CS-300

Doctor Adamo

Kyunghoon Lee

10 August 2024

**Project 1**

**// Vector – Milestone 1**

// Open and read data from file, parse each line, check for format errors

void loadCourses(String filename) {

assign filename or path

open file using ifstream

if file does not open successfully

return error

while not at end of file,

getline for each line

create stringstream to hold line

create token variable to store each block of course data

create vector of strings to store tokens

while splitting stringstream into tokens by commas,

add each token to end of tokens vector

if size of tokens vector < 2,

return error

continue to next line

if token at index[2] is in tokens vector,

continue to next line

else return error

close file

}

// Create course objects and store in appropriate data structure

Define structure Course

Define members: courseNumber, courseName, prerequisites

Create vector<Course> courses

For each line in the file,

courseNumber = token at index 1

courseName = token at index 0

while not at the end of the line,

add remaining tokens to prerequisites vector

add course to the end of structure Course

// Search data structure for a specific course and print out course information and prerequisites

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

get courseNumber as input

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

else

display “Course not found”

}

// Function to sort courses

sortCourses(courses) {

Sort courses by courseNumber

}

// Menu display

displayMenu() {

while true:

Display the following:

“1. Load Course Data

2. Print All Course Data

3. Print Course Information

9. Exit”

switch menuChoice:

case 1:

call loadCourses(filename)

case 2:

call sortCourses(courses)

for each course in courses

display course.courseNumber

case 3:

prompt user for input, set to courseNumber

call searchCourse(courses, courseNumber)

case 9:

exit program

default:

display “Please choose an option on the menu.”

}

**// Hash Table – Milestone 2**

// Open and read data from file, parse each line, check for format errors

void loadCourses(String filename) {

assign filename or path

open file using ifstream

if file does not open successfully

return error

while not at end of file,

getline for each line

create stringstream to hold line

create token variable to store each block of course data

create vector of strings to store tokens

while splitting stringstream into tokens by commas,

add each token to end of tokens vector

if size of tokens vector < 2,

return error

continue to next line

if token at index[2] is in tokens vector,

continue to next line

else return error

close file

}

// Create course objects and store in appropriate data structure

Define class HashTable

Define structure Course

Define members: courseNumber, courseName, prerequisites

Void insertCourse(hashTable, courseNumber, courseName, prerequisites)

course = new Course

course.courseNumber = courseNumber

course.courseName = courseName

course.prerequisites = prerequisites

HashTable[courseNumber] = course

For each line in the file, call insertCourse

courseNumber = token at index 1

courseName = token at index 0

while not at the end of the line,

add remaining tokens to prerequisites vector

add course to the end of structure Course

// Search data structure for a specific course and print out course information and prerequisites

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

get courseNumber as input

assign input to key

If key is found in HashTable

Display course information

If prerequisites is not empty

Display prerequisites information

Else

Return error: course not found

}

// Transfer HashTable to a vector for sorting

sortingList(HashTable) {

Create vector courseList

For each entry in HashTable

Append each match to courseList

Return courseList

}

// Function to sort courses

sortCourses(courses) {

Sort courses by courseNumber

}

// Menu display

displayMenu() {

while true:

Display the following:

“1. Load Course Data

2. Print All Course Data

3. Print Course Information

9. Exit”

switch menuChoice:

case 1:

call loadCourses(filename)

case 2:

set courses = sortingList(HashTable)

call sortCourses(courses)

for each course in courses

display course.courseNumber

case 3:

prompt user for input, set to courseNumber

call searchCourse(courses, courseNumber)

case 9:

exit program

default:

display “Please choose an option on the menu.”

}

**// Binary Search Tree – Milestone 3**

// Open and read data from file, parse each line, check for format errors

void loadCourses(String filename) {

assign filename or path

open file using ifstream

if file does not open successfully

return error

while not at end of file,

getline for each line

create stringstream to hold line

create token variable to store each block of course data

create vector of strings to store tokens

while splitting stringstream into tokens by commas,

add each token to end of tokens vector

if size of tokens vector < 2,

return error

continue to next line

if token at index[2] is in tokens vector,

continue to next line

else return error

close file

}

// Create course objects and store in appropriate data structure

Define structure Course

Define members: courseNumber, courseName, prerequisites

Define structure TreeNode

Define members: course, Node \*left, Node \*right

Define TreeNode constructor() {

Left equal to nullptr

Right equal to nullptr

}

Initialize TreeNode with a course

Define class BinarySearchTree

Define members: root = nullptr

Define function insertCourse(course) {

If root is nullptr:

root = new node Course

Else:

Call addNode function, pass root and course

}

Define function addNode(root, course) {

If course.courseNumber is less than node.course.courseNumber:

If left node is nullptr:

New node becomes left

Else:

Recursively call addNode() down left side of tree

Else if course.courseNumber is greater than node.course.courseNumber:

If right node is nullptr:

New node becomes right

Else:

Recursively call addNode() down right side of tree

}

// Search data structure for a specific course and print out course information and prerequisites

Void searchCourse(Tree<course> courses, String courseNumber) {

Set current node to courseNumber input

While current is not nullptr:

If courseNumber matches current node:

Output matching course information

If courseNumber is less than current node:

Recursively traverse left side of tree

Output when match is found

Else:

Recursively traverse right side of tree

Output when match is found

Return empty course (if no match is found)

}

// In-order traversal function

Void inOrder(Node\* root)

If root is null

Return empty tree

Call inOrder(root->left) to check left subtree

Display course information and prerequisites

Call inOrder(root->right) to check right subtree

Display course information and prerequisites

// Menu display

displayMenu() {

while true:

Display the following:

“1. Load Course Data

2. Print All Course Data

3. Print Course Information

9. Exit”

switch menuChoice:

case 1:

call loadCourses(filename)

case 2:

call inOrder

for each course in courses

display course.courseNumber

case 3:

prompt user for input, set to courseNumber

call searchCourse(courses, courseNumber)

case 9:

exit program

default:

display “Please choose an option on the menu.”

