Project One

Nicole Swanson

CS 300 DSA: Analysis and Design

June 11, 2023

**Pseudocode**

Read file

Open file

While good

Read lines

Else

Error

Structure course

String course name

String course number

Vector prerequisite

Structure node

Course name and number

Create key with course number

Retrieve node with new key

Set equal to new node

Vector

Function search

Print enter course number to search

Initialize course number variable

Get user input string

Set boolean variable F equals false

For i equals zero to length of course vector

If course i course number equal to course number

Print course found

T equals true

If T not true

Print course not found

Function print

Print course details

For i equals zero to length of course vector

Print course number

Print course name

Print course prerequisite

For i equals zero to length of course i prerequisite

Print course i prerequisite i

Hash table

Create key with course number

Retrieve node with new key

Set equal to new node

While node does not equal null

If node course number is equal to course number

Set total prerequisites equal to node prerequisite size

For prerequisite p in total prerequisites

Add prerequisites p to total prerequisites

Print total prerequisites

Else

Set node equal to next

Print course information

Create key with course number

Set equal to new node

While node does not equal null

If node course number equals course number

Print course information

For prerequisites of course

Print prerequisite information

Else

Set node equal to next

Unsigned integer key

Return key table size

Insert course

Set unsigned integer key equal to course id converted to integer

Check if node empty

While not empty

Loop till empty node found

Insert course

Else

Insert course

Parse file line of course

If line size equal two

Set course to new course

Set course name equal to line [0]

Set course number equal to line [1]

Set prerequisite to empty vector

Return new course

Else

Vector temporary prerequisite

For integer i equals 2 and is less than line size

Push back temporary prerequisites line [i]

Set course equal to new course

Set course name equal to line [0]

Set course number equal to line [1]

Set prerequisite to temporary prerequisite vector

Return new course

New hash table

String temporary prerequisite vector

Read file lines

If file opens

While get line

Read string

While good

Create temporary variable for string

Push string into temporary prerequisite vector

Insert into hash table

Clear temporary prerequisite vector

Tree

If root equals null

Root equals new node

Else

Add nodes root and course

Add node

If node bid id is greater than 0

If node course does not equal null

Set node right equal to new node

Else

Add nodes right and course

Search nodes

Set current node equal to root

While current node does not equal null

If course number matches

Return current course

Else if current node compared to course number is greater than 0

Set current node equal to left

Else

Set current node equal to right

Return course

Integer prerequisite courses

Search course number

While course prerequisite does not equal 0

For each prerequisite in course

Search course name, prerequisite, and number

Increment prerequisites

Print course info

Search course number

Print course name and number

While course prerequisite does not equal 0

For each prerequisite in course

Search course name, prerequisite, and number

Print course prerequisite

Parse line

If line size equals 2

Set course equal to new course

Set course name equal to line [0]

Set course number equal to line [1]

Set prerequisite to an empty vector

Return new course

Else

Vector temporary prerequisite

For integer i equals 2 and is less than line size

Push back temporary prerequisites line [i]

Set course equal to new course

Set course name equal to line [0]

Set course number equal to line [1]

Set prerequisite to temporary prerequisite vector

Return new course

New tree

String temporary prerequisite vector

Read file lines

If file opens

While get line

Read string

While good

Create temporary variable for string

Push string into temporary prerequisite vector

Insert into binary tree

Clear temporary prerequisite vector

Menu

Output menu options

While input not 1,2,3,4

If input 1

Load data structure (load file before can sort)

If input 2

Print course list (alphabetically ordered list)

If input 3

Print course (for individual course title with its prerequisites)

If input 4

Exit

Else

Error

Print alphanumerically sorted list

Make partitions

Set low to first

Set high to last

Set midpoint to high and low divided by 2

Set pivot to midpoint

While pivot less than high

Decrement

Else

Swap lower values to left of pivot and higher values to right

Set temporary to low

Set low to high

Set high to temporary

Quick sort

Set midpoint to 0

Set low to begin

Set high to end

If begin greater than or equal to end

Return

Else

Quick sort

Print courses

For integer i equals 0 and is less than course size

If node does not equal null

Check left first

Check next right

Print in order include course name, id, and prerequisite

**Runtime Analysis**

| **Vector** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Read file** | **1** | **1** | **1** |
| **File fail** | **1** | **n** | **n** |
| **Set course name** | 1 | 1 | 1 |
| **Set course number** | 1 | 1 | 1 |
| **Initialize course prerequisite vector** | 1 | 1 | 1 |
| **Push back prerequisite vector** | 1 | n | n |
| **Print the prerequisite course information** | 1 | n | n |
| **Total Cost** | | | 3n +4 |
| **Runtime** | | | O(n) |

| **Hash table** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create key** | **1** | **n** | **n** |
| **If the course is the same as course Number** | **1** | **1** | **1** |
| **Set prerequisite to size** | **1** | **n** | **n** |
| **Add prerequisites to total prerequisites** | **1** | **1** | **1** |
| **Set table size** | 1 | n | n |
| **For all courses** | 1 | n | n |
| **Print out the course information** | 1 | 1 | 1 |
| **Total Cost** | | | 4n + 3 |
| **Runtime** | | | O(n) |

| **Tree** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **If root equals null** | 1 | n | n |
| **Add nodes and course** | 1 | n | n |
| **Set node to root** | 1 | 1 | 1 |
| **If course number matches return course** | 1 | n | n |
| **Set current nodes to left and right** | 1 | n | n |
| **While prerequisite not 0** | 1 | n | n |
| **Search course name** | 1 | 1 | 1 |
| **Set course lines to 0 and 1** | 1 | 1 | 1 |
| **Print courses and prerequisites** | 1 | 1 | 1 |
| **Total Cost** | | | 5n + 4 |
| **Runtime** | | | O(n) |

A vector is fast at reading a file. It is also easy to add items by appending them to the end of the vector. A down side is because each item has to be checked to find a match. This process can make this process slower for finding a specific course. A hash table uses a key to quickly search a list. They are however not good for sorting. Items need to be extracted before sorting. The more buckets are filled the longer this process will take. A binary search tree reads items in order. This makes the search a list fast if the tree is balanced. Otherwise, the process can be time consuming. In the end I believe that a binary search tree is the way to go for this project. It is able to quickly print a sorted list. The tree does not require sorting as it is traversed in order making it the best option for a program that’s goal is to print stored items in alphanumeric order.