1 | n | n |
| append char to courseName field | 1 | n | n |
| if location == 2 | 1 | n | n |
| append char to prereq1 field | 1 | n | n |
| if location == 3 | 1 | n | n |
| append char to prereq2 field |  | n | n |
| if location == 4 | 1 | n | n |
| append char to prereq3 field | 1 | n | n |
| if location == 5 | 1 | n | n |
| append char to prereq4 field | 1 | n | n |
| increment char index | 1 | n | n |
| if location < 1 | 1 | n | n |
| print current object has less than 2 parameters | 1 | n | n |
| return 2 | 1 | 1 | 1 |
| increment current string index | 1 | n | n |
| while not at the end of course objects | 1 | n | n |
| if current course prereqs do not match another courseNumber | 1 | n | 1 |
| print the prereq that is an error | 1 | n | n |
| return 3 | 1 | 1 | 1 |
| chars = last three chars of courseNumber | 1 | n | n |
| key = last three chars cast to int % some int | 1 | n | n |
| copy course to hashtable using key as index | 1 | n | n |
| delete course from vector of courses | 1 | n | n |
| **Total Cost** | | | 33n + 9 |
| **Runtime** | | | O(n) |

**Read File and Create Course Objects (Tree)**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| open filestream to filepath | 1 | 1 | 1 |
| if filestream is not open | 1 | 1 | 1 |
| print error message | 1 | 1 | 1 |
| return 1 | 1 | 1 | 1 |
| else | 1 | 1 | 1 |
| print that filepath has been opened | 1 | 1 | 1 |
| initialize empty string vector strings | 1 | n | n |
| while filestream has not reached end-of-file | 1 | n | n |
| read each line into strings | 1 | n | n |
| while not at end of strings | 1 | n | n |
| char = first char of current string | 1 | n | n |
| location = 0 | 1 | n | n |
| new course object | 1 | n | n |
| while not at end of current string | 1 | n | n |
| if char is a comma | 1 | n | n |
| location = location + 1 | 1 | n | n |
| continue | 1 | n | n |
| if location == 0 | 1 | n | n |
| append char to courseNumber field | 1 | n | n |
| if location == 1 | 1 | n | n |
| append char to courseName field | 1 | n | n |
| if location == 2 | 1 | n | n |
| append char to prereq1 field | 1 | n | n |
| if location == 3 | 1 | n | n |
| append char to prereq2 field |  | n | n |
| if location == 4 | 1 | n | n |
| append char to prereq3 field | 1 | n | n |
| if location == 5 | 1 | n | n |
| append char to prereq4 field | 1 | n | n |
| increment char index | 1 | n | n |
| if location < 1 | 1 | n | n |
| print current object has less than 2 parameters | 1 | n | n |
| return 2 | 1 | 1 | 1 |
| increment current string index | 1 | n | n |
| while not at the end of course objects | 1 | n | n |
| if current course prereqs do not match another courseNumber | 1 | n | 1 |
| print the prereq that is an error | 1 | n | n |
| return 3 | 1 | 1 | 1 |
| chars = last three chars of courseNumber | 1 | n | n |
| num = chars cast to int | 1 | n | n |
| if root node is null | 1 | n | n |
| copy course to root node | 1 | n | n |
| else if num < last three chars of current node | 1 | n | n |
| call addNode with left child as new argument | 10n | n | 10n2 |
| else if num > last three chars of current node | 1 | n | n |
| call addNode with right child as new argument | 10n | n | 10n2 |
| else | 1 | n | n |
| copy course to current node | 1 | n | n |
| delete course from vector of courses | 1 | n | n |
| **Total Cost** | | | 20n2 + 38n + 9 |
| **Runtime** | | | O(n2) |

Based on the above analyses, we can see that the vector and hashtable data structures generally offer the best time cost for this application. The tree data structure is limited as it has two places where it potentially calls the addNode function n amount of times, thus increasing the complexity from O(n) to O(n2). As a result, I plan to use the vector data structure in the final project. I chose the vector structure as it offers a balanced time cost based on this Big-O analysis, and I have more experience using and incorporating the vector structure than I do with the hashtable or tree structures.

Will Kendall