**CS300**

**Aidan Farhi**

**12/06/24**

**Binary Search Tree - Pseudocode**

/\* Course object definition. \*/

Struct Course {

String courseNumber

String courseName

Vector<String> prerequisites

}

/\* Gets a field from a comma separated String. \*/

String getField(string line) {

get the index of the first comma in the line

get a substring starting at position zero until the index of the first comma

return the substring

}

/\* Checks whether the given line is valid. \*/

Boolean isValidLine(String line) {

initialize a Vector of Integers to hold indexes of commas

// collect all indexes of comma occurences in the line

for every character in the line:

if the character is a comma:

add the current index to the Vector of indexes

// there must be at least one comma for the line to be valid.

// one comma means there are potentially two fields.

if the size of the Vector of indexes is greater than or equal to one:

for every index in the Vector of indexes:

// this handles the case when the last field of the line is empty.

if the index is equal to the length of the line minus one:

return False

// this handles the case where a field is empty within the line.

if the character in the line at position index plus one is a comma:

return False

return True

else:

return False

}

/\* Loads course numbers from a file into a Set of strings. \*/

void loadCourseNumbers(Set<String> courseNumbers, String filename) {

open the file

while the file has a next line:

read the next line into a String

call isValidLine(line)

if the line is valid:

call getField(line) and store the resulting String

add the String to the courseNumbers Set

}

/\* Loads lines of course data from a CSV file into a BinarySearchTree of Course objects. \*/

void loadCourses(BinarySearchTree<Course> courses, String fileName) {

open the file

initialize a Set of Strings to hold course numbers called courseNumbers

call loadCourseNumbers(courseNumbers, fileName)

// iterate through all the lines of the file

while the file has a next line:

read the next line into a String called line

call isValidLine(line)

if the line is valid:

initialize an empty Course object

// first get the course number

call getField(line) and store the resulting String

store the String in Course.courseNumber of the Course object

slice off the front of the line up to the first comma, including the comma

// next get the course name

call getField(line) and store the resulting String

store the String in Course.courseName of the Course object

slice off the front of the line up to the first comma, including the comma

// add the prerequisites if there are any

while the length of the line is greater than zero:

call getField(line) and store the resulting course number String

append the String to Course object's Vector of prerequisites

slice off the front of the line up to the first comma, including the comma

// validate that the prerequisites are existing courses

for every course number in the Course.prerequisites Vector:

if the course number does not exist in the course numbers Set:

// this effectively skips the loading of this Course object

// into the BinarySearchTree because the prerequisites check failed

continue to the next iteration of the while loop

// this only happens if the prerequisites check is successfull

insert the Course object into the courses BinarySearchTree

}

/\*\*

\* Searches a BinarySearchTree of Course objects for a given course number.

\* If the course is found, the course information is printed, else a course

\* not found message is printed.

\*/

void searchCourse(BinarySearchTree<Course> courses, String courseNumberToFind) {

store a reference to the root node of the tree in a variable

while the node reference is not NULL:

if the node's course number is less than the courseNumberToFind:

update the node reference to point to node.left

else if the node's course number is greater than the courseNumberToFind:

update the node reference to point to node.right

else:

if the node's course number matches the courseNumberToFind:

print out the course information

return

// this will only print if a matching course was not found

print that the course was not found

}

/\* Prints all CS Courses in alphanumeric order \*/

void printCSCourses(BinarySearchTree<Course> courses) {

// perform an in-order traversal of the BinarySearchTree

for course in courses.InOrder():

if course starts with "CSCI":

print out the course information

}

/\*\*

\* Prints out a menu, prompts user for input, and handles it.

\* Returns true unless user wants to exit the program.

\*/

Boolean handleMenu(BinarySearchTree<Course> courses, String fileName) {

print menu options

get menu choice from user input

if menu choice equals 1:

call loadCourses(courses, fileName)

else if menu choice equals 2:

call printCSCourses(courses)

else if menu choice equals 3:

get courseNumber from user input

call searchCourse(courses, courseNumber)

else if menu choice equals 9:

print goodbye message

return false

else:

print menu option not valid

return true

}

/\* The one and only main function \*/

void main() {

initialize a Boolean variable called keepRunning and set to true

initialize a BinarySearchTree object called courses

initialize a String variable called fileName and store the name of the file

while keepRunning is true:

call handleMenu(courses, fileName)

store return value in keepRunning

}

**Binary Search Tree – Runtime Analysis**

void loadCourses(BinarySearchTree<Course> courses, String fileName)

