James Soto

11-29-2022

CS 300 Analysis and Design

**Project 1**

**Vector Pseudocode**

Class Course{

Vector <string> prerequisite;

String courseName;

String courseNumber;

Opening and Reading File:

Vector<Course> courses

Declare courseLines: string

OPENFILE<filename> for Read using ifstream

Check return value of open function

If return is equal to -1

Return the error “File not found”

While not End of File(filename)

READFILE (filename)

Parse each line separated by comma and push to the vector as courseLines

If line size = 2

Create new course variable from the line and push into the vector course

Else

Return invalid data file

Course Objects Structures:

For i = 2; i < line size

If index = 0

Add to new course id

If index = 1

Add to new course name

If index >= 2

If index courseLines size -1 does not equal the course id

return "prerequisites not found'

if element index > 2

pushback to new.course

CLOSEFILE(<filename>)

Print Course Information and Prerequisites:

Declare search Id: string

Declare course found: boolean

get user input

if course id is equal to search Id

course found is equal to true

print course id, course name, prerequisites

else

if course found is equal to false

print course not found

**Hash Table Pseudocode**

Opening and Reading File:

Declare courseLines: string

OPENFILE<filename> for Read using fstream

Check return value of open function

If return is equal to -1

Return the error “File not found”

Else

File is found

While not End of File(filename)

READFILE (filename)

Parse each line separated by comma and push to the vector as courseLines

If line size < 2

Return error invalid data file

If line size = 2

Create new course variable from the line and push into the vector course

Else if line size > 2

If parameter is equal to parameter on line 1

Continue

else

Return invalid data file

CLOSEFILE(<filename>)

Course Objects Structures:

Declare HashTable class

Create constructor

Create Destructor

Create methods and logic for HashTable

Initialize and read variables from file

Create keys for each entry by using modulo operator

While file is open

Store the course information in the hash table bucket

Declare key pointer found: Boolean

Initialize and read variables from file

get user input for search

assign input to key pointer

if key pointer value is found

print course id, course name, prerequisites

for each prerequisite

print the course and prerequisite

else

if key pointer value is not found

print course not found

**Binary Search Tree Pseudocode**

Opening and Reading File:

Initialize Course Structure struct

OPENFILE<filename> for read using fstream

Check return value of open function

If return is equal to -1

Return the error “File not found”

Else

File is found

While not End of File(filename) – loop through file will not EOF

READFILE (filename)

Parse each line separated by the comma

If line size < 2

Return error invalid data

If line size = 2

Add course id, and course name

Else if line size > 2

Add prerequisites until a new line is found

Continue

else

Return Error

CLOSEFILE(<filename>)

Course Objects Structures:

Create Tree and add Nodes

Define the binary tree class

Create a root that points to nullptr

Create insert methods

If the root is nullptr

current Course is root

Else if

If course number is less than the root

Add to the left

If course number is less than the leaf add to the left

If course number is greater than the lead add to the right

Add the course number

Else if

Course number is greater than the root

Add to the right

If course number is less than the leaf add to the left

If course number is greater than the lead add to the right

Add the course number

Return course

Print Course Information and Prerequisites:

While the current root not null

get user input for search

Store the information in vector

Read the course number from user entry

Search the entered course number in the vector

For loop

If course number matches

Print the course information and prerequisites

else

if course number does not match

print course not found

**Menu Pseudocode**

Create integer name for switch statement

Int userInput = 0;

While user input != 4

Print out Menu

Option 1: Load Data Structure

Print message data structure must be loaded before other functions work.

Option 2: Print Course List

Option 3: Print Course

Option 4: Exit

Take in user input

Switch(userInput)

Case 1: Load Data Structure

course = loadCourses(course)

print out program loaded course.size()

break

Case 2: Print Course List

quicksort(course, course.size() -1)

print alphanumerically CourseList

break

Case 3: Print Course

Print message asking user to enter course number to search

Display

CSCI100, Introduction to Computer Science

CSCI101, Introduction to Programming in C++

CSCI200, Data Structures

MATH201 Discrete Mathematics

CSCI300 Introduction to Algorithms

CSCI301 Advanced Programming in C++

CSCI350 Operating Systems

CSCI400 Large Software Development

Cin >> userInput as course number

courseSearch(course)

print course number, course name, prerequisites courses

break

Default:

Print Exit

Break

**Alphanumeric Program Pseudocode**

Sort the course information

void quickSort(vector<Course>& course, int begin, int end) {

int midPointIndex = 0;

if (begin >= end) {

return;

}

midPointIndex = partition(course, begin, end);

quickSort(course, begin, midPointIndex);

quickSort(course, midPointIndex + 1, end);

**Print the sorted list to a display**

quickSort(course, 0, course.size() - 1);

cout << course.size()