}

**Runtime Analysis**

**// Vector**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# times executed** | **Total cost** |
| Create vector Courses | 1 | 1 | 1 |
| Assign filename or path and open file | 1 | 1 | 1 |
| If file does not open successfully | 1 | 1 | 1 |
| Return error | 1 | 1 | 1 |
| While not at end of file | 1 | n | n |
| Getline for each line | 1 | n | n |
| Create stringstream to hold line | 1 | n | n |
| Create token variable to store each block of course data | 1 | n | n |
| Create vector of strings to store tokens | 1 | n | n |
| While splitting stringstream into tokens by commas | 1 | n | n |
| Add each token to end of tokens vector | 1 | n | n |
| If size of tokens vector < 2 | 1 | n | n |
| Return error | 1 | n | n |
| If token at index[2] is in tokens vector | 1 | n | n |
| Create course object | 1 | n | n |
| Add course object to end of courses vector | 1 | n | n |
| Close file | 1 | 1 | 1 |
|  |  | **Total Cost** | 12n + 5 |
|  |  | **Runtime** | O(n) |

**// Hash Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# times executed** | **Total cost** |
| Create HashTable | **1** | **1** | **1** |
| Create vector Courses | **1** | **1** | **1** |
| Assign filename or path and open file | **1** | **1** | **1** |
| If file does not open successfully | **1** | **1** | **1** |
| Return error | **1** | **1** | **1** |
| While not at end of file | 1 | n | n |
| Getline for each line | 1 | n | n |
| Create stringstream to hold line | 1 | n | n |
| Create token variable to store each block of course data | 1 | n | n |
| Create vector of strings to store tokens | 1 | n | n |
| While splitting stringstream into tokens by commas | 1 | n | n |
| Add each token to end of tokens vector | 1 | n | n |
| If size of tokens vector < 2 | 1 | n | n |
| Return error | 1 | n | n |
| If token at index[2] is in tokens vector | 1 | n | n |
| Create course object | 1 | n | n |
| Call insertCourse to add object to vector | 5 | n | 5n |
| Close file | 1 | 1 | 1 |
|  |  | **Total Cost** | 17n + 5 |
|  |  | **Runtime** | O(n) |

**// Binary Search Tree**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# times executed** | **Total cost** |
| Create TreeNode | 1 | 1 | 1 |
|  |  |  |  |
| Create vector Courses | 1 | 1 | 1 |
| Assign filename or path and open file | 1 | 1 | 1 |
| If file does not open successfully | 1 | 1 | 1 |
| Return error | 1 | 1 | 1 |
| While not at end of file | 1 | n | n |
| Getline for each line | 1 | n | n |
| Create stringstream to hold line | 1 | n | n |
| Create token variable to store each block of course data | 1 | n | n |
| Create vector of strings to store tokens | 1 | n | n |
| While splitting stringstream into tokens by commas | 1 | n | n |
| Add each token to end of tokens vector | 1 | n | n |
| If size of tokens vector < 2 | 1 | n | n |
| Return error | 1 | n | n |
| If token at index[2] is in tokens vector | 1 | n | n |
| Create course object | 1 | n | n |
| Call addNode to add course | log(n) | n | n \* log(n) |
| If course.courseNumber is less than node.course.courseNumber | 1 | n | n |
| If left node is null, new node becomes left | 1 | n | n |
| Else recursively call addNode() down left side of tree | 1 | n | n |
| Else if course.courseNumber is greater than nde.course.courseNumber | 1 | n | n |
| If right node is null, new node becomes right | 1 | n | n |
| Else recursively call addNode down right side of tree | 1 | n | n |
| Close file | 1 | 1 | 1 |
|  |  | **Total Cost** | 17n + n \* log n + 6 |
|  |  | **Runtime** | O(log n) |

An advantage to the vector data structure in this case is its speed and efficiency, as it had the lowest runtime of 12n + 5. However, a disadvantage to using the vector data structure is that operations such as inserting or deleting elements can be more costly in terms of memory or runtime.

An advantage to using hash tables as a data structure is their ability to be resized as needed, allowing for the use of larger data sets without a major loss in efficiency. On the other hand, a major disadvantage is the additional complexity of implementing chaining and linked lists as a way of handling collisions.

An advantage to binary search trees is the inherent sorting provided by in-order traversal, saving time when it comes to sorting. However, the logarithmic nature of a binary search can result in major loss of efficiency as the tree grows in complexity.

For this project, I would recommend the use of the vector data structure. The data set of courses and their associated information is not overly complex, which would be a good fit for the use of a vector. While insertion or deletion operations may take more time with a vector, I do not think it would need to happen to the point of inefficiency after the initial data entry has been completed.