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **open the file** | 1 | 1 | 1 |
| **initialize a Set of Strings to hold course numbers** | 1 | 1 | 1 |
| **call loadCourseNumbers(courseNumbers, fileName)** | n | 1 | n |
| **while the file has a next line:** | 1 | n | n |
| **read the next line into a String called line** | 1 | 1 | 1 |
| **if the line is valid:** | 1 | 1 | 1 |
| **initialize an empty Course object** | 1 | 1 | 1 |
| **call getField(line) and store the resulting String** | 1 | 1 | 1 |
| **store the String in Course.courseNumber of the Course object** | 1 | 1 | 1 |
| **slice off the front of the line up to the first comma, including the comma** | 1 | 1 | 1 |
| **while the length of the line is greater than zero:** | 1 | n | n |
| **call getField(line) and store the resulting String** | 1 | 1 | 1 |
| **append the String to Course object’s Vector of prerequisites** | 1 | 1 | 1 |
| **slice off the front of the line up to the first comma, including the comma** | 1 | 1 | 1 |
| **for every course number in the Course.prerequisites Vector:** | 1 | n | n |
| **if the course number does not exist in the course numbers Set:** | 1 | 1 | 1 |
| **continue to the next iteration of the while loop** | 1 | 1 | 1 |
| **insert the Course object into the courses BinarySearchTree** | n | 1 | n |
| **Total Cost** | | | 5n + 13 |
| **Runtime** | | | O(n) |

**Vector - Pseudocode**

/\* Course object definition. \*/

Struct Course {

String courseNumber

String courseName

Vector<String> prerequisites

}

/\* Get's a field from a comma separated String. \*/

String getField(string line) {

get the index of the first comma in the line

get a substring starting at position zero until the index of the first comma

return the substring

}

/\* Checks whether the given line is valid. \*/

Boolean isValidLine(String line) {

initialize a Vector of Integers to hold indexes of commas

// collect all indexes of comma occurences in the line

for every character in the line:

if the character is a comma:

add the current index to the Vector of indexes

// there must be at least one comma for the line to be valid.

// one comma means there are potentially two fields.

if the size of the Vector of indexes is greater than or equal to one:

for every index in the Vector of indexes:

// this handles the case when the last field of the line is empty.

if the index is equal to the length of the line minus one:

return False

// this handles the case where a field is empty within the line.

if the character in the line at position index plus one is a comma:

return False

return True

else:

return False

}

/\* Loads course numbers from a file into a Set of strings. \*/

void loadCourseNumbers(Set<String> courseNumbers, String filename) {

open the file

while the file has a next line:

read the next line into a String

call isValidLine(line)

if the line is valid:

call getField(line) and store the resulting String

add the String to the courseNumbers Set

}

/\* Loads lines of course data from a CSV file into a Vector of Course objects. \*/

void loadCourses(Vector<Course> courses, String fileName) {

open the file

initialize a Set of Strings to hold course numbers called courseNumbers

call loadCourseNumbers(courseNumbers, fileName)

// iterate through all the lines of the file

while the file has a next line:

read the next line into a String called line

call isValidLine(line)

if the line is valid:

initialize an empty Course object

// first get the course number

call getField(line) and store the resulting String

store the String in Course.courseNumber of the Course object

slice off the front of the line up to the first comma, including the comma

// next get the course name

call getField(line) and store the resulting String

store the String in Course.courseName of the Course object

slice off the front of the line up to the first comma, including the comma

// add the prerequisites if there are any

while the length of the line is greater than zero:

call getField(line) and store the resulting course number String

append the String to Course object's Vector of prerequisites

slice off the front of the line up to the first comma, including the comma

// validate that the prerequisites are existing courses

for every course number in the Course.prerequisites Vector:

// this effectively skips the loading of this Course object

// into the Vector because the prerequisites check failed

if the course number does not exist in the course numbers Set:

continue to the next iteration of the while loop

// this only happens if the prerequisites check is successfull

add the Course object to the courses Vector

}

/\*\*

\* Searches a Vector of Course objects for a given course number.

\* If the course is found, the course information is printed, else a course

\* not found message is printed.

\*/

void searchCourse(Vector<Course> courses, String courseNumberToFind) {

for every Course object in the courses Vector:

if the Course.courseNumber equals the courseNumberToFind

print Course.courseNumber

print Course.courseName

for every prerequisite in the Course.prerequisites Vector:

print the prerequisite

}

/\* Prints all CS Courses in alphanumeric order \*/

void printCSCourses(Vector<Course> courses) {

initialize a Vector of Strings to hold the course numbers called courseNumbers

for course in courses:

add course.courseNumber to courseNumbers Vector

// use language specific method or function to sort the Vector

courseNumbers.Sort()

for courseNumber in courseNumbers:

print courseNumber

}

/\*\*

\* Prints out a menu, prompts user for input, and handles it.

