Sean Flynn

12/7/23

CS-300 Project One

The data structure that I will be using for my project will be a binary search tree. The reason I am choosing BTS is because by nature a BTS is already sorted. Because one of the requirements is to print the course information in a sorted order, this makes BTS a good choice as it saves time by not needing to sort the data before printing. In addition searching a BTS is still relatively quick. This makes the code easy to write because the tree will be sorted during it's creation, and search through a BTS is a straight forward process.

Hash tables would not be a good choice for this project over a BTS. Hash tables are useful for quickly searching but are not very good for sorting. The data would need to be removed from the hash table, and then sorted for it to be able to print in order. If the information did not need to be sorted then I would choose a hash table as this would make the search function fast and easy. A vector would also need to be sorted in this way, and it would also be slower to search, making it the worst choice for this project. It would be slower in general to search than a has table, and would need to be sorted unlike a BTS. For this case, because we need to both search and sort the data the binary search tree is the best choice.

**Binary Search Tree**

CREATE new struct Course

VARIABLE courseName

VARIABLE courseID

LIST classPreReqs

CREATE new struct NODE

VARIABLE struct Course

VARIABLE Node pointing left

VARIABLE Node pointing right

CREATE class BinarySearchTree

CREATE method for adding nodes (See below)

CREATE method for searching nodes (See below)

CREATE method for printing nodes (See below)

CREATE new LIST check-prereq

CREATE instance of BinarySearchTree (BTS)

// Open the file

CREATE fstream object

OPEN file

// Check if file is open

IF file is open

FOR each line in file

// Check minimum requirements to be entered as a course

IF line has at least 2 tokens

CREATE new course

SET courseName to token 1

SET courseID to token 2

// Add course prerequisites

IF number of tokens > 2

FOR each remaining token

APPEND to classPreReqs

APPEND to check-prereq

ENDFOR

ENDIF

CALL BTS addNode -> new course

ELSE

PRINT "Invalid course"

ENDIF

ENDFOR

CLOSE file

ELSE

PRINT "File not found"

ENDIF

// Verify all prerequisite courses are valid

FOR each item in check-prereqs

CALL BTS search where key is the item

IF search finds key, remove all instances from check-prereqs

IF check-prereqs is not empty

PRINT "Invalid prerequisite course"

PRINT contents of check-prereqs

BTF methods:

**Add:**

**Insert Method -**

IF root is empty

CREATE new node at root ASSIGN bid

RETURN

ELSE

CALL addNode

ENDIF

**addNode Method-**

IF the node >= bid

IF the node to the left is empty

CREATE new node ASSIGN bid

ELSE

RECALL node to the left

ENDIF

ELSE

IF the node to the right is empty

CREATE new node ASSIGN bid

ELSE

RECALL node to the right

ENDIF

ENDIF

WHILE node is not empty

IF correct node is found

RETURN NODE

ELSE IF node is < key

Move node to the left

ELSE

Move node to the right

ENDIF

ENDWHILE

**Print:**

//In order printing

IF left node is not empty

RECALL to the left

ENDIF

PRINT data

IF right node is not empty

RECALL to the right

ENDIF

**Search:**

CREATE node ASSIGN to root

WHILE node is not empty

IF correct node is found

RETURN NODE

ELSE IF node is < key

Move node to the left

ELSE

Move node to the right

ENDIF

ENDWHILE

Print:

In order printing

IF left node is not empty

RECALL to the left

ENDIF

PRINT data

IF right node is not empty

RECALL to the right

ENDIF

**Hash Table Data Structure Pseudocode**

START

// Create the Course class

CREATE new CLASS Course

VARIABLE courseName

VARIABLE courseID

LIST classPreReqs

CREATE new VECTOR courses

CREATE new LIST preReqsList

CREATE hash key function

**Load Data Function**

// Open the file

CREATE fstream object

OPEN file

// Check if file is open

IF file is open

FOR each line in file

// Check minimum requirements to be entered as a course

IF line has at least 2 tokens

CREATE new course

SET courseName to token 1

SET courseID to token 2

// Add course prerequisites

IF number of tokens > 2

FOR each remaining token

APPEND to classPreReqs

APPEND to check-prereq

ENDFOR

ENDIF

GET hash key based on course ID number

INSERT course to courses at hash key

ELSE

PRINT "Invalid course"

ENDIF

ENDFOR

CLOSE file

ELSE

PRINT "File not found"

ENDIF

// Verify all prerequisite courses are valid

FOR each item in check-prereqs

CALL BTS search where key is the item

IF search finds key, remove all instances from check-prereqs

IF check-prereqs is not empty

PRINT "Invalid prerequisite course"

PRINT contents of check-prereqs

**Print in order function**

CREATE vector sorted\_courses

FOR each object in courses

APPEND to sort\_courses

WHILE object points to next

APPEND to sort\_courses

Advance pointer

CALL built in sort function on sorted\_courses vector

FOR each item in sorted\_courses

PRINT course information

ENDIF

**Search Function**

OBTAIN search key from user

CALL hashing function on key

WHILE

IF data at hashed key node is correct

RETURN data

ELSE IF next pointer is not null

ADVANCE pointer to next node

ELSE

RETURN null

ENDIF

**Vector Data**

START

// Create the Course class

CREATE new CLASS Course

VARIABLE courseName

VARIABLE courseID

VARIABLE preReq\_1

VARIABLE preReq\_2

CREATE new VECTOR courses

CREATE new LIST preReqsList

**Load Data Function**

// Open the file

CREATE fstream object

OPEN file

// Check if file is open

IF file is open

FOR each line in file

// Check minimum requirements to be entered as a course

IF line has at least 2 tokens

CREATE new course

SET courseName to token 1

SET courseID to token 2

// Add course prerequisites

IF number of tokens > 2

FOR each remaining token

APPEND to classPreReqs

APPEND to check-prereq

ENDFOR

ENDIF

APPEND course to courses

ELSE

PRINT "Invalid course"

ENDIF

ENDFOR

CLOSE file

ELSE

PRINT "File not found"

ENDIF

// Verify all prerequisite courses are valid

FOR each item in check-prereqs

CALL BTS search where key is the item

IF search finds key, remove all instances from check-prereqs

IF check-prereqs is not empty

PRINT "Invalid prerequisite course"

PRINT contents of check-prereqs

CALL built in sort function on courses vector

**Search function**

OBTAIN searchKey from user

FOR each item in courses

IF searchKey is equal to course name

PRINT course information

BREAK

ENDIF

ENDFOR

**Print in order function**

// The vector is already sorted using the built in sort function

FOR each item in courses

PRINT course information

**Menu**

WHILE user choice is not case 4

PRINT menu options

OBTAIN user choice

SWITCH CASE

CASE 1: Load data

CALL load data function

CASE 2: Ordered print

CALL print function

CASE 3: Search print

OBTAIN search key

CALL search function

CASE 4: Exit

ENDWHILE

END Program

Run Time Analysis

Vector:

Creating - O(n)

File Line - O(1)

Inserting - O(1)

Printing - O(n)

Hash Table:

Creating - O(n)

File Line - O(1)

Inserting - O(1)

Printing - O(1)

Binary Search Tree:

Creating - O(n)

File Line - O(nlogn)

Inserting - O(nlogn)

Printing - O(logn)