CSCI100, Introduction to Computer Science

CSCI101, Introduction to Programming in C++

CSCI200, Data Structures

MATH201 Discrete Mathematics

CSCI300 Introduction to Algorithms

CSCI301 Advanced Programming in C++

CSCI350 Operating Systems

CSCI400 Large Software Development

**Evaluation and Analysis**

**Vector**

| **Vector**  **Read File and Create Courses** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Read file with fstream | 1 | 1 | 1 |
| Get each line from file until none left | 1 | n | n |
| Create Vector course for each line in file | 1 | n | n |
| Get string token for each line item | 1 | 1 | 1 |
| While course contains 2-line items | 1 | n | n |
| If token = 1; set token to course number and increment count | 1 | n | n |
| Else if token = 2; set token to course name and increment count | 1 | n | n |
| Else if token = 3; set token to prerequisite course and increment count | 1 | n | n |
| Else display error in file format | 1 | 1 | 1 |
| Pushback course items | 1 | n | n |
|  |  |  |  |
| **Total Cost** | | | 7n + 3 |
| **Runtime** | | | O(n) |

**HashTable**

| **HashTable**  **Read File and Create Courses** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Read file with fstream | 1 | 1 | 1 |
| Get each line from file until none left | 1 | n | n |
| Create course for each line in file | 1 | n | n |
| Get string token for each line item | 1 | n | n |
| While course contains 2-line items | 1 | n | n |
| If token = 1; set token to course number and increment count | 1 | n | n |
| Else if token = 2; set token to course name and increment count | 1 | n | n |
| Else if token = 3; set token to prerequisite course and increment count | 1 | n | n |
| Else display error in file format | 1 | n | n |
| Pushback course items | 1 | n | n |
| Create key for course to hash | 1 | 1 | 1 |
| Create node to retrieve key | 1 | 1 | 1 |
| If node equals null | 1 | n | n |
| Assign node to key | 1 | n | n |
| Else if node’s key is equal to UINT\_MAX | 1 | n | n |
| Assign node key to key | 1 | n | n |
| Assign old node key to course | 1 | n | n |
| Assign node’s next to null | 1 | n | n |
| Else | 1 | n | n |
| While node’s next != null | 1 | n | n |
| Set node to next node | 1 | n | n |
| Create new node newCourse using course and key | 1 | 1 | 1 |
| Set node’s next to newCourse | 1 | n | n |
|  |  |  |  |
|  |  |  |  |
| **Total Cost** |  |  |  |
| **Runtime** | | | 19n + 4 |
|  | | | O(n) |

**Binary Search Tree**

| **Binary Search Tree**  **Read File and Create Courses** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Read file with fstream | 1 | 1 | 1 |
| Get each line from file until none left | 1 | n | n |
| Create course for each line in file | 1 | n | n |
| Get string token for each line item | 1 | n | n |
| While course contains 2-line items | 1 | n | n |
| If token = 1; set token to course number and increment count | 1 | n | n |
| Else if token = 2; set token to course name and increment count | 1 | n | n |
| Else if token = 3; set token to prerequisite course and increment count | 1 | n | n |
| Else display error in file format | 1 | 1 | 1 |
| Pushback course items | 1 | n | n |
| If root of tree is null, current node becomes root | 1 | 1 | 1 |
| While current node is not null | 1 | n | n |
| If node is less than root node, traverse left | 1 | n | n |
| If current ->left node is null, current node becomes left node | 1 | 1 | 1 |
| Else recursively traverse node’s left sub tree | 1 | n | n |
| Else if current -> right node is null, current node becomes right node | 1 | 1 | 1 |
| Else recursively traverse node’s right sub tree | 1 | n | n |
|  |  |  |  |
| **Total Cost** |  |  |  |
| **Runtime** | | | 12n + 5 |
|  | | | O(n) |

**Advantages/Disadvantages**

**Vector**

Advantages:

1. Insertion at the back is in constant time
2. Vectors can be searched in O(n) time
3. Attributes are combined with objects

Disadvantages

1. Complex data structure
2. Algorithms for manipulative and analysis functions are complex and may be processing intensive.
3. Location of each vertex needs to be stored explicitly.
4. Needs to be sorted to take advantage of the search function

**HashTable**

Advantages:

1. Insert, delete and search operations are very fast and can be done in O(1) time.
2. Hash tables can store large amounts of data.
3. Direct access to data or items

Disadvantages

1. Hash functions tend to produce duplicate keys, which cause problems with storing data values, known as collisions
2. Hash collisions are common
3. Hash tables become inefficient if it contains a collision
4. Hash tables do not allow null values

**Binary Search Tree**

Advantages:

1. Fast and efficient at inserting, deleting, searching, and traversals when the tree is balanced
2. Retrieve items in indexed order
3. Insert and delete in O(n) runtime
4. Used to reflect the relationship between nodes/data

Disadvantages

1. Tree must be balanced
2. Slower when compared to vector
3. If tree is imbalanced it increases the complexity

**Recommendation**

Based on the Big O analysis and going over the results I would recommend the Binary Search Tree. The reason I chose the Binary Search Tree is because it uses traversal which allows for the display of the courses in alphabetical order in a more efficient manner when compared to using a Vector or Hashtable data structure. BST allows for the data to traverse in order as it becomes available thus not requiring any sorting when compared to vectors or Hashtables that require a sorting algorithm to efficiently display the courses in alphanumerical order. Searching a BST involves the comparison of the key values. If the key value is equal to root key then, search successful, if lesser than root key then search the key in the left subtree and if the key is greater than root key then search the key in the right subtree. Overall the binary search trees advantages as a data structure outweigh the use of Vectors or a Hashtable.