\* Returns true unless user wants to exit the program.

\*/

Boolean handleMenu(Vector<Course> courses, String fileName) {

print menu options

get menu choice from user input

if menu choice equals 1:

call loadCourses(courses, fileName)

else if menu choice equals 2:

call printCSCourses(courses)

else if menu choice equals 3:

get courseNumber from user input

call searchCourse(courses, courseNumber)

else if menu choice equals 9:

print goodbye message

return false

else:

print menu option not valid

return true

}

/\* The one and only main function \*/

void main() {

initialize a Boolean variable called keepRunning and set to true

initialize a Vector object called courses

initialize a String variable called fileName and store the name of the file

while keepRunning is true:

call handleMenu(courses, fileName)

store return value in keepRunning

}

**Vector – Runtime Analysis**

void loadCourses(Vector<Course> courses, String fileName)

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **open the file** | 1 | 1 | 1 |
| **initialize a Set of Strings to hold course numbers** | 1 | 1 | 1 |
| **call loadCourseNumbers(courseNumbers, fileName)** | n | 1 | n |
| **while the file has a next line:** | 1 | n | n |
| **read the next line into a String called line** | 1 | 1 | 1 |
| **if the line is valid:** | 1 | 1 | 1 |
| **initialize an empty Course object** | 1 | 1 | 1 |
| **call getField(line) and store the resulting String** | 1 | 1 | 1 |
| **store the String in Course.courseNumber of the Course object** | 1 | 1 | 1 |
| **slice off the front of the line up to the first comma, including the comma** | 1 | 1 | 1 |
| **while the length of the line is greater than zero:** | 1 | n | n |
| **call getField(line) and store the resulting String** | 1 | 1 | 1 |
| **append the String to Course object’s Vector of prerequisites** | 1 | 1 | 1 |
| **slice off the front of the line up to the first comma, including the comma** | 1 | 1 | 1 |
| **for every course number in the Course.prerequisites Vector:** | 1 | n | n |
| **if the course number does not exist in the course numbers Set:** | 1 | 1 | 1 |
| **continue to the next iteration of the while loop** | 1 | 1 | 1 |
| **add the Course object to the courses Vector** | 1 | 1 | 1 |
| **Total Cost** | | | 4n + 14 |
| **Runtime** | | | O(n) |

**Hash Table - Pseudocode**

/\* Course object definition. \*/

Struct Course {

String courseNumber

String courseName

Vector<String> prerequisites

}

/\* Get's a field from a comma separated String. \*/

String getField(string line) {

get the index of the first comma in the line

get a substring starting at position zero until the index of the first comma

return the substring

}

/\* Checks whether the given line is valid. \*/

Boolean isValidLine(String line) {

initialize a Vector of Integers to hold indexes of commas

// collect all indexes of comma occurences in the line

for every character in the line:

if the character is a comma:

add the current index to the Vector of indexes

// there must be at least one comma for the line to be valid.

// one comma means there are potentially two fields.

if the size of the Vector of indexes is greater than or equal to one:

for every index in the Vector of indexes:

// this handles the case when the last field of the line is empty.

if the index is equal to the length of the line minus one:

return False

// this handles the case where a field is empty within the line.

if the character in the line at position index plus one is a comma:

return False

return True

else:

return False

}

/\* Loads course numbers from a file into a Set of strings. \*/

void loadCourseNumbers(Set<String> courseNumbers, String filename) {

open the file

while the file has a next line:

read the next line into a String

call isValidLine(line)

if the line is valid:

call getField(line) and store the resulting String

add the String to the courseNumbers Set

}

/\* Loads lines of course data from a CSV file into a HashTable of Course objects. \*/

void loadCourses(HashTable<String, Course> courses, String fileName) {

open the file

initialize a Set of Strings to hold course numbers called courseNumbers

call loadCourseNumbers(courseNumbers, fileName)

// iterate through all the lines of the file

while the file has a next line:

read the next line into a String called line

call isValidLine(line)

if the line is valid:

initialize an empty Course object

// first get the course number

call getField(line) and store the resulting String

store the String in Course.courseNumber of the Course object

slice off the front of the line up to the first comma, including the comma

// next get the course name

call getField(line) and store the resulting String

store the String in Course.courseName of the Course object

slice off the front of the line up to the first comma, including the comma

// add the prerequisites if there are any

while the length of the line is greater than zero:

call getField(line) and store the resulting course number String

append the String to Course object's Vector of prerequisites

slice off the front of the line up to the first comma, including the comma

// validate that the prerequisites are existing courses

for every course number in the Course.prerequisites Vector:

// this effectively skips the loading of this Course object

// into the HashTable because the prerequisites check failed

if the course number does not exist in the course numbers Set:

continue to the next iteration of the while loop

// this only happens if the prerequisites check is successfull

add the Course object to the courses HashTable using Course.courseNumber as the key and the Course object as the value

}

/\*\*

\* Searches a HashTable of Course objects for a given course number.

