Project 1 Hashtables revisited.

The runtime of code will be noted as // followed by Big O notation.

Struct Course

Attributes:

courseNumber: String

title: String

prerequisites: Vector

// accessing attributes is a O(1) operation.

Class HashTable

Attributes:

table: List of Vectors of Courses

size: Integer

// accessing attributes is a O(1) operation

hash(String courseNumber):

Return sum of ASCII values of courseNumber modulo size

O(N) where length is courseNumber

insertCourse(Course course):

index = hash(course.courseNumber) // O(1)

If table[index] is null: // O(1)

table[index] = [course] // O(1)

Else: // O(1)

table[index].append(course) // O(1)

// O(1) + 5 average O(n) + 5 worst case, where n is number of courses due to collisions.

searchCourse(String courseNumber):

index = hash(courseNumber) // O(1)

If table[index] is not null: // O(1)

For course in table[index]: // O(N)

If course.courseNumber == courseNumber: // O(1)

Return course

// O(1) + 3 average O(n) + 3 worst case, where n is number of courses due to collisions.

getAllCourses():

allCourses = List of Courses // O(1)

For bucket in table: // O(1)

If bucket is not null: // O(1)

For course in bucket: // O(1)

allCourses.append(course) // O(1)

Return allCourses

// O(n) + 5 average where n is number of courses due to collisions.

readFile(String: filename):

Try: // O(1)

Open file filename for reading// O(1)

lines = file.readlines() // O(1)

file.close() // O(1)

Return lines

Catch IOError: // O(1)

Print "Error: File could not be opened." // O(1)

Return List of String

parse(List of String: lines): // O(1)

courses = List of Courses: // O(1)

courseNumbers = Set of String // O(1)

For line in lines: // O(n)

tokens = line.split(',') // O(1)

If len(tokens) < 2: // O(1)

Print Error // O(1)

Return (False, []) // O(1)

courseNumber = tokens[0].strip() // O(1)

title = tokens[1].strip()// O(1)

prerequisites = [token.strip() for token in tokens[2:]] // O(1)

course = Course(courseNumber, title, prerequisites) // O(1)

courses.append(course) // O(1)

courseNumbers.add(courseNumber) // O(1)

For course in courses: // O(n)

For prerequisite in course.prerequisites: // O(n)

If prerequisite not in courseNumber: // O(1)

Print Error // O(1)

Return (False, [])

Return (True, courses) // O(n) + 6 average since reading file line by line if no error.

buildCourses(Vector courses):

hashTable = HashTable(size=courses.length) // O(1)

For course in courses: // O(n)

hashTable.insertCourse(course) // O(1)

Return hashTable

//O(n) + 2 average run time where n is number of courses.

Void printCourses(HashTable hashTable):

courses = hashTable.getAllCourses() O(n)

courses.sort(key=lambda course: course.courseNumber) // O(n log n)

For course in courses:// O(n)

Print course.title, course.courseNumber, course.prerequisites //O(1)

Main():

filename = "courses.txt" //O(1)

lines = readFile(filename) O(n)

If lines: //O(1)

valid, courses = parse(lines) //O(n)

If valid: //O(1)

hashTable = buildCourses(courses) O(n)

printCourses(hashTable) //O(1)

Else: //O(1)

Print "File validation failed." //O(1)

O(n log n) + 6 with use of helper functions

Void mainMenu()

While True: // O(n)

Print "1: Load file data into data structure" // O(1)

Print "2: Print an alphanumerically ordered list of all the courses in the Computer Science department" // O(1)

Print "3: Print the course title and the prerequisites for any individual course" // O(1)

Print "9: Exit the program" // O(1)

Input choice // O(1)

Switch choice: // O(1)

Case "1": // O(1)

buildCourses() // O(n)

Case "2": // O(1)

printCourses() // O(n)

Case "3": // O(1)

courseNumber = Input "Enter course number:" // O(n)

course = hashTable.search(courseNumber) // O(n)

If course is not null: // O(1)

Print course.title, course.courseNumber, course.prerequisites // O(1)

Else: // O(1)

Print "Course not found."

Case "9":

Print "Exiting the program." // O(1)

Break

Default: // O(1)

Print "Invalid choice. Please enter a number from the menu." // O(1)

// O(n) + 16 average depending on selection.

Total average is O(n log n) complexity.