\* If the course is found, the course information is printed, else a course

\* not found message is printed.

\*/

void searchCourse(HashTable<String, Course> courses, String courseNumberToFind) {

check if the courses HashTable has a key that matches courseNumberToFind

if there is a matching key:

get the Course object associated with the courseNumberToFind key

print Course.courseNumber

print Course.courseName

for every prerequisite in the Course.prerequisites Vector:

print the prerequisite

else:

print out message that the course number was not found

}

/\* Prints all CS Courses in alphanumeric order \*/

void printCSCourses(HashTable<String, Course> courses courses) {

initialize a Vector of Strings to hold the course numbers called courseNumbers

// use language specific method or function to get a list of keys

for key in courses.Keys():

add key to courseNumbers Vector

// use language specific method or function to sort the Vector

courseNumbers.Sort()

for courseNumber in courseNumbers:

print courseNumber

}

/\*\*

\* Prints out a menu, prompts user for input, and handles it.

\* Returns true unless user wants to exit the program.

\*/

Boolean handleMenu(HashTable<String, Course> courses, String fileName) {

print menu options

get menu choice from user input

if menu choice equals 1:

call loadCourses(courses, fileName)

else if menu choice equals 2:

call printCSCourses(courses)

else if menu choice equals 3:

get courseNumber from user input

call searchCourse(courses, courseNumber)

else if menu choice equals 9:

print goodbye message

return false

else:

print menu option not valid

return true

}

/\* The one and only main function \*/

void main() {

initialize a Boolean variable called keepRunning and set to true

initialize a HashTable object called courses

initialize a String variable called fileName and store the name of the file

while keepRunning is true:

call handleMenu(courses, fileName)

store return value in keepRunning

}

**Hash Table – Runtime Analysis**

void loadCourses(HashTable<String, Course> courses, String fileName)

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **open the file** | 1 | 1 | 1 |
| **initialize a Set of Strings to hold course numbers** | 1 | 1 | 1 |
| **call loadCourseNumbers(courseNumbers, fileName)** | n | 1 | n |
| **while the file has a next line:** | 1 | n | n |
| **read the next line into a String called line** | 1 | 1 | 1 |
| **if the line is valid:** | 1 | 1 | 1 |
| **initialize an empty Course object** | 1 | 1 | 1 |
| **call getField(line) and store the resulting String** | 1 | 1 | 1 |
| **store the String in Course.courseNumber of the Course object** | 1 | 1 | 1 |
| **slice off the front of the line up to the first comma, including the comma** | 1 | 1 | 1 |
| **while the length of the line is greater than zero:** | 1 | n | n |
| **call getField(line) and store the resulting String** | 1 | 1 | 1 |
| **append the String to Course object’s Vector of prerequisites** | 1 | 1 | 1 |
| **slice off the front of the line up to the first comma, including the comma** | 1 | 1 | 1 |
| **for every course number in the Course.prerequisites Vector:** | 1 | n | n |
| **if the course number does not exist in the course numbers Set:** | 1 | 1 | 1 |
| **continue to the next iteration of the while loop** | 1 | 1 | 1 |
| **add the Course object to the courses HashTable using Course.courseNumber as the key and the Course object as the value** | 1 | 1 | 1 |
| **Total Cost** | | | 4n + 14 |
| **Runtime** | | | O(n) |

**Final Analysis and Recommendation**

Each data structure has its own strengths and weaknesses depending on the operation being performed in the program. The loadCourses and printCSCourses operations have a O(n) worst-case time complexity for each data structure. The searchCourse operation will have different runtimes based on the data structure being used. If it is a hash table, the average time complexity will be O(1) assuming the keys are evenly distributed and the number of prerequisites for the course being searched for is not of size n. If a balanced binary search tree is being used the operation will have a O(log n) time complexity with the same assumption. Performing the same operation using a vector would be O(n) in the worst case.

Based on the analysis of each data structure, I would recommend using a hash table. It has the lowest average time complexity for the search operation and performs similarly to other data structures for loading and printing courses. Additionally, if a removal operation were added to the program, a hash table would have an average runtime complexity of O(1) whereas a binary search tree would take O(log n) time, and a vector would take O(n) in the